A COMBINED EXPERIMENTAL AND SIMULATIONAL APPROACH TO THE STUDY OF OPTICAL EMISSIONS RESULTING FROM THE DISSOCIATIVE ELECTRON–ION RECOMBINATION OF SMALL POLYATOMIC IONS

by

BETHANY LACHELE FOLEY

(Under the Direction of NIGEL GRAHAM ADAMS)

ABSTRACT

Current experimental approaches to and theoretical descriptions of the study of electron–ion recombination are presented with focus on dissociative recombination. The experimental apparatus, chemical methodology and means of data analysis used in these studies are described. Photoemissions due, or likely to be due, to electron–ion recombination are reported and quantified for the following recombining parent ions: CH_5^+, HCO^+, CO_2^+, HCO_2^+, CS_2^+, NH_4^+, HNO^+, N_2OH^+, SH_3^+ and XeH^+. Practical issues regarding the composition of the plasma in which the recombinations occur and the characteristics of injection and mixing of the reagent parent gases are discussed; solutions to difficulties are suggested. A program that simulates rovibronic spectra is described. Also, a program that simulates the idealized injection into the reactive flow of a neutral reagent gas is described.

INDEX WORDS: Electron–ion recombination, Dissociative recombination, Spectroscopic products, Plasma chemistry, Gas–phase ion chemistry, Rotational spectroscopy, Rovibronic transitions, Simulation, Flowing afterglow, Rotational temperatures, Vibrational distributions, Mixing of gases

A COMBINED EXPERIMENTAL AND SIMULATIONAL APPROACH TO THE STUDY OF OPTICAL EMISSIONS RESULTING FROM THE DISSOCIATIVE ELECTRON–ION RECOMBINATION OF SMALL POLYATOMIC IONS

by

BETHANY LACHELE FOLEY

B.S., Auburn University at Montgomery, 1988

M.Ed., The University of Georgia, 1992

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2002

© 2002

Bethany Lachele Foley

All Rights Reserved

A COMBINED EXPERIMENTAL AND SIMULATIONAL APPROACH TO THE STUDY OF OPTICAL EMISSIONS RESULTING FROM THE DISSOCIATIVE ELECTRON–ION RECOMBINATION OF SMALL POLYATOMIC IONS

by

BETHANY LACHELE FOLEY

Approved:

Major Professor:	Nigel G. Adams
Committee:	I. Jonathan Amster Lucia M. Babcock David P. Landau Darwin W. Smith

Electronic Version Approved:

Gordhan L. Patel
Dean of the Graduate School
The University of Georgia
August, 2002

ACKNOWLEDGMENTS

There are three people I need to acknowledge here because there are no other places in the dissertation where it is appropriate to reference them. The first is my major professor, Nigel Adams. Among many other things, he taught me how to use, maintain and perform experiments with a flowtube. He also tried desperately to get me to be more positive in my writing style, but, alas, I fear that I am a lost cause. The second person is Mark Drucker, a graduate student who was working in the lab when I arived. On a number of occasions, we worked together to take data, splitting 15–20 hour days between us. If I included every piece of data that we took together, the dissertation might well be double the current length. The third person is Alan Ferrenberg, who listened to my babblings about simulations of reactive flow and worked with me to create the simulation discussed in Chapter 5.

The other people and institutions I would like to thank are: Jon Amster, Lucia Babcock, David Landau and Darwin Smith for being on my committee and for providing much needed guidance; Mike Duncan and Fritz Schaefer, who were previously on my committee (and were replaced for logistical reasons), and who also provided much needed guidance; Darwin Smith for being a mentor to me for many years, for listening to me gripe, for offering advice and being good company, and on a few occasions, for helping me to find employment; David Landau for being willing to work with me on a class during a very difficult period in my life; Kent Ervin for sending me his program that calculates Franck–Condon factors for transitions in diatomic molecules; Robert LeRoy for helping me to understand the simulation of bound–continuum spectra; Hélèn Lefebvre–Brion for helping me overcome a lack in self–confidence by saying, quite matter–of–factly, “you need to solve the Schrödinger

Equation;" Fritz Schaefer, Wesley Allen and Andrei Loginov for indulging (and answering) my questions about the things that diatomic molecules do; Bill Kerr for employing me and letting me learn a little food science along the way; The University of Georgia and The National Science Foundation for providing significant monetary support; Justin Fermann for introducing me to the C programming language and showing me how to install the Linux operating system for the first time; All the people who produce and maintain free and/or open–source software, but especially the developers and maintainers of Linux, Slackware, TEX [Knuth 1986], gcc, Gnuplot, Ghostscript/Ghostview, Netscape, xfig and XV; Sean Ohlinger and Wavefunction, Inc., for letting me keep the copy of Spartan'02 they accidentally left after a workshop [Wavefunction 2001] (see also Chapter 4); Missy Berry, Jeff Butler, Brian Decker, Mark Drucker, Neyoka Fisher, Kelly Kerr and Ted Williams, the other group members and student workers in the lab, who helped keep the equipment running, provided occasional technical assistance, and made for great company at lunch; Bill Foley and Betty Lincoln, for being kind, generous, loving and supportive parents, and who also partially funded the beginnings of my simulation work (though they might not be aware of that); James Lincoln for being kind to and supportive of me and mostly for being a good husband to Betty Lincoln; Kip Matthews and June Kaufman for trying to heal what I found to be painful rather than the things I think many others would assume were causing me pain; Agnes Scott College for being a great place to work; Alan Bowman, Nancy Heiges and Lisa Jennings for being good, loyal friends during times that were very troubling for me; Last, but absolutely not least, with last names omitted to protect the innocent, Aaron, Aaron, Alan, Allison, Andrew, Bill, Bill, Bob, Bruce, Carla, Charlene, Christina, Cynthia, Dan, Deanna, Don, Ed, Ed, Gabriela, George, George, Georgia, Greg, Greg, Hunter, Jamie, Jeff, Jeff, Jerry, Jim, Jimmy, Jody, John, John, Joseph, Justin, Kat, Kevin, Kevin, Kim, Kirk, Lisa, Mark, Martin, Mike, Nancy, Neyoka, Neysa, Olwen, Rick, Todd and Warren, for being good

people whose presence made life just a little more pleasant. If I left anyone out, it's because of exhaustion and not to be taken personally.

TABLE OF CONTENTS

APPENDICES

LIST OF ABBREVIATIONS

[ ] . . . . . concentration of

[] . . . . . reference

Å . . . . . Angstroms

amu . . . . . atomic mass units

cm . . . . . centimeters

cm^{-1} . . . . . wavenumbers

Cyl . . . . . cylinder

DP . . . . . diffusion pump

DR . . . dissociative recombination

e^- . . . . . an electron

EA . . . . . electron affinity

e–i . . . . . electron–ion

eV . . . . . electron volts

FA . . . . . flowing afterglow

ft . . . . . feet

GPC . . . . gated photon counter

gr/mm . . grooves per millimeter

h . . . . . Planck's constant

$h\nu$ . . . . . a photon

i–n . . . . . ion–neutral

in . . . . . inches

IP . . . . . ionization potential

J . . . . . Joules

k . . . . . reaction rate constant

k_B . . . . . Boltzmann's constant

LIF . . . laser–induced fluorescence

m . . . . . meters

MeV . . . . . 10^6 electron–volts

mm . . . . . millimeters

MS . . . . . mass spectrometer

mT . . . . . millitorr

nm . . . . . nanometers

n–n . . . . . neutral–neutral

$\mathcal{O}(\,)$ . . order of magnitude of ( )

PA . . . . . proton affinity

PE . . . . . potential energy

PMT . . . . photomultiplier tube

RIP . . . . . resonant ion pair

RP . . . . . reactant port

s . . . . . seconds

SIFT . . . . selected ion flow tube

slm . . . standard liters per minute

ST . . . . . sieve trap

τ . . . . general spatial coordinate

τ_{rad} . . . . . radiative lifetime

Tls^{-1} . . . . torr–liters per second

UV–VIS . . . . ultraviolet–visible

VUV . . . . . vacuum ultraviolet

XY . . . . . diatomic molecule

XYZ . . . . polyatomic molecule

CHAPTER 1
ELECTRON–ION RECOMBINATION:
DEFINITIONS AND INTRODUCTION

1-1 Definitions

Electron–ion recombination is the process by which a positive ion, I^+, and an electron, e^-, react with each other:

$$I^+ + e^- \rightarrow products \qquad 1$$

The process can occur, with varying efficiencies, between any positive ion and an electron. Depending on the nature of the process, more specific terms are often used. When the products are a single neutral particle and a photon, the process is called *radiative recombination*. When a second electron is involved in the process, it is termed *dielectronic recombination*. When a neutralized polyatomic ion breaks into smaller fragments, the process is called *dissociative recombination* (DR).

Recently it has been found that those smaller fragments are sometimes a positive and a negative ion, e. g.

$$XY^+ + e^- \rightarrow X^+ + Y^- \qquad 2$$

In this case, the process is said to have produced a *resonant ion pair* (RIP) [See Appendix A]. The study of the process is very new, and there is an apparent disagreement on whether RIP formation should be considered a separate process from DR, even though both processes result in "dissociation." For example, one paper [Lange 1999] has the title "Threshold Effects and Ion–Pair Production in the Dissociative Recombination of HD^+," which suggests that they consider ion–pair production a subset of

the possible outcomes of DR. Another paper [Zong 1999] makes an obvious distinction between the processes, avoiding the term *recombination* altogether except when speaking of the formation of neutral products. Instead, they use "electron collisions with molecular ions." In this manuscript, RIP will be considered a subset of DR, but only neutral products of DR will be discussed in detail.

Many other electron–ion recombination processes are well known and more solidly defined. For example, when the products are stabilized by the release of a photon, the term *radiative stabilization* is used. If collisions with other particles become involved in radiative stabilization of the electron–ion system, the process is called *collisional radiative recombination*. The ejection of an electron from the intermediate complex (either the incident electron or some other electron) is called *autoionization*. For a brief, but more technical description of the various processes than is presented here, the reader is referred to the *Atomic, Molecular and Optical Physics Handbook* published by The American Institute of Physics [Drake 1996, Section 52.1].

Recombination between atomic ions and electrons is very slow, with a rate coefficient of $\sim 4 \times 10^{-12}\ cm^3 molecule^{-1} s^{-1}$ [Murad 1986], so is of little interest to the type of study described in this document. Even so, it is important to know something of the process, as it provides a foundation from which to better understand the more complicated process of polyatomic ion recombination. When recombination between a solitary atomic ion and an electron does occur, energy must be lost from the system in order for the neutral to stabilize (and not autoionize). In the case of a single atom, the paths available are interactions with the atomic core (e.g., electron capture), translational excitation and photo–emission. Mechanisms for rotational or vibrational excitation, of course, do not exist, in contrast to the case with diatomic and polyatomic ions. Of the three available mechanisms, translational excitation plays very little part in the process. Rarely are there efficient mechanisms by which an incident electron can transfer translational energy to an ion. Even at high kinetic

energies, the electron is more likely to cause other electron(s) to be released from the ion than to cause the ion as a whole to move [the process is well known and is called *electron impact ionization*]. So, it is unlikely that energy possessed by the electron will be turned into sufficient kinetic energy in the electron–ion pair to stabilize the system. Nor do nuclear interactions play an important role. It is certainly possible, in some cases, to study the recombination of an ambient electron with an unstable nucleus, but, if the literature contains any references to that sort of experiment, this author has been unable to find them.

Whatever the case, the dominant stabilization processes for atomic–ion/electron recombination are collisional stabilization or photoemission. The only other significant non–radiative option available is, as was mentioned earlier, the re–ejection of an electron, where the net result is that recombination does not occur. In fact, in most of these interactions, autoionization is by far the dominant process. If one compares, in general, the efficiency for changes in internal motion, e. g., the re–ejection of an electron (timescales $\leq 10^{-14}$ s [Adams 1992]), and the emission of a photon ($\tau \geq 10^{-9}$ s), one finds that the re–ejection is significantly more efficient. It is this relative efficiency for re–ejection compared to radiation that makes atomic–ion/electron recombination such an inefficient process.

But even though inefficient, the process is worthy of study for certain applications. Radio astronomers, for example, use emissions from very high–orbital transitions of atomic hydrogen to study the structures and properties of objects in the interstellar medium [Dupree 1970]. These transitions are called *radio recombination lines* [DePree 2000]. One might argue that these "recombination" lines are merely transitions between high–level atomic orbitals. This is true; but, the lines are generally thought to be the cascade of an electron down the orbitals of a hydrogen atom as the electron and proton become successively more stabilized to form a neutral atom. So, it is not unreasonable to consider this to be sub–processes of recombination.

At higher pressures and/or higher ionization (charged particle) densities, the process of *collisional radiative recombination* [Drake 1996] can increase the probability of any recombination, including recombinations between single atoms and electrons. It occurs when one or more "third" bodies are involved in the recombination process. The third body might be any other ion or neutral and can also be an electron. At high pressures, collisions with "third" bodies may be important. In this process, the effect of collisions on the cascade of the electron to lower energy states may be coupled to the emission of radiation. Generally speaking, collisional effects are more important for high n levels where the radiative lifetimes are long. Radiation becomes more important as the electron transits to lower n levels. If one also considers re–absorption of light and effects of electron temperature and plasma pressure, the process becomes complex. However, at the pressures used in this study, such three–body interactions happen infrequently — far less frequently than binary DR — and can be safely ignored.

Dissociative recombination (DR) can only occur when the ion is capable of fragmentation upon recombination, so, DR applies only to molecular ions. DR occurs when the addition of an electron to a molecular ion results in the formation of two or more particles as products. The process is often followed by photoemission from at least one of the product fragments. Rate coefficients for DR are generally on the order of $10^{-6} - 10^{-7} cm^3 s^{-1}$ [Johnsen 1987]. Although it is possible for the recombination of a polyatomic ion to occur without product fragmentation, the process is not significant compared to dissociative recombination.

The most commonly observed products of DR are neutral atoms and molecules, but, as mentioned above, ion–pair production is also possible. Little, if any, information on resonant ion pair (RIP) production,

$$XY^+ + e \rightarrow X^+ + Y^- \qquad 3$$

is available for the case of triatomic and larger molecules. However, based solely on results for diatomic species, the possibility of the process being significant for larger polyatomics cannot be ignored. Many of these diatomic systems were studied at high interaction energies (k_BT corresponding to effective temperatures much greater than 300 K). For example, in the recombination of HeH^+, Larson and Orel [Larson 1999] calculated that the branching fraction for ion–pair formation might be as high as 10% of the total at energies on the order of 20 eV. For some species, such large energies are necessary for the system to be able to produce an ion pair. However, RIP has been observed to occur at lower energies for some systems. Zong, et al., [1999] observed ion–pair formation in the recombinations of HD^+ and HF^+. Although they found that a threshold energy of about 1.91 eV is necessary for RIP production in the collision of an electron with HD^+, they report finite rate coefficients for RIP in HF^+ at energies as low as 10^{-4} eV. RIP occurs more readily in HF^+ because the net energy change during the process is small. In the case of HD^+, the difference between the energy for dissociation to H^+ and D (D_{ion} = 2.6677 eV [Zong 1999]) and the electron affinity of D (EA_D, 0.7546 eV [Zong 1999]) requires an energy of 1.913 eV be added to the system for RIP formation. However, in the case of HF^+, the electron affinity (EA_F = 3.412 eV, [Zong 1999]) and the dissociation energy of the ion (D_{ion} = 3.417 eV, [Zong 1999]) are within thermal energies of each other. Zong, et al., [1999] also found for this system that, at 0 eV, the ratio of RIP to the production of neutral products (DR, in their paper) is about 0.3. It is possible that, for larger polyatomic species, further studies will find that the additional vibration and rotation in the channels available for the dispersal of energy will make the RIP channel less important. However, without further experimental and theoretical investigation, significant RIP formation in the DR of polyatomic ions remains only a possibility at the present time.

Whatever the products, electron–ion recombination is, generally speaking, highly energetic. The energy gained by electron and ion is equal to the recombination energy

of the ion plus the relative kinetic (thermal) energies of the ion and the electron. That energy must be dissipated in some manner; for triatomic or larger ions, there are numerous possibilities, most involving the cleavage of one or more bonds. Stabilization can also be provided by translational, rotational, vibrational or electronic excitation in the fragments. Indeed, the breaking of one or more bonds is often accompanied by the emission of light from a number of ro–vibronic states in the fragments. A common symbolism for the process is:

$$\begin{aligned} XYZ^+ + e^- &\rightarrow XY^* + Z^* \\ &\hookrightarrow XY + h\nu \end{aligned} \qquad 4$$

where the asterisk (*) denotes excitation (electronic, vibrational, etc.) in that product. Note that this symbolism does not reflect all possibilities for all types of polyatomic ion; it is merely a representation of the primary classes of products from the process. Note also that the equations do not include the neutral molecule corresponding to the unfragmented polyatomic ion. It is unlikely that a particle analogous to the neutralized ion ever exists as anything more stable than a transition state (except, perhaps, for larger molecular structures; the detailed mechanisms of dissociative recombination will be discussed more thoroughly in Section 1-2.).

The possible emission of light makes the process interesting because it can be studied passively — that is, information can be obtained about the process without interfering with the process. This aspect is valuable for monitoring remote environments such as ionized clouds of gas in the interstellar medium. Even the absence of photoemission, where there is sufficient energy for the production for certain photons, can tell us much about the dynamics of the process.

1-2 Theoretical Descriptions of Dissociative Recombination

Details and summaries of various theoretical treatments of dissociative recombination have been presented elsewhere; so, only a brief synopsis of the theory, as it applies to the current study, will be provided here. For example, although any description of the mechanism by which an electron and an ion recombine will, to some extent, apply to ion–pair formation as well as to neutral products, only the formation of neutral products will be considered in any detail. Computational models will not be presented; instead, the focus will be on theory. However, the Reference section contains references to works that address all of these issues and many others. In Appendix A there is a listing of references arranged by category.

Since an electron has very little mass compared to a nucleus, it is not appropriate to consider the collisional momentum of the electron as being the direct reason that the nuclei move apart. As was stated in the last section, even at high kinetic energies, the electron is more likely to cause the ejection of other electron(s) than to move the ion as a whole. Certainly, though, it is possible for a recombining polyatomic ion to dissociate, and this generally occurs. In such a case, the fragments typically gain translational linear momentum with respect to each other. But, the translational momentum does not result directly from the addition of momentum from the electron.

A better way to think of the process is that the addition of the incident electron changes the shape of the potential field in which the nuclei find themselves. Due to this change in the potential field, a bond might be broken between two atomic cores in the molecule (i. e., the system moves onto a repulsive potential). In this case, the two parts of the molecule might experience a repulsive force and begin to move apart. The observation of vibrational, rotational and translational momentum in DR products has prompted theorists to investigate the source of such excitation. For example, Bates [1993b] presented a theory describing mechanisms for vibrational excitation in polyatomic DR products. The theory involves contributions from two

Figure 1-2.1

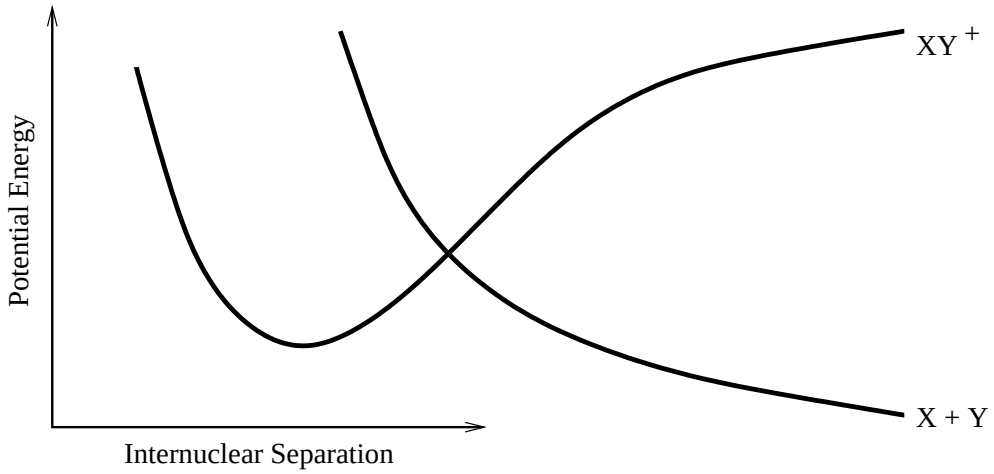


Figure 1-2.1: Schematic diagram of a curve crossing. If recombination occurs by curve crossing, the potential energy curve for the unbound, neutral state (X + Y) crosses that for the bound ion, XY^+. This process becomes likely when the curve crossing occurs at an internuclear separation probable for the ion. In this schematic, the crossing is not near the potential minimum for the bound ion, and one would expect vibrational excitation in the ion to be necessary for recombination to occur via the curve crossing mechanism.

mechanisms that he called *impulse* and *relic*. A summary description of this theory can be found a few paragraphs below, just after a summary of current ideas involving the change in potential caused by the incident electron.

Two important DR mechanisms are called *crossing* and *tunneling*, respectively. Their names are based on the relative positions of potential energy curves that describe the process. In the crossing mechanism (see Figure 1-2.1) there is direct interaction between the potential energy curves for a bound ionic state and an unbound neutral state (a *curve crossing*). No such direct interaction is present in the case of the tunneling mechanism (see Figure 1-2.2). In 1994, Bates [Bates 1994] published

an article detailing the history, development, and current state of understanding of the two mechanisms. Briefly, this is as follows.

When describing DR, the potential energy curves that are presented usually include, but are not limited to, one curve describing the bound ion and a curve describing at least one unbound, neutral state. Schematic diagrams of possible arrangements of these curves are shown in Figures 1-2.1 and 1-2.2. Figure 1-2.3 is another schematic describing one type of behavior that a dissociating system might undergo. Curves for specific systems can be found in many of the papers in the reference section, especially those categorized under "Computational and Theoretical — DR" in Appendix A.

In the crossing mechanism, the potential energy curves for the recombining ion and the dissociating products cross each other at a molecular geometry probable for the ion (see Figure 1-2.1). Upon neutralization by the electron, the system is considered to leave the bound curve (XY^+ in Figure 1-2.1) and begin to separate along the unbound potential curve (X + Y). Quantum–mechanical descriptions of systems allow all internuclear separations: some are simply far more probable than others. So, the physical interpretation of the curve–crossing mechanism is that there exists a probable internuclear separation for the recombining ion that has the same energy as the (neutral) dissociated particles would have at that same separation. Thus, the dissociation portion of the recombination is the reverse of a half–collision between the two unbound particles. Geometries for these systems are typically described in terms of the separation between the two atoms involved in the bond that is breaking. With more complex polyatomics, there is, of course, the possibility that such a simple description will not be sufficient. Even so, the one–dimensional picture serves to illustrate the mechanism well

Sometimes, the potential curves never cross or only cross at high energies. If there is no common energy/separation accessible to the bound ion and to the neutral products, the recombination may proceed via quantum mechanical tunneling (see

Figure 1-2.2

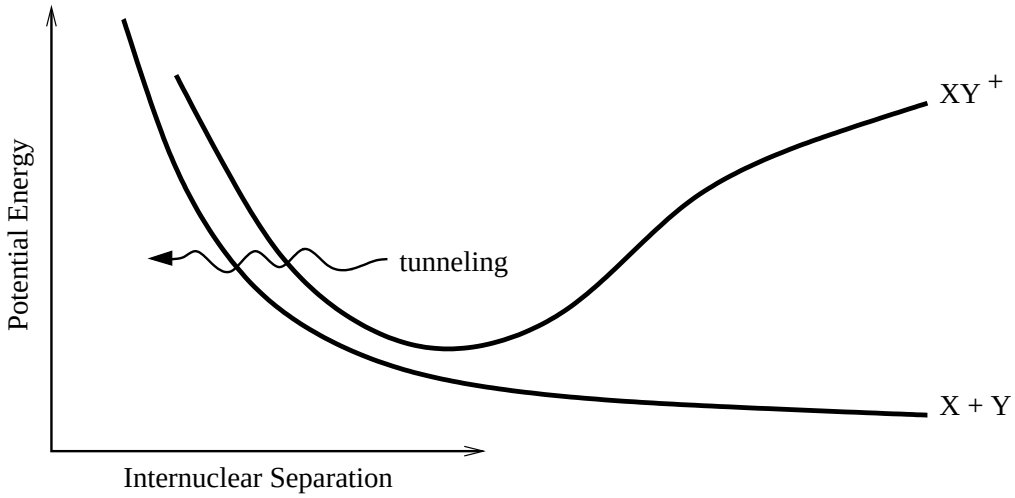


Figure 1-2.2: Schematic diagram of the relative positions of two potential curves for a system in which recombination via tunneling is likely. In this diagram, the potential curve for the unbound, neutral state (X + Y) does not cross the potential curve for the bound ion, XY^+, at an internuclear separation probable for the ion. No intermediate states are involved in the interaction.

Figure 1-2.2), from a point on the bound curve across to the isoenergetic point on the unbound curve.

When two (or more) potential curves cross each other, there is also, in some circumstances, the possibility for the curves to "mix" and produce what is termed an *avoided crossing*. This can occur whenever the two curves describe what Herzberg [Herzberg 1989] called "states of the same species." His use of the term "species" does not refer to the atomic or molecular identity of the particle. Instead, it refers to the characteristics of two electronic states within the same molecule. The concept is similar to the splitting of rotational states that would otherwise be degenerate, except that, here, it is being applied to a standard potential energy curve constructed for a rotationless, vibrationless system. For the avoided crossing to occur, all symmetry

Figure 1-2.3

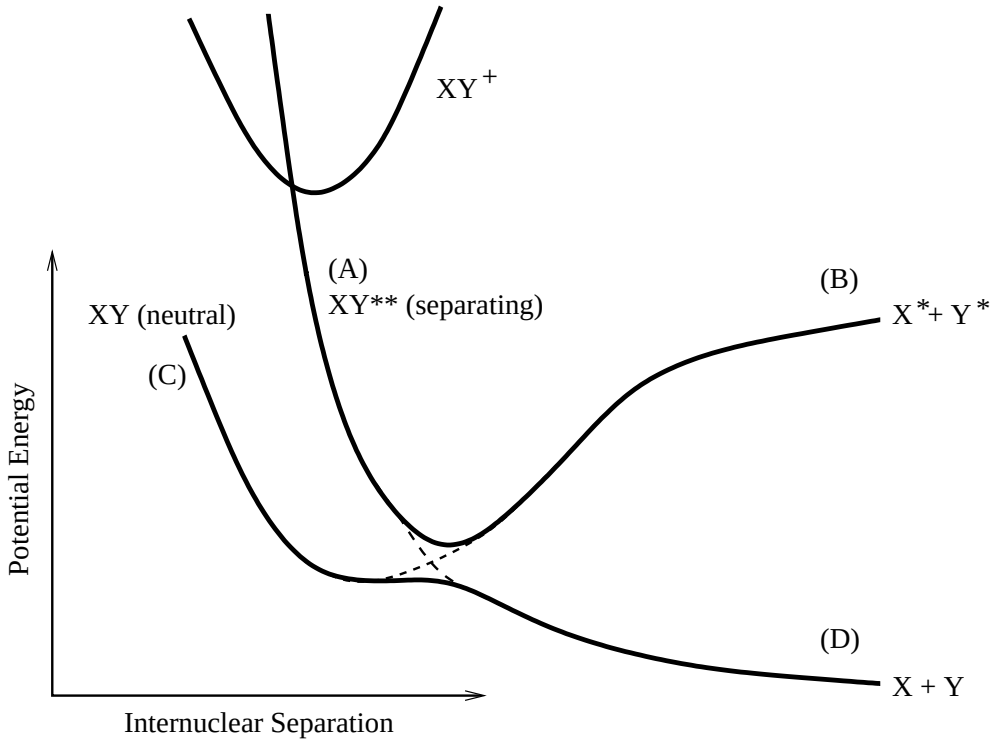


Figure 1-2.3: Schematic for an avoided crossing. The intent of this figure is to depict the possible "routes" that can be taken by the newly–recombined activated complex, XY** in the event that it encounters an avoided crossing. Consider the complex to be dissociating along path (A). The dashed lines represent the crossing that is being avoided. Note: the middle curve (A–B) describes an excited bound state of XY correlating to excited states of the separated atoms X and Y; the ground state neutral, XY, is weakly bound and will apparently predissociate along (D) at some excited rotational or vibrational level. See text for further discussion of the figure.

---

properties must be the same; these include: Λ, the total orbital angular momentum; S, the electron spin vector; Ω, the vector describing the interaction of Λ and S; for Σ states, the symmetry of the wavefunction reflected across the internuclear axis (+ or $-$); and, for homonuclear systems, the parity (*gerade* or *ungerade*);

If the crossing is avoided, the potential curves that would cross in a first approximation serve as something similar to asymptotes to the proper curves. The term "avoided" is only partially descriptive, however. A crossing is only fully avoided in the limit of infinitely slow change of internuclear separation. So, for high vibrational levels or rapid dissociations, the crossing will not be completely avoided, and in the limit of high velocity the crossing will never be avoided. Herzberg [1989, ca p 295] describes this effect in detail. The practical effect is that the "avoided crossing" and the "crossing" exist simultaneously. Figure 1-2.3, a schematic of an avoided crossing, depicts two possible states that could result from the recombination of the ion XY^+. Consider the activated complex, XY**, resulting from the ion, XY^+, having just been neutralized by an electron. The neutralized ion initially separates along the repulsive potential labeled (A). The presence of the avoided crossing implies that the activated complex is most likely to dissociate via path (B). However, the system can also dissociate along (D). Given that the momentum of the system is in the direction of separation, the complex is unlikely to take route (C) and become the bound neutral molecule, XY. But, even if it does "go" that route, either directly or via path (B), the complex contains sufficient energy to dissociate and will do so unless it is stabilized via collision or radiation. A system $(N_2^+ + e)$ whose product potentials include an avoided crossing has been discussed by Bates [Bates 1994, ca. page 446] as having been studied both computationally [Guberman 1991] and experimentally [Helm 1989]. The "avoided" route $(N(^2D) + N(^4S))$, equivalent to (D) in Figure 1-2.3, was found by both studies to be the primary channel for dissociation (88%). However, another study [Queffelec 1985] found the branching ratio for the avoided channel to be, at most, 0.15.

Theoretical descriptions of DR can include transitions involving any number of electrons, up to the total number of electrons present in the system. However, the

descriptions are sometimes limited to single– or two–electron transitions due to mathematical complexity or computational expense.

In the case of tunneling, the transitions are often considered to be single–electron transitions. Bates discusses only two classes of single–electron transition in detail. In one, the *valence decrease* class, the incident electron enters an orbital around one atom in the ion. The occupation of this orbital precipitates the dissolution of one (or more) of the existing bonds between the atom to which the electron was added and other atoms in the ion. The other class, consisting of cluster ions containing a *proton–bridge bond*, i. e. $A \cdots H^+ \cdots B$, is intuitively similar. It is only in the mathematical treatment that distinctions arise [Bates 1994]. In this case, the incident electron moves into the 1s orbital around the bridging proton. Again, this precipitates dissolution of a bond. As in the former class, the bond broken is between the bridging hydrogen and one or more of the sub–particles for which it served as a bridge.

In the case of a curve–crossing (Figure 1-2.1), a single–electron transition is not possible [Bates 1994]. Two dominant descriptions of the curve–crossing process are called *direct* and *indirect* (see Figure 1-2.4). In *direct* recombination, the incident electron initiates a two–electron transition directly to the repulsive state of the unbound neutral. An expression for the recombination coefficient for the direct mechanism is presented in Equation 6 with discussion following. *Indirect* recombination, a theory initiated by Bardsley [1968a,b], and described by Bates [1994, ca. p. 437], involves two radiationless transitions. Here, the first transition is into a Rydberg state of the neutralized ion. Then, the second transition is into the same unbound, repulsive state as accessed in the direct mechanism. Bates [1994] gives the rate coefficient for the *indirect* mechanism as:

$$\alpha'_I = \int \left\{ \frac{8}{\pi m (k_B T)^3} \right\}^{1/2} \epsilon \, \sigma'(E) \exp\left(\frac{-\epsilon}{k_B T} \right) \mathrm{d}\epsilon \qquad 5$$

Figure 1-2.4

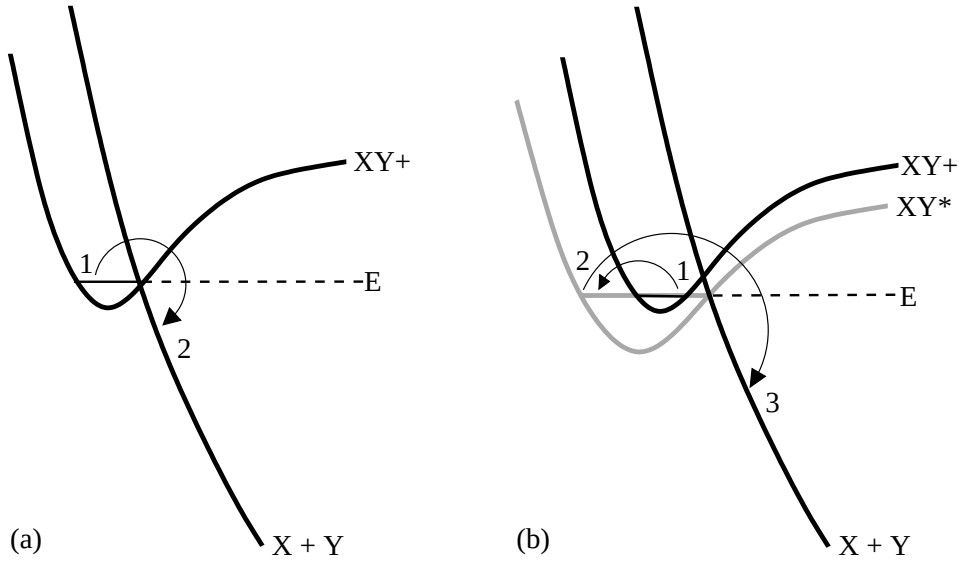


Figure 1-2.4: Schematic diagram depicting the difference between the direct and indirect mechanisms. (a) In the direct mechanism, the system moves directly from the bound state of the ion, XY^+ (1), to the repulsive neutral curve, X + Y (2). (b) In the indirect mechanism, the system (1) first transits to a high vibrational level of a Rydberg state of the neutral, XY^* (2), and then transits to the repulsive neutral curve, X + Y (3). In either case, the transitions do not involve radiation, so the system retains its initial total energy, E.

Here, π, k_B and T have their usual meanings [see the List of Abbreviations]; ϵ is the energy of the incident electron and $\sigma'(E)$ is the cross section for an electron with energy ϵ.

The following expression, given by Bates [1994] describes the recombination coefficient, $\alpha_D(v)$ for a *direct, two–electron* transition from vibrational level v in the ion where there exists a favorable curve–crossing to a repulsive neutral potential curve and where all products dissociate:

$$\alpha_D(v) = \frac{Srh^3}{2(2\pi m k_B T)^{3/2}} \cdot \frac{2\pi\hbar}{me^4} \int \left|\psi_v(R) V(R) \phi_\epsilon(R)\right|^2 \mathrm{d}R \;\; \exp\left(\frac{-\epsilon}{k_B T}\right) \mathrm{d}\epsilon \qquad 6$$

where S is defined below; r is the statistical weight of the neutral compared to the corresponding ground–state ion; m is the reduced mass of the electron–ion system; T is the temperature of the system; π, k_B, h and $\hbar$ have their standard meanings [see the List of Abbreviations]; e is the elementary charge; $\psi_v^*(R)$ is the vibrational wavefunction for the ion; $V(R)$ represents the "interaction causing the autoionization;" ϕ_ϵ is the wavefunction describing the separating particles; R is the internuclear distance; and ϵ is the energy of the system.

Bates's [1994] derivation of Equation 6 begins with a modification of an expression for the dielectronic recombination rate coefficient:

$$\alpha(p) = \frac{A(st)A(au)}{A(st)+A(au)} \frac{rh^3 \exp\left(-\epsilon/kT\right)}{2(2\pi mkT)^{3/2}} \qquad 7$$

In this expression, $A(au)$ is the autoionization rate for a system consisting of a coupled ion–electron pair with an excited state, p, at energy ϵ above the ionization potential of the neutral particle. $A(st)$, in terms of plasma dynamics, is the rate at which neutrality of the plasma is maintained by loss of energy in excess of ϵ; in terms of dissociative recombination, it is the rate at which the electron–ion complex is stabilized by dissociation. Equation 7 is based on the assumption that the system is in steady–state and is valid for recombinations via level p. The expression in Equation 6 is then generated by first making the following substitution:

$$\frac{A(st)}{A(st)+A(au)} \equiv S \qquad 8$$

This definition allows S to be treated separately from the integral in Equation 6. The separate treatment of S is based on the assumption that the stabilization rate will be much greater than the autoionization rate (due to the rapid dissociation of the recombined ion), and the value of S is generally considered to be approximately one. Next, it is necessary to find an expression for the autoionization rate, $A(au)$, that remains after the substitution of S. Bates uses methods familiar to quantum

mechanics in his treatment of $A(au)$:

$$A(au) = \frac{2\pi\hbar}{me^4}\left|\int \psi^*(R)V(R)\phi_\epsilon(R)\,\mathrm{d}R\right|^2 \qquad 9$$

The symbols in Equation 9 are defined after Equation 6, above.

Although the theoretical descriptions can become complex, a visceral understanding of the processes and of the effects of environmental factors, such as temperature, can be gained by inspection of the relevant potential energy curves. For example, one very important concept is the effect of collision, or interaction, energy on the potential curves. The reason the concept is important is that the collision energy effectively changes the position of the potential curve for the recombining ion, but not for the products [Bates 1994, e. g., p. 436 & 466]. That this is the case can be deduced by considering other possibilities. For example, consider that the ion is in vibrational state v of the electronic state described by potential (a) in Figure 1-2.5. If the system is placed at a higher energy by the proximity of an electron with kinetic energy, E, that extra energy is not likely to correspond to vibrational excitation in the ion. The potential field generated by the motion of the electron might distort the electronic structure surrounding the ion, but it is not expected, necessarily, to provide vibrational excitation. Instead, it is the vertical position of the potential that will change. The potential for the neutral products of recombination, however, will not be influenced by the presence of the high–energy electron (especially since the electron is no longer free). Figures 1-2.6 — 1.2.8 illustrate the effect for a few hypothetical systems, with Figures 1-2.6 and 1-2.7 focusing on the effect where a curve–crossing appears.

The effect of interaction energy in the case of a curve–crossing is easily seen by plotting the potential curve for the ion with and without an increase of interaction energy (collision energy). Figure 1-2.6 is a schematic of a system where an increase in electron energy favors the curve–crossing mechanism for low vibrational levels of

Figure 1-2.5

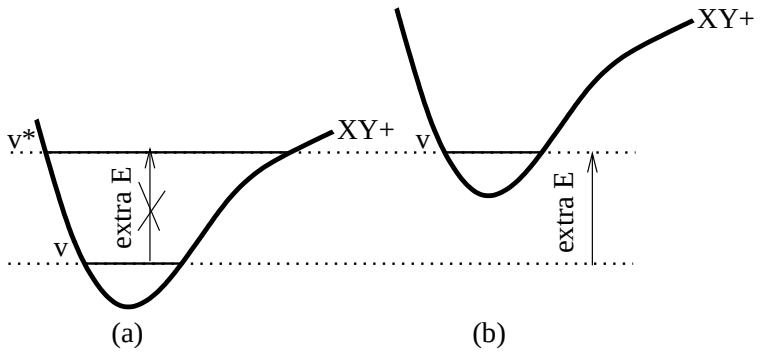


Figure 1-2.5: Illustration of the effect of the kinetic energy of an electron on the potential of the ion. Rather than causing vibrational excitation (a), the presence of the energetic electron raises the ion's potential (b) relative to the neutral products (not pictured).

the ion. To account for the energy of the electron, the position of the PE curve for the ion, initially at (a), must be raised to (b). Note that the unbound curve, (c), does not move. The opposite situation is depicted in Figure 1-2.7. For that system at low electron energy, the unbound potential crosses the ion potential near the potential minimum. But, the addition of electron energy effectively moves the crossing point to higher vibrational levels of XY^+.

In the case of tunneling (Figure 1-2.8), the situation is not as simple. One must consider the probability that the electrons already bound to the ion and the neutralizing electron will arrange themselves in a manner consistent with formation of two unbound particles. A discussion of the electron–orbital implications appears below, and certainly the incident electron affects the shape of the orbitals. But, for the sake of this argument, consider that the electron's effect on the orbitals is not altered by a change in the electron temperature. Assume that the only change is in the relative energies of the two states, ionic and unbound. Consider Figure 1-2.8 as depicting the

Figure 1-2.6

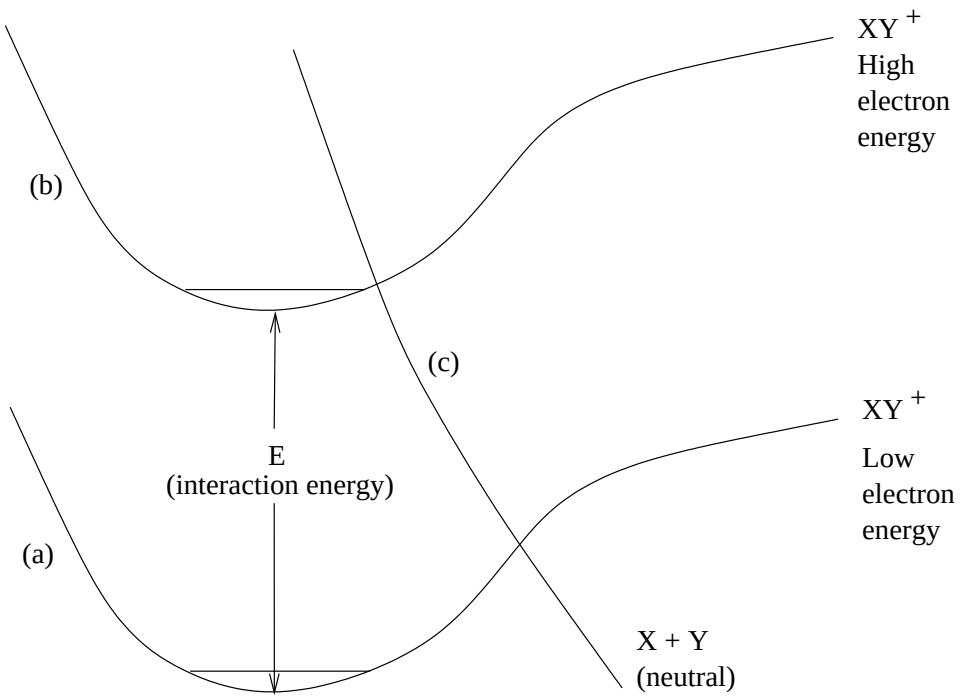


Figure 1-2.6: Hypothetical system where an increase in interaction energy, moving the curve from (a) to (b), favors curve–crossing DR from low vibrational levels of the recombining ion (see text).

potential recombination of an ion, XY^+, in its zeroth vibrational level. A horizontal, isoenergetic line is included for convenience, as are schematic representations of bound and unbound vibrational wavefunctions at that energy. If the orbital shapes do not change, then changing the electron temperature only changes the particular unbound wavefunction that is to be considered when calculating an overlap. Since there is not an exchange of photons (an implicit assumption thus far), the change in state must be isoenergetic, and only the wavefunctions at that energy (ignoring uncertainties) are important. If the position of the bound curve were moved higher or lower, the nature of the overlap would necessarily change. Intuitive prediction of the manner in which the overlap will change is not a trivial task, but, a few generalizations can

Figure 1-2.7

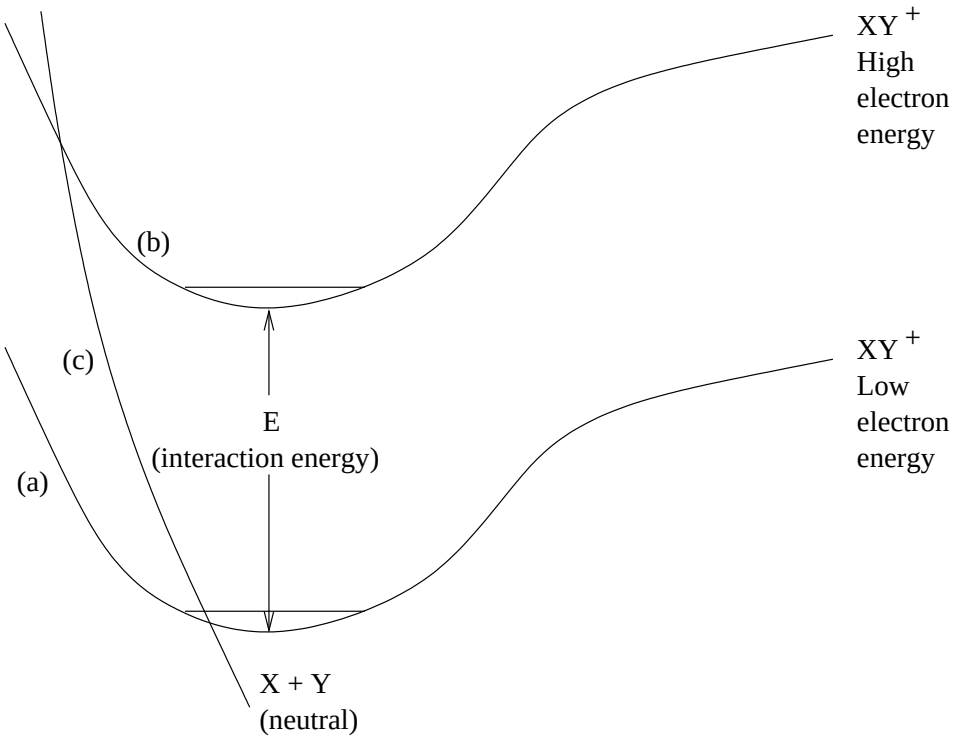


Figure 1-2.7: Hypothetical system where an increase in interaction energy, moving the curve from (a) to (b), does not favor curve–crossing DR from low vibrational levels of the recombining ion (see text).

be made. For example, in the situation depicted in the Figure 1-2.8, a decrease in interaction energy would cause an increase in DR probability, since a low–energy unbound wavefunction would have a broad peak coincident with the zeroth vibrational level of the ion.

It is not completely correct, as was assumed for the arguments in the tunneling example, to assume that the incident electron does not have an effect on the orbitals, and that a difference in initial electron velocity will not change the magnitude of that effect. A brief look at elementary electrodynamics [Bueche 1986, or most undergraduate physics texts] informs us that the magnetic field induced by the motion

Figure 1-2.8

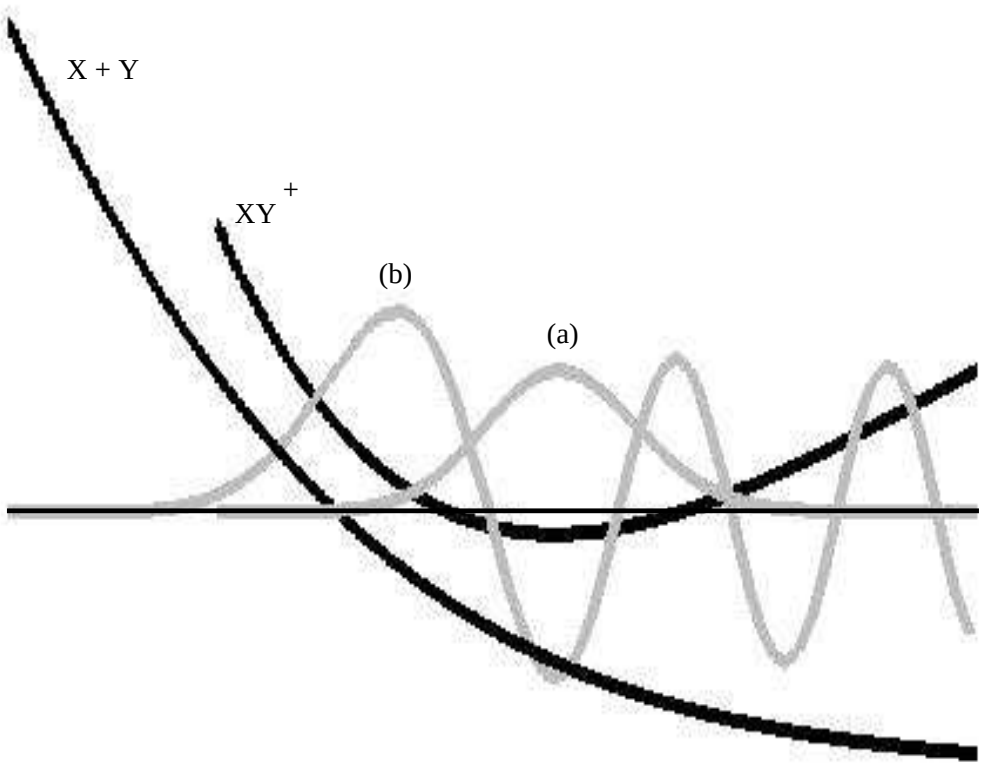


Figure 1-2.8: Schematic depicting some of the factors important to tunneling DR (see text). The factors illustrated are: the bound potential curve of the ion (XY^+) and the wavefunction for vibration of the ion in the zeroth vibrational level (a); the unbound potential curve for the product neutral (X + Y) and the wavefunction for the unbound system at an energy coincident with the energy of the zeroth vibrational level of the ion (b). The wave functions are not normalized.

of charged particles is directly proportional to the current, or speed, of the charged particles. So, in this way, the initial velocity of the electron is certainly expected to have an effect on the recombination.

The initial trajectory of the electron with respect to the ionic structure should have an effect as well. Although the ion is positively charged, overall, the existence of electronic structure implies the possibility that different effective charges will be experienced at different collision geometries. Harland and Vallance [Harland 1998] present detailed accounts of orientation effects in electron–impact ionization. Both processes, electron–impact ionization and electron–ion recombination, involve the collision of an electron with a particle. If orientation is important for electron–impact ionization then there should be orientation effects present for electron–ion recombination as well.

To illustrate this, consider the dissociative recombination of a heteronuclear diatomic ion, HF^+. It is generally well known that most of the electron cloud surrounding HF^+ is "owned" by the fluorine atom, and that the bulk of the positive charge in the vicinity of the hydrogen. If an electron collides with this ion, it seems natural to assume that an electron approaching from the hydrogen side encounters a different environment than one approaching from the fluorine side. Of course, the ion will generally be vibrating and likely rotating, which complicates the picture presented here. Uncertainties about the wave/particle characteristics of the electron complicate things further. However, the electron is not approaching the ion from all directions at the same time. In other words, the electron, be it particle or wave, must approach the ion from a particular direction. It is possible that an electron approaching the ion could take a trajectory such that it collides with the ion in an area that is more fluorine atom than hydrogen ion. Since fluorine is the most electronegative element known, with a large electron affinity, one might expect the electron to enter an orbital around the fluorine. In the case of a subsequent dissociation, an ion–pair would be formed. Indeed, recent storage–ring experiments have found [Zong 1999] that the interaction between an electron and an HF^+ ion at 0 eV produces an ion–pair about 1/3 as often as it produces two neutral atoms.

It is clear from the above that the trajectory of the incoming electron needs to be considered. The influence of the electron on the ion's electronic orbitals has been shown to be important, as has the effect of collision energy on the position of the potential curve for the ion. The effect of vibrational overlap between the initial and final states has been discussed. Many other possible interactions have not been discussed here, but should be considered. These include, at least, the s–character or p–character of the incident electron, the influence of Rydberg states (the *indirect* mechanism), rotational excitation and, most importantly, the changes within the atomic and molecular orbitals.

As mentioned earlier, another part of the process that aroused the interest of Bates [1993b] was the suggestion that some diatomic products of polyatomic recombination are vibrationally excited [Duley 1992]. He presented two theoretical approaches to account for this, which he called the *relic* and *impulse* mechanisms. The relic mechanism is based on the notion that, for example, in the recombination of triatomic ion XYZ^+, to form X + YZ, a portion of the wavefunction describing the molecular orbital around the ion is associated with the neutral, YZ. The nature of the overlap between this part of the molecular orbital of XYZ^+ and the free YZ molecule will be different for different vibrational levels. According to Bates, it is this "relic" of the wavefunction that serves as one mechanism for vibrational excitation in the neutral diatomic product. His other mechanism is based on the fact that, as the two products, X and YZ, repel each other, the initial repulsion will affect atom Y before atom Z, therefore providing an impulse to Y, initiating vibration in the product. His treatments involve the simple harmonic oscillator assumption for vibrations and he treated directly only diatomic products in the ground electronic state. Neither of these mechanisms has held up well to comparison with experiment for vibrational excitation in electronically excited states [Adams 1994a].

Since electron–ion recombination involves a radiationless transition between a bound state and an unbound state, the process has similarity to predissociation. Herzberg [1989, ca pp 420-421] suggests that the dominant feature controlling vibrational excitation in the case of predissociation is simply the overlap between the vibrational wavefunctions of the bound and unbound states at the interaction energy. So, it is possible that a similar relationship holds in the case of DR.

If an electron is in some electronic orbital of very large n[†] (Rydberg state or beyond) of a polyatomic molecule, XYZ, then the ion core, XYZ^+ could be considered to be, largely, ignorant of the high–orbital electron. Indeed, there will be little difference to the ion core if the electron is removed altogether. In such a case, the potential energy curve of the ion core would have nearly the same sort of shape as if it were considered to be a solitary ion. This line of reasoning doesn't take one directly to an overall recombination coefficient, but it does lead to a means for calculating the relative populations of vibrational excitation in the products. Viewed from this perspective, DR is very similar to predissociation. In fact, it is, essentially, equivalent to predissociation. The primary difference is that instead of dissociating from a low–lying bound electronic state, the dissociation occurs from a high–level Rydberg state. So, the relative vibrational population resulting from DR can be calculated in a manner similar to that for any other electronic transition — essentially, by taking a Franck–Condon overlap between the two states and squaring the result. It is probable that this description will lead only to approximate results, but it is simple and can serve as a starting point for the development of more rigorous calculations. Consider the relative vibrational excitation in the diatomic product XY from the DR of ion RXY^+, where R is one or more atoms, and X is bonded directly to Y. Y is, of course, not bound to atoms other than X, but that situation could be handled in a manner analogous to this treatment. In the following expression, $\psi_{v'_{RXY}}(\tau)$ is the

[†] The principal quantum number.

wavefunction describing vibration in the ion, $\psi_{v''_{XY}}(\tau)$ is the wavefunction describing vibration in XY and $f_{v''_{XY}}$ is the *fractional* excitation in vibrational level v''_{XY} of XY. τ is a generalized spatial coordinate, the applicable volume element. In the simplest case, τ could be replaced by r, the internuclear distance between R and X. Note that the quantum number v''_{XY} is not necessarily the same as the quantum number v'_{RXY}.

$$f_{v''_{XY}} \propto \left| \int \psi_{v'_{RXY}}(\tau) \psi_{v''_{XY}}(\tau)\, d\tau \right|^2 \qquad 10$$

In essence, this is the Franck–Condon overlap between the two vibrational wavefunctions. Note also that the calculation can be simplified somewhat. Since the vibrational wavefunction for the diatomic molecule decays rapidly to zero beyond the classical turning points, it is only necessary to consider that part of the ion's wavefunction in the vicinity of the diatomic product.

A few reasonable simplifications and approximations make such a calculation trivial, but not very predictive at this time. The technique holds promise, but it will be necessary to have better descriptions of the ionic states than are currently available [Jacox 2002] before it can be thoroughly tested. If accurate descriptions of recombining ions make the method more predictive, it would be useful in its simplicity, both conceptually and computationally.

One recent paper uses methods that have similarity to the preceding discussion, but with a far more rigorous approach and considerable success [Tomashevsky 1998]. They base their calculations on the *relic* and *impulse* mechanisms proposed by Bates [1993b]. But, their method also includes the effects of Franck–Condon overlap. They do not treat Bates's two mechanisms separately, but include them both in their calculations. They use both classical and quantum mechanical solutions to the equations of motion. Their results compare far more favorably to experiment [Adams 1994a and Butler 1997] than did either of Bates's mechanisms alone.

1-3 Experimental Studies

Experimental studies of electron–ion recombination have been carried out on a number of systems. Appendix A lists references arranged by category, and there is a large section devoted to studies of electron–ion recombination. Although the focus of this document is on photoemissions from dissociative recombination, it is useful to consider other studies of the process as well.

Studies of dissociative recombination can be classified by the type of information they determine about the process, the main information types being kinetic and product–related. Studies of recombination kinetics generally base results on the decay of an ion signal as detected by a mass spectrometer, Langmuir probe, or other ionization detection device. These studies can be successfully carried out without the researcher being concerned about the nature of the products. Product–related studies are the opposite — they are less concerned with the rate at which the reaction proceeds than with the final outcome of the recombination. Section 1-2 contains a somewhat detailed discussion of different mechanisms of recombination, including the different product classes that are possible. Some experiments, of course, study kinetics and products at once. See Appendix A under heading "Experimental Results — DR" for references. Experiments with more than one goal (e. g. those that study both kinetics and products) appear under multiple sub–headings.

The two most important techniques currently being used are the flowing afterglow [Adams 1988a] and the ion storage ring [next paragraph]. Either of these techniques can be used to study kinetics or products, if the experiment and the experimental apparatus are designed appropriately. An older technique, the predecessor of the flowing afterglow, is the stationary afterglow [Schmeltekopf 1963, Weller 1967]. Also, Mitchell and Rowe [Mitchell 2000] have made suggestions for new experiments, all involving storage rings or flowing afterglows.

Use of storage rings is fairly new, with details of the experiment and apparatus being described elsewhere [Strömholm 1996 and other references in this paragraph], so the method will only be described briefly here, mostly with respect to typical results obtained. The storage rings used for the studies are ASTRID (Aarhus, Denmark) [Jensen 1999], CRYRING (Stockholm, Sweden) [Zong 1999], TARN (Tokyo, Japan) [Tanabe 2000], and the TSR (Heidelberg, Germany) [Habs 1989]. Although the devices are called rings, they are shaped more like squares or rectangles with rounded corners. The basic feature of the device is the ability to contain ions electromagnetically for relatively long periods of time (of the order of a few seconds). This feature allows for vibrational, rotational and off–axis[†] translational cooling of ions prior to reaction; but, the cooling is radiative and is limited by the temperature of the walls. The ions travel around the ring at high energies, encountering different environments as dictated by experimental needs. The square–ish shape allows for sections of long, straight trajectories, and curved sections allow neutrals or selected ions to be extracted while the rest of the beam is bent around the orbital path. The apparatus gives the researchers great flexibility in terms of manipulating reactants and in observing products. Until very recently, storage rings were mainly used to study small HX^+, H_2X^+ ions and their deuterated analogs [Amitay 1999, Jensen 1999, Lange 1999, Schneider 2000a,b, Strömholm 1996, Zong 1999]. Even the most recent [Larson 1998, Semaniak 1998, several articles in Larsson 2000] studies have not been for very massive ions, and most concentrate on H_nX^+ variants ($n \leq 5$). A recent conference proceedings [Larsson 2000] contains articles discussing these experiments. Often, the interaction energies are very large (several eV to several MeV). But, lower energy interactions, as are comparable to the work presented in this document, are usually included [see previous list]. Since the numbers of ions recombining is low, these experiments do not observe photoemissions. But, they are able to deduce the energy

[†] e. g., perpendicular to the forward motion of the ions around the ring.

content of the products in some cases and, therefore, deduce their electronic state [Strömholm 1996, Zong 1999]. The studies nearly always concern themselves with calculation of recombination cross sections [all references in this paragraph]. They sometimes also report recombination rate coefficients [Amitay 1999, Zong 1999] and branching ratios for the various product pathways [Jensen 1999, Larson 1998, Semaniak 1998]. These studies keep the ions trapped in the ring prior to recombination long enough for vibrational relaxation by photoemission, but most do not determine the vibrational state of the ion directly. One study [Amitay 1999], however, selects for the initial vibrational state of the ion. The vibrational state is determined using a technique called coulomb explosion imaging (CEI). This technique strips the electrons off an ion very rapidly. The nuclei then move rapidly apart by coulombic repulsion (hence "explosion"). The researchers use the fragment pattern to determine the vibrational state of the nuclei at the point at which the electrons were stripped.

The new research reported in this dissertation employed a flowing afterglow, and details of design and use can be found in Chapter 2. The primary advantage of using a flowing afterglow, rather than other devices such as a stationary afterglow, is that the behavior of a reaction in time is stretched out along a flow–tube. This feature is particularly useful to kinetic studies. The familiar rate law describing a bimolecular recombination (here, e–i recombination):

$$Rate = \frac{\mathrm{d}[R]}{\mathrm{d}t} = -\alpha[e^-][XY^+] \qquad 11$$

is no longer written in terms of time, but of distance:

$$\frac{\mathrm{d}[R]}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}z} = \frac{\mathrm{d}[R]}{\mathrm{d}z} = -\frac{\alpha}{v}[e^-][XY^+] \qquad 12$$

where, R stands for either reactant, the electron, e^-, or the ion, XY^+, and [] indicates the concentration of the reactant. α is the rate coefficient, v is the velocity of the plasma, t is the reaction time, and z is distance, axially, along the tube. The disadvantage is that studies are complicated by flow effects such as diffusion. Since

the particles are charged, the species are lost to recombination at the walls as well as to processes occurring within the volume of the flow. The process governing the loss at the walls is called *ambipolar diffusion* and is discussed in texts dealing with plasmas [for example, Adams 1975, Drake 1996, Hazeltine 1998]. With a sufficiently high pressure ($\sim$ 1 torr) of a neutral carrier gas, such as helium, and a reasonably high number density of electrons ($\sim 10^{10} cm^{-3}$) loss to the walls is slow compared to loss due to recombination [Adams 1984]. In this case, the standard second–order kinetics, as written in equation 13 provides a sufficient description, with a well–known solution [for example, Atkins 1998]. If we use position references of z_1 for the initial position along the tube and z_2 for the final position along the tube, then an expression relating electron concentration and position to the recombination coefficient is:

$$\ln\left(\frac{[XY^+]_{z_1}[e^-]_{z_2}}{[e^-]_{z_1}([XY^+]_{z_1} - [e^-]_{z_1} + [e^-]_{z_2})}\right) = \alpha\frac{[e^-]_{z_1} - [XY^+]_{z_1}}{v}(z_2 - z_1) \qquad 13$$

Variations on this expression can be formulated for dependence on ion density as a function of position or for both densities as a function of position. Such variations are useful for different experimental setups. The formulation given in Equation 14 is useful in cases where it is convenient to measure electron density as a function of position (such as with a movable Langmuir probe), but where the ion density could only be known at the start position, perhaps due to the presence of multiple ion types. In cases where there is only one positive ion type and there are no negative ions, it is acceptable to assume that the ion density equals the electron density, since the plasma is quasi–neutral. It is, then, only necessary to ensure that sufficient XY precursor is added to make certain that all the positive ions are converted to XY^+. If this is the case, the reaction reduces to one of general type:

$$\frac{\mathrm{d}[R]}{\mathrm{d}x} = -\frac{\alpha}{v}[e^-]^2 \qquad 14$$

to which the solution is well known as:

$$\frac{1}{[e^-]_{z_2}} - \frac{1}{[e^-]_{z_1}} = \alpha\frac{(z_2 - z_1)}{v} \qquad 15$$

For a thorough discussion of the usefulness of this sort of technique, see the temperature dependence work by Adams and Smith [Adams 1984]. In 1987, Johnsen [Johnsen 1987] published a comparative review of recombination rate coefficients available at that time. Although many of the recent rate coefficient determinations have come from storage ring experiments, the flowing afterglow has still been used in a number of studies. For example, rate coefficients have been determined recently in a flowing afterglow for recombinations of H_3^+, D_3^+ and HCO^+ [Laubé 1998a]. Another group has been studying the recombinations of larger hydrocarbons noting isomeric differences where possible [Rebrion–Rowe 2000a,b].

A few experiments in afterglows (either flowing or stationary) have been concerned with atomic or molecular products of DR but not with direct photoemission from those products. These experiments often involve laser–induced fluorescence (LIF) of the products [Herd 1990, Adams 1991, Gougousi 1997a,b] and/or vuv absorption [Kley 1977, Herd 1990, Adams 1991, Williams 1999]. In the LIF studies, the OH radical is monitored as a product of recombination. The OH density is determined by excitation of molecules from the ground ($X^2\Pi$) state up to an excited state ($A^2\Sigma^+$) and then observing the light emitted (fluorescence) when the OH returns to its ground state. This method was initially used to quantify OH in the $X^2\Pi$ v=0 and v>0 states [Herd 1990, Adams 1991]. More recently, yield has been determined for relative populations of the v=0 and v=1 vibrational levels in OH [Gougousi 1997a,b]. These latter studies did not determine the contribution for levels greater than v=1. The absorption studies were used to examine H atom production [Herd 1990, Adams 1991] and N (^2D) production [Kley 1977]. If one considers the storage–ring experiments, a larger number of products have been determined, but even then, the body of information is hardly extensive.

The number of studies that have examined photo–emissions as products of recombination is also small relative to the number of possibilities. This number, however,

appears to be growing, so there might soon be a considerable body of information on the subject. One of the earliest mentions of the phenomenon comes from a study of emissions in a flowing afterglow [Taieb 1977]. In this study, the experimenters injected neutral molecules into a plasma flow and observed the resulting emissions. They found that some of the emissions came from neutral species that were smaller than the injected molecules. The researchers suggested that those emissions might have been the result of recombinations of larger ions with electrons. Based on experiments done since then, it now seems very likely that they were correct in that assertion. Thus far, there have been no observations of emissions from triatomic or larger products. Only two papers, both by Zipf [1979, 1980b] report observation of atomic emission as a result of recombination. Table 1-3.1 lists ions and products for studies since the observation in 1977 by Taieb and Broida. Several of the studies listed in the Table will be discussed in greater detail in Chapter 4 of this document.

Table 1-3.1

Emitting States for Observed Photoproducts of Recombination

Molecule	**State(s)**	**Parent Ion †**	**Reference**	**Note**
O	^1S	O_2^+	Zipf 1979, 1980	
CH	$A^2\Delta$, $B^2\Sigma^-$, $C^2\Sigma^+$	CH_4	Tsuji, 1991	
	$A^2\Delta$, $B^2\Sigma^-$	CH_4 + H_2	This work	a, b
CO	$A^1\Pi$	CO_2^+	Valée, 1986	a
			Tsuji, 1995	
			This work	a
		HCO^+/COH^+	This work	a, b
		HCO_2^+	This work	a, b
	$a^3\Pi_r$	HCO^+	Adams, 1994a,b	
			Butler, XXXX	
			This work	
		DCO^+	Butler, XXXX	
		HCO^+/COH^+	Johnsen, 2000	
			This work	b
		CO_2^+	Tsuji, 1995	
			Skrzypkowski, 1998	
			Johnsen, 2000	
			This work	a
		HCO_2^+	This work	b
	$a'^3\Pi_r$, $d^3\Delta_i$, $e^3\Sigma^-$	HCO^+/COH^+	Johnsen, 2000	
		CO_2^+	Tsuji, 1995	
	$a'^3\Pi_r$, $d^3\Delta_i$	CO_2^+	This work	
		HCO_2^+	This work	a, b
CS	$A^1\Pi$	CS_2^+	This work	b
	$a^3\Pi_r$	CS_2^+	This work	b

Table continues

† If neutral parents are listed, the exact identity of the parent is complex or uncertain. The neutrals listed are the species injected into the plasma.

NH	$A^3\Pi_i$	$NH_3 + H_2$	This work	b
		HNO^+	This work	b
		N_2OH^+	Foley, 1993	
			Johnsen, 2000	
	$c^1\Pi$	$NH_3 + H_2$	This work	a, b
		N_2OH^+	This work	b
N_2	$B^3\Pi_g$	N_2H^+	Adams, 1994a,b	
			Butler, XXXX	
		N_2D^+	Butler, XXXX	
		N_2O^+	Tsuji, 2000	
		N_2OH^+	This work	a, b
NO	$A^2\Sigma^+$	HNO^+	Johnsen, 2000	
	$A^2\Sigma^+$, $B^2\Pi_r$	N_2O^+	Johnsen, 2000	
		HNO^+	This work	b
OH	$A^2\Sigma^+$	H_2O^+	Sonnenfroh, 1993	c
			Johnsen, 2000	
		O_2H^+	Foley, 1993	
		N_2OH^+	Foley, 1993	
			Johnsen, 2000	
		HNO^+	This work	b
		HCO_2^+	Johnsen, 2000	
			This work	b
SH	A?	H_3S^+	This work	d

Notes:

a: Apparent vibrational excitation in recombining ion.

b: Tentative: likely due to recombination, but other sources are possible.

c: $T_e \sim 1000$k.

d: Electronic state uncertain.

CHAPTER 2

EXPERIMENTAL APPARATUS AND DESIGN

The first part of this chapter, Section 2-1, describes the physical design of the experimental apparatus. First, the overall layout of the system is presented. Following that, the major components are discussed separately and in moderate detail.

The chemical aspects of the experiments are described in Section 2-2. As in Section 2-1, a general overview of the design is followed by a more detailed discussion of important reactions.

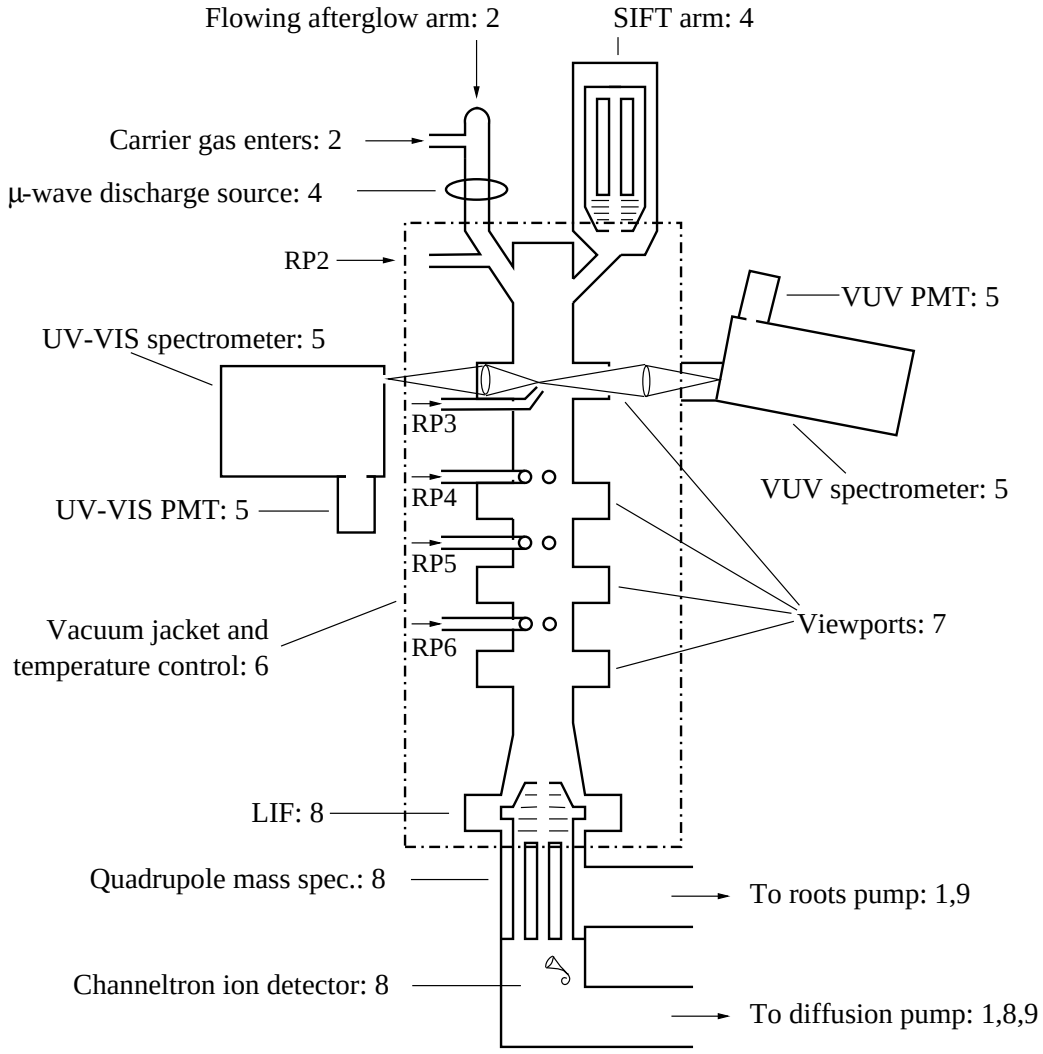
2-1 Experimental Apparatus

The experiments described in Chapters 3–5 were carried out in a *flowing afterglow* (FA), a schematic of which is given in Figure 2-1.1. A flowing afterglow consists of a tube, often stainless steel, through which one or more gases flow. Reactions begin, generally, at or near the upstream end, where ionization is produced in a neutral gas, called a *carrier* gas. In these studies, He was used exclusively as the carrier gas, but other gases, N_2 for example, can be used. The carrier gas makes up the bulk of the gaseous pressure in the apparatus and acts as the medium in which the chemical reactions take place. Since the gas also inhibits, or "buffers," diffusion of the reacting species, it is sometimes referred to as a *buffer* gas, but will be referred to here exclusively as a carrier gas. The partially ionized carrier gas then flows down the length of the tube and carrier is pumped away at the downstream end. To control the chemistry occurring in the flow, additional gases are added at various points along the tube.

Figure 2-1.1

Gas manifolds (not shown here): 1

Inlet ports RP1-RP6: 3



Pumping system (not shown here): 9

Figure 2-1.1: Schematic of the flowing afterglow apparatus. Numbers following labels indicate the subsection(s) of Section 2-1 in which that component is discussed. The figure is not drawn to scale, but is somewhat representative of the relative sizes of the real components. To reduce clutter, the gas manifolds, gas flow systems and pumping systems are illustrated separately in Figures 2-1.2, 2-1.3 and 2-1.7.

This section is devoted to discussions and figures describing the flowing afterglow and ancillary equipment used in the studies presented here. Figure 2-1.1 is a schematic of the central portions of the apparatus, i. e., the "reaction vessel." In the figure, portions of the apparatus are numbered; discussions of numbered portions appear on the following pages in the order indicated by the numbering. Portions of the apparatus that are not pertinent to flowing afterglow reactions, but which are included for completeness' sake, will only be discussed briefly here. Some of the systems needed for experimentation are not included in this schematic because their inclusion would render the figure unnecessarily cluttered. Separate figures (see below) are devoted to those systems.

2-1.1 The Gas Manifolds, Flow Control and Reagent Purification

This subsection describes the systems that control the movement of gases into and out of the flowtube. Also included here is a discussion of gas purification.

Coarse flow of gases into the system is handled by two gas manifolds: one used primarily for helium and the other, primarily for reagent gases. Gas flows are more precisely controlled by a series of valves and flow controllers within a gas manifold. During experimentation, the outflow of gases from the flowtube is controlled by use of a Roots pump. Gases are removed from other parts of the system, and from the flowtube when idle, via a series of diffusion pumps. Gases that are input in high concentrations (He, Ar and H_2) are purified by passage through one or two molecular sieves. These components of gas–flow control — the gas manifolds, inflow and outflow of gases and reagent purification — are discussed separately.

The gas manifolds

The gas manifolds consists of two sections. The first section, the *helium manifold* (Figure 2-1.2), primarily controls the flow of the carrier gas, which, in this study, is exclusively helium. The other section, the *reactant manifold* (Figure 2-1.3), directs

Figure 2-1.2

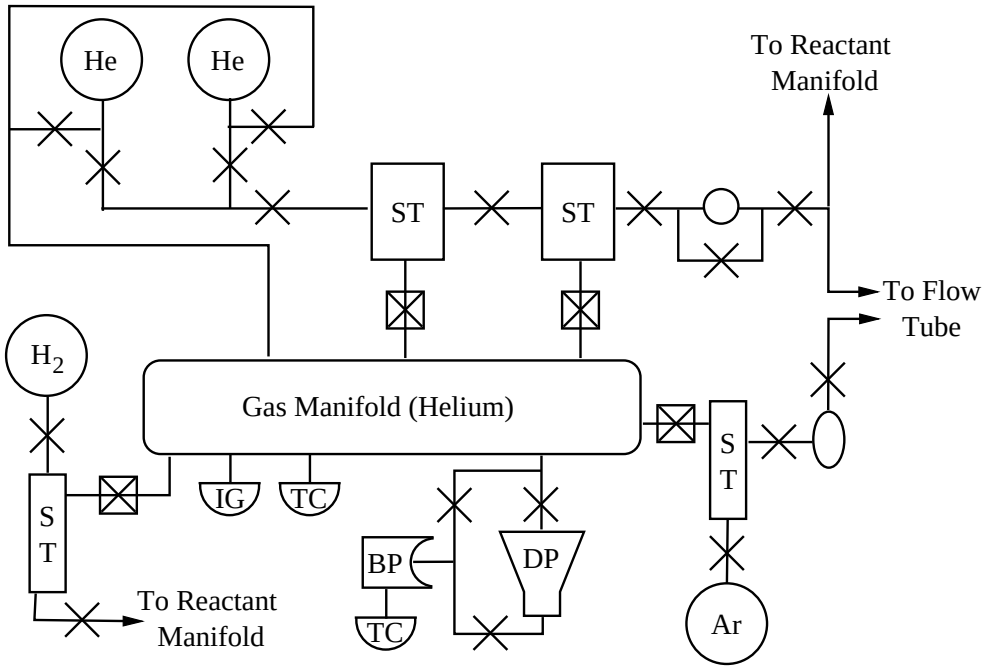


Figure 2-1.2: Schematic of the Helium Manifold.

Legend for Figures 2-1.2 & 2-1.3:

×, ⊠ Shut–off valve ⬯ Needle valve ○ Flow controller —— Gas line

→ Flow direction ⬭ Capillary tube ◯ Gas cylinder and regulator

DP Diffusion pump BP Backing pump GPV Granville–Phillips valve

Pressure gauge **TC** Thermocouple Gauge ST, S T Sieve trap (large, small)

$\mathbf{P}_{in}$ Pressure into capillary $\Delta\mathbf{P}$ Pressure difference across capillary **IG** Ion Gauge

He Helium **Ar** Argon $\mathbf{H}_2$ Hydrogen **RG** Reactant Gas **RP** Reactant Port

the flow of almost all the other gases. The exceptions are hydrogen and argon. Since they are often input in larger quantities and usually purified in cooled molecular sieves (see Table 2-1.1), they are also controlled from the helium manifold. Helium and hydrogen can also be, and routinely are, directed through the reactant manifold.

Figure 2-1.3

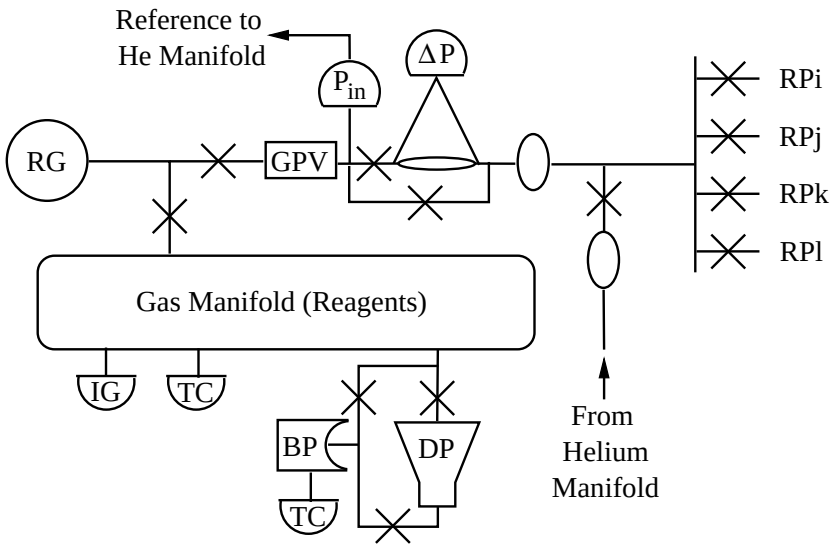


Figure 2-1.3: Schematic of the Reactant Manifold gas–inlet systems. Only one inlet system (of four) is illustrated here. Each of the four systems leads from one cylinder to a set of four reactant inlet ports. Volatile liquids are injected into the system by replacing the gas cylinder and regulator with a vial of the liquid: liquid vapor pressures are low enough that a regulator is not necessary. A legend describing the symbols used here is given with Figure 2-1.2 (previous page). The letters i, j, k and l associated with the reactant ports (RP's) refer to any four of the reactant ports RP1-RP6.

Both manifolds are designed (Figures 2-1.2 & 2-1.3) so that each section can be shut off from all the others. This allows for evacuation or modification of each section at any time, but especially during experimentation, with no significant disturbance of flow through other sections. Each system attaches separately to a primary manifold. The primary manifold (labeled "Gas Manifold" in the figures) is roughed by a mechanical pump and then brought to a pressure on the order of 10^{-5} to 10^{-7} Torr by a diffusion pump (see subsection 2-1.9 for more technical information on pumps).

Shutoff valves are used wherever gas flow needs only be turned on or off. Mechanical needle valves, alone or in conjunction with more precise instruments (generally a Granville–Phillips™ valve), are used where reagent gas throughput needs to be more carefully regulated.

Bulk gas flow control

During experimentation, the bulk throughput of helium is regulated by an MKS™ flow controller. He pressure in the tube is controlled by throttling the Roots pump. For typical experimentation, the helium throughput was 222 Tls^{-1} (16.0 slm, standard liters per minute, as measured on the MKS™) and the Roots pump was throttled such that the pressure inside the tube was 1.8 Torr. Since the argon needed only be injected in excess, rather than in precise quantities, its input was typically controlled by a needle valve. The valve was opened until the reading on the Baratron™ pressure gauge changed by about 1 mT. In a number of cases H_2 was also added using this method.

Reactant gas flow control

The injection of other reagent gases needed to be controlled much more precisely. To this end, a combination of a Granville–Phillips™ valve and a mechanical needle valve was used, together with measurements of pressure in a calibrated capillary, to determine reagent gas flow. The pressure into the capillary tube (p_1 or p_{in}, below) was primarily regulated by the Granville–Phillips™ valve. The pressure drop across the capillary ($p_1 - p_2$ or Δp, below) was primarily controlled by adjusting the mechanical needle valve. Using this combination the throughput appropriate to each experiment was achieved. The range of injection throughputs is about $3{\times}10^{-4}$ to 2 Tls^{-1}.

The throughput, Q, of a viscous gas can be measured by using a variant of Poiseuille's formula:

$$Q = \frac{(p_1^2 - p_2^2)\pi r^4}{16\ell\eta} = \frac{(p_{in} - \frac{\Delta p}{2})\Delta p\,\pi r^4}{8\ell\eta} \qquad 16$$

where r and ℓ are the radius and length of the tube, η is the viscosity of the gas, p_1 or p_{in} is the pressure into the capillary, p_2 is the pressure out, Δp is pressure difference across the capillary, and π has its usual trigonometric meaning.

This method worked well for most reagent gases, since they were injected at pressures well suited to the Granville–Phillips™ valves and the pressure measuring devices. The attaching gas (see Section 2-2), however, needed to be injected at pressures near the low end of the usefulness of the valves. At these low pressures, without servo devices installed, the pressure measurements for the gases tended to rise or fall slowly, requiring continual manual adjustment during that phase of experimentation. Also, at such low throughputs, the flow becomes non–viscous, and Poiseuille's equation (above) is no longer valid. Mixtures of low–flow gases with helium can be used to overcome this problem.

Gas purification

For cleaning the gases, traps filled with molecular sieve were used. These *sieve traps*, are constructed of subdivided, stainless steel tubes (see Figure 2-1.4) filled with the appropriate sieve material (see Table 2-1.1). The specifications given in the table refer to the manufacturer's description of the material. The traps, except for the one used to purify argon, are cooled by liquid nitrogen. Since the boiling point for argon is higher than that for nitrogen, its trap was immersed in a methanol bath cooled to around -90° C by the cooling finger of a two stage refrigerator. In all cases, impurity gases, such as O_2, H_2O, etc., condense onto the cold sieve material, purifying the primary gas.

Figure 2-1.4

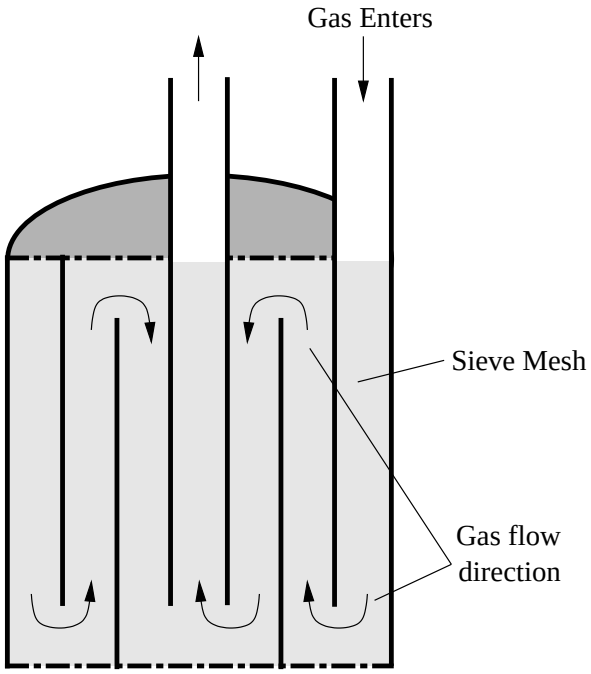


Figure 2-1.4: Cross–sectional diagram of the sieve trap used to purify the helium used in these studies.

Table 2-1.1

Reactant Purification

Gas	Helium	Argon	Hydrogen	Reactant Gases
Purification Type	Molecular Sieve 5A & 13X	Molecular Sieve 5A & 13X	Molecular Sieve 3A	Generally Not Purified

Since the pressure of helium in the flowtube is so much larger than that for other gases, it is necessary that the helium be very pure. During experimentation, the number density of helium is several orders of magnitude higher than that of the reacting species. Helium that is 99.95% pure contains impurities at far higher concentrations than that of the reactants. In order to sufficiently purify the helium, it

is passed, successively, through two identical, specially designed sieve traps (see Figure 2-1.4). Each trap is designed such that the helium makes four full passes through the sieve material before exiting the trap. Since the reactant gases are used in such relatively small amounts, any impurities they have are generally not significant, and, for this study, they were usually not purified. However, it should be noted that an impurity more reactive than the species of interest, even when present in small concentration, can become significant. In these cases, higher purity reagents are used.

Figure 2-1.5

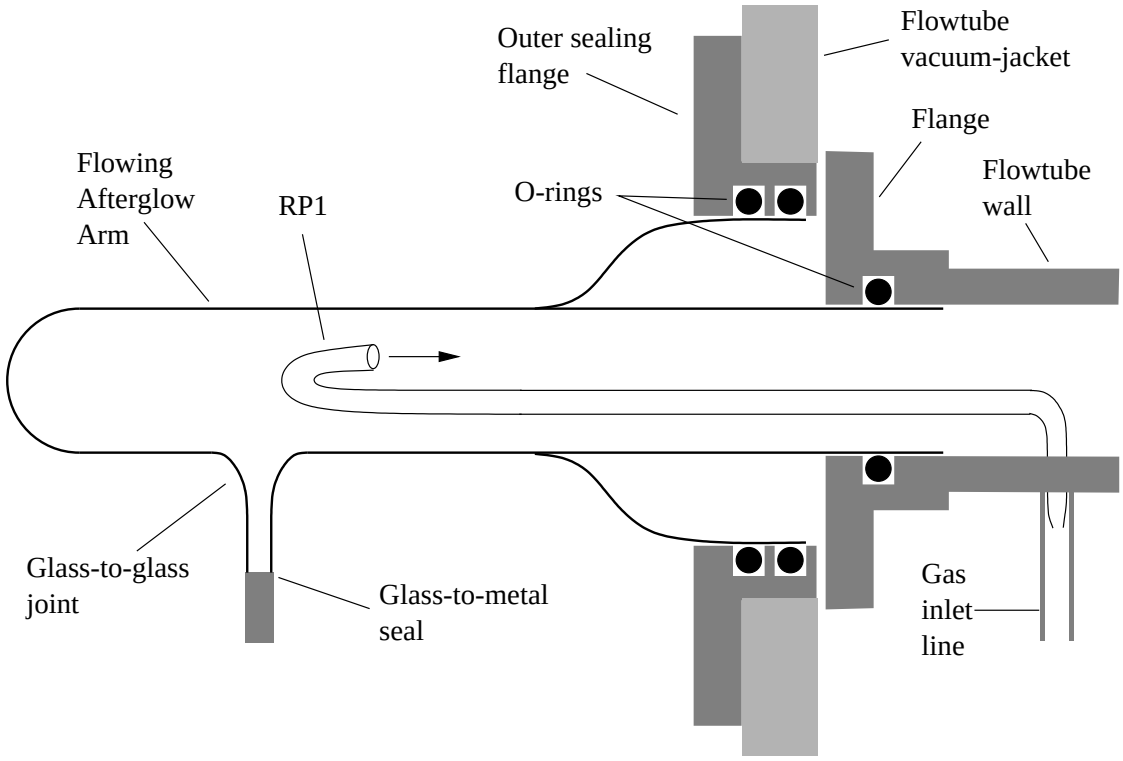


Figure 2-1.5: Schematic of the flowing afterglow arm of the flowtube and RP1, the flowing afterglow's helium inlet port. Both the flowing afterglow arm and RP1 are made of pyrex. The microwave discharge cavity (see Figures 2-1.1, 2-1.7 and 2-1.8) is not illustrated here.

2-1.2 The Flowing Afterglow Arm

The primary point of entry for the helium, and its sole point of entry as a carrier gas, is through the flowing afterglow arm of the flowtube. The flowing afterglow arm is a specially made pyrex tube (Figure 2-1.5). Near the upstream end, helium is let in through a port that attaches to the gas manifold via a glass–to–metal seal. At the downstream end, the flowing afterglow arm splits into two concentric tubes. The inner tube fits into the flowtube, and the outer tube, the vacuum jacket.

Helium can also be introduced through a pyrex reactant port, RP1, situated inside the flowing afterglow arm (see figure). RP1 consists of a bent $\frac{1}{4}''$ pyrex tube that fits into an inlet port just inside the flowtube (Figure 2-1.5). Its purpose is to begin the helium flow in the center of the flowing afterglow arm. With this construction, the helium upstream of the port is hardly flowing, whereas the helium downstream of it is flowing rapidly. Thus, by controlling the position of the microwave cavity (Section 2-1.4) along the length of the FA arm, large variations in the downstream ionization density can be produced. Its glass construction ensures that it does not interfere with the oscillation of the microwaves within the discharge cavity. The inlet for RP1 (see Figure 2-1.5) is situated inside the flowtube to facilitate temperature control if required in an experiment.

2-1.3 Inlet Ports

At the time these experiments were conducted, there were six inlet ports along the length of the flowtube. Reactant ports (RP's) RP1, RP2 and RP3 were used in this study. RP4, RP5, and RP6, were not used in this study, but are presented here for completeness. Schematics of RP2 and RP3, and a representative for RP4-RP6 can be found in Figure 2-1.6. Figure 2-1.7 contains photographs of RP1 and RP3. Approximate positions of the ports are illustrated on Figure 2-1.1. A schematic of RP1 was given in Figure 2-1.5, with accompanying discussion in Section 2-1.2.

Figure 2-1.6

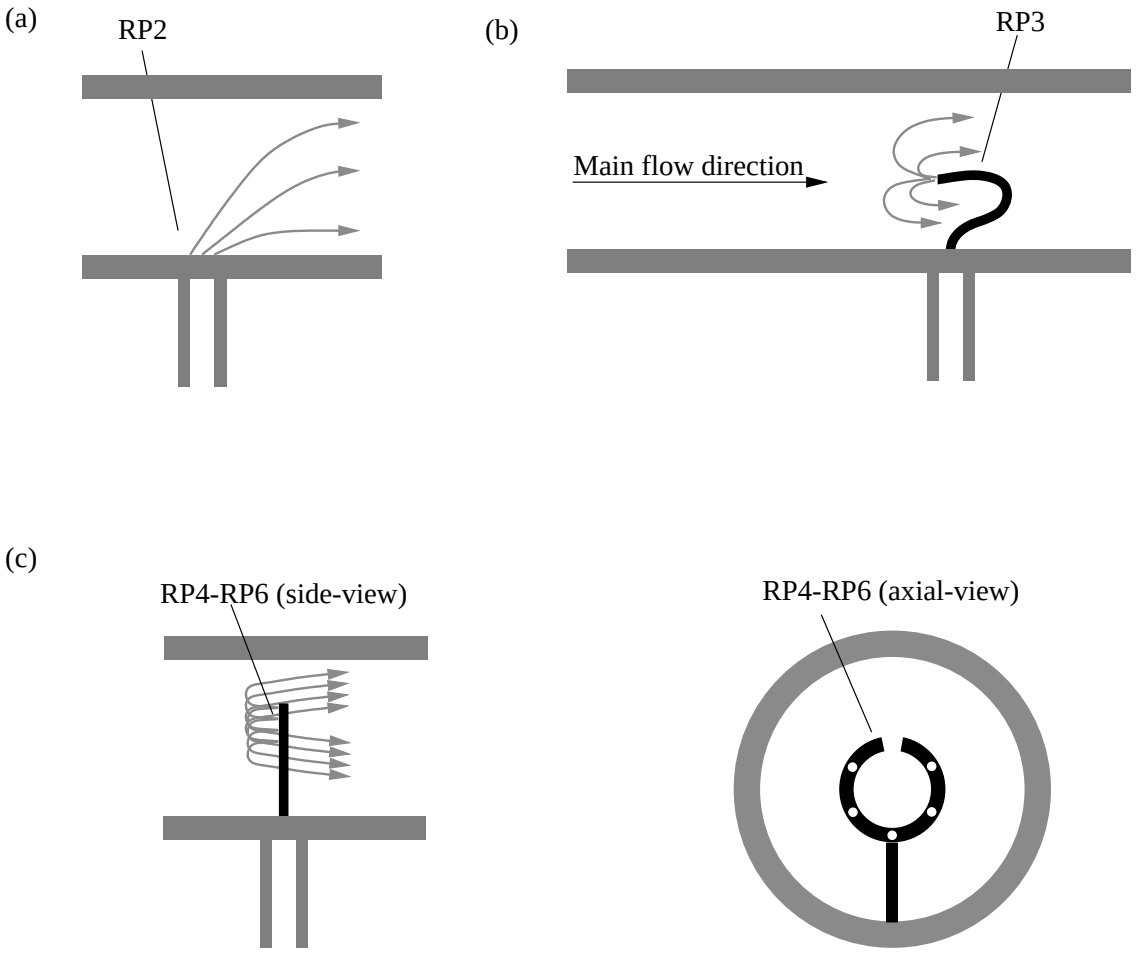


Figure 2-1.6: Schematics of the inlet ports: (a) RP2, which is simply an opening in the side of the flowtube; (b) RP3 is a tube 0.065″, outer diameter; the curvature directs the gas flow into the center of the flowtube in a direction opposite the helium flow. (c) RP's 4-6 are closed–end tubes 0.060″ outer diameter, bent into a partial circle and bored with six 1 mm holes. Note that the ring injection speeds the mixing of reactant gases evenly into the flow. In (b) and (c) a small flow of purified He is often mixed with the reactant gas to speed injection into the flowtube.

Reactant port two (RP2) is merely an opening in the wall of the flowtube (Figure 2-1.6 a). It is used to inject gases that alter the plasma composition necessary for

Figure 2-1.7

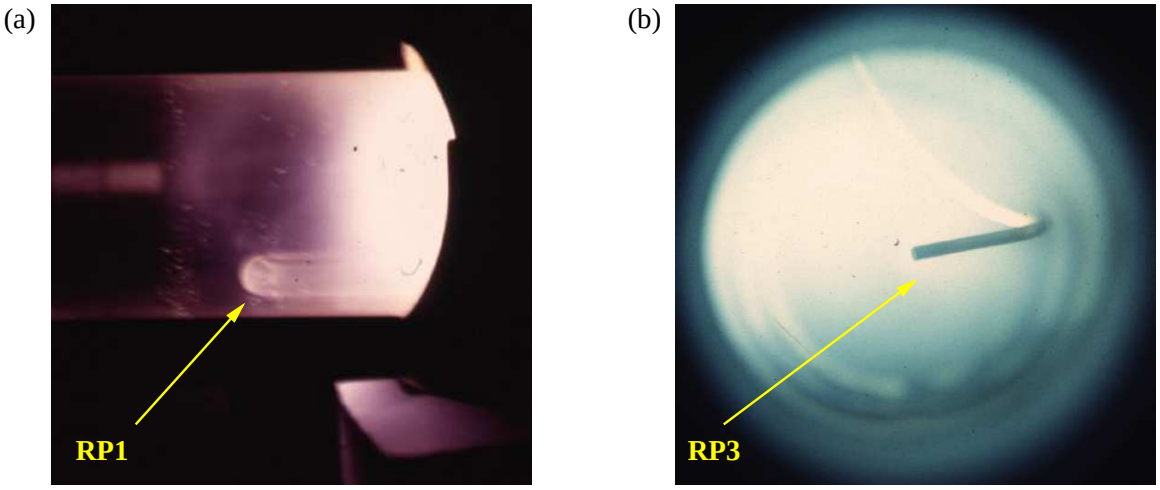


Figure 2-1.7: Color photographs of RP1 (a) and RP3 (b). (a) The dark object surrounding the flowing afterglow arm (in which RP1 is suspended) is the microwave discharge source. The color of the gas within and surrounding RP1 is due to excitation in the helium as it flows through the μ-wave cavity. (b) The greenish glow surrounding RP3 is due to the combination of helium emissions that are propagated down the flowtube and emissions arising from the addition of argon at RP2. No reagents are being added at RP3 (see Chapters 3 and 4 for photographs of RP3 when a reactant gas is being added).

the reactions occurring downstream at ports RP3-RP6. A simple opening is sufficient because the gases do not need to be mixed rapidly into the bulk flow. In these studies, the only gases injected through RP2 are the attaching gas (see Section 2-2) and argon.

Reactant port three (RP3) consists of a single, thin–walled, 0.065″ diameter tube (Figure 2-1.7 b). The tube is bent so that the gas in injected into the flowtube in a direction opposite the flow of the carrier gas (Figure 2-1.6 b). This reverse injection causes the portion of the reaction observed by the spectrometers to be maximized. A perspective equivalent to that of the spectrometer is shown in the photograph in

Figure 2-1.7 b. The profile of the injection plume is either roughly paraboloid or spherical depending on the nature of the interaction between the injected species and the oncoming plasma (see Chapter 5).

Reactant ports four through six (RP4-RP6) are *ring ports*. They are made from a piece of $0.065''$ diameter tubing that has been bent into a circle and into one side of which several holes have been bored (Figure 2-1.6 c). The gas exits the port through these holes. The holes face upstream so the gas exits contra–flow (like RP3). The purpose of this design is to minimize the time required for the injected gas to mix with the other gas(es) in the flow.

2-1.4 Gas Ionization

Gas ionization in the flowtube is generated in either a microwave discharge cavity situated around the FA arm or in an ionization source appropriate for use with a SIFT (defined shortly). It is not practical (or, so far, desirable) to have ionization produced in both simultaneously. Only the microwave cavity is of interest to this study, and it is discussed in some detail below. SIFT stands for "Selected Ion Flow Tube." A SIFT selects single ion types by mass and injects them into a flow of neutral He carrier gas. Ions from a remote ion source are selected using a quadrupole mass spectrometer. A series of electrical lenses then directs the stream of single–mass ions into the flowtube. The apparatus has been described elsewhere in detail [Adams 1988a].

The microwave discharge cavity was made specifically for this flowtube. It is constructed of two hollow cylindrical brass sections. The two sections (Figure 2-1.8) fit top–into–bottom and are cut so that the two sections fit loosely around the glass flowing afterglow arm. A power supply (Opthos™) provides a source of intense microwave radiation. Adjustment of a tuning rod and screw (see diagram) tune the cavity for resonance with the microwaves, helping to maximize absorption of the radiation by the flowing helium. The discharge source works by causing electrons to be accelerated

Figure 2-1.8

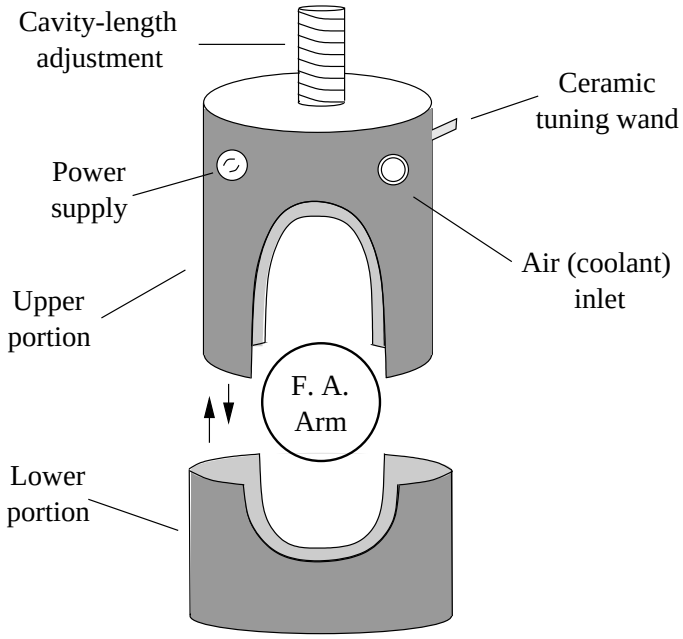


Figure 2-1.8: A three dimensional drawing of the microwave discharge source. The top piece fits into the bottom piece, as shown, and around the flowing afterglow arm.

back and forth within the microwave field generated by the resonant oscillation of the microwave radiation. A spark provided by a tesla coil provides an initial supply of electrons to the cavity to initiate the discharge. A few of these electrons are trapped inside the oscillating microwave field. As they move back and forth, they impact the flowing helium atoms, causing electronic excitation and ionization.

The position of the microwave discharge source along the flowing afterglow arm is adjustable. The position of the cavity affects the ionization density in the tube (see Section 2-1.2). When higher ionization densities are desired, the cavity is slid to its most downstream position (next to the flowtube's vacuum jacket). Higher ionization densities are desirable for recombination studies. But, higher ion densities are also likely to cause damage to the ion detection system. So, whenever mass spectra are being taken, the cavity is positioned (approximately) around the crook in

RP1. Consequently, mass spectra are not taken while recombination spectra are being recorded. However, the flow control and pumping systems are stable and the number and type of ions present in the tube is not likely to change during a typical run. On several occasions, this assumption was tested (by taking mass spectra between or after recombination experiments) and was found to be valid.

2-1.5 The Spectrophotometers and Supporting Equipment

The two monochromators used in these experiments cover a wavelength range from $\sim$ 140 — 800 nm, with about 200 nm of wavelength overlap for comparison between the two: the vacuum ultraviolet (VUV) spectrometer has an upper limit of around 400 nm due to the range of the photomultiplier tube, and the ultraviolet–visible (UV–VIS) spectrometer's efficiency has a lower limit of about 180 nm due to absorption by the air. Both spectrometers disperse light with diffraction gratings, detect photons using cooled photomultiplier tubes and record numbers of photons using gated photon–counting devices interfaced to computers.

Table 2-1.5.a

Spectrometer Specifications

Spectrometer	**UV–VIS**	**VUV**
Grooves per Millimeter (gr/mm)	1800	3600
Blazing Wavelength(nm)	250	130
Path Length (m)	0.66	1.0
Wavelength Range (nm)	180-800	140-400
Typical Device Pressure (Torr)	~ 760	$\sim 5 \times 10^{-5}$
Photomultiplier Tube (Thorn EMI)	98160B	9426B
Gated Photon Counter (Stanford Research Systems)	SR400	SR400

The VUV spectrometer is evacuated via a turbomolecular pump to $\sim 5{\times}10^{-5}$ Torr (a turbomolecular pump is used here instead of a diffusion pump to eliminate the condensation of oil onto the diffraction grating, which would reduce its efficiency).

Table 2-1.5.b

Typical Operating Conditions

Spectrometer	**UV–VIS**	**VUV**
Slit Width	0.27 mm	0.81 mm
PMT Power	1.85 kV	1.85 kV
Scan Rate	1.5 nm/min (180-400 nm) 5.0 nm/min (400-800 nm)	1.5 mn/min
Count Period	4 sec (180-400 nm) 3 sec (400-800 nm)	4 sec

The diffraction gratings for the two spectrometers are interchangeable. The blaze and number of grooves per millimeter are chosen for efficiency in the desired wavelength ranges. The grooves are blazed to enhance diffraction in the wavelength region and diffraction order of interest. Table 2-1.5.a lists specifications for the two systems, and Table 2-1.5.b summarizes typical operating conditions.

2-1.6 The Vacuum Jacket and Temperature Control

The structure of the flowtube, including the vacuum jacket, is illustrated in Figure 2-1.1. The vacuum jacket is used for thermal insulation during experiments. In these studies, it served mainly to minimize the effects of any leaks that might have existed along the length of the flowtube and to provide a light–proof enclosure for the flowtube. The tube itself can be heated with heating tape or cooled by tubes filled with coolant, thus enabling studies of the temperature dependence of reactions.

2-1.7 Viewports

Viewports are positioned along the flowtube at several locations (see Figure 2-1.1). The ports of interest to this study are those that allow for observation of RP3. In addition to the ports indicated on the figure, there are ports in equivalent positions above and below the flowtube (only ports to either side of the tube are shown in the

figure). The photograph in Figure 2-1.7(b) was taken from the viewport directly above RP3. Spectra were taken through the side ports, as is shown in Figure 2-1.1. The material for each viewport window was chosen based on spectral transmission. For example, windows associated with the VUV spectrometer are made of magnesium fluoride (MgF_2), which transmits light down to about 140 nm.

Table 2-1.6

Mass Spectrometer Specifications

Mass Range (typical, amu)	*Type*	*Detector*	*Typical Working Pressure (Torr)*
0 - 200	Quadrupole	Channeltron	$< 1 \times 10^{-4}$

2-1.8 Ion Mass Analysis and Detection

The detection system is composed of a quadrupole mass filter with a Channeltron™ channel electron multiplier as detector (see Figure 2-1.1). Signals from the Channeltron are interpreted by a pulse counter which is interfaced to a computer that records and displays the data. See figures Appendix D for sample data output from the mass spectrometer. While the flowtube is idle, a diffusion pump maintains vacuum within the flowtube and the detection chamber. But, during experimentation, the primary flow is directed to a Roots pump. An electronic lens 0.3 mm from the orifice draws ions from the flow into the detection region. Here, the diffusion pump previously used to evacuate both flowtube and detection chamber keeps the detection chamber under sufficient vacuum for operation of the mass spectrometer and ion detector.

2-1.9 Pumping Systems

In all but two cases, the various parts of the system are pumped by diffusion pumps backed by standard mechanical pumps. The two exceptions are the turbo-

Figure 2-1.9

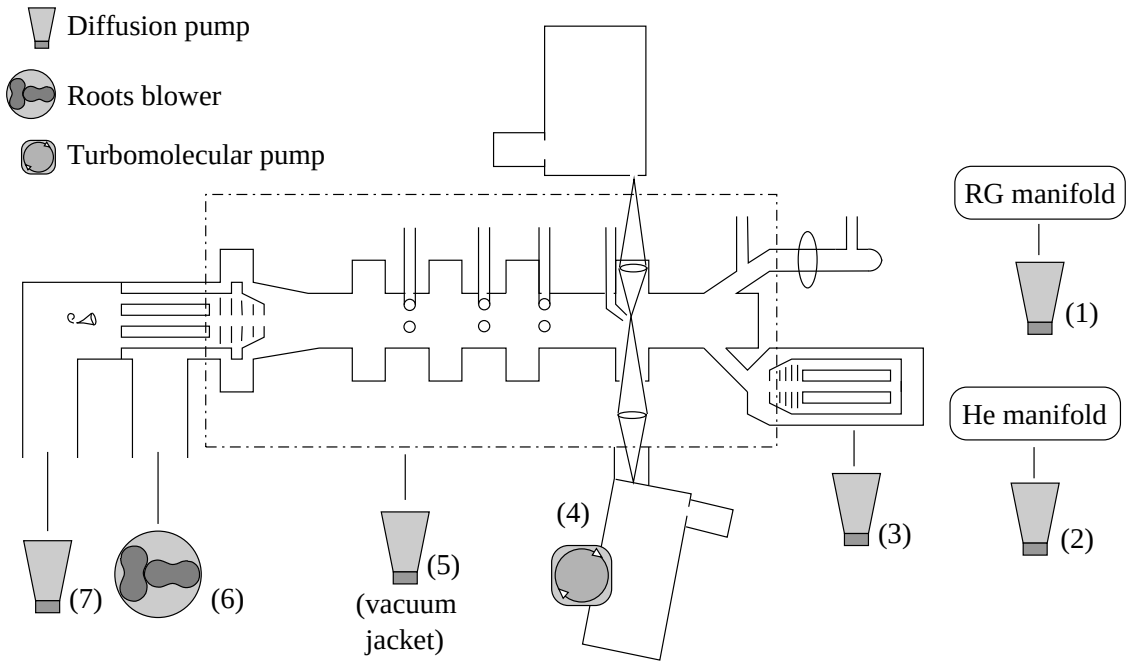


Figure 2-1.9: Schematic of pumps and their approximate positions along the apparatus. See the legend for pump types represented by the various symbols. The numbers assigned to the pumps correspond to those in Table 2-1.7. Each pump shown here requires a backing pump, specifications for which are in Table 2-1.7.

molecular pump used for the VUV spectrometer (see section 2-1.5) and a large Roots blower used during experiments to maintain a steady pressure for constant throughput of carrier and reactant gases. Pump positions, sizes, approximate loads, pumping speeds and other information are contained in Figure 2-1.9 and Table 2-1.7

2-1.10 Computer Interface and Supporting Software

Except during initial tests of equipment, all data were collected on–line using computer interfaces (RS232 and IEEE) to the gated photon counter for data analysis and to the mass and optical spectrometers for component control. All software used was written by group members in Quick Basic.

Table 2-1.7

Pump Number	*Section*	*Pump Type*[a]	*Model Number*[b]	*Backing Pump Size (in)*
1	Reactant Manifold	DP	63	5
2	Helium Manifold	DP	63	5
3	Sift Chamber	DP	100	8
4	VUV Spectrophotometer	TMP	STP300	12
5	Tank	DP	63	5
6	Flowtube	Roots	17215	—
7	Flowtube/Detection	DP	160	18

[a] DP = Diffusion Pump; TMP = Turbomolecular Pump. [b] Diffusion pumps and backing pumps manufactured by Edwards™; turbomolecular pump manufactured by Seiko Seiki; Roots pump manufactured by Stokes Vacuum.

2-2 Experimental Design

This section outlines the sequence of reactions used to control the chemistry occurring in the flowtube. The reaction scheme is well established, and similar descriptions have appeared elsewhere [Adams 1992]. Discussions of apparent deviations from this chemistry are given in Chapter 4.

The goal of these experiments is to observe and quantify photons emitted as a result of the reactions of small, polyatomic ions with electrons. Such reactions are generally highly exothermic. If only the exothermicity is considered, a variety of products is possible, but, not all possible products are, in fact, observed. Reactions 2 and 3 describe two possible outcomes from a hypothetical recombination.

$$XYZ^+ + e^- \rightarrow XY^* + Z \qquad 17$$

$$\hookrightarrow XY + h\nu$$

$$\rightarrow X + Y + Z^* \qquad 18$$

$$\hookrightarrow Z + h\nu$$

In order to observe emissions from recombination, especially weak emissions from less–populated states or forbidden transitions, it is necessary to have a relatively large number of recombination reactions occur in a relatively small volume. Arranging this is a difficult task, and the primary challenge in these experiments. Polyatomic ions react with electrons very rapidly (with rate constants on the order of 10^{-6} cm^3s^{-1}. [Johnsen 1987]). So, it is necessary to keep the polyatomic ions and the electrons separated until they are in a location such that the results of their reaction can be recorded.

This is achieved by creating a stable, flowing plasma that is not recombining and then injecting a neutral reagent into the the flow in a region convenient to observation. Initially, the flowing plasma contains only free electrons and positive atomic ions† within a neutral, helium carrier gas. If an unprotonated recombining ion is desired, the appropriate parent polyatomic gas is injected into the plasma at the point where the plasma flows through a region observed by the spectrometers. The atomic ions charge–transfer rapidly to the polyatomic molecules, and the newly–formed polyatomic ions react with the ambient electrons almost immediately. When a protonated recombining ion is desired, the procedure is similar except that H_2 is also added in great excess. Details of these procedures are discussed below.

With the reactions thus confined mainly to one region of the flow, emissions from recombination can more easily be observed. However, there are other species besides the recombining ions present in the flow, many in excited states. In any spectra taken, emissions from processes associated with these species, as well as emissions from the ion source, may be present, along with emissions due to recombination. In order to differentiate recombination emissions and emissions due to these other processes, two spectra are taken. The first is the spectrum resulting from the sequence of reactions

† See Chapter 1 for a discussion of the stability of atomic ions in the presence of electrons.

as described in this section. The second spectrum is taken after a small amount of a gas that rapidly attaches electrons, an *attaching gas*, is added upstream of the reaction region (at RP2, 27.62 cm upstream of RP3). Addition of this gas removes free electrons from the flow, making electron–ion recombination no longer possible. The amount of attaching gas required is not sufficient to significantly affect the ion chemistry in the flow. At the high pressures used (1.8 Torr), diffusion is minimal so effects on this due to the switch from electrons to negative ions will not be significant. Comparisons of the spectra taken before and after the addition of the attaching gas help to differentiate the various classes of emissions. A brief description of these spectral comparisons is in Section 2-2.2, but a more detailed analysis of the technique can be found in Chapter 3. Specific examples can be found in Chapter 4.

2-2.1 Sequence of Reactions Resulting in Controlled Recombination

As was stated earlier, the primary challenge with these experiments is to somehow cause a large number of electrons and positive, polyatomic ions to coexist in the same small region so that their reactions can be observed. The next several paragraphs describe the sequence of reactions used for that purpose in the experiments presented here.

At the very upstream end of the flowtube, neutral helium is injected through RP1[†] into a region within the microwave discharge source. Inside the self–sustaining discharge source, free electrons (Section 2-1.4) ionize the neutral helium:

$$He + e^{-*} \longrightarrow He^{+} + 2e^{-} \qquad 19$$

Since the pressure of the helium is high ($\sim$ 1.8 Torr), the He^{+} undergoes a three–body reaction very rapidly [Ikezoe 1987] to produce He_2^{+}.

$$He^{+} + 2He \xrightarrow{k\sim 6\times 10^{-32} cm^6 s^{-1}} He_2^{+} + He^{*} \qquad 20$$

[†] See Figures 2-1.1 and 2-1.5 for locations of the reactant ports (RP's) and for diagrams of the flowtube.

(*) represents translational excitation of the third helium atom, providing the necessary stabilization of the helium dimer ion.

Argon injected into the flow at RP2 then reacts with the helium dimer ion to produce Ar^+ [Ikezoe 1987].

$$He_2^+ + Ar \xrightarrow{k\sim 2\times 10^{-10} cm^3 s^{-1}} Ar^+ + 2He \qquad 21$$

The argon will also undergo Penning ionization [Bolden 1970, Schmeltekopf 1970, Lindinger 1974] with any helium metastable species that were created in the discharge:

$$He(2s^1S,\, 2s^3S) + Ar \xrightarrow{k\sim 1\times 10^{-11} cm^3 s^{-1}} Ar^+ + He + e^{-*} \qquad 22$$

The * indicates translational excitation in the electron. This creates a condition where the primary reactive constituents of the flow are the Ar^+ and electrons. Gas–phase reaction between a positive atomic ion and an electron is extremely slow (see Chapter 1), so this plasma is quite stable.

While the He_2^+ is also quite stable and could be used as the charge carrier for the system, Ar has a much lower ionization energy (15.76 rather than 22.23 eV for He_2) [Radzig 1985 for all ionization energies this paragraph]. At 22.23 eV, the dihelium ion would transfer enough energy to not only ionize the polyatomic molecule, but also to fragment it. Most of the polyatomics of interest to these studies have ionization energies around 9-13 eV. Charge transfer from argon is far less destructive. Use of a heavier noble gas, such as krypton (IE = 14.00 eV) or xenon (12.13 eV) would cause even less excitation in the newly ionized polyatomic. This is a possible variation to try in future studies.

In cases where a protonated recombining ion is desired, hydrogen is injected at RP3. The following series of reactions occurs [Ikezoe 1987], resulting in the production of H_3^+, which will readily proton transfer to most neutral atoms and molecules:

$$Ar^+ + H_2 \xrightarrow{k\sim 7\times 10^{-10} cm^3 s^{-1}} ArH^+ + H \qquad 23$$

Table 2-2.1

Typical Experimental Conditions

Helium Throughput	*16.0 slm*	*Argon Pressure* . . .	*∼1 mTorr*
Hydrogen Throughput .	*∼1–4 mTorr*	*Flowtube Pressure* . .	*1.8 Torr*
XYZ Throughput . . .	*∼15* $\frac{mTorr \cdot L}{s}$	*Ionic Species Velocity*	*4500 cm/s*
Attaching Gas Throughput	*∼0.4* $\frac{mTorr \cdot L}{s}$	*Neutral Species Velocity*	*3000 cm/s*

$$ArH^+ + H_2 \xrightarrow{k\sim 2\times 10^{-9} cm^3 s^{-1}} H_3^+ + Ar \qquad 24$$

A small amount of H_2^+ is also produced by Reaction 8, but that also reacts with H_2 to produce H_3^+ [Ikezoe 1987].

Either alone or while the hydrogen is being added, a neutral molecule, XYZ, is injected (also at RP3). In cases where hydrogen has been added, both of the following reactions occur, with the second reaction dominating. In cases where hydrogen is not added, only the first reaction occurs [both reactions and coefficients: Ikezoe 1987].

$$Ar^+ + XYZ \xrightarrow{k\sim (5\times 10^{-10}) cm^3 s^{-1}} XYZ^+ + Ar \qquad 25$$

$$H_3^+ + XYZ \xrightarrow{k\sim (1\times 10^{-9}) cm^3 s^{-1}} XYZH^+ + H_2 \qquad 26$$

The newly created ion then recombines readily with the ambient electrons. A few possible outcomes of a generic recombination are given in Reactions 2 and 3. Although one might initially consider it to be possible, the following reaction is highly inefficient:

$$XYZ^+ + e^- \xrightarrow{very\ inefficient} XYZ + h\nu \qquad 27$$

The reasons for this are discussed in Chapter 1.

2-2.2 Distinguishing Emissions Due to Recombination from Other Emissions

In order to distinguish radiations due to e–i recombination from radiations due to other side processes, as mentioned above, a small amount of a gas that rapidly attaches electrons is added at RP2. Often, as illustrated below, the attaching gas (ABC) will dissociate in some manner upon attaching an electron.

$$ABC + e^- \rightarrow AB^- + C \quad 28$$

$$\rightarrow AB + C^- \quad 29$$

Regardless of the attachment process, the addition of the gas removes electrons from the flow, thus preventing electron–ion recombination.

Attaching gases are chosen for the speed with which they attach electrons and for the small energy changes associated with that attachment. It is also necessary that the attaching gas and the products of its attachment to an electron be unreactive with other species in the flow. Only very small amounts of the attaching gas are required, so the side–reactions are minimized. The three attaching gases used in this study are SF_6, CH_3I and CCl_4 which yield SF_6^- (with some SF_5^-)m I^- and CL^-, respectively.

Spectra are taken before and after the addition of this attaching gas. A spectrum taken without an attaching gas present is referred to an *origin* spectrum (since this spectrum is the one to which others are compared), and the other is called a *background* spectrum. The two spectra are compared by ratio and by difference. As is implied by the names, a difference plot is make by plotting the difference between the origin and background spectra, and the ratio plot, by plotting their ratio. Each type of comparison has advantages and disadvantages. They are discussed in detail in Chapter 3 and examples are presented in Chapter 4.

CHAPTER 3

METHODS OF DATA ANALYSIS

Methods of data analysis are presented. The order of presentation follows the order in which the methods were typically applied to the data.

3-1 Correction of Data for Optical System Transmission Efficiency

In order to properly compare the intensity of emissions in different spectral regions, a mathematical description of the transmission efficiency of the optical path from photoemission through detection was constructed. This efficiency takes account of all devices in the optical train: windows, mirrors, gratings, lenses, photomultiplier tubes and, where appropriate, air. The efficiencies of individual optical components were from factory issued response curves and taken at intervals of 20 nm. The absorption due to air was taken from a geophysical reference and made little difference to the final efficiency. The transmission efficiencies of the individual components, expressed as a fraction of one, were multiplied together at each wavelength to give the overall system efficiency.

The entire collection of data points versus wavelength was not well described by a polynomial; so, the spectral range was broken into smaller segments. Each of these segments was fit to a polynomial of order between 2 and 10 (see Appendix E for the exact form of the polynomials). Photon counts were then corrected for transmission efficiency by dividing the observed numbers of counts by the value of the polynomial at each wavelength. In many cases, the thermal background noise

Figure 3-1.1

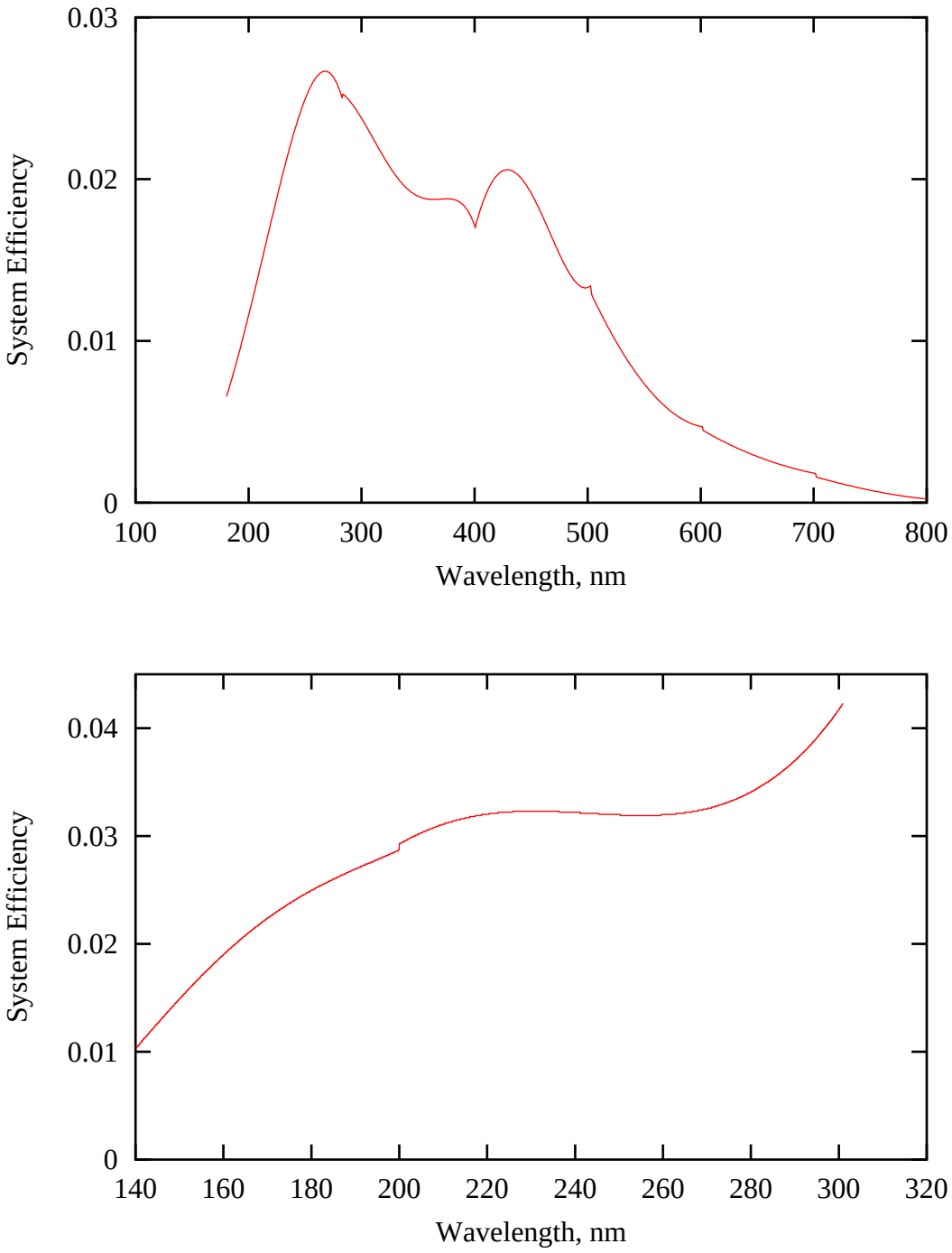


Figure 3-1.1: Optical transmission efficiencies for the UV-VIS (upper graph) and VUV (lower graph) spectrometry systems.

from the photomultiplier tube (PMT) became significant. Whenever this occurred, it was necessary to subtract out as much of the noise as possible prior to correction.

The transmission efficiency could not properly be corrected between 400-415 nm because of the addition of a cut–off filter for which the efficiency is unreliable in that region. The filter is inserted into the optical train when taking spectra above 400 nm to block second order dispersions from the grating.

In all, this method of correcting the spectra for transmission efficiency has performed well. The simulation of the argon emissions in the base plasma (see Section 4-2) indicates a shift in the true efficiency that is not accounted for by the transmission efficiency polynomials. The shift occurs at about 440 nm. Efficiencies from that point down to about 415 nm appear to underestimate the corrected number of photon counts by about a factor of two. Efficiencies within that region appear to be correct relative to each other, and efficiencies above that region also appear to be correct relative to each other (within 5-10% in both cases). The cause of this discrepancy is yet unknown, but could be related to the cut–off filter. This finding does not impact the data presented here; there are no emissions in this region that are significant to the results. No systems have been found that have allowed such rigorous testing of the UV–VIS efficiency below 400 nm or in the VUV. But, there have been many opportunities for comparisons of the relative intensities of a series of vibrational transitions from a single upper vibrational level, and no significant discrepancies have been found.

3-2 Identification of Photoemissions

The identification of photoemissions in the data is not a straightforward procedure. For diatomic molecules, the task has become fairly automated. For atomic species, because of the great number of possibilities, the process is more tedious, but

is possible to make identifications and the results are reliable. For polyatomics, unless a published spectrum is available, identification is difficult. A list of transition wavelengths can also be useful, but precise relative intensities and other information related to band structure are often not included.

Intense or well–resolved atomic species can generally be identified by simple comparison of observed wavelengths to those tabulated in the literature. There are usually a number of intense emissions from the same atom within range of our spectrometers that can be used to confirm identification. If a single upper atomic state decays to a series of lower states, as is the case with the Ar emissions described in Section 4-2, or if a number of intense emissions from a given atom are present at proper relative intensities, identification is certain. Collections of atomic line positions, often with transition probabilities, can be found in a number of locations [for example: Bashkin 1975, NIST 2001b, Radzig 1985].

The identification of diatomic emissions was made reliable by a program for calculating Franck–Condon factors [Ervin 1993]. The program depends on a Morse potential for its calculations. While a Morse potential is not a perfect model for diatomic motion, it is very close and greatly facilitates identification. A program that automates input and output for Ervin's program was also developed. Collectively, the process is called "CASI," an acronym for computer assisted spectral identification. CASI provides a graphical representation of the spectrum to expect from vibrational transitions between two electronic states in a diatomic molecule. The graph consists of a series of vertical lines at expected transition wavelengths. Each line's height is scaled to the intensity expected for that vibronic transition. Across the top of the graph, the vibrational quantum numbers for each transition are indicated. A sample graph from the most recent manifestation of CASI is presented in Figure 3-2.1a. The graph simulates transition wavelengths and intensities for the first five vibrational levels of the CO $a^3\Pi_r \rightarrow X^1\Sigma_g^+$ transition. For comparison, an observed CO $a \rightarrow X$

Figure 3-2.1a & b

Figure captions for Figures 3-2.1 a and b

Figure 3-2.1a: Simulation of vibronic transition positions and intensities as output from the graphical portion of the set of computer programs called CASI (see text). The simulation presented here is of the CO $a \rightarrow X$ emission spectrum expected from the lowest five vibrational levels of the a state. All upper-state vibrational levels are assigned equal populations.

The vertical lines in the lower portion of the graph correspond to the band origin positions for each v→v transition included in the simulation. The intensities of the lines are "pseudo-Einstein A coefficients," P_T (see *Relative vibrational populations* in Section 3-3).

Symbols in the upper portion of the graph identify the upper and lower vibrational quantum numbers corresponding to the line directly beneath. Since vibrational transitions are often clustered, the symbols in the upper graph are color-coded to the vertical lines in the lower graph. The result is not visually appealing, but is easier to read than a monochromatic system. Closed symbols and crossed ×'s correspond to the vibrational quantum number in the upper electronic state; open symbols and uncrossed ×'s, similarly, to the lower state. The value of the vibrational quantum number is read from the axis at the right of the graph.

Figure 3-2.1b: An experimental spectrum taken of the plasma containing recombining CO_2^+ ions for comparison with the graphical output from CASI (Figure 3-2.1a). Identifications are omitted to illustrate the usefulness of spectral simulation in identification. For this spectrum with identifications, see Chapters 4 and 5.

Figure 3-2.1a

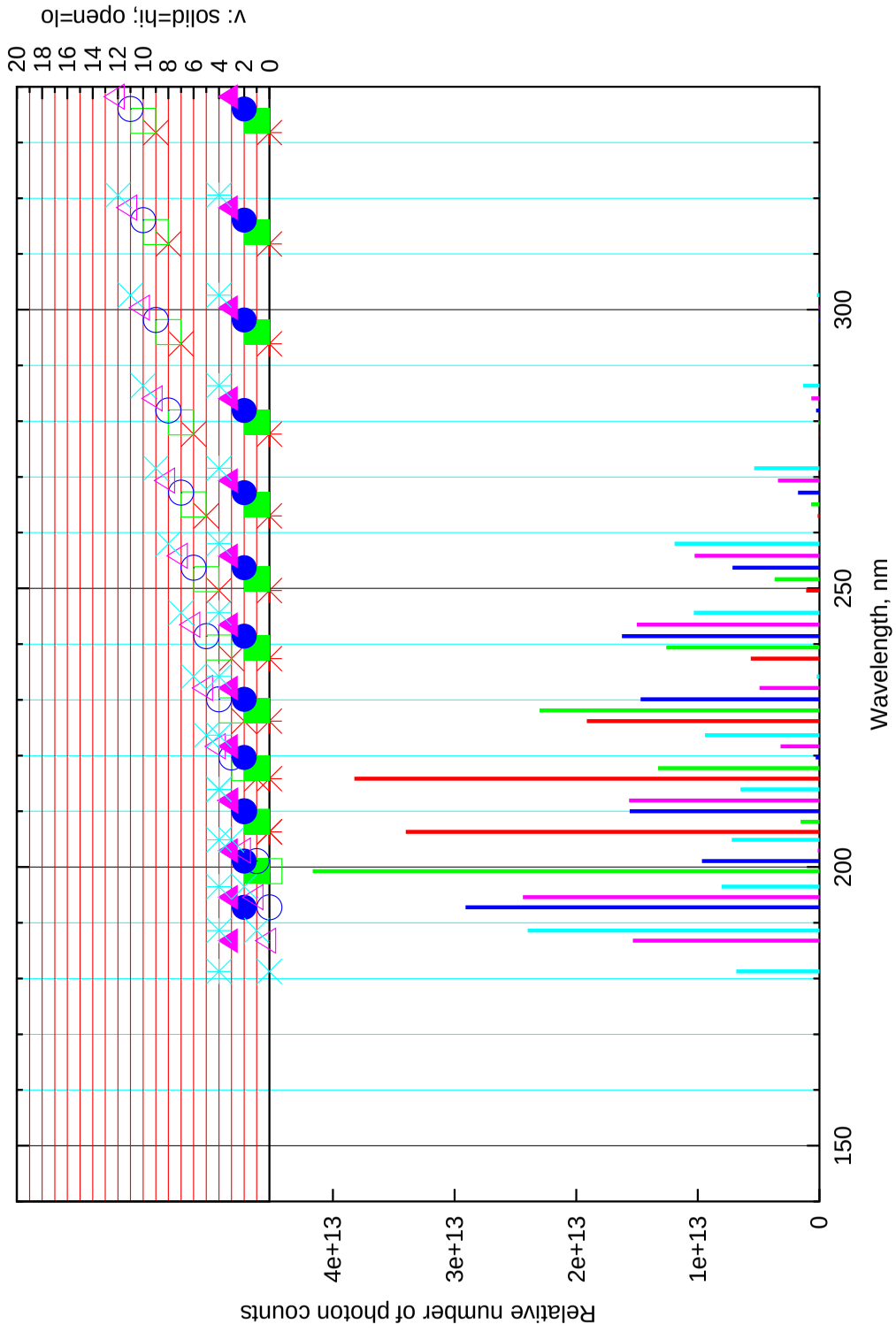


Figure 3-2.1a: See caption on previous page.

Figure 3-2.1b

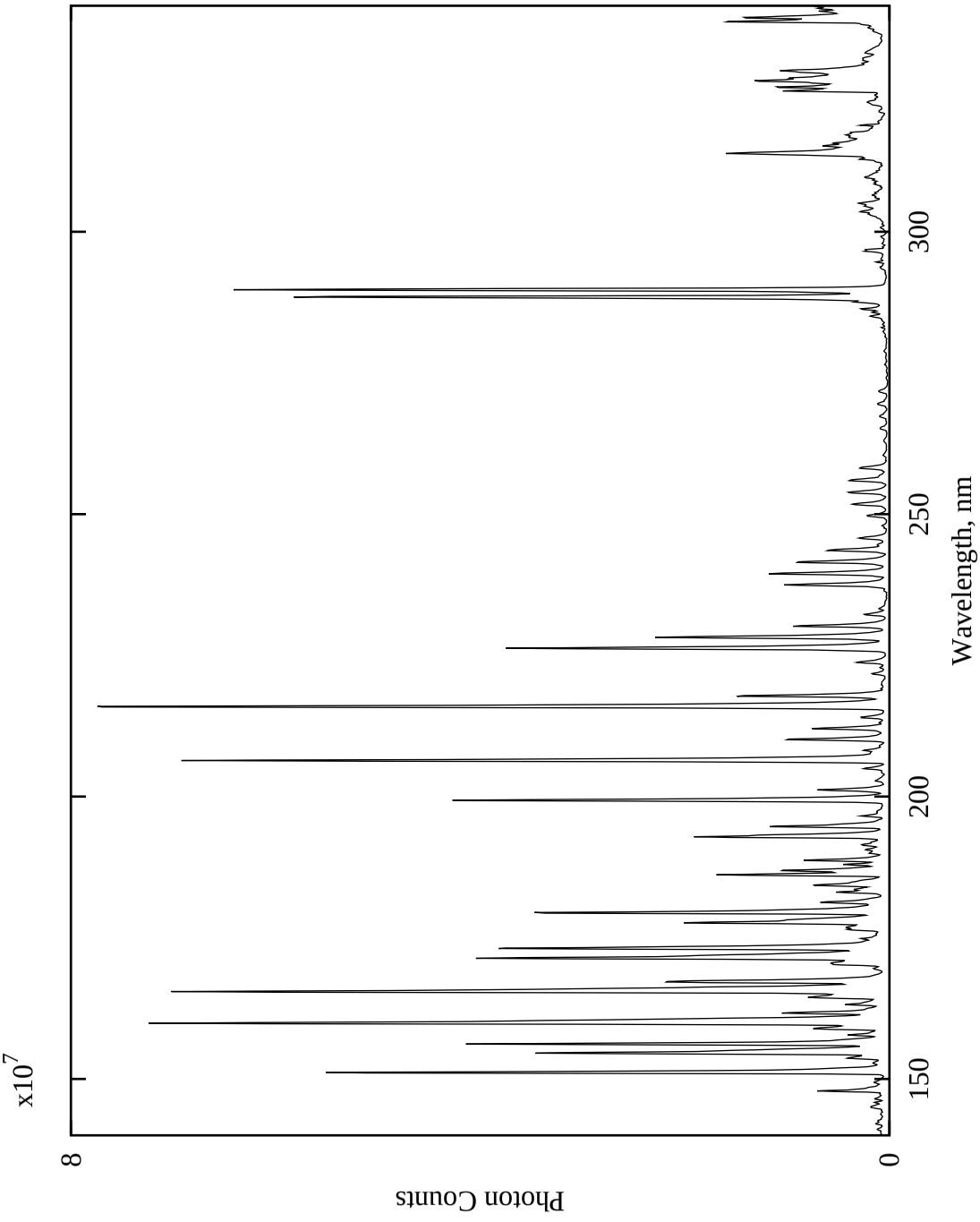


Figure 3-2.1b: See caption before Figure 3-2.1a.

spectrum in the same region is given in part b of the same figure. The usefulness of the program is self–evident. Note that the program does not assign relative populations to the vibrational states, so it does not automatically reproduce all of the intensities in the observed spectrum.

While some spectroscopic constants exist for polyatomic molecules, only the fundamental frequencies are generally reported. Anharmonicities must be inferred, and software for generating Franck–Condon factors for polyatomics is not available. So, unless rotational resolution is available (which is not always the case), identification of polyatomic emissions must proceed via less well defined methods. Typically, published spectra at similar resolution are consulted. Because each spectrum has individual characteristics, identification is quite certain. But, if a published spectrum is not available, identifications are made by estimates based on energetic considerations. Even when the published spectrum is available, care must be taken. Emissions from different species can have similar appearance. For a diatomic example of this, see the discussion on the recombination of H_3S^+ (Section 4-1.8).

In all of the cases, energetics and predictions based on species known to be present in the flow can only be considered guiding principles. On many occasions, a species thought too energetic or otherwise unlikely to be present in the flow has been identified. Two obvious examples are the identifications of the excited states of Ar^+ and He_2. Hints for the presence of He_2 came from a tabulated list of species and corresponding wavelengths [Pearse 1976], but the identification of Ar^+ excited states as a possible emission source came only from extensive and thorough analysis.

3-3 Assignment of Vibrational Populations and Rotational Temperatures

Given the tools and the information available, it has been possible to assign relative populations for atomic and diatomic species for which there are reliable, tabulated spectroscopic constants. Little is generally known about the spectra of polyatomics;

so, many of them could not be treated in detail. This is of little importance here since the photoproducts of dissociative recombination that are identified, either tentatively or with certainty, all result from transitions in diatomic or atomic species. Assignment of the relative populations of electronic states (whether atomic, diatomic or polyatomic) is discussed at the start of Chapter 5, where such discussion is more convenient.

Relative vibrational populations

Within any electronic state of a diatomic molecule, there exists a number of vibrational and rotational states. The relative populations of vibrational states can be determined by comparing intensities of their transitions to transition probabilities. For v→v transitions between two electronic states in a molecule, the probability, P_T, that a transition will occur is proportional to the square of the Franck–Condon factor, $F_{i,j}$ multiplied by the cube of the transition energy, ν:

$$P_T \propto \nu^3 F_{i,j}^2 \qquad 30$$

where i and j represent the vibrational quantum numbers for the upper and lower electronic states, respectively. ν is typically expressed as a transition frequency in wavenumbers. For calorimetric detection methods, the observed intensity, which is not the same as the transition probability, depends on ν^4. These experiments employ photon counting rather than calorimetric detection, so the cubed factor is appropriate. The term used for P_T is "pseudo–Einstein A coefficient" because it contains all transition probability information that is not constant for transitions between different vibrational levels in two electronic states.

The relative population for a series of vibrational states, v, within the same electronic state can be determined by dividing the observed emission intensity, I_{obs} by the appropriate P_T:

$$Relative\ Vibrational\ Population = I_{obs,v}/P_{T,v} \qquad 31$$

Populations determined in this manner are only as good as the Franck–Condon factors (FCF's) used to calculate them. In general, as is expected, FCF's based on a Morse potential fit observed data better near the bottom of the potential well, e. g., for low vibrational quantum numbers, but become progressively less accurate for higher vibrational excitation. Despite this restriction, the relative vibrational populations presented in Chapter 4 are reliable. Rotational characteristics are usually similar for different vibrational levels in the same electronic state; so, changing the rotational temperature has little effect on calculated relative vibrational populations. The time the species spends in viewing range of the spectrometer does not introduce uncertainty either (see Chapter 5) since all the vibrational levels are decaying from a single upper electronic state.

Rotational temperatures

Rotational temperatures were determined by fitting results from the program RVESIM (rotational vibrational electronic simulation, see Chapter 6) to the observed rotational emission manifold. RVESIM, at this time, treats only Hund's Cases a and b and does not treat transitions between the two cases. The rules for transition frequencies and intensities are taken from Herzberg [1989]. For the results presented here, all rotational populations were assumed to be thermal. This is a reasonable assumption since the frequent collisions with He ($10^7 s^{-1}$) are efficient at rotational relaxation. For two of the emitting species, NH and OH, the rotational distribution might not be thermal; see Chapters 4 and 6 for further details. Again, the assigned rotational temperatures can only be as good as the spectroscopic constants used to determine them. In nearly all cases, spectroscopic constants were taken from NIST [2001a]. Where necessary, supplemental values were obtained from another source [Radzig 1985].

Figure 3-3.1a

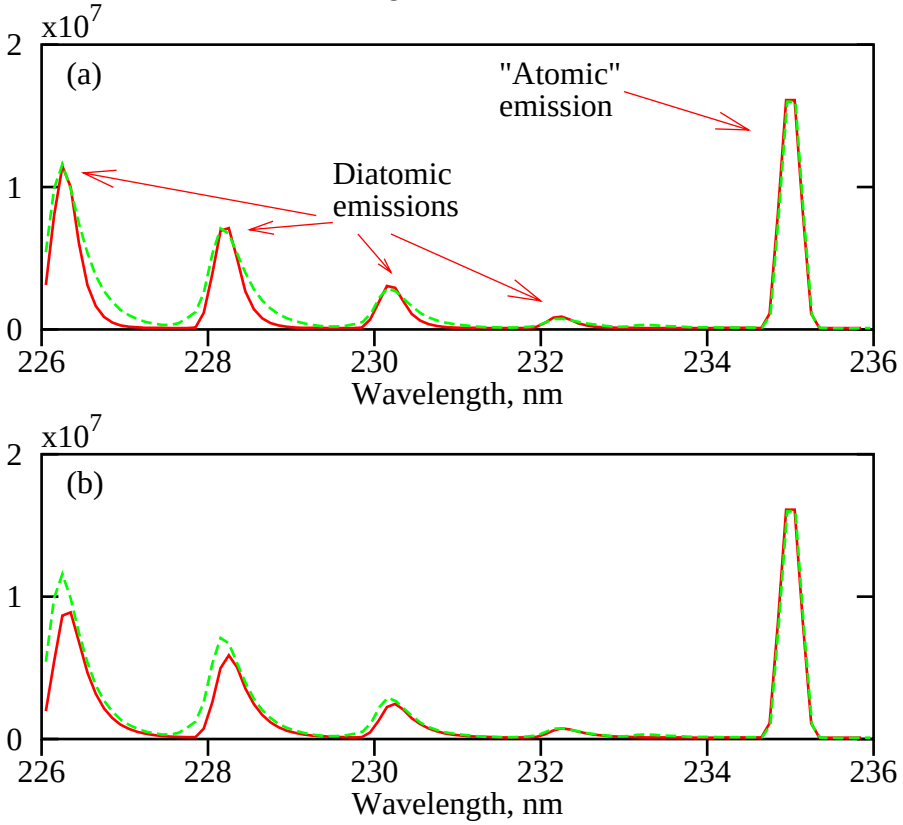


Figure 3-3.1a: Illustration of the effect of assumed rotational temperatures on electronic state populations determined by fitting RVESIM simulations to the rotational transition manifold. Solid, red lines are results from the simulation. Green, dashed lines are experimental data (CO_2^+ recombination) except for the "atomic" peak, which has been included for illustration. Each diatomic peak is a single vibronic transition of CO $a \rightarrow X$. The width of each diatomic peak is due to the rotational temperature and spectral resolution of the photon detection system. The width of the atomic peak is due only to spectral resolution. (a) A rotational temperature of 200 K is too low to reproduce the molecular features, but the maximum intensities of the features are correct. (b) When the temperature is taken as 400 K, the peak shapes mimic experimental data better, but the maxima of the molecular features change relative to the atomic peak.

Figure 3-3.1b

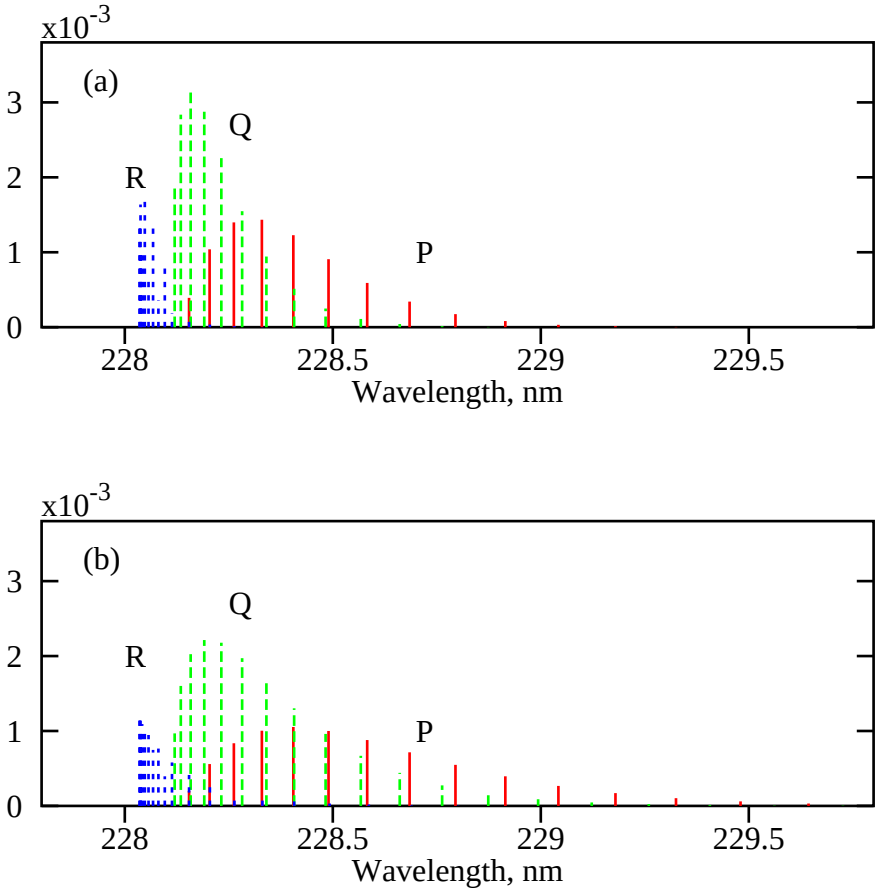


Figure 3-3.1b: Illustration of the effect of assumed rotational temperatures on rotational transition intensities. Each vertical line is a rotational transition within the simulation of the peak near 228 nm in the graphs in Figure 3-3.1a. P, Q and R branches are indicated on each graph. (a) When a temperature of 200 K is assumed, fewer upper-state rotational levels are significantly populated, and the simulated peak is narrower and taller. (b) When 400 K is the assumed temperature, a greater number of upper-state rotational levels have significant populations, resulting in a less intense peak maximum.

It is important to note that the assigned rotational temperature will affect the reported relative electronic population. This is illustrated in Figure 3-3.1. When the temperature of a species is increased, the number of rotational sub–states within each vibronic state is increased. The number of total species in the electronic state must be divided among the number of sub–states populated. In order for a feature to retain a constant relative intensity when the temperature is increased, the occupation of the electronic state must be increased. For this reason, it is important to note the temperature at which relative electronic populations were determined. If the relative temperature determined for a state is taken to be too high a value, the relative population of the electronic state will have been overestimated, too. The intensity change between Figure 3-1.1a (a) and (b) might not appear to be significant on the graph, but the CO a state population relative to that of the "atomic" line must be increased by about 25% to make the maxima match again in (b).

3-4 Plots of Spectral Differences and Ratios

The addition of an electron attaching gas is expected to decrease the intensity of emissions from electron–dependent processes more than other processes. In practice, the situation isn't so straightforward. Such practical issues are presented in detail in Chapter 4 and will be mostly discussed there. Spectra taken without (origin) and with (background) an attaching gas added are compared by taking their difference and ratio. A schematic illustration of such comparisons is presented in Figure 3-4.1. Both the ratio and the difference plot have their usefulness and their limitations. Neither is, by itself, an infallible method for determining which emissions are photoproducts of dissociative recombination (DR).

The advantage to a difference plot is that it retains the original appearance of the emission. For this reason, the difference plot is often invaluable for identifying and quantifying emissions, as it can be used to subtract out emissions from other

Figure 3-4.1

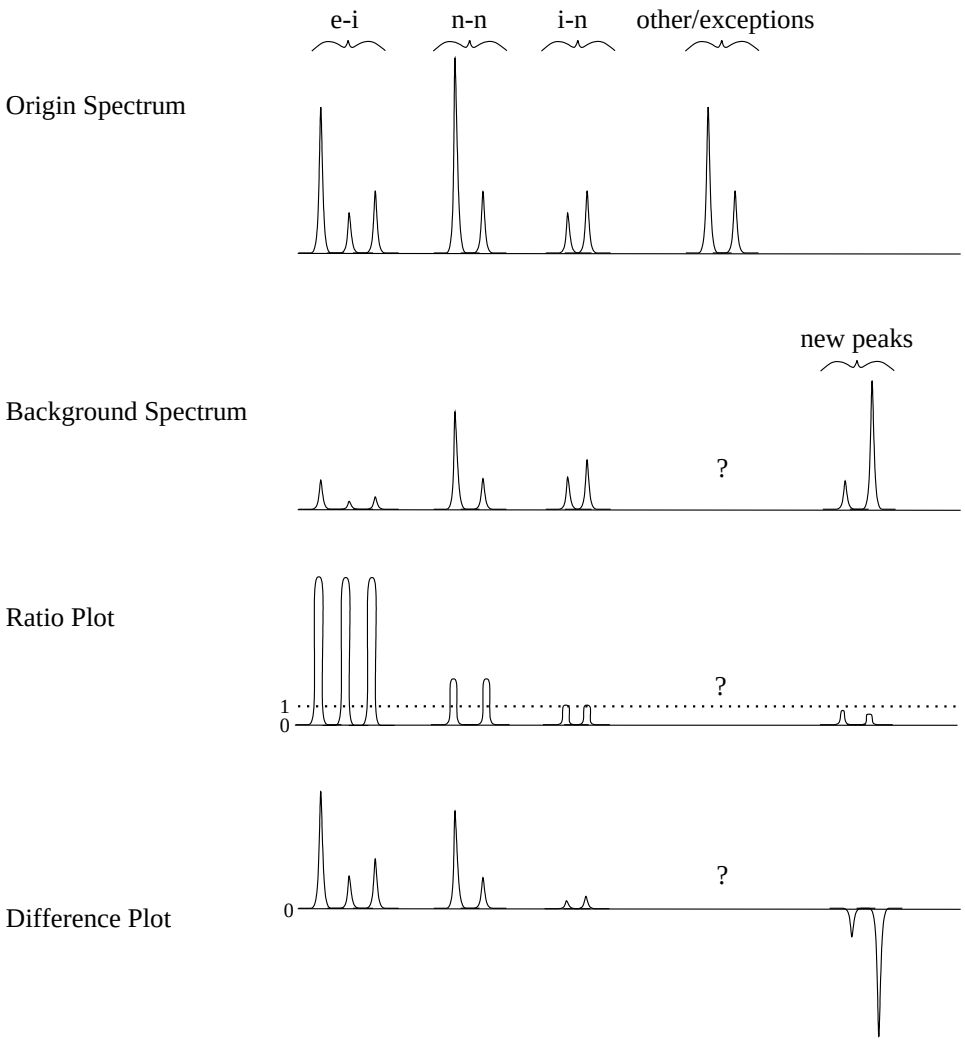


Figure 3-4.1: Schematic illustration of ratio and difference plots when applied to emissions arising from different sources. *e-i:* electron-ion recombination; *n-n:* neutral-neutral reactions; *i-n:* ion-neutral reactions.

processes. There are two difficulties with the difference plot. The first is that if there are emissions resulting from DR that are of low intensity, they can be easily missed in a difference plot. One of the original reasons for creating a ratio plot was to address this issue (see later). The other problem is that some emissions do not behave as expected and will be quenched even though they cannot be products of

recombination. An example of this is illustrated schematically in Figure 3-4.1; see the emissions categorized as arising from a neutral–neutral reaction ("n–n"). The emission in the origin spectrum is so intense that, although much of it was quenched, it remains a prominent feature in the difference plot. Other examples are given in Chapter 4.

Ratio plots are useful for detecting DR photoproducts at low intensity. In the ideal case, the ratio for a set of emissions resulting from the same process is the same regardless of the initial intensity of the peak (see Figure 3-4.1). In reality, the presence of system noise and the limit on the number of counts to integers complicates the use of the ratio plot at low signal levels.

The ratio, r, at each point includes the native origin counts, o, the native background counts, b, and the average level of noise $n \pm \delta n$:

$$r = \frac{o + n \pm \delta n}{b + n \pm \delta n} \qquad 32$$

The desired ratio is a "true ratio," R which does not include noise:

$$R = o/b \qquad 33$$

If o and b are both large compared to $n \pm \delta n$, then $r \approx R$. But the emissions for which the ratio plot is most useful are those that are at low intensities, where o and b are not likely to be large. In the case that they are both small relative to $n \pm \delta n$, then for all points $r \approx 1$. In the case that o and b are both very small, the best solution, of course, is to increase the count period, thereby increasing the signal to noise ratio.

In order to make the ratio plot the most useful in the cases where it is most needed, the effects of noise in the system must be addressed. One way to address noise in the system is to attempt to subtract it out. A number of counts, c is subtracted from both the origin and the background spectra prior to taking the ratio. c must be chosen so that it is the maximum value possible subject to the requirement that $c < n - \delta n$. This is generally effective. If o and b are on the order of δn, then they

Figure 3-4.2

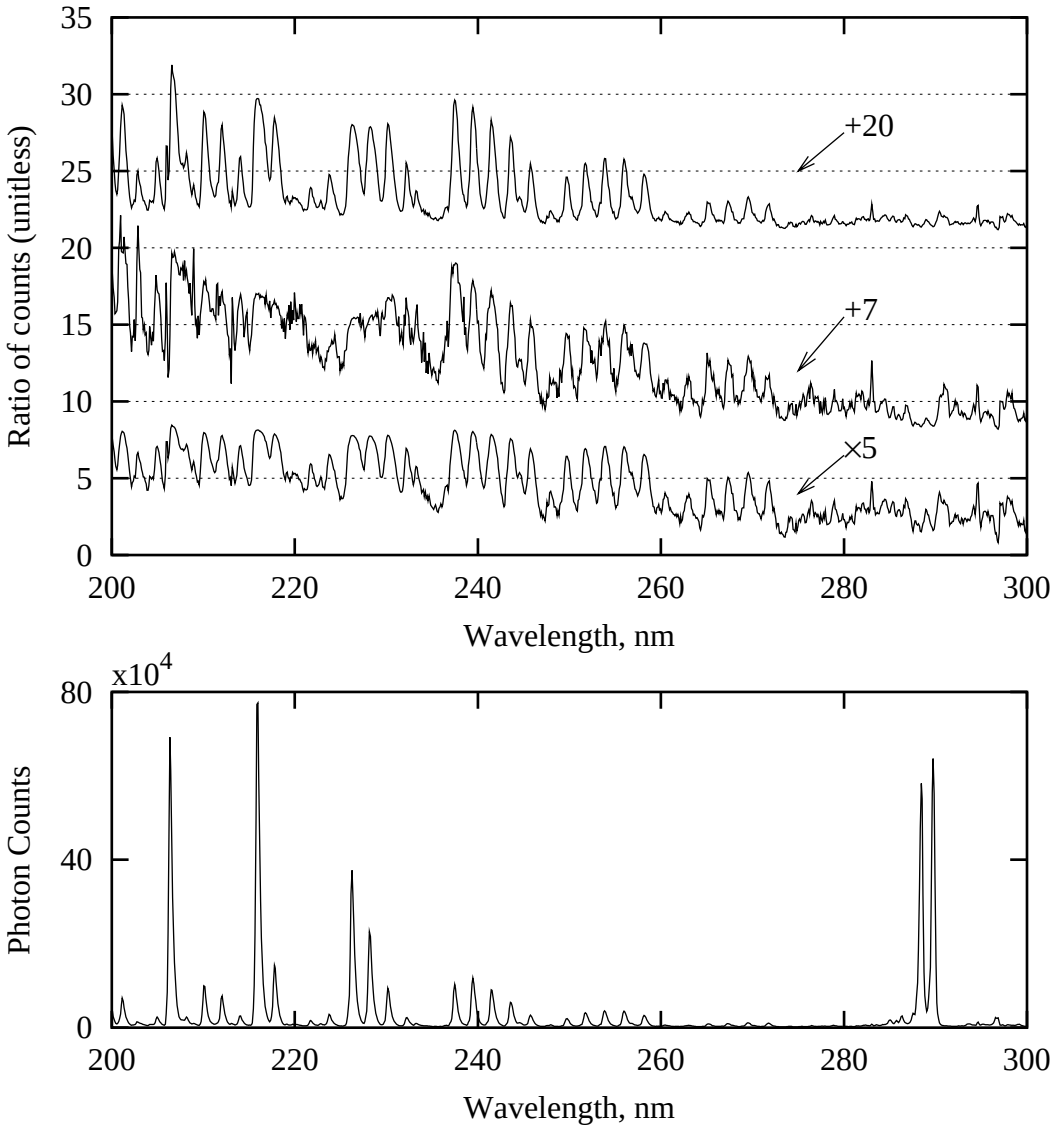


Figure 3-4.2: Comparison of three methods for generating ratio plots (top graph) to the origin spectrum (bottom graph) from the data used to generate them. The ratio methods (see text) are: *upper line:* simple ratio; *middle line:* ratio using subtraction of noise; *lower line:* adjusted relative difference. The top two lines are offset and the adjusted relative difference is multiplied by a factor of five for clarity. In the lower graph, the emissions from 200 to about 280 are CO $a \rightarrow X$ emissions and the double peak near 290 is the CO_2^+ $\tilde{B} \rightarrow \tilde{X}$ emission.

will probably never be detected in the first place. In most cases, they will be more easily detected using this subtraction method.

An alternate method for addressing system noise was suggested by Brian Decker [1993]. He calls this method an "adjusted relative difference," the symbol $d_{a,r}$ will be used here. It is a hybrid of ratio and difference plots:

$$d_{a,r} = \frac{o - -b}{\frac{1}{2}(o+b)+n} \qquad 34$$

It does a far better job at producing rectangular peak shapes than do the plain ratio or the ratio with noise subtracted. But, it is not quite as good as the subtracted method for low initial intensities. A comparison of the three methods can be found in Figure 3-4.2.

3-5 Color Photographs

Color photographs were taken of the reactive flow near RP3 under a variety of conditions in order to examine the mixing of the reactant gas with the oncoming plasma. Measurements were taken from features in the photos. Details are provided in Chapter 5.

3-6 Assessment of Vibrational Relaxation in Recombining Ions

Whenever possible, it is desirable to ensure that the recombining ion is in a vibrationally relaxed state, or, at least, in a known state of vibrational excitation; this is desirable because vibrational excitation complicates analysis, making detailed understanding of the recombination mechanism more difficult. The methods by which the extent of collisional relaxation in the recombining ions was assessed are presented in this section (information regarding collisional relaxation comes from the text by J. D. Lambert [1977]).

Modes of Collisional Relaxation and Their Efficiencies

Collisional vibrational relaxation of molecular species can proceed via three transfer mechanisms: vibration → vibration, vibration → rotation and vibration→ translation. In all cases, transfer is most likely to occur if the amount of energy, ΔE_T, that must be put into translation is small. The probability, P of such a transfer is usually proportional to an exponential in ΔE_T:

$$P \propto \mathrm{e}^{-\Delta E_T} \qquad 35$$

Resonant vibrational relaxation is very fast. For instance, CO_2 will relax itself in fewer than 10 collisions, but N_2O takes about 50 collisions to relax CO_2 (from its lowest asymmetrical stretch mode). Infrared–active modes relax more efficiently than infrared–inactive modes. A vibrational energy difference (between modes on different molecules) of ≤ 50 cm^{-1} is considered resonant. For differences over 200 cm^{-1}, relaxation is effectively vibrational–translational.

For non–resonant (vibrational–translational) relaxation, an empirical relationship [Lambert 1977] is available. It estimates the number of collisions, Z, required to relax a molecule as:

$$Z = \mathrm{e}^{C\nu_{min}} \qquad 36$$

where ν_{min} is the frequency of the lowest energy vibrational mode expressed in cm^{-1}, and C is a constant. For atoms containing one or more hydrogen atoms, $C = 0.0083$. For other molecules, the number is 0.017. The equation is intended for vibrational relaxation from v=1 to v=0 in a pure sample of the gas and was developed for neutral species. But, even with those limitations, its use should provide a rough estimate, which is sufficient for our purposes.

A similar set of rules is not available for vibration–rotation transfer; therefore, only pseudo–quantitative statements can be made. As was stated earlier, the greater the energy that must be transferred to translation, the lower the probability for

transfer. Since vibrational quanta are generally much larger than rotational quanta, this means transfer favors large changes in J (ΔJ) or large values for B (the rotation constant). This effect is countered by the requirement that angular momentum be conserved. So, for small molecules, like N_2 and O_2, the most probable transitions are $\Delta J = \pm 0, 2$ with $\Delta J = \pm 4$ being three orders of magnitude less probable [Lambert 1977, p. 51]. Larger (more massive) molecules are more likely to undergo large changes in J. For I_2 [Lambert 1977], the optimum ΔJ is $\pm 6 - 8$ with a maximum around $\Delta J = 14$. None of the molecules in this study are as massive as I_2, the closest being CS_2^+ (76 amu), which has about 30% the mass of I_2. So, small changes in J should be favored. The rigid–rotor value for the rotational constant of CS_2 is about 0.1 cm^{-1}, and the lowest vibrational frequency for CS_2^+ is around 300 cm^{-1}. Since the CS_2 will, presumably, be acquiring energy from vibration in CS_2^+, it makes sense to consider positive values for ΔJ. Assuming the large value given for ΔJ for I_2 ($\Delta J = +14$), the starting value of J (in the CS_2) would need to be 32. For smaller ΔJ values, the starting J is a larger number; for example, 215 for $\Delta J = +6$. The Boltzmann probability at 300 K for a J value of 215 is about 2×10^{-10}. For the lower value of J the Boltzmann probability is about 0.6, but this value of J will relax far less efficiently. The point is that vibration–rotation transfer will be more efficient than pure vibration–translation transfer, but it is still unlikely to be significantly efficient. Given that, and the lack of a convenient formula for comparison, the empirical formula given in Equation 7 will be used in cases where vibrational resonance is absent.

3-7 Rate Comparisons for Several Relevant Processes

A number of comparisons are made between the rates at which certain processes occur in the plasma. In all cases, the comparison is between two different processes that a single species might undergo. For example, it is useful to estimate the number of collisions that a molecular ion will undergo (with atomic or molecular species)

before it recombines. This estimation allows one to judge the likelihood of vibrational relaxation prior to recombination. The methods used to make this, and other, comparisons are described below; the results of these comparisons are in Table 4-1.7.

For all parent ions, a comparison, $f_{\nu,XYZ}$, is made between its recombination rate, $Rate_{XYZ^+,e^-}$, and the collision rate, $Z_{XYZ^+,XYZ}$, of the ion with its neutral parent, XYZ. A similar comparison, $f_{\nu,He}$, is made between the recombination rate and the collision rate, $Z_{XYZ^+,He}$, of the ion and the bulk neutral helium. The subscript ν is used to indicate that the reason for this comparison is vibrational relaxation of the recombining ion.

For these comparisons, the collision density for a mixture of ideal gases will be used. The collision density, $Z_{A,C}$, between two non–reactive, neutral species A and C is given by [physical chemistry texts, for example, Atkins 1998]:

$$Z_{A,C} = [A][C]\sigma\left(\frac{8k_BT}{\pi\mu}\right)^{1/2} \qquad 37$$

where σ is the hard–sphere collisional cross section, μ is the reduced mass and all other symbols have their usual meanings. A binary rate law for the reaction of A with some other reagent B is:

$$Rate_{A,B} = k[A][B] \qquad 38$$

where k is the rate constant. The rate at which A collides with C can be compared to the rate at which A reacts with B, even if the concentration of A is unknown, by taking the ratio of $Z_{A,C}$ and $Rate_{A,B}$, i. e.:

$$f_{\nu,XYZ} = \frac{Z_{XYZ^+,XYZ}}{Rate_{XYZ^+,e^-}} \qquad 39$$

$$f_{\nu,He} = \frac{Z_{XYZ^+,He}}{Rate_{XYZ^+,e^-}} \qquad 40$$

For protonated parent ions, two other comparisons, f_{R,Ar^+} and f_{R,XYZ^+}, are also made. f_{R,Ar^+} is the ratio of the rate at which Ar^+ will react with H_2 compared to

the rate at which Ar^+ will react with the unprotonated neutral parent. f_{R,XYZ^+} is the ratio of the rate at which XYZ^+ will react with H_2 compared to the rate at which it will recombine.

$$f_{R,Ar^+} = \frac{Rate_{Ar^+,H_2}}{Rate_{Ar^+,XYZ}} \qquad f_{R,XYZ^+} = \frac{Rate_{XYZ^+,H_2}}{Rate_{XYZ^+,e^-}} \tag{41}$$

In all cases, an electron density of $1{\times}10^{10}$ cm^{-3} is assumed. Where a recombination coefficient is not known, $1{\times}10^{-6}$ $cm^3 s^{-1}$ is assumed if the molecule is triatomic or larger. $2{\times}10^{-7}$ $cm^3 s^{-1}$ is assumed for diatomics. Atomic ions not treated in this manner. The rate constant for all reactions between H_2 and XYZ^+ is assumed to be $1{\times}10^{-9}$ $cm^3 s^{-1}$. Other ion–neutral rate constants that are not known are also assigned a value of $1{\times}10^{-9}$ $cm^3 x^{-1}$. The temperature is assigned a value of 300 K.

CHAPTER 4

DATA ANALYSIS AND RESULTS

In this chapter, the results from a number of experiments will be presented. In the first two sections, results pertinent to the main goal of these studies — the observation and characterization of photoemissions from dissociative recombination — are given. In the last section, experimental findings that were not the original focus of the experiments are included. Relevant portions of the spectra from which conclusions were made are included with the discussion of each set of experimental results.

4-1 Spectral Responses to the Addition of an Attaching Gas

In the ideal case, the addition of a small quantity of attaching gas will remove electrons from the flow, thereby quenching emissions that result from recombination, but not others. This assumption appears to be mostly accurate, but results from the experiments presented in this section and in Section 4-3 indicate that the situation is a bit more complex.

Spectra were taken at a single flow of CO_2 (through RP3[†]) and a series of flows of the attaching gas CCl_4 (upstream, through RP2[†]). The spectra, taken at a resolution of about 0.5 nm (full width at half maximum, FWHM), include examples of features that can and cannot arise from recombination of ground–state CO_2^+ ions. Samples of these spectra are presented in Figures 4-1.1 and 4-1.2. Both figures compare the

[†] "RP" refers to "reactant port." See Section 2-1.

Figures 4-1.1 and 4-1.2

Figures 4-1.1 and 4-1.2: Plots of spectral peak intensities in a plasma containing recombining CO_2^+ ions at a series of CCl_4 flows (CCl_4 added at RP2).

Each graph contains a series of lines, each corresponding to a different throughput of CCl_4. Since the intensities of the lines change as a function of wavelength, line styles (solid, dash, etc.) differ to help guide the eye (they do not have additional meanings). Line that have, overall, greater intensity correspond to larger throughputs of CCl_4. CCl_4 throughputs range from 0 to 0.001 Torr*L/s.

Photon counts are not corrected for optical system transmission efficiency. The resolution in these spectra is about 0.5 nm, FWHM.

Figure 4-1.1

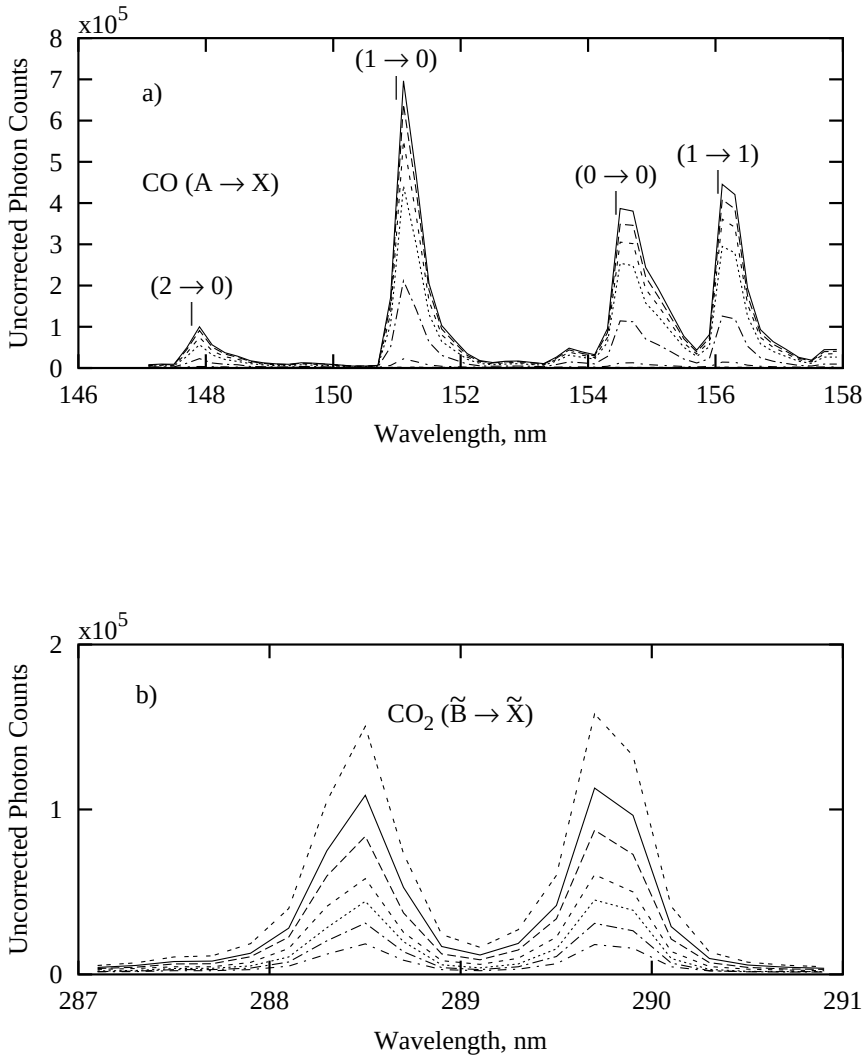


Figure 4-1.1: (a) Emissions from CO are possible products of recombination. (b) Emissions from CO_2^+ are not likely to be products of recombination. The CO_2^+ emissions in the spectra are believed to be due to reactions of CO_2 with He metastable atoms (see text). See also the caption on the previous page.

Figure 4-1.2

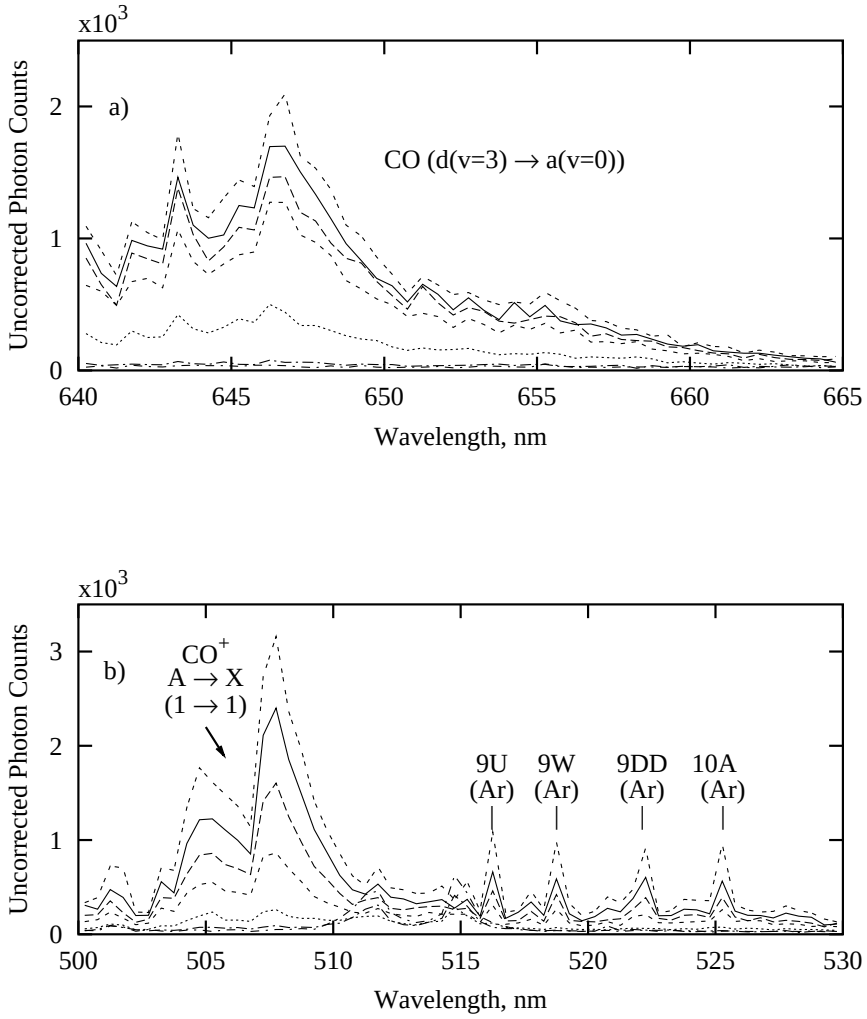


Figure 4-1.2: (a) Emissions from CO are possible products of recombination. (b) Emissions from CO^+ are not likely to be products of recombination. The CO^+ emissions in the spectra are believed to be due to reactions of CO_2 with He^+ atoms or He_2^+ molecules in vibrationally excited states (see text). The production mechanism for the Ar emissions is uncertain (see Section 4-3). See also the caption preceding Figure 4-1.1.

behavior of a set of emissions thought to result from recombination (top graph) to a set of emissions that are not expected to depend on electron density (bottom graph). Note that all features in all four spectra respond to the addition of the attaching gas. There is a difference between their behaviors, though: the intensities of the emissions in the lower graphs decrease more regularly (as graphed here) than the emissions in the upper graphs.

To examine these different responses, emission intensities at selected wavelengths from the spectra in Figures 4-1.1 and 4-1.2 are plotted as a function of CCl_4 throughput on ln–linear (Figure 4-1.3) and on linear–linear scales (Figure 4-1.4). Figures 4-1.5 and 4-1.6 contain similar information, but the data plotted in graph (a) on both figures are not shown in the spectra in Figures 4-1.1 and 4-1.2. Curves superimposed on the data points in all eight graphs were fit by hand. Note the difference between the behavior of emissions that are likely to be products of recombination (CO, Figures 4-1.3 — 4-1.6) and emissions that cannot result from recombination (CO_2^+, Figures 4-1.3 and 4-1.4; CO^+, Figures 4-1.5 and 4-1.6). Note especially that the CO emissions decay linearly as a function of attaching gas flow as is expected from previous analysis of reaction kinetics associated with the addition of the attaching gas to the flow [Foley 1993].

If the CO emissions result primarily from dissociative recombination, as is expected, the x–axis intercept corresponds to an electron density of about 1×10^{11} cm^{-3}. This figure correlates well with the electron density expected for the region around RP2.

The initial decays of the CO_2^+ and CO^+ emissions in Figures 4-1.4 and 4-1.6 are also linear. A linear decay is expected for any species that is destroyed, either directly or indirectly, by the addition of the attaching gas rapidly enough that the amount left at RP3 is determined stoichiometrically. In graphs of emissions from both ions the initial, linear portion of the decay corresponds to a species that responds to the

Figures 4-1.3 — 4-1.6

Figures 4-1.3 — 4-1.6: Comparison of the change in emission intensity for selected emissions as a function of throughput of the attaching gas CCl_4 when an emission is (a) probably due to recombination and (b) probably not due to recombination.

Photon counts have not been corrected for optical system transmission efficiency. Wavelengths chosen (see legends) are the points at which the maximum number of counts were observed for that feature. Throughputs are approximate. To convert the throughput to the number density of the attaching gas once it has fully expanded into the flowtube, multiply the throughput by 2.4×10^{14}.

Numbers in parentheses in the legend for transitions in diatomic molecules (CO and CO^+) are the vibrational quantum numbers for the upper and lower levels ($v' \rightarrow v''$) involved in the transition.

Curves superimposed onto data points are fits by hand.

See also the additional information below.

Figures 4-1.3 and 4-1.5: Semilogarithmic plots of spectral emission intensities versus CCl_4 throughput. Data points were taken up to CCl_4 flows much higher than that typically used during experimentation.

Figures 4-1.4 and 4-1.6: Linear plots of spectral emission intensities versus CCl_4 throughput. These graphs are of the same data as in Figures 4-1.3 and 4-1.5, but on linear scales with an abbreviated range of CCl_4 throughputs. Note especially the linear portions of the curves.

Figure 4-1.3

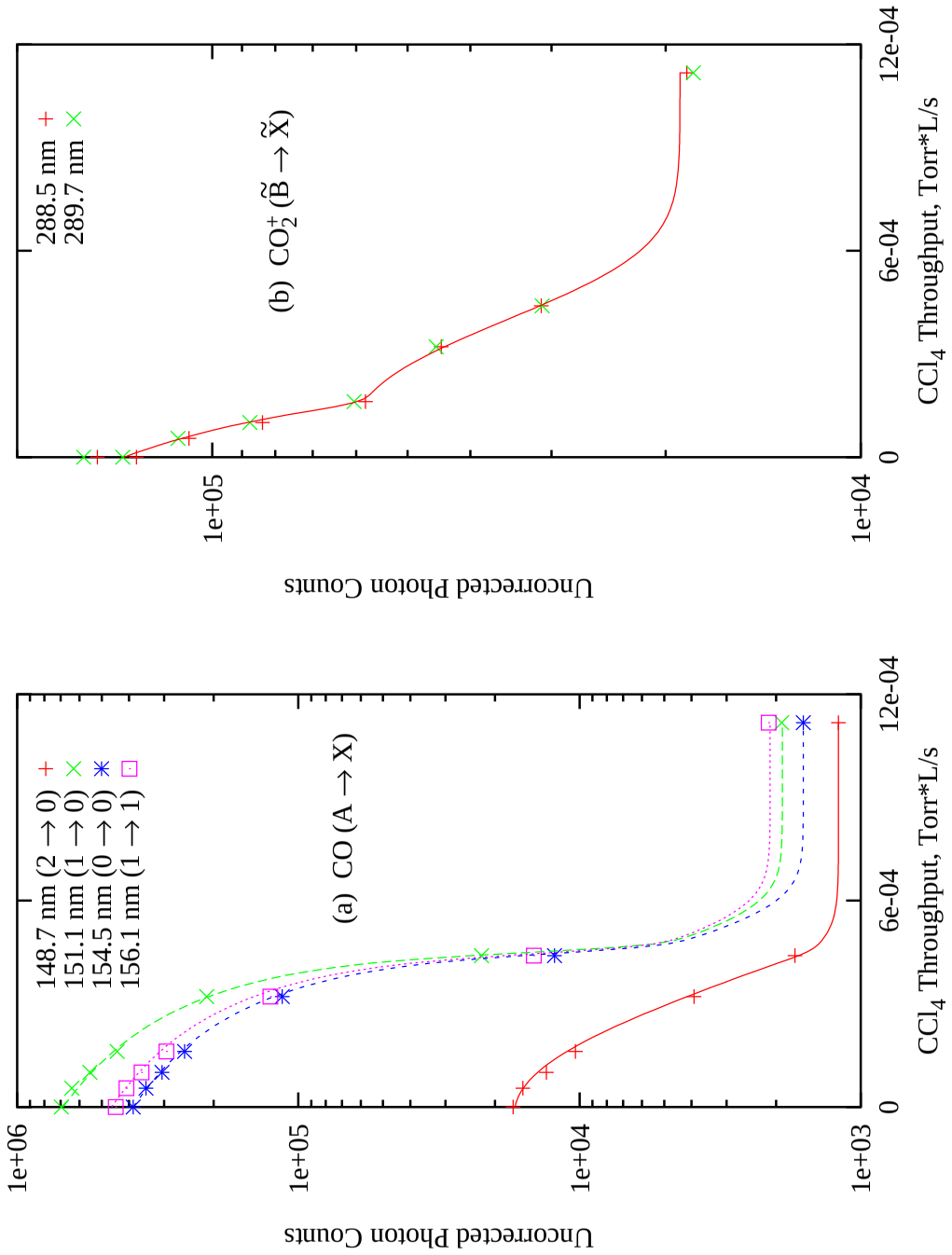


Figure 4-1.3: Selected emissions intensities versus CCl_4 flow. Emissions in (a) are from the CO $A \rightarrow X$ transition and are probably due to recombination. Emissions in (b) are from the CO_2^+ $\tilde{B} \rightarrow \tilde{X}$ transition and are not likely to result from recombination. See also additional caption on previous page.

Figure 4-1.4

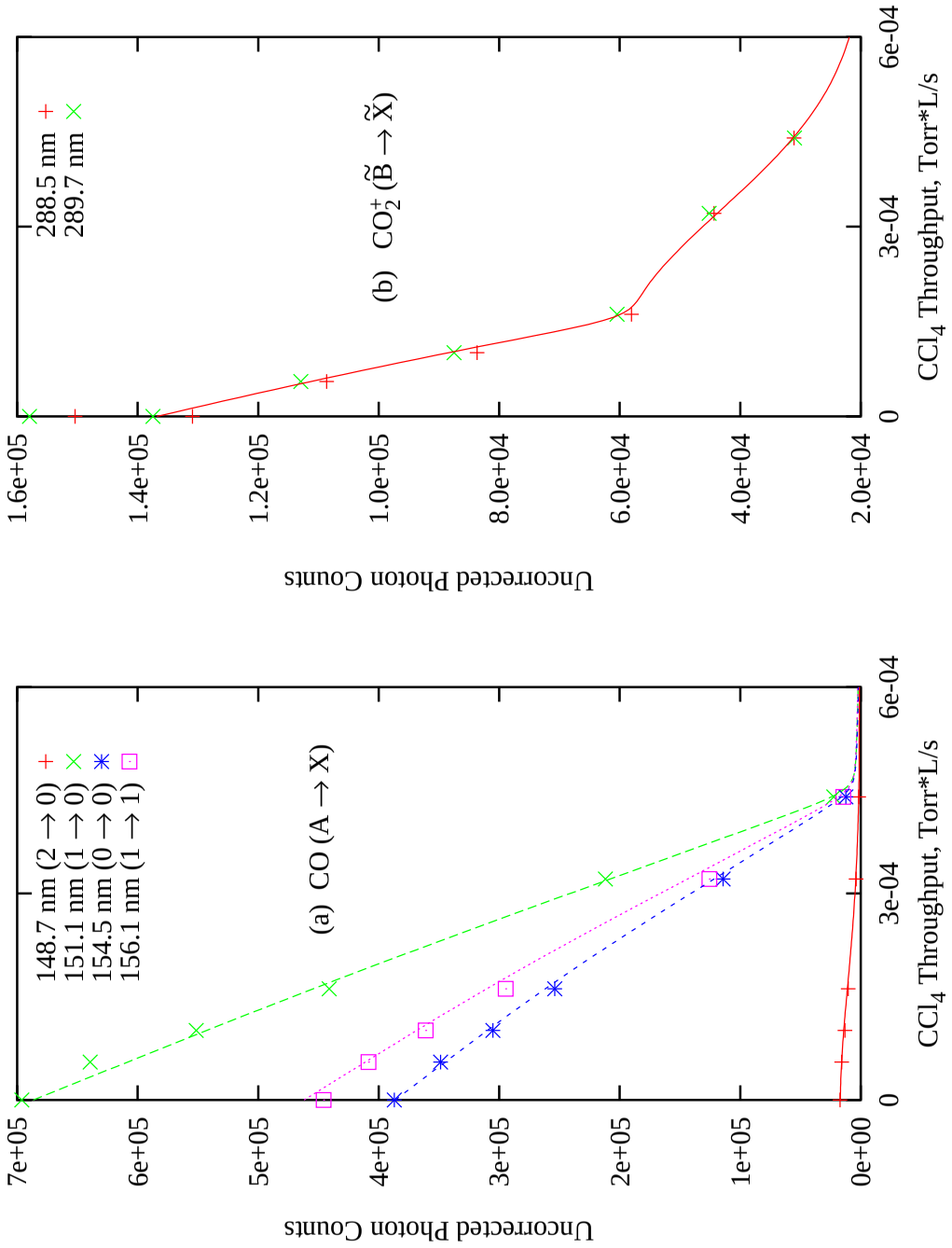


Figure 4-1.4: Selected emissions intensities versus CCl_4 flow. Emissions in (a) are from the CO $A \rightarrow X$ transition and are probably due to recombination. Emissions in (b) are from the CO_2^+ $\tilde{B} \rightarrow \tilde{X}$ transition and are not likely to result from recombination. See also additional caption preceding Figure 4-1.3.

Figure 4-1.5

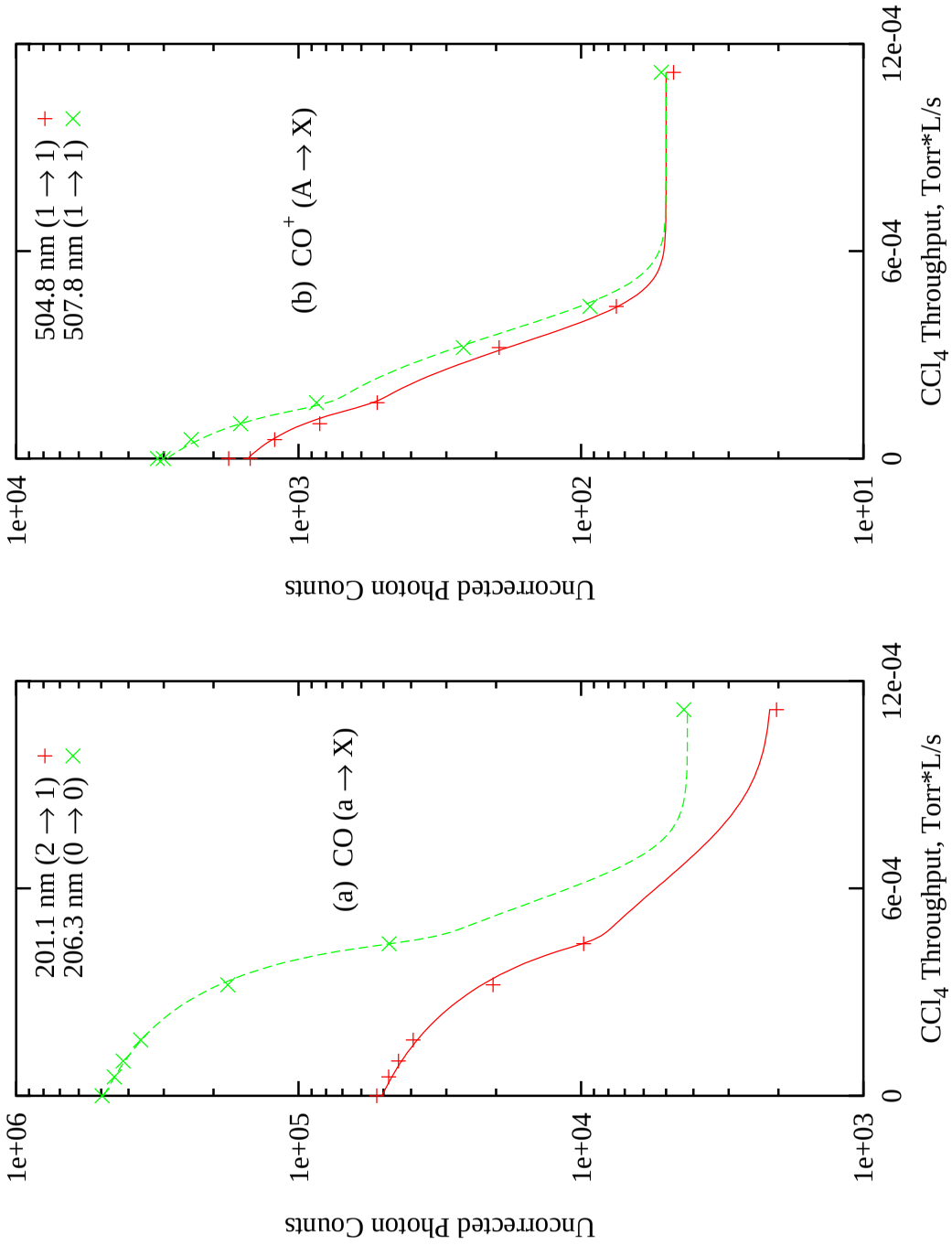


Figure 4-1.5: Selected emissions intensities versus CCl_4 flow. Emissions in (a) are from the CO $a \rightarrow X$ transition and are probably due to recombination. Emissions in (b) are from the CO^+ $A \rightarrow X$ transition and are not likely to result from recombination. See also additional caption preceding Figure 4-1.3.

Figure 4-1.6

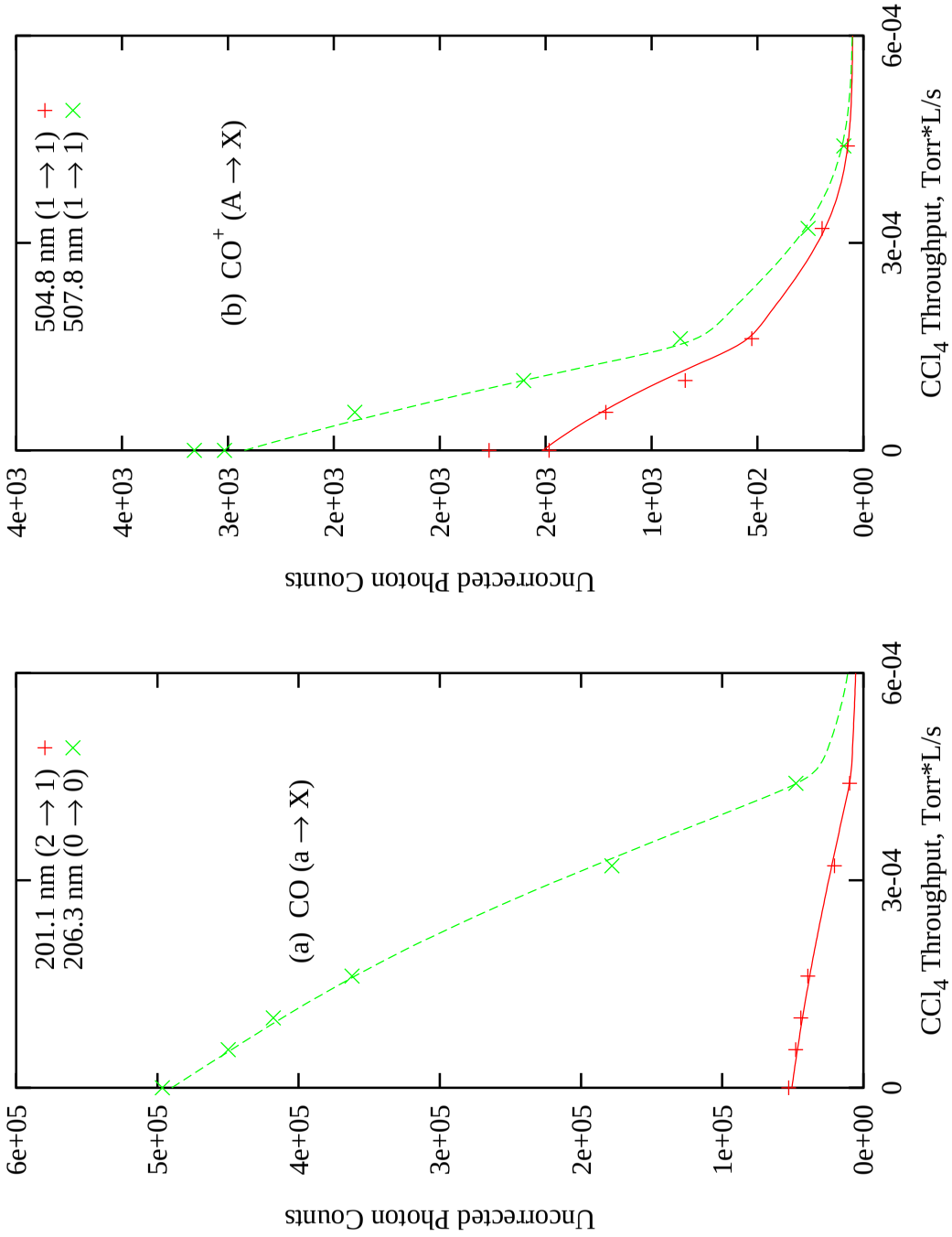


Figure 4-1.6: Selected emissions intensities versus CCl_4 flow. Emissions in (a) are from the CO $a \rightarrow X$ transition and are probably due to recombination. Emissions in (b) are from the CO^+ $A \rightarrow X$ transition and are not likely to result from recombination. See also additional caption preceding Figure 4-1.3.

Table 4-1.1

Energy Changes of Relevant Processes Occurring in the Flow

Process	ΔE (eV)†	Uncertainty	Reference
$H^+ + e- \rightarrow H$	-13.5985	A	Radzig, 1985
$H_3^+ \rightarrow H_2 + H^+$	4.38	$< 10\%$	Radzig, 1985
$He(^1S_0) \rightarrow He$	-20.616	B	NIST, 2001
$He(^3S_1) \rightarrow He$	-19.820	B	NIST, 2001
$He^+ + e- \rightarrow He$	-24.5876	A	Radzig, 1985
$He_2^+ + e- \rightarrow 2He$	-22.22	$< 1\%$	Radzig, 1985
$He_2^+ \rightarrow He + He^+$	2.36	$< 1\%$	Radzig, 1985
$HeH^+ \rightarrow H^+ + He$	1.845	$< 1\%$	Radzig, 1985
$Ar(4s[3/2]_2^0\ ^3P_2^0) \rightarrow Ar$	-11.548	B	NIST, 2001
$Ar(4s'[1/2]_0^0\ ^3P_0^0) \rightarrow Ar$	-11.723	B	NIST, 2001
$Ar^+ + e- \rightarrow Ar$	-15.760	A	Radzig, 1985
$Ar^+(^4P^0,\ ^4D^0,\ ^2D^0) \rightarrow Ar^+$	-19.223 to -19.762	B	NIST, 2001
$ArH^+ \rightarrow Ar + H^+$	3.87	$< 1\%$	Radzig, 1985

Table 4-1.1: Energetics for several processes known to be occurring in the reactive mixture in the flowtube. Many of these occur in conjunction with other processes during reactions; for example, $H_3^+ \rightarrow H_2 + H^+$ does not occur alone, but does occur as a half-reaction in a proton transfer from H_3^+ to a parent gas. Energetics refer to the ground states of all relevant species unless otherwise indicated. A: these numbers are certain to ± 2 in the last significant figure. B: the accuracy of these numbers is limited, to ± 1 in the last digit, by the conversion from wavenumbers (using 1 cm^{-1}=8065.5 eV). † Negative values of ΔE indicate an exothermic process.

Table 4-1.2

Energetics for Processes Relevant to Section 4-1.1

Process	**Products**	ΔE $(eV)^{\dagger}$	**Accessible States**	**Notes**
$CO_2^+ + e^-$	$CO + O$	-8.3	CO: $a^3\Pi_r(11)$, $a'^3\Sigma^+(9)$, $d^3\Delta_i(5)$, $e^3\Sigma^-(2)$, $A^1\Pi(0)$, $I^1\Sigma^-(1)$, $D^1\Delta(0)$	*a*
	$C + O_2$	-2.4	O_2: $a^1\Delta_g(7)$, $b^1\Sigma_g^+(4)$	*b*
	$CO^+ + O^-$	+4.2	—	*c*
$CO_2^+(\tilde{A}) + e^-$	$CO + O$	-11.8	CO: See Notes	*d*, *e*
	$CO^+ + O^-$	+0.7	—	*c*
$CO_2^+(\tilde{B}) + e^-$	$CO + O$	-12.6	CO: See Notes	*d*, *e*
	$CO^+ + O^-$	-0.1	—	*f*
$CO_2^+(\tilde{C}) + e^-$	$CO + O$	-13.9	CO: See Notes	*d*, *e*
	$CO^+ + O^-$	-1.4	—	*f*
$CO_2 + Ar^+$	$CO_2^+ + Ar$	-2.0	—	
	$CO^+ + O + Ar$	+3.7	—	
	$CO + O^+ + Ar$	+3.3	—	
$CO_2 + He_2^+$	$CO_2^+ + 2He$	-8.4	$\tilde{A}$, $\tilde{B}$, $\tilde{C}$	
	$CO^+ + O + 2He$	-2.8	CO^+: $A^2\Pi_i(0)$	
	$CO + O^+ + 2He$	-3.2	—	
$CO_2 + He^+$	$CO_2^+ + 2He$	-10.8	$\tilde{A}$, $\tilde{B}$, $\tilde{C}$	
	$CO^+ + O + He$	-5.1	CO^+: $A^2\Pi_i(14)$	
	$CO + O^+ + Ar$	-5.5	—	
$CO_2 + He 2s^1S_0$	$CO_2^+ + He 1s^1S_0$	-6.8	$\tilde{A}$, $\tilde{B}$, $\tilde{C}$	
	$CO^+ + O + He 1s^1S_0$	-1.2	—	
$CO_2 + He 2s^3S_1$	$CO_2^+ + He 1s^1S_0$	-6.0	$\tilde{A}$, $\tilde{B}$, $\tilde{C}$	
	$CO^+ + O + He 1s^1S_0$	-0.4	—	

Table Notes:

† Negative values of ΔE indicate an exothermic processes.

a Transitions from the D and I states are not observed.

b None of the accessible products are observed.

c Resonant ion pair production (RIP) is not energetically accessible.

d For comparison, the CO X-state bond energy is 11.09 eV and the ionization potential of CO is 14.01 eV.

e All tabulated electronic states of CO [NIST 2001] are accessible, but some are not accessible at all vibrational levels.

f RIP is energetically accessible.

presence of the attaching gas in a manner similar to electrons, e. g., stoichiometrically, and that is present in about half the concentration of electrons at RP3 (the x–axis intercept corresponds to a number density of $\sim 5 \times 10^{10}$ cm$^{-}3$). The other, less linear portions of the CO_2^+ curves most probably correspond to production of the emissions via Penning ionization reactions of metastable helium atoms with CO_2. Metastable He atoms, like the He_2^+, should have mostly reacted with Ar and no longer be present in significant quantities at RP3. He^+ reacts slowly with Ar [Ikezoe 1987], but it is not expected to be present in great quantities (see below). But, emissions in the plasma (Section 4-3) indicate the presence of the metastables in the flow. Additionally, there are few species whose presence in the flow is plausible (see Table 4-1.1) and that possess sufficient energy to create CO_2^+ in excited electronic states (see Table 4-1.2).

If the portion of the CO_2^+ emissions that persist after the addition of the attaching gas is due to Penning ionization, it must necessarily involve He species that are not affected by the presence of electrons. The electronically excited states of He, He_2, Ar and Ar^+ that are observed in spectra from the base plasma (when no reactant gases have been added — see Section 4-3) are all quenched by the addition of an attaching gas. So, none of those species are likely to be the source of the emissions. Since the injected argon is expected to react away any helium metastable species, a mechanism by which the helium metastables could persist is needed. The most likely possibility is that the metastable concentration in the flow exceeds that of the injected Argon. Such a situation would also facilitate the propagation of the observed emissions from He and Ar species.

That the CO^+ $A \rightarrow X$ emissions are present at all is unexpected. There are few species that could be present in the flowtube that are energetic enough to produce those emissions, see Tables 4-1.1 and 4-1.2. In addition to those listed in the tables, atomic helium states at or above $3s^3S_1$ have sufficient energy, as do all electronically

excited states of Ar^+ (assuming deexcitation to the ground state of neutral Ar for lower levels of Ar^+).

Kinetic models do not support the presence of He_2^+, He^+. Neither do they support the presence of either of the He metastables unless they are initially present in concentrations in excess of 3×10^{13} cm^{-3}. Instead, they show that these states should have almost completely reacted away by the time the plasma reaches RP3. Of these, the species with the greatest concentration should be He^+, but its concentration is expected to be very low. Between RP1 and RP2, the termolecular reaction to form He_2^+ is the primary loss mechanism for He^+:

$$He^+ + 2He \xrightarrow{k\sim1\times10^{-31}cm^3/s} He_2^+ + He^* \qquad 42$$

where the asterisk indicates translational excitation in the He. Assuming initial concentrations of 1×10^{11} cm^{-3} for He^+ and 6×10^{16} cm^{-3} for neutral He at RP1 with no diffusive losses, the concentration of He^+ at RP2 should be about 1×10^{10} cm^{-3}. If reactions with Ar are ignored, the concentration of He^+ at RP3 should be about 1×10^{9} cm^{-3}. If diffusive losses are still ignored, the concentration of electrons should be 1×10^{11}, so it is reasonable to expect the He^+ concentration to be, at most, a tenth of the electron concentration at RP2 and much lower at RP2. The plots in Figures 4-1.3 — 4-1.6, however, indicate a species that is highly energetic and is present at about half the concentration of the electrons. Since the emissions observed in the plasma (see Section 4-3) definitely indicate the presence of excited states of He and He_2 (with evidence of excitation to near the ionization limit of both species), a mechanism is implied by which He^+ and He_2^+ are present as well, despite what the kinetic models predict. If the process described in Section 4-3:

$$He(n \geq 3) + He \rightleftharpoons He_2^+ + e \qquad 43$$

is occurring, then the presence of He_2^+ is explained. But, there is no process available for explaining a similar "hidden" population of He^+.

Figure 4-1.7a

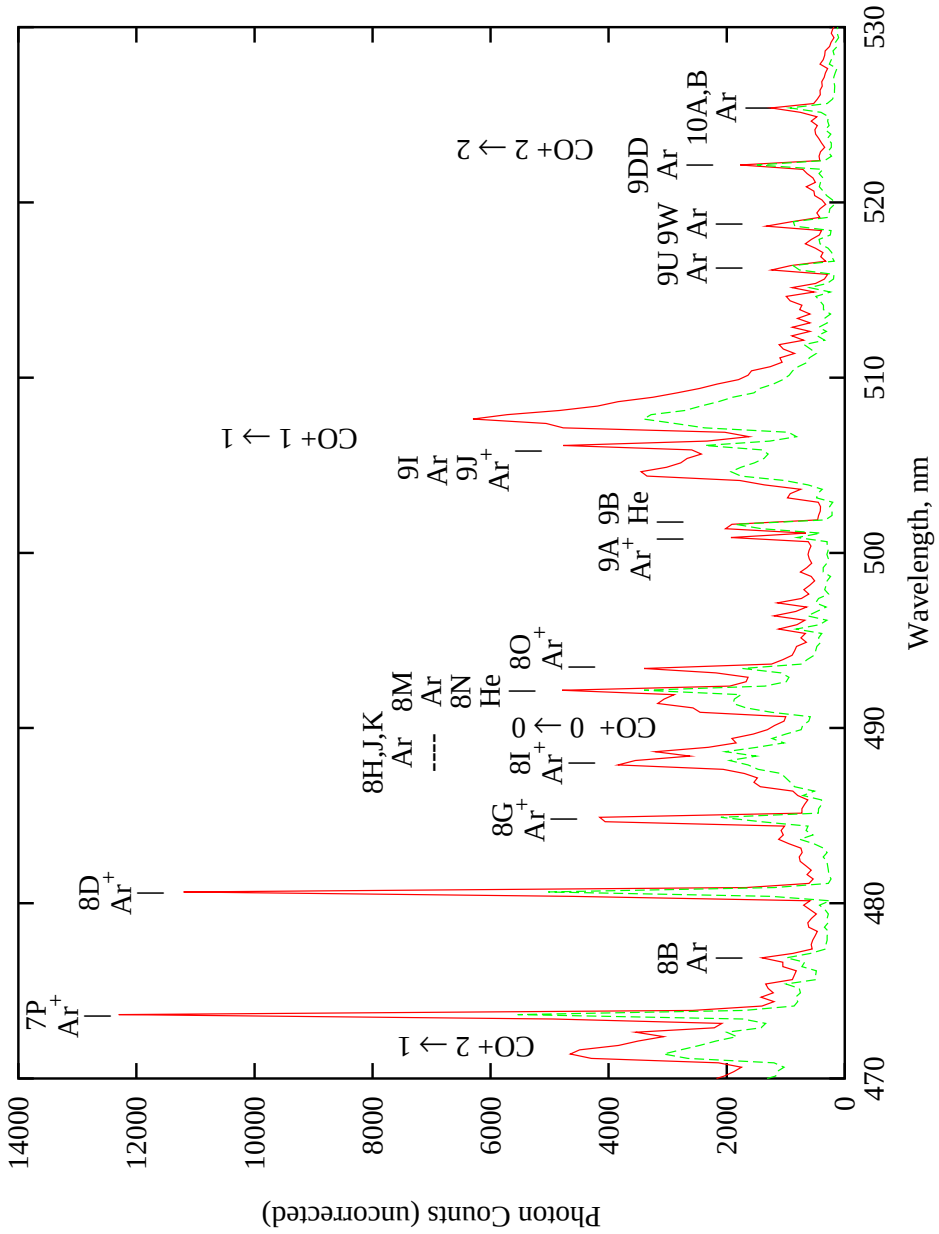


Figure 4-1.7a: Comparison of typical origin spectra (not corrected for optical system efficiency) for a plasma containing recombining CO_2^+ ions (red, solid line) and a plasma containing recombining HCO_2^+ ions (green, dashed line). Labels of the form $n_1 \rightarrow n_2$ denote the upper and lower vibrational levels, respectively, of the CO^+ $A \rightarrow X$ transition. The key to the alpha-numeric labels is in Appendix B-b. See text for further discussion.

Figure 4-1.7b

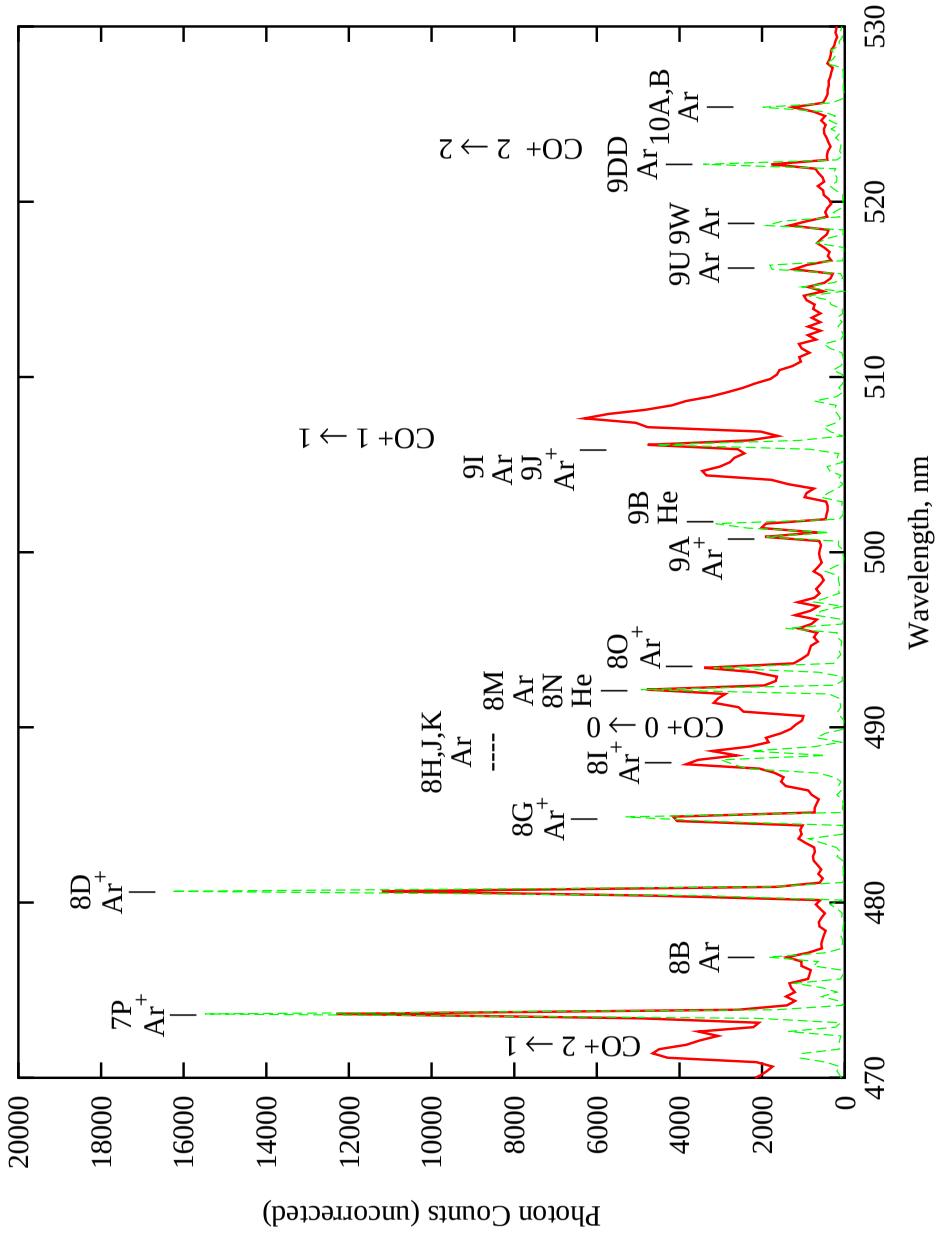


Figure 4-1.7b: Comparison of a typical origin spectrum (not corrected for optical system efficiency) for a plasma containing recombining CO_2^+ ions (red, solid line) to the spectrum obtained when no reagent gases besides He (at RP1) and Ar (at RP2) have been added to the flow (green, dashed line). See the caption of Figure 4-1.7a for information on labeling. See text for further discussion.

There is another reason to doubt that He^+ is the species involved in the production of the CO^+ emissions. Only the lowest three vibrational levels of the CO^+ A state are observed in the spectra (see Figures 4-1.7 for sample emissions). There is sufficient energy to populate the CO^+ A state to v=14 in the reaction of He^+ with CO_2; therefore, observation of vibrational states above v=2 would be likely (though not necessary). It should be noted that, by the energetics calculations provided in Table 4-1.2, only the v=0 vibrational level of $CO^+(A)$ is accessible from the reaction of CO_2 with He_2^+. But note that there is likely to be vibrational excitation in the He_2^+, especially if it is formed by reactions of excited helium atoms ($n \geq 3$) and ground state He, as is suggested in Reaction 2.

The production of the CO^+ emissions by reaction of CO_2 with excited states of He, Ar and Ar^+, whose emissions are observed in the base plasma, can also be discounted. The origin spectra for CO_2^+ and HCO_2^+ are compared in Figure 4-1.7. The wavelength range 470-530 nm was chosen because it contains intense emissions from CO^+, He, Ar and Ar^+. The amount of CO_2 added in the HCO_2^+ spectra is about half that added in the CO_2^+ spectra. As is expected, the intensities of the CO^+ emissions in the HCO_2^+ spectra are lower than those in the other. If the CO_2 were reacting with the Ar^+, the intensities of the Ar^+ emissions should be lower in the CO_2^+ spectrum. Instead, they are significantly lower in the HCO_2^+ spectrum. Note, also, that the relative intensities of the He and Ar neutral emissions did not change significantly. The conclusions, therefore, are that the H_2 reacts with the excited Ar^+, but that the CO_2 does not react, or reacts slowly, with any of the excited species (He, Ar and Ar^+), or, if the reaction occurs, that it does not produce CO^+ in the A state.

Part (b) of Figure 4-1.7 compares emissions from the plasma when only He and Ar have been added to emissions when CO_2 has been added as well. The atomic emission intensities are definitely lower in the spectra where CO_2 has been added. It is certainly possible that the decrease is due to reactions of the excited species

in the flow with the injected CO_2. But, other explanations should be mentioned. One, the flowtube pressure was 1.9 Torr for the plasma–only emissions, higher than the 1.8 Torr when CO_2 was added as well; the additional pressure might favor the production or maintenance of the excited species. Two, the in–flow of argon in the system, under typical conditions is only controlled to one unit of precision, and the change in emission intensity is not greater than that. Three, the injection of the CO_2 at RP3 disturbs the flow of the oncoming plasma and might be causing many emitting species to be diverted out of the main viewing area of the spectrometers.

Based on analysis of the current set of data, then, two conclusions can be drawn. One, it is likely that reactions of CO_2 with vibrationally excited states of He_2^+ produce most or all of the observed CO^+ emissions and part of the observed CO_2^+ emissions. Two, the remaining CO_2^+ emissions are produced by reaction of CO_2 with metastable states of He. These conclusions should be tested. The electron density as a function of attaching gas could be measured with a Langmuir probe. If the electron density is affected less than is expected by the addition of the attaching gas, then a buffering reaction such as the one given in Reaction 4 might be occurring. Similarly, given a robust ion–mass detection system, the concentration of He_2^+ ions, if there are any at all, could be measured as a function of attaching gas flow. To test for a large population of He metastables, the presence of CO_2^+ emissions could be monitored as a function of Ar flow, preferably while a suitable amount of attaching gas is also being added (to quench the electron–dependent emissions). If the additional Ar quenches the emissions, then there are probably many more metastable species in the flow than are expected.

If such a buffering reaction is, indeed, an important mechanism, then it is necessary to explain why the attaching gas affects the He_2^+ as well. The mechanism is fairly simple. First, the attaching gas, AG, reacts with electrons:

$$AG + e^- \xrightarrow{k\sim 3\times 10^{-7} cm^3/s} A + G^- \qquad 44$$

Here, the exact nature of the products should not be taken literally: the attachment might not be dissociative. Rather, the point is that some negative ion, G^- has been formed which will then react rapidly with He_2^+ via mutual neutralization:

$$G^- + He_2^+ \xrightarrow{k\sim 5\times 10^{-8} cm^3/s} G + 2He \qquad 45$$

While the negative ion can also react with Ar^+, it is expected to do so more slowly unless G^- is polyatomic ion. If G^- is atomic (as it is in the attachment of CCl_4 and CH_3I), then the reaction with Ar^+ will be slower, due to the increased number of available channels in the diatomic He_2^+, unless there is an accidental resonance. In the event that G^- is also polyatomic, there will be many accessible channels for reactions with both He_2^+ and Ar^+, with more expected for He_2^+. In all of these cases, of course, the presence of favorable energetics, e. g., curve–crossings, will increase the reactivity of G^- with Ar^+.

Ratio and Difference Plots

Recall from Section 3-4 that a common method for determining the presence of emissions resulting from recombination is to take spectra without and with the presence of an attaching gas. These spectra are called "origin" and "background," respectively. They are compared to each other by taking their ratio and difference. This section discusses practical issues regarding the use of ratio and difference plots.

In recording a set of data, it is desirable to take one origin spectrum and only one background spectrum. The reason for this is entirely practical. The time required to take one origin and one background spectrum across the full 700 nm available spectral range, at a resolution that is only barely high enough for analysis, is around 15 to 20 hours. During this time, the constant attention of the researcher is required and large quantities of reagents are being used. To produce curves like those found in Figures 4-1.3-6 for the entire range would require, then, 60-80 hours. Of course, the time taken can be greatly reduced by taking a lower resolution spectrum first and then

studying regions of interest in more detail. But, even so, research costs are greatly increased if the intensity of every interesting feature must be followed as a function of attaching gas flow.

Before the data presented in this section were analyzed, the understanding of the reactions occurring in the flow was as follows. Side reactions were known to occur in fairly significant quantities [Foley 1993]. But, it was deduced, quite reasonably, that the addition of an attaching gas would always affect emissions due to recombination much more strongly than emissions arising from other sources. Exotic excited states of helium, argon and dihelium were not thought to be present in great quantity. However, in the last section, it was shown that those exotic states are, indeed, present in significant quantities. In this section, it will be shown that a difference or ratio plot is not, by itself, enough data to conclusively identify an emission as being a photoproduct of electron–ion recombination.

The data plotted in Figures 4-1.3 — 4-1.6 can be turned into a series of ratios or differences that would be observed at that wavelength upon addition of each quantity of attaching gas. Calculating a difference is just as simple as calculating a ratio. But, if two features have very different initial intensities, the difference between origin and background will be difficult to compare without knowledge of the initial intensity. Ratios are easy to compare without knowing the initial intensity. So, in order to more easily make meaningful comparisons, ratios of counts are presented here. The ratios of counts observed at several single wavelengths as a function of CCl_4 flow are presented in Figure 4-1.8. The ratio at each point was taken by dividing the number of counts at zero–flow of CCl_4 by the number of counts at each flow of CCl_4. Points in that figure are connected only to help guide the eye.

The emission ratios plotted in Figure 4-1.8 arise from a variety of different sources (indicated on the figure), only two of which, the CO emissions, are likely to be products of the recombination of CO_2^+. Note that at low flows of CCl_4, the two CO

Figure 4-1.8

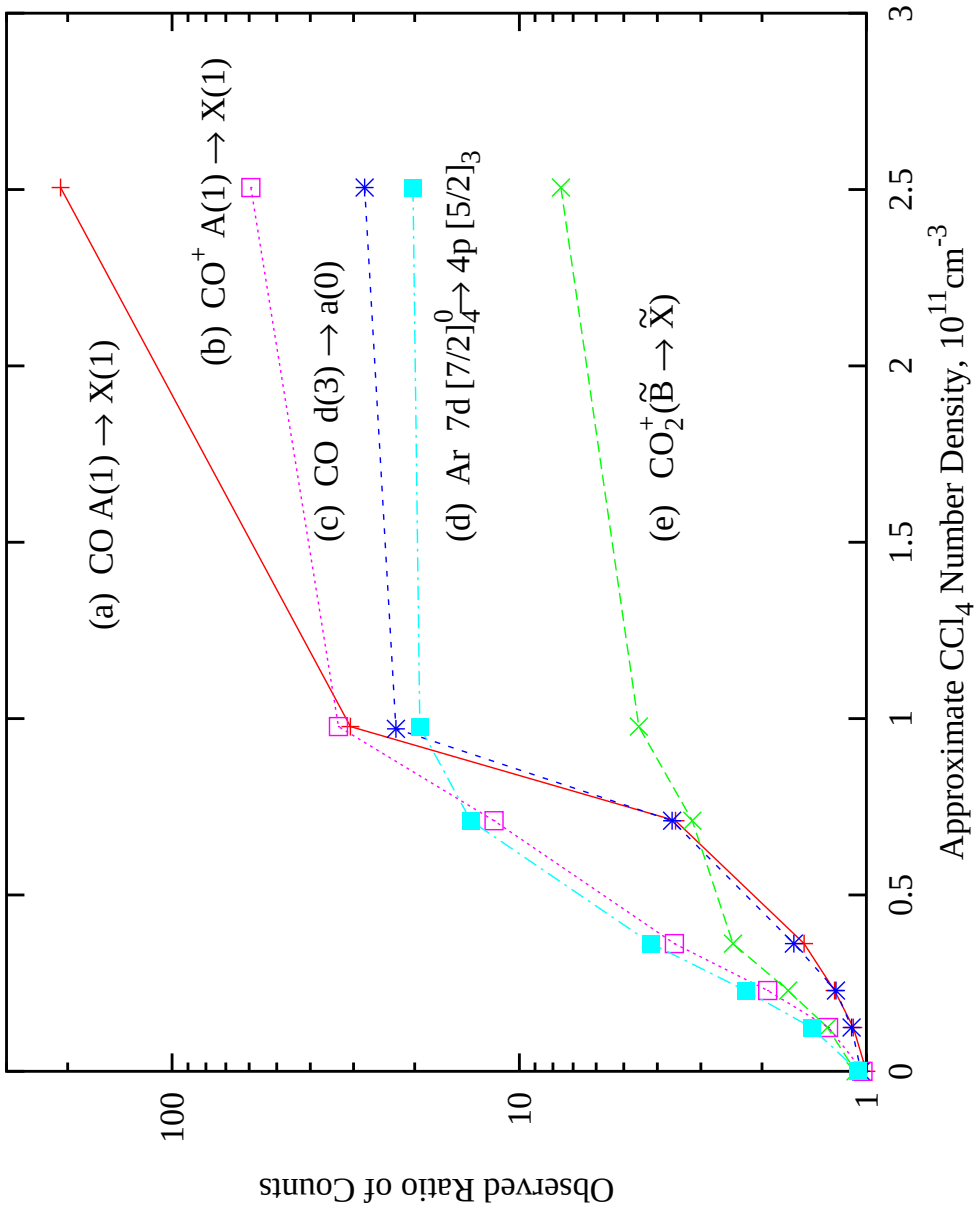


Figure 4-1.8: Comparison of the heights in a ratio plot† that would be observed for several different types of emission as a function of approximate CCl_4 throughput. Emissions b, d and e are unlikely to result from recombination. Emissions a and c that are thought to be due to recombination. See also Figures 4-1.1 and 4-1.2. Transition wavelengths: (a) 156.1 nm, (b) 507.8 nm, (c) 641.8 nm, (d) 522.3 nm, (e) 288.5 nm. † See Section 3-4.

features have very similar ratios. This result is encouraging. Not only are the initial intensities of the peaks different (2,000 counts for the $d \rightarrow a$ and 40,000 for the $A \rightarrow X$), but they also occur in different spectral regions. The difference between their ratios at high flows is a function of the signal–to–noise for the particular peak.

The problematic result is that at low CCl_4 flows, where recombination emissions behave most predictably, emissions that are not expected to result from recombination have comparable or, in most cases, larger ratios. The reason for this is, obviously, that the attaching gas, or a derivative of the attaching gas, is reacting with species in the flowtube other than electrons. But, knowing this does not make a ratio plot more useful.

The next figure, 4-1.9, is a bit cluttered, but shows that the data plotted in Figure 4-1.8 are not exceptional. Figure 4-1.9 is similar to 4-1.8 but includes all the major peak–tops in Figures 4-1.1 and 4-1.2, as well as those from other data taken at the same time. Data points are connected only to help guide the eye and are colored according to the general line shape. Although each set of points is not labeled separately, a listing of the features grouped by color can be found in Table 4-1.3.

Single ratio and difference plots do not provide sufficient evidence that a feature is a photoproduct of recombination. But, features due to recombination can still be identified using this technique. What is required, though, for conclusions to be certain, is to follow a given feature as a function of small flows of attaching gas. If the emission decays linearly with attaching gas throughput, with an x–axis intercept corresponding to the electron density in the injection region, the feature can be identified as a photoproduct of recombination.

Another option is to find a better way to control the makeup of the basic plasma composition (see Section 4-3 for a detailed description of the base plasma). As can be seen by inspection of Table 4-1.2, the excited states of CO^+ and CO_2^+ (which

Figure 4-1.9

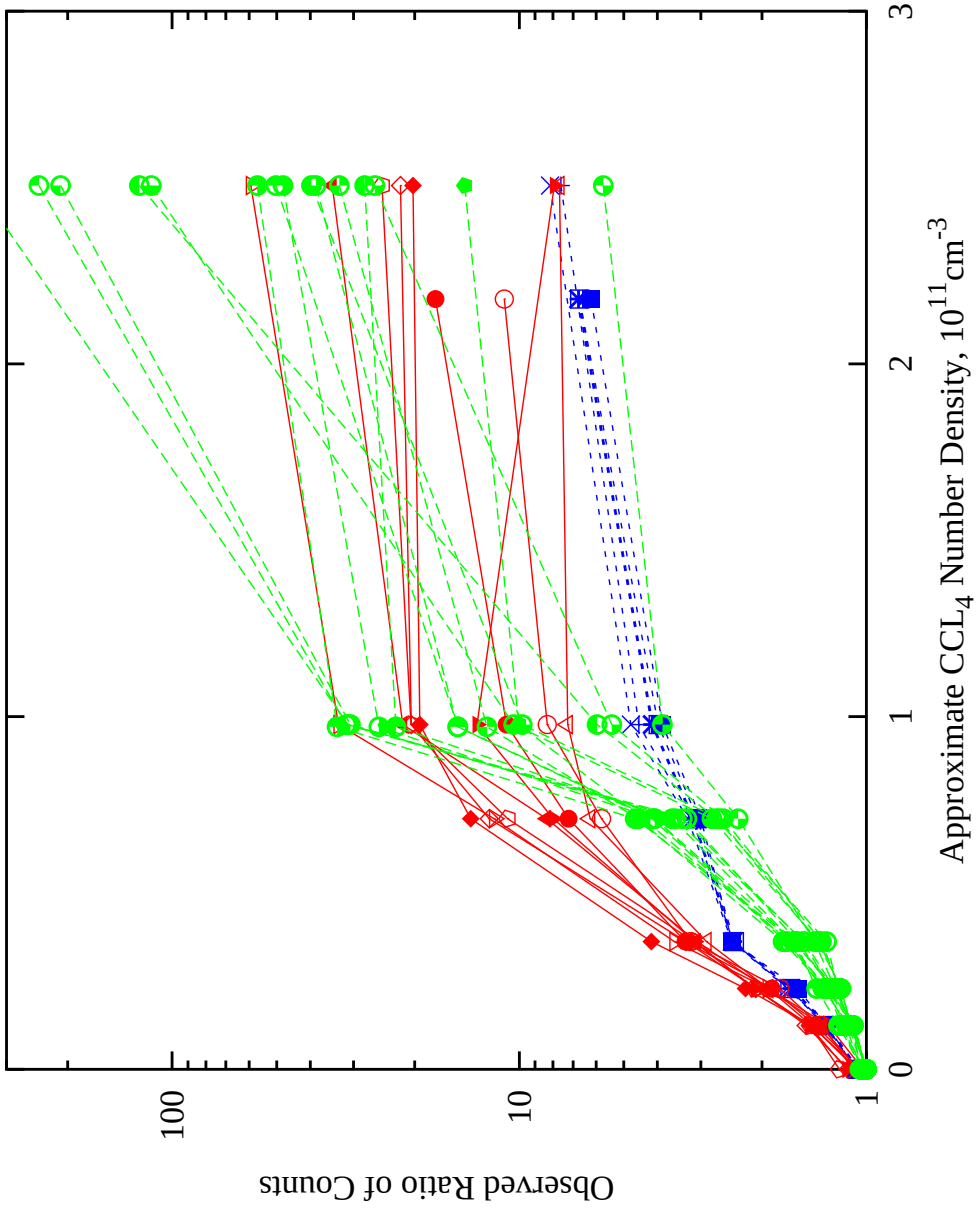


Figure 4-1.9: Height in ratio plots (see Figure 4-1.8) as a function of approximate attaching gas (CCl_4) throughput for many emissions. Note that the quenching of emissions appears, at low throughputs, to arise from three mechanisms. See Table 4-1.3 for further information including a list of feature identifications for each line color. Line styles differ for viewing in black and white copy (see Table 4-1.3).

Table 4-1.3

Features in Figure 4-1.9

Green Lines (long dash):			
148.7009	CO $A(2) \rightarrow X(0)$	151.1021	CO A(1)→X(0)
154.5039	CO $A(0) \rightarrow X(0)$	156.1047	CO A(1)→X(1)
199.3007	CO $a(1) \rightarrow X(0)$	201.1016	CO a(2)→X(1)
205.3038	CO $a(4) \rightarrow X(3)$	206.3043	CO a(0)→X(0)
210.1063	CO $a(2) \rightarrow X(2)$	598.2526	CO d(4)→a(0)
601.2545	CO $d(4) \rightarrow a(0)$	603.756	CO d(4)→a(0)
641.7511	CO $d(3) \rightarrow a(0)$	643.252	CO d(3)→a(0)
646.7542	CO $d(3) \rightarrow a(0)$		
Blue Lines (short dash):			
288.5008	CO_2^+ $\tilde{B}\rightarrow\tilde{X}$	289.7014	CO_2^+ $\tilde{B}\rightarrow\tilde{X}$
337.3012	CO_2^+ $\tilde{A}\rightarrow\tilde{X}$ [a]	337.9015	CO_2^+ $\tilde{A}\rightarrow\tilde{X}$ [a]
340.1026	CO_2^+ $\tilde{A}\rightarrow\tilde{X}$ [a]		
Red Lines (solid):			
341.7035	Unknown	343.1042	Unknown
501.2508	He 9B	504.7532	CO^+ $A(1) \rightarrow X(1)$
507.7552	CO^+ $A(1) \rightarrow X(1)$	516.2607	Ar 9U
518.7622	Ar 9W	522.2643	Ar 9DD
525.2662	Ar 10A		

Table 4-1.3: Figure 4-1.9 compares the height that would be observed in a ratio plot (see Section 3-4) for many different emissions as a function of CCl_4 flow. Lines are colored according to their behavior at number densities less than $2 \times 10^{12} cm^{-3}$. There are too many lines to easily identify each individually on the graph. The point is that all emissions respond to attaching gas flow, with processes related to three different mechanisms. The nature of these mechanisms is discussed in the text. [a] tentative assignment.

---

have not been previously detected) can, energetically speaking, only be produced by plasma species whose presence is unwanted. It would be useful if the ions and neutrals in the plasma could be electronically relaxed so that the excited states are no longer present (see Section 4-2). This should begin at the ionization source, with

enhanced production of ground state He^+ ions and a reduction in excited neutral He species. The types of species produced could be better controlled if there were a device capable of precisely controlling the energy with which the free electrons impact the helium (this is not possible with current technology). To select for ground state He^+ ions, the energy would need to be controlled to within a couple eV. The first excited state of He^+ is 40 eV above the ground state, leaving a significant energy gap between the energy needed to ionize He ($\sim$ 25 eV) and the energy needed to excite the ion. If the electrons resonating in the microwave cavity were provided with an energy intermediate between 25 and 40 eV, they should be more likely to ionize the helium atoms than to merely excite them. If the energy were kept close to 25 eV, then electrons that have already ionized a helium atom would not be left with sufficient energy ($\sim$19 eV) to electronically excite other He atoms.

4-2 Photoproducts of Dissociative Recombination

Techniques used for analysis, difference and ratio plot, comparisons to Franck–Condon factors, etc., will be mentioned or implied frequently. These techniques are discussed in moderate detail in Chapter 3.

A survey of emissions likely to be products of recombination is presented here. The systems to be presented and the subsection in which they can be found are:

Parent Molecule(s)	Primary Ion(s)	Section Number	Parent Molecule(s)	Primary Ion(s)	Section Number
CH_4/H_2	CH_5^+	4-1.1	NH_3/H_2	NH_4^+	4-1.6
CO/H_2	HCO^+	4-1.2	NO/H_2	HNO^+	4-1.7
CO_2	CO_2^+	4-1.3	N_2O/H_2	N_2OH^+	4-1.8
CO_2/H_2	CO_2H^+	4-1.4	H_2S/H_2	SH_3^+	4-1.9
CS_2	CS_2^+	4-1.5	Xe/H_2	XeH^+	4-1.10

It is important, for each system, to determine whether the observed emissions can possibly be produced by processes other than electron–ion recombination. To facilitate this determination, two tables of energetics are included here. The first, Table

4-1.1 (presented in Section 4-1), contains enthalpy changes associated with processes involving species known to be present in the plasma. The second, Table 4-2.11, lists the energies required to generate, from neutral, ground state parent molecules, the excited states whose emissions are observed in the spectra. Ground state exothermicities for relevant recombination paths are included in the tables (Tables 4-2.1 – 4-2.10) accompanying the discussion of each recombining system. In the case of unprotonated recombining ions, the exothermicity of charge transfer from ground state Ar^+ is also included; for protonated recombining ions, the exothermicity for proton transfer from H_3^+ is included. When the tabulated information is not sufficient, the reader will need to refer to other sources for the pertinent data, e. g., the references in Table 4-1.1.

It is also important to determine the relative frequencies with which certain processes occur. For all recombining ions, a primary issue is the likelihood of collisional vibrational relaxation prior to recombination. Collisional vibrational relaxation proceeds most efficiently when there exist normal modes that are resonant between the excited species and the relaxing agent. But, for small molecules, vibrational frequencies are often different in the neutral species than in the ion (see Section 3-6 for further discussion on vibrational relaxation). Also, while a greater number of collisions increases the likelihood of non–resonant relaxation, the injection of the necessary amount of reagent can cause other problems, such as forcing the injected gas several centimeters up the flowtube. In order to facilitate discussion of the extent of vibrational relaxation, Table 4-2.12 contains an estimated number of collisions, $Z_{V,T}$, required to relax an ion via non–resonant (e. g., vibrational–translational) relaxation and an estimated number of collisions the ion will undergo with the neutral parent, $f_{v,XYZ}$, and with He, $f_{v,He}$, before it recombines.

When the recombining ion is protonated, two additional relationships are calculated. The first is a comparison of the frequency with which Ar^+ will react with

H_2 to the frequency with which it will react with the unprotonated, neutral parent. This gives an idea whether a significant portion of the observed spectrum is likely to be due to recombination of the unprotonated ion. The second relationship is the frequency with which any unprotonated ions formed will react with H_2 rather than recombining. For this last relationship, the nature of the products is less important than the fact that there will be fewer unprotonated parent ions recombining. These comparisons are discussed in Section 3-7 and in relevant discussions in Sections 4-2.1 — 4-2.10.

In cases where spectra were obtained at high enough resolution to determine precise values for vibrational level populations, normal–mode frequencies for the parent molecules and ions are listed in Table 4-2.13. In a number of cases, experimental values for one or more normal–mode frequencies were not available. Because of this, normal–mode frequencies were calculated for each of the molecules in its ground electronic state. In order to maintain as much consistency as possible, all colliding partners were treated with identical calculation methods. The calculations used the Hartree–Fock approximation with self consistent field. Since approximate values are sufficient for this comparison, the basis sets were chosen for economy. A slightly larger basis set was required for the nitrogen containing species (see the table for specifics). The implications of results presented in Table 4-2.13 are discussed in each of the appropriate sections below.

The program RVESIM (rotational vibrational electronic simulation) was used to determine relative vibrational and electronic populations and to estimate the temperature of each of the emitting species. See Chapters 6 and 3, respectively, for further discussion of the program and of the techniques employed in data analysis. For each recombination, one or more graphs comparing the experimental spectrum to the simulated spectrum are presented in the section discussing that recombination.

Specific information regarding observed spectra can be found in the tables, figures and discussions in Sections 4-2.1 — 4-2.10. Pre–run mass spectra for as many runs as are available are included in Appendix D. Recombination spectra with simulation fits are included in the text. Brief information specific to reaction conditions for each experiment is given in the table associated with that experiment. Unless otherwise indicated, all experiments share the following characteristics: He throughput is 220 Tls^{-1} and the overall flowtube pressure is 1.8 Torr; Ar is added at about 1 mT pressure through RP2; H_2 is also added at about 1 mT pressure through RP3; the throughput of the attaching gas is approximately $9 \times 10^{10}\text{cm}^{-3}$.

All recombining systems surveyed here, except CO_2^+, were studied before the diagnostic experiments described in Section 4-1.1. Although emissions observed in spectra from those other systems are likely to be photoproducts of recombination, recombination cannot unequivocally be identified as their source. Despite the fact that these survey experiments were not conducted under the circumstances dictated by the results from the CO_2^+ studies, they contain useful information that can facilitate future studies.

4-2.1 Emissions from a plasma dominated by recombination of CH_5^+ ions

The Ar^+ in the system is expected to react with CH_4 rather than with H_2 about 20% of the time (see Table 4-2.12). Of that 20%, only about 20% will recombine before reacting with H_2. Therefore, $\leq$4% of the recombinations whose emissions appear in these spectra will be due to CH_2^+, CH_3^+ or CH_4^+ recombining rather than CH_5^+.

The emissions in this spectrum are very weak. They are dwarfed by the helium and argon emissions in the base plasma. But, the spectrum fit by RVESIM clearly indicates the presence of emissions from the $A \rightarrow X$ and $B \rightarrow X$ transitions of CH (see Figure 4-2.1 and Table 4-2.1). The recombination of ground state CH_5^+ does not release sufficient energy to produce the B state of CH. The presence of the B state

Table 4-2.1

Emissions from a plasma containing recombining CH_5^+ ions

Recombination products [exothermicity,[a] eV]:
CH_4 + H [7.97]; CH_3 + H_2 [7.94]; CH + $2H_2$ [3.29]
Non-destructive protonation exothermicity[a,b] [1.26]

Product	CH	
State	$A^2\Delta^n$	$B^2\Sigma^{-m,n}$
τ, μs [a]	0.5	0.36
v_{max}, v^*_{max} [b]	0, 5	j, 5
Rel. Pop. [c]	1	0.4
Rot. Temp., K [d]	2500 K	2000 K
v [e]		
0	1	1

Reagent(s)[r] (cm^{-3}): CH_4, 3.6×10^{12} ; H_2, 2.3×10^{13}

Attaching gas[r] (cm^{-3}): SF_6, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

is probably due to excitation in the recombining ion. The extra energy needed by the recombining system is .05 eV (406 cm^{-1}). At room temperature, the Boltzmann distribution predicts that about 12% of the molecules possess this much energy. So, 12% of the CH_5^+ ions will have this much energy even if they have been collisionally relaxed.

The results presented here are in agreement with earlier results on a similar system [Tsuji 1991]. The previous work concerns recombinations of CH_n^+ where $n = 2 - 4$. In that work, the $C^2\Sigma^+$ state of CH was also observed. The C state of CH is

Figure 4-2.1

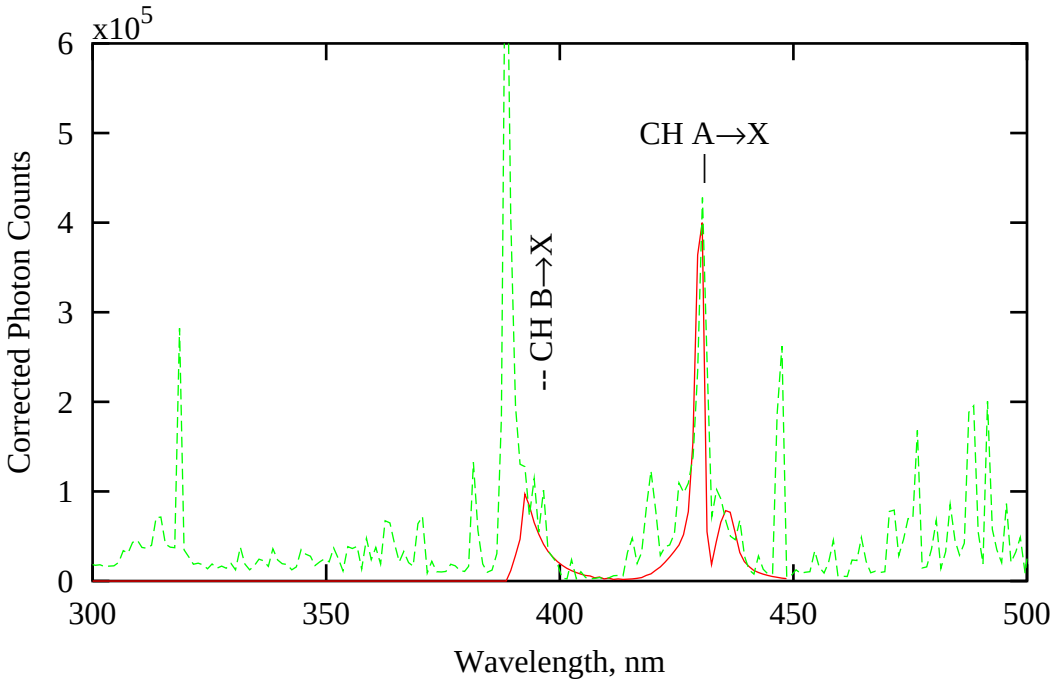


Figure 4-2.1: Comparison of the simulated spectrum (red, solid) to the experimental spectrum (green, dashed) observed in a plasma containing recombining CH_5^+ ions. See Table 4-2.1 for additional information. Emissions not identified here are emissions in the base plasma; see Appendix B for specific identifications. Observed photon counts are corrected for the optical system transmission efficiency.

not accessible in the recombinations of either CH_4^+ or CH_5^+. So, it is not expected here and was not observed. Rotational temperatures were assigned to individual vibrational levels in the previous work. Such precision is not possible in these studies, but rotational temperatures reported here (see Table 4-2.1) are within the range of temperatures reported in the other study. The temperatures reported by the previous work range from 500 to 3200 K for the A state and 600 to 3200 for the B state; rotational temperatures for the C state were not reported. Vibrational distributions in the two studies are also similar, but not exactly the same. While both studies

Table 4-2.2

Emissions from a plasma containing recombining HCO^+ ions

Recombination products [exothermicity,[a] eV]:
H + CO [7.45,[†] 9.18[‡]]; C + OH [0.73,[†] 2.46[‡]]; O + CH [-0.17,[†] 1.55[‡]]
Non-destructive protonation exothermicity[a,b] [1.78,[†] 0.04[‡]]

Product	CO				
State	$A^1\Pi$ [n,o]	$a^3\Pi_r$ [n]	$a'^3\Sigma^+$ [m,o]	$d^3\Delta_i$ [m,o]	$e^3\Sigma^-$ [m,n,o]
τ, μs [a]	0.011	8000	10[l]	7.3[l]	2[l]
v_{max}, v^*_{max} [b] †	*j*, 6	6, 16	3, 17	*j*, 12	*j*, 9
v_{max}, v^*_{max} [b] ‡	5, 6	16, 16	16, 17	11, 12	9, 9
Rel. Pop. [c]	0.15	0[s]	1	1	≤0.04
Rot. Temp., K [d]	600 K	—	400 K	600 K	*g*
v [e]					
0	1	–	*i*	*i*	1
1	0.1		*i*	1[f]	
2	0.1		*i*	0.5	
3	0.05		*i*	0.5	
4			*i*	0.2	
5			1[f]	0.1	
6			0.4		
7			0.1		

†HCO^+; ‡COH^+

Reagent(s)[r] (cm^{-3}): CO, 5.2×10^{12} ; For protonated parent gas: H_2, 2.2×10^{13}

Attaching gas[r] (cm^{-3}): SF_6, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

report population only in the v=0 level of the B state, the A state populations differ. Scaled relative to 1, the results from Tsuji, et al, are 1, 0.4 and 0.1 for vibrational

Figure 4-2.2a

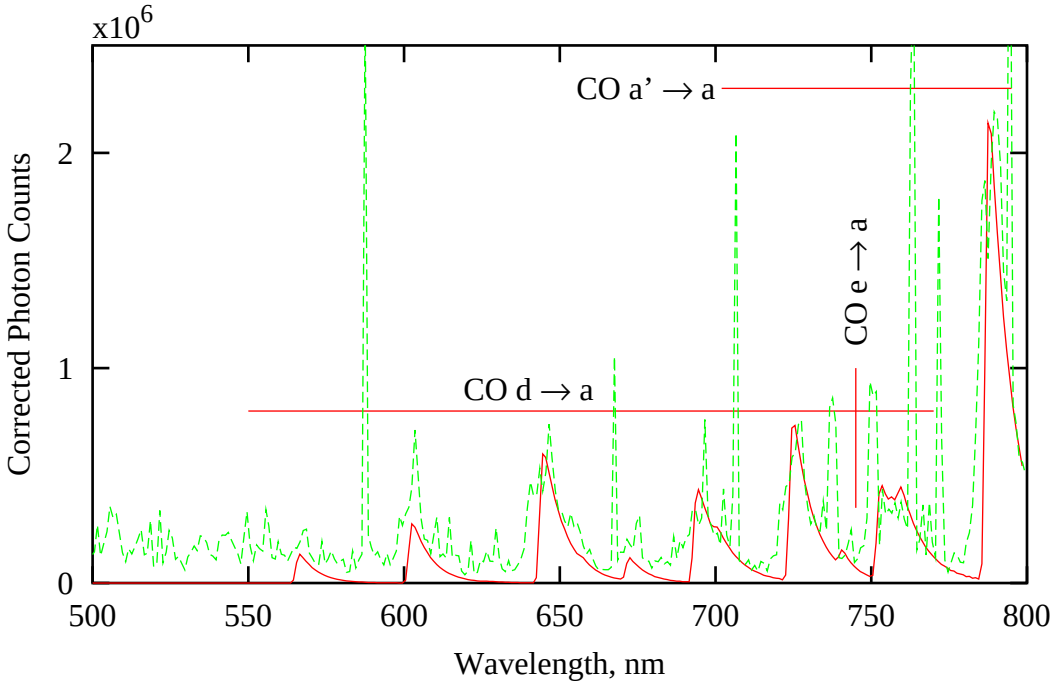


Figure 4-2.2a: Comparison of the simulated spectrum (red, solid) of triplet-triplet transitions to the spectrum (green, dashed) observed in a plasma containing recombining HCO^+ and COH^+ ions. See Table 4-2.2 for additional information. Emissions from the base plasma are not simulated here; see Appendix B for specific identifications. Observed photon counts are corrected for the optical system transmission efficiency.

---

levels 0, 1 and 2, respectively, of the CH A state. Here, only population in the v=0 state is reported.

There is slim evidence of cluster formation in the mass spectrum (see Appendix D). The peak at mass 29 could be $C_2H_5^+$, but the presence of a peak around 19 amu (H_3O^+) suggests a significant level of impurities in the flow. Because of this, the peak at 29 amu is probably N_2H^+ instead. Additionally, the ion detection system nearly a meter downstream of RP3, giving ample time for ion–molecule reactions to occur.

Figure 4-2.2b

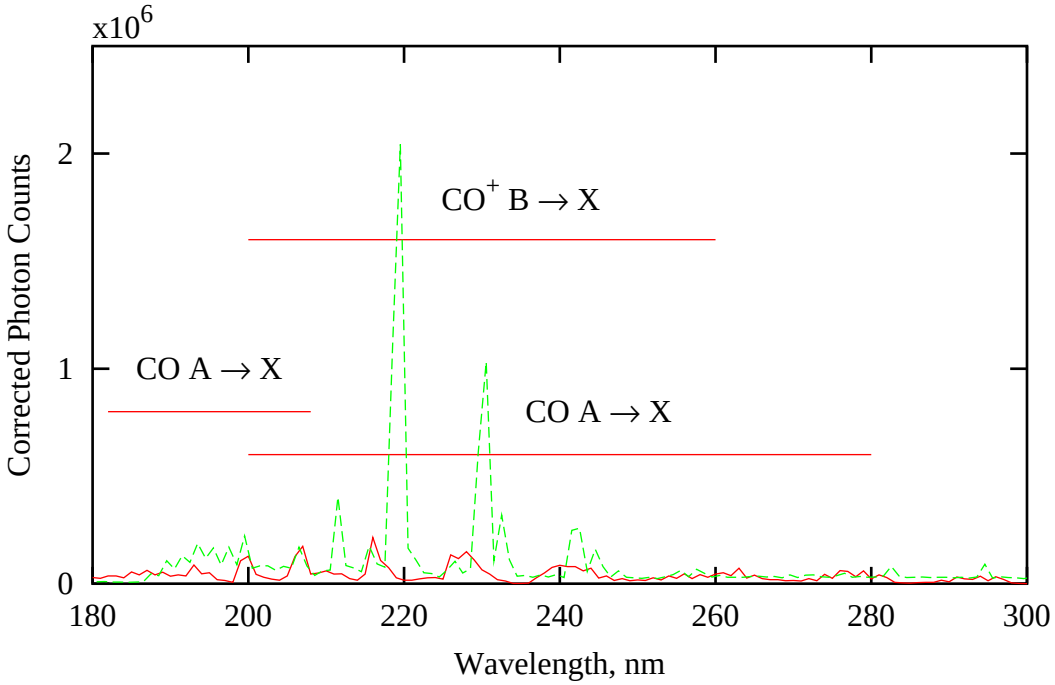


Figure 4-2.2b: Comparison of the simulated (red, solid) CO $a \rightarrow X$ cascade (from higher triplet levels) spectrum to the spectrum (green, dashed) observed in a plasma containing recombining HCO^+ and COH^+ ions. See Table 4-2.2 for additional information. The very intense emissions that are not simulated here are emissions from CO^+ $B \rightarrow X$. Weaker emissions from 180-200 nm are CO $A \rightarrow X$ transitions. Observed photon counts are corrected for the optical system transmission efficiency.

Since clusters do not appear to be forming in the injection region, an increased flow of CH_4 would probably improve the signal to noise in the spectrum. Since the emissions are so weak despite the relatively short radiative lifetimes of the CH A and B states (0.5 and 0.36 μs, respectively), it is possible that they are produced by the reaction with CH_4 of one of the excited species known to exist in the flow. The ionic species in the flow do not possess enough energy to produce the A and B states, but several neutral species could (see Tables 4-1.1 and 4-2.11).

Figure 4-2.2c

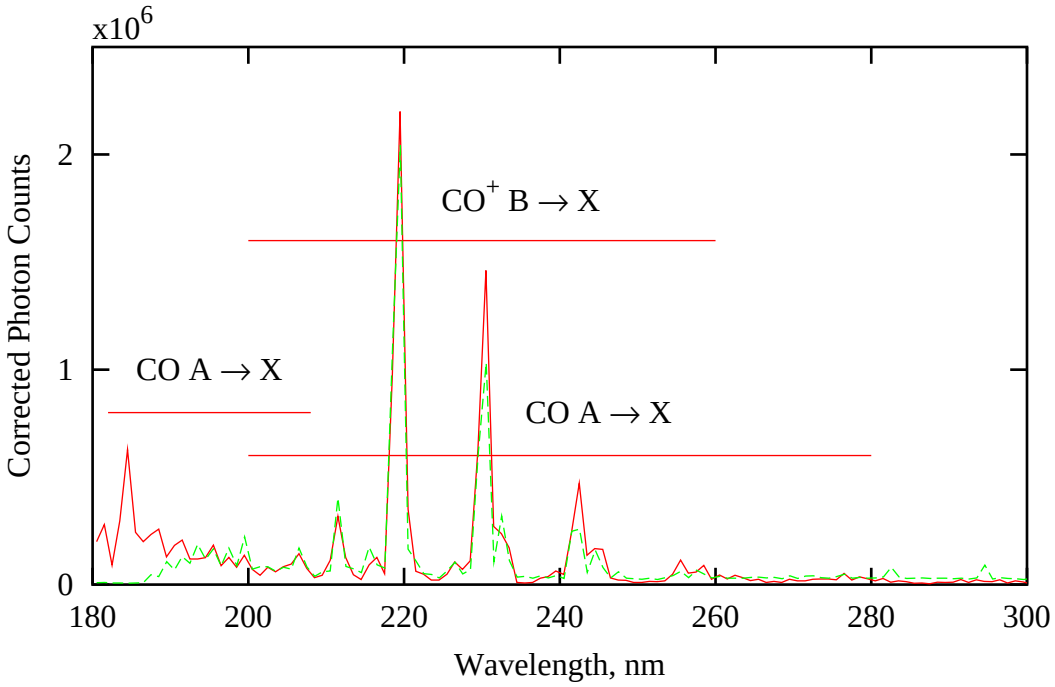


Figure 4-2.2c: Comparison of the 180-300 nm region of the simulated spectrum (red, solid) to the spectrum (green, dashed) observed in a plasma containing recombining HCO^+ and COH^+ ions. See Table 4-2.2 for additional information. Included in this simulation are cascade from the a', d and e states of CO, the very intense emissions from CO^+ $B \rightarrow X$ and the weaker emissions (180-200 nm) of CO $A \rightarrow X$ transitions. No native population of CO was included. Observed photon counts are corrected for the optical system transmission efficiency.

4-2.2 Emissions from a plasma dominated by recombination of HCO^+ and COH^+ ions

The triplet–triplet transitions (d, e and $a' \rightarrow a$, see Figure 4-2.2a) are the most obvious features likely to result from recombination. The lower wavelengths (180-280 nm) are dominated by CO^+ $B \rightarrow X$ emissions (see Figures 4-2.2b and 4-2.2c). Some

Table 4-2.3

Relative Populations of Emissions Resulting from the Recombination of CO_2^+

Recombination products [exothermicity,[a] eV]:
CO + O [8.26]; C + O_2 [2.27]
Non-destructive charge transfer exothermicity[a,b] [1.98]

Product	CO				
State	$A^1\Pi$	$a^3\Pi_r$	$a'^3\Sigma^+$	$d^3\Delta_i$	$e^3\Sigma^-$
τ, μs [a]:	0.011	8000	10^l	7.3^l	2^l
v_{max}, v^*_{max} [b]	0, 13	11, k	9, k	5, k	2, 19
Rel. Pop. [c]	0.06	1^s	0.09	0.06	$\leq$0.003
Rot. Temp., K [d]	600 K	300 K	400 K	400 K	g
v [e]					
0	1	1	i	i	1
1	0.7	0.6	i	1	
2	0.2	0.4	i	0.9	
3	0.1	0.2	i	0.9	
4	0.05	0.1	i	0.5	
5			1	0.2	
6			0.1	0.05	
7			0.1		

Reagent(s)[r] (cm^{-3}): CO_2, 1×10^{14}

Attaching gas[r] (cm^{-3}): CH_3I, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

transitions from CO $A \rightarrow X$ are evident as well. See table 4-2.2 for a summary of the results.

The results from the simulation (Figure 4-2.2b) of the spectrum due only to cascade (see Chapter 6) from the a′, d and e states, implies that no CO $a^3\Pi_r$ is

Figure 4-2.3a

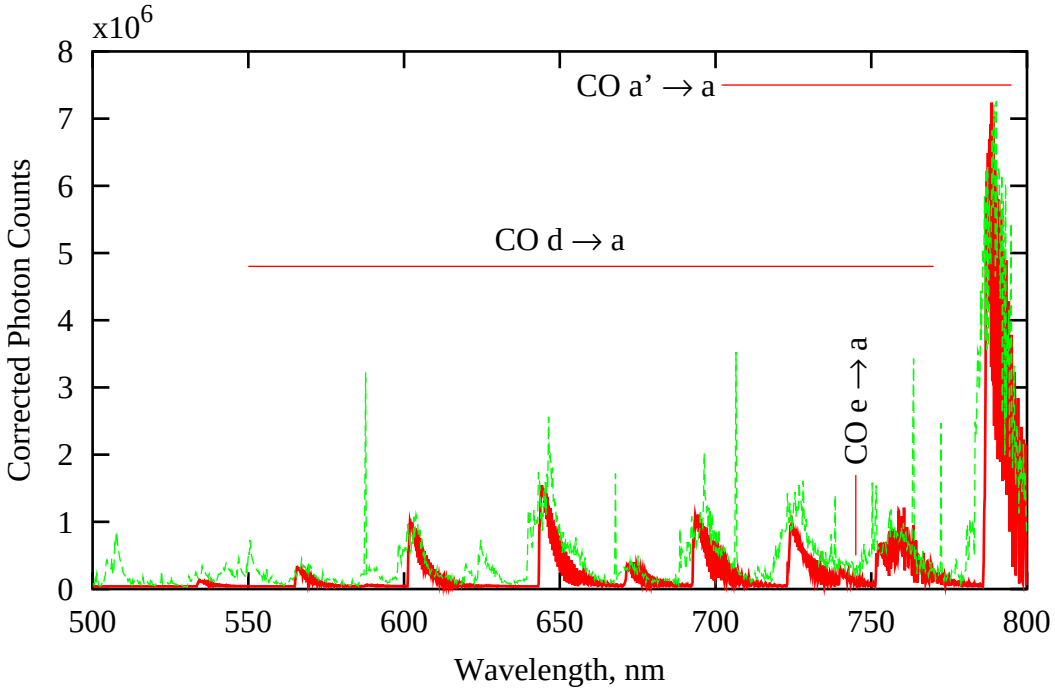


Figure 4-2.3a: Comparison of the simulated spectrum (red, solid) of triplet-triplet transitions to the spectrum (green, dashed) observed in a the recombination of CO_2^+. See Table 4-2.3 for additional information. Molecular emissions not simulated here are CO^+ $A \rightarrow X$. Atomic emissions not simulated are emissions in the base plasma; see Appendix B for specific identifications. Observed photon counts are corrected for the optical system transmission efficiency.

---

produced directly from any process. Note that the simulation is based on the estimates of the simulator rather than on a more rigorous mathematical deduction. The most uncertain estimate is that the CO molecules spend 0.3 ms in the viewing region (see Table 4-2.10, note *s*). If the time in the viewing region is less than this, then some CO a might be produced directly. A longer residence time would indicate that the system efficiencies at low and high wavelengths are incorrect with respect to each other. Even with the uncertainties, it appears that only small quantities, if any, of the

Figure 4-2.3b

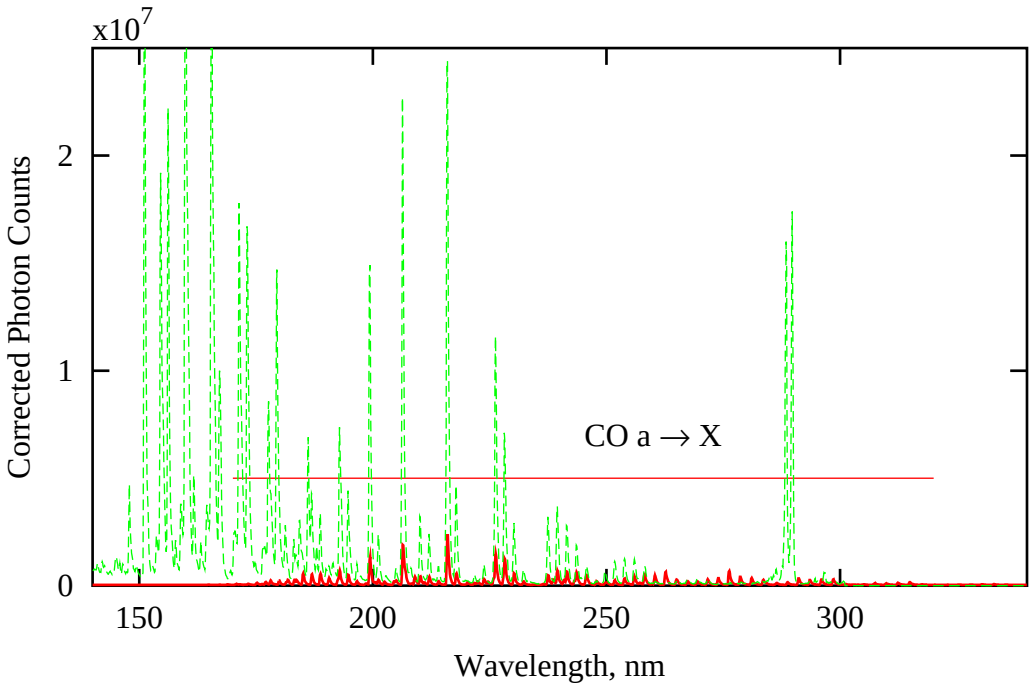


Figure 4-2.3b: Comparison of the simulated (red, solid) CO $a \rightarrow X$ cascade (from higher triplet levels) spectrum to the spectrum (green, dashed) observed in the recombination of CO_2^+. See Table 4-2.3 for additional information. The very intense emissions that are not simulated here are: 280-300nm, CO_2^+ $\tilde{B} \rightarrow \tilde{X}$; 140-200 nm, CO $A \rightarrow X$; and, 180-300 nm, CO $a \rightarrow X$ produced directly from the recombination of CO_2^+ (see next figure for a simulation including CO $a \rightarrow X$ produced directly). Observed photon counts are corrected for the optical system transmission efficiency.

---

a state of CO are being produced directly by any process (including recombination). The presence of CO d and A emissions (and possibly e) indicates that either some COH^+ is reacting or that the HCO^+ is vibrationally excited.

The results for the *a* state do not agree with previous findings [Adams 1994a,b]. In that study, emissions from higher triplet states were not observed and therefore, cascade was not considered. Emissions from the A state were also not observed. The

Figure 4-2.3c

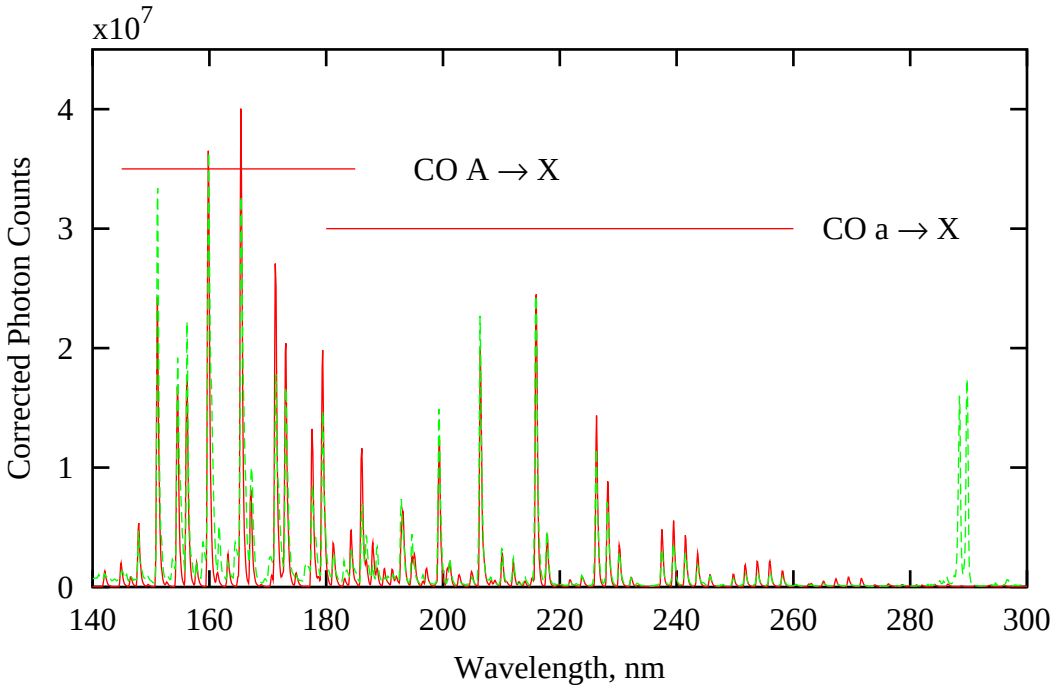


Figure 4-2.3c: Comparison of the 140-300 nm region of the simulated spectrum (red, solid) to the spectrum (green, dashed) observed in the recombination of CO_2^+. See Table 4-2.3 for additional information. Included in the simulation are cascade from the a', d and e states of CO, and CO $a \rightarrow X$ and $A \rightarrow X$ emissions from native (nascent) populations. The intense CO_2^+ $\tilde{B} \rightarrow \tilde{X}$ emission (280-300 nm) is not included in the simulation. Observed photon counts are corrected for the optical system transmission efficiency.

reason for the difference is not yet known. The inability to quantitatively consider cascade from higher states has been a hindrance elsewhere [Tsuji 1998, Johnsen 2000b], and is one of the reasons that RVESIM (see Chapter 6) was created. Emissions from the a, a′, d, e and A states of CO were observed in another study [Johnsen 2000b], but they were not quantified.

4-2.3 Emissions from a plasma dominated by recombination of CO_2^+ ions

Emissions from the recombination of CO_2^+ have been studied here, extensively, and elsewhere to varying degrees [Johnsen 2000b, Skrzypkowski 1998, Tsuji 1995, Valée 1986, Weller 1967]. This system required extensive study here because it produced an unexpected result that has not been indicated in other studies: emissions from a species, CO^+, that should not be a product of recombination, were quenched by the addition of an attaching gas (this occurs with the HCO_2^+ system as well). Also, vibrational levels that are not energetically accessible products of recombination of the ground state ion were observed (see Tables 4-2.3). These were not observed in great quantity, and the exothermicity of charge transfer from Ar^+ is sufficient to account for the residual excitation. The unexpected behavior of the CO^+ was addressed in detail in Section 4-1.

Since a series of emissions from a plasma dominated by CO_2^+ recombinations were recorded as a function of attaching gas flow, it is possible to identify some of them as being photoproducts of recombination. The emissions from neutral CO all decayed linearly as a function of attaching gas flow, with intercepts corresponding to electron densities typical of the region around RP2. The electronic states observed, their temperatures and relative electronic and vibrational populations, are included in Table 4-2.3. Relative vibrational populations were determined using RVESIM (see Chapter 6).

Previous studies [Tsuji 1998, Skrzypkowski 1998, Johnsen, 2000b] were unable to estimate the amount of CO a state that is produced nascently (directly) in the recombination of CO_2^+ rather than by cascade from the higher triplet states (a', d and e). Tsuji, et al., suggested that the recombination of CO_2^+ produces exclusively the upper states (A, a', d and e) of CO. However, in the current study, cascade from the upper states is not sufficiently intense to account for the observed $a \rightarrow X$ emissions. Tsuji, et al., also report significant population in the e state at v=2 and 3, which is

not observed here. In other respects, there is fair agreement between this work and that of Tsuji, et al. [1998], but in the case of the d state populations, the current results are in better agreement with their predictions based on statistical theory:

CO d	v:	1	2	3	4	5	6
This work:		1	.9	.9	.5	.2	.05
Tsuji et al. (obs.)		1	1.4	1.6	0.5	0.1	0.09
Tsuji et al. (pred.)		1	0.7	0.5	0.3	0.1	0.02

CO a'	v:	5	6	7
This work:		1	.1	.1
Tsuji et al. (obs.)		1.00	0.40	0.11
Tsuji et al. (pred.)		1	0.7	0.52

CO A	v:	0	1	2	3	4
This work:		1	.7	.2	.1	.05
Tsuji et al. (obs.)		1.00	0.58	0.09		

Here, "pred." refers to Tsuji, et al.'s predicted populations based on statistical theory and "obs." indicates their experimental values. Values for relative vibrational distributions are scaled for comparison with this study so that values of one occur at the same vibrational level in both data sets. Skrzypkowski, et al., [1998] reported vibrational distributions for the CO a state based on the $a \rightarrow X$ emissions. They were also unable to account for cascade. But, according to the simulation in the present study, the contribution from cascade is small, so their vibrational populations should be fairly accurate. The agreement with the current study is quite good:

CO a	v:	0	1	2	3	4
This work:		1	.6	.4	.2	.1
Skrzypkowski, et al.		1	0.51	0.35	0.20	

They [Skrzypkowski 1998] describe the distribution as approximately Boltzmann with a vibrational temperature of 4200 ± 300 K. They do not report a rotational temperature.

Tsuji, et al., generally report separate rotational temperatures for each vibrational level. Currently, RVESIM (see Chapter 6) is not capable of making that sort of distinction. To the extent possible, however, the temperatures reported here and by Tsuji, et al., will be compared. For the A state, they report temperatures for the v=0, 1 and 2 levels of 1000, 700 and 400 K, respectively. Here, our reported value is 600 K, well within their range. The temperatures they report for the a' state range from 400-550 K, which is also in agreement with the value presented here (400 K). However, they report temperatures in the range 500-750 K for the d state, which is somewhat higher than the 400 K observed here.

4-2.4 Emissions from a plasma dominated by recombination of HCO_2^+ ions

Here, as is also the case in the CO_2^+ system, CO^+ $A \rightarrow X$ emissions are present and quenched by the addition of an attaching gas (see also Sections 4-1 and 4-2.3). Also, other species in the flow (He^*, He^+, He_2^*, Ar^*, Ar^{+*}) possess sufficient energy to create all observed neutral emissions. However, with the exception of excited states of Ar neutrals, those species contain so much energy that to observe only such low–energy products seems unlikely. For example, the more energetic of the two metastable Argon atoms (11.72 eV), could react with CO_2 to produce the v=0 level of the $a^3\Pi_r$ state of CO but not any other excited states. On the other hand, the lowest energy metastable He state contains sufficient energy (19.82 eV) to either completely dissociate CO_2 into atoms or to produce CO^+ (ground state) and an O atom. Thus, the ionized species either do not have enough energy to produce excited states of neutral CO (ground state Ar^+) or the have energies greatly in excess of that needed (He^+, He_2^+, Ar^{+*}). The emissions are also intense, and reactions between

Table 4-2.4

Emissions from a plasma containing recombining HCO_2^+ ions

Recombination products [exothermicity,[a] eV]:
CO_2 + H [8.00]; CO + OH [6.92]; O_2 + CH [0.02]
Non-destructive protonation exothermicity[a,b] [1.23]

Product	CO					OH[h]
State	$A^1\Pi$	$a^3\Pi_r$	$a'^3\Sigma^{+m,o}$	$d^3\Delta_i{}^{m,o}$	$e^3\Sigma^{-m,n,o}$	$A^2\Pi_i{}^m$
τ, μs [a]	0.011	8000	10^l	7.3^l	2^l	0.69
v_{max}, v^*_{max} [b]	j, 0	3, 10	j, 8	j, 3	j, 0	9, k
Rel. Pop. [c]	0.07	0.7^s	1	0.8	$\leq$0.05	0.12
Rot. Temp., K [d]	800 K	300 K	400 K	400 K	g	1000 K
v [e]						
0	0.8	1	i	i	1	0.7
1	1	0.5	i	1^f		1
2	0.4	0.3	i	0.4		.2
3	0.1	0.3	i	0.3		
4		0.1	i	0.2		
5			1^f	0.2		
6			0.2	0.1		
7			0.2			
8			0.1			
9			0.1			

Reagent(s)[r] (cm^{-3}): CO_2, 6×10^{13} ; H_2, 1×10^{14}

Attaching gas[r] (cm^{-3}): CH_3I, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

neutrals are slow, so it is unlikely that they are the result of a reaction between two neutral species.

Figure 4-2.4a

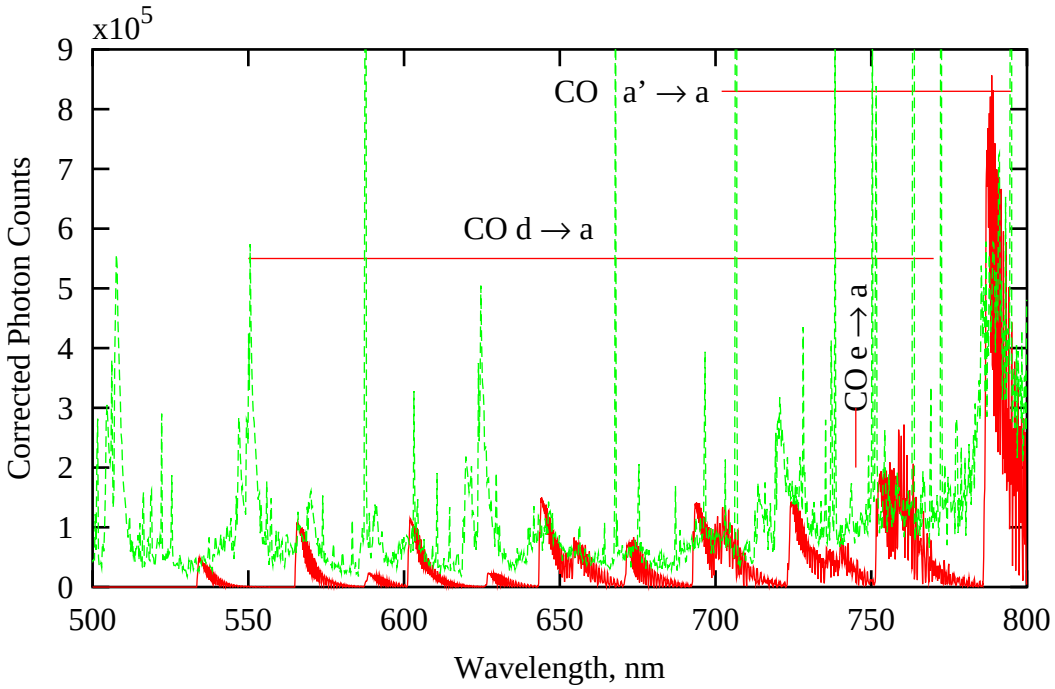


Figure 4-2.4a: Comparison of the simulated spectrum (red, solid) of triplet-triplet transitions (indicated on plot) to the spectrum (green, dashed) observed in a plasma containing recombining HCO_2^+ ions. See Table 4-2.4 for additional information. Molecular emissions not simulated here are CO^+ $A \rightarrow X$. Atomic emissions not simulated are emissions in the base plasma; see Appendix B for specific identifications. Observed photon counts are corrected for the optical system transmission efficiency.

It is also very likely that the primary recombining ion is HCO_2^+. Inspection of Table 4-2.12 shows that Ar^+ is expected to react with CO_2 with half the frequency of reactions with H_2. But, once formed, the CO_2^+ will only recombine once for every 20 reactions with H_2. So, only 1-2% of the recombining ions are expected to be CO_2^+. The differences between the HCO_2^+ spectrum (see Figures 4-2.4) and the CO_2^+ spectrum support this assertion as well. In the CO_2^+ recombination, the CO $a \rightarrow X$ emissions are considerably more intense than the a', d and $e \rightarrow a$. Additionally,

Figure 4-2.4b

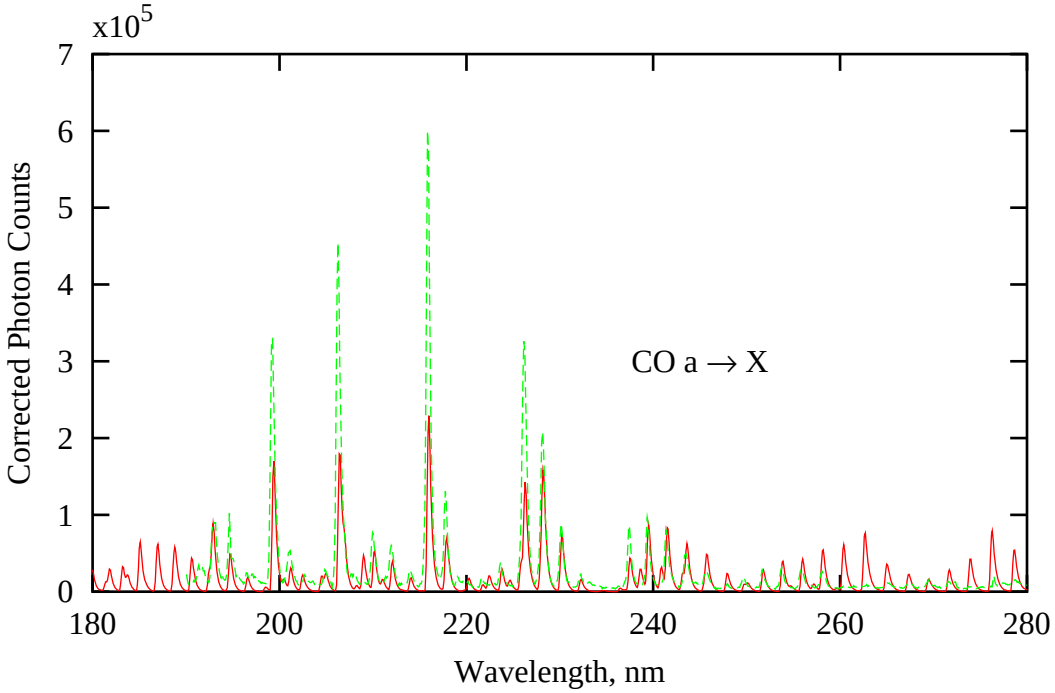


Figure 4-2.4b: Comparison of the simulated (red, solid) CO $a \rightarrow X$ cascade (from higher triplet levels) spectrum to the spectrum (green, dashed) observed in a plasma containing recombining HCO_2^+ ions. See Table 4-2.4 for additional information. The CO $a \rightarrow X$ emissions simulated here are due only to cascade from the higher electronic states a', d and e. Observed photon counts are corrected for the optical system transmission efficiency.

---

there is an intense OH emission in the HCO_2^+ spectrum that cannot be due to the recombination of CO_2^+.

As can be seen in Table 4-2.4, although emissions from several other states are observed, only the $a^3\Pi_r$ (v=3) state of CO and the $A^2\Pi_i$ (v=9) state of OH are energetically accessible in the recombination of ground state HCO_2^+. That high vibrational levels are observed indicates that the HCO_2^+ is in a vibrationally excited state. It is unlikely that a significant quantity of CO_2^+ is recombining, and this is

Figure 4-2.4c

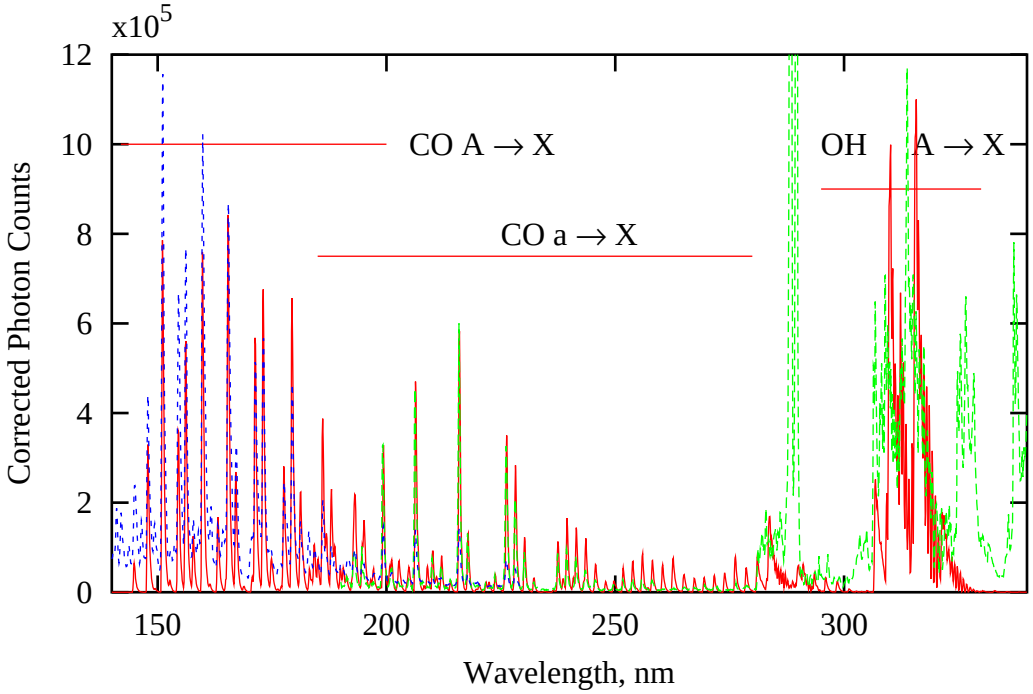


Figure 4-2.4c: Comparison of the 140-300 nm region of the simulated spectrum (red, solid) to the spectrum (UV-VIS: green, dashed and VUV: blue, dashed) observed in a plasma containing recombining HCO_2^+ ions. See Table 4-2.4 for additional information. Included in the simulation are cascade from the a', d and e states of CO and CO $a \rightarrow X$, CO $A \rightarrow X$ and OH $A \rightarrow X$ emissions from native (nascent) populations. The intense CO_2^+ $\tilde{B} \rightarrow \tilde{X}$ (280-300 nm) and CO_2^+ $\tilde{A} \rightarrow \tilde{X}$ (300-340 nm) emissions are not included in the simulation. Due to the strong deviation of the OH A state from Hund's case b, the observed $A \rightarrow X$ transition is not simulated well here. An anomalously high noise level in the VUV spectrum (blue, dashed) does not prevent assessment of the relative vibrational levels within the A state, but does create uncertainty about the population of the A state relative to the others. Observed photon counts are corrected for the optical system transmission efficiency.

Table 4-2.5

Emissions from a plasma containing recombining CS_2^+ ions

Recombination products [exothermicity,[a] eV]:

S + CS [5.51]; C + S_2 [2.53]

Non-destructive charge transfer exothermicity[a,b] [5.51]

Product	CS	
State	$A^1\Pi$	$a^3\Pi_r$ [n]
τ, μs [a]	0.22	16000
v_{max}, v^*_{max} [b]	4, *k*	15, *k*
Rel. Pop. [c]	0.7	1[t]
Rot. Temp., K [d]	800 K	700 K
v [e]		
0	1	1
1	0.7	1
2	0.7	1
3	0.5	0.6
4	0.4	0.2
5	0.2	0.1

Reagent(s)[r] (cm^{-3}): CS_2, 1.4×10^{12} ; CO_2, 1.7×10^{14}

Attaching gas[r] (cm^{-3}): CH_3I, 8×10^{10}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

supported by the low intensity of the CO $a \rightarrow X$ emissions that would otherwise be more intense. In fact, it appears to be excited beyond what is available from the protonation of the CO_2. The greatest amount of excitation required above the total recombination energy available, including protonation, is that of the highest observed vibrational level in the A state (7100 cm^{-1}). The cause for this discrepancy is un-

Figure 4-2.5a

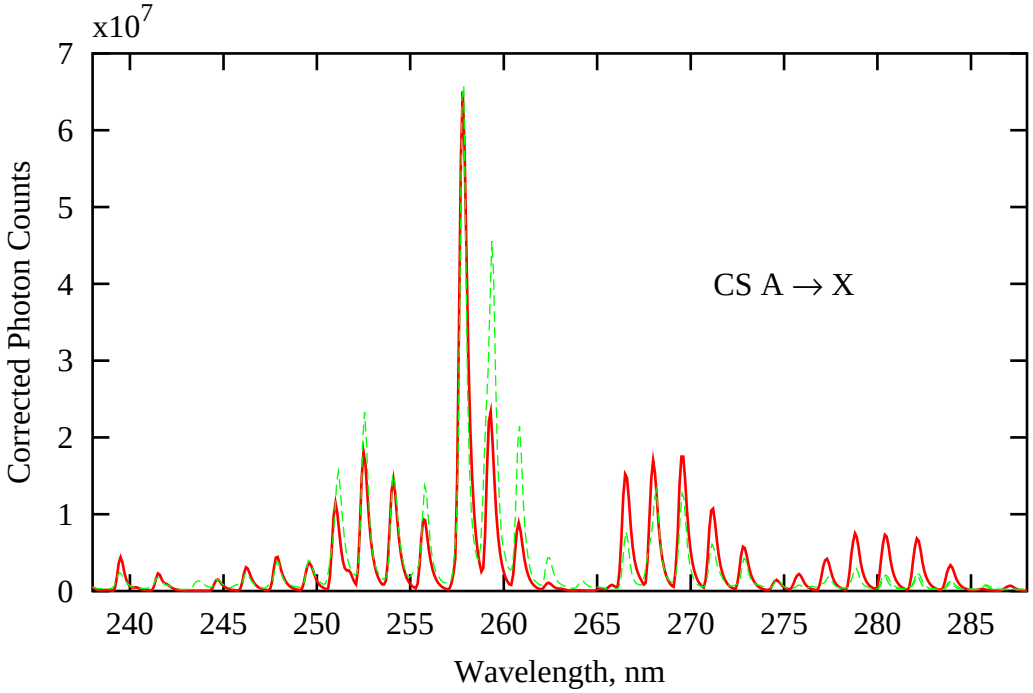


Figure 4-2.5a: Comparison of the simulated (red, solid) CS $A \rightarrow X$ spectrum to the spectrum (green, dashed) observed in a plasma containing recombining CS_2^+ ions. See Table 4-2.5 for additional information. Observed photon counts are corrected for the optical system transmission efficiency.

certain and indicates that intensities of individual features need to be observed as a function of attaching gas flow to determine their source.

In the only other known study of the recombination of HCO_2^+ [Johnsen 2000b], there is only a mention of the observation of the OH $A \rightarrow X$ transition, and relative vibrational populations are not reported.

4-2.5 Emissions from a plasma dominated by recombination of CS_2 ions

The $A \rightarrow X$ and $a \rightarrow X$ transitions of CS were observed in this system (Table 4-2.5 and Figure 4-2.5). All observed states in the CS_2^+ recombination are energetically

Figure 4-2.5b

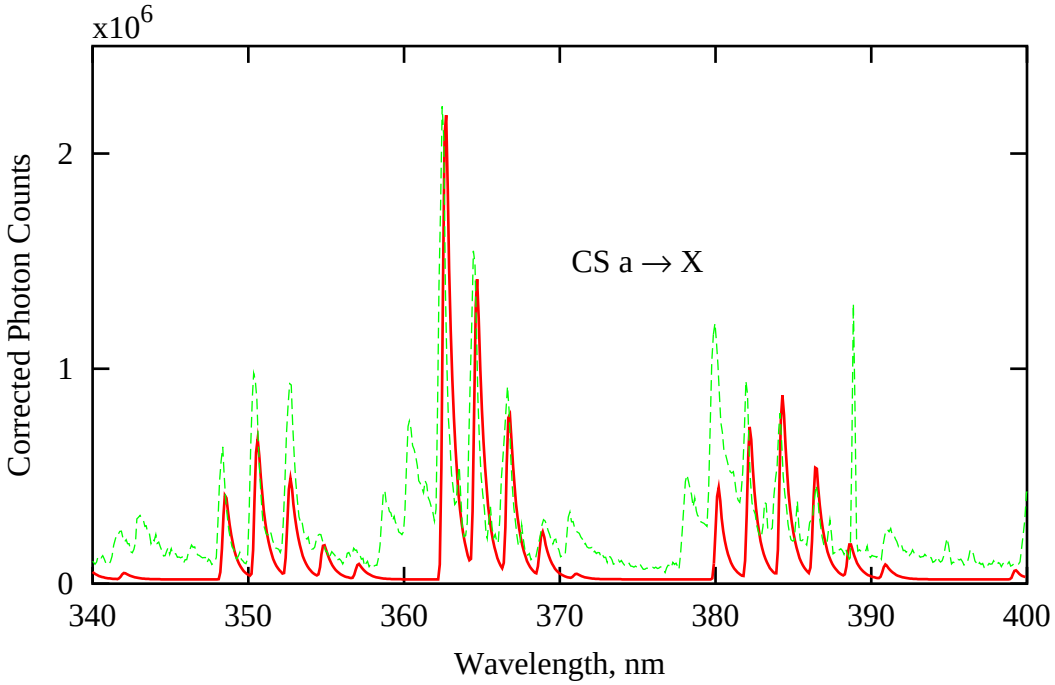


Figure 4-2.5b: Comparison of the simulated (red, solid) CS $a \rightarrow X$ spectrum to the spectrum (green, dashed) observed in a plasma containing recombining CS_2^+ ions. See Table 4-2.5 for additional information. The molecular peaks not included in the simulation are tentatively identified as CO^+ $A \rightarrow X$ emissions. The atomic line is a He emission from the base plasma; see Appendix B for further identification. Observed photon counts are corrected for the optical system transmission efficiency.

---

accessible. The v=5 level of the A state is not accessible by recombination energy, but has a Boltzmann probability of about 0.74 at 289 K.

CO_2 was added to facilitate vibrational relaxation of CS_2^+ ions prior to recombination. CO_2 has vibrational modes that are slightly more resonant with CS_2^+ than are modes of CS_2, and, in particular, the differences in energy between a few states are favorable (see also Tables 4-2.5 and 4-2.13a):

CS_2^+ Mode [ν, eV]	CO_2 Mode [ν, eV]	$\Delta\nu$, cm^{-1}

Table 4-2.6

Emissions from a plasma containing recombining NH_4^+ ions

Recombination products [exothermicity,[a] eV]:
NH + H_2 + H [0.37]
Non-destructive protonation exothermicity[a,b] [4.47]

Product	NH[h]	
State	$A^3\Pi_i$ [m]	$c^1\Pi$ [m]
τ, μs [a]	0.43	0.44
v_{max}, v^*_{max} [b]	j, 2	j, j
Rel. Pop. [c]	1	0.2
Rot. Temp., K [d]	3000 K	3000 K
v [e]		
0	1	1
1	0.3	

Reagent(s)[r] (cm^{-3}): NH_3, 3.6×10^{12} ; H_2, 2.3×10^{13}

Attaching gas[r] (cm^{-3}): SF_6, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a–r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

Symmetric Stretch [617]	Bend [667]	-50
Asymmetric Stretch [1200]	Symmetric Stretch [1333]	-133

Since the CS_2^+ frequencies are lower than the CO_2 frequencies, vibrational relaxation would have to be accompanied by translational and/or rotational relaxation.

The spectrum is at fairly high resolution and uncluttered by impurity emissions. The presence of the CO_2 results in the production of emissions from the recombination of CO_2^+ (see Figure 4-2.2), but these only overlap a few of the less intense CS $A \rightarrow X$ emissions, and are easily accounted for in the simulated spectrum. The only significant

Figure 4-2.6

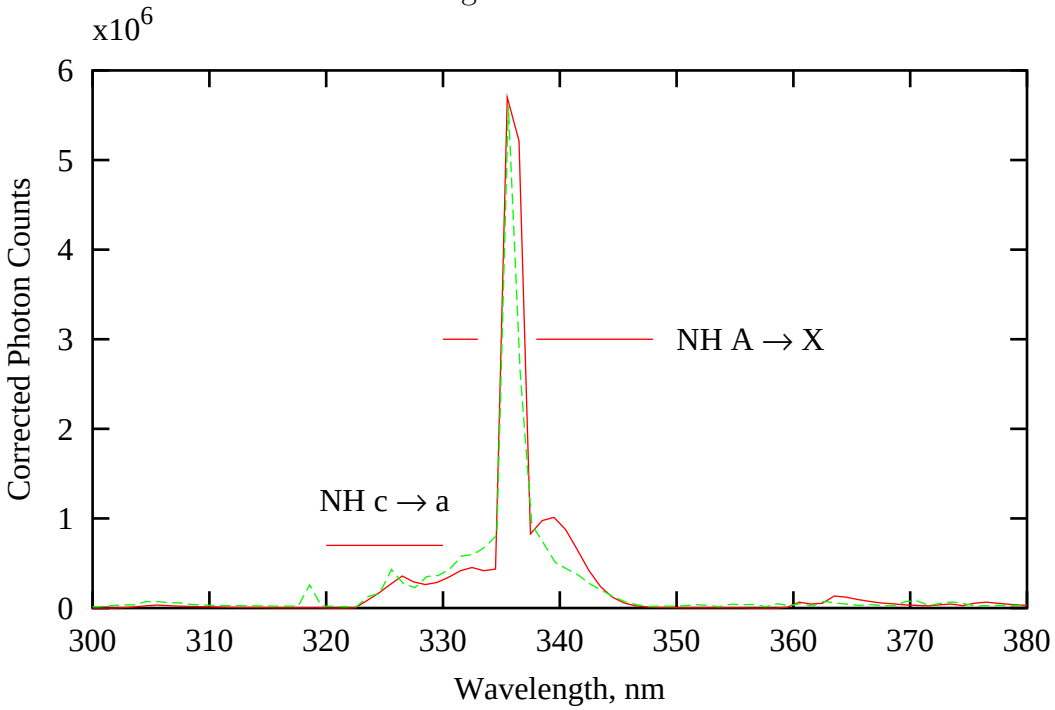


Figure 4-2.6: Comparison of the simulated (red, solid) spectrum to the spectrum (green, dashed) observed in a plasma containing recombining NH_4^+ ions. See Table 4-2.6 for additional information. The molecular species included in the simulation are the $c \rightarrow a$ (as visible here, roughly 320–335 nm) and $A \rightarrow X$ (roughly 330–350 nm) transitions of NH. Observed photon counts are corrected for the optical system transmission efficiency.

sources of uncertainty in the results arise from the models used during analysis. The calculated Franck Condon factors, since they are calculated from a Morse potential, are best near v=0.

4-2.6 Emissions from a plasma dominated by recombination of NH_4^+ ions

It is not energetically possible, in the recombination of ground state NH_4^+ to produce the observed excited states of NH (Figure 4-2.6 and Table 4-2.6), so either the

emissions are being produced by some process other than recombination, significant amounts of NH_3^+ are recombining or the NH_4^+ is vibrationally excited. If all emissions are due to NH_4^+ recombining, then it must have obtained energy at least 5900 cm^{-1} in excess of the protonation exothermicity in order to populate the v=0 level of the NH c state (this is the only vibrational level of the c state that is observed). Ar^+ reacted with NH_3 about once for every 3.6 reactions with H_2, and any ions formed from that reaction will react with H_2 about 4.6 times for every recombination. So, the NH_3^+ ion might account for as many as 4% of the recombinations. Data oon emissions from the recombination of NH_3^+ is not available in this study or from the literature for comparison. This system will require a more detailed investigation before emissions can be definitely identified as being products of the recombination of vibrationally excited NH_2^+.

Accurate simulation of the NH $A \rightarrow X$ transition was difficult (see Figure 4-2.6). It is likely that the algorithm used in RVESIM (see Chapter 6) does not treat NH well. It is also possible that the rotational distribution in the NH is nonthermal. Without a higher resolution spectrum, it is difficult to tell which of these is the case.

4-2.7 Emissions from a plasma dominated by recombination of HNO^+ ions

The combination of low resolution and overlapping features makes quantitation of species populations very difficult in this case (Figure 4-2.7). But, the electronic species listed are present (Table 4-2.7), even though the exact vibrational and electronic distributions are uncertain.

Values for the proton affinity of NO available in the literature do not specify whether the protonation occurs at N or O. An ab–initio calculation (Hartree–Fock with the 6-31+G* basis set in Spartan '02 by Wavefunction, Inc.) indicates an energy difference of about 0.09 eV with the higher energy state being protonation at O. In

Table 4-2.7

Emissions from a plasma containing recombining HNO^+ ions

Recombination products [exothermicity,[a] eV]:
H + NO [8.09]; O + NH [4.80]; N + OH [5.98]
Non-destructive protonation exothermicity[a,b,q] [1.13]

Product	NO		NH[h]	OH[h]
State	$A^2\Sigma^+$	$B^2\Pi_r$[m,n,p]	$A^3\Pi_i$	$A^2\Sigma^+$
τ, μs [a]	0.2	3.1	0.43	0.69
v_{max}, v^*_{max} [b]	9, 13	k, k	6, 6	5, 11
Rel. Pop. [c]	1	$\leq$0.2	0.7	0.3
Rot. Temp., K [d]	3000 K	—	4000 K	2000 K
v [e]				
0	1	(1)	1	1
1	0.7	(1)		0.3
2	0.5			0.1

Reagent(s)[r] (cm^{-3}): NO, 1.7×10^{13} ; H_2, 3×10^{13}

Attaching gas[r] (cm^{-3}): SF_6, 5.2×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

terms of vibrational and electronic states, that is a small amount of energy, and much less than the protonation exothermicity, regardless where the proton attaches.

Previous results [Johnsen 2000b] report the NO $A \rightarrow X$ bands as the only significant photoproduct of the recombination of HNO^+; however, they do not specify a vibrational distribution.

Figure 4-2.7

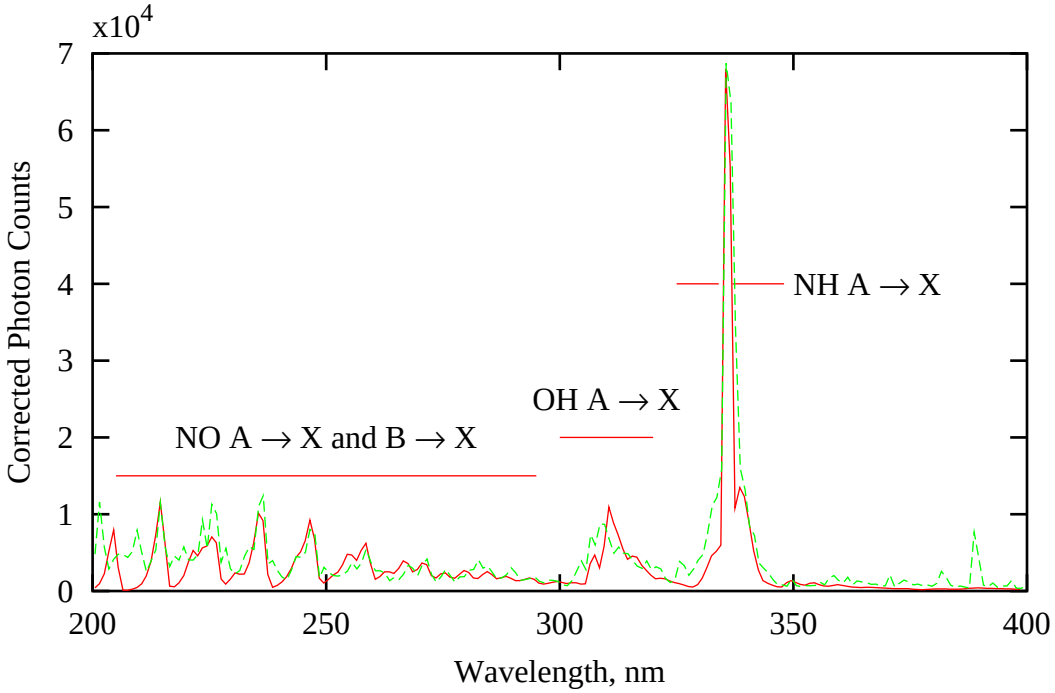


Figure 4-2.7: Comparison of the simulated (red, solid) spectrum to the spectrum (green, dashed) observed in a plasma containing recombining HNO^+ ions. See Table 4-2.7 for additional information. The molecular species included in the simulation are the NO $B \rightarrow X$ and $A \rightarrow X$ (overlapping and intense from 200–300 nm), the OH $A \rightarrow X$ transition (roughly 300–325 nm) and the NH $A \rightarrow X$ transition (roughly 325-350 nm). Observed photon counts are corrected for the optical system transmission efficiency.

4-2.8 Emissions from a plasma dominated by recombination of N_2OH^+ ions

The published data [Foley 1993] for this system were taken at the lower resolution of the data sets in this series, and therefore, quantitative information about the NH state could not be presented. Results from analysis of the higher resolution data in this study are given in Table 4-2.8.

Table 4-2.8

Emissions from a plasma containing recombining N_2OH^+ ions

Recombination products [exothermicity,[a] eV]:
NNO + H [7.64,[†] 7.90[‡]]; N_2 + OH [10.34,[†] 10.61[‡]]; NH + NO [5.91,[†] 6.17[‡]]
Non-destructive protonation exothermicity[a,b] [1.58,[†] 1.32[‡]]

Product	NH[h]		N_2	OH[h]
State	$A^3\Pi_i$ [m]	$c^1\Pi$ [m,n]	$B^3\Pi_g$ [m,o]	$A^2\Sigma^+$ [m]
τ, μs [a]	0.43	0.44	8	0.69
v_{max}, v^*_{max} [b] †	6, k	1, k	15, k	k, k
v_{max}, v^*_{max} [b] ‡	7, k	2, k	13, k	k, k
Rel. Pop. [c]	0.07	0.07	1	0.4
Rot. Temp., K [d]	4000 K	3000 K	300 K	2000 K
v [e]				
0	1	1	i	1
1	0.5	0.5	i	0.4
2	0.1		1[f]	0.1
3			0.4	
4			0.5	
5			0.4	
6			0.3	
7			0.2	
8			0.1	
9			0.1	

†$NNOH^+$; ‡$HNNO^+$

Reagent(s)[r] (cm^{-3}): N_2O 2.6×10^{13} ; H_2, 3×10^{13}

Attaching gas[r] (cm^{-3}): CH_3I, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

Figure 4-2.8

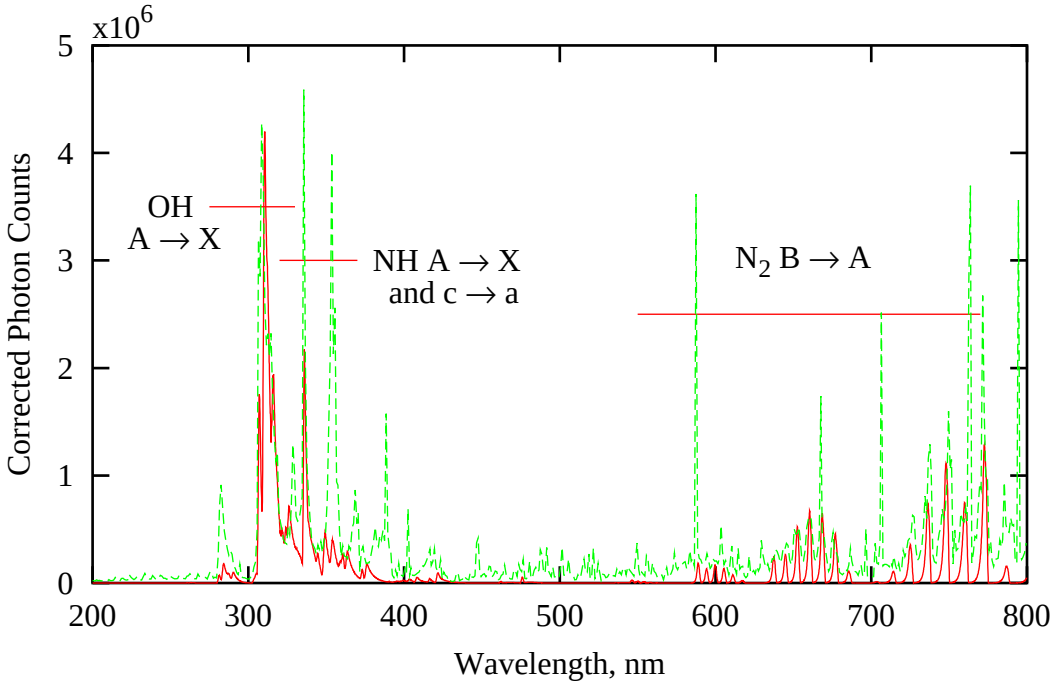


Figure 4-2.8: Comparison of the simulated (red, solid) spectrum to the spectrum (green, dashed) observed in a plasma containing recombining N_2OH^+ and HN_2O^+ ions. See Table 4-2.8 for additional information. The molecular species included in the simulation are the OH $A \rightarrow X$ (roughly 280–310), the NH $A \rightarrow X$ and less intense $c \rightarrow a$ (overlapping from about 325–350 nm) and the N_2 $B \rightarrow A$ transition (roughly 600–800 nm). Emissions from N_2O^+ (roughly 325–400 nm) are not included in the simulation. Other emissions are from the base plasma; see Appendix B for further identification. Observed photon counts are corrected for the optical system transmission efficiency.

All observed states are energetically accessible in the recombination of either protonated isomer. This system is one of the systems more likely to be vibrationally relaxed according to the approximate empirical formula by Lambert (see Table 4-2.12

and Section 3-6). A comparison of the output from the simulation to the observed spectrum is given in Figure 4-2.8.

The current data include previously unobserved emissions from the $c \rightarrow a$ transition of NH and the $B \rightarrow A$ transition of N_2. The N_2 $B \rightarrow A$ emissions have previously been observed for recombinations of N_2O^+ [Tsuji 2000], but those emissions have not previously been reported in the recombination of N_2OH^+. Since there is N_2O^+ present in the experimental spectrum, it is possible that some N_2O^+ ions are recombining. However, the N_2O^+ ion will react with H_2 six times for every recombination (see Table 4-2.12), Also, the NO $A \rightarrow X$, NO $B \rightarrow X$ and N_2 $C \rightarrow B$ emissions observed previously [Johnsen 2000b] in the recombination of N_2O^+ are either too weak for positive identification or not present at all in this study, indicating that only small amounts, if any, of N_2O^+ ions are recombining. The observed NH and N_2 emissions are thus very likely to origination from the recombination of protonated N_2O^+.

4-2.9 Emissions from a plasma dominated by recombination of H_3S^+ ions

The emissions in this spectrum are only identified tentatively (see Table 4-2.9). They appear to be from SH, but do not match known spectroscopic constants. One of the peaks has an appearance very similar to the $A \rightarrow X$ emission of NH, but, this is not possible because the pre–run mass spectrum (see Appendix D) does not indicate the presence of nitrogen in any form. Figure 4-2.9 presents, among other things, a comparison between the emission observed for this system and the NH emission observed in the NH_4^+ recombination.

If the emission is, indeed, SH, there are two possibilities. One is that the literature values for T_{00} are all too high. The lack of certainty in the value was mentioned in a recent paper [Resende 2001]. The lowest value mentioned there, also the one reported by NIST [2001a], is around 30662 cm^{-1}. The transition observed in these spectra corresponds, if it is the 0-0 transition, to a T_{00} of around 28600 cm^{-1}. The other

Table 4-2.9

Emissions from a plasma containing recombining H_3S^+ ions

Recombination products [exothermicity,[a] eV]:
H + H_2S [6.29]; SH + H_2 [6.89]
Non-destructive protonation exothermicity[a,b] [2.93]

Product	SH
State	[$\sim A^2\Sigma^{+\,p}$]
τ, μs [a]	—
v_{max}, v^*_{max} [b]	—
Rel. Pop. [c]	1
Rot. Temp., K [d]	[800 K]
v [e]	
0	[1]

Reagent(s)[r] (cm^{-3}): H_2S, 3.5×10^{12} ; H_2, 6.2×10^{13}

Attaching gas[r] (cm^{-3}): SF_6, 1×10^{11}

Relative electronic populations, relative vibrational populations and rotational temperatures are new data. All other values were taken or derived from other sources [Radzig 1985, NIST 2001]. [a−r] Notes for Tables 4-2.1 — 4-2.9 are in Table 4-2.10

possibility is that the observed transition is from v=1, but the Franck Condon factors (including those calculated by Resende, et al.) are very inaccurate, and such an identification is inconclusive (see Figure 4-2.9 for details). Since the Franck Condon factors are independent of T_e, the possibility of a misplaced T_e is possible. It could also be possible that this is some other transition of SH that is similar to the $A \rightarrow X$, but none has, so far, been identified. The issue could be resolved if high–resolution study of the system were performed. According to the Resende paper, experimental production of the $A \rightarrow X$ transition has been difficult. If the observed transition is

Figure 4-2.9

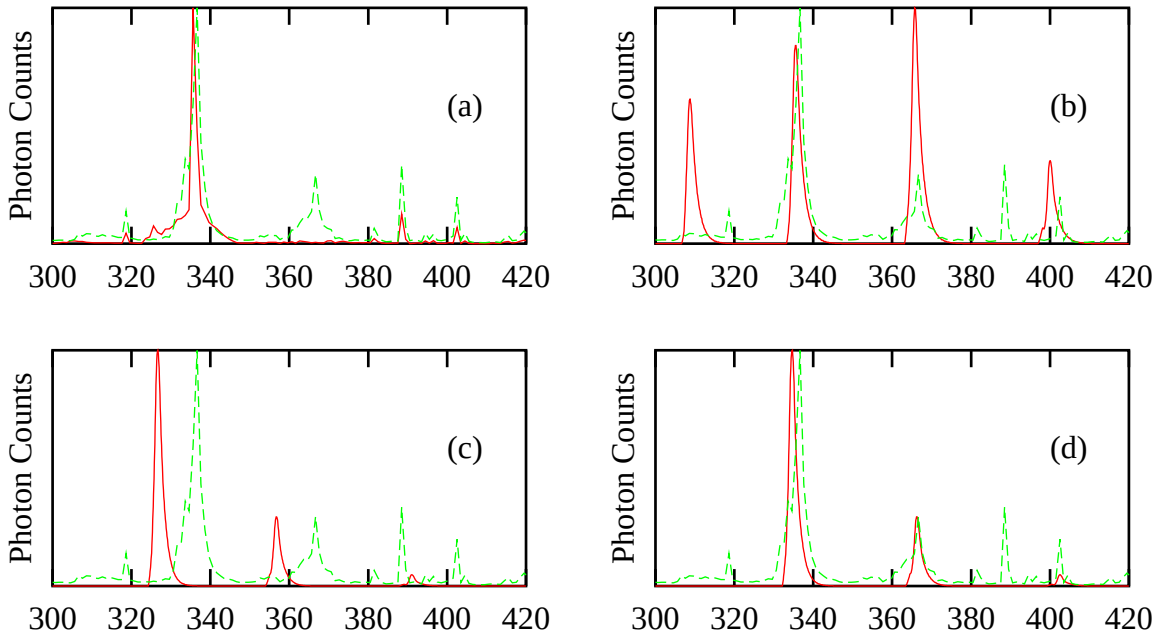


Figure 4-2.9: Illustration of the similarity to the SH $A \rightarrow X$ transition of the unknown emission in the spectrum from a plasma containing recombining SH_3^+ ions. (a) Although one of the major features in the H_2/H_2S system (green, dashed) is similar to the NH $A \rightarrow X$ transition (red, solid, taken from the NH_4^+ data presented earlier), they are not the same emission. (b) Transitions from the SH $A(v = 1) \rightarrow X$ state match energies for the two observed features, but the relative intensities are not correct. (c) A simulation of emissions from SH $A(v = 0) \rightarrow X$ matches the observed intensities, but the T_{00} appears to be shifted. (d) Appearance of the spectrum if the simulated SH $A \rightarrow X$ transition is calculated with Te changed from 31040 to 30310.

the SH $A \rightarrow X$, then it is produced quite easily in the flowing afterglow and could be studied in some detail for its spectroscopic significance.

Figure 4-2.10

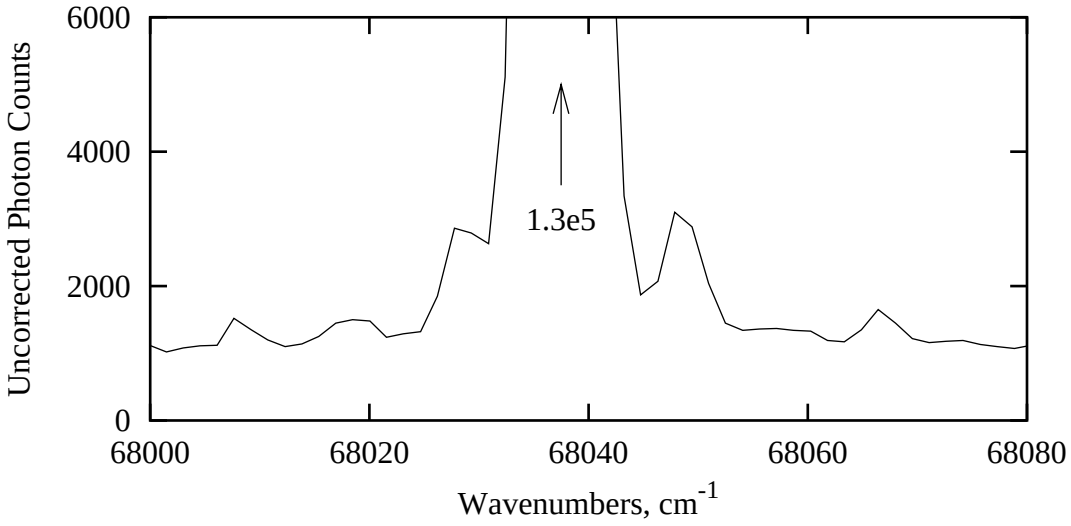


Figure 4-2.10: Detail of Xe emission from the recombination of XeH^+. The oscillatory behavior evident to either side of the main emission is probably due to the variation in the proximity of the H and Xe atoms when the Xe emits.

4-2.12 Emissions from a plasma dominated by recombination of XeH^+ ions

Reagents added at RP3, number density: Xe, 6.2×10^{13} *;* H_2, 1.6×10^{13} . *Attaching gas added at RP2, number density:* CH_3I, 1×10^{11} .

XeH^+ has been found to recombine fairly rapidly ($\alpha_e = 8.3 \times 18^{-8}$ $cm^{-3}s^{-1}$) [Le Padellec 1997]. The recombination is exothermic by 8.5 eV, an energy nearly resonant with the $^3P_1^0\,6s[3/2]_1^0$ state of atomic Xe [Radzig 1985]. Since that state has a radiative lifetime of about 280 ns [Radzig 1985], observation of that atomic emission as a photoproduct of recombination is likely and, has, indeed, been observed (see Figure 4-2.10). Given the short lifetime, the hydrogen atom is likely, in a small percentage of cases, to be close to the xenon atom at the time of de–excitation. This proximity of the two atoms should have an effect on the observed spectrum. Figure 4-2.10 clearly shows the oscillatory behavior typical of a bound → continuum transition.

This recombination could be very useful as an intensity calibration since a photon has to be emitted for every recombination.

Table 4-2.10

Notes for Tables 4-2.1 — 4-2.9

[a] Where possible, exothermicities were calculated exclusively from thermodynamic data found in the NIST WebBook [NIST 2001], with supplemental information taken, as needed, from the compilation by Radzig and Smirnov [Radzig 1985]. Radiative lifetimes come exclusively from the latter.

[b] v_{max} is the quantum number for the maximum vibrational level that can be populated by the recombination of a ground-state parent ion. The determination of v^*_{max} includes the protonation or charge transfer exothermicities, as appropriate, as well as the recombination energy. So, v^*_{max} is the quantum number for the maximum vibrational level that can be populated by the recombination of a parent ion in an excited state assuming that the excess energy is due only to the charge or proton transfer.

[c] The relative population of each electronic state is based on the sum of photons observed to originate from that state. Implicit in this assumption is that any given upper state decays exclusively via the observed emissions. This assumption is necessary because state-to-state transition probabilities are generally not available for molecules (diatomic or otherwise). See also notes *d*, *i* and *j*. For emitting states with long radiative lifetimes, see also notes *s* and *t*.

[d] Rotational temperatures are based on results from RVESIM (see Chapter 6), which uses a fairly simple algorithm. States that do not conform well to Hund's cases a and b, or that exhibit strong splitting, will not be treated well by the program. Electronic states that represent two or more near-degenerate variants (Π states for systems with an odd number of electrons, for example) for which only one set of

spectroscopic constants is available will also be treated poorly by RVESIM. In all cases, thermal distributions are good to about 50% of the significant digit. Note that rotational distributions affect relative vibrational and electronic populations.

[e] Relative vibrational populations are based on Franck-Condon factors calculated from a Morse potential. The vibrational potentials of many diatomic species are approximated fairly well by a Morse potential, but few are truly Morse potentials. Vibrational populations will be most reliable at lower vibrational quanta (v) where the Morse potential is most reliable. See also notes *h* and *i*.

[f] Due to a rapidly changing system efficiency in the spectral region from about 750 nm up, the populations of the following states might be overestimated: CO d, v=1 and a′, v=5; N_2 B, v=2.

[g] Emissions from the *e* state of CO are weak, if present, and obscured by emissions from the *d* and *a′* states in all spectra considered here. So, it is not possible to determine a rotational temperature for the *e* state.

[h] Either the *A* states of OH and NH have non-thermal rotational distributions or they deviate strongly from the standard model used in RVESIM (see Chapter 6). See also note *d*.

[i] Either there are no emissions from this state within the available spectral range or the emissions have very small transition probabilities.

[j] This state is not energetically accessible, but it is observed.

[k] There is sufficient energy to either vibrationally dissociate this state or to populate it at v≥20 (calculated using a Morse potential).

[l] Radiative lifetimes for the following electronic states of CO are available only for single vibrational states: *a′*, v=4; *d*, v=1; *e*, v=4.

[m] This emission is partly or totally obscured by another emission.

[n] This emission or set of emissions were of sufficiently low intensity as to make positive identification and/or quantification difficult.

[o] One or more intense emissions from this state occur outside the available spectral range.

[p] There are uncertainties about the spectroscopic characteristics of the SH A state [Resende 2001], and this identification is uncertain. See text.

[q] Values for protonation at different positions in this molecule are not available (as they are for CO and NNO, for example).

[r] Reagent gases were added through RP3. Attaching gases were added through RP2.

[s] The emissions from this state are partly or wholly due to cascade from the higher, triplet, electronic states of CO. This observation has been used to set an upper limit for the time CO molecules in the a state spend in viewing range of the spectrometers. The upper limit assumes that: (1) the higher triplet states (d, a' and e) will all have radiated within the viewing region; (2) that the $a \rightarrow X$ emissions are collected starting from the moment of the formation of the a state; (3) the simulation algorithm employed in RVESIM and the corrections for optical system transmission efficiency are sufficiently accurate. Based on these assumptions, the maximum time the a state was within the viewing region is about 0.3, 1 and 0.7 ms, respectively, for the HCO^+, CO_2^+ and HCO_2^+ systems.

[t] CS a state populations are based on the assumption that CS molecules in the a state were formed within the viewing region of the spectrometers and that they stayed within that region for about 2 ms.

Table 4-2.11

Table 4-2.11: The energies required to produce observed emissions directly from neutral, unprotonated parent molecules (that is, not via recombination). The first column contains the parent molecule. The second column contains one or more ground state fragmentation paths, with required energy for the production of the indicated vibronic state. In the third column are listed the excited states observed for that system with total energy (including the energy from the second column) required for the indicated excitation. Letters in the third column indicate excited electronic states and numbers in parentheses indicate the vibrational level for which the energy was calculated. Energies were calculated for the zeroth vibrational level and the highest observed vibrational level (out of all spectra) for observed electronic states. All energies are given in eV and enclosed in square brackets.

Table 4-2.11

Energies Required to Produce Observed Emissions

From Neutral Parent (Reagent) Molecules

Parent	Children [Required Energy, eV]				
	To Ground–State	With Indicated Excitation			
Processes Involving Ions:					
CH_4	$CH + H^+ + H_2$ [22.8]	CH:	A(0) [25.8];	A(2) [26.5];	B (0) [26.1]
CO_2	$O^+ + CO$ [19.1]	CO:	$a(0)$ [25.3];	$a(4)$ [26.1];	
			$A(0)$ [27.3];	$A(5)$ [28.2]	
CS_2	$S^+ + CS$ [14.9]	CS:	$a(0)$ [18.4];	$a(4)$ [19.0];	
			$A(0)$ [19.8];	$A(5)$ [20.4]	
H_2S	$H^+ + SH$ [17.5]	SH:	$A(0)$ [21.5]		
NH_3	$NH + H_2^+$ [19.8]	NH:	$A(0)$ [23.7];	$A(1)$ [24.1];	$c(0)$ [25.4]
N_2O	$N_2 + O^+$ [15.4]	N_2:	$B(0)$ [22.8];	$B(9)$ [24.6]	
	$N^+ + NO$ [19.5]	NO:	$A(0)$ [25.1];	$A(2)$ [25.7];	$B(0)$ [25.3]
Neutral Processes:					
CH_4	$CH + H + H_2$ [9.2]	CH:	$A(0)$ [12.3];	$A(2)$ [12.9];	$B(0)$ [12.5]
CO_2	$O + CO$ [5.5]	CO:	$a(0)$ [11.7];	$a(4)$ [12.5];	
			$A(0)$ [13.7];	$A(5)$ [14.5]	
CS_2	$S + CS$ [4.6]	CO:	$a(0)$ [8.1];	$a(4)$ [8.6];	
			$A(0)$ [9.5];	$A(5)$ [10.1]	
H_2S	$H + SH$ [3.9]	SH:	$A(0)$ [7.9]		
NH_3	$NH + H_2$ [4.4]	NH:	$A(0)$ [8.3];	$A(1)$ [8.6];	$c(0)$ [10.0]
N_2O	$N_2 + O$ [1.7]	N_2:	$B(0)$ [9.2];	$B(9)$ [11.0]	
	$N + NO$ [5.0]	NO:	$A(0)$ [10.6];	$A(2)$ [11.2];	$B(0)$ [10.7]

See description on previous page

Table 4-2.12

Table 4-2.12: Comparisons of relative frequencies for several important processes occuring in the reactive flow. For definitions of all comparisons, f, see Section 3-7. Also included in the table are results for use of the empirical equation for the number of collisions, $Z_{V,T}$, required to relax a gas by vibrational–translational energy transfer (see Equation 3-7 and all of Section 3-6). The value of the lowest vibrational frequency for the ion, ν_{min}, is also given. The letter following ν_{min}, indicates the source of the number as NIST [NIST 2001], Herzberg [Herzberg 1991] or Calculation using Hartree–Fock self–consistent field theory [Wavefunction, Inc. 2001]. The particular basis set varied according to the complexity of the ion, but was always part of the 6-31(G) series of basis sets.

Keep in mind the limitations of the empirical form of $Z_{V,T}$ (Equation 3-7, also below). It assumes pure, neutral gas and describes the average number of collisions needed to relax the least energetic mode by a single vibrational quantum. The number is still useful, though. It provides us with a best–case scenario. If the number of collisions prior to recombination is less than or even comparable to $Z_{V,T}$ then there is little chance that a significant number of the recombining ions are vibrationally relaxed.

Information regarding reagent concentrations and, where available, recombination coefficients, can be found at the beginning of the section for that reactive system (Sections 4-1.1-10).

The following definitions are also in Sections 3-7 and 3-8.

$$Z_{V,T} = \mathrm{e}^{C\nu_{min}} \qquad 3\text{-}7$$

$$f_{\nu,XYZ} = \frac{Z_{XYZ^+,XYZ}}{Rate_{XYZ^+,e^-}} \qquad 3\text{-}10$$

$$f_{\nu,He} = \frac{Z_{XYZ^+,He}}{Rate_{XYZ^+,e^-}} \qquad 3\text{-}11$$

$$f_{R,Ar^+} = \frac{Rate_{Ar^+,H_2}}{Rate_{Ar^+,XYZ}} \qquad f_{R,XYZ^+} = \frac{Rate_{XYZ^+,H_2}}{Rate_{XYZ^+,e^-}} \qquad 3\text{-}12$$

Table 4-2.12

Relative Frequencies for Selected Processes

Reactive System	Recombining Ion	ν_{min} (cm^{-1})	$Z_{V,T}$ (collisions)	$f_{\nu,XYZ}$	$f_{\nu,He}$	f_{Ar^+}	f_{XYZ^+}
CO_2	CO_2^+	462 (H)	2600	7.0	5400		
CO_2/H_2	HCO_2^+	598 (C)	140	5.1	7900	2.1	20
CH_4/H_2	CH_5^+	605 (C)	150	0.1	2500	4.7	4.6
$(CH_3)_2OH^+/H_2$	HCO_2^+	144 (C)	3	0.2	2500	4.7	4.0
CO/H_2	HCO^+	983 (C)	3500	1.8	2.7×10^4	33	2×10^4
	COH^+	54 (C)	1.6				
CS_2	CS_2^+	305 (H)	180	11	5400		
The collision partner for CS_2^+ is CO_2.							
H_2O/H_2	H_3O^+	526 (N)	79	373	2700	15	20
H_2S/H_2	H_3S^+	1033 (N)	5300	0.3	7300	13	12
NH_3/H_2	NH_4^+	1447 (N)	1.6×10^5	0.1	1900	3.6	4.6
NO/H_2	HNO^+	1055 (C)	6400	2.9	1.4×10^4	1.3	3.0×10^4
	NOH^+	1301 (C)	4.9×10^4				
N_2O/H_2	N_2OH^+	475 (C)	51	2.1	6400	4.3	6.0
O_2/H_2	O_2H^+	1307 (C)	5.1×10^4	0.09	5400	84	4.6
OCS	OCS^+	419 (N)	1200	0.01	5400		
OCS/H_2	$OCSH^+$	368 (C)	21	0.01	5400	80	4.2
	$HOCS^+$	410 (C)	30				
Xe/H_2	XeH^+	2180 (C)	7.2×10^7	15	3.4×10^4	—	—

See the description on the previous page.

Tables 4-2.13a and 4-2.13b

Tables 4-2.13a and 4-2.13b: ν_{HF} values were calculated in Spartan '02 [Wavefunction, Inc 2001] using Hartree Fock (HF) self–consistent field (SCF) at the 6-31G* (4-2.13a) and 6-31+G* (4-2.13b) levels of theory. Experimental values are from NIST [NIST 2001] except starred (*) values [Herzberg 1991]. Values for infrared inactive modes are italicized and underlined. [a] The $HOCO^+$ molecule is bent in the ground state (according to the HF calculation): the OCO portion of the molecule is linear, but the hydrogen is angled with respect to the other three atoms. In the OCO $\parallel$ bend mode, the OCO portion of the molecule bends in the plane formed by the angled hydrogen. Similarly, in the OCO $\perp$ bend mode, the OCO portion is bending perpendicular to the plane formed by the hydrogen. $HNNO^+$ and $NNOH^+$ have similar structures, with analogous designations. [b] The $\parallel$ and $\perp$ designations are with respect to the nodal plane of the highest occupied molecular orbital.

Table 4-2.13a

Normal–Mode Frequencies (cm^{-1}) for Selected Molecules and Ions

CO_2

Mode	ν_{HF}	$\nu_{Exp.}$
Bend	746.34	667
Symm. Str	*1517.62*	*1333*
Asym. Str	2583.29	2349

CO_2^+

Bend	426.8	462.6-719.7
Symm. Str	*1355.5*	*1244.3*
Asym. Str	961.37	1422

CS_2

Mode	ν_{HF}	$\nu_{Exp.}$
Bend	445.49	396*
Symm. Str	*724.94*	*657.98**
Asym. Str	1582.9	1535.4*

HCO_2^+

Mode	ν_{HF}	$\nu_{Exp.}$
OCO ∥ Bend[a]	598.28	—
OCO ⊥ Bend[a]	660.828	—
HOC Bend	1107.99	—
Symm. OCO Stretch	1369.13	1500
Asym. OCO Stretch	2696.08	2300
OH Stretch	3736.23	3375.37

CS_2^+

Mode	ν_{HF}	$\nu_{Exp.}$
Bend ∥[b]	310.21	328.3
Bend ⊥[b]	397.25	333.8
Symm. Str	*663.04*	*617*
Asym. Str	1152.66	1200

See the description on the previous page.

Table 4-2.13b

Normal–Mode Frequencies for Selected Molecules and Ions

$HNNO^+$

Mode	ν_{HF}	$\nu_{Exp.}$
ONN ∥ Bend	713.69	–
ONN ⊥ Bend	698.34	–
HNN Bend	472.23	–
Symm. ONN Stretch	1632.25	–
Asym. ONN Stretch	2639.47	–
NH Stretch	3621.14	–

$NNOH^+$

Mode	ν_{HF}	$\nu_{Exp.}$
NNO ∥ Bend	474.52	–
NNO ⊥ Bend	532.94	–
HON Bend	1534.34	–
Symm. NNO Stretch	1077.36	–
Asym. NNO Stretch	2813.25	–
OH Stretch	3744.05	3330.91

N_2O

Mode	ν_{HF}	$\nu_{Exp.}$
Bend	669.45	588.78*
Symm. Str	1376.83	1284.9*
Asym. Str	2623.09	2223.76*

See the description preceding Table 4-2.13a.

4-3 Other (Non–Recombination) Results

The presence of emissions (see Section 4-1) that behave, to some degree, as though they arise from electron–ion recombination, but that are not energetically accessible products of the process, led to more detailed investigations of the experimental procedure. In particular, the sequence of reactions that were assumed to culminate in dissociative recombination have been considered. These investigations have produced a number of interesting, and often unexpected, results.

The primary focus of these investigations is the flowing plasma, called here the *base plasma*, into which the other reactants were injected. Section 4-3.1 presents a detailed description of the species present in the base plasma, including probable production mechanisms. Some of the more interesting species observed in the spectra were electronically excited states of neutral dihelium and of Ar^+, also called Ar(II). In Section 4-3.2, the effect of addition of an attaching gas on the base plasma is discussed. It is found that the addition of an attaching gas quenches nearly all the emissions. The exceptions are a few excited states of atomic helium. A possible explanation for this behavior is presented.

4-3.1 Characterization of the He and He–Ar Plasmas

Spectra were taken of the base plasma before and after the addition of argon. That is to say, one set of spectra were taken in a pure He afterglow, where only Reactions 2-3 and 2-4 could occur and another set of spectra were taken after the addition of Ar, where 2-5 could also occur. Most of the emissions in these spectra have been identified, and a few of the identifications are surprising. Details of the contents of the plasmas before and after the addition of argon can be found in a series of graphs in Appendix B. Relative populations for the various species have been determined using the atomic simulation feature in RVESIM (rotational vibrational

Table 4-3.1

Table 4-3.1: Atomic helium transitions observed in the base plasma and the relative upper state populations implied by the emission intensities. Upper state populations are corrected for J–value statistics (see text); values presented are relative to each other with the most populated state assigned a value of one. Underlined transitions survived addition of the attaching gas (see section 4-2.2). Legend: ns and np are the orbital occupied by the electron that changes during the transition. nm is the wavelength at which the transtion occurs. Pop. is the relative population for that state. Numerical entries of the form $X(Y)$ are abbreviations for $X \times 10^{Y}$.

Table 4-3.1

Observed Atomic Helium Transitions and Implied Relative Upper State Populations

Transitions to:	$2p\ ^3P^0_{0,1,2}$			From Series:		$ns\ ^3S_1$		
ns	10s	8s	6s	5s	4s	3s		
nm	356.3	365.2	386.8	412.1	471.3	706.5		
Pop.	6.6(-4)	9.4(-4)	1.2(-3)	1.1(-2)	4.1(-2)	5.8(-1)		

Transitions to:	$2s\ ^3S_1$			From Series:		$np\ ^3P^0_{0,1,2}$		
np	10p	9p	8p	7p	6p	5p	4p	3p
nm	267.7	269.6	272.3	276.4	282.9	294.5	318.8	388.9
Pop.	4.8(-3)	6.9(-3)	8.4(-3)	1.9(-2)	5.4(-2)	4.6(-2)	4.9(-2)	1.5(-1)

Transitions to:	$2p\ ^3P^0_{0,1,2}$			From Series:		$nd\ ^3D_{1,2,3}$		
nd	10d	9d	8d	7d	6d	5d	4d	3d
nm	355.4	358.7	363.4	370.5	382.0	402.6	447.2	587.6
Pop.	1.9(-3)	1.8(-3)	3.2(-3)	5.5(-3)	8.3(-3)	9.2(-2)	1.1(-1)	5.5(-1)

Transitions to:	$2p\ ^1P^0_1$			From Series:		$ns\ ^1S_0$		
ns	8s	7s	6s	5s	4s	3s		
nm	393.6	402.4	416.9	443.8	504.8	728.1		
Pop.	7.9(-3)	1.7(-2)	1.4(-2)	2.5(-2)	1.5(-1)	2.8(-1)		

Transitions to:	$2s\ ^1S_0$			From Series:		$np\ ^1P^0_1$		
np	10p	9p	8p	7p	6p	5p	4p	3p
nm	323.1	325.8	329.7	335.5	344.8	361.4	396.5	501.6
Pop.	1.3(-3)	2.4(-3)	2.4(-3)	1.4(-3)	6.2(-3)	9.8(-3)	2.6(-2)	4.2(-1)

Transitions to:	$2p\ ^1P^0_1$			From Series:		$nd\ ^1D_2$		
nd	10d	9d	8d	7d	6d	5d	4d	3d
nm	383.4	387.2	392.7	400.9	414.4	438.8	492.2	667.8
Pop.	1.2(-3)	2.8(-3)	4.3(-3)	4.4(-2)	4.7(-2)	5.1(-2)	2.5(-1)	1.0(0)

See the description on the previous page.

electronic simulation, see Chapter 6). These are graphed and discussed below. Mass spectra for these plasmas are also included in Appendix B.

The presence of excited states of atomic helium in the μ-wave discharge is not surprising. It is, however, somewhat surprising to find so many excited states present in the flow downstream at RP3. Indeed, all singly excited states of He(I) [NIST 2001b] that are sufficiently intense and that fall within the range of the optical spectrometers are observed. The helium is excited by collisions with high energy electrons in the microwave discharge. Some of the photons, then, are carried down the flowtube by absorption and re–emission. There is also the possibility that some of the light is simply being reflected along the interior wall of the flowtube. But, considering the results presented in the next section (4-3.2), internal reflection seems an unlikely explanation for the observed radiation. Many of the observed emissions correspond to decay to one of the helium metastable states. One of them, the $2s^1S_0$ state, is metastable because transition to the ground state is forbidden under the selection rule $\Delta\ell = \pm 1$.[†] The other, the $2s^3S_1$ state, is metastable for the same reason and also because the decay to the ground state would violate the somewhat less rigorous selection rule $\Delta S = 0$.[†] A complete list of the observed atomic helium transitions is in Table 4-3.1 and also in Appendix B.

Included in the list of transitions in Table 4-3.1 are the relative populations of the upper states as implied by the observed transition intensities. In these cases, and throughout this document, observed state populations have been corrected for J-value statistics, where J is the overall angular momentum quantum number for a given electronic state. The degeneracy for a state in level J is $2J + 1$. Observed relative populations are divided by the relevant degeneracy.

[†] ℓ is the quantum number describing the angular momentum of the orbital occupied by the electron that changes orbitals during the transition and S is the sum of the spin quantum numbers for all the electrons in the atom.

In addition to excited states of atomic helium, a number of excited states of neutral dihelium, He_2 [NIST 2001a, Radzig 1985], are observed. These are identified on the series of spectra and associated keys in Appendix B. The emissions involve both singlet and triplet states of the dihelium, but emissions involving the triplet states are by far the more intense. Since transition probabilities for dihelium are not generally available, it it not possible to make statements about relative populations of the states. That is to say, the singlet states might well be populated more than the triplet states, but the triplet states might simply radiate more readily. It is not known whether the He_2 is produced in or downstream of the μ-wave source.

It is likely that these molecules are produced by three–body association of electronically excited helium atoms and neutral helium atoms. Experimental studies [Ludlum 1967] have shown that the metastable triplet dihelium ground state is produced by such a mechanism:

$$2He + He(2s\,^3S_1) \xrightarrow{\sim 2\times 10^{-34}\ cm^6/\#^2 s} He_2(2s\,^3\Sigma_u^+) + He + 1.96\ eV \quad 46$$

No studies have be conducted concerning similar production of electronically excited states of dihelium. However, it is certainly possible that similar processes might occur involving three–body interactions between some other excited state of helium and two other helium atoms that are likely to be in the ground state, but that might be excited as well. Such associations would be far more likely where the energy barrier is small. For example [Ginter 1970 & 1965b], two separated He atoms in states $1s2p\,^3P$ and $1s^2\,^1S$ correlate with the dihelium molecular states $2p\pi\,^3\Pi_g(b)$, $3d\sigma\,^3\Sigma_u^+(f)$ and $3d\pi\,^3\Pi_u(f)$. But, only the b state is likely to be populated significantly unless the activation energy necessary for the f states ($\sim$ 0.5 eV) is available:

$$2He + He(2p\,^3P) \xrightarrow{low\ E} He_2(2p\pi\,^3\Pi_g) + He + 2.51\ eV \quad 47$$

Excited states that could not be produced in this manner could perhaps be produced by electron–impact excitation of existing dihelium molecules or by the radiative de–excitation of dihelium molecules in higher electronic states. When the excited states

of dihelium were initially characterized [Brown 1971 & 1973, Ginter 1965a,b, 1966 & 1969, Gloersen 1965, Orth 1976], the states were produced by passing helium through a discharge, a procedure similar to that used to produce ionization in these studies. A third production mechanism, which is discussed in more detail in the next section (4-3.2) involves the bimolecular reaction of a helium atom in its ground state with another He atom in a singly excited state with $n \geq 3$ [Lichten 1974]:

$$He + He(n \geq 3) \rightarrow He_2^+ + e^- \qquad 48$$

It is very likely (see next section) that this mechanism is significant.

In situations where argon was added, one class of species found in the plasma, excited states of Ar^+, was not expected. Table 4-3.2 lists the emissions observed as well as the apparent relative populations of the upper states. Also listed are maximum populations of states that, if present, corresponded to transitions too faint to be considered significant. Only a narrow range of the possible excited states of Ar^+ [Bashkin 1975, NIST 2001b] are observed to be present in sufficient amounts for positive identification. Given the narrow range of states observed, it seems likely that the excitation is being caused by a reaction rather than by collisions with energetic electrons. Since argon is a noble gas, there are few species present in the tube that will excite an argon ion rather than neutralize it. But excited states of helium and dihelium are in abundant supply in the flowtube and are the likely sources of the excitation.

Which helium species, atomic or diatomic, is (or are) causing the excitation of the Ar^+ is uncertain. Energetically speaking, it is possible that metastable states of atomic helium are reacting with the Ar^+ in its ground state ($3s^2 3p^5\, ^2P^0$), exciting the Ar^+ and deexciting the helium. The energies are (see Figure 4-3.1): 20.616 eV and 19.82 eV for the singlet and triplet He metastables and a range of 19.22–19.76 eV for the excitation above ground state observed in the Ar^+. The main problem with this

Table 4-3.2

Relative Upper State Populations for Observed Ar^+ Transitions

Configuration and Term	**Level Energy**	**Relative Population***
$3s^2 3p^4[^3P]4p^1\ ^4P^0_{5/2}$	19.22	0.9
$3s^2 3p^4[^3P]4p^1\ ^4P^0_{3/2}$	19.26	1.0
$3s^2 3p^4[^3P]4p^1\ ^4P^0_{1/2}$	19.31	0.3
$3s^2 3p^4[^3P]4p^1\ ^4D^0_{7/2}$	19.49	0.7
$3s^2 3p^4[^3P]4p^1\ ^4D^0_{5/2}$	19.55	0.4
$3s^2 3p^4[^3P]4p^1\ ^4D^0_{3/2}$	19.61	0.2
$3s^2 3p^4[^3P]4p^1\ ^4D^0_{1/2}$	19.64	0.1
$3s^2 3p^4[^3P]4p^1\ ^2D^0_{5/2}$	19.68	0.2
$3s^2 3p^4[^3P]4p^1\ ^2D^0_{3/2}$	19.76	0.1
$3s^2 3p^4[^3P]4p^1\ ^2P^0_{1/2}$	19.80	(0.02)
$3s^2 3p^4[^3P]4p^1\ ^2P^0_{3/2}$	19.87	(0.003)
$3s^2 3p^4[^3P]4p^1\ ^4S^0_{3/2}$	19.97	(0.002)
$3s^2 3p^4[^3P]4p^1\ ^2S^0_{1/2}$	19.97	(0.02)
Other states through energy:	24.83	(< 0.005)

*Relative populations are corrected for J–value statistics (see Section 4-2.1). () Populations in parentheses are maximum values possible. Their inclusion does not necessarily imply that those states are present; the intensities are too weak for more certain quantification.

mechanism is that, while the energies are certainly similar, they are hardly resonant. So, it would be necessary that the substantial 0.06-1.40 eV excess of translational energy be carried away. The Ar^+ and the He could become translationally excited or

Figure 4-3.1

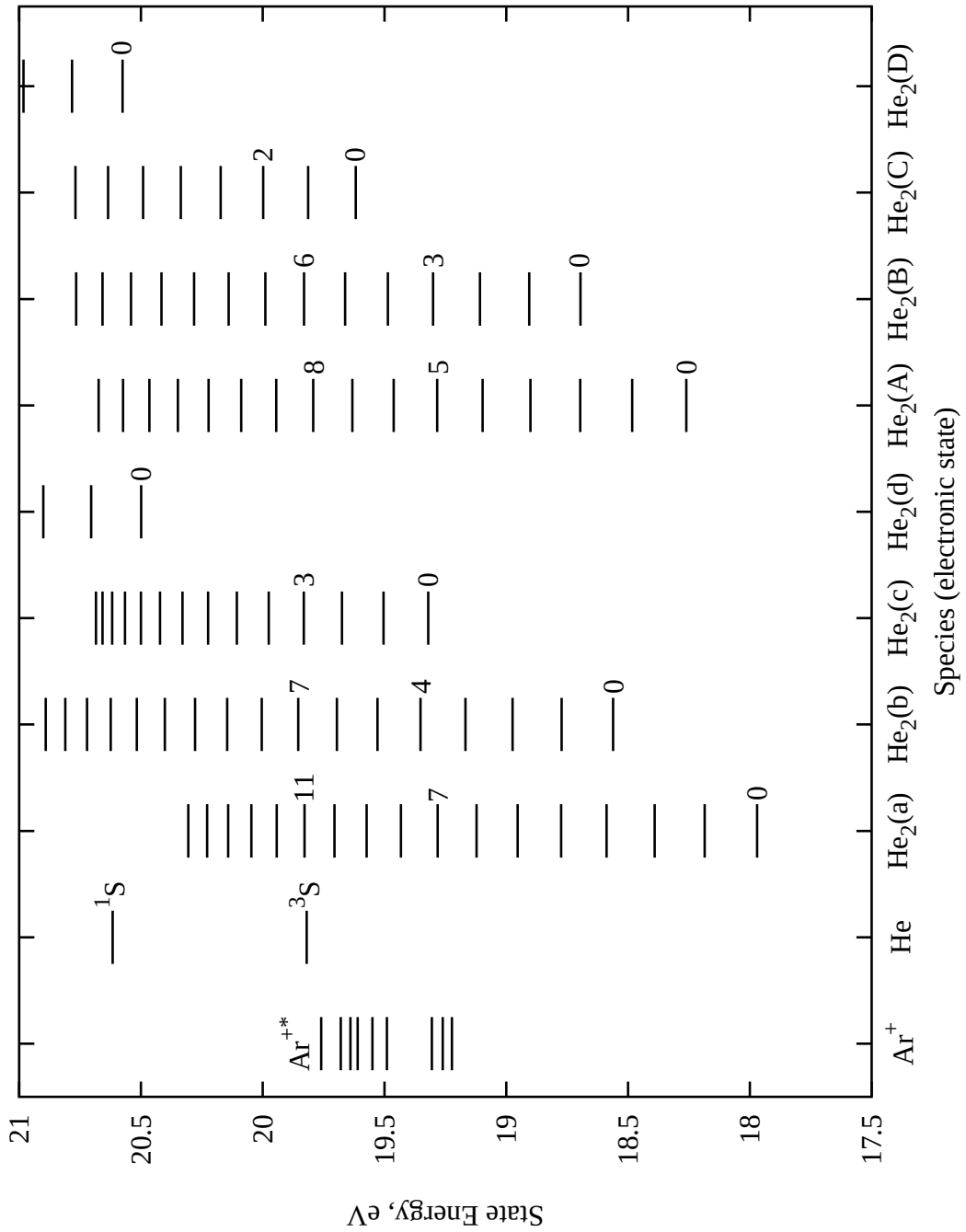


Figure 4-3.1: Comparison of upper–state energy levels for the observed transitions of Ar^+ to energies of species that could excite Ar^+ to the observed energies. Numbers beside diatomic levels are vibrational quanta. Vibrational energies assume Morse–like potentials and are derived from tabulated constants [NIST, 2001].

a third body could be involved, e. g.:

$$He(2s\,^1S_0,\ 2s^3S_1) + Ar^+(3s^2 3p^5\,^2P^0) + M \xrightarrow{possible,\ unlikely} He(1s^2\,^1S_0) + Ar(3s^2 3p^4[^3P]4p:\ ^4P^0,\ ^4D^0,\ ^2D^0) + M^* \quad 49$$

where M represents the third body. Without more detailed experimental or computational information, it is impossible to predict the probability of such a reaction. If, for example, the energy transfer occurs at a set of internuclear separations corresponding to a position on the repulsive curve between two ground state helium atoms (one of them having just entered the ground state as a result of the reaction; this is the ground state of He_2), then the reaction, and the large amount of translational excitation would be possible.

However, comparisons of the spectra taken before and after the addition of argon do not readily support Reaction 8. Intensities of emissions from atomic helium do not change significantly after argon is added to the flow. Instead, intensities of the dihelium emissions change (see Figure 4-3.2). So it is likely that He_2 is the source of the excitation in the argon ions:

$$He_2(excited) + Ar^+(3s^2 3p^5\,^2P^0) \xrightarrow{possible} 2He(1s^2\,^1S_0)^* + Ar(3s^2 3p^4[^3P]4p:\ ^4P^0,\ ^4D^0,\ ^2D^0)^* \quad 50$$

where the asterisk (*) indicates possible translational excitation. It is necessary, for the emissions observed here, that the dihelium be in some sort of excited state. The lowest vibrational levels of the $a\,2s\,^3\Sigma_u^+$, $b\,2p\,^3\Pi_g$, $A\,2s\,^1\Sigma_u^+$ and $B\,2p\,^1\Pi_g$ states, the observed destination states for dihelium emissions, are not energetic enough to excite the argon ion to the observed upper states. The minimum vibrational levels with sufficient energy, assuming Morse–potential behavior, are: a, 7; b, 4; A, 5; and B, 3. A non–thermal population would be required for population of these vibrational levels. But in the plasma environment, with excitation by energetic electrons, non–thermal

Figure 4-3.2

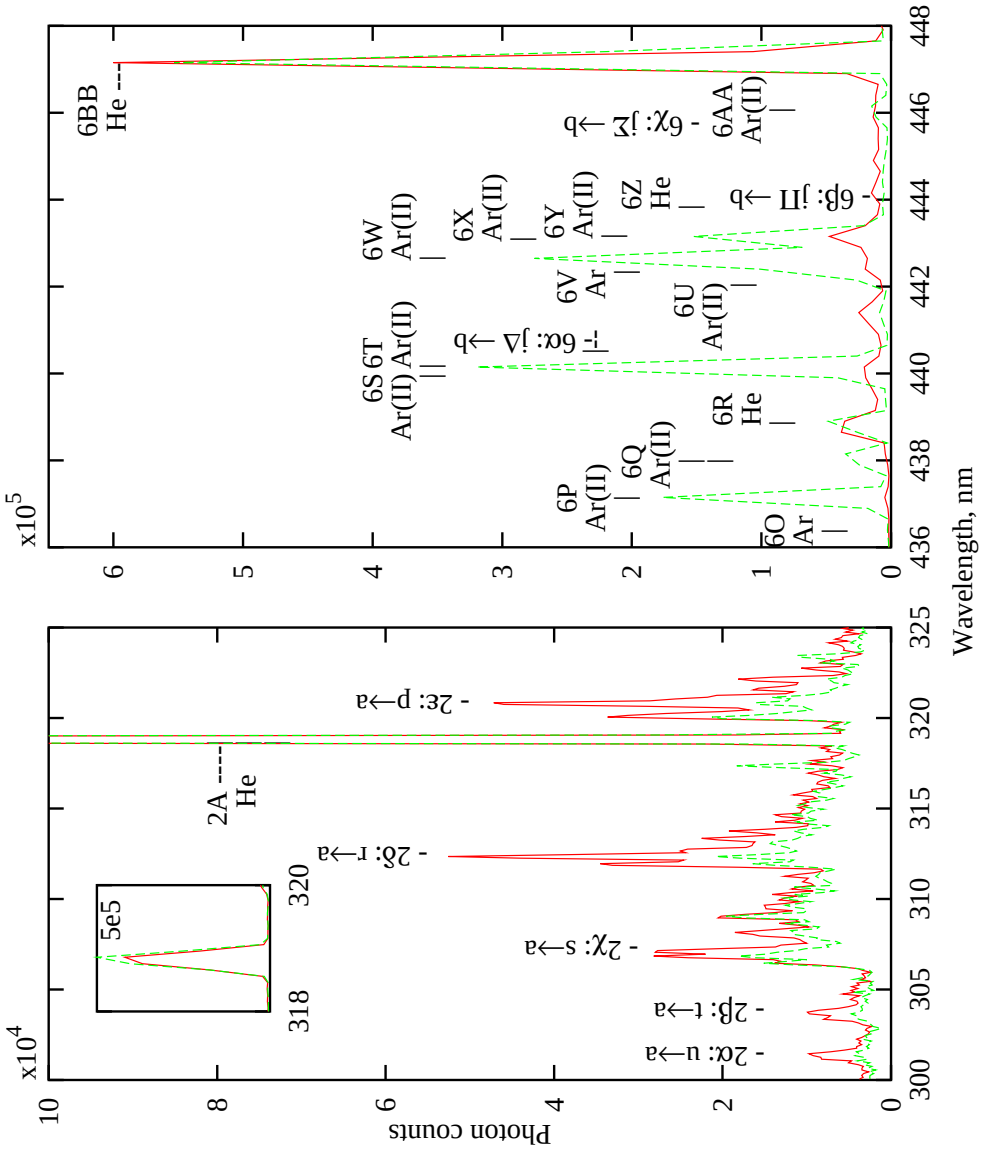


Figure 4-3.2: Comparison of spectra before (solid, red) and after (dashed, green) the addition of Ar. The wavelength regions were chosen because emissions from He, He_2 and Ar^+ are prominent features in them. Note that the intensities of the atomic He emissions (upper-case, un-rotated, alpha-numeric labels) do not change significantly while the intensities of the dihelium emissions (rotated labels with lower-case Greek and Roman letters) decrease, and, of course, the Ar^+ emissions appear. Identifications refer to the legend key found in Appendix B.

Table 4-3.3

Ar^+ Far VUV Emissions Expected if He_2 Is Source of Ar^+ Excitation

			Transition Wavelengths (nm)			
Ar^+ Upper State			Ground $[3/2]^\dagger$		Ground $[1/2]^\ddagger$	
Configuration	Term	Energy	nm	$A_(k,i)$	nm	$A_(k,i)$
$3s^2 3p^4\ (^3P)\ 4s^1$	$^2P_{3/2}$	17.140	72.336	$2.3\times10^{+01}$	73.093	$4.5\times10^{+00}$
$3s^2 3p^4\ (^3P)\ 4s^1$	$^2P_{1/2}$	17.266	71.809	$9.5\times10^{+00}$	72.555	$1.9\times10^{+01}$
$3s^2 3p^4\ (^3P)\ 3d^1$	$^2P_{1/2}$	17.942	69.104		69.794	
$3s^2 3p^4\ (^3P)\ 3d^1$	$^2P_{3/2}$	18.061	68.649		69.330	
$3s^2 3p^4\ (^1D)\ 4s^1$	$^2D_{3/2}$	18.427	67.286		67.940	*
$3s^2 3p^4\ (^1D)\ 4s^1$	$^2D_{5/2}$	18.454	67.185	*	—	
$3s^2 3p^4\ (^3P)\ 3d^1$	$^2D_{3/2}$	18.657	66.456	*	67.095	*
$3s^2 3p^4\ (^3P)\ 3d^1$	$^2D_{5/2}$	18.733	66.187	*	—	

$\dagger\ 3s^2 3p^5\ ^2P^0_{3/2}$ [0.00 eV]; $\ddagger\ 3s^2 3p^5\ ^2P^0_{1/2}$ [0.18 eV]

Table 4-3.3: List of emissions not observable with the current experimental apparatus that are expected when the Ar^+ is being excited by He_2 in the a, b, A and B states. Where transition probabilities ($A_{k,i}$) are reported by NIST, they are included here. Transitions without transition probabilites but that are reported by NIST are indicated by an asterisk (*). Transitions with neither $A_{k,i}$ nor an asterisk were calculated from Ar^+ energy levels as reported by NIST. Only transitions that satisfy the following selection rules are included: $\Delta\ell = \pm1$, $\Delta L = 0, \pm1$, $\Delta J = 0, \pm1$, $\Delta S = 0$.

vibrational populations are often observed, usually being produced as the result of decay from one or more higher electronic states. Any of the higher electronically excited states, themselves, have sufficient energy to excite the argon to the observed levels. In the case that any of these states is the reactive species, the excess energy

could be carried away by translational motion in the three bodies: the two ground state, neutral helium atoms and one excited state argon ion.

A comparison of the relevant energy levels (Figure 4-3.1) shows that the most isoergic species are the c and C states of He_2. Since transitions to these states are not observed in the pure He spectrum, it is not likely that they are the source. Of course, as stated earlier, the transition probabilities are not known, so the states could certainly be present if the transition probabilities are low. But, since emissions involving the a, b, A and B states of He_2 change so significantly upon addition of Ar, it seems likely that these states, perhaps in addition to others, are participants in Reaction 9.

If this is the case, then the population of Ar^+ states with energies less than 19.22 eV but greater than the T_0's for the a, b, A and B states of He_2 (17.97, 18.56, 18.26 and 18.70 eV, respectively) is expected. All of the electronically excited states of Ar^+ in this energy range radiate at wavelengths less than 100 nm, which is outside the range accessible to the equipment used in this study. Table 4-3.3 lists the wavelengths of transitions from Ar^+ states with the relevant energies that could be observed by the appropriate equipment if the ion is being excited by lower vibrational levels of the a, b, A and B states of He_2. Only those transitions with favorable selection rules are included in the table. Transition probabilities are given where available. Unless further information proves otherwise, it appears that the excited states of Ar^+ are being produced by reaction with neutral dihelium molecules.

4-3.2 Behavior of the Plasma with Attaching Gas Added

Although a spectrum of the base plasma (i. e., He or He with Ar added) after the addition of an attaching gas was not taken, the effect of an attaching gas on the base plasma can still be determined. Many other spectra were taken with and without the addition of an attaching gas, including spectra of reactive mixtures in

Table 4-3.4

Experimental Conditions for Discussion in Section 4-3.2

Spectrum Label	Reagent Gas	Attaching Gas	Spectral Resolution	Wavelength Range	Recombinatio Expected?
A	H_2	CH_3I	0.4 nm	280-400	N
				550-600	
				700-750	
B	H_2	CCl_4	0.4 nm	250-300	N
				625-675	
C	CO	SF_6	1.2 nm	180-800	N
D	H_2, NH_3	SF_6	1.2 nm	180-800	Y

Legend: Spectrum Label: A convenient label that applies to figures in this subsection. Reagent Gas: The reagent(s) added at RP3. Attaching Gas: The attaching gas used (added at RP2). Spectral Resolution: Approximate spectral resolution (full width at half maximum). Wavelength Range: The spectral range(s) that were used for this discussion (other ranges might be available). Recombination Expected?: If emissions due to dissociative recombination were expected to be observed in these spectra, 'Y,' else 'N.'

which recombination is not expected to occur. In any case, the addition of gases at reactant port 3 (RP3) has little effect on the spectrum of the base plasma except in cases where the injected reagent produces emissions that coincide with the base emissions (see Figure 4-3.3 or the series in Appendix B. It is likely that species injected at RP3 react with species in the base plasma, but where this is the case, the relevant emissions will be affected in the origin spectrum as well as in the background spectrum. Since the concern here is the effect of the attaching gas, such reactions only cause a problem to the extent that they may quench emissions that can provide useful information about processes occurring in the plasma.

Figure 4-3.3

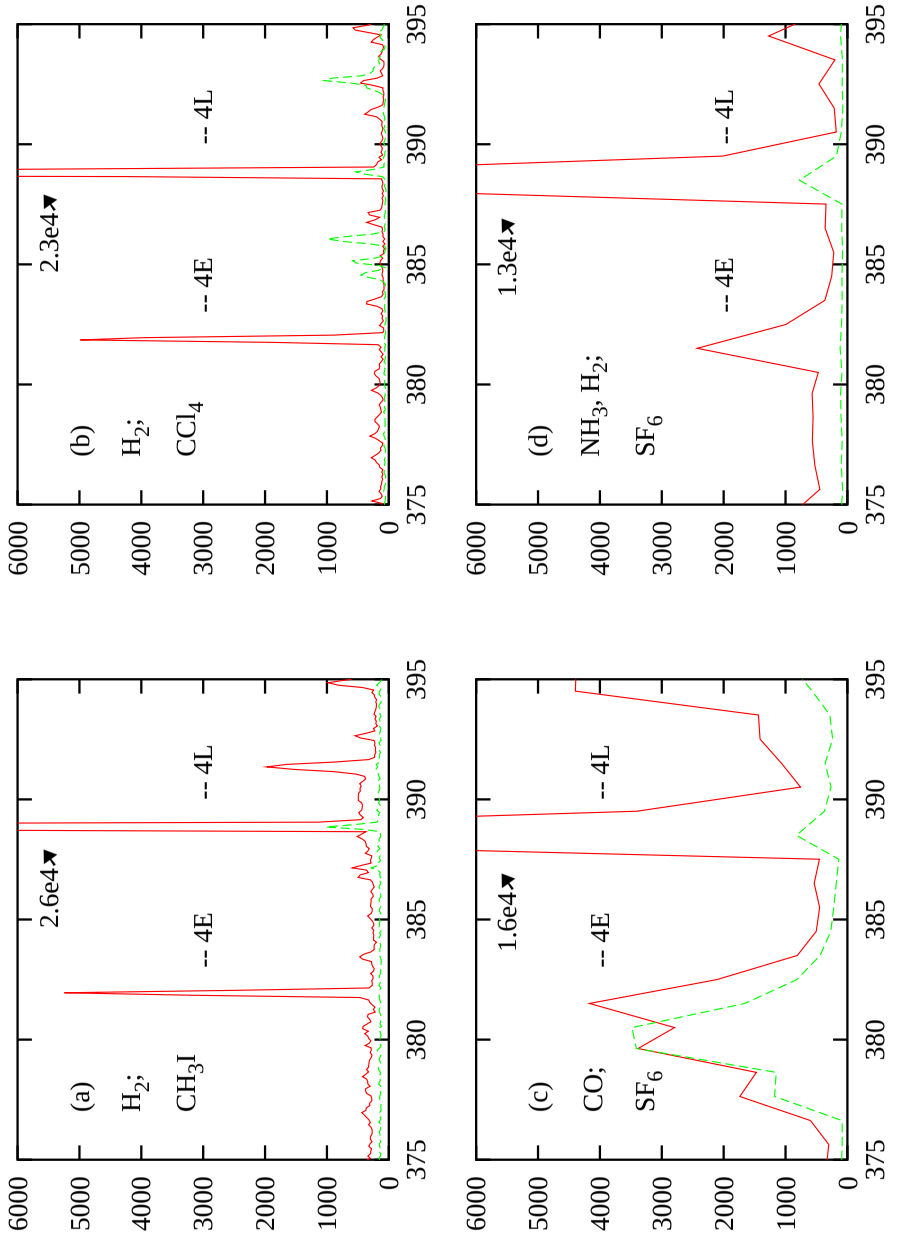


Figure 4-3.3: Comparison of spectra without (red, solid) and with (green, dashed) an attaching gas added. Letters (A, B, C, D) correspond to those in Table 4-3.4, where the experimental conditions are detailed. Molecular species indicated on the graphs are reactant gases and attaching gases. The two dominant atomic peaks in all four spectra are He (4E and 4L in Appendix B). The molecular emission at 380 in (c) is the $A(4) \rightarrow X(0)$ transition of CO^+.

Four spectra, in particular, are useful for determining the effect of an attaching gas. The experimental conditions under which they were taken are given in Table 4-3.4. They were taken at different times and using three different attaching gases. The two at low resolution cover larger spectral regions than the high resolution spectra. But comparisons between the low and high resolution spectra show that the two taken at lower resolution, despite the lower resolution, are reliable sources of information concerning the behavior of plasma emissions upon addition of an attaching gas. Figure 4-3.3 compares the four data sets over a small wavelength range. The small range was chosen for clarity within a single figure; the data shown there are typical for regions where none of the fours sets contain emissions due to the addition of reagents at RP3.

Somewhat surprisingly, the addition of an attaching gas was found to quench nearly all of the emissions in the base plasma. This is in contradiction to what was previously thought [Foley 1993]. The explanation given in the paper in 1993 was based on a more limited set of data than is available here. In the data published in 1993, it was noted that an atomic helium emission ($3p^3P^0_{0,1,2} \rightarrow 2s^3S_1$, at 388.86 nm) was quenched upon addition of the attaching gas, SF_6. The explanation given went as follows. Both singlet He metastables ($2s^1S_0$) and triplet metastables ($2s^3S_1$) were present in the flow. Photons corresponding to transitions between both of those states and higher electronic states of He were generated in the microwave discharge and carried down the flowtube by absorption and re–emission by metastable He atoms in the flow. But, the photons involving the triplet metastable were carried more efficiently because singlet metastables were being converted to triplet metastables via collisions with ambient electrons as they flowed down the tube. When the attaching gas was added, the singlet metastables were no longer converted to triplet metastables and, so, the intensity of the helium transition at 388.86 nm decreased.

Subsequent experiments have shown that the process described in the preceding argument cannot be the dominant process, although it certainly might be occurring.

Table 4-3.5

Atomic Helium Emissions Remaining After Addition of an Attaching Gas

Term(Hi)	Term(Lo)	nm	Label	$\%_f$ (A) (I_i)	$\%_f$ (B) (I_i)	$\%_f$ (C) (I_i)	$\%_f(D)$ (I_i)
$3p\,^3P_1^0$	$2s\,^3S_{0,1,2}$	388.86	4L	3.9 (25871)	2.3 (23474)	5.2 (15890)	5.9 (13097)
$4d\,^3D_{1,2,3}$	$2p\,^3P_{0,1,2}^0$	447.15	6BB			5.6 (6499)	6.7 (5232)
$4d\,^1D_2$	$2p\,^1P_1^0$	492.19	8N			14.4 (3925)	7.0 (2633)
$3p\,^1P_1^0$	$2s\,^1S_0$	501.57	9B			13.8 (3579)	14.7 (3248)
$3d\,^3D_{1,2,3}$	$2p\,^3P_{0,1,2}^0$	587.56	12O	31.0 (24654)		25.0 (13917)	29.9 (12529)
$3d\,^1D_2$	$2p\,^1P_1^0$	667.82	17P		34.0 (2038)	30.9 (2613)	35.4 (2589)
$3s\,^3S_1$	$2p\,^3P_{0,1,2}^0$	706.53	19B	60.6 (6711)		39.6 (3497)	48.6 (3081)
$3s\,^1S_0$	$2p\,^1P_1^0$	728.14	20D	71.3 (923)		54.1 (355)	61.4 (345)

Legend: Label: These labels correspond to the key in Appendix B. Letters A, B, C and D have the same meanings as in Table 4-3.4 and Figure 4-3.3. I_i: The intensity (photon counts not corrected for optical system transmission intensity) of the peak before addition of the attaching gas. $\%_f$: Percent of I_i remaining after the addition of the attaching gas.

If the previous argument were correct, intensities of photons from transitions to the singlet He metastable should increase upon the addition of the attaching gas (since they are no longer being converted to triplet metastables). This is not the case. The intensities of all He emissions are reduced upon addition of an attaching gas; most are quenched entirely.

Since the attaching gas, which is added in very small quantities, quenches the emissions, a few statements about the emissions can be made. One, the emissions are probably not being carried down the flowtube by internal reflection along the walls: it is unlikely that the attaching gas is absorbing the radiation so uniformly across the spectrum, especially since several different attaching gases quench the emissions so similarly as to be indistinguishable from each other. Two, since three different attaching gases affect the emissions similarly, it is unlikely that the changes are due, in a significant manner, to reactions involving the attaching gas (this is expected due to the relative reaction rates of neutrals with neutrals compared to that of electrons with neutrals or other species). Therefore, the presence of electrons in the tube is somehow crucial to the propagation of the emissions. This conclusion is unexpected, initially, since the emissions all correspond to neutral states of helium and dihelium.

A possible explanation can be deduced from the curious fact that some of the atomic helium emissions are not completely quenched upon addition of the attaching gas (Table 4-3.5). Different transitions survive by different amounts, ranging from about 70% down to around 5% (see table). The destination (lower) states for the surviving transitions are also the destination states for transitions that do not survive addition of the attaching gas. So, it is not likely that the destination state makes the difference. Instead, the upper state seems to determine whether a peak is still present after addition of the attaching gas, with the tendency for a peak to survive having the following trend:

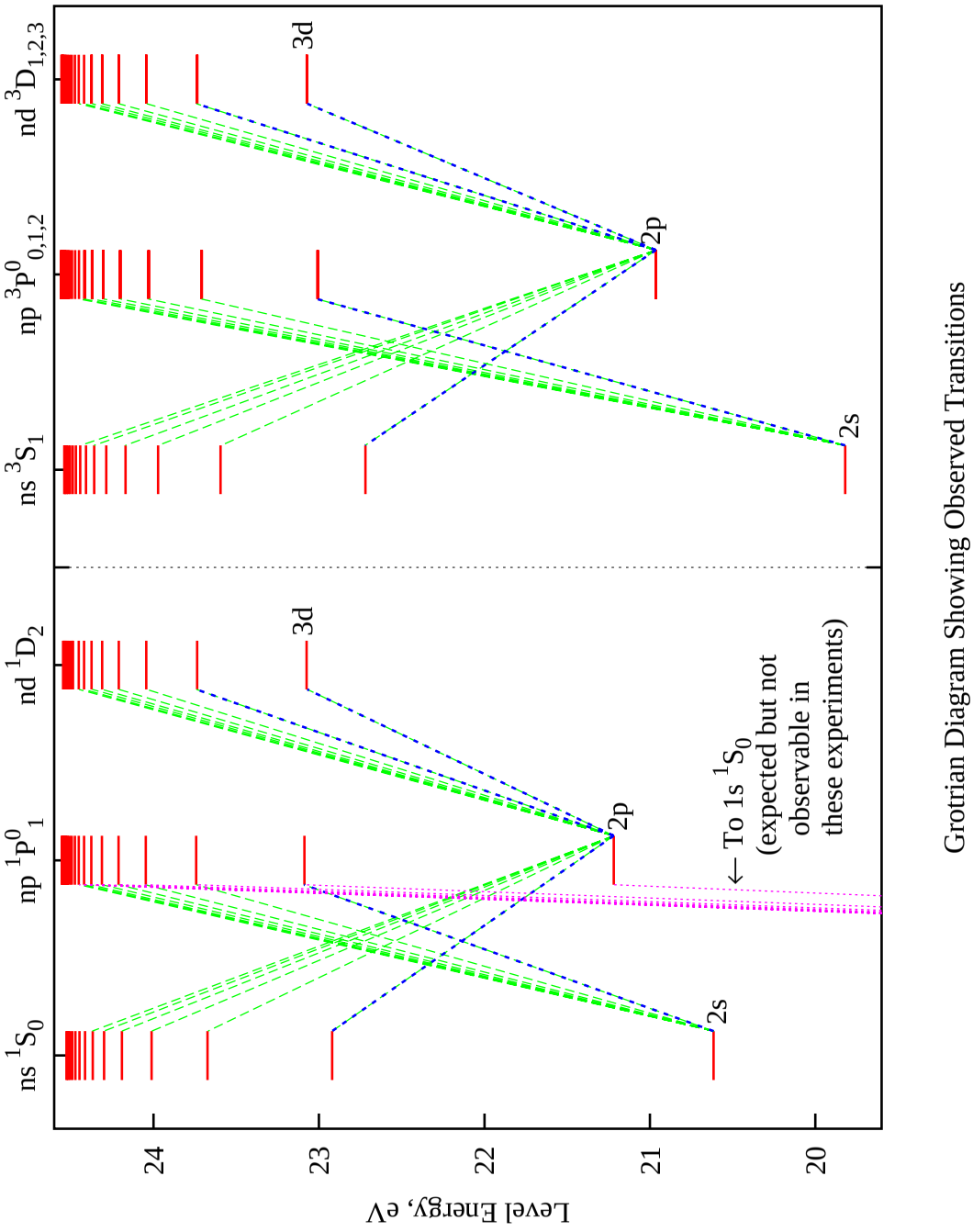
$$3s > 3d > 3p, 4d$$

It is possible that the transition wavelength is important, but it cannot be the only important characteristic of the transitions. If it were, then all atomic helium transitions in those wavelength ranges would survive with similar percentages of initial peak intensity. This is not observed. Neither is initial peak intensity the deciding factor, as inspection of Table 4-3.5 indicates.

Figure 4-3.4

Figure 4-3.4: Partial Grotrian diagram showing the atomic helium lines observed in these experiments. Also shown (and so noted) are transitions to the lower state that are certainly present but that are not observable with the equipment used in these studies. The heavy (blue) lines indicate transitions that survived addition of the attaching gas. Data contained here is from NIST [2001].

Figure 4-3.4



See the caption on the previous page.

A likely explanation is that the atomic He emissions are propagated down the tube by the following equilibrium process:

$$He(1s^2) + He(1s^1ns^1;\ n \geq 3) \rightleftharpoons He_2^+(excited) + e^- \qquad 51$$

The forward and reverse reactions are, generally, studied separately [Berlande 1970, Collins 1972a,b, Ferguson 1965, Johnson 1972a,b, Lichten 1974]. But under the experimental conditions used in this study (high densities and pressures of the relevant species), both forward and reverse reactions are possible. The only requirement, and it is a requirement that should cause concern, is that the exited states of He_2^+ do not react with Ar. The forward reaction has been studied for $n = 5$ [Collins 1972a] and the reaction rate reported as $8 \times 10^{-11} cm^3/s$. Rate coefficients for other values of n are not available. There is disagreement over the rate of the reverse reaction [references this paragraph and Bardsley 1970], and data as a function of initial state of He_2^+ are not available. The reverse reaction for ground state He_2^+ ions is known to be slow [Berlande 1970]. But, that might not be the case when the He_2^+ is created via the forward Reaction in 10. In that case, the product He_2^+ is likely to be in a high vibrationally and/or electronically excited state: not at all the He_2^+ expected from the three body route:

$$He^+ + 2He \rightarrow He_2^+ + He^* \qquad 52$$

The equilibrium described in Reaction 10 is consistent with the data if the forward (and possibly also reverse) rates are much slower for the lowest values of n. If this is the case, then the addition of the attaching gas quenches the emissions by limiting production of the upper state rather than by removing the lower state from the plasma. In other words, the atomic helium emissions are emissions resulting from dissociative recombination. After the attaching gas is added, only the forward reaction in 10 can occur and, eventually, all the excited state helium atoms react away,

producing He_2^+ ions. If the forward rates (and, possibly, but not necessarily, the corresponding reverse rates) are small for the $3s$, $3p$, $3d$ and $4d$ states, then addition of the attaching gas will have less effect on those emissions.

It is likely that the diatomic helium emissions are also products of the reverse of Reaction 10. Lichten, et. al [1974], reported such a recombination as the source of the dihelium in their study. The fact that the dihelium emissions are quenched by the addition of the attaching gas supports this argument as well. Since the Ar^+ peaks are also quenched upon addition of the attaching gas, their presence can be tied to the reaction in 4-10 as well.

If Reaction 10 is occurring, then the unexpected behavior of the CO^+ emissions (Section 4-1) is explained, and steps can be made to reduce it's influence on the chemistry in the plasma. So long as energetic species that depend on the electron concentration are present, it will be difficult to confidently identify photoproducts of recombination.

CHAPTER 5

THE FLOW OF GASES IN THE REACTION REGION

In this chapter a number of issues surrounding the injection of reactant gases at RP3 are discussed. In particular, how spatial and temporal characteristics of the injection plume affect the data presented in Chapter 4. In some cases, these characteristics limit the precision with which relative populations of observed electronic states can be determined. Since this is most important, it will be presented first.

5-1 The Size and Temporal Characteristics of the Injection Plume

The relative populations of observed electronic states, given in Tables 4-2.1 – 4-2.9, were calculated from the observed number of counts, taking into account the radiative lifetime for that state. Photons resulting from various processes in the flowtube are only collected for the time that the emitting species are within the viewing region of the optical spectrometers. If A is an electronically excited atomic or molecular species, the total number of photons, I_t, observed from an initial population, A_0, within a given time, t is:

$$I_t \propto A_0\left(1 - \mathrm{e}^{-t/\tau}\right) \qquad 53$$

where τ is the radiative lifetime of the emitting species. This equation can be derived by recognizing the radiative lifetime as the inverse of a unimolecular decay rate constant. In that case, the number of A left after time, t, is:

$$A = A_0\mathrm{e}^{-t/\tau} \qquad 54$$

That implies that the number (or number density, as appropriate) of photons, $A_{h\nu}$, emitted by time t is:

$$A_{h\nu} = f_{h\nu} A_0 (1 - \mathrm{e}^{-t/\tau}) \qquad 55$$

where $f_{h\nu}$ is the probability that A will radiate via the channel of interest: if A decays to more than one lower electronic (or other) state, either all emissions must be considered or the fraction of decays to a single lower state must be known. Then the number, I_t, of photons observed will be proportional to the number emitted (the proportionality constant will depend on specific experimental circumstances).

It is important to know the overall probability that a state will radiate: an Einstein A coefficient, while useful, does not give sufficient information. Put more simply, the inverse of the radiative lifetime gives the frequency with which the upper state is likely to radiate. The Einstein A coefficient gives the probability that the radiation will decay into a particular lower state. So, the radiative lifetime available is used to determine the overall number of transitions. Where probabilities for individual channels are known (for instance, where Franck–Condon factors are known), they are used to predict relative importance of individual channels.

Consider making a comparison between emissions from the decay of a several different electronic states in order to determine their relative populations. The period of time over which the radiating states are viewed is very important, and is the central point of this section. In two cases, Equation 1 can be simplified. If the ratio of t to τ is large for all excited states, then the relative population of each state can be determined by finding the relative number of photons emitted from each state. This can be verified by substituting $t = \infty$ into Equation 1. If the ratio of t to τ is very small for all species, then the relative populations can be determined by multiplying the observed number of photons by τ. This can be verified by taking a truncated

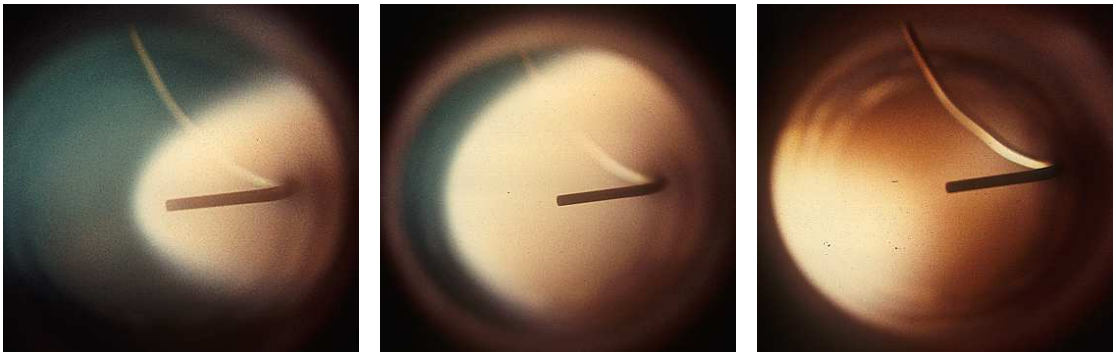
expansion of the exponent in Equation 1:

$$\begin{aligned} I_t &\propto A_0\left(1 - \mathrm{e}^{-t/\tau}\right) \\ &\propto A_0(1 - (1 + \frac{-t/\tau}{1!})) \quad small\ t/\tau \\ &\propto A_0 t/\tau \end{aligned} \tag{56}$$

This is not straightforward since the different emitting states in a given data set often do not have radiative lifetimes that are of similar magnitude, let alone radiative lifetimes that fall neatly into one of the two extremes presented in the last paragraph. In those cases, it is necessary to know the amount of time that the collection of emitting species is in viewing range of the observation device. Note that this is a different time from the counting period of the observation device. In flowing studies in steady state, the latter is relevant only in terms of enhancing signal to noise ratios.

The time the species spend in the viewing region can often be ignored because many electronic transitions have short radiative lifetimes, but, exceptions to this exist, for example, the $a^3\Pi_r$ state of CO. The radiative lifetimes of the a and $A^1\Pi$ states of CO are 8 ms and 11 ns, respectively [Radzig 1985]. 99% of molecules in the A state will have radiated within about 50 ns. It will take about 40 ms for 99% of the a state to have decayed, a time roughly the same as the travel time to the end of the flowtube ($\sim$100 cm). An upper–limit estimate of the time the gases are within view of the spectrometers is 2 ms. In that time, all of the A state is expected to have completely radiated, but only about 22% of the a state will have. A similar lower–limit estimate of the time the species spend in viewing range of the spectrometers is 0.5 ms (the basis of these estimates is discussed shortly). To illustrate the significance of this, let us assume that equal numbers of photons are observed from the two states. If a viewing–region time of 0.5 ms is assumed, then the relative populations of the A and a states are 0.06 to 1; if 2 ms is assumed, the relative populations are 0.22 to 1.

Figure 5-1.1



$Q \approx 0.04\ Tls^{-1}$ $Q \approx 0.5\ Tls^{-1}$ $Q \approx 2\ Tls^{-1}$

Figure 5-1.1: Photographs of the reaction region at three throughputs, Q, of CO_2. The CO_2 is injected into the typical reactive plasma described in Chapters 2 and 4. In all three photos, the plasma is flowing from left to right, and the reagent gas, CO_2, is being injected through RP3 into and against the flow of the plasma. RP3 is the small tube that begins near the top of the photographs, curves around and ends approximately in the center of each photograph. The He throughput is 220 Tls^{-1}. The first two photographs represent conditions similar to those, respectively, in the top and bottom graphs in Figure 5-1.2. In the third photo, sufficient CO_2 is being injected that the gas travels upstream beyond the viewing region. The distance traveled upstream is unknown, but an estimate is made in the text. Photographs by Aaron Powell.

In the experiments being described here, the amount of time the reagents spend in view of the spectrometers also depends on the throughput with which they are injected through RP3. This time difference is illustrated fairly plainly in Figure 5-1.1. The three pictures are photographs of the region around RP3 under typical experimental conditions, but varying the throughput of the gas being injected at RP3. The photographs in this figure were taken, using standard photographic techniques,

through a viewport perpendicular to the flow, situated on top of the flowtube. The circular boundary, outside which the photographs are black, is the edge of the viewport. The throughputs given in the figure correspond to number densities of, left to right, 9.2×10^{12}, 1.2×10^{14} and 4.8×10^{14} cm^{-3} once the CO_2 has expanded to fill the flowtube.

The time estimates used earlier come from assuming that once the plume has been turned back it will have the same velocity as the bulk gases. The distance across the viewport is about 5 cm. If one assumes the slower, neutral particle velocity, 3000 cm/s, then the viewing time is about 1.7 ms or ≈2 ms. Similarly, if the gas exits the tube, turns back immediately and travels half that distance with the higher speed of the ions in the flow, 4500 cm/s, it will have spent about 0.5 ms traversing the window.

Spectra from the recombination of CO_2^+ taken under different injection conditions illustrate the combined effects of injection throughput and product radiative lifetime (Figure 5-1.2). The throughputs for the upper and lower spectra are similar to those in the left and middle photographs, respectively, in Figure 5-1.1. Note that the intensity of the CO $a \rightarrow X$ transitions change relative to the constant intensities of the CO $A \rightarrow X$ and CO_2^+ $\tilde{B}\rightarrow\tilde{X}$ emissions. The radiative lifetime of the CO_2^+ $\tilde{B}$ state is not known, but since the transition is not spin forbidden, it is expected to be more similar to that for the CO A state than to the a state. Indeed, conversely, if the time the species spent within range of the spectrometers were known, and it were sufficiently short, a radiative lifetime could be assigned to the CO_2^+ emissions.

The values reported for the relative populations of the a and A states of CO in the recombination of CO_2^+ (see Table 4-2.3) are based on experimental conditions similar to those in the center photograph in Figure 5-1.1. This set of conditions was chosen because the time spent in the viewing region is maximized, but none of the reagents have traveled upstream, out of range of the optical spectrometers. Uncertainty regarding the time the radiating species are in view of the spectrometer

Figure 5-1.2

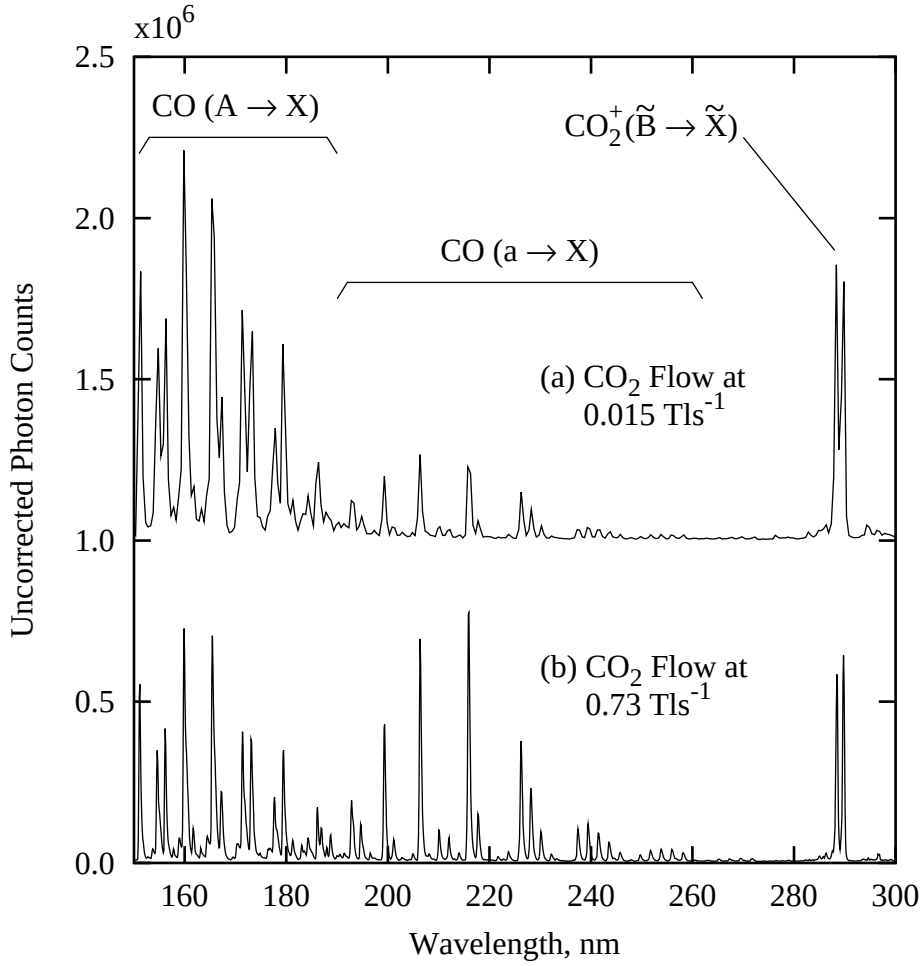


Figure 5-1.2: Comparison of intensities of spectral features observed in a plasma containing CO_2^+ ions for different reagent gas flows (CO_2, here). The upper graph has been offset on the abscissa by 1×10^6 for ease of comparison. The throughputs indicated correspond, once the CO_2 has expanded to fill the tube, to number densities of (a) 3.5×10^{12} and (b) 1.7×10^{14} cm^{-3}. The conditions under which the spectra were taken are similar to those for the first two photographs in Figure 5-1.1. The resolutions (FWHM) and count periods are approximately (a) 1.5 nm, 2 s and (b) 0.5 nm, 4 s. The difference in resolution does not account for the difference in relative heights between the CO $A \rightarrow X$ and $a \rightarrow X$. See accompanying text for further details.

limits the precision of the results. This is a difficult problem, but if a good number for the injection velocity and the time variation of the velocity were known for a known average travel distance, a better estimate of this time could be made.

An estimate of the injection velocity can be made, but the estimate is not very reliable. A throughput, essentially, measures the number of a given species that pass a certain point in a given time. If the pressure, p of the gas is known, the throughput, Q, can be converted to a velocity using the relationship:

$$Q = pav \qquad 57$$

where a is the cross–sectional area of the tube where the pressure is measured and v is the velocity of the gas. The only point at which the throughput and the pressure are known simultaneously is at the measurement of pressure drop across the capillary (see Chapter 2). The capillary tube exit velocity is not necessarily the same as the velocity with which the molecules exit RP3; only the throughput is a constant. The velocity of the gas will not necessarily remain constant since it must pass through a considerable length (decimeters, at least) of tubing with diameter changes. For the central photograph in Figure 5-1.1, the throughput is about 0.5 Tls^{-1} and the pressure at the exit point from the capillary tube is 53 Torr. Assuming an inner diameter for the capillary tube of 0.05 cm, the velocity of the gas is about 1200 cm/s. For the lower flow shown in Figure 5-1.1, the injection velocity, by this estimation, is around 270 cm/s.

To make a calculation of residence time within viewing region of the optical spectrometers is not trivial. The problem is that the distance traveled by the injected gas depends on the injection velocity, the mass of the injected species and the frequency of collision with the oncoming He/Ar mixture. The latter, of course, depends on the number density, temperature and collisional cross sections of all species in the flow. Two of those (temperature and cross section) can be assumed to be constant when

the gas is first injected, but the number density decreases due to expansion as soon as the gas exits RP3. Very soon, all of these variables change. As the CO_2 reacts, it not only changes size, but becomes an ion. Since a head–on collision is not necessary for charge transfer, a number of CO_2 molecules will become ions while still moving into the plasma. An ion interacts with an oncoming plasma in a very different manner than a neutral species. After the ion recombines, it becomes two or more different particles with different masses and collisional cross sections and temperatures. After that, the cross sectional areas of some of the product neutrals will change as excited species radiate.

A typical distance traveled by an injected particle can be determined empirically. The apparent size of the plume can be measured in a series of photographs similar to those shown in Figure 5-1.1. The distance from the terminus of RP3 to the visible edge of the plume has been measured relative to the size of the viewport in each photograph. A plot of the distance traveled upstream as a function of throughput is fairly linear (see Figure 5-1.3). This is true both for injection of a species likely to be involved in ion chemistry (CO_2) and for the injection of a gas likely to react only as a neutral (Ne injected into a He–only plasma). So, even if it is difficult to determine the amount of time the species spend in front of the viewing region, it is possible to estimate the volume of the reaction plume and to predict the distance upstream that an injected gas will travel. From the linear fits in Figure 5-1.3, the injected gas in the rightmost photograph in Figure 5-1.1, which has traveled upstream past the edge of the viewing window, traveled about 10 cm upstream.

A plot of the capillary tube exit velocity versus upstream distance is, as expected, non–linear. However, if a throughput is considered to be an energy, then it makes sense that the distance traveled upstream would be linear in the square of the particle velocity. This is indeed the case (Figure 5-1.4). If the kinetic energy were the sole concern, then the ratio of the slopes for CO_2 injection and Ne injection should be

Figure 5-1.3

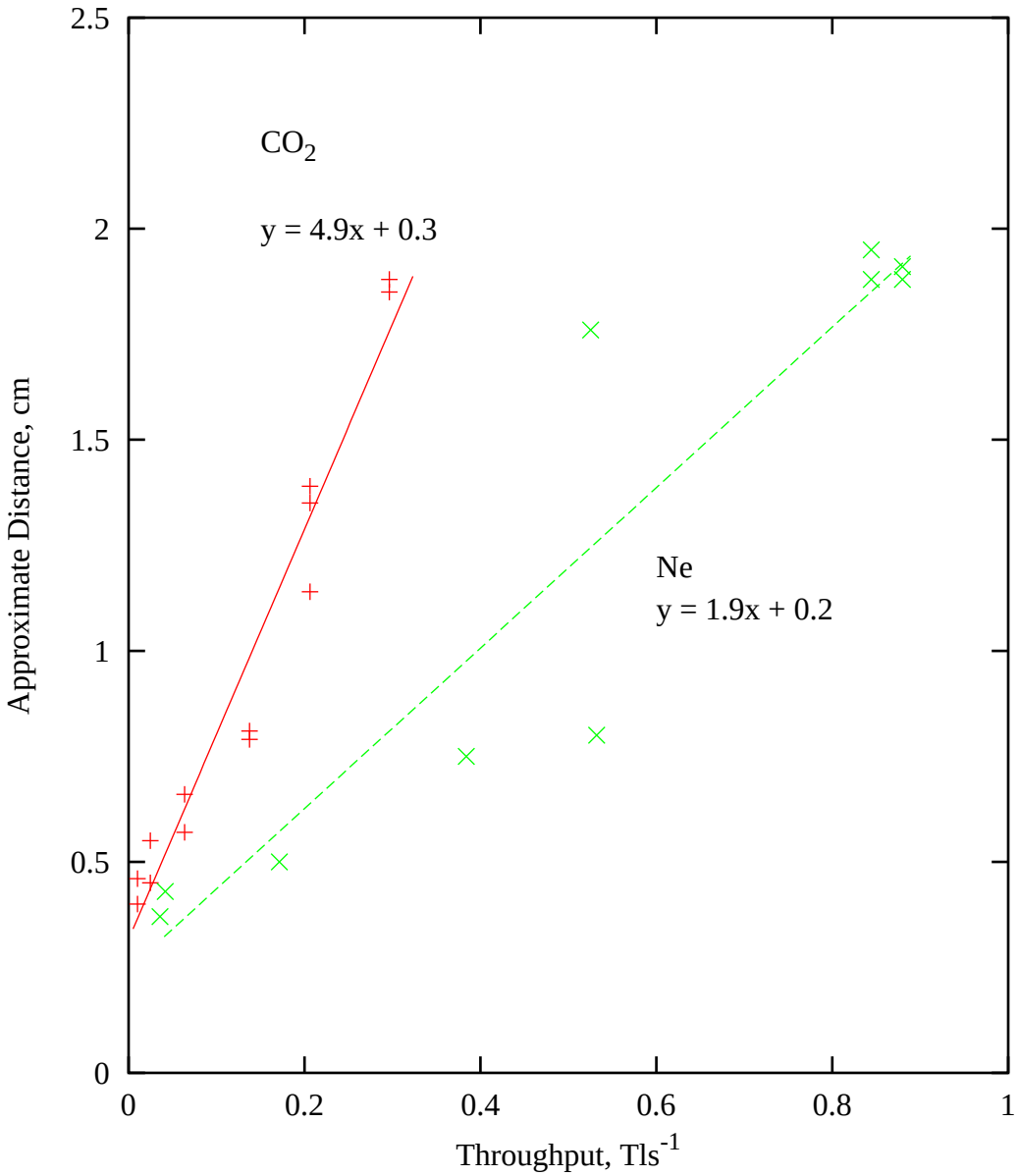


Figure 5-1.3: Upstream distance traveled by parent gas injected through an upstream facing port as a function of throughput for CO_2 injected into a He and Ar plasma (red +'s) and for Ne injected into a He-only plasma (green ×'s). Linear best-fit lines with their equations are also displayed.

Figure 5-1.4

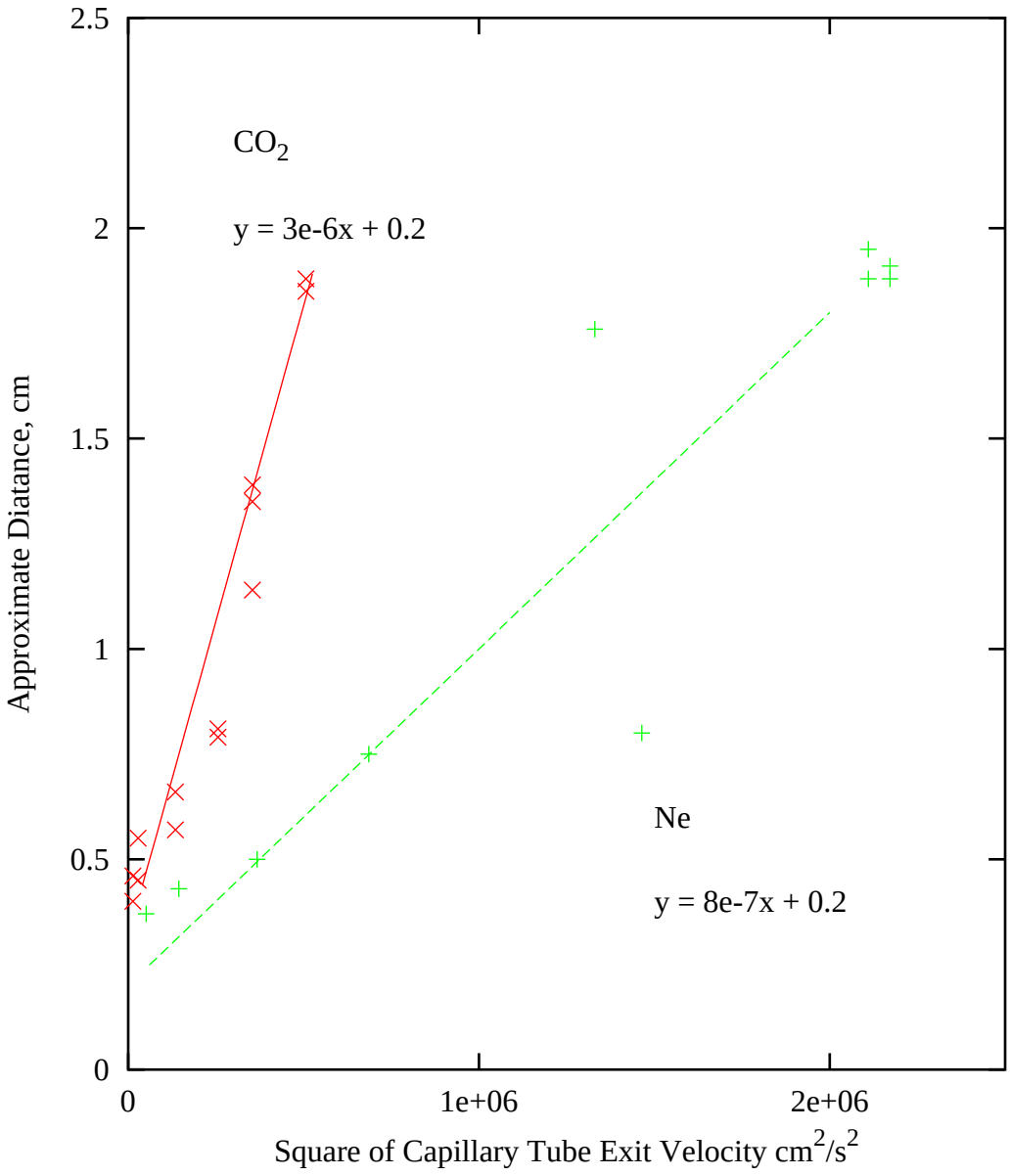


Figure 5-1.4: Upstream distance traveled by parent gas injected through an upstream facing port as a function of the square of capillary tube exit velocity for CO_2 injected into a He and Ar plasma (red +'s) and for Ne injected into a He-only plasma (green ×'s). Best-fit curves and equations are also displayed on the plot. In the equations, x is the square of the velocity.

the ratio of their masses (2.2), but the exact value is actually 4.3. The difference in collisional cross section for the two primary collision pairs (0.34 nm^2 for CO_2/He and 0.22 nm^2 for Ne/He [Spartan '02 calculation]) will make a difference in the distance traveled, too. But, the distance traveled will decrease with an increase in collisional cross section, so its inclusion will only serve to make the numbers even less alike. The additional presence of Ar in the plasma will also make a difference. But, since the argon number density is about one thousandth that of He, its effect is expected to be small. What is more likely is that the assumption about the equivalence of the injection velocity and the capillary tube exit velocity is erroneous, and the injection velocity is higher.

Despite the capillary tube exit velocity not being equal to the injection velocity, it is still a valuable assumption. If the injection velocity is known, then the time between exiting RP3 and turning back can be estimated. At the point at which the injected gas turns back, its velocity is zero. That makes the average velocity, assuming constant deceleration, equal to half the injection velocity. For the central photograph in Figure 5-1.1, the average capillary tube exit velocity is about 600 cm/s. So, given these assumptions, the time it takes for the oncoming plasma to stop the CO_2 is about 2 ms.

The slopes of the lines describing distance traveled upstream as a function of capillary tube exit velocity are not in the ratio of the masses of CO_2 and Ne. However, the slopes for the distance as a function of throughput are nearly in that ratio (2.6 slope ratio versus 2.2 mass ratio). If a good predictive model for the exit velocity as a function of throughput could be made, then the relative populations of electronic states with differing radiative lifetimes could be determined much more accurately.

5-2 The Shape and Spectral Characteristics of the Injection Plume

One interesting characteristic of the injection plume is the dependence of its shape on the reaction conditions. This difference is illustrated in Figure 5-2.1. The figure compares the injection plumes of two reactive systems. The photographs were chosen for comparison because the distance the injected gas travels upstream is similar. The photograph on the right is somewhat difficult to interpret: the light from the reaction between metastable helium atoms the injected neon (the He/Ne laser reaction) is bright and reflects off the port on the other side of RP3 from the camera. But, even with the difficulty, it is fairly clear that the two plumes are different. The most striking difference is that the plume on the left has a very sharply defined front, while the plume on the right has less of a "front" than a "leading diffusion zone." The difference between the shapes lies in the nature of the species emitting light.

In the photograph on the right, the injected Ne is reacting with neutral helium atoms to produce neutral neon atoms. The neutral neon atoms then radiate in the red. Since all the species involved in the production of the photons are neutral, then the positions at which the species react and then radiate should be controlled, in some sense, by diffusion. When mixing is diffusion controlled, the distribution of species is random–kinetic: a few should diffuse very quickly away, while others diffuse more slowly. Here, the "tag" for the presence of neon atoms is the red glow. The glow is of the form expected for a diffusing species, i. e., brighter near the center from which the species diffuse (RP3) and fading as the distance from RP3 grows (the yellow–ish feature near the center of the plume is discussed later). The purplish glow to the left of the plume (faintly visible in the photograph) is the combined effect of a number of mainly He but also He_2 emissions (see Section 4-2 for details).

In the photograph on the left, all of the dominant emissions depend on the presence of the CO_2^+ ion. The visible emissions (from about 400 to about 700 nm) are produced by the CO_2^+ $\tilde{A}\rightarrow\tilde{X}$, CO^+ $A \rightarrow X$ and the CO $d \rightarrow a$ transitions. The

Figure 5-2.1

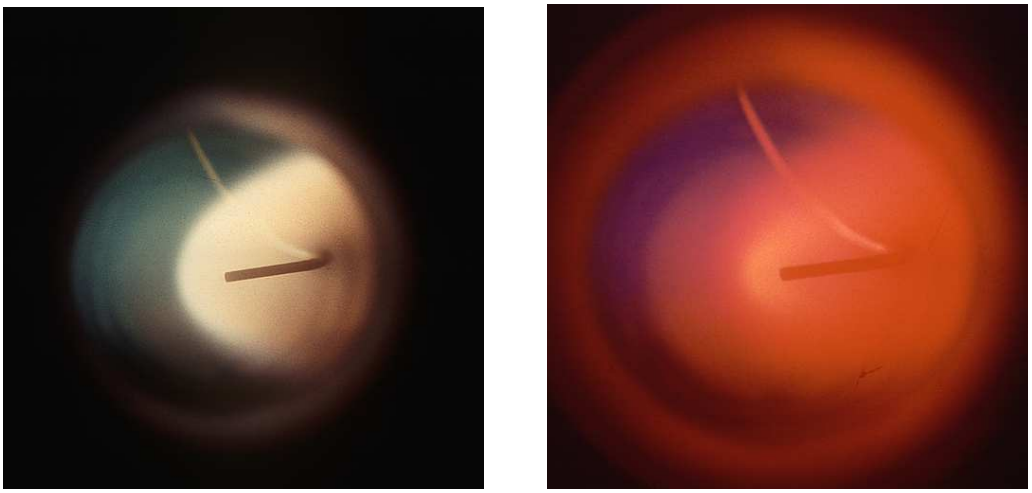


Figure 5-2.1: Comparison of the plume shape when the emitting species is a product of reactions involving ions (left) and reactions involving neutrals (right). In the reaction on the left, CO_2 (the gas injected through RP3) is ionized by the oncoming plasma and then rapidly recombines with ambient electrons. In the reaction on the right, Neon, injected through RP3, reacts primarily with neutral He metastable atoms in the plasma, producing, among other emissions, the red glow characteristic of common He/Ne lasers. See text.

---

only neutral in that list, the CO, results from the recombination of CO_2^+ ions and is, therefore, the result of an ion–related process. The greenish glow to the left of the plume (faintly visible in the photograph), is produced by the combination of He, Ar and Ar^+ and, to a small extent, He_2 emissions (see Section 4-2 for details).

Unlike the emissions in He/Ne photograph, the emissions in the CO_2/He/Ar photograph are brightest at the sharply defined outer edge of the plume. If the situation were diffusion controlled, this would not be the case. Two mechanisms give rise to the distinctive appearance of the plume. The first is that charges that were moving from left to right before the CO_2 was injected are being transferred to the

CO_2. Charge transfer does not require a head–on collision. Indeed, according to the harpooning mechanisms used to describe a number of reactions [e. g., Atkins 1998] the electron frequently changes host (via a long–range electron transfer) without any close collision at all. When the charges transfer, then, they are, to some extent, being transferred to very massive objects that are moving in a direction opposite to the original motion of the charge. Now there is the situation of a charged particle moving into a plasma. While the plasma, overall, is neutral, it is not neutral in a local sense. The newly–charged CO_2^+ ions will be repelled by the oncoming plasma by long–distance ($\sim$ Debye length) interactions, which are Coulomb–like, without having to undergo collisions. The second mechanism responsible for the appearance of the plume concerns the recombination of the CO_2^+ ions. The area behind the front, after the ions have recombined, will be ionization–poor and neutral CO_2 molecules will continue to diffuse to the point where the oncoming plasma meets the advancing front. The effect of these circumstances is to concentrate the CO_2^+ molecules, whose positions are recorded in the photograph. The light is brightest, and whitest, near the front because many of the species formed by reactions between CO_2 and the plasma have very short radiative lifetimes.

The glow not only dims but appears also to redden or become browner as the flow continues downstream. It is difficult to be certain from a photograph, but if the effect is real, then it might be due to diminishing populations of the CO_2^+ $\tilde{A}$ and CO^+ A states. The CO_2^+ $\tilde{A}$ radiates at the blue end of the spectrum (up to about 500 nm) and the CO^+ radiates all the way across the visible spectrum. But, the CO $d \rightarrow a$ transition has three broad peaks near 600, 650 and 700 nm, with the brightest being around 650.

The yellow–ish feature in the He/Ne plume

The observant reader [Ferrenberg 1996] will probably have noticed a yellow–ish band of color just to the left of RP3 in the photograph on the right in Figure 5-2.1. The

Figure 5-2.3

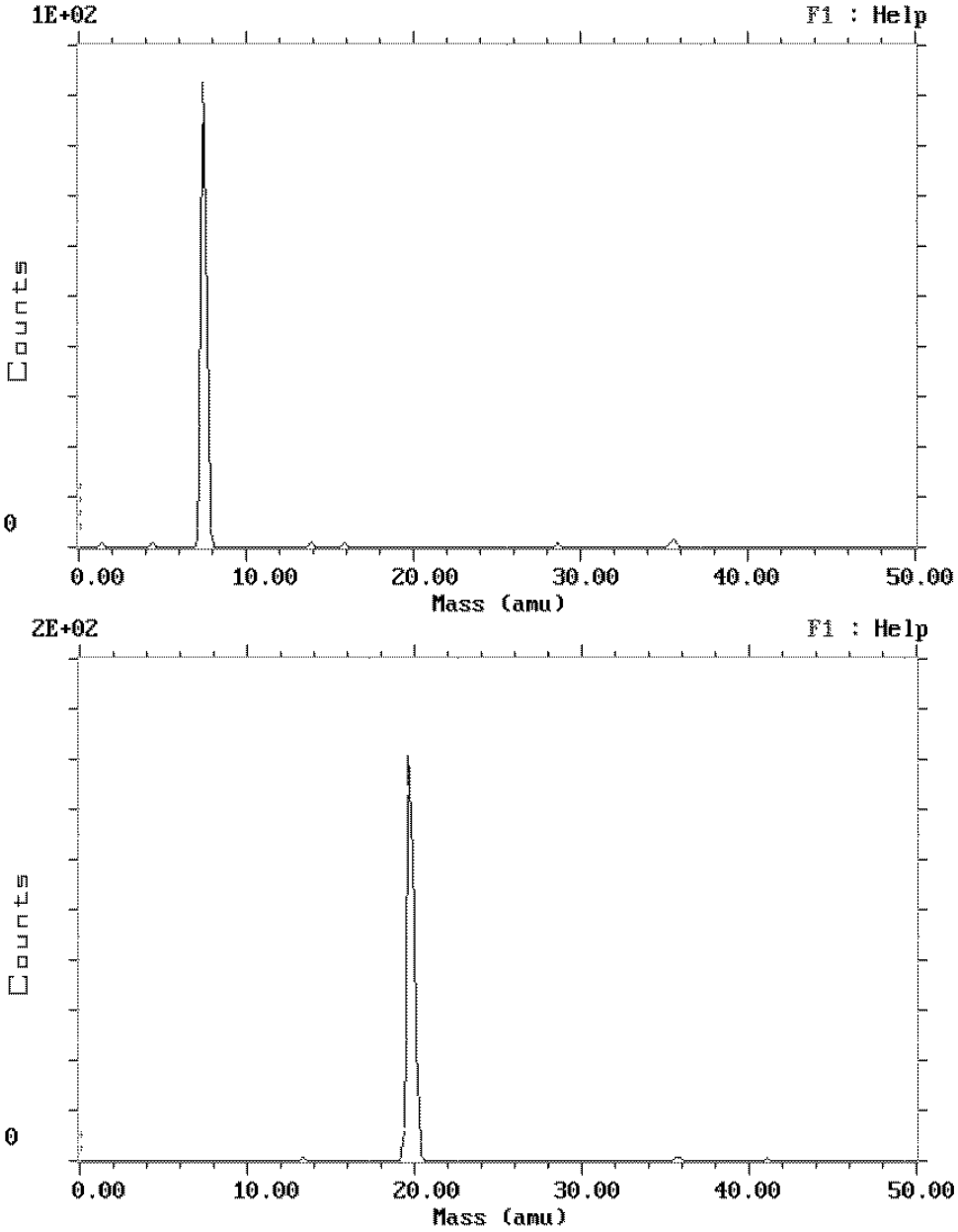


Figure 5-2.3: Mass spectra taken prior to taking the most recent optical spectra and pictures of the He/Ne reaction. The top spectrum was taken before neon was injected and is dominated by the presence of the He_2^+ ion. The bottom mass spectrum clearly shows a dominant peak at 20 amu without the expected 22 amu isotope at about 10% of the intensity of the 20 amu peak. At this resolution, the two masses, if both present, would be well resolved.

Figure 5-2.4

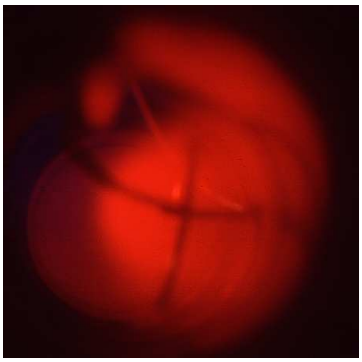


Figure 5-2.4: Photograph of the region around RP3 using the isotopically pure neon. The photograph is at much better resolution than is shown here. An additional ring-shaped port was added prior to the time this photo was taken. The ring-shaped port encircles RP3. It confuses the photograph, but does not significantly alter the shape of the plume. Even with a better copy of the photo, there is no apparent yellow band. An obstructing piece of permanently-placed equipment prevented exact duplication of the previous photographs.

---

concentration of the yellow into a narrow band is reminiscent of the shape of the front in the CO_2 system where the emissions are primarily due to ion–related processes. A set of spectral data taken at the time of the photographs seemed to show a couple transitions that were affected significantly by the addition of an attaching gas. So, the experiments were repeated with a new bottle of neon to confirm the presence of the yellow band was not a photographic artifact. According to the mass spectrum, the new sample of neon was isotopically pure (20 amu, Figure 5-2.2). Another mass spectrum taken prior to the addition of the neon (same figure) indicates that the system was very clean. Photographs taken using neon from the new bottle (Figure 5-2.3) do not contain the yellow band. The spectra, also, do not indicate electron dependent emissions that would be perceived as yellow. The bottle of neon used in

the earlier photographs was not isotopically pure. So, it is possible that the yellow feature in the previous photograph was real. If so, then it was produced either by an impurity in the neon or possibly by some isotopic effect, though the latter is not as likely as the former.

5-3 Simulation of the Injection Plume at RP3

Empirical investigations of flow characteristics are certainly useful. But, given the complexity of the interactions near RP3, a more complete understanding of the reactive flow can only come from building models and testing them for consistency with experimental findings. To that end, the first phase of a computational model of the flow has been developed. The details have been described elsewhere [Foley 1997], so only a synopsis will be presented here.

A computer program was developed that models the injection of neutral particles from an orifice into a larger opposing gas flow. The earliest models were full molecular dynamics simulations assuming hard walled collisions between spheres of finite size. The flows that could be simulated using these methods occurred in boxes that were, at most, a few thousand Å on a side: particles injected through the virtual RP3 were rarely turned around in the simulations. These simulations were event–driven: that is, the simulation determined the next pair of particles (or particle and wall) that would collide and moved all other particles to their new positions at the time of that collision. The collision was allowed to occur, and the colliding particle(s) were assigned new trajectories and velocities. Then the next collision was found and the process repeated. Although the simulation size was limited, the results were realistic (Figure 5-3.1).

The simulation of flows within boxes of somewhat more realistic size and flow characteristics became possible by treating the oncoming flow statistically, with the

Figure 5-3.1

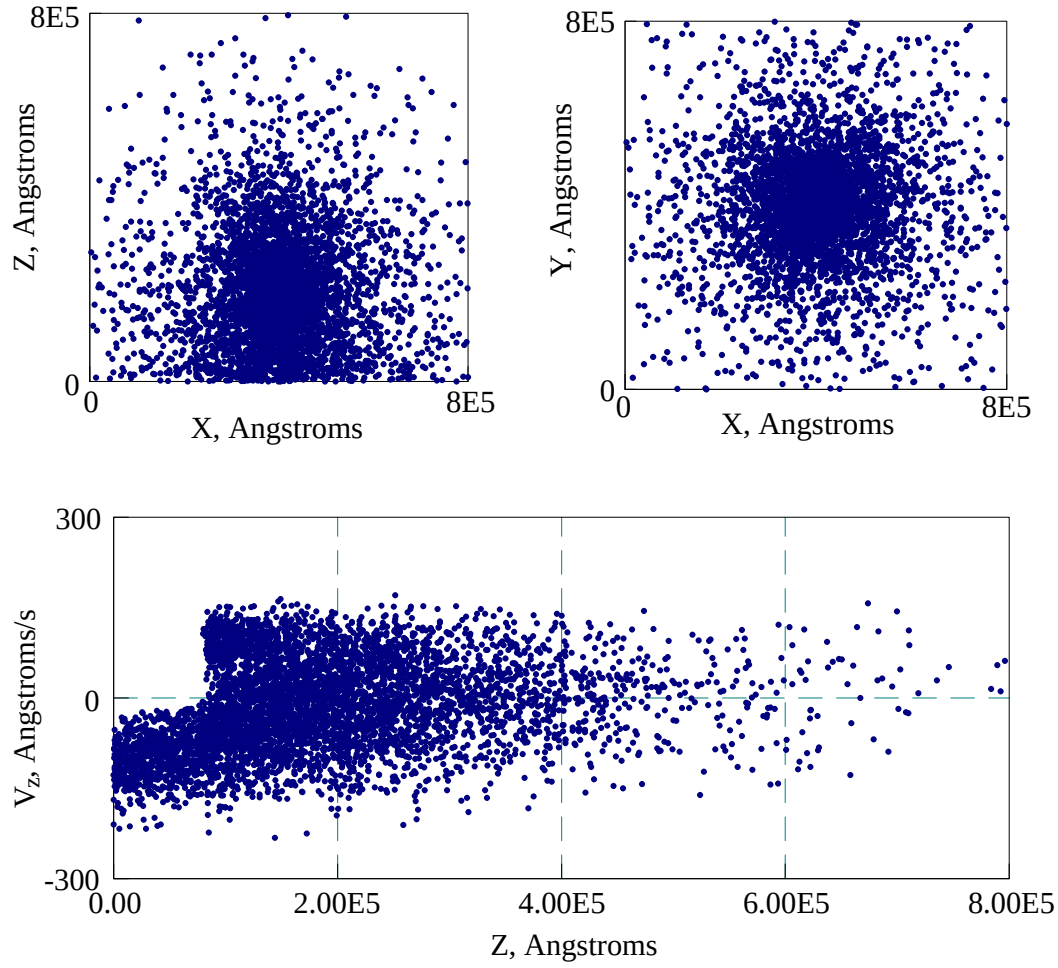


Figure 5-3.1: Simulated injection plume from an event-driven simulation of 1000 CO_2 particles injected into counterpropagating neutral helium (the helium atoms are not displayed in the graphs). Particles are injected through a virtual port located centrally along X and Y (upper right), and at 10,000 Å from zero along Z (top left and bottom). They exit the port in the positive Z direction and encounter He atoms flowing in the negative Z direction. *Upper left:* Side view of the simulated plume. *Upper right:* View along the z direction showing the symmetry of the simulated plume. *Lower graph:* A plot of the velocity in the z direction, V_z, as a function of position along z. There are no positive values of V_z for positions behind the injection port.

event–driven algorithm being replaced by a temporally controlled simulation. In the current simulation, a single particle is assigned an initial trajectory and velocity according to a Maxwell–Boltzmann distribution. Taking account of the distance the particle is likely to travel within a constant timestep, a collision probability is assigned. A random number is chosen, and if the collisions probability is greater than the random number, the collision occurs; otherwise, the particle completes its trajectory. Using this algorithm, the trajectories of many particles can be traced. A relatively small collection of these single particles, collectively, represent the statistical distributions of a much larger set of particles. Using this algorithm, the basic shape of a diffusion controlled injection plume was generated.

As was stated earlier, this is merely the first phase in a more complete simulation. A more complete simulation would also take account of charge distributions, collisions between injected particles and reactions. The final simulation will probably not be limited to stochastic methods like that described in the last paragraph. Where statistical distributions could not be predicted, the simulation would need to include more familiar techniques such as molecular mechanics and dynamics and the numerical treatment of equations (such as reaction rate laws) known to describe certain processes. The simulation will also need to reproduce photoemissions from the flow, the subject of the next chapter.

CHAPTER 6

SIMULATOR FOR ROVIBRONIC SPECTRA

A computer program that simulates rovibronic spectra is described. A brief rationale for the program (Section 6-1) will be followed by a list of features available in the current version (Section 6-2) and a description of input and output files (Section 6-3). Section 6-4 summarizes the theoretical descriptions on which the simulation is based. Section 6-5 describes the storage and retrieval of data within the program. Since the program was written with the intent that it should be modified and expanded, Section 6-6 contains a list of suggested changes and additions. Finally, Section 6-7 contains a comparison of spectra simulated by the program to experimental spectra observed in this study and collected from the literature. There are numerous other comparisons to experimental data from the studies presented here in Section 4-2.

The source code for the program is written in the programming language C and is contained in Appendix C. The name of the program is specified by the user when the program is compiled and within an input file. But, within this dissertation, the program is referred to as "RVESIM" which stands for rotational vibrational electronic simulation.

6-1 Spectral Simulation

There are many reasons why one might wish to simulate a spectrum. An obvious reason is to facilitate identification of spectral features. Even if feature identification is not a concern, simulation can simplify quantification of properties such as temperature or relative population. The simulation can also be used for design: the necessary or

optimum spectral range for an application can be determined based on simulations of the frequencies of light absorbed or emitted.

Relatively simple rovibronic (rotational, vibrational, electronic) spectra can be easily simulated. A great deal of information exists regarding atomic and molecular spectra of stable diatomic molecules. References such as Huber and Herzberg's *Constants of Diatomic Molecules* [Herzberg 1997] or collections of data like Radzig and Smirnov's *Reference Data on Atoms, Molecules and Ions* [Radzig 1985] contain detailed information concerning the spectroscopic behavior of a large number of molecules. The National Institute for Standards and Technology (NIST) provides a variety of spectral resources online [NIST 2001a,b], including an updated version of Herzberg's reference. There is less information available on small polyatomics, but a few texts contain information. Herzberg [1991] included a table of spectral information in the back of his book *Molecular Spectra and Molecular Structure, V III: Electronic Spectra and Electronic Structure of Polyatomic Molecules*. More recently, Marilyn Jacox [1994] published spectral information for a number of species in a monograph for the *Journal of Physical and Chemical Reference Data* entitled *Vibrational and Electronic Energy Levels of Polyatomic Transient Molecules*. The collection was supplemented in 1998 [Jacox 1998], and the entire collection is also available online [Jacox 2002]. Other spectroscopic data are spread throughout the literature.

Despite the availability of information, there are few computer programs available that simulate spectra of diatomic molecules. Two programs that simulate rovibronic spectra exist, but they only simulate rotational structure within one vibronic transition at a time and do not allow for the inclusion of other spectral features (atomic transitions, for example) [Western 2002, Robinson 1994].

The reason why there are so few simulation programs becomes apparent once one begins preparation for writing such a program. The rules governing the transitions are

complex and deviations from the most common simplifications are frequent (although such deviations are usually relatively small). Even the simplest system, the diatomic molecule, is quite complex (as will be discussed below).

In it's current form, RVESIM can simulate two of the main types of rovibronic spectra (see below) based on generalized energy, population and intensity formulae. However, it was written so that it will be relatively simple to alter and expand. It is well–documented, open–source (there is no cost for use or redistribution) and easily available (can be downloaded from the internet or obtained from the author for copying and shipment costs). Because of this, changes could be made or additions incorporated as needed. Details of its current capabilities and limitations are given in the next section.

6-2 Program Features

In its current form, the program calculates positions and intensities for rovibronic transitions in diatomic molecules. At present, it can only handle transitions from Hund's Case (a) to (a) and Case (b) to (b) [e. g., Atkins 1998, Herzberg 1989, Steinfeld 1985]. It does not yet simulate Hund's Cases (c) or (d). If the rotational distribution is thermal, the program will calculate relative rotational–level populations and transition intensities automatically. If the distribution is not thermal, it must be entered from an input file (see Section 6-3). Details concerning calculations of energy–levels, populations and transition intensities are given below in Section 6-4.

Any number of states of any number of diatomics can be simulated at the same time. The limit to the number of molecules and states is determined only by computational resources, primarily the amount of random access memory (RAM) available. Each electronic state can be assigned a separate temperature (currently, individual vibrational levels cannot be assigned separate temperatures). Alternatively, overall temperatures can be assigned to each diatomic molecule or for all diatomic species.

Table 6-2.1

Current Program Capabilities

Diatomic Molecules — States

Number of molecules and states is unlimited in program.

States must be Hund's Case a or Case b.

Any values of Λ, Ω and S may be used

Thermal rotational population distributions are calculated internally.

Alternate rotational distributions can be entered from file**.

Temperatures can be specified at global, molecule or state level.

Diatomic Molecules — Transitions

Hund's Case a→a and b→b for emission only.

Satellite bands can be assigned a constant intensity level

(level set to zero by default).

Hönl–London factors for rotational transition intensities can be turned on or off.

Electronic cascade is accounted for automatically.

Transitions calculated elsewhere can be included by input file.

Other Atoms or Molecules

Transition lines must be determined externally and included by input file.

Equipment-Specificity

Adjusts for photon–counting or calorimetric** detection methods.

Adjusts for detectors that scan or that step** from point to point.

Mimics apparatus resolution (triangular peak shape).

*** Indicates features that have not yet been adequately tested.*

Since the program was initially written to facilitate data analysis in studies of the photoproducts of dissociative recombination (which are not necessarily thermal), relative vibrational populations must be entered by the user, and are not assigned thermal distributions within the program. Each electronic and vibrational state of each molecule must also be assigned a relative population.

Atomic or other single–peak transitions can be entered from a file and included in the simulation. This feature allows easy import of positions and intensities calculated by other programs or obtained from the literature.

A program like this, ideally, should be capable of much more than the current program can do. Many additional features were considered at the design stage for inclusion later. Others are nearly complete but have not been activated or debugged. Table 6-6.1 lists these features and the relative ease with which they could be included. Table 6-2.1 lists the features already present and working.

6-3 Input and Output Files

The program requires a number of input files and creates a selection of output files. Details regarding the input and output files can be determined with a small amount of effort by inspection of the source code (see Appendix C-c). More extensive documentation will be available on the internet so long as web-space is available. A listing of the files and their contents is also given here.

Input Files:

Names of required files appear in **boldface**. Files that might become necessary if certain input values are used appear in *italics*. Files relating to features that are not yet working in the program appear in regular type. Input file formats are described in Appendix C-b.

Main: This is the master file for the simulation. It tells the program what sort of simulation to do and how to do it.

.rvesimconfig: This is an optional configuration file. If the program does not find this file in the directory from which it is called, it will write a brief message to that effect and continue to run. The file is mainly used during debugging after additions or changes have been made to the program.

State File: This file is necessary if the user requests a rovibronic simulation. It contains information about the states involved in the transitions to be simulated.

Transition File: This file is necessary if the user requests a rovibronic simulation. It contains information about the transitions the user wants the program to include.

Franck–Condon Factor File: There must be one of these files for each transition listed in the Transition File. The format is based on output from a Franck–Condon factor program written by Kent Ervin [Ervin 1993]. The file contains Franck–Condon factors for the transitions.

Rotational Distribution File: This file becomes necessary if the user specifies that the rotational populations are to be input from a file. It contains energy levels and relative populations for rotational levels in a given vibronic state.

Efficiency File: This file becomes required if the EFFICIENCY entry is set to any value other than 1. This file contains the efficiency curve for the experimental detection system that is used to obtain spectra to which simulation output will be compared.

Currently, this feature is not working and the only useful value for EFFICIENCY is 1.

Atomic File: This file becomes necessary if the user specifies inclusion of one or more files containing positions of discrete transitions. The file must contain transition wavelengths, relative populations and transition probabilities.

External Spectrum ("Other") File: This file is called "other" within the program. This file becomes necessary if the user specifies inclusion of external spectral features. The spectral features in this file will be added to the final simulation without modification for spectral resolution, intensity or optical system efficiency.

Experimental File: This file becomes necessary if the user requests the display of a graphical comparison between the simulation and the user's experimental data. The file contains the experimental spectrum to use for comparison.

Output Files:

Since the number of output files is potentially very large, the program creates sub–directories within the directory from which the program is called and places output files within those directories. In some cases, the directories, themselves, contain sub–directories. The following list of output files is categorized by the directory, and sub–directory if appropriate, where the output file is located.

In the main input file, the user must specify an OUTPREF, a prefix for all files and all first–level directories created by the program. The prefix may contain any characters that the computer's operating system will allow within file names. In the list that follows, OUTPREF is abbreviated to "PREF."

Directories within program–created directories (sub–directories) are not prefixed. Some files within directories created by the program contain the prefix and others do not. In the following, MOL is a variable for the name of a diatomic molecule, s is the variable representing a state and v represents a vibrational level.

Files placed within the directory from which the program is called:

PREF_parameter.txt: This file contains the parameters used by the program. It also contains a number of messages written by the program. For example, if a transition is omitted from the simulation, the omission is declared and briefly explained in the parameter file.

PREF_debug.txt: The purpose of this file is to assist in debugging and checking the program. This file is only written if the value of DEBUG in the configuration file, .rvesimconfig, is set to an integer value greater than -1. It will be an empty file unless one or more of the other debugflag values is less than the value of DEBUG. If all debugflag values are less than DEBUG, copious quantities of detailed statements regarding the program's progress will be written to this file. "Copious," in this instance, means a file with a size on the order of megabytes or gigabytes.

Files placed within the PREF_molecules/MOL directories:

MOL_s_rot.dat: This file contains energies (in wavenumbers) and relative populations, categorized by value of J or K (rotational quantum numbers) for each rotational level in each vibrational level in the thermal rotational distribution calculated for state s.

si–sj_vk–vl_NAT.dat: Transition frequencies and intensities calculated for each $J' \rightarrow J$ transition from vibrational level k of electronic state i to vibrational level l of electronic state j. These positions and intensities are sep-

arated into P, Q (as applicable) and R branches, with values of J and/or K indicated. The NAT stands for "native." These are the intensities for transitions arising from user–specified parameters (see the next entry).

si–sj_vk–vl_CAS.dat: The contents of this file are similar to the contents of the file above, but the intensities are due to electronic cascade. For example, if a user specifies to simulate the a'–a, d–a and e–a and a–X transitions of CO, a certain portion of the light observed for the a–X transition will be due to cascade from the a', d and e states. In that case, the a–X_v*k*–v*l*_CAS.dat files will contains only those portions of the a–X transition determined by the program to be due to transitions from the higher states.

Files placed within the PREF_simulations directory:

PREF_Sim_All: This file contains the final output from the entire simulation.

Atomic_lines: This file contains the portion of the final simulation due to atomic lines.

Mol_native_pop: This file contains the portion of the final simulation due to "native," or user–defined, diatomic molecular populations.

Mol_cascade_pop: This file contains the portion of the final simulation due diatomic molecular populations arising from transitions from higher states.

6-4 State Energies, Relative Intensities and Populations, and Selection Rules

The program was designed to balance precision with ease of use. Greater precision than what is currently available in the program is certainly possible. For example, Kovács [1969] has published a book containing energy levels and transition intensities for almost every imaginable type of transition. However, not only are there hundreds of formulae, but the formulae also often require parameters that are not routinely available in tabulations of spectroscopic data. For example, many of his formulae

require knowledge of a spin–orbit interaction coupling constant. While such information must certainly exist for a large number of diatomics, it is not generally available in convenient form. So, his formulae are not used in the program. Rather, the program is written so that the user needs only have a standard spectroscopic reference at hand [e. g., Herzberg 1989, NIST 2001a,b or Radzig 1985].

The equations used in the simulation come primarily from Herzberg [1989] with occasional augmentation or modification using information in Steinfeld's text [1985]. Only Hund's Cases a and b are modeled, for two reasons. Cases a and b are the two simplest cases: they do not require knowledge of the extent to which the electronic configuration of the molecule will couple to the rotational axis. Additionally, since cases c and d involve coupling to the axis of rotation, modeling them properly would require transition from case a or b as the rotational quantum numbers increase, which would involve the difficulties mentioned above regarding Kovács.

Obviously, the simulation can only be as good as the model upon which it is based, so, results from the simulation should be interpreted with care, especially if the molecules or states are unusual in some manner. For example, some electronic states of He_2, begin to transition to or toward Hund's Cases c or d as soon as there is rotational excitation. For these molecules the simulation produced by this program cannot be better than a good approximation. See Section 6-7 for further discussion on this.

The model is also only as good as the data used as input. For example, if Franck–Condon factors are calculated for high vibrational levels using Morse potentials, as is common, one should expect there to be errors in the simulated intensities and positions for those vibronic transitions. In the program, for example, the rotational constant, D_e, is calculated using an ideal–case equation (Eq. 2, below). Since this factor is usually very small, the error will be small, but could become significant for some cases.

Spectroscopic Constants:

The program uses standard spectroscopic constants for electronic states of diatomic molecules. For convenience, they are defined here, but definitions of them can be found in any spectroscopy text that deals with electronic transitions in small molecules.

ω_e: The fundamental vibrational frequency for the state.

$\omega_e\chi_e$: The vibrational anharmonicity for the state.

B_e: The principle rotational constant.

α_e: The constant describing the interaction between vibration and rotation in the molecule.

B_v: The rotational constant modified for vibrational interaction:

$$B_v = B_e - \alpha_e(v + \frac{1}{2}) \tag{58}$$

D_e: Rotational constant describing cetrifugal effects:

$$D_e = \frac{4B_e^3}{we^2} \tag{59}$$

β_e: A correction term for the vibrational interaction with the centrifugal distortion (see below). Herzberg [1989, p 108] offers an expression for β_e, but currently, the program always assumes β_e to be zero. It could be included quite easily, except that it is not widely tabulated and the correction is also generally very small.

D_v Correction for the centrifugal distortion due to the rotation of the molecule:

$$D_v = D_e + \beta_e(v + \frac{1}{2}) \tag{60}$$

J: The final rotational quantum number. This is the quantum number that has taken account of all relevant angular momenta.

K: The rotational quantum number that takes account of all relevant angular momenta except for electron spin.

Energy Levels (all energies are entered in cm^{-1}):

T_e: The term energy for the state. This is the energy at the bottom of the attractive potential well between the two atoms in the molecule.

T_0: The energy of the zeroth vibrational level.

G_v: The energy associated with vibrational level v in the molecule:

$$G_v = \omega_e(v + \frac{1}{2}) - \omega_e\chi_e(v + \frac{1}{2})^2 \tag{61}$$

$F_v(J)$: For Hund's Case a:

$$F_v(J) = B_v(J(J+1) - \Omega^2) - D_v J^2(J+1)^2 \tag{62}$$

For Hund's Case b:

$$F_v(J) = B_v K(K+1) - D_v K^2(K+1)^2 \tag{63}$$

Note that J is not included explicitly in the right–hand side of the equation. This is accurate. The program does not account for splitting between the various J's for each K. So, each J is assigned the energy associated with its parent value of K. The splitting between those levels could be added in later, but the effect is small (on the order of a few cm^{-1} for a large effect) and the equations needed change according to the type of electronic state.

$T_v(J)$ The energy level associated with rotational quantum J of vibrational state v of the relevant electronic state relative to the bottom of the potential well for the electronic ground state:

$$T_v(J) = T_e + G_v + F_v(J) \tag{64}$$

Thermal Rotational Populations

Populations in the program are all assigned values relative to one. The program is designed to stop including rotational states in the simulation after the state populations fall below a cutoff fraction of the most populated rotational state. The cutoff fraction is input by the user. To make the initial estimate of the number of rotational levels to include, the program uses a simple equation for the rotational energy levels. It is taken at $v = 0$ and centrifugal effects on the energy are neglected ($D_v = 0$):

$$Population_{estimate} \propto (2J+1) \exp\left(-\frac{F_0'(J)}{k_B T}\right) \tag{65}$$

where the form of $F_0'(J)$ depends on Hund's case (see Equations 5 and 6, above).

When the actual population for a given rotational level is calculated, the full form of the appropriate $F_v(J)$ is used:

$$Population \propto (2J+1) \exp\left(-\frac{F_v(J)}{k_B T}\right) \tag{66}$$

Note that since the pre–exponential factor is a statistical description of available orientations in space, it depends on whatever quantum number represents the total angular momentum in the system. So, for Hund's Cases (a) and (b), the important quantum number is J in the pre–exponential factor. Even when the value of F_v is calculated as an approximation based on K, as for Hund's Case (b) in this program, the exponential is multiplied by $(2J + 1)$.

Selection Rules:

The relevant selection rules are widely available [for example, Herzberg 1989]. But, again, for convenience, they are summarized and included here.

Global Selection Rules:

- $\Delta J = 0, \pm 1$ except that $J = 0 \not\leftrightarrow J = 0$
- $+ \leftrightarrow -$ only (the $\pm$ here refers to overall symmetry of the wavefunction and not the $\pm$ value assigned to Σ electronic states)

- For homonuclear diatomics:
  - $g \leftrightarrow u$ only
  - $s \leftrightarrow s$ and $a \leftrightarrow a$ only

Rules that apply to both Hund's Case (a) and Hund's Case (b):

- $\Delta\Lambda = 0, \pm 1$
- $\Delta S = 0$ (this rule often broken)
- For $\Lambda = 0 \leftrightarrow \Lambda = 0$, $+ \leftrightarrow +$ and $- \leftrightarrow -$ only ($\pm$ here refers to reflection symmetry through internuclear axis)

Rules applying only to Case a–a transitions:

- $\Delta\Sigma(spin) = 0$ only
- $\Delta\Omega = 0, \pm 1$
- For $\Omega = 0 \leftrightarrow \Omega = 0$, $\Delta J = \pm 1$ only
- Bands with $\Delta J = 0$ and $\Delta\Lambda = 0$ (but not for $\Lambda = 0 \leftrightarrow \Lambda = 0$):
  - Intensities for $\Delta J = 0$ fall off rapidly, with intensities decreasing approximately as $(1/J)\mathrm{e}^{-E_r/k_B T}$.

Rules applying only to Case b–b transitions:

- For $\Lambda = 0 \leftrightarrow \Lambda = 0$:
  - $\Delta K = \pm 1$ only.

  Else:
  - $\Delta K = 0. \pm 1$
- Bands with $\Delta K = 0$ and $\Delta\Lambda = 0$ (but not for $\Lambda = 0 \leftrightarrow \Lambda = 0$):
  - Intensities for $\Delta K = 0$ fall off rapidly as the value of K increases, with intensities decreasing approximately as $(1/K)\mathrm{e}^{-E_r/kT}$.
- Intensities for bands where ($\Delta J \neq \Delta K$) also have intensities that drop off very rapidly with increasing K. The bands are called 'satellite bands.' In this program, their intensites are assigned a value of zero unless the user specifies some other value.

Figure 6-4.1

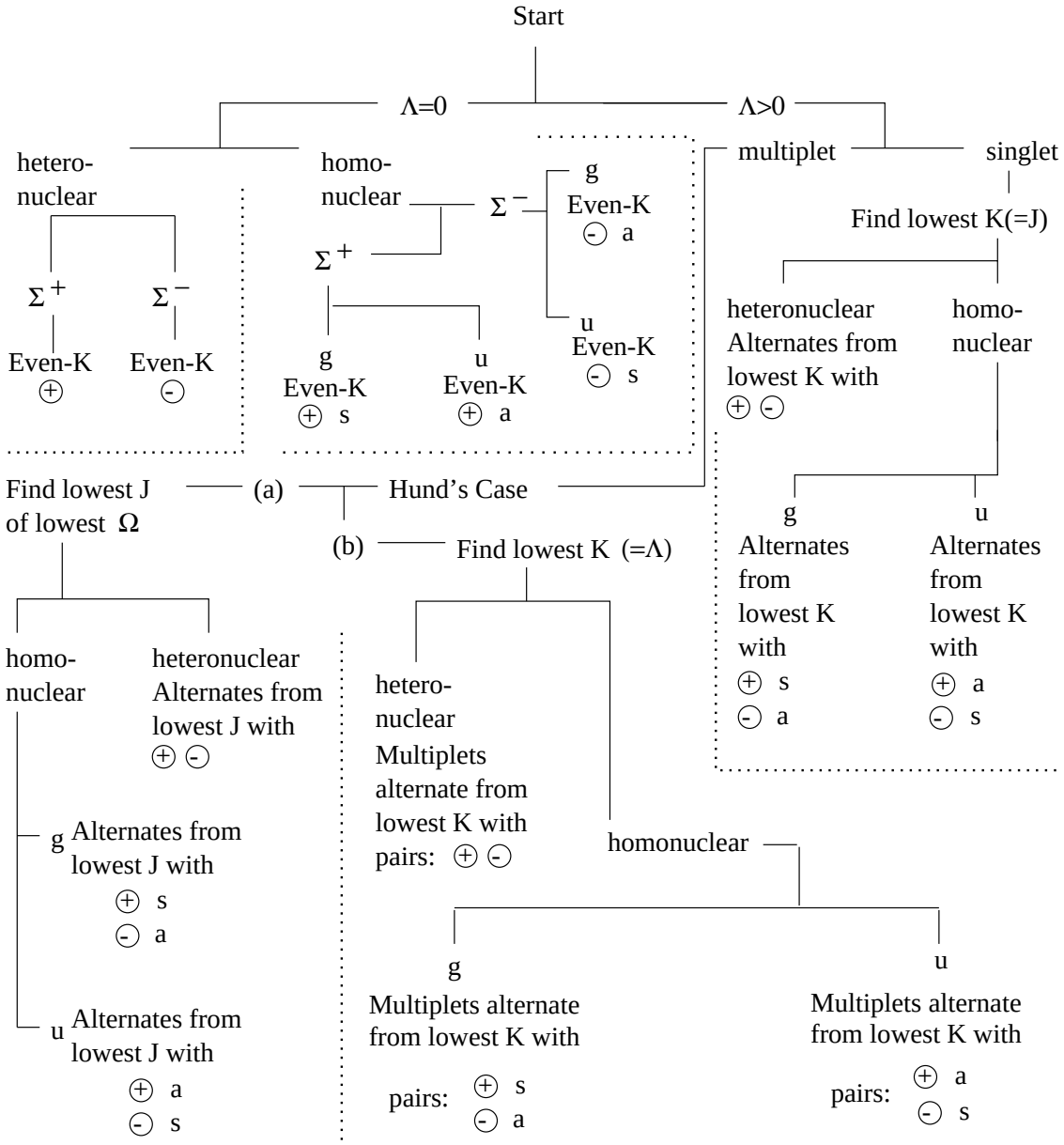


Figure 6-4.1: Flowchart for determining symmetry properties of rotational states. Chart shows nuclear symmetry for bosons only. For fermions, interchange the 'a' and 's' designations, but not the '+' and '-' designations. Λ refers to the electronic orbital angular momentum for the state. J and K are rotational quantum numbers. Encircled + or - signs refer to the overall symmetry of a given rotational state. g and u refer to gerade and ungerade states. s and a stand for symmetric and antisymmetric.

Determination of symmetry and nuclear effects:

In some cases, the overall symmetry of a given state must be determined (the selection rules will not always account for this automatically). The task is complicated further if the diatomic is homonuclear and nuclear spin statistics must also be considered. The necessary steps are outlined in the flowchart in Figure 6-4.1. Note that the flowchart only specifies symmetries for bosons (nuclei with integral spins). In the case of fermions (nuclei with half–integral spins), 's' and 'a' designations, but not any others, should be interchanged.

Transition Intensities:

All factors associated with transition intensities: transition probabilities, Franck–Condon Factors, Hönl–London Factors and species population are assigned values relative to one. It is up to the user to determine how to scale the overall simulation to match experimental observation.

The relative populations of each portion of the simulation are given in the input file. Populations of individual diatomic species, i, are given by a simple statistical formula:

$$Pop_i = f_{rve} * f_{mol} * f_{st} * f_{vib} * f_{rot} \qquad 67$$

f_{rve} represents the relative weight of the rovibronic portion of the simulation compared to other parts of the simulation; f_{mol}, is the relative population of the molecule; f_{st} is the relative population of each electronic state for a given molecule; f_{vib} is the population of one vibrational level relative to other vibrational levels within its electronic state; and, f_{rot} is the relative population of the rotational level to other rotational levels in that vibrational level. All but the relative rotational level must be included in the input file. If desired, the relative rotational levels can be input as well. Relative populations of other species included in the simulation are calculated in an analogous and obvious fashion.

Table 6-4.1

Relative Hönl–London Factors for Emission

$\Lambda' =$	$J' =$	**Relative Hönl–London Factor**
$\Lambda'' + 1$	$J'' - 1(P)$	$\frac{(J+1-\Lambda)(J+2-\Lambda)}{2(2J^2+3J+1)}$
	$J''(Q)$	$\frac{(J+\Lambda)(J+1-\Lambda)}{2J(2J+1)}$
	$J'' + 1(R)$	$\frac{(J+\Lambda)(J-1+\Lambda)}{2J(2J+1)}$
Λ''	$J'' - 1(P)$	$\frac{(J+1+\Lambda)(J+1-\Lambda)}{2J^2+3J+1}$
	$J''(Q)$	$\frac{\Lambda^2}{J(J+1)}$
	$J'' + 1(R)$	$\frac{(J+\Lambda)(J-\Lambda)}{J(2J+1)}$
$\Lambda'' - 1$	$J'' - 1(P)$	$\frac{(J+1+\Lambda)(J+2+\Lambda)}{2(2J^2+3J+1)}$
	$J''(Q)$	$\frac{(J-\Lambda)(J+1+\Lambda)}{2J(J+1)}$
	$J'' + 1(R)$	$\frac{(J-\Lambda)(J-1-\Lambda)}{2J(2J+1)}$

Table 6-4.1: Hönl–London factors [Herzberg 1989] scaled so that each represents a probability, out of one, that the upper state will decay to each of the possible lower states. Λ is the electronic orbital angular momentum quantum number and J is the rotational quantum number. A single prime ($'$) designates an upper state and a double prime ($''$), a lower state.

The relative transition intensity, $Irel_{i,j}$, then, from any rotational level i to another rotational level j is given by:

$$Irel_{i,j} = Pop_i * TFrel * FCFrel * HLrel_{i,j} \qquad 68$$

where $TFrel$ is the fraction of the upper electronic state that is expected to radiate, $FCFrel$ is the relative Franck–Condon factor and $HLrel_{i,j}$ is the relative Hönl–London factor for the transition. The relative Hönl–London factors (see Table 6-4.1) are derived from the general–case Hönl–London factors found in Herzberg [Herzberg

1989, p208-209]. The relative Franck–Condon factor, $FCFrel$, contains the effects due to the transition frequency (raised to either the third or fourth power as required by the detection technique) at the band origin. The program is currently being rewritten so that the effect of transition frequency is included at the level of each rotation→rotation transition, which is more accurate.

6-5 Data Structures

Data in this program are stored in arrays, structures or combinations of arrays and structures. An array is an addressed list of data storage locations within a computer's memory. Structures are named collections of specified types of storage location. Arrays only contain one type of information — integer, float, character, etc. Structures can contain any number of different types of data. For example, a structure might contain a string, two float values, and an array of integers (or any other combination possible within the programming language). Structures can also contain other structures or arrays of other structures, a capability often exploited in this program. In this section, the more complex data structures employed in the program are described.

Arrays of data in this program frequently occupy 3, 4 or more dimensions. Regardless the number of dimensions required for convenient organization of the stored data, data in this program are physically stored linearly, in a 1D array of memory addresses. In order to access the information in a manner convenient to the needs of programming, the data is indexed in a regular fashion

Figure 6-5.1 graphically illustrates the indexing for a four–dimensional array. This particular 4D array stores Franck–Condon factors for a series of $v' \rightarrow v''$ transitions. A 4D array is necessary when the upper and lower electronic states are Hund's Case a and where one or both states can exist with multiple values of Ω, and where the Franck–Condon factors are different for each $\Omega' \rightarrow \Omega''$ transition. In this case, it is

necessary to generate indexing such that the Franck–Condon factor for each $v' \rightarrow v''$ transition of each $\Omega' \rightarrow \Omega''$ transition occupies a unique address in the array. That makes 4 dimensions. The following equation exemplifies the manner in which individual storage locations within multi–dimensional arrays are addressed within the program.

$$Address = \Omega'_i * N_{\Omega''} * N_{v'} * N_{v''} + \Omega''_j * N_{v'} * N_{v''} + v'_k * N_{v''} + v''_\ell \qquad 69$$

where Ω'_i, Ω''_j, v'_k and v''_ℓ are the indexes (integers starting with zero) corresponding to the current values of the high–state and low–state values of Ω and v, respectively, and $N_{\Omega''}$, $N_{v'}$ and $N_{v''}$ are the total number of low state Ω's and high and low state vibrational quanta. The equation above can be simplified algebraicly, but the current form makes its meaning more readily understood.

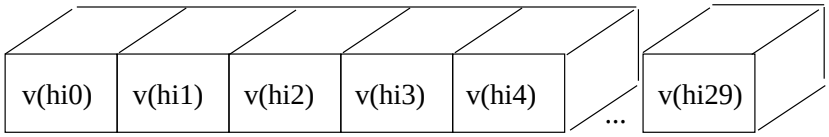
Sometimes, because of the greater flexibility, structures are used to store multi-dimensional arrays rather than the linear addressing described in the last paragraph. As an illustrative example, consider the way the program keeps track of states and transitions for a certain molecule. Here is the 'Molecule' structure used in the program (with comments):

```
typedef struct{
      int cp, cT; /* change flags for relative population
                  and temperature */
      char Mol[41]; /* name of molecule */
      double pop; /* relative population for molecule */
      int states; /* number of electronic states for
                  this molecule */
      State *s; /* array of electronic state information */
      int trans; /* number of electronic transitions for
                  this molecule */
      Trans *t; /* array of electronic transition information */
      double T; /* temperature if specified at molecule level */
}Molecule;
```

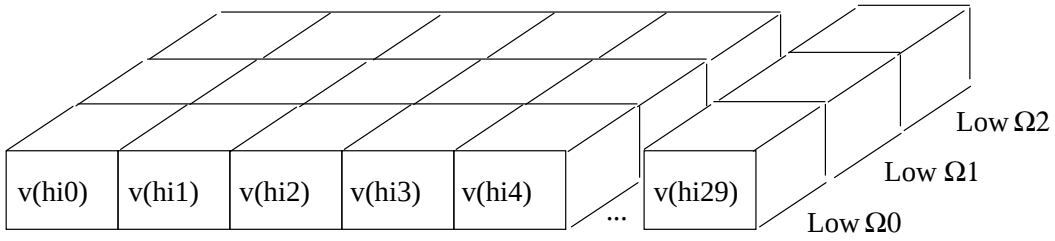
After this declaration is made, "Molecule" is now a data type, (e. g., integer, character, etc.) to be used as needed in the program.

Figure 6-5.1

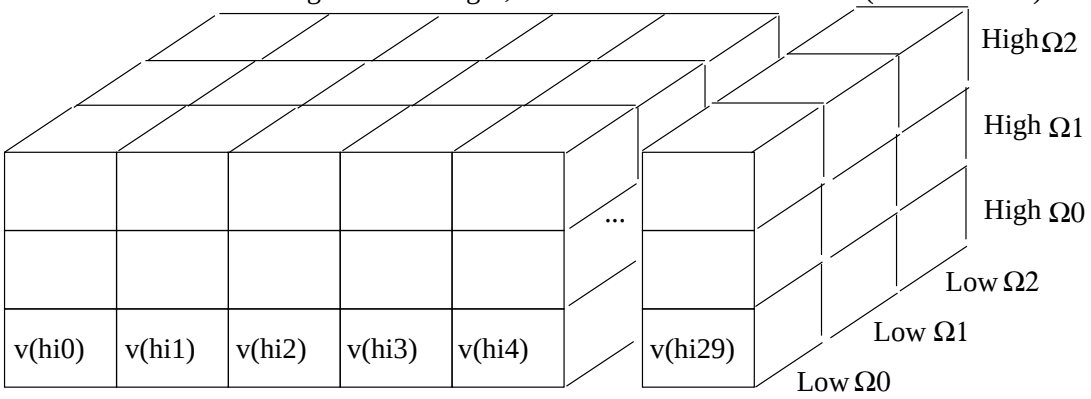
Each of these boxes contains 30 Franck-Condon Factors (2 Dimensions)



If there are several Low-State Omegas, the data is structured like this (3 Dimensions):



If there are also several High-State Omegas, the data is structured like this (4 Dimensions):



Each address above is given by: (current high Omega number)*(number low omegas)*30*30
+ (current low Omega number)*30*30
+ (high state vibrational quantum number index)*30
+ (low state vibrational quantum number index)

Figure 6-5.1: Graphical representation of the indexing used in multi–dimensional arrays in the program. Each of the boxes is, itself, a one–dimensional array (the 30 Franck–Condon factors). So, the row of boxes at the top is two dimesions and so forth. Dimensionalities higher than 4 can be visualized by considering each box to be a multi–dimensional array, itself.

The Molecule structure, itself, contains two structures. Each molecule considered in the simulation can be expected to exist in a number of electronic states — at least two if there is one transition — and might also undergo a number of transitions. The stars next to the s and the t indicate that those variables are "pointers." A pointer is simply a marker — an address within the computer's memory. Once the program is running and it reads the number of states to assign to a given molecule, it will allocate exactly the amount of memory needed to store the information. After it allocates that memory, the pointer will be pointing to an array. The data types State and Trans are structures similar to the Molecule structure except that they contain, not surprisingly, information pertaining to the states for that molecule and the transitions it should undergo. A portion of the State structure is:

```
typedef struct {
      char Name[21]; /* The letter designation for
                  the state (a', B, c, etc.).  */
      double pop; /* relative population for this state */
      char Case[11]; /* Hund's Case:  a or b */
      ...
      rotset *r; /* pointer to set of rotational
                  distribution info.
}State;
```

Note that the State structure also contains a pointer, *r. There will be one "rotset" for each $v' \rightarrow v''$ transition. Each rotset, itself, contains arrays corresponding to rotational energy levels and relative populations.

Accessing information in this sort of structure is quite straightforward. Let us expand upon an example from before. Assume that the molecules He_2, N_2 and CO are being simulated, and that they appear in the state file in that order. If the variable name 'MOL' is assigned to the Molecule structure, then the name "He2" would be assigned to MOL[0].Mol, "N2" would be assigned to MOL[1].Mol, and "CO", to MOL[2].Mol.

Further, assume that the a', d, e, a, A and X states of the CO molecule are part of the simulation, and that they appear in that order in the transition file. In this case state a' would be assigned position 0 in the array of states, state d would be assigned position 1, and so on. Thus, the value assigned to MOL[2].s[2].Name would be "e." Since the chain of square–braces and periods can become lengthy, short names for pointers (e. g., "s" and "r") are recommended.

These structures are also explained in the supporting documentation for the program and are well documented within the code. Any given piece of data can be located with just a little effort and modified if necessary.

6-6 Possible Changes and Additions

As was mentioned in earlier sections, the program, in its current form, can only perform a limited number of tasks. But, it was designed to be changed and to be somewhat easy to change. Table 6-6.1 lists a number of proposed changes and additions.

6-7 Comparisons of Program Output with Experimental Data

The calculations that the program does have been tested thoroughly. Several small test systems were run and the results compared to equivalent calculations performed in Microsoft Excel. The instructions in the program or, if necessary, the spreadsheet, were adjusted until the two agreed. At this point, it was assumed that the program was performing the calculations it was intended to perform. Obviously, having performed the intended calculations doesn't mean that the intended calculations were appropriate.

Output from the program has also been compared to experimental data acquired for this study and to data acquired elsewhere [Brown 1971 & 1973]. Results of these comparisons are summarized in a series of tables and figures in this section. Captions

on the figures and tables are detailed and will not be repeated here. A number of other comparisons can also be found in Chapter 4 of this manuscript.

The comparisons show that the program performs its assigned tasks. As is expected, though, when reality deviates from the model used, the simulation only approximates the experimental data.

Table 6-6.1

Suggested Program Additions

See below for key to difficulty levels

Capability	**Difficulty Level**
Diatomic Molecules — States	
Hund's Case's c & d	3
Temperature specification at vibrational level	4
Addition of higher-precision spectroscopic constants	2
Energy level splitting — simple equations	4
Energy level splitting — complex equations	5
Energy levels and populations by user-defined equations	6(5)
Energy levels by ab-initio calculation	6(5)
Diatomic Molecules — Transitions	
Hund's Case a↔b	3
Other transitions (c↔c, b↔d, etc.)	4
Variable intensity for satellite bands	4
Additional options for Hönl–London Factors	2
Cascade between vibrational or rotational levels	5
Add absorption as well as emission	4
Bound↔unbound transitions	4
Raman transitions	6(5)
Transitions by ab–initio calculations	6(5)
Polyatomic Molecules	
Energy levels and populations	6(6)
Transitions	6(6)

Equipment– and Experiment– Specificity	
Transmission Efficiency curve from values in file	1
Transmission Efficiency curve from standard equation	3
Transmission Efficiency curve from user–defined equation	6(5)
Alternate line shapes	3
Doppler bradening, natural linewidth, etc.	2
Other Transitions	
Include spectral features from external sources	2
User–Friendliness	
Inclusion of interactive mode	5
Addition of graphical–user–interface (GUI)	6(6)
Inclusion of graphical display of results	5
Addition of experimental data to graphical display	3

Key to difficulty levels. Each item is followed by an approximate description of the work needed to accomplish that task.

Level Sample set of tasks.

0 Values that are already present in the program must be included in calculations or outputs that are already written into the program.

1 Existing code must be called or simple new code written and called. The result will be used in calculations or outputs that already exist or are very simple to write or to copy and modify.

2 Existing code must be copied and modified for a new task. Decision–making statements (e. g., "if" or "switch") must be augmented or written anew, with the result included in a simple calculation or output that might or might not already exist.

3 Existing code must be copied and modified in a complex manner. Some of the modifications might be subtle. Significant debugging is expected. Decision–

making statements must be written or augmented, with results included in calculations or output.

4 This level describes two situations. In the first, the necessary modification is level 2 or 3, but there are other level 2 or 3 modifications that must be completed first. In the second, this modification is non–trivial and changes must be made in several of the existing functions (which generally complicates debugging).

5 This level is similar to 3 or 4 except that the amount of work needed is greatly increased. For example, a complex function must be written from scratch or an existing function greatly expanded or modified.

6 This number indicates that this feature has not been designed or is in the earliest stages of design. The number in parentheses is an estimate of the difficulty level. If the number in parentheses is 6, the amount of work required is uncertain.

Table 6-7.1

Example Comparison of Simulated Transition Frequencies to Experimental Data[*]

He_2: $e^3\Pi_g(v=0) \rightarrow a^3\Sigma_u^+(v=0)$

	P		Q		R	
K''	ν_{sim}	$\nu_{sim-exp}$	ν_{sim}	$\nu_{sim-exp}$	ν_{sim}	$\nu_{sim-exp}$
1			21506.31	-0.12	21534.98	0.31
3	21459.17	0.31	21502.15	-0.13	21559.40	1.30
5	21423.21	1.36	21494.68	-0.11	21580.30	2.88
7	21384.23	2.89	21483.93	-0.14	21597.63	5.03
9	21342.36	4.99	21469.96	-0.14	21611.33	7.61
11	21297.78	7.60	21452.81	-0.15	21621.36	10.64
13	21250.66	10.74	21432.57	-0.20	21627.69	14.23
15	21201.18	14.38	21409.33	-0.22	21630.32	18.14
17	21149.56	18.50	21383.19	-0.26	21629.25	22.37
19	21096.00	23.02	21354.26	-0.32	21624.49	26.85
21	21040.75	27.97	21322.69	-0.24	21616.08	31.53
23	20984.04	33.32	21288.61	-0.44	21604.06	36.30
25	20926.15	39.11	21252.19	-0.49	21588.51	41.18
27	20867.34	45.30	21213.60	-0.60		

[*][Brown 1971]

Table 6-7.2

*Example Comparison of Simulated Transition Frequencies to Experimental Data**

He_2: $e^3\Pi_g(v=2) \rightarrow a^3\Sigma_u^+(v=1)$

	P		Q		R	
K''	ν_{sim}	$\nu_{sim-exp}$	ν_{sim}	$\nu_{sim-exp}$	ν_{sim}	$\nu_{sim-exp}$
1			23006.36	0.03	23033.26	0.47
3	22959.72	0.28	23000.05	-0.04	23053.76	1.44
5	22921.67	1.22	22988.71	-0.11	23069.02	2.89
7	22878.87	2.56	22972.38	-0.21	23078.99	4.86
9	22831.48	4.43	22951.10	-0.36	23083.61	7.31
11	22779.64	—	22924.93	-0.55	23082.84	10.11
13	22723.55	9.50	22893.94	-0.80	23076.66	13.29
15	22663.38	12.59	22858.23	-1.15	23065.05	16.80
17	22599.34	—	22817.91	-1.48	23048.02	20.53
19	22531.65	—	22773.08	—	23025.58	24.51
21	22460.55	—	22723.88	—	22997.78	28.43
23	22386.27	—	22670.46	—	22964.65	32.47
25	22309.09	—	22612.98	—	22926.26	36.50

*[Brown 1971], "—" indicate unreported values.

Table 6-7.3

*Example Comparison of Simulated Transition Frequencies to Experimental Data**

He_2: $J^1\Delta_u(v=0) \rightarrow B^1\Pi_g(v=0)$

	P		Q		R	
K''	ν_{sim}	$\nu_{sim-exp}$	ν_{sim}	$\nu_{sim-exp}$	ν_{sim}	$\nu_{sim-exp}$
1					21655.71	-5.89
2			21626.55	-6.00	21669.13	-17.94
3	21582.83	-5.91	21625.42	-17.83	21682.19	-23.64
4	21567.19	-18.19	21623.97	-23.96	21694.94	-51.63
5	21551.28	-23.61	21622.25	-51.27	21707.41	-45.00
6	21535.16	-52.03	21620.32	-45.71	21719.68	-94.55
7	21518.90	-44.97	21618.26	-93.77	21731.81	-67.12
8	21502.61	-95.03	21616.16	-68.39	21743.91	-140.11
9	21486.37	-66.91	21614.11	-138.77	21756.05	-88.20
10	21470.30	-140.66	21612.24	-90.07	21768.37	-184.61
11	21454.52	-88.12	21610.65	-182.61	21780.98	-106.60
12	21439.17	-185.23	21609.50	-109.20	21794.02	-225.65
13	21424.40	-106.46	21608.92	-222.85	21807.64	—
14	21410.38	-226.32	21609.09	-124.31	21822.00	—
15	21397.27	-120.56	21610.18	—	21837.28	—
16	21385.27	-261.95	21612.37	-133.79	21853.67	—
17	21374.57	-128.85	21615.87	—	21871.36	—
18	21365.39	-290.22	21620.88	-135.82	21890.57	—
19	21357.95	-129.64	21627.64	—	21911.52	—
20	21352.49	—	21636.37	-128.60	21934.45	—

*[Brown 1973], "—" indicate unreported values. See Figure 6-7.3 and caption.

Figure 6-7.1

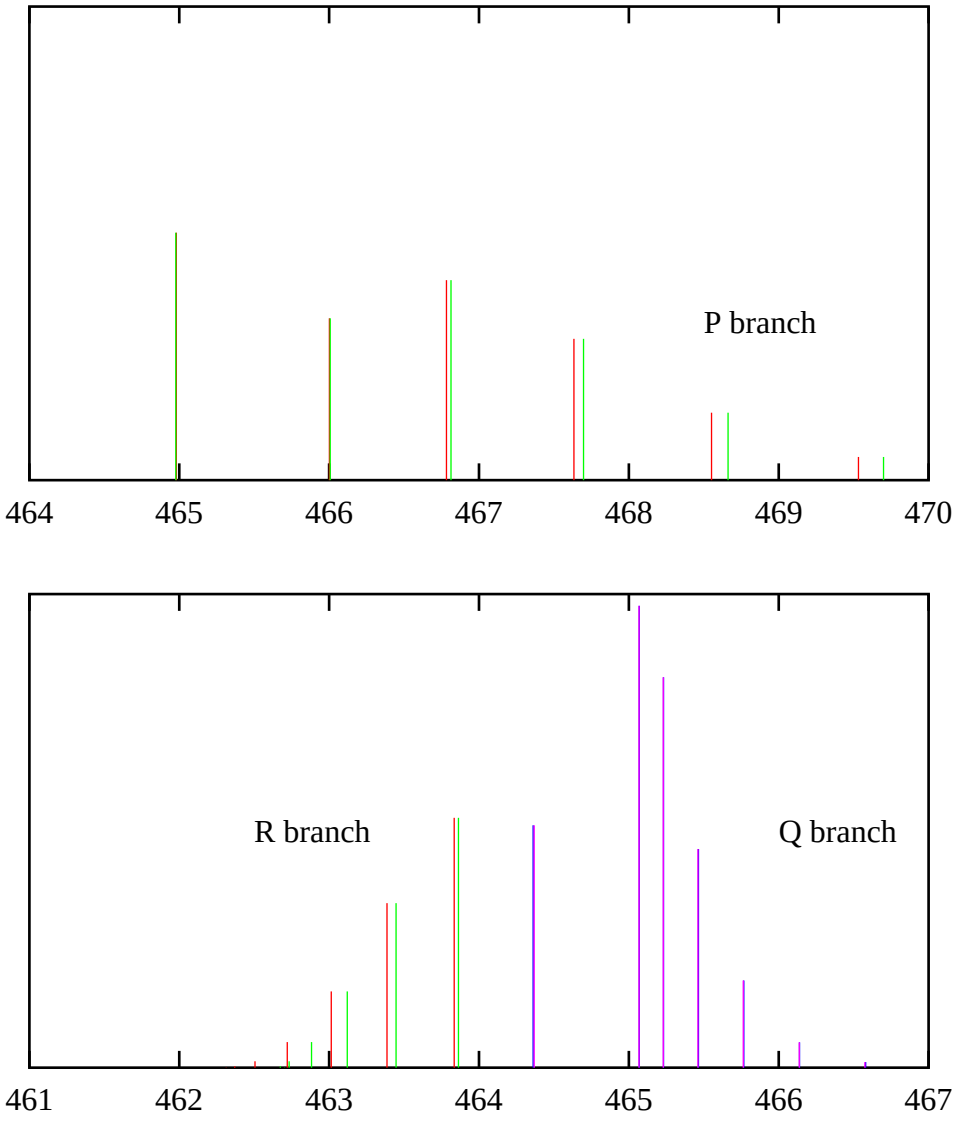


Figure 6-7.1: Graphical comparison of simulated line positions (red and blue lines) with experimental positions (green, magenta) [Brown 1971] for the $e^3\Pi_g(v = 0) \rightarrow a^3\Sigma_u^+(v = 0)$ transition of He_2. The x–axis is wavelength in nm. Y-axes are scaled similarly, but the intensity is arbitrary. Since Brown and Ginter do not report detailed line intensities, the relative intensities are those determined by the simulation. Most of the positions for the Q branch are nearly coincident, and the differences are not resolved on this graph. See also Figure 6-7.4.

Figure 6-7.2

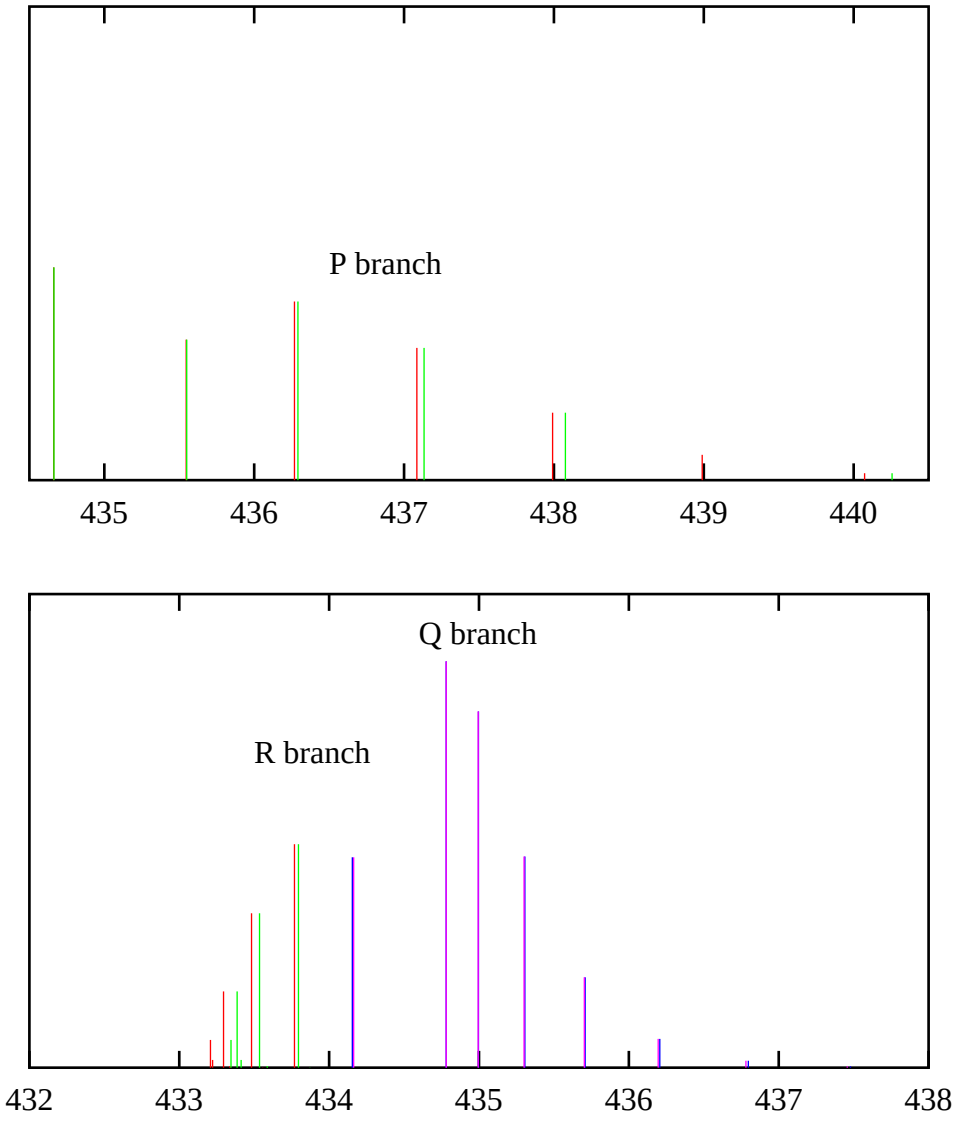


Figure 6-7.2: Graphical comparison of simulated line positions (red and blue lines) with experimental positions (green and magenta) [Brown 1971] for the $e^3\Pi_g(v=2) \rightarrow a^3\Sigma_u^+(v=1)$ transition of He_2. The x–axis is wavelength in nm. Y-axes are scaled similarly, but the intensity is arbitrary. Since Brown and Ginter do not report detailed line intensities, the relative intensities are those determined by the simulation. Some of the positions for the Q branch are nearly coincident, and the differences are not resolved on this graph.

Figure 6-7.3

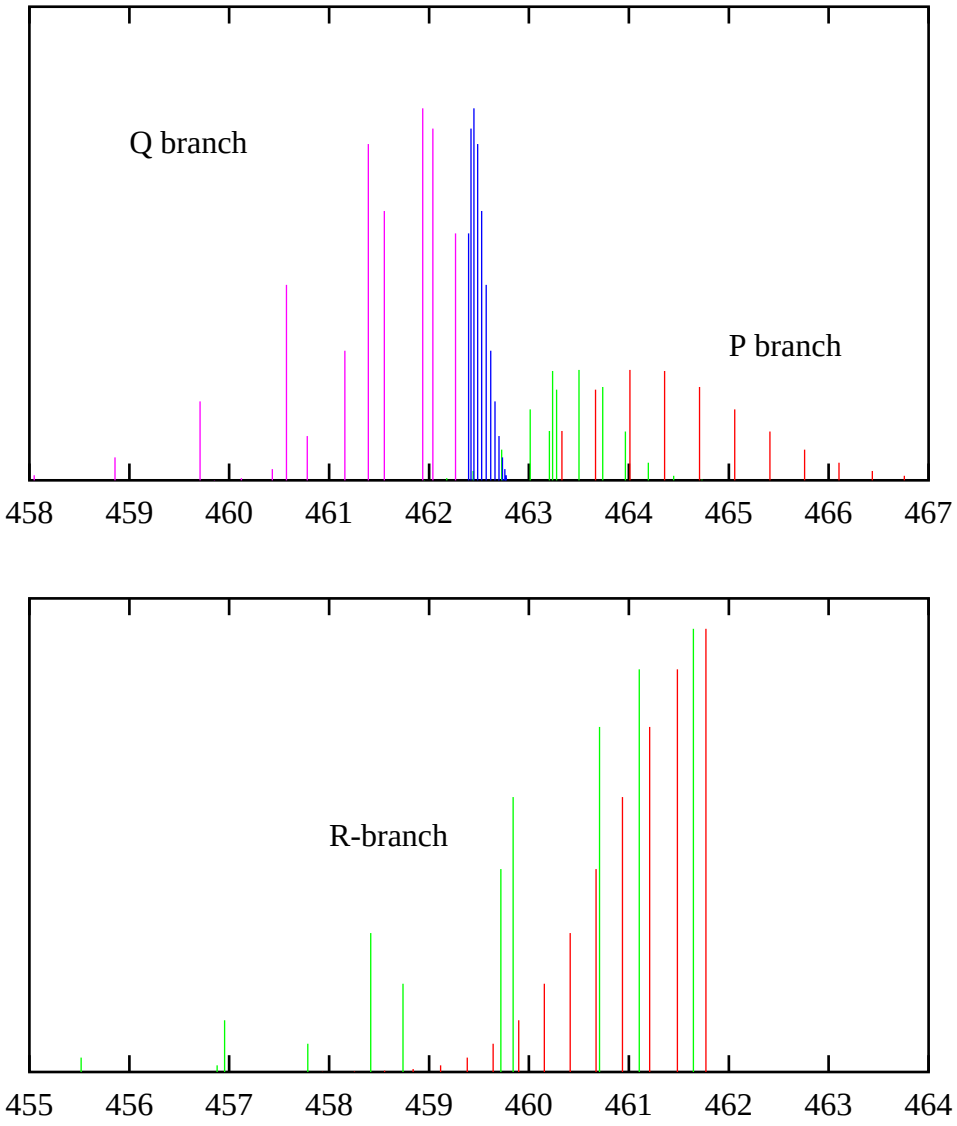


Figure 6-7.3: Graphical comparison of simulated line positions (red and blue lines) with experimental positions (green, magenta) [Brown 1973] for the $J^1\Delta_u(v = 0) \rightarrow B^1\Pi_g(v = 0)$ transition of He_2. Values for ω_e, $\omega_e\chi_e$ and α_e, not reported for this transition, were set to zero in the simulation; this accounts for most of the poor agreement. The x–axis is wavelength in nm. Y-axes are scaled similarly, but the intensity is arbitrary. Since Brown and Ginter do not report detailed line intensities, the relative intensities are those determined by the simulation.

Figure 6-7.4

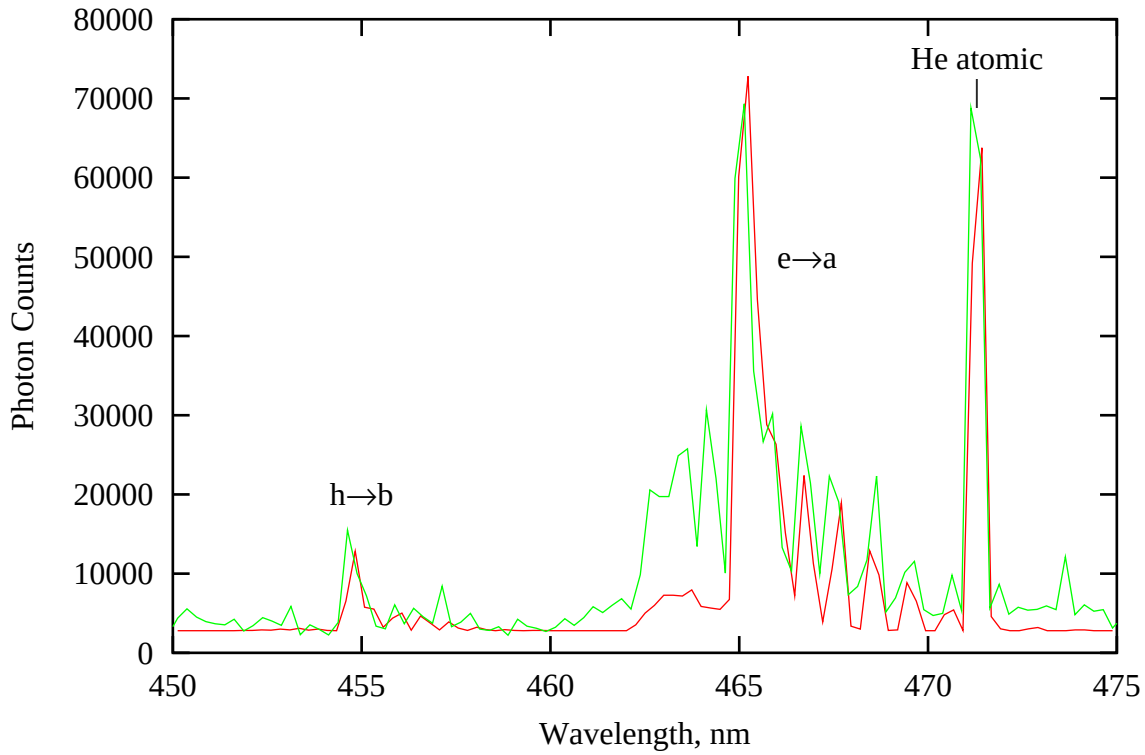


Figure 6-7.4: Comparison of simulation (red, solid) to experimental data obtained during this study (green, dashed). Transitions included in the simulation are $h^3\Sigma_u^+$ $(v=0) \rightarrow b^3\Pi_g(v=0)$ and $e^3\Pi_g(v=0) \rightarrow a^3\Sigma_u^+(v=0)$ of He_2 and $4s^3S_1 \rightarrow 2p^3P^0_{0,1,2}$ of atomic He. Emissions from He_2 transitions other than $e \rightarrow a$ and $h \rightarrow b$ are present in the experimental spectrum, but have not been included in this simulation. But, inclusion of these states in the simulation is problematic since spectroscopic constants ω_e, $\omega_e\chi_e$ and α_e are not reported for the J states (see also Figure 6-7.3). The difference in intensity between the simulated and the experimental spectrum is due to an indexing error within the program. The intensity should be higher in the simulation (see Figure 6-7.1). The simulation resolution is 0.5 nm FWHM, which is approximately the same as the experimental resolution.

CHAPTER 7

SUMMARY AND FUTURE DIRECTIONS

The work presented in this dissertation is summarized, and suggestions for continuations of the work are made.

7-1 Summary

A detailed study of the standard experimental procedure for producing and classifying optical emissions resulting from dissociative electron ion recombination (DR) has produced unexpected results. Of primary importance among these results is the observation that emissions which cannot result solely from DR are quenched by the addition of an electron–attaching gas. Investigation of the plasma in which the recombinations occur has revealed emissions from excited states of He, He_2, Ar and Ar^+ that are also quenched in this way. A satisfactory mechanism for this behavior has not been found, but several hypotheses, and means for testing some of the hypotheses, are suggested. Means for identifying recombination emissions in the absence of a satisfactory mechanism are also discussed. In order to facilitate future studies, a survey of spectra containing emissions that are likely to be due to recombination is presented.

Emissions from the recombination of CO_2^+ were observed according to the protocol suggested by the studies mentioned above. Rotational temperatures and relative vibrational and electronic populations have been determined for a number of products of that recombination. These results indicate that photoproducts of DR can be positively identified, and that other systems should be treated similarly.

A computer program, RVESIM (rotational vibrational electronic simulation), has been written that simulates rovibronic (rotational, vibrational, electronic) spectra. This makes identification and quantification of emissions simpler and more reliable. The inclusion of effects due to cascade from higher electronic states allows nascent populations of lower–lying electronic states to be determined. The program can be altered, and users of the program are encouraged to make changes and additions as needed or desired.

A photographic and spectroscopic study of the flow characteristics near the reactant gas inlet port, where emissions are detected, has also been made. The determination of relative electronic state populations is dependent on the time the species spend in viewing range of the spectrometers and, therefore, on the throughput of the injected reagent gas. An empirical dependence of injection plume size as a function of reagent gas throughput is given, but the relationship of plume size to observation time is neither obvious nor trivial. A computer program that simulates the mixing of neutral gases in the injection region has been developed, and suggestions for additions to the program are made.

7-2 Future Directions

The mechanism by which the plasma emissions (and also the CO^+ emissions) are affected by the attaching gas needs to be determined in more detail. Inherent in determining this mechanism will be determining the mechanism by which the emissions are maintained within the plasma. When these mechanisms are identified, it will be possible for the presence of unwanted species (excited states of He, He_2, Ar and Ar^+) in the afterglow to be diminished or eliminated. This will make recombination studies more straightforward. The populations of these species could also be lessened by enhancing the creation of cold He^+ ions at the ionization source (see the end of Section 4-1). In either case, the survey of recombination experiments presented in

Section 4-2 should be repeated in more detail, making use of the information presented in Chapter 4.

The computer program, RVESIM, needs to be expanded to make it easier to use, include many other types of transition and to make it more generally applicable. A list of suggested changes and additions to the program is given in Chapter 6.

A means for more accurately determining the time emitting species spend in the viewing region should be devised. This will need to be done empirically, but general models should be developed (for example, those independent of reagent type). These models can include or be independent of the program presented in Section 5-3. A camera of some sort, e. g., with a charge coupled device (CCD) array, which can record spectra at each position in the flow, might prove useful for such studies.

REFERENCES

Abdallah Jr., J.; Palmer, N.; Gekelman, W.; Maggs, J.; Clark, R. E. H. "Time–Dependent Kinetics Model for a Helium Discharge Plasma." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(32). pp 1001-1008. **1999**.

Abramowitz, M.; Stegun, I. A., Eds. "Bessel Functions of Fractional Order" in *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, Inc.. New York. **1965**.

Adams, N. G.; Church, M. J.; Smith, D. "An Experimental and Theoretical Investigation of the Dynamics of a Flowing Afterglow Plasma." *Journal of Physics D: Applied Physics*. V(8). pp 1409-1422. **1975**.

Adams, N. G.; Smith, D.; Eds: Lindinger, W.; Märk, T. D.; Howorka, F. "Temperature Dependence of Positive–Ion Molecule Reactions" in *Swarms of Ions and Electrons in Gases*. Springer–Verlag. Wien. **1984**.

Adams, N. G.; Smith, D. "Gas Phase Ionic Reactions." *Science Progress*. V(71). N(281). pp 91-116. **1987**.

Adams, N. G.; Smith, D. "Flowing Afterglow and SIFT" in *Techniques for the Study of Ion–Molecule Reactions. Farrar, J. M.; Saunders Jr., W. H., Eds.*. John Wiley & Sons Ltd.. New York. **1988 a**.

Adams, N. G.; Smith, D. "Ionic Reactions in Atmospheric, Interstellar and Laboratory Plasmas." *Contemporary Physics*. V(29). N(6). pp 559-578. **1988 b**.

Adams, N. G.; Herd, G. R.; Geoghegan, M.; Smith, D.; Canosa, A.; Gomet, J. C.; Rowe, B. R.; Queffelec, J. L.; Morlais, M. "Laser Induced Fluorescence and Vacuum Ultraviolet Spectroscopic Studies of H–Atom Production in the Dissociative Recombination of Some Protonated Ions." *Journal of Chemical Physics*. V(94). N(7). pp 4852-4857. **1991**.

Adams, N. G.; Eds: Adams, N. G. and Babcock, L. M. *Advances in Gas Phase Ion Chemistry*. JAI Press, Ltd.. Greenwich, CT. **1992**.

Adams, N. G.; Babcock, L. M. "Vibrational Excitations in the Products of Electron–Ion Recombination: A Test of Theory for Ions of Interstellar Significance." *Astrophysical Journal*. V(434). pp 184-187. **1994 a**.

Adams, N. G.; Babcock, L. M. "Optical Emissions from the Dissociative Recombination of N_2H^+ and HCO^+." *Journal of Physical Chemistry*. V(98). N(17). pp 4564-4569. **1994 b**.

Adams, N. G.; Babcock, L. M.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Identity and Degree of Excitation of the Products of Dissociative Electron–Ion Recombination" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Airy, G. B. *An Elementary Treatise on Partial Differential Equations Designed for the Use of Students in the University*. Macmillan and Co.. Cambridge. **1873**.

Allison, D.C.; Browne, J. C.; Dalgarno, A. "Collision–induced De–excitation of Metastable Helium." *Proceedings of the Physical Society of London*. V(89). pp 41-44. **1966**.

Amitay, Z.; Baer, A.; Dahan, M.; Levin, J.; Vager, Z.; Zajfman, D.; Knoll, L.; Lange, M.; Schwalm, D.; Wester, R.; Wolf, A.; Schneider, I.; Suzor–Weiner,

A. "Dissociative Recombination of Vibrationally Excited HD^+: State–Selective Experimental Investigation." *Physical Review A*. V(60). N(5). pp 3769-3785. **1999**.

Andersen, L. H.; Eds.: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Electron Scattering Experiments" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Anderson, D. N.; Bernhardt, P. A. "Modeling the Effects of an H_2 Gas Release on the Equatorial Ionosphere." *Journal of Geophysical Research*. V(83). N(A10). pp 4777-4790. **1978**.

Arfa, M. B.; Lescop, B.; Cherid, M.; Fanjoux, G. "Penning Ionization of the CO_2 Molecule by He* ($2\,^1S$) Metastable Ions." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(31). pp 4813-4820. **1998**.

Atkins, P. W. *Physical Chemistry, 6^{th} Ed.*. W. H. Freeman and Co.. New York. **1998**.

Baer, T.; Guyon, P. M. "Autoionization and Isotope Effect in the Threshold Photoelectron Spectrum of $^{12}CO_2$ and $^{13}CO_2$." *Journal of Chemical Physics*. V(85). N(9). pp 4765-4778. **1986**.

Bailey, R. T.; Cruickshank, F. R. "Laser Studies of Vibrational, Rotational, and Translational Energy Transfer" in *Gas Kinetics and Energy Transfer, 3^{rd} Ed.*. Publisher unknown.. City unknown.. **1978**.

Banavar, J. R.; Koplik, J.; Willemsen, J. F.; Eds.: Landau, D. P.; Mon, K. K.; Schüttler, H.-B. "Molecular Dynamics of Slow Viscous Flows" in *Computer Simulation Studies in Condensed Matter Physics III, Springer Proceedings in Physics 53*. Springer–Verlag. Berlin. **1991**.

Banks, P. M.; Williamson, P. R.; Raitt, W. J. "Space Shuttle Glow Observations." *Geophysical Research Letters*. V(10). N(2). pp 119-121. **1983**.

Baras, F.; Mansour, M. M.; Garcia, A. L.; Mareschal, M. "Particle Simulation of Complex Flows in Dilute Systems." *Journal of Computational Physics*. V(119). pp 94-104. **1995**.

Bardsley, J. N. "Configuration Interaction in the Continuum States of Molecules." *Journal of Physics B: Atomic and Molecular Physics*. V(1). pp 349-364. **1968 a**.

Bardsley, J. N. "The Theory of Dissociative Recombination." *Journal of Physics B: Atomic and Molecular Physics*. V(1). pp 365-380. **1968 b**.

Bardsley, J. N.; Biondi, M. A. "Dissociative Recombination." *Advances in Atomic and Molecular Physics*. V(6). pp 1-57. **1970**.

Bashkin, S.; Stoner Jr., J. O. *Atomic Energy Levels and Grotrian Diagrams*. American Elsevier Publishing Co, Inc.. New York. **1975**.

Berlande, J.; Cheret, M.; Deloche, R.; Gonfalone, A.; Manus, C. "Pressure and Electron Density Dependence of the Electron–Ion Recombination Coefficient in Helium." *Physical Review A*. V(1). N(3). pp 887-896. **1970**.

Bartschat "Electron–Impact Excitation of Helium from the $1\,^1S$ and $2\,^3S$ States." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(31). pp L469-L476. **1998**.

Baruch, M. C.; Sturrus, W. G.; Gibson, N. D.; Larson, D. J. "Electric–Field Effects in Photodetachment from Cl^- and S^- Ions in a Microwave Field." *Physical Review A*. V(45). N(5). pp 45-. **1992**.

Bastian, G.; Queffelec, J.-L. "Étude et Réalisation d'Un Spectrographe Sous Vide Destiné a l'Étude de l'Émission dans l'U. V. Lointain d'Un Jet de Plasma en Atmospheère Raréfiée." *Revue de Physique Appliquée*. V(3). pp 243-249. **1968**.

Bates, D. R. "Dissociative Recombination when Potential Energy Curves Do Not Cross." *Journal of Physics B: Atomic, Optical and Molecular Physics*. V(25). pp 5479-5488. **1992**.

Bates, D. R.; Eds: Rowe, B. R.; Mitchell, J. B. A.; Canosa, A. "Polyatomic Ion Dissociative Recombination" in *Dissociative Recombination: Theory, Experiment and Applications*. Plenum Press. New York. **1993 a**.

Bates, D. R. "Vibrational Excitation of Products of Dissociative Recombination." *Mon(thly). Not(ices). (of the) R(oyal). Astron(omical). Soc(iety)..* V(263). pp 369-374. **1993 b**.

Bates, D. R. "Dissociative Recombination: Crossing and Tunneling Modes." *Advances in Atomic, Molecular and Optical Physics*. V(34). pp 427-497. **1994**.

Bernardin, D.; Sero–Guillaume, O. E.; Sun, C. H. "Multispecies 2D Lattice Gas with Energy Levels: Diffusive Properties." *Physica D*. V(47). pp 169-188. **1991**.

Bhushan, K. G. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Metastable Lifetime Measurements with an Electrostatic Ion–Trap: NO^+" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Birge, R. T. "The Band Spectra of Carbon Monoxide." *Physical Review*. V(28). pp 1157-1181. **1926**.

Blöcker, J. H.; Reinsch, E. -A.; Rosmus, P.; Werner, H. -J.; Knowles, P. J. "Theoretical Radiative Transition Probabilities of the CS^+ ion." *Chemical Physics*. V(147). pp 99-108. **1990**.

Boffard, J. B.; Lagus, M. E.; Anderson, L. W.; Lin, C. C. "Electron–Impact–Excitation Cross Sections Out of the $2\,^3S$ Metastable Level of Helium at High Energies." *Physical Review A*. V(59). N(5). pp 4079-4082. **1999**.

Bohme, D. K.; Adams, N. G.; Mosesman, M.; Dunkin, D. B.; Ferguson, E. E. "Flowing Afterglow Studies of the Reactions of the Rare–Gas Molecular Ions He_2^+, Ne_2^+, and Ar_2^+ with Molecules and Rare–Gas Atoms." *Journal of Chemical Physics*. V(52). N(10). pp 5094-5101. **1970**.

Bolden, R. C.; Hemswort, R. S.; Shaw, M. J.; Twiddy, N. D. "Measurement of Penning Ionization Cross Sections for Helium 2^3S Metastables Using a Steady-State Flowing Afterglow Method." *Journal of Physics B: Atomic and Molecular Physics*. V(3). N(1). pp 61-. **1970**.

Brau, C. A.; Ewing, J. J. "Emission Spectra of XeBr, XeCl, XeF, and KrF." *Journal of Chemical Physics*. V(63). N(11). pp 4640-4647. **1975**.

Brieger, L.; Bonomi, E. "A Stochastic Cellular Automaton Simulation of the Non–Linear Diffusion Equation." *Physica D*. V(47). pp 159-168. **1991**.

Brosa, U.; Stauffer, D. "Vectorized Multisite Coding for Hydrodynamic Cellular Automata." *Journal of Statistical Physics*. V(57). N(1 & 2). pp 399-403. **1989**.

Brosa, U. "Direct Simulation of a Permeable Membrane." *Le Journal de Physique*. V(51). N(11). pp 1051-1053. **1990**.

Brosa, U.; Stauffer, D. "Simulation of Flow Through a Two–Dimensional Random Porous Media." *Journal of Statistical Physics*. V(63). N(1 & 2). pp 405-. **1991**.

Brown, C. M.; Ginter, M. L. "Spectrum and Structure of the He_2 molecule: Characterization of the Singlet and Triplet States Associated with the UAO's 4s, 4dσ, 4dπ, and 4dδ.." *Journal of Molecular Spectroscopy*. V(46). pp 256-275. **1973**.

Brown, C. M.; Ginter, M. L. "Spectrum and Structure of the He_2 Molecule VI: Characterization of the States Associated with the UAO's 3pπ and 2s." *Journal of Molecular Spectroscopy*. V(5). pp 302-316. **1971**.

Bueche, F. J. *Introduction to Physics for Scientists and Engineers, 4^{th} Ed.*. McGraw–Hill Book Co.. New York. **1986**.

Burke, P. G.; West, J. B., Eds. "Ab Initio Methods for Electron–Molecule Collisions" in *Electron–Molecule Scattering and Photoionization*. Plenum Press. New York. **1988**.

Burshtein, A. I.; Krissinel, E.; Mikhelashvili; M. S. "Binary Photoionization Followed by Charge Recombination." *Journal of Physical Chemistry*. V(98). pp 7319-7324. **1994**.

Butler, J. M.; Babcock, L. M.; Adams, N. G. "Effects of Deuteration on Vibrational Excitation in the Products of the Electron Recombination of HCO^+ and N_2H^+." *Molecular Physics*. V(91). pp 81-90. **1997**.

Caledonia, G. E.; Holtzclaw, K. W.; Krech, R. H.; Sonnenfloh, D. M.; Leone, A.; Blumberg, W. A. M. "Mechanistic Investigations of Shuttle Glow." *Journal of Geophysical Research*. V(98). N(A3). pp 3725-3730. **1993**.

Callear, A. B. "An Overview of Molecular Energy Transfer in Gases." *Gas Kinetics and Energy Transfer*. V(3). pp (unknown). **1978**.

Capelle, G. A.; Broida, H. P. "Lifetimes and Quenching Cross Sections of $I_2(B\,^3\Pi^+_{Ou})$." *Journal of Chemical Physics*. V(58). N(10). pp 4212-4222. **1973**.

Carata, L.; Orel, A.; Suzor–Weiner, A. "Dissociative Recombination of He_2^+ Molecular Ions." *Physical Review A*. V(59). N(4). pp 2804-2812. **1999**.

Carata, L. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Core Excited Resonances in the Dissociative Recombination of CH^+ and CD^+" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000 a**.

Carata, L. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Dissociative Recombination of He_2^+ Molecular Ions" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000 b**.

Carrington, A.; Leach, C. A.; Marr, A. J.; Pyne, C. H.; Shaw, A. M.; Viant, M. R.; West, Y. D. "Near–dissociation Microwave Spectra of Rare–gas Diatomic Ions." *Chemical Physics Letters*. V(212). N(5). pp 473-479. **1993**.

Carroll, P. K. "Structure of the Triplet Bands of CO." *Journal of Chemical Physics*. V(36). N(11). pp 2861-2869. **1962**.

Castaño, F.; de Juan, J.; Martinez, E. "The Calculation of Potential Energy Curves of Diatomic Molecules: The RKR Method.." *Journal of Chemical Eductation*. V(60). N(2). pp 91-93. **1983**.

Chakrabarti, S.; Sasseen, T. P.; Lampton, M.; Bowyer, S. "Observations of Terrestrial far UV Emissions by the Faust Telescope." *Geophysical Research Letters*. V(20). N(6). pp 535-538. **1993**.

Chauvrin, R. "Explicit Periodic Trend of van der Waals Radii." *Journal of Physical Chemistry*. V(96). N(23). pp 9194-9197. **1992**.

Chenoweth, D. R. *Isothermal Viscous/Knudsen Blowdown of a Container with Nearly Constant Outgassing and Back–Pressure*. Sandia Laboratories. Livermore, CA. **1972**.

Child, M. S.; Essén, H.; LeRoy, R. J. "An RKR–like Inversion Procedure for Bound–Continuum Transitions Intensitites." *Journal of Chemical Physics*. V(78). N(11). pp 6732-6740. **1983**.

Chow, K. W.; Smith, A. L.; Waggoner, M. G. "Absorption Coefficients of Helium Between 599 and 610 Å; Transition Moment for He_2 $A^1\Sigma^+_u \leftarrow X^1\Sigma^+_g$." *Journal of Chemical Physics*. V(55). N(9). pp 4208-4213. **1971**.

Cliffe, K. A.; Kingdon, R. D.; Schofield, P.; Stopford, P. J. "Lattice Gas Simulation of Free–Boundary Flows." *Physica D*. V(47). pp 275-280. **1991**.

Collins, C. B.; Johnson, B. W.; Shaw, M. J. "Study of Excitation Transfer in a Flowing Helium Afterglow Pumped with a Tuneable Dye Laser. I. Measurement

of the Rate Coefficient for Selected Quenching Reactions Involving He(5^2P)*." *Journal of Chemical Physics*. V(57). N(12). pp 5310-5316. **1972 a**.

Collins, C. B.; Johnson, B. W. "Study of Excitation Transfer in a Flowing Helium Afterglow Pumped with a Tuneable Dye Laser. II. Measurement of the Rate Coefficient for the Rotational Relaxation of He $_2(3p^3\Pi_g)^*$." *Journal of Chemical Physics*. V(57). N(12). pp 5317-5321. **1972 b**.

Cornubert, R.; D'Humiéres, D.; Levermore, D. A "Knudsen Layer Theory for Lattice Gases." *Physica D*. V(47). pp 241-259. **1991**.

Dalgarno, A "Dissociative Recombination in Astrophysical Environments" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Datz, S. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Three–Body Breakup Dynamics in Dissociative Recombination" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Davies, P. G.; Flower, D. R. "Electron Capture From Metastable He Atoms by He^{2+} Ions." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(31). pp 3639-3646. **1998**.

Decker, B. Personal communication. **1993**.

Dekiya, H.; Tsuji, M.; Nishimura, Y. "Emission Spectra Produced from the Reactions of Ne^+ and Ar^+ Ions with OCS at Thermal Energy." *Chemical Physics Letters*. V(100). N(6). pp 494-498. **1983**.

Depree, C. Personal communication. **2000**.

Dixon, R. N. "The carbon monoxide flame bands." *Proceedings of the Royal Society A*. V(275). pp 431-446. **1963**.

Djurić, N. "Ionic Fragment Production in Dissociative Electron–Molecular Ion Collisions" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Dotan, I.; Viggiano, A. A. "Temperature, Kinetic Energy, and Rotational Temperature Dependencies for the Reactions of Ar^+ $(^2P_{3/2})$ with O_2 and CO." *Chemical Physics Letters*. V(209). N(1, 2). pp 67-71. **1993**.

Drake, G. W. F., Ed. *Atomic, Molecular and Optical Physics Handbook*. American Institute of Physics. Woodbury, NY. **1996**.

Drukarev, G. F.; Translated by Chomet, S.; Translation assisted by: Hasted, J. B. "General Equations Describing the Collision of an Electron with an Arbitrary Atom or Ion" in *The Theory of Electron–Atom Collisions*. Academic Press. London. **1965**.

Dubrulle, B.; Frisch, U.; Hénon, M.; Rivet, J.-P. "Low–Viscosity Lattice Gases." *Physica D*. V(47). pp 27-29. **1991**.

Dudeck, M.; Poissant, G.; Rowe, B. R.; Queffelec, J. L.; Morlais, M. "Plasma Diagnostics by Langmuir Probes and UV Absorption." *Journal of Physics D: Applied Physics*. V(16). pp 995-1005. **1983**.

Duley, W. W.; Williams, D. A. "IR Emission from Vibrationally Excited Molecules as a Probe of Chemistry in Cold, Dark, Interstellar Clouds." *Monthly Notices of the Royal Academy of Sciences*. V(257). pp 13p-14p. **1992**.

Dunlop, P. J.; Bignell, C. M. "Diffusion Coefficients and Thermal Diffusion Factors for He–CO_2, He–N_2O and He–COS." *Ber(eichnung?). Bunsenges(?). Phys(ische?). Chem(ie?)..* V(99). N(1). pp 77-79. **1995**.

Dunning, T. H.; Hay, J. P. "The Covalent and Ionic States of the Rare Gas Monofluorides." *Journal of Chemical Physics*. V(69). N(1). pp 134-149. **1978**.

Dünweg, B.; Landau, D. P. "Phase Diagram and Critical Behavior of the Si–Ge Unmixing Transition: A Monte Carlo Study of a Model with Elastic Degrees of Freedom." *Physical Review B*. V(48). N(19). pp 14182-14197. **1993**.

Dupree, A.; Goldberg, L. "Radiofrequency Recombination Lines." *Annual Review of Astronomy and Astrophysics*. V(?). N(200?). pp 231-. **1970**.

Eisberg, R. M. "Collision Theory" in *Fundamentals of Modern Physics*. John Wiley & Sons.. New York. **1967**.

Eland, J. H. D.; Berkowitz, J. "Formation and Predissociation of CO_2^+ ($\tilde{C}\ ^2\Sigma_g^+$)." *Journal of Chemical Physics*. V(6). N(15). pp 2782-2787. **1977**.

Engler, V. C. "Berechnung von Übergangswahrscheinlichkeiten zwischen Quasi - Adiabatisch - Transformierten Spezieszustánden under Verwendung Anharmonischer Oszillatorfunktionen." *Zeitschriff Physiche Chemie*. V(265). N(6 S.). pp 1193-1200. **1984**.

Ernst, M. H. "Mode–Coupling Theory and Tails in CA Fluids." *Physica D*. V(47). pp 198-211. **1991**.

Errea, L. F.; Harel, C.; Jouin, H.; Méndez, L.; Pons, B.; Riera, A. "Common Translation Factor Method." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(27). pp 3603-3634. **1994**.

Ervin, K. Personal communication. **1993**.

Ewing, J. J.; Brau, C. A. "Emission Spectrum of XeI* in Electron–Beam–Excited Xe/I_2 mixtures." *Physical Review A*. V(12). N(1). pp 129-132. **1975**.

Fairbairn, A. R. "Band Strengths in Forbidden Transitions: The Cameron Bands of CO." *J(ournal). (of) Quant(?). Spectrosc(opic?). Radiat(ive?). Transfer*. V(10). pp 1321-1328. **1970**.

Federer, W.; Dobler, W.; Howorka, F.; Lindinger, W.; Durup–Ferguson, M.; Ferguson, E. E. "Collisional Relaxation of Vibrationally Excited NO+(ν) Ions." *Journal of Chemical Physics*. V(79). pp 1543. **1983**.

Ferguson, E. E.; Fehsenfeld, F. C.; Schmeltekopf, A. L. "Dissociative Recombination in Helium Afterglows." *Physical Review*. V(138). N(2A). pp A381-A385. **1965**.

Ferguson, E. E.; Fehsenfeld, F. C.; Schmeltekopf, A. L. "Flowing Afterglow Measurements of Ion–Neutral Reactions." *Advances in Atomic and Molecular Physics*. V(5). N(1). pp 1-56. **1969**.

Fernando, R. P.; Smith, I. W. M. "Relaxation of NO($\nu = 1$) by Radical Species." *Journal of the Chemical Society, Faraday Transactions 2*. V(77). pp 459-468. **1981**.

Ferrenberg, A. M. Personal communication. **1996**.

Flannery, M. R.; Vrinceanu, D.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Recombination at Ultra Cold Energies" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Foley, B. L.; Adams, N. G.; Lee, H. S. "Identification of Radiative Emissions Resulting from Electron Recombination of O_2H^+ and N_2OH^+ Ions." *Journal of Physical Chemistry*. V(97). N(20). pp 5218-5223. **1993**.

Foley, B. L.; Ferrenberg, A. M.; Adams, N. G.; Eds.: Landau, D. P.; Mon, K. K.; Schüttler, H.-B. "Simulation of a Reacting Plasma: Preliminary Results" in *Computer Simulation Studies in Condensed Matter Physics IX*. Springer–Verlag. Berlin. **1997**.

Fox, J. L.; Dalgarno, A. "Electron Energy Deposition in Carbon Dioxide." *Planetary and Space Science*. V(27). pp 491-502. **1979**.

Fox, J. L.; Hac, A. "Velocity Distributions of C Atoms in CO^+ Dissociative Recombination: Implications for Photochemical Escape of C from Mars." *Journal Of Geophysical Research–Space Physics*. V(104). N(A11). pp 24729-24737. **1999**.

Fox, J. L. "Applications of Velocity Distributions of Atomic Products in Dissociative Recombination" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Frisch, U.; Hasslacher, B.; Pomeau, Y. "Lattice–Gas Automata for the Navier–Stokes Equation." *Physical Review Letters*. V(56). N(14). pp 1505-1508. **1986**.

Frisch, U. "Relation Between the Lattice–Boltzmann Equation and the Navier–Stokes Equations." *Physica D*. V(47). pp 231-232. **1991**.

Gerling, R. W.; Eds: Landau, D. P.; Mon, K. K.; Schüttler, H.-B. "Classification of Cellular Automata" in *Computer Simulation Studies in Condensed Matter Physics III: Springer Proceedings in Physics 53*. Springer–Verlag. Berlin. **1991**.

Gianturco, F. A.; Scialla, S.; Eds: Burke, P. G.; West, J. B. "Electron Scattering by Polyatomic Molecules: Recent Advances in Theory and Calculations" in *Electron–Molecule Scattering and Photoionization*. Plenum Press. New York. **1988**.

Ginter, M. L. "The Spectrum and Structure of the He_2 Molecule Part II: Characterization of the Singlet States Associated with the UAO's 3s and 2pπ.." *Journal of Molecular Spectroscopy*. V(17). pp 224-239. **1965 a**.

Ginter, M. L. "The Spectrum and Structure of the He_2 Molecule Part III: Characterization of the Triplet States Associated with the UAO's 3s and 2pπ.." *Journal of Molecular Spectroscopy*. V(18). pp 321-343. **1965 b**.

Ginter, M. L. "Spectrum and Structure of the He_2 Molecule IV: Characterization of the Singlet and Triplet States Associated with the UAO's 3dσ, 3dπ and 3dδ.." *Journal of Chemical Physics*. V(45). N(1). pp 248-262. **1966**.

Ginter, M. L.; Battino, R. "Potential–Energy Curves for the He_2 Molecule.." *Journal of Chemical Physics*. V(52). N(9). pp 4469-4474. **1970**.

Gloersen, P.; Dieke, G. H "Molecular Spectra of Hydrogen and Helium in the Infrared." *Journal of Molecular Spectroscopy*. V(16). pp 191-204. **1965**.

Glosík, J.; Plašil, R.; Zakouřil, P.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Measurement of Recombination of Cluster Ions" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Golde, M. F. "Interpretation of the Oscillatory Spectra of the Inert–Gas Halides." *Journal of Molecular Spectroscopy*. V(58). pp 261-273. **1975**.

Golde, M. F.; Ho, Y.-S. "Chemi–ionization Reactions of State–Selected Electronically Excited Ar($^3P_0$) and Ar($^3P_2$) Atoms." *Journal of Chemical Physics*. V(82). N(7). pp 3160-3168. **1985**.

Golde, M. F.; Ho, G. H.; Tao, W.; Thomas, J. M. "Collisional Deactivation of $N_2(A^3\Sigma_u^+,\ v = 0-6)$ by CH_4, CF_4, H_2, H_2O, CF_3Cl and CF_2HCl." *Journal of Physical Chemistry*. V(93). N(3). pp 1112-1119. **1989**.

Golubkov, M. G. et al.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Difficulties Inherent in the Theory of Low–Temperature Dissociative Recombination and Possible Ways of their Obviation" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Gougousi, T.; Johnsen, R.; Golde, M. "Yield Determination of OH(v=0,1) Radicals Produced by the Electron–Ion Recombination of H_3O^+ Ions." *Journal of Chemical Physics*. V(107). N(7). pp 2430-2439. **1997**.

Gougousi, T.; Johnsen, R.; Golde, M. F. "Yield Determination of OH (v=0,1) Radicals Produced by the Electron–Ion Recombination of Protonated Molecules." *Journal of Chemical Physics*. V(107). N(7). pp 2440-2443. **1997**.

Graedel, T. E.; Crutzen, P. J. "The Changing Atmosphere." *Scientific American*. V(Sep.). pp 59-68. **1989**.

Grebowsky, J. M.; Taylor, H. A.; Pharo III, M. W.; Reese, N. "Thermal Ion Perturbations Observed in the Vicinity of the Space Shuttle." *Planetary and Space Science(?)*. V(35). N(4). pp 501-513. **1987**.

Green, N. J. B.; Pimblott, S. M.; Brocklehurst, B. "Spin Effects on Spur Kinetics: Re–encounters and the Independent Pairs Approximation." *Journal of the Chemical Society Faraday Transactions*. V(91). pp 223-229. **1995**.

Greer, W. A. D.; Pratt, N. H.; Stark, J. P. W. "Spacecraft Glows and Laboratory Luminescence: Evidence for a Common Reaction Mechanism." *Geophysical Research Letters*. V(20). N(8). pp 731-734. **1993**.

Guberman, S. L.; Goddard, W. A., III "Nature of the Excited States of He_2.." *Physical Review A*. V(12). N(4). pp 1203-1221. **1975**.

Guberman, S. L. "Dissociative Recombination of the Ground State of N_2^+." *Geophysical Research Letters*. V(18). N(6). pp 1051-1054. **1991**.

Guberman, S. L.; Goddard, W. A., III "On the Origin of Energy Barriers in the Excited States of He_2.." *Chemical Physics Letters*. V(14). N(4). pp 460-465. **1972**.

Guna, M.; Simons, L.; Hardy, K.; Peterson, J. "Branching Ratios Among Three Product Channels of the Dissociative Recombination Reaction in Ar_2^+." *Physical Review A*. V(60). N(1). pp 306-313. **1999**.

Gundel, L. A.; Setser, D. W.; Clyne, M. A. A.; Coxon, J. A.; Nip, W. "Rate Constants for Specific Product Channels from Metastable $Ar(^3P_{2,0})$ Reactions and Spectrometer Calibration in the Vacuum Ultraviolet." *Journal of Chemical Physics*. V(64). N(11). pp 4390-4410. **1976**.

Habs, D.; et al. "Title Unknown." *Nucl. Instrum. Methods in Phys. Res. B*. V(43). pp 390+. **1989**.

Halkier, A.; Roberts, M.; Linderberg, M. "Towards a Theory of Dissociative Recombination." *Pramana — Journal of Physics*. V(50). N(6). pp 547-554. **1998**.

Hanly, J. R.; Koffman, E. B. *Problem Solving and Program Design in C, 2^{nd} Ed.*. Addison–Wesley Publishing Co.. Reading, MA. **1996**.

Hardalupas, Y.; Taylor, A. M. K. P.; Whitelaw, J. H. "Particle Dispersion in a Vertical Round Sudden–Expansion Flow." *Philosophical Transactions of the Royal Society of London A*. V(341). pp 411-442. **1992**.

Hardy, K. A. "Kinematic Explanation of the Observation of Very Narrow Linewidths from Dissociative–Recombination Reactions." *Physical Review A*. V(58). N(2). pp 1256-1260. **1998**.

Hardy, K. A. "Dissociative Recombination in Gas Discharges" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Harland, P. W.; Vallance, C.; Eds.: Adams, N. G.; Babcock, L. M. "Positive Ion–Electron Impact Ionization Cross Sections: Theory and Experiment" in *Advances in Gas–Phase Ion Chemistry*. JAI Press, Inc.. Greenwich, CT. **1998**.

Hasslacher, B. "Spontaneous Curvature in a Class of Lattice Gas Field Theories." *Physica D*. V(47). pp 19-23. **1991**.

Hazeltine, R. D.; Waelbroek, F. L. *The Framework of Plasma Physics*. Perseus Books. Reading, MA. **1998**.

Headrick, L. B.; Fox, G. W. "New Measurements on the Fourth Positive Bands of Carbon Monoxide." *Physical Review*. V(35). pp 1033-1037. **1930**.

Hearn, C. H.; Joens, J. A. "The Near U. V. Absorption Spectrum of CS_2 and SO_2 at 300K." *J(?). Quant(?). Spectrosc(?). Radiat(?). Transfer*. V(45). N(2). pp 69-75. **1991**.

Heidner III, R. F.; Kasper, J. V. K "Observation of Vibrationally Excited H_2 in Active Hydrogen." *Journal of Chemical Physics*. V(51). pp 4163-4164. **1969**.

Helm, H.; Cosby, P. C. "Product Branching in Predissociation of the $e^1\Pi_u$, $e'^1\Sigma_u^+$ and $b'^1\Sigma_u^+$ States of N_2." *Journal of Chemical Physics*. V(90). N(8). pp 4208-4215. **1989**.

Herbst, E.; Terzieva, R.; Talbi, D. "Calculations on the Rates, Mechanisms, and Interstellar Importance of the Reactions between C and NH_2 and between N and CH_2." *Monthly Notices Of The Royal Astronomical Society*. V(311). N(4). pp 869-876. **2000**.

Herd, C. R.; Adams, N. G.; Smith, D. "OH Production in the Dissociative Recombination of H_3O^+, HCO_2^+ and N_2OH^+: Comparison with Theory and Interstellar Implications." *Astrophysical Journal*. V(349). pp 388-392. **1990**.

Hernandez, G. "The Signature Profiles of $O(^1S)$ in the Airglow." *Planetary and Space Science*. V(19). pp 467-476. **1971**.

Herzberg, G. "Diatomic Radicals and Ions" in *Spectra and Structures of Simple Free Radicals: An Introduction to Molecular Spectroscopy*. Dover Publications. New York. **1971**.

Herzberg, G. "Rydberg Molecules." *Annual Reviews in Physical Chemistry*. V(38). pp 27-56. **1987**.

Herzberg, G. *Molecular Spectra and Molecular Structure III. Electronic Spectra and Electronic Structure of Polyatomic Molecules*. Krieger Publishing Co.. Malabar, FL. **1988**.

Herzberg, G.; Huber, K. P. *Molecular Spectra and Molecular Structure: Constants of Diatomic Molecules*. Van Nostrand Reinhold. New York. **1997**.

Herzberg, G. *Molecular Spectra and Molecular Structure I. Spectra of Diatomic Molecules*. Krieger Publishing Co.. Malabar, FL. **1989**.

Higuera, F. J.; Liñán "Choking Conditions for Nonuniform Viscous Flows." *Physics of Fluids A*. V(5). N(3). pp 768-770. **1993**.

Hiller, B.; McDaniel, J. C.; Rea Jr, E. C.; Hanson, R. K. "Laser–Induced Fluorescence Technique for Velocity Field Measurements in Subsonic Gas Flows." *Optics Letters*. V(8). N(9). pp 474-476. **1983**.

Hoffman, J. M.; Hays, A. K.; Tisone, G. C. "High–Power UV Noble–Gas–Halide Lasers." *Applied Physics Letters*. V(28). N(9). pp 538-539. **1976**.

Hollo, S. D.; Hartfield, R. J.; McDaniel, J. C. "Planar Velocity Measurement in Symmetric Flow Fields with Laser–Induced Iodine Fluorescence." *Optics Letters*. V(19). N(3). pp 216-218. **1994**.

Howard, M. J.; Smith, I. W. M.; Eds.: Jennings, K. R.; Cundall, R. B.; Margerum, D. W. "The Kinetics of Radical–Radical Processes in the Gas Phase" in *Progress in Reaction Kinetics, 12^{th} Ed*. Pergamon Press. Oxford. **1983 a**.

Howard, M. J.; Smith, I. W. M. "Vibrational Relaxation" in *Progress in Reaction Kinetics, 12^{th} Ed.*. Pergamon Press. Oxford. **1983 b**.

Huber, K. P; Herzberg, G.; Data arranged by: Gallagher, J. W.; Johnson, R. D. III "Constants of Diatomic Molecules" in *NIST Chemistry WebBook, NIST Standard Reference Database Number 69, Eds. P.J. Linstrom and W.G. Mallard. (http://webbook.nist.gov)*. National Institute of Standards and Technology. Gaithersburg MD. **2001**.

Hunton, D. E.; Swider, W. "Variations of Water Vapor Concentration in the Shuttle Environment." *Journal of Spacecraft*. V(25). N(2). pp 139-145. **1988**.

Ikezoe, Y.; Matsuoka, S.; Takebe, M.; Viggiano, A. *Gas Phase Ion–Molecule Rate Constants Through 1986*. Ion Reaction Research Group Society of Japan. Tokyo. **1987**.

Jacox, M. E. "Vibrational and Electronic Energy Levels of Polyatomic Transient Molecules." Monograph Number 3. *Journal of Physical and Chemical Reference*

Data. American Chemical Society and American Institute of Physics for the National Institute for Standard and Technology. Gaithersburg, MD. **1994**.

Jacox, M. E. "Vibrational and Electronic Energy levels of Polyatomic Transient Molecules. Supplement A.." *Journal of Physical and Chemical Reference Data*. V(27). N(2). pp 115-393. **1998**.

Jacox, M. E. "Vibrational and Electronic Energy Levels of Polyatomic Transient Molecules" in *NIST Chemistry WebBook, NIST Standard Reference Database Number 69, Eds. P.J. Linstrom and W.G. Mallard. (http://webbook.nist.gov)*. National Institute of Standards and Technology. Gaithersburg MD. **2001**.

Janev, R. K. "Role of Dissociative Recombination and Related Molecular Processes in Fusion Edge Plasmas" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Jenny, E. "Unidimestional Transient Flow with Consideration of Friction, Heat Transfer and Change of Section." *The Brown Boveri Review*. V(Nov.). pp 447-461. **1950**.

Jensen, M.; Bilodeau, R.; Heber, O.; Pedersen, H.; Safvan, C.; Urbain, X.; Zajfman, D.; Andersen, L. "Dissociative Recombination and Excitation of H_2O^+ and HDO^+." *Physical Review A*. V(60). N(4). pp 2970-2976. **1999**.

Johnsen, R. "Microwave Afterglow Measurements of the Dissociative Recombination of Molecular Ions with Electrons." *International Journal of Mass Spectrometry and Ion Processes*. V(81). pp 67-84. **1987**.

Johnsen, R.; Gougousi, T.; Golde, M. F. "Flowing–Afterglow Measurements of the Recombination of H_3^+ and D_3^+ Ions" in *Proceedings of the 1995 Conference on Dissociative Recombination: Theory, Experiment and Applications III*. World Scientific. Singapore. **2000 a**.

Johnsen, R.; Skrzypkowski, M.; Gougousi, T.; Golde, M. F.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Spectroscopic Emissions from the Recombination of N_2O^+, N_2OH^+/HN_2O^+, CO_2^+, HCO^+/COH^+, H_2O^+, NO_2^+, HNO^+ and LIF Measurements of the H Atom Yield from H_3^+" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Application, IV*. World Scientific. Singapore. **2000 b**.

Johnson, A. W.; Gerardo, J. B. "Dissociative Recombination of Electrons with He_2^+." *Physical Review Letters*. V(28). N(17). pp 1096-1098. **1972**.

Johnson, A. W.; Gerardo, J. B. "Ionizing Collisions of Two Metastable Helium Atoms (2^3S)." *Physical Review A*. V(7). N(5). pp 925-928. **1973**.

Jones, P. L.; Mead, R. D.; Kohler, B. E.; Rosner, S. D.; Lineberger, W. C. "Photodetachment Spectroscopy of C_2^- Autodetaching Resonances." *Journal of Chemical Physics*. V(73). N(9). pp 4419-4432. **1980**.

Judge, D. L.; Lee, L. C. "Cross Sections for the Production of CO($a'\,^3\Sigma^+$, $d\,^3\Delta_i$, and $e\,^3\Sigma^- \rightarrow a\,^3\Pi$) Fluorescence Through Photodissociation of CO_2." *Journal of Chemical Physics*. V(58). N(1). pp 104-107. **1973**.

Jursic, B. S. "Complete Basis Set and Gaussian Computational Study of Dissociative Recombination Process of the Cyanogen Ion (CN^+)." *Abstracts Of Papers Of The American Chemical Society*. V(218). N(2). pp 227-. **1999**.

Jursic, B. S. "Complete Basis Set and Gaussian Computational Study of Dissociative Recombination Process of the Cyanogen Ion (CN^+)." *Journal Of Molecular Structure–Theochem*. V(498). pp 123-131. **2000**.

Katayama, D. H. "Collision Induced Electronic Energy Transfer Between the $A^2\Pi_{ui}$ ($v = 4$) and $X^2\Sigma_g^+$ ($v = 8$) Rotational Manifolds of N_2^+." *Journal of Chemical Physics*. V(81). N(8). pp 3495-3499. **1984**.

Katayama, D. H. "Direct Observation of Electronic Intramolecular Energy Transfer Through a Large Energy Gap." *Physical Review Letters*. V(54). N(7). pp 657-660. **1985**.

Ketvirtis, A. E.; Simons, J. "Dissociative recombination of $H3O^+$." *Journal Of Physical Chemistry A*. V(103). N(33). pp 6552-6563. **1999**.

Kley, D.; Lawrence, G. M.; Stone, E. J. "The Yield of $N(^2D)$ Atoms in the Dissociative Recombination of NO^+." *Journal of Chemical Physics*. V(66). N(9). pp 4157-4165. **1977**.

Klots, C. E. "Rate Constants for Unimolecular Decomposition at Threshold." *Chemical Physics Letters*. V(38). N(1). pp 61-64. **1976**.

Knuth, D. E *The TEXbook*. Addison–Wesley Publishing Co.. Reading, MA. **1986**.

Koch, D. G.; Melnick, G. J.; Fazio, G. G.; Rieke, G. H.; Low, F. J.; Hoffmann, W.; Young, E. T.; Urban, E. W.; Simpson, J. P.; Witteborn, F. C.; Gautier III, T. N.; Poteet, W. "Overview of Measurements from the Infrared Telescope on Spacelab 2." *Astro(nomical?). Lett(ers?). and Communications*. V(27). pp 211-222. **1988**.

Kofsky, I. L.; Barrett, J. L. "Spacecraft Glows from Surface–Catalyzed Reactions." *Planetary and Space Science*. V(34). N(8). pp 665-681. **1986**.

Kohring, G. A. "Calculation of the Permeability of Porous Media Using Hydrodynamic Cellular Automata." *Journal of Statistical Physics*. V(63). N(1 & 2). pp 411-. **1991 a**.

Kohring, G. A. "Limitations of a Finite Mean Free Path for Simulating Flows in Porous Media." *Journal of Physics II*. V(1). pp 593-597. **1991 b**.

Kohring, G. A. "Effect of Finite Grain Size on the Simulation of Fluid Flow in Porous Media." *Journal of Physics II*. V(1). pp 87-90. **1991 c**.

Kohring, G. A.; Eds: Landau, D. P.; Mon, K. K.; Schüttler, H.-B. "Large–Scale Lattice–Gas Simulations: A Case Study in Parallel Programming" in *Computer Simulation Studies in Condensed Matter Physics VI*. Springer–Verlag. Berlin. **1993**.

Kolts, J. H.; Setser, D. W. "Rate Constants for ArF* Formation from Reactions of $Ar(^3P_{2,0})$ with Fluorine Containing Molecules and the Pressure Dependence of the C to B State Ratios for ArF*, KrF* and XeF*." *Journal of Physical Chemistry*. V(82). N(15). pp 1766-1768. **1978**.

Komatsu, H.; Kano, S. S.; Takuma, H.; Shimizu, T. "Crossed–Beam Study of Initial Vibrational Distribution of KrF Excimer Produced in the Reaction of Kr* + F_2." *Japanese Journal of Applied Physics*. V(31 Pt. 2). N(3A). pp L280-L282. **1992**.

Kong, X. P.; Cohen, E. G. D. "A Kinetic Theorist's Look at Lattice Gas Cellular Automata." *Physica D*. V(47). pp 9-18. **1991**.

Kovács, I. *Rotational Structure in the Spectra of Diatomic Molecules*. American Elsevier Publishing Company, Inc.. New York. **1969**.

Kraemer, W. P.; Malmqvist, P. A. "Dissociative Recombination of HeH^+. I. Rovibrational Spectrum of HeH Rydberg states." *Theoretical Chemistry Accounts*. V(100). N(1-4). pp 65-77. **1998**.

Krassovsky, V. I.; Shefov, N. N.; Yarin, V. I. "Atlas of the Airglow Spectrum 3000-12400 Å." *Planetary and Space Science*. V(9). pp 883-915. **1962**.

Kriegl, M.; Richter, R.; Tosi, P.; Federer, W.; Lindinger, W. "The Vibrational Quenching of $O_2^+(\nu)$ by Kr as a Function of Relative Kinetic Energy." *Chemical Physics Letters*. V(124). N(6). pp 583-585. **1986**.

Krohn, S. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Vibrational population Diagnostics for Dissociative Recombination Experiments at the TSR Using Coulomb Explosion Imaging" in *Proceedings of the 1999 Conference on*

Dissociative Recombination: Theory, Experiment and Applications, IV. World Scientific. Singapore. **2000**.

Lambert, J. D. *Vibrational and Rotational Relaxation in Gases*. Clarendon Press. Oxford. **1977**.

Landau, D. P.; Binder, K. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press. New York. **2000**.

Lang, K. R. *Astrophysical Formulae: A Compendium for the Physicist and Astrophysicist*. Springer–Verlag. Berlin. **1978**.

Lange, M.; Levin, J.; Gwinner, G.; Hechtfischer, U.; Knoll, L.; Schwalm, D.; Wester, R.; Wolf, A.; Urbain, X.; Zajfman, D. "Threshold Effects and Ion–Pair Production in the Dissociative Recombination of HD^+." *Physical Review Letters*. V(83). N(24). pp 4979-4982. **1999**.

Lange, M. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "New Measurements of the Dissociative Recombination Cross Section of HD^+ and the Importance of Electron Beam Geometry" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Larson, Å.; Le Padellec, A.; Semaniak, J.; Strömholm, C.; Larsson, M.; Rosén, S.; Peverall, R.; Danared, H.; Djuric, N.; Dunn, G.; Datz, S. "Branching Fractions in Dissociative Recombination of CH_2^+." *Astrophysical Journal*. V(505). pp 459-465. **1998**.

Larson, Å.; Orel, A. "Dissociative Recombination of HeH^+: Product Distributions and Ion–Pair Formation." *Physical Review A*. V(59). N(5). pp 3601-3608. **1999**.

Larson, Å.; et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Theoretical Study of Ion–Pair Formation in Collisions of Electrons with HD^+" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Larsson, M.; Lepp, S.; Dalgarno, A.; Stromholm, C.; Sundstrom, G.; Zengin, V.; Danared, H.; Källberg, A.; af Ugglas, M.; Datz, S. "Dissociative Recombination of H_2D^+ and the Cosmic Abundance of Deuterium." *Astronomy and Astrophysics*. V(309). pp L1-L3. **1996**.

Larsson, M.; Mitchell, J. B. A.; Schneider, I. F., Eds. *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Laubé, S.; Le Padellec, A.; Sidko, O.; Rebrion–Rowe, C.; Mitchell, J.; Rowe, B. "New FALP–MS Measurements of H_3^+, D_3^+ and HCO^+ Dissociative Recombination." *Journal of Physics B: Atomic, Molecular and Optical Physics*. V(31). pp 2111-2128. **1998 a**.

Laube, S.; Lehfaoui, L.; Rowe, B. R.; Mitchell, J. B. A. "The Dissociative Recombination of CO_2^+." *Journal Of Physics B–Atomic Molecular And Optical Physics*. V(31). N(18). pp 4181-4189. **1998 b**.

Laudenslager, J. B.; Huntress, W. T.; Bowers, M. T. "Near Thermal Energy Charge Transfer Reactions of Rare Gas Ions with Diatomic and Simple Polyatomic Molecules: The Importance of Franck–Condon Factors and Energy Resonance on the Magnitude of Rate Constants." *Journal of Chemical Physics*. V(61). N(11). pp 4600-4617. **1974**.

LaVallée, P.; Boon, J. P.; Noullez, A. "Boundaries in Lattice Gas Flows." *Physica D*. V(47). pp 233-240. **1991**.

Lawniczak, A.; Dab, D.; Kapral, R.; Boon, J. P. "Reactive Lattice Gas Automata." *Physica D*. V(47). pp 132-158. **1991**.

Lawrence, G. M. "Quenching and Radiation Rates of CO ($a\,^3\Pi$)." *Chemical Physics Letters*. V(9). N(6). pp 575-577. **1971**.

Lawrence, G. M. "Photodissociation of CO_2 to Produce $CO(a\,^3\Pi)$." *Journal of Chemical Physics*. V(56). N(7). pp 3435-3442. **1972**.

Leber, E. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Resonance and Threshold Phenomena in Electron Attachment to Molecules and Clusters" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Lee, F. W.; Collins, C. B. "Measurement of the Rate Coefficients for the Bimolecular and Termolecular De–Excitation Reactions of $He(2\,^3S)$ with Ne, Ar, N_2, CO, CO_2, and CH_4*." *Journal of Chemical Physics*. V(65). N(12). pp 5189-5197. **1977**.

Lefebvre–Brion, H.; Field, R. W. *Perturbations in the Spectra of Diatomic Molecules*. Academic Press, Inc.. Orlando, FL. **1986**.

Lengyel, V. I.; Navrotsky, V. T.; Sabad, E. P. *Resonance Phenomena in Electron–Atom Collisions*. Springer–Verlag. City?. **1992**.

LePadellec, A.; Laube', S.; Sidko, O.; Rebrion–Rowe, C.; Rowe, B. R.; Sarpal, B.; Mitchell, J. B. A. "The Dissociative Recombination of KrH^+ and XeH^+." *Journal of Physics B: Atomic, Molecular and Optical Physics*. V(30). pp 963-967. **1997**.

LePadellec, A.; Mitchell, J. B. A.; AlKhalili, A.; Danared, H.; Kallberg, A.; Larson, A.; Rosen, S.; afUgglas, M.; Vikor, L.; Larsson, M. "Storage Ring Measurements of the Dissociative Recombination and Excitation of the Cyanogen Ion CN^+ ($X\,^1\Sigma^+$) and a $^3\Pi$, $\nu = 0$)." *Journal Of Chemical Physics*. V(110). N(2). pp 890-901. **1999**.

LeRoy, R. J.; Keogh, W. L.; Child, M. S. "An Inversion Procedure for Oscillatory Continuum Spectra: Method and Application to NaK." *Journal of Chemical Physics*. V(89). N(8). pp 4564-4578. **1988**.

LeRoy, R. J.*BCONT 1.4: A Computer Program for Calculating Absorption Coefficients, Emission Intensities or (Golden Rule) Predissociation Rates*. University of Waterloo Chemical Physics Research Report. **1993**.

Levine, I. N. *Quantum Chemistry, 4^{th} Ed.*. Allyn and Bacon, Inc.. Boston. **1983**.

Lichten, W.; McCusker, M. V.; Vierima, T. L. "Fine Structure of the Metastable $a^3\Sigma_u^+$ State of the Helium Molecule." *Journal of Chemical Physics*. V(61). N(6). pp 2200-2212. **1974**.

Lindinger, W.; Schmeltekopf, A. L.; Fehsenfeld, F. C. "Temperature Dependence of De-Excitation Rate Constants of $He(2^3S)$ by Ne, Ar, Xe, H_2, N_2, O_2, NH_3 and CO_2." *Journal of Chemical Physics*. V(61). N(7). pp 2890-2895. **1974**.

Lo, G.; Setser, D. W. "Improved KrF(B) and KrF(X) State Potentials." *Journal of Chemical Physics*. V(100). N(8). pp 5432-5440. **1994**.

Loginov, A. V.; Eds.: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Vibrational Population of Rare Gases Excimers Excited by Electric Discharge" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Ludlum, K. H.; Larson, L. P.; Caffrey, J. M, Jr. "Activation Energy for the Three–Body Reaction of Helium Triplet Atom with Normal Helium." *Journal of Chemical Physics*. V(46). N(1). pp 127-130. **1967**.

Luo, M.; Jungen, M. "The H3O Rydberg Radical." *Chemical Physics*. V(241). N(3). pp 297-303. **1999**.

Macagno, E. C.; Hung, T.-K. "Computational and Experimental Study of a Captive Annular Eddy." *Journal of Fluid Mechanics*. V(28). N(1). pp 43-64. **1967**.

Machuzak, J. S.; Burke, W. J.; Retterer, J. M.; H(?i)nton, D. E.; Jasperse, J. R.; Smiddy, M. "Effects of Thruster Firings on the Shuttle's Plasma and Electric Field

Environment." *Journal of Geophysical Research*. V(98). N(A2). pp 1513-1530. **1993**.

Marci, P. A.; Barrachina, R. O. "Theory of Electron Capture in the Continuum of Neutral Helium Projectiles." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(31). pp 1303-1312. **1998**.

Marinelli, W. J.; Kessler, W. J.; Green, B. D.; Blumberg, W. A. M. "Quenching of $N_2(a\ ^1\Pi_g,\ v'=0)$ by N_2, O_2, CO, CO_2, CH_4, H_2 and Ar." *Journal of Chemical Physics*. V(90). N(4). pp 2167-2173. **1989**.

McCarthy, I. E.; Weigold, E. "Ionisation" in *Electron–Atom Collisions, 1st Ed.*. Cambridge University Press. Cambridge. **1995**.

McCay, T. D.; Powell, H. M.; Busby, M. R. "Direct Mass Spectrometric Measurements in a Highly Expanded Rocket Exhaust Plume." *Journal of Spacecraft*. V(15). N(3). pp 133-138. **1978**.

McDaniel, J. C.; Hiller, B.; Hanson, R. K. "Simulataneous Multiple–Point Velocity Measurements Using Laser–Induced Iodine Fluorescence." *Optics Letters*. V(8). N(1). pp 51-53. **1983**.

McEwan, M. J.; Eds.: Adams, N. G.; Babcock, L. M. "Flow Tube Studies of Small Isomeric Ions" in *Advances in Gas Phase Ion Chemistry*. JAI Press, Inc.. Greenwich, CT. **1992**.

McQuarrie, D. A. *Statistical Mechanics*. Harper and Rowe. New York. **1976**.

McQuarrie, D. A. *Quantum Chemistry*. University Science Books. Mill Valley, CA. **1983**.

Mende, S. B.; Garriott, O. K.; Banks, P. M. "Observations of Optical Emissions on STS-4." *Geophysical Research Letters*. V(10). N(2). pp 122-125. **1983**.

Mies, F. H.; Smith, A. L. "Bandlike Structure from Continuum–Continuum Emission: The He_2 600-Å Bands." *Journal of Chemical Physics*. V(45). N(3). pp 994-1000. **1966**.

Mies, F. H. "Calculated Vibrational Transition Probabilities for $OH(X\ ^2\Pi)$." *Journal of Molecular Spectroscopy*. V(53). pp 150-188. **1974**.

Miller, D. W.; Adelman, S. A. "Vibrational Energy Transfer in Fluids." *International Reviews in Physical Chemistry*. V(13). N(2). pp 359-386. **1994**.

Miller, T. A.; Freund, R. S.; Zegarski, B. R.; Jost, R.; Lombardi, M.; Derourard, J. "Observation of Singlet–Triplet Anticrossings in $^4He_2^*$." *Journal of Chemical Physics*. V(63). N(9). pp 4042-4046. **1975**.

Miller, W. H. "Uniform Semiclassical Approximations for Elastic Scattering and Eigenvalue Problems." *Journal of Chemical Physics*. V(48). N(1). pp 464-467. **1968**.

Miller, W. H. "Tunneling Corrections to Unimolecular Rate Constants, with Application to Formaldehyde." *Journal of the American Chemical Society*. V(101). N(23). pp 6810-6814. **1979**.

Mistrik, I.; Reichle, R.; Muller, U.; Helm, H.; Jungen, M.; Stephens, J. A. "Ab Initio Analysis of Autoionization of H_3 Molecules Using Multichannel Quantum–Defect Theory and New Quantum–Defect Surfaces." *Physical Review A*. V(6103). N(3). pp 3410-. **2000**.

Mitchell, J. B. A.; Rowe, B. R.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Electron–Molecule Collisions: New Experiments, New Ideas" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Mitsuke, K.; Takami, T.; Ohno, K. "Kinetic Energy Dependence of Partial Cross Sections for the Collisional Ionzation of H_2O, H_2S, O_2 and Ar with $He(2\ ^3S)$." *Journal of Chemical Physics*. V(91). N(3). pp 1618-1625. **1989**.

Miyazaki, T.; Harada, A.; Ikuta, N. "Cross Section Dependent Transport Properties of Thermal Gases." *Journal of the Physical Society of Japan*. V(64). N(2). pp 456-462. **1995**.

Moazzen–Ahmadi, N.; McKellar, A. R. W.; Amano, T. "Diode Laser Spectroscopy of Gas Phase C_5: The ν_3 Fundamental and Associated Hot Bands." *Journal of Chemical Physics*. V(91). N(4). pp 2140-2147. **1989**.

Monnery, W. D.; Svrcek, W. Y.; Mehrotra, A. K. "Viscosity: A Critical Review of Practical Predictive and Correlative Methods." *The Canadian Journal of Chemical Engineering*. V(73). pp 3-40. **1995**.

Moore, C. B. "Vibration→Vibration Energy Transfer." *Advances in Chemical Physics*. V(23). pp 41-83. **1973**.

Morgan, J. E.; Phillips, L. F.; Schiff, H. I. "Studies of Vibrationally Excited Nitrogen Using Mass Spectrometric and Calorimeter–Probe Techniques." *Discussions of the Faraday Society*. V(33). pp 118-121. **1962**.

Morrison, R. J. S.; Conaway, W. E.; Zare, R. N. "Effect of Internal and Translational Energy on the $NH_3^+(\nu) + D_2$ Ion–Molecule Reaction." *Chemical Physics Letters*. V(113). N(5). pp 435-440. **1985**.

Morse, P. M. "Diatomic Molecules According to the Wave Mechanics. II. Vibrational Levels.." *Physical Review*. V(34). pp 57-64. **1929**.

Mostefaoui, T. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "The Influence of Vibrational Excitation on the Dissociative Recombination of NO^+" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Müller, U. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Three–Body Breakup of Laser–Excited Triatomic Hydrogen Molecules" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000 a**.

Müller, U. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Photoionization of the Triatomic Hydrogen Molecule" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000 b**.

Mulliken, R. S. "Rare–gas and Hydrogen Molecule Electronic States, Noncrossing Rule, and Recombination of Electrons with Rare–gas and Hydrogen Ions." *Physical Review*. V(136). N(4A). pp A962-A965. **1964**.

Murad, E.; Lai, S. "Effect of Dissociative Electron–Ion Recombination on the Propagation of Critical Ionization Discharges." *Journal of Geophysical Research*. V(91). N(A12). pp 13745-13749. **1986**.

Murad, E.; Knecht, D. J.; Viereck, R. A.; Pike, C. P.; Kofsky, I. L.; Trowbridge, C. A.; Rall, D. L. A.; Ashley, G.; Twist, L.; Elgin, J. B.; Setayesh, A., et al. "Visible Light Emission Excited by Interaction of Space Shuttle Exhaust with the Atmosphere." *Geophysical Research Letters*. V(17). N(12). pp 2205-2208. **1990**.

NATO Advanced Research Workshop "Lattice Gas Methods for PDE's." *Physica D*. V(47). N(1 & 2). pp 299-339. **1991**.

Neau, A.; AlKhalili, A.; Rosen, S.; LePadellec, A.; Derkatch, A. M.; Shi, W.; Vikor, L.; Larsson, M.; Semaniak, J.; Thomas, R.; Nagard, M. B.; Andersson, K.; Danared, H.; afUgglas, M. "Dissociative Recombination of D_3O^+ and H_3O^+: Absolute Cross Sections and Branching Ratios." *Journal Of Chemical Physics*. V(113). N(5). pp 1762-1770. **2000**.

NIST, Linstrom, P. J and Mallard, W. G., Eds. *NIST Chemistry WebBook, NIST Standard Reference Database Number 69 (http://webbook.nist.gov)*. National Institute of Standards and Technology. Gaithersburg MD. **2001 a**.

NIST *Atomic Spectra Database (http://physics.nist.gov/cgi-bin/AtData/main_asd)*. National Institute of Standards and Technology. Gaithersburg MD. **2001 b**.

Noullez, A.; Boon, J.-P "Long–Time Correlations in a 2D Lattice Gas." *Physica D*. V(47). pp 212-215. **1991**.

Oka, T. "H_3^+ in the Diffuse Interstellar Medium: The Enigma Related to Dissociative Recombination" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Oran, E. S.; Boris, J. P. *Numerical Simulation of Reactive Flow*. Elsevier Science Publishing Co., Inc.. New York. **1987**.

Oran, E. S.; Boris, J. P.; DeVore, C. R. "Reactive–Flow Computations on a Massively Parallel Computer." *Fluid Dynamics Research*. V(10). pp 251-271. **1992**.

Orel, A. E. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Wave Packet Studies of Dissociative Recombination and Dissociative Excitation of Molecular Ions" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Orient, O. J.; Martus, K. E.; Chutjan, A; Murad, E. "Recombination of 5-eV O(^3P) Atoms with Surface–Adsobed NO: Spectra and Their Dependence on Surface Material and Temperature." *Physical Review A*. V(45). N(5). pp 2998-3003. **1992**.

Orient, O. J.; Chutjian, A.; Martus, K. E.; Murad, E. "Observation of CN A→X and B→X Emissions in Gas–Phase Collisions of Fast O(^3P) Atoms with HCN." *Physical Review A*. V(48). N(1). pp 427-431. **1993**.

Orr, B. J.; Smith, I. W. M. "Collision–Induced Vibrational Energy Transfer in Small Polyatomic Molecules." *Journal of Physical Chemistry*. V(91). N(24). pp 6106-6119. **1987**.

Orth, F. B.; Ginter, M. L. "The Spectrum and Structure of the He2 Molecule: Characterization of the Triplet States Associated with the UAO's 4pσ, 5pσ and 6pσ.." *Journal of Molecular Spectroscopy*. V(61). pp 282-288. **1976**.

Osamura, Y.; Fukuzawa, K.; Terzieva, R.; Herbst, E.; "A Molecular Orbital Study of the $HC_3NH^+ + e^-$ Dissociative Recombination and Its Role in the Production of Cyanoacetylene Isomers in Interstellar Clouds." *Astrophysical Journal*. V(519). N(2 pt 1). pp 697-704. **1999**.

Pearse, R. W. B.; Gaydon, A. G. *The Identification of Molecular Spectra, 4^{th} Ed.*. John Wiley & Sons. New York. **1976**.

Petravic, J.; Isbister, D. "Pressure Tensor of the Hard–Disk Lorentz Gas." *Physical Review E*. V(51). N(5). pp 4309-4318. **1995**.

Pichl, L.; Nakamura, H.; Horacek, J. "Analytical Treatment of Singular Equations in Dissociative Recombination." *Computer Physics Communications*. V(124). N(1). pp 1-18. **2000**.

Pubanz, G. A.; Maroncelli, M.; Nibler, J. W. "CARS Spectra of van der Waals Complexes: The Structure of the CO_2 Dimer." *Chemical Physics Letters*. V(120). N(3). pp 313-317. **1985**.

Queffelec, J. L.; Rowe, B. R.; Morlais, M.; Gomet, J. C.; Valleé, F. "The Dissociative Recombination of N_2^+ ($v = 0, 1$) as a Source of Metastable Atoms in Planetary Atmospheres." *Planetary and Space Science*. V(33). N(3). pp 263-270. **1985**.

Rabadan, I.; Tennyson, J. "An Ab Initio Calculation of Electron Impact Vibrational Excitation of NO^+." *Journal Of Physics B–Atomic Molecular And Optical Physics*. V(32). N(19). pp 4753-4762. **1999**.

Radzig, A. A. *Smirnov, B. M.*. Reference Data on Atoms, Molecules and Ions. Springer–Verlag. **Berlin**. 1985

Rapaport, D. C.; Eds.: Landau. D. P.; Mon, K. K.; Schüttler, H.-B. "Molecular Dynamics: A New Approach to Hydrodynamics?" in *Computer Simulation Studies in Condensed Matter Physics: Springer Proceedings in Physics 33*. Springer Verlag. Berlin. **1988**.

Rapp, D. "Interchange of Vibrational Energy between Molecules in Collisions." *Journal of Chemical Physics*. V(43). pp 316-317. **1965**.

Ray, D.; Robinson, R. L.; Gwo, D.-H.; Saykally, R. J. "Vibrational Spectroscopy of van der Waals Bonds: Measurement of the Perpendicular Bend of ArHCl by Intracavity Far Infrared Laser Spectroscopy of a Supersonic Jet." *Journal of Chemical Physics*. V(84). N(3). pp 1171-1180. **1986**.

Read, D. N. "Rotational and Vibrational Structure of the Fourth Positive Bands of Carbon Monoxide." *Physical Review*. V(46). pp 571-575. **1934**.

Read, F. H. *Electromagnetic Radiation*. John Wiley & Sons. Bristol, England. **1980**.

Rebrion–Rowe, C.; et al. "The Recombination of Hydrocarbon Ions with Electrons" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000 a**.

Rebrion–Rowe, C.; Mostefaoui, T.; Laube, S.; Mitchell, J. B. A. "The Dissociative Recombination of Hydrocarbon Ions, III. Methyl–Substituted Benzene Ring Compounds." *Journal of Chemical Physics*. V(113). N(8). pp 3039-3045. **2000 b**.

Resende, S. M.; Ornellas, F. R.; "Radiative and Predissociative lifetimes of the A $^2\Sigma^+$ State (v'=0,1) of SH and SD: A Highly Correlated Theoretical Investigation.." *Journal of Chemical Physics*. V(115). N(5). pp 2178-2187. **2001**.

Riccardi, G.; Bauer, C.; Lim, H. "Boundary and Obstacle Processing in a Vectorized Model of Lattice Gas Hydrodynamics." *Physica D*. V(47). pp 281-295. **1991**.

Rind, D.; Chiou, E.-W.; Chu, W.; Oltmans, S.; Lerner, J.; Larsen, J.; McCormick, M. P.; McMaster, L. "Overview of the Stratospheric Aerosol and Gas Experiment II Water Vapor Observations: Method, Validation, and Data Characteristics." *Journal of Geophysical Research*. V(98). N(D3). pp 4835-4856. **1993**.

Robinson, D. Personal communication. **1992**.

Rosen, S.; Peverall, R.; Larsson, M.; LePadellec, A.; Semaniak, J.; Larson, A.; Strömholm, C.; van der Zande, W. J.; Danared, H.; Dunn, G. H "Absolute Cross Sections and Final–State Distributions for Dissociative Recombination and Excitation of CO^+ (v=O) Using an Ion Storage Ring." *Physical Review A*. V(57). N(6). pp 4462-4471. **1998**.

Rosen, S.; Derkatch, A.; Semaniak, J.; Neau, A.; AlKhalili, A.; LePadellec, A.; Vikor, L.; Thomas, R.; Danared, H.; afUgglas, M.; Larsson, M "Recombination of Simple Molecular Ions Studied in Storage Ring: Dissociative Recombination of H_2O^+." *Faraday Discussions*. V(115). pp 295-302. **2000**.

Sauter, M.; Keller, N.; Nakel, W. "Influence of Projectile–Nucleus Interaction on Relativistic Electron–Impact Ionization." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(31). pp L947-L950. **1998**.

Sawaryn, A.; Sokalski, W. A. "Cumulative Atomic Multipole Moments and Point Charge Models Describing Molecular Charge Distribution." *Computer Physics Communications*. V(52). pp 397-408. **1989**.

Schattschneider, P. *Fundamentals of Inelastic Electron Scattering*. Springer–Verlag. Wien. **1986**.

Schloss, J. H.; Jones, R. B.; Eden, J. G. "Wavelength Dependence of the Photoassociation of $Kr(4p^6\ ^1S_0)$ - $F(2p^5\ ^2P)$ Collision Pairs." *Chemical Physics Letters*. V(191). N(12). pp 195-202. **1992**.

Schmeltekopf Jr.; A. L.; Broida, H. P. "Short Duration Visible Afterglow in Helium." *Journal of Chemical Physics*. V(39). N(1). pp 1261-1268. **1963**.

Schmeltekopf, A. L.; Fehsenfeld, F. C.; Gilman, G. I.; Ferguson, E. E. "Reaction of Atomic Oxygen Ions with Vibrationally Excited Nitrogen Molecules." *Planetary and Space Science*. V(15). pp 401-406. **1967**.

Schmeltekopf, A. L.; Ferguson, E. E.; Feshsenfeld, F. C. "Afterglow Studies of the Reactions He^+, $He(2^3S)$, and O^+ with Vibrationally Excited N_2." *Journal of Chemical Physics*. V(48). N(7). pp 2966-2973. **1968**.

Schmeltekopf, A. L.; Fehsenfeld, F. C. "De-Excitation Rate Constants for Helium Metastable Atoms with Several Atoms and Molecules." *Journal of Chemical Physics*. V(53). N(8). pp 3173-3177. **1970**.

Schneider, I. F. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Dissociative Recombination of H_3^+ and Predissociation of H_3" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000 a**.

Schneider, I. F. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Effects of Channel Mixing on Very Slow Direct Dissociative Recombination Processes" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000 b**.

Segatz, J.; Rannacher, R.; Wichmann, J.; Orlemann, C.; Dreier, T.; Wolfrum, J. "Detailed Numerical Simulations in Flow Reactors: A New Approach in Measuring Absolute Rate Constants." *Journal of Physical Chemistry*. V(100). pp 9323-9333. **1996**.

Sekiya, H.; Tsuji, M.; Nishimura, Y. "Emission Spectra Produced from the Reactions of Ne^+ and Ar^+ Ions with OCS at Thermal Energy." *Chemical Physics Letters*. V(100). N(6). pp 494-498. **1983**.

Semaniak, J.; Larson, Å.; Le Padellec, A.; Strömholm, C.; Larsson, M.; Rosén, S.; Peverall, R.; Danared, H.; Djuric, N.; Dunn, G.; Datz, S. "Dissociative Recombination and Excitation of CH_5^+: Absolute Cross Sections and Branching Ratios." *Astrophysical Journal*. V(498). pp 886-895. **1998**.

Sheehan, C.; LePadellec, A.; Lennard, W. N.; Talbi, D.; Mitchell, J. B. A. "Merged Beam Measurement of the Dissociative Recombination of HCN^+ and HNC^+." *Journal Of Physics B–Atomic Molecular And Optical Physics*. V(32). N(14). pp 3347-3360. **1999**.

Shukla, P.; Sharma, R. R.; Singh, P. "Decay of a Saw–Tooth Profile in Chemically Reacting Gases." *International Journal of Engineering Science*. V(32). N(3). pp 527-533. **1994**.

Singh, H.; Singh, S.; Deb, B. M. "Lattice Gas Automata: A Tool for Exploring Dynamical Processes." *Proceedings of the Indian Academy of Sciences (Chemical Sciences)*. V(106). N(2). pp 539-551. **1994**.

Skrzypkowski, M.; Gougousi, T.; Johnsen, R.; Golde, M. "Measurement of the Absolute Yield of $CO(a\,^3\Pi)$ + O Products in the Dissociative Recombination of CO_2^+ Ions with Electrons." *Journal of Chemical Physics*. V(108). N(20). pp 8400-8407. **1998**.

Slanger, T. G.; Black, G. "The Perturbation Spectrum of CO." *Chemical Physics Letters*. V(4). N(9). pp 558-560. **1976**.

Smith, A. L. "Observing the Effect of a Change in Mass on the de Broglie Wavelength: The 600-Å Bands of $^3He_2$." *Journal of Chemical Physics*. V(47). pp 1561-1562. **1967**.

Smith, D.; Plumb, I. C. "An Appraisal of the Single Langmuir Probe Technique in the Study of Afterglow Plasmas." *Journal of Physics D: Applied Physics*. V(5). pp 1226-1238. **1972**.

Smith, D.; Adams, N. G.; Dean, A. G.; Church, M. J. "The Application of Langmuir Probes to the Study of Flowing Afterglow Plasmas." *Journal of Physics D: Applied Physics*. V(8). pp 141-152. **1975**.

Smith, I. W. M.; Ed.: Capitelli, M. "Vibrational Energy Transfer in Collisions Involving Free Radicals." *Topics in Current Physics: Nonequilibrium Vibrational Kinetics*. V(39). pp 113-157. **1986**.

Sobczynski, R.; Beaman, R.; Setser, D. W.; Sadeghi, N. "Generation of $Kr(^3P_0)$ Atoms in a Flow Reactor: Reactions with CO, N_2, NF_3 and F_2." *Chemical Physics Letters*. V(154). N(4). pp 349-356. **1989**.

Sonnenfroh, D. M.; Caledonia, G. E. "Collisional Desorption of NO by Fast O atoms." *Journal of Geophysical Research*. V(98). N(A12). pp 21605-21610. **1993 a**.

Sonnenfroh, D. M.; Caledonia, G. E.; Lurie, J. "Emission from OH(A) Produced in the Dissociative Recombination of H_2O^+ with Electrons." *Journal of Chemical Physics*. V(98). N(4). pp 2872-2881. **1993 b**.

Spartan '02. See: Wavefunction, Inc.

Stark, G.; Yoshino, K.; Smith, P. L. "A High–Resolution Study of the Vacuum Ultraviolet Spectrum of CS: The $B^1\Sigma^+ - X^1\Sigma^+$ System." *Journal of Molecular Spectroscopy*. V(124). pp 420-429. **1987**.

Stedman, D. H.; Steffenson, D.; Niki, H. "The Reaction Between Active Hydrogen and Cl_2. Evidence for the Participation of Vibrationally Excited H_2." *Chemical Physics Letters*. V(7). N(2). pp 173-174. **1970**.

Steinfeld, J. I. *Molecules and Radiation: An Introduction to Modern Molecular Spectroscopy, 2^{nd} Ed.*. The MIT Press. Cambridge, MA. **1985**.

Strömholm, C.; Semaniak, J.; Rosén, S; Danared, H.; Datz, S.; van der Zande, W.; Larsson, M. "Dissociative Recombination and Dissociative Excitation of $^4HeH^+$: Absolute Cross Sections and Mechanisms." *Physical Review A*. V(54). N(4). pp 3086-3094. **1996**.

Succi, S.; Benzi, R.; Higuera, F. "The Lattice Boltzmann Equation: A New Tool for Computational Fluid–Dynamics." *Physica D*. V(47). pp 219-130. **1991**.

Suzuki, K.; Kuchitsu, K. "Emission Spectra of $CH(A\ ^2\Delta)$ from Methane in an Argon Flowing Afterglow." *Chemical Physics Letters*. V(56). N(1). pp 50-53. **1978**.

Swenson, G. R.; Mende, S. B.; Clifton, K. S. "Ram Vehicle Glow Spectrum: Implication of NO_2 Recombination Continuum." *Geophysical Research Letters*. V(12). N(2). pp 97-100. **1985**.

Swenson, G. R.; Leone, A.; Holtzclaw, K. W.; Caledonia, G. E. "Spatial and Spectral Characterization of Laboratory Shuttle Glow Simulations." *Journal of Geophysical Research*. V(96). N(A5). pp 7603-7612. **1991**.

Tachikawa, H. "Reaction Mechanism of the Astrochemical Electron Capture Reaction $HCNH^+ + e^- \rightarrow HNC + H$: a Direct Ab–initio Study." *Physical Chemistry Chemical Physics*. V(1). pp 4925-4930. **1999**.

Tagaki, H.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Higher Order Contribution in the Dissociative Processes of Diatomic Molecules" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Taieb, G.; Broida, H. P. "Neutral and Ionic Emission from Collisionally Excited Additives to an Helium Afterglow." *Chemical Physics*. V(21). pp 313-316. **1977**.

Talbi, D. "Quantum Chemical Calculations to Help Understanding of Electronic Dissociative Recombination Processes" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications IV*. World Scientific. Singapore. **2000**.

Tanabe, T. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Dissociative Recombination at the TARN II Storage Ring" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Tennyson, J. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Resonance Curves for Dissociative Recombination Using the R–Matrix Method" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Tichy, M.; Javahery, G.; Twiddy, N. D.; Ferguson, E. E. "Vibrational Quenching of $HCl^+(\nu = 1)$ and $DCl^+(\nu = 1)$ by Ar and Kr." *Chemical Physics Letters*. V(144). N(2). pp 131-135. **1988**.

Tomashevsky, M.; Herbst, E.; Kraemer, W. "Classical and Quantum–Mechanical Calculations of $HCO^+ + e \rightarrow CO(\nu) + H$." *Astrophysical Journal*. V(498). pp 728-734. **1998**.

Torr, M. R.; Hays, P. B.; Kennedy, B. C.; Walker, J. G. C. "Intercalibration of Airglow Observatories with the Atmosphere Explorer Satellite." *Planetary and Space Science*. V(25). pp 173-184. **1977**.

Torr, M. R.; Torr, D. G. "The Dissociative Recombination of O_2^+ in the Ionosphere." *Planetary and Space Science*. V(29). N(9). pp 999-1010. **1981**.

Tsuji, M.; Obase, H.; Matsuo, M.; Endoh, M.; Nishimura, Y. "$CS^+(B\,^2\Sigma^+ - A\,^2\Pi_i)$ Emission Produced from Dissociative Charge–Transfer Reactions of He^+ with CS_2 and OCS at Thermal Energy." *Chemical Physics*. V(50). pp 195-207. **1980**.

Tsuji, M.; Furusawa, M.; Nishimura, Y. "ArF*, KrF* and XeF* Emissions Produced from Dissociative Ion Recombination Reactions of Ar^+, Kr^+ and Xe^+ with SF_6^- in the Flowing Afterglow." *Chemical Physics Letters*. V(166). N(4). pp 363-368. **1990**.

Tsuji, M.; Kobarai, K.; Kouno, H.; Obase, H.; Nishimura, Y. "Formation of $CH(A\,^2\Delta$, $B\,^2\Sigma^-$, $C\,^2\Sigma^+)$ by Electron–Ion Recombination Processes in the Argon and Krypton Afterglow Reactions of CH_4." *Journal of Chemical Physics*. V(94). N(2). pp 1127-1133. **1991**.

Tsuji, M.; Furusawa, M.; Mizuguchi, T.; Muraoka, T.; Nishimura, Y. "Dissociative Excitation of CF_4, CCl_4, and Chlorofluoromethanes by Collisions with Argon and Helium Active Species." *Journal of Chemical Physics*. V(97). N(1). pp 245-255. **1992**.

Tsuji, M.; Kouno, H.; Matsumura, K.; Funatsu, T.; Nishimura, Y.; Obase, H.; Kugishima, H.; Yoshida, K. "Dissociative Charge–Transfer Reactions of Ar^+ with Simple Aliphatic Hydrocarbons at Thermal Energy." *Journal of Chemical Physics*. V(98). N(3). pp 2011-2022. **1993**.

Tsuji, M.; Nakamura, M.; Nishimura, Y. "Nascent Rovibrational Distribution of $CO(A\ ^1\Pi)$ Produced in the Recombination of CO_2^+ with Electrons." *Journal of Chemical Physics*. V(103). N(4). pp 1413-1421. **1995**.

Tsuji, M.; Nakamura, M.; Nishimura, Y. "Nascent Rovibrational Distributions of CO ($d^3\Delta_i$, $e^3\Sigma^-$, $a'^3\Sigma^+$) Produced in the Dissociative Recombination of CO_2^+ with Electrons." *Journal of Chemical Physics*. V(108). N(19). pp 8031-8038. **1998**.

Tsuji, M.; Tanoue, T.; Tanaka, Y.; Nishimura, Y. "Vibrational Distribution of N_2 ($B^3\Pi_g$) Produced from Dissociative Recombination of N_2O^+ in a Helium Flowing Afterglow." *Chemistry Letters*. V(6). pp 592-593. **2000**.

Uehigashi, A.; Sugiura, S.; Morinishi, K.; Satofuka, N. "Numerical Investigation Using Compressible Navier–Stokes Equations for Low–Speed Flow in Pipes with Varying Cross Sections." *JMSE International Journal*. V(35). N(4). pp 507-512. **1992**.

Urbain, X.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Long–Range Effects in the Formation of He_2^+ and the Dissociative Recombination of HD^+" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Valée, F.; Rowe, B. R.; Gomet, J. C.; Quéffelec, J. L.; Morlais, M. "Observation of the Fourth Positive System of CO in Dissociative Recombination of Vibrationally Excited CO_2^+." *Chemical Physics Letters*. V(124). N(4). pp 317-320. **1986**.

van den Berg, H. R.; ten Seldam, C. A.; van der Gulik, P. S. "Compressible Laminar Flow in a Capillary." *Journal of Fluid Mechanics*. V(246). pp 1-20. **1993**.

van der Hoef, M. A.; Frenkel, D. "Tagged Particle Diffusion in 3D Lattice Gas Cellular Automata." *Physica D*. V(47). pp 191-197. **1991**.

van der Zande, W. J.; Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. "Dissociative Recombination of Diatomics: Do We Understand Product State Branching?" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Velazco, J. E.; Setser, D. W. "Bound–Free Emission Spectra of Diatomic Xenon Halides." *Journal of Chemical Physics*. V(62). N(5). pp 1990-2000. **1975**.

Velazco, J. E.; Kolts, J. H.; Setser, D. W. "Quenching Rate Constants for Metastable Argon, Krypton, and Xenon Atoms by Fluorine Containing Molecules and Branching Ratios for XeF* and KrF* Formation." *Journal of Chemical Physics*. V(65). N(9). pp 3468-3480. **1976**.

Viereck, R. A.; Mende, S. B.; Murad, E.; Swenson, G. R.; Pike, C. P.; Culbertson, F. L.; Springer, R. C. "Spectral Characteristics of Shuttle Glow." *Geophysical Research Letters*. V(19). N(12). pp 1219-1222. **1992**.

Vikor, L.; Al–Khalili, A.; Danared, H.; Djuric, N.; Dunn, G. H.; Larsson, M.; Le Padellec, A.; Rosén, S.; af Ugglas, M. "Branching Fractions in the Dissociative Recombination of NH_4^+ and NH_2^+ Molecular Ions." *Astronomy and Astrophysics*. V(344). pp 1027-1033. **1999**.

Vilensky, B.; Havlin, S.; Taitelbaum, H.; Weiss, G. "Momentum Effects in Reaction–Diffusion Systems." *Journal of Physical Chemistry*. V(98). pp 9325-9328. **1994**.

Wang, L.-S.; Reutt, J. E.; Lee, Y. T.; Shirley, D. A. "High Resolution UV Photoelectron Spectroscopy of CO_2^+, COS^+ and CS_2^+ Using Supersonic Molecular Beams." *Journal of Electron Spectroscopy and Related Phenomena*. V(47). pp 167-186. **1988**.

Wavefunction, Inc.; Q–Chem, Inc. *Spartan '02*. **2001**.

Weller, C. S.; Biondi, M. A. "Measurements of Dissociative Recombination of CO_2^+ Ions with Electrons." *Physical Review Letters*. V(19). N(2). pp 59-61. **1967**.

Western, C. *Diatomic Molecule Spectrum Simulation, http://www.chm.bris.ac.uk/-pt/western/pgopher/simulate.htm*. Bristol University Chemistry Dept.. Bristol, U.K.. **2002**.

Willey, K. F.; Yeh, C. S.; Robbins, D. L.; Duncan, M. A. "Photodissociation Spectroscopy of Mg^+-CO_2." *Chemical Physics Letters*. V(192). N(2, 3). pp 179-184. **1992**.

Williams, T. L.; Babcock, L. M.; Adams, N. G. "Dissociative Charge Transfer in Reactions of CCl_4 and SF_6 with Ions Having Recombination Energies Between 6.4 eV and 24.5 eV." *International Journal Of Mass Spectrometry*. V(187). N(Spec. Issue SI). pp 759-772. **1999**.

Wilson Jr., E. B. "The Statistical Weights of the Rotational Levels of Polyatomic Molecules, Including Methane, Ammonia, Benzene, Cyclopropane and Ethylene." *Journal of Chemical Physics*. V(3). pp 276-285. **1935**.

Wlodek, S.; Fox, A.; Bohme, D. K. "Gas–Phase Reactions of Si^+ and $SiOH^+$ with Molecules Containing Hydroxyl Groups: Possible Ion–Molecule Reaction Pathways toward Silicon Monoxide, Silanoic Acid, and Trihydroxy–, and Trimethoxy–, and Triethoxysilane." *Journal of the American Chemical Society*. V(109). pp 6663-6667. **1987**.

Wu, W.; Prior, M. H.; Bräuning, H. "Orientation–Dependent Dissociative Charge Transfer." *Physical Review A*. V(57). N(1). pp R5-R8. **1998**.

Wulf, E.; von Zahn, U. "The Shuttle Environment: Effects of Thruster Firings on Gas Density and Composition in the Payload Bay." *Journal of Geophysical Research*. V(91). N(A3). pp 3270-3278. **1986**.

Wyttenbach, T.; Evard, D. D.; Maier, J. P. "Two–Photon Absorption Spectroscopy of Ion Beams: CO_2^+ $\tilde{C}$ $^2Sigma_g^+$ State Characterization." *Journal of Chemical Physics*. V(90). N(9). pp 4645-4650. **1989**.

Yarkony, D. R. "On the Quenching of Helium 2^3S: Potential Energy Curves for, and Nonadiabatic, Relativistic, and Radiative Couplings Between, the $a^3\Sigma_u^+$, $A^1\Sigma_u^+$, $b^3\Pi_g$, $B^1\Pi_g$, $c^3\Sigma_g^+$, and $C^1\Sigma_g^+$ States of He_2." *Journal of Chemical Physics*. V(90). N(12). pp 7164-7175. **1989**.

Yee, J. H.; Abreu, V. J. "Visible Glow Induced by Spacecraft–Environment Interaction." *Geophysical Research Letters*. V(10). N(2). pp 126-129. **1983**.

Yee, J.-H.; Abreu, V. J.; Dalgarno, A. "The Atmosphere Explorer Optical Glow Near Perigee Altitudes." *Geophysical Research Letters*. V(12). N(10). pp 651-654. **1985**.

Yu, Y. C.; Setser, D. W. "Krypton Chloride and Krypton Fluoride Thermochemistry and Formation and Relaxation Kinetics in Ar, N_2, and CF_4 Buffer Gas." *Journal of Physical Chemistry*. V(94). pp 2934-2942. **1990**.

Zaiffman, D. et al. Eds: Larsson, M.; Mitchell, J. B. A.; Schneider, I. F. "Dissociative Recombination Cross Section of Simple Molecular Ions: From Initial States to Final States" in *Proceedings of the 1999 Conference on Dissociative Recombination: Theory, Experiment and Applications, IV*. World Scientific. Singapore. **2000**.

Zanetti, G. "Counting Hydrodynamic Modes in Lattice Gas Automata Models." *Physica D*. V(47). pp 30-35. **1991**.

Zare, R. N.; Bernstein, R. B. "State–to–State Reaction Dynamics." *Physics Today*. V(Nov.). pp 43-50. **1980**.

Zhirnov, N. I.; Shadrin, O. P. "Calculation of Franck–Condon Factors with Pschl–Teller Wavefunctions I: The Probabilities of Some Vibrational Transitions in the $D^1\Sigma_u^+$–$B^1\Pi_g$ Band System of the He_2 Molecule.." *Optics and Spectroscopy (English Translation)*. V(24). pp 478-480. **1968**.

Zhou, Y.; Ratnavelu, K.; Zhong, Z.; McCarthy, I. E. "Electron Impact on Excited Helium." *Journal of Physics B: Atomic and Molecular Optical Physics*. V(31). pp L959-L965. **1998**.

Zipf, E. C. "The OI(^1S) State: Its Quenching by O_2 and Formation by the Dissociative Recombination of Vibrationally Excited O_2^+ Ions." *Geophysical Research Letters*. V(6). N(10). pp 881-884. **1979**.

Zipf, E. C. "The Dissociative Recombination of Vibrationally Excited N_2^+ Ions." *Geophysical Research Letters*. V(7). N(9). pp 645-648. **1980 a**.

Zipf, E. C. "A Laboratory Study on the Dissociative Recombination of Vibrationally Excited O_2^+ Ions." *Journal of Geophysical Research*. V(85). N(A8). pp 4232-4236. **1980 b**.

Zittel, P. F.; Sedam, M. A. "Vibrational Relaxation of OCS(ν_3)." *Chemical Physics Letters*. V(148). N(6). pp 486-491. **1988**.

Zong, W.; Dunn, G.; Djurić,N.; Larsson, M.; Greene, C.; Al–Khalili, A.; Neau, A.; Derkatch, A.; Vikor, L.; Shi, W.; Le Padellec, A.; Rosén, S.; Danared, H.; af Ugglas, M. A. "Resonant Ion Pair Formation in Electron Collisions with Ground State Molecular Ions." *Physical Review Letters*. V(83). N(5). pp 951-954. **1999**.

Zwillinger, D. *Handbook of Differential Equations, 2^{nd} Ed.*. Academic Press, Limited. Unknown City. **1992**.

APPENDIX A

REFERENCES CATEGORIZED BY CONTENT

Computational and Theoretical — DR

Theory and/or summaries of computational techniques:

Adams, 1975 Bates, 1992 Bates, 1993a
Bates, 1993b Bates, 1994 Burke, 1988
Dalgarno, 2000 Golubkov, 2000 Halkier, 1998
Hardy, 1998 Harland, 1998 Larson, 2000
Pichl, 2000 Schneider, 2000b Tachikawa, 1999
Talbi, 2000 Tennyson, 2000 Tomashevsky, 1998
Urbain, 2000 van der Zande, 2000

Description or use of a computational technique (computations involving any mechanism of DR any any type of product):

Bates, 1992 Carata, 1999 Carata, 2000a
Carata, 2000b Golubkov, 2000 Guberman, 1991
Hardy, 1998 Jursic, 1999 Jursic, 2000
Ketvirtis, 1999 Kraemer, 1998 Müller, 2000b
Mulliken, 1964 Orel, 2000 Osamura, 1999
Schneider, 2000a Tachikawa, 1999 Tagaki, 2000
Tennyson, 2000 Tomashevsky, 1998

Comparison of theory or computation to experimental data:

Adams, 1994 Guberman, 1991 Lange, 1999
Schneider, 2000a

Computational and Theoretical — Flow or Reactive Flow Simultation

Theory and/or summaries of computational techniques:

Abdallah, 1999 Callear, 1978 Cornubert, 1991
Dubrulle, 1991 Ernst, 1991 Frisch, 1991
Gerling, 1991 Hasslacher, 1991 Higuera, 1993
Jenny, 1950 Kong, 1991 Landau, 2000
Monnery, 1995 Petravic, 1995 Shukla, 1994
van der Berg, 1993

Description or use of a computational technique:

Banavar, 1991 Baras, 1995 Bernardin, 1991
Brieger, 1991 Brosa, 1989 Brosa, 1990
Brosa, 1991 Chenoweth, 1972 Cliffe, 1991
Foley, 1997 Frisch, 1986 Green, 1995
Kohring, 1991a Kohring, 1991b Kohring, 1991c
Kohring, 1993 Landau, 2000 LeVallée, 1991
Lawniczak, 1991 Macagno, 1967 Miyazaki, 1995
NATO, 1991 Noullez, 1991 Oran, 1992
Rapaport, 1988 Riccardi, 1991 Segatz, 1996
Singh, 1994 Succi, 1991 Uehigashi, 1992
van der Hoef, 1991 Vilensky, 1994 Zanetti, 1991

Comparison of theory or computation to experimental data:

Macagno, 1967 Segatz, 1996 (?) Shukla, 1994

Computational and Theoretical — Other

Atmospheric and Interstellar:

Anderson, 1978 Duley, 1992 Fox, 1999
Fox, 2000 Osamura, 1999

Spacecraft:

Caledonia, 1993 Murad, 1986 Swenson, 1991

Spectroscopic (including structural calculations):

Blöcker, 1990 Castaño, 1983 Dunning, 1978
Engler, 1984 Ginter, 1970 Guberman, 1972
Guberman, 1975 Hardy, 1998 Herzberg, 1971
Herzberg, 1987 Herzberg, 1988 Herzberg, 1989
Kraemer, 1998 Lefebvre–Brion, 1986
LeRoy, 1993 LeRoy, 1988 Lo, 1994
Luo, 1999 Mies, 1974 Morse, 1929
Ortenberg, 1963 Read, 1980 Resende, 2001
Sawaryn, 1989 Wilson, 1935 Yarkony, 1989
Zhirnov, 1968

Reactions (including those involving electrons and/or photons, collisions, scattering and relaxation processes):

Allison, 1966 Burshtein, 1994 Child, 1983
Davies, 1998 Drukarev, 1965 Dunning, 1978
Errea, 1994 Gianturco, 1988 Harland, 1998
Herbst, 2000 Lambert, 1977 Laudenschlager, 1974
Lengyel, 1992 Marci, 1998 McCarthy, 1995
Miller, 1994 Miller, 1968 Miller, 1979
Mistrik, 2000 Rabadan, 1999 Rapp, 1965
Resende, 2001 Schattschneider, 1986
Zhou, 1998

Experimental Technique and Apparatus

Experimental techniques and apparatus descriptions for study involving e–i recombination:

Adams, XXXX Habs, 1989 Hardy, 2000
Janev, 2000 Jensen, 1999 Krohn, 2000
Lange, 2000 Mitchell, 2000 Tanabe, 2000
Zong, 1999

Other experimental technique (includes fluid flow):

Adams, 1975 Bastian, 1968 Dudeck, 1983
Ferguson, 1969 Hardalupas, 1992 Harland, 1998
Hiller, 1983 Hollo, 1994 McDaniel, 1983
Mitchell, 2000 Schmeltekopf, 1963 Smith, 1972
Smith, 1975

Experimental Results — DR

Kinetics:

Adams, XXXX Adams, 1984 Amitay, 1999
Glosík, 2000 Jensen, 1999 Johnsen, 1987
Johnsen, 2000a Lange, 1999 Larsson, 1996
Larsson, 1998 Laubé, 1998 a,b LePadellec, 1997
Mostefaoui, 2000 Rebrion–Rowe, 2000a
Rebrion–Rowe, 2000b Schneider, 2000 a
Semaniak, 1998 Strömholm, 1996 Torr, 1981
Weller, 1967 Zipf, 1980 a Zipf, 1980 b
Zong, 1999

† *These are the papers that focus on RIP formation. Other papers in this section might also mention the process.*

‡ *Although spectroscopic products of DR also imply particulate products, these entries are not duplicated in the "Products, general" section.*

Spectroscopic data (includes photoelectron and bound–unbound processes):

Baer, 1986 Baruch, 1992 Bhushan, 2000
Birge, 1926 Brau, 1975 Brown, 1971
Brown, 1973 Carrington, 1993 Carroll, 1962
Chow, 1971 Dixon, 1963 Ewing, 1975
Fairbairn, 1970 Ginter, 1965 a Ginter, 1965 b
Ginter, 1966 Gloersen, 1965 Golde, 1975
Headrick, 1930 Hearn, 1991 Heidner, 1969
Hoffman, 1976 Jacox, 2001 Jones, 1980
Judge, 1973 Kolts, 1978 Komatsu. 1992
Lawrence, 1971 Lawrence, 1972 Lichten, 1974
Loginov, 2000 Mies, 1966 Miller, 1975
Moazzen–Ahmadi, 1989 Müller, 2000 a
Orient, 1992 Orient, 1993 Orr, 1987
Orth, 1976 Pubanz, 1985 Ray, 1986
Read, 1934 Schloss, 1992 Sekiya, 1983
Slanger, 1976 Smith, 1967 Stark, 1987
Suzuki, 1978 Valée, 1986 Velazco, 1975
Wang, 1988 Willey, 1992 Wyttenbach, 1989
Yu, 1990 Zittel, 1988

Other physical data (atmospheric constituents, reactions in the ISM, spacecraft glows, etc.):

Banks, 1983 Chakrabarti, 1993 Chauvrin, 1992
Dunlop, 1995 Dupree, 1970 Grebowsky, 1987
Greer, 1993 Hernandez, 1971 Hunton, 1988
Koch, 1988 Kofsky, 1986 Krassovsky, 1962
Machuzak, 1993 McCay, 1978 Mende, 1983
Murad, 1990 Murad, 1986 Oka, 2000
Rind, 1993 Swenson, 1985 Torr, 1977
Torr, 1981 Viereck, 1992 Wulf, 1986
Yee, 1983 Yee, 1985

Other references

Mathematical techniques:

Abramowitz, 1965 Airy, 1873 Zwillinger, 1992

Physical data collections:

Adams, 1988 Huber, 2001 Jacox, 2001
NIST, 2001 Radzig, 1985

Handbooks and other general collections:

Drake, 1996 Eisberg, 1967 Graedel, 1989
Lang, 1978 Larsson, 2000 Radzig, 1985

Textbooks:

Atkins, 1998 Bueche, 1986 Hanly, 1996
Hazeltine, 1998 Herzberg, 1988 Herzberg, 1989
Knuth, 1986 Landau, 2000 Levine, 1983
McQuarrie, 1976 McQuarrie, 1983 Oran, 1987
Read, 1980 Steinfeld, 1985

APPENDIX B:

CHARACTERIZATION OF THE HELIUM–ARGON PLASMA

This appendix contains the spectra taken of the base plasma when only helium had been added and when both helium and argon had been added. Graphs of the spectra are grouped by wavelength range and numbered. Each number corresponds to two or three graphs: (a) the He–only spectrum; (b) the spectrum with Ar added; (c) for certain spectral ranges, the first two spectra plotted together. Appendix B-b is a key to the graphs in B-a and contains detailed spectral information regarding each of the labeled emissions.

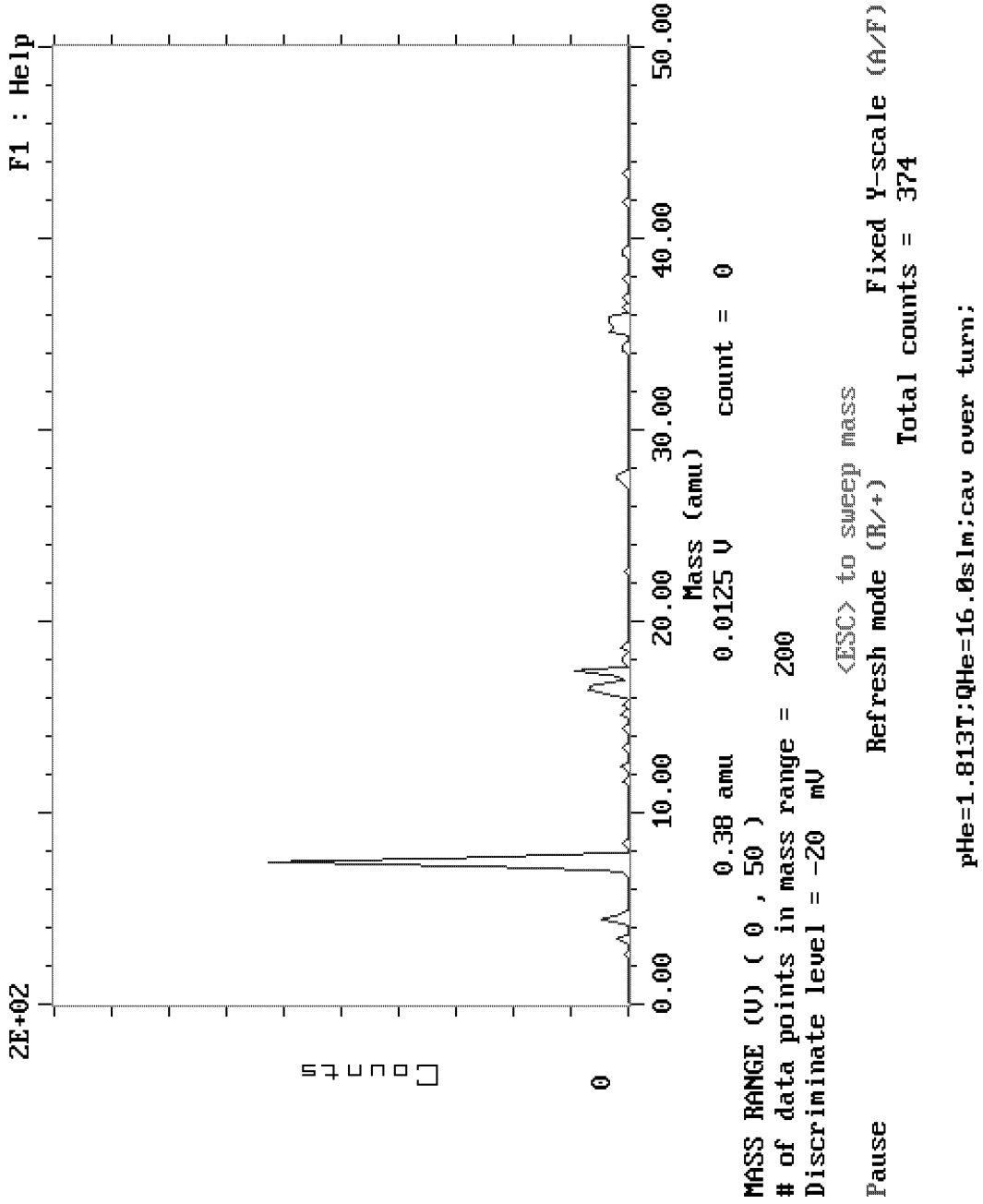
Atomic emissions are labeled with the graph number and a letter from the English alphabet. Below each atomic emission is the identity of the atom responsible for that emission, e. g.: $\overset{2D}{\underset{He}{}}$ signifies label D on graph 2 and that the emission is atomic helium emission.

All molecular species that are not explicitly identified on a graph are He_2. These emissions are labeled with the graph number, a letter from the Greek alphabet, and a short–hand notation for the electronic transition corresponding to the emission, e. g.: $2\gamma: \ n \rightarrow a$ signifies label γ on graph 2 and that the emission is the $n \rightarrow a$ emission of diatomic neutral helium.

B-a Graphs

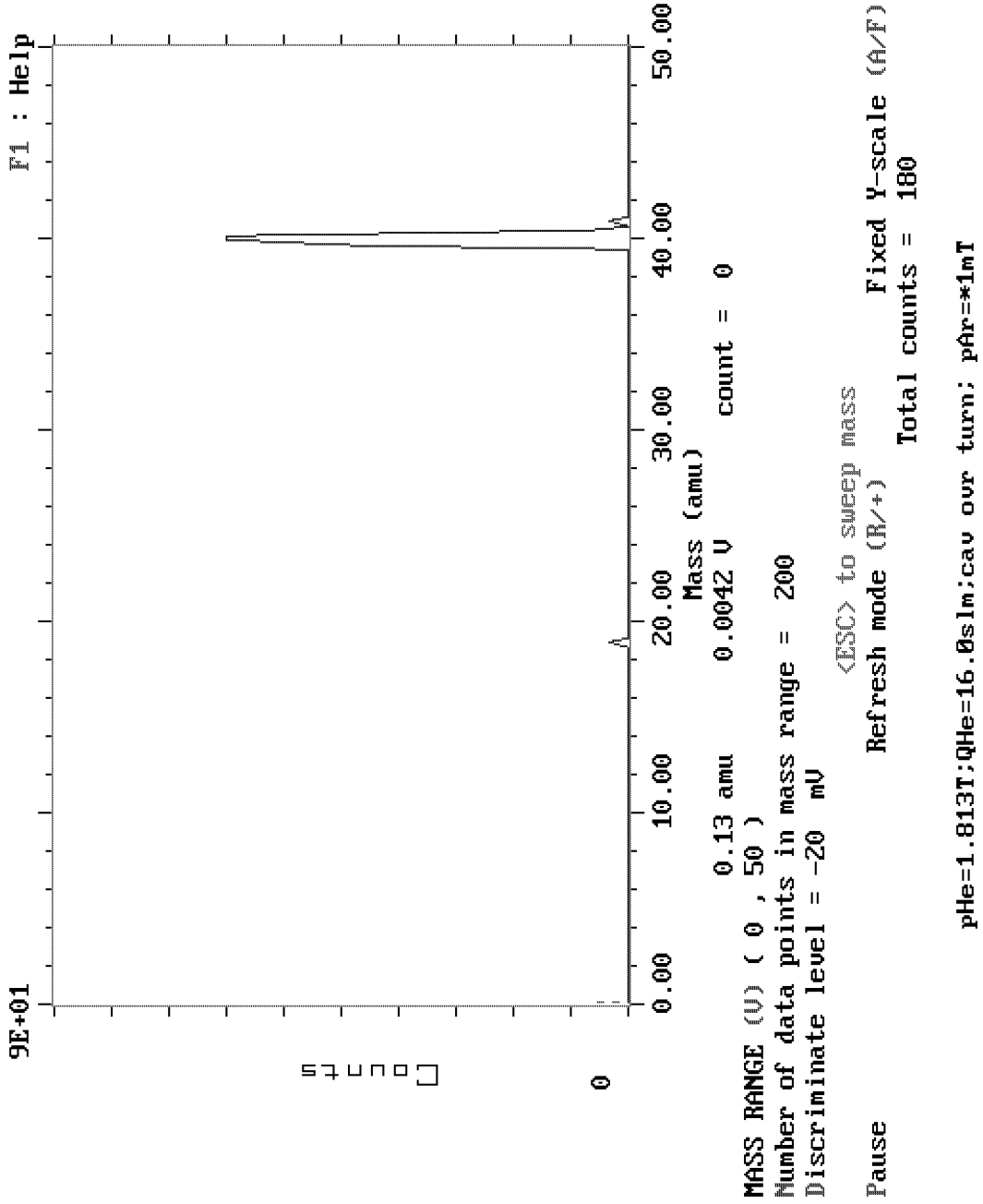
The first two graphs (i & ii) are mass spectra taken prior to taking the spectra. The graphs immediately following the mass spectra (iii & iv) contain emissions that were difficult to classify. No emissions from any know helium or argon species are present in those spectra. The other spectra are as described above.

Graph B-a.i



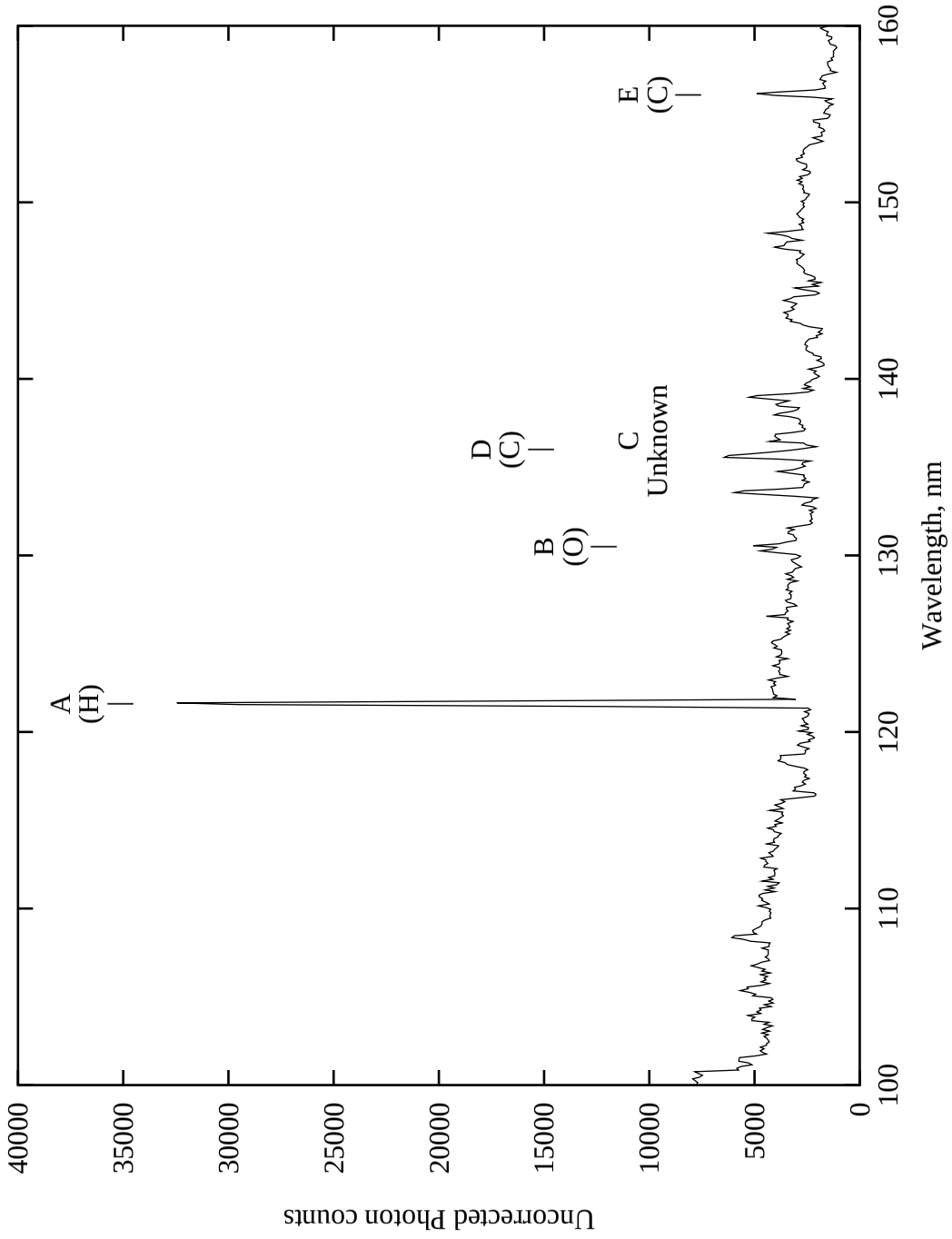
Graph B-a.i: Mass spectrum taken before the spectra were taken. The mass indicated by a peak is the number at the right of the peak. Only helium had been added when this spectrum was taken.

Graph B-a.ii



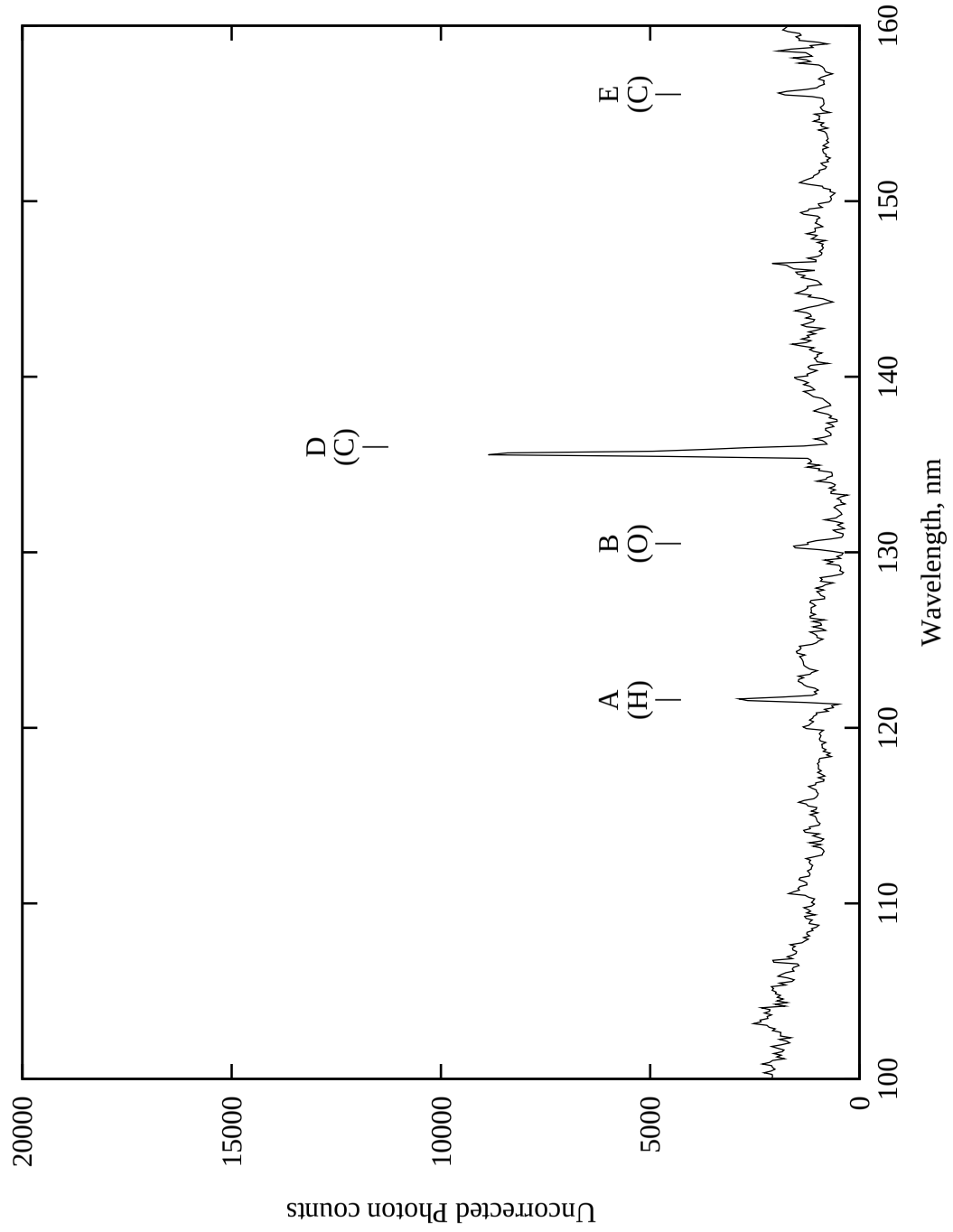
Graph B-a.ii: Mass spectrum taken before the spectra were taken. The mass indicated by a peak is the number at the right of the peak. This spectrum was taken after the addition of argon.

Graph B-a.iii (a)



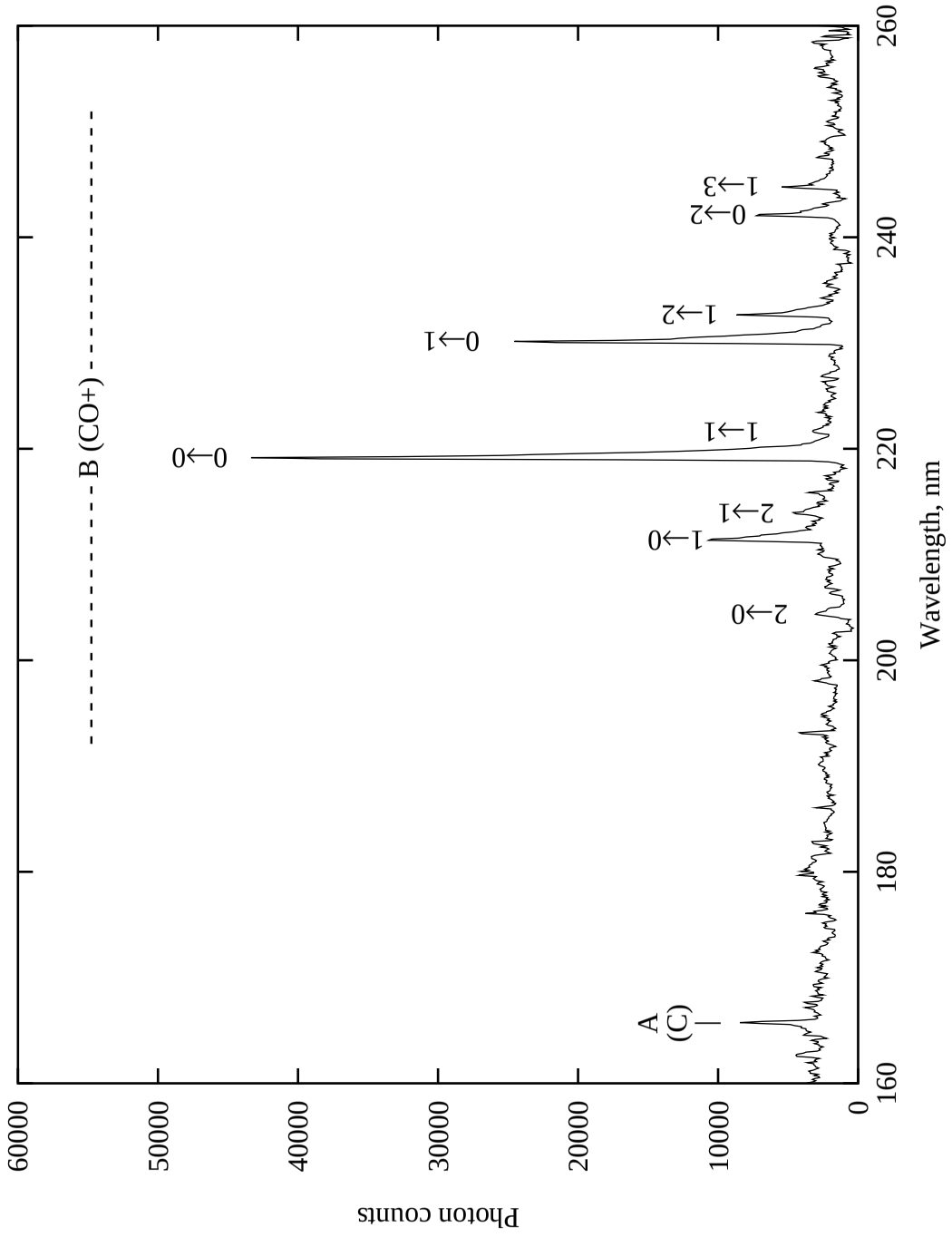
Graph B-a.iii (a): Spectrum of the helium plasma. *All assignments on this plot are tentative.* See Appendix B-b for assignments.

Graph B-a.iii (b)



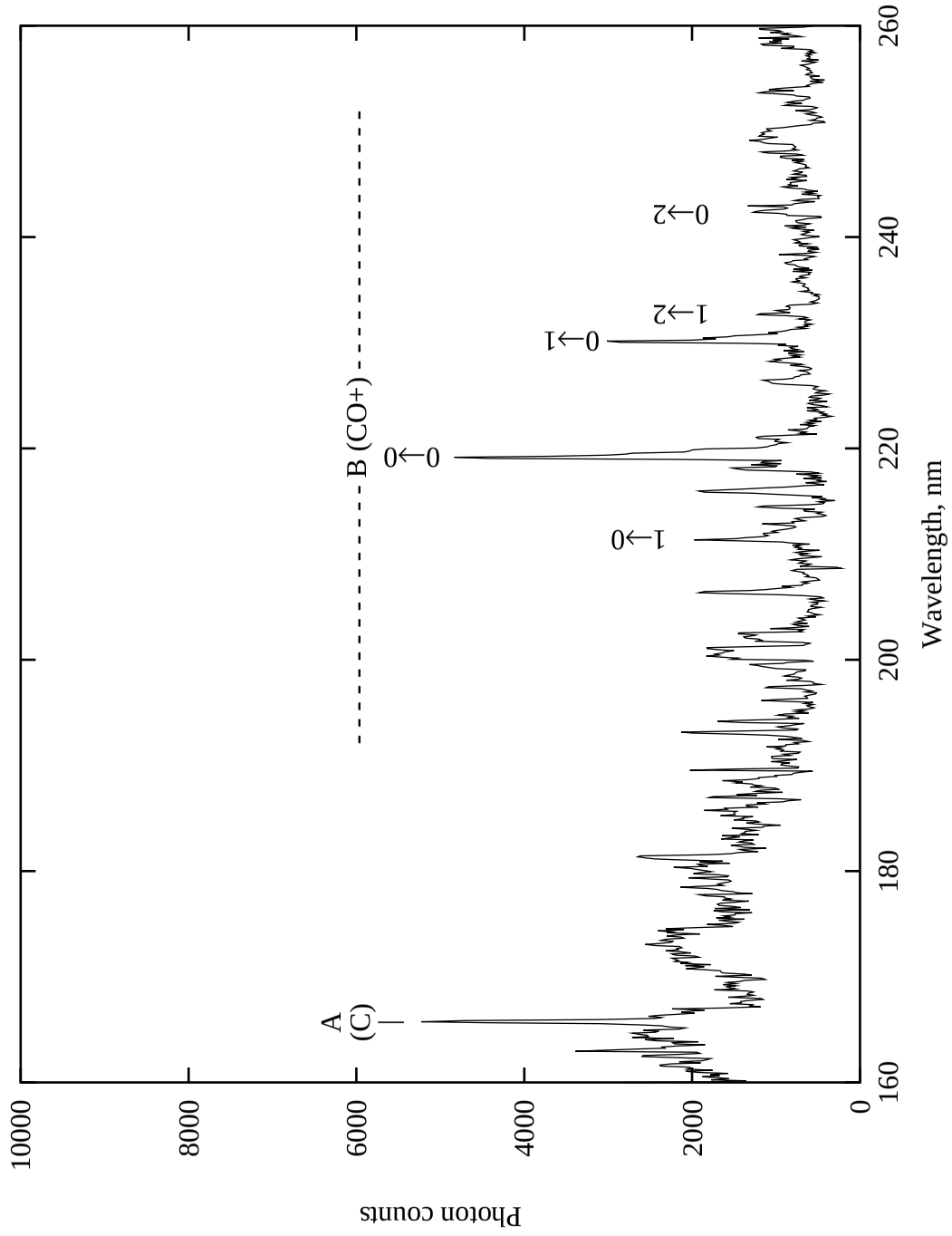
Graph B-a.iii (b): Spectrum of the helium-argon plasma. *All assignments on this plot are tentative.* See Appendix B-b for assignments.

Graph B-a.iv (a)



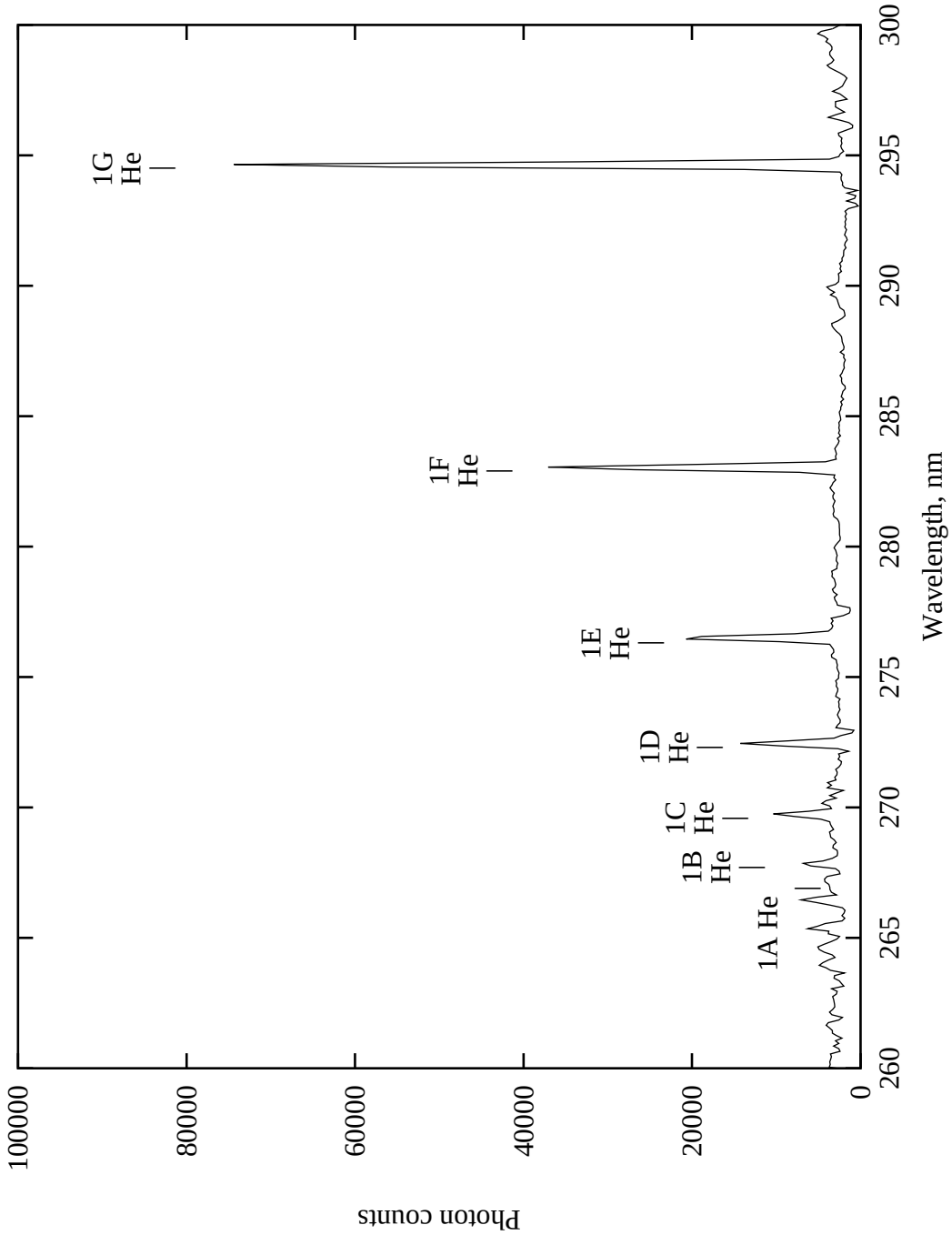
Graph B-a.iv (a): Spectrum of the helium plasma. *Atomic assignments on this plot are tentative.* See Appendix B-b for assignments.

Graph B-a.iv (b)



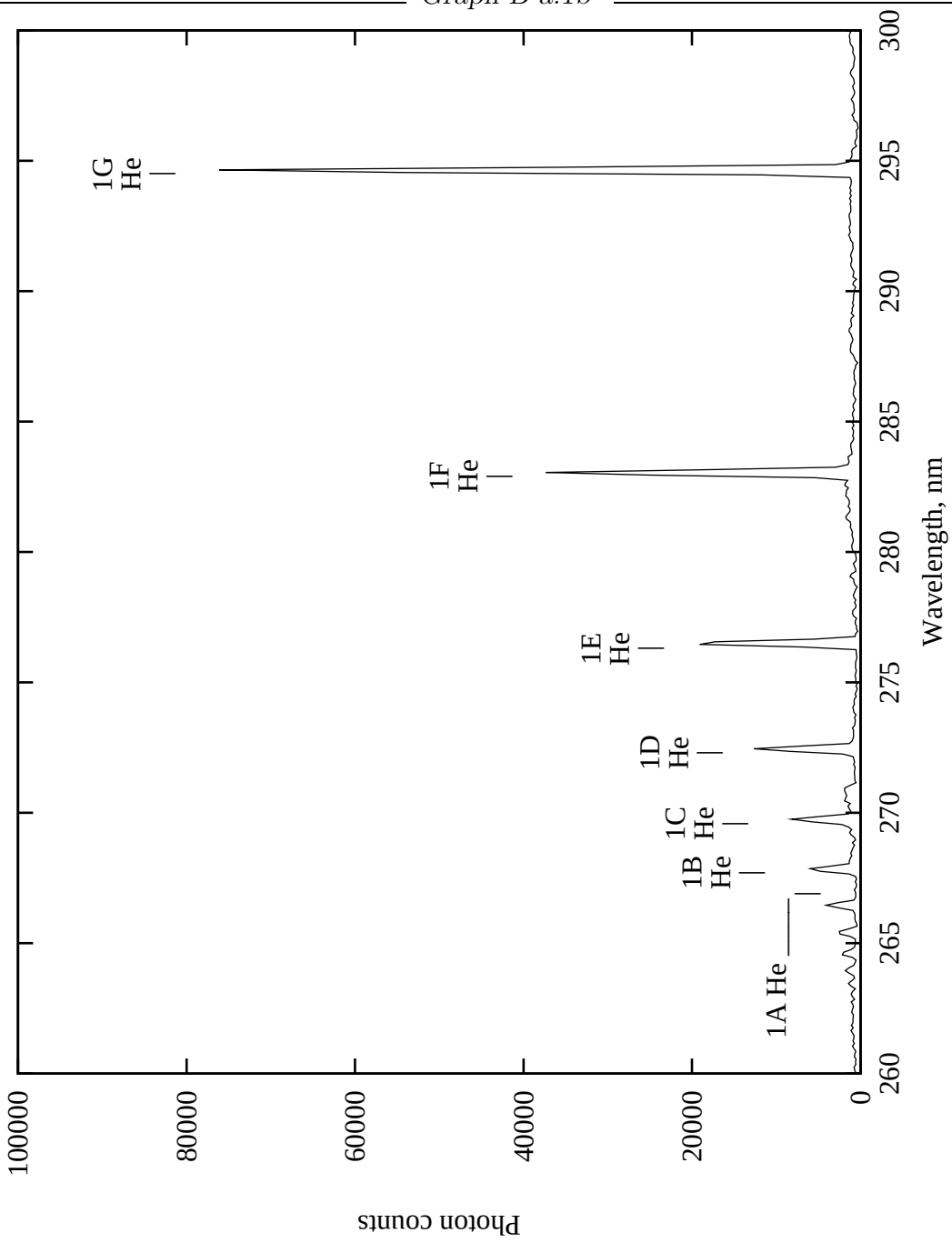
Graph B-a.iv (b): Spectrum of the helium-argon plasma. *Atomic assignments on this plot are tentative.* See Appendix B-b for assignments.

Graph B-a.1a



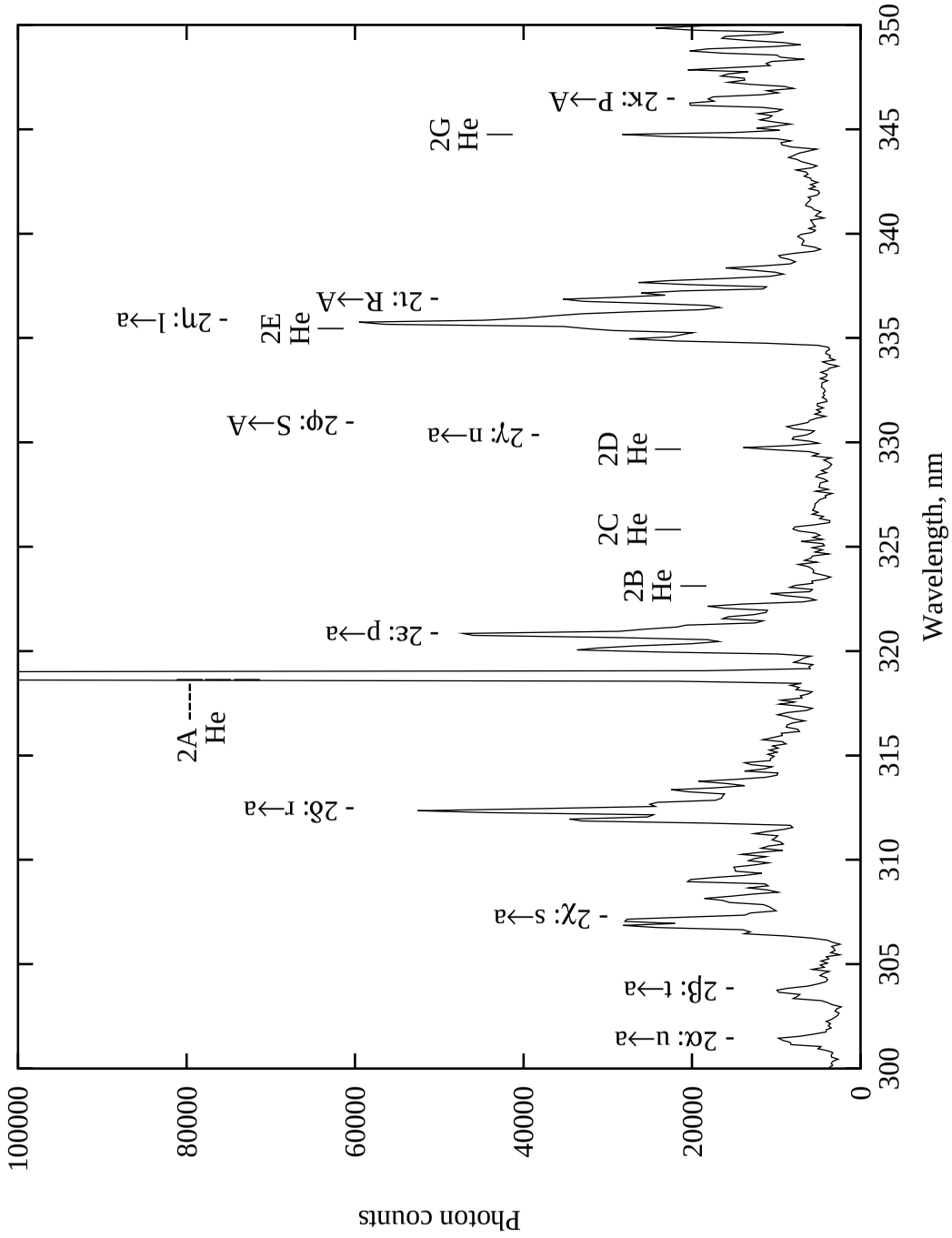
Graph B-a.1a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.1b



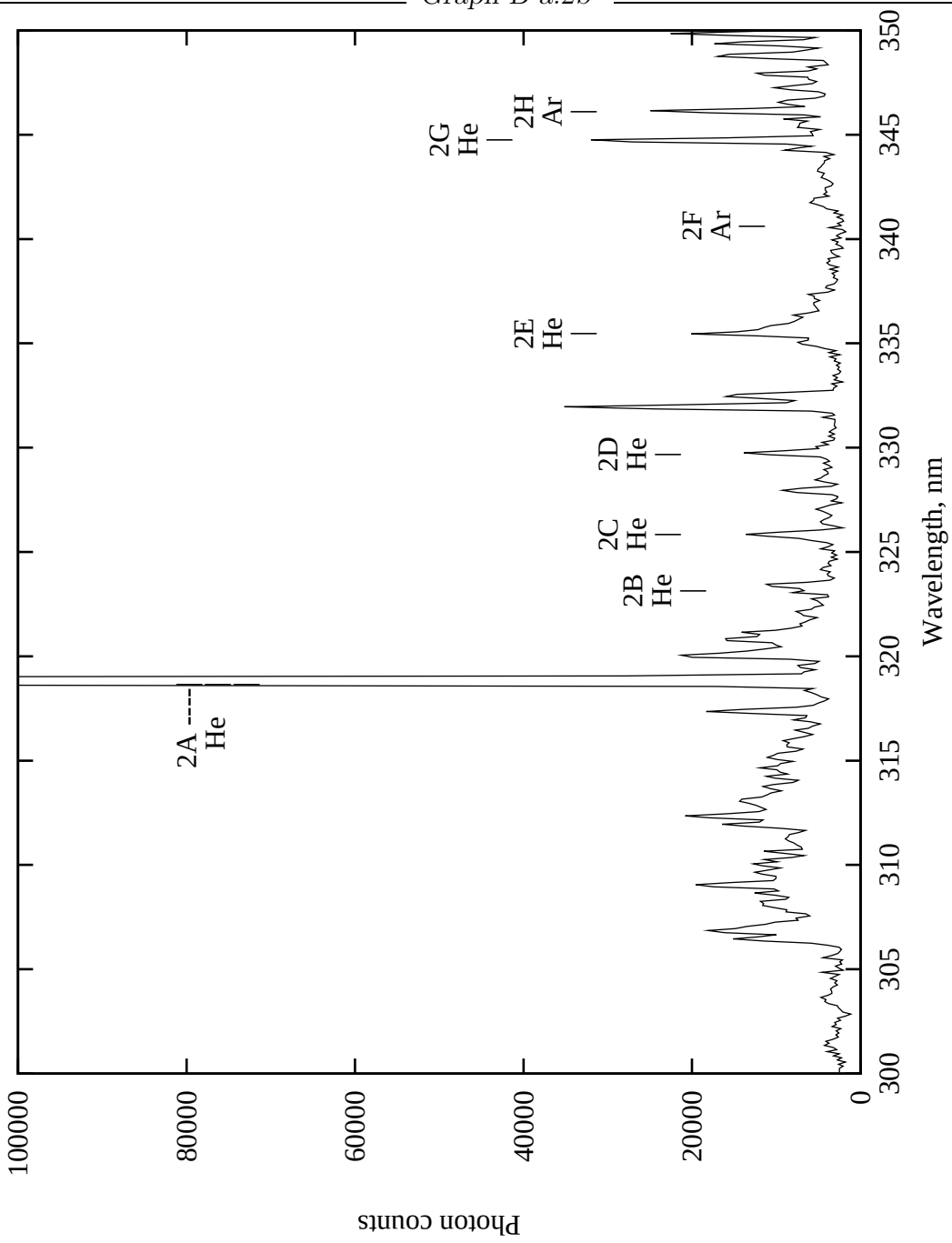
Graph B-a.1b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.2a



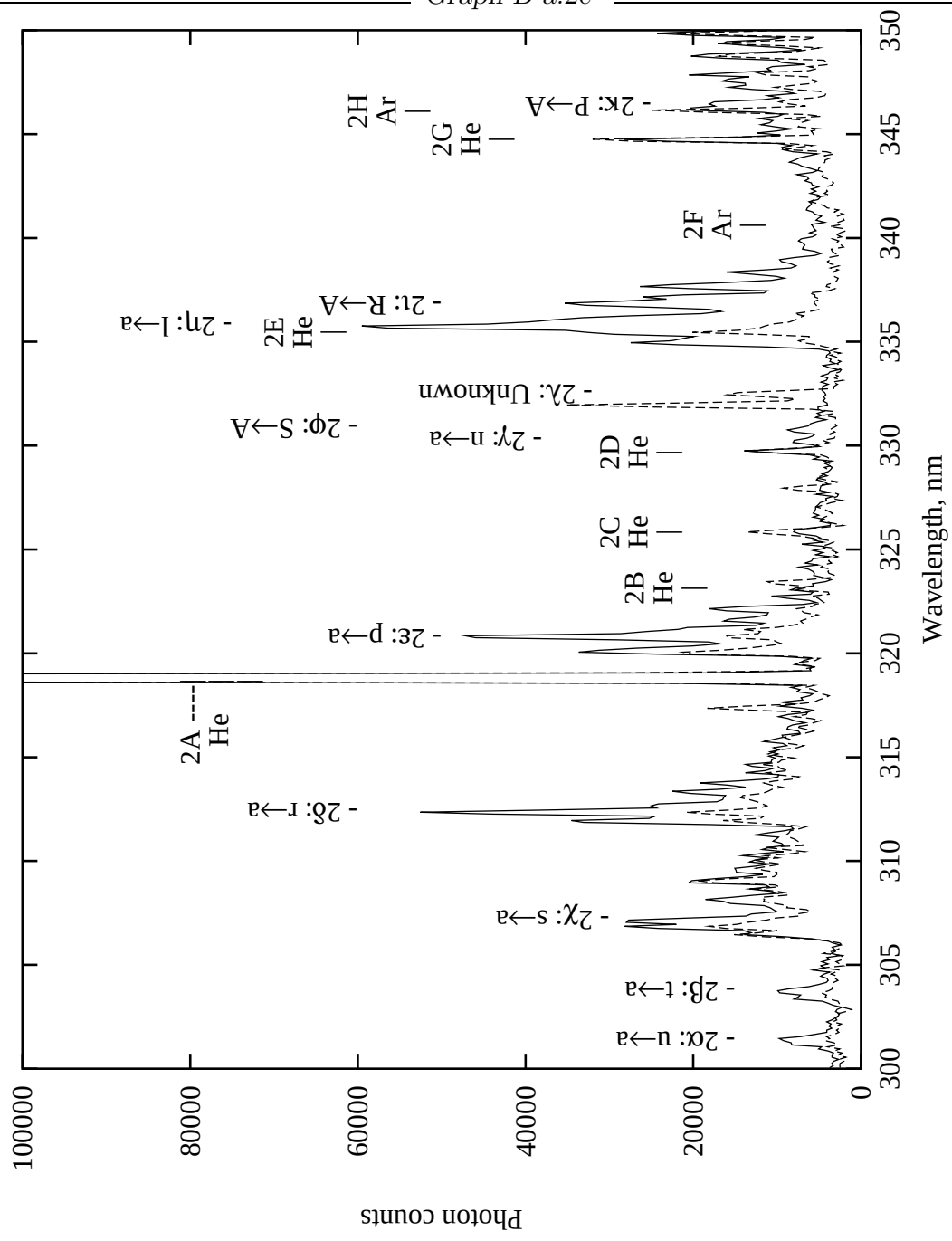
Graph B-a.2a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.2b



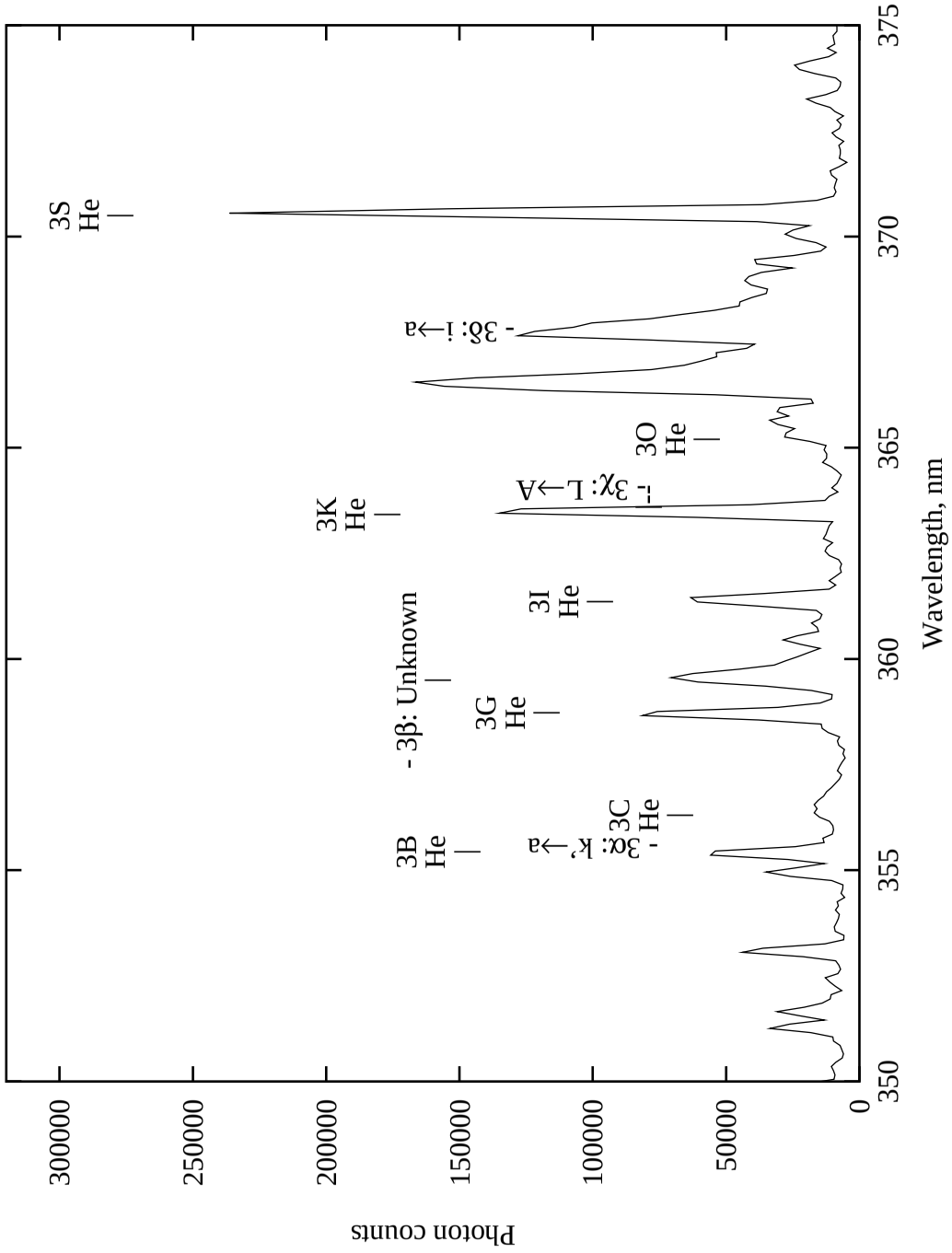
Graph B-a.2b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.2c



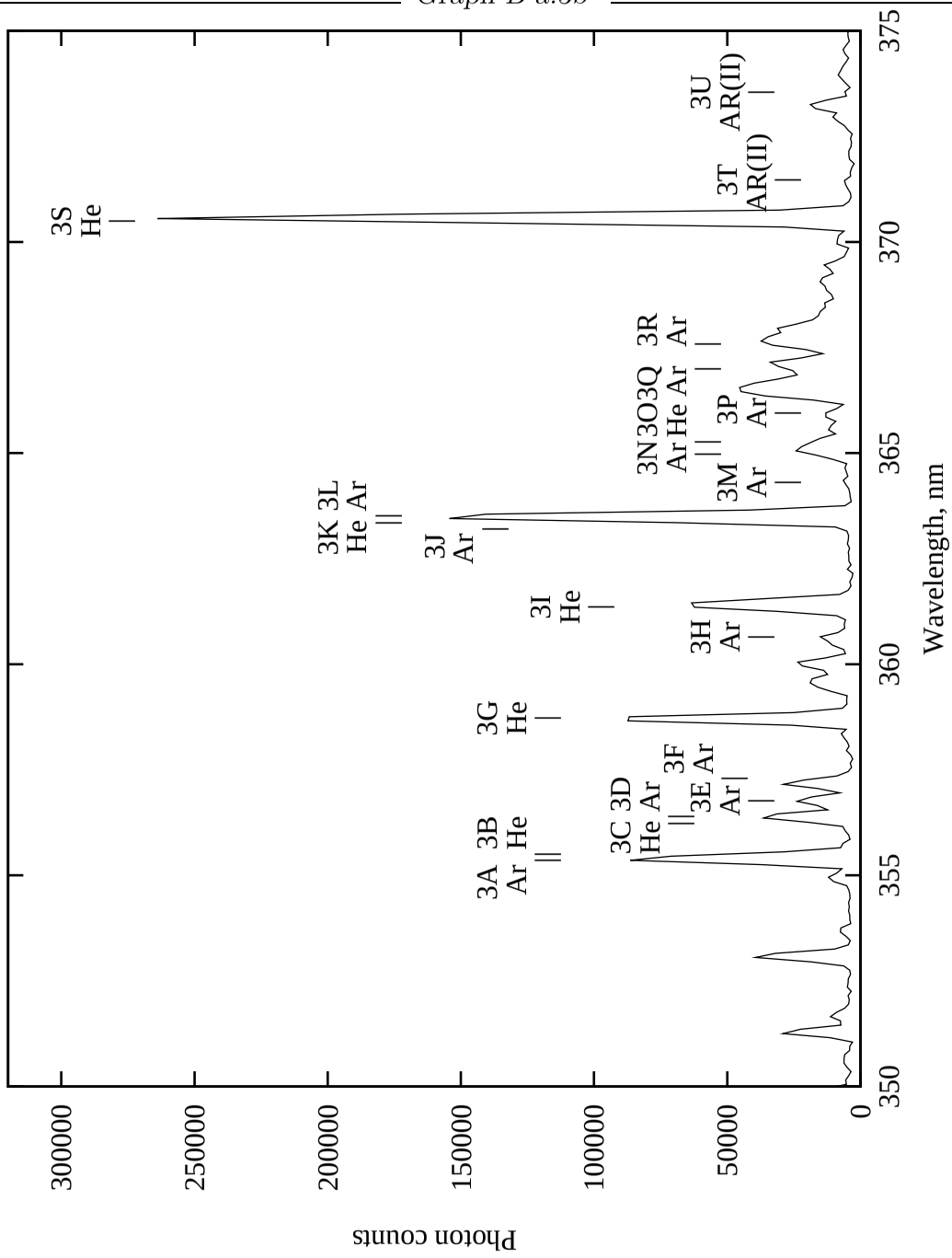
Graph B-a.2c: Direct comparison of spectra taken before (solid line) and after (dashed line) the addition of argon. The emissions are labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.3a



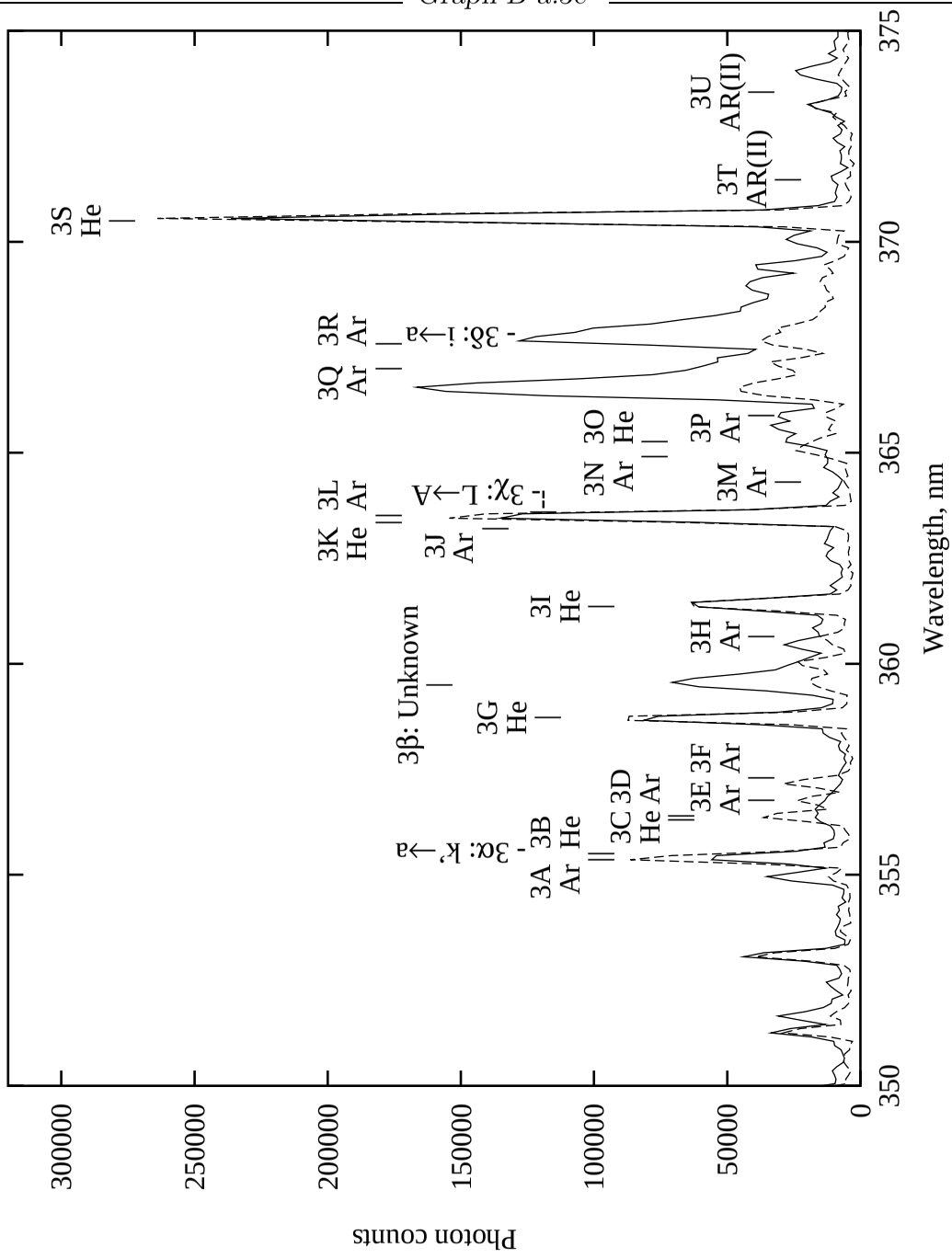
Graph B-a.3a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.3b



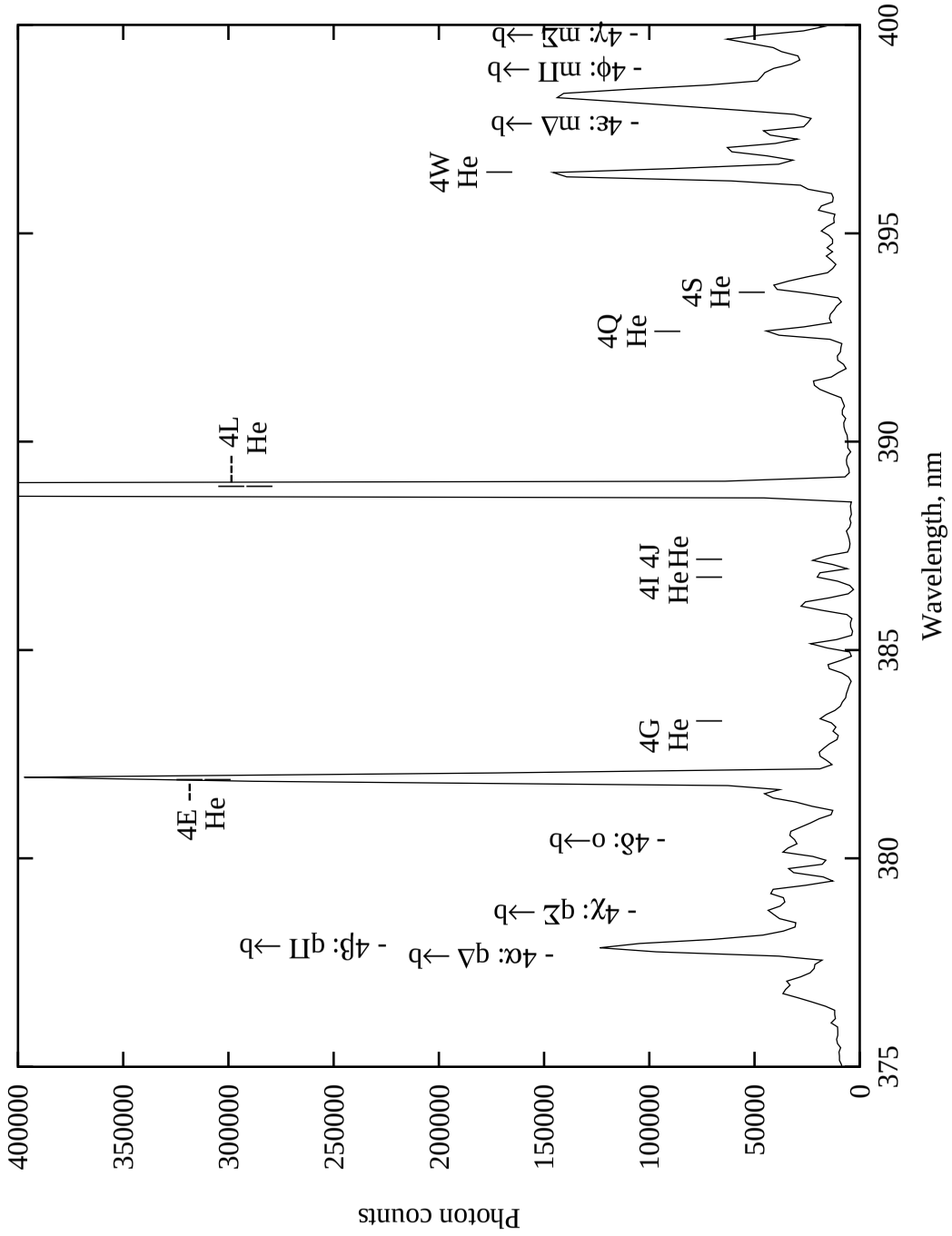
Graph B-a.3b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.3c



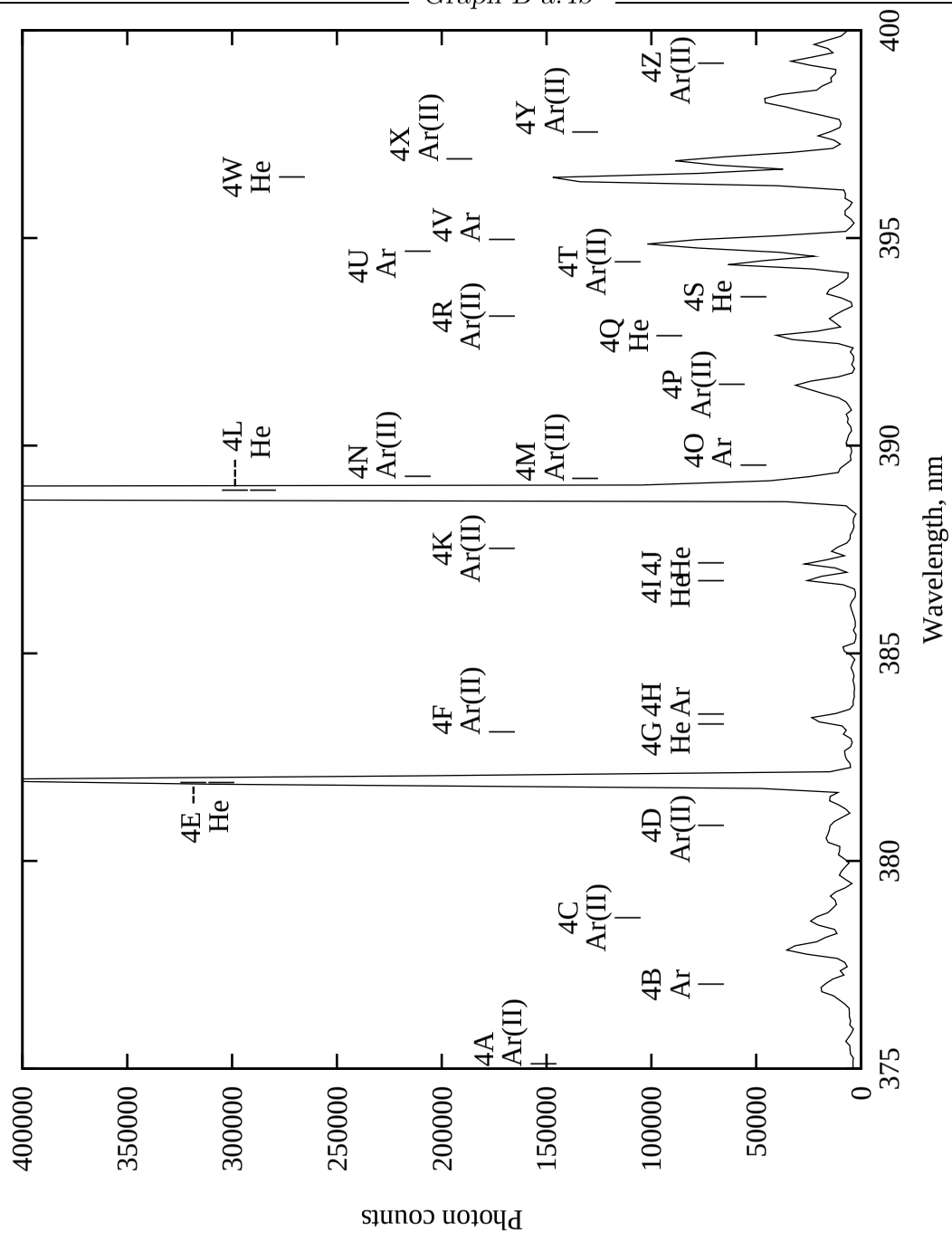
Graph B-a.3c: Direct comparison of spectra taken before (solid line) and after (dashed line) the addition of argon. The emissions are labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.4a



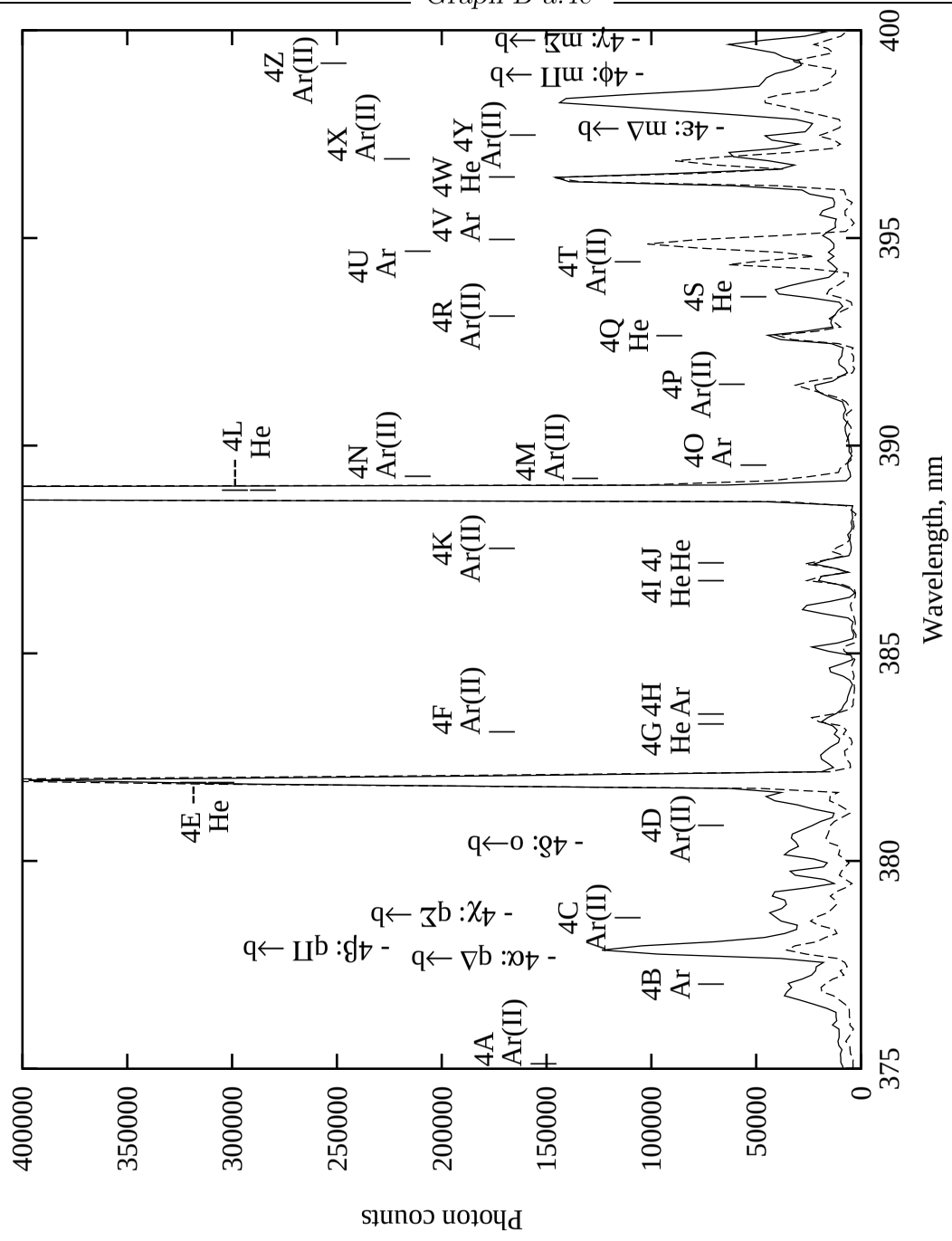
Graph B-a.4a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.4b



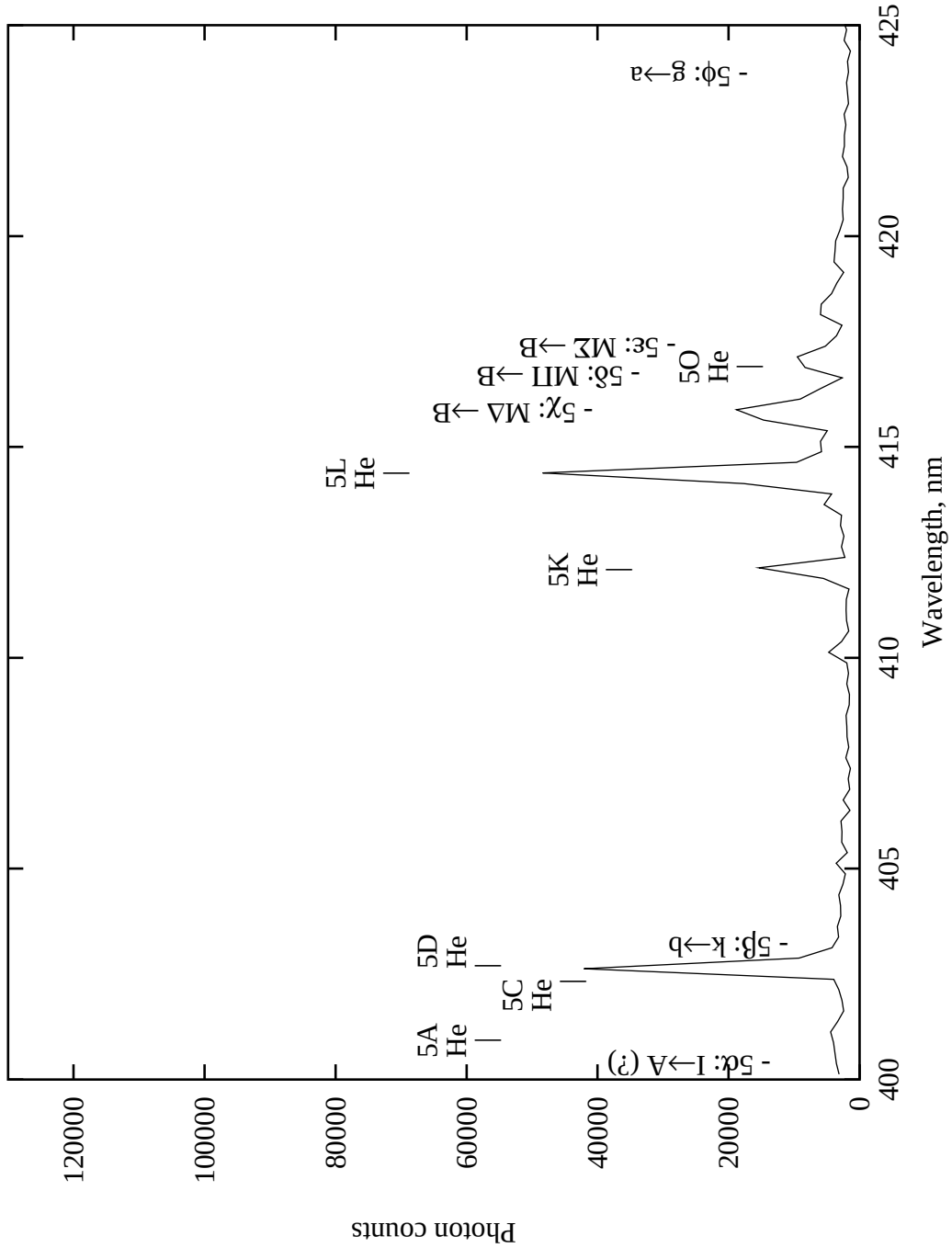
Graph B-a.4b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.4c



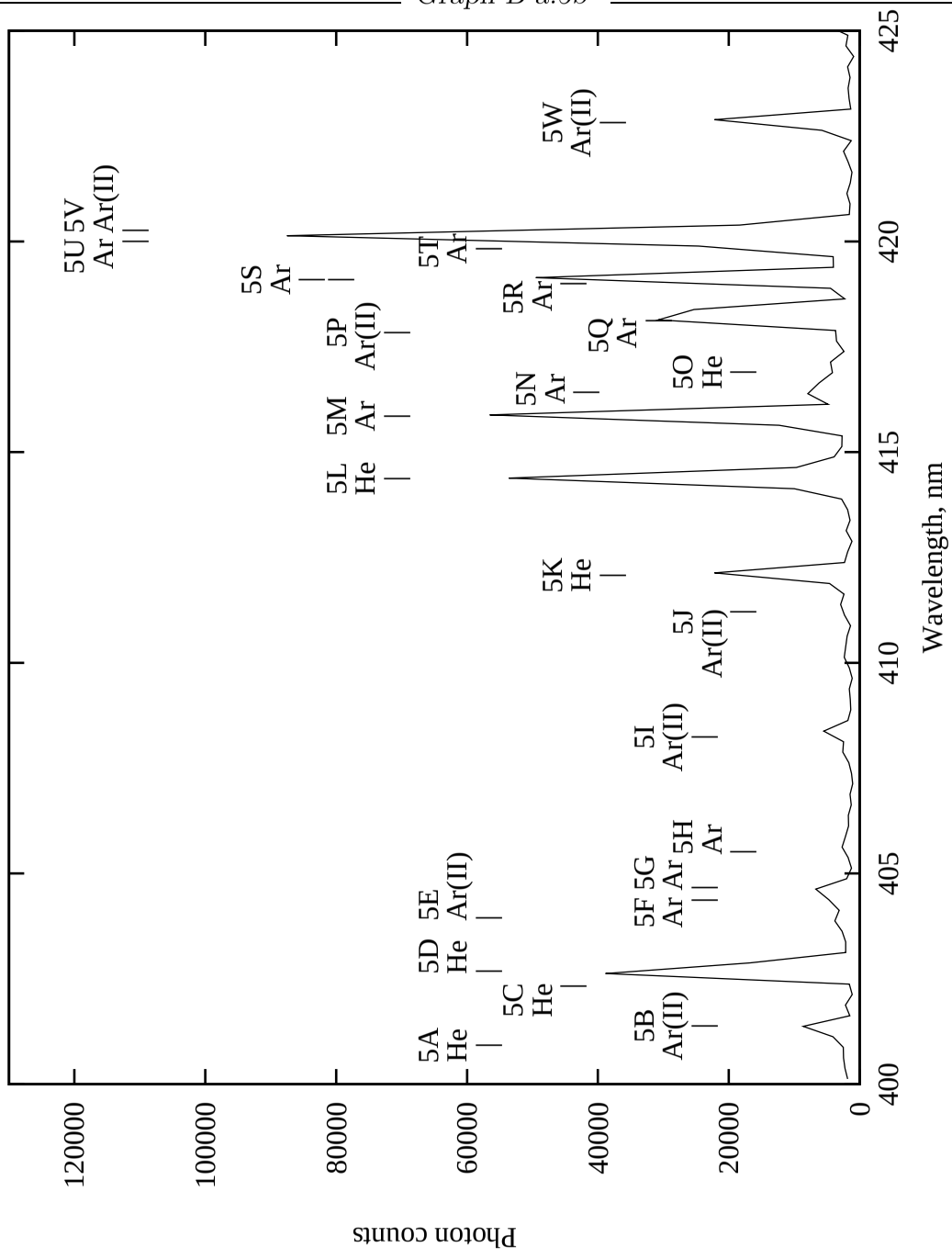
Graph B-a.4c: Direct comparison of spectra taken before (solid line) and after (dashed line) the addition of argon. The emissions are labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.5a



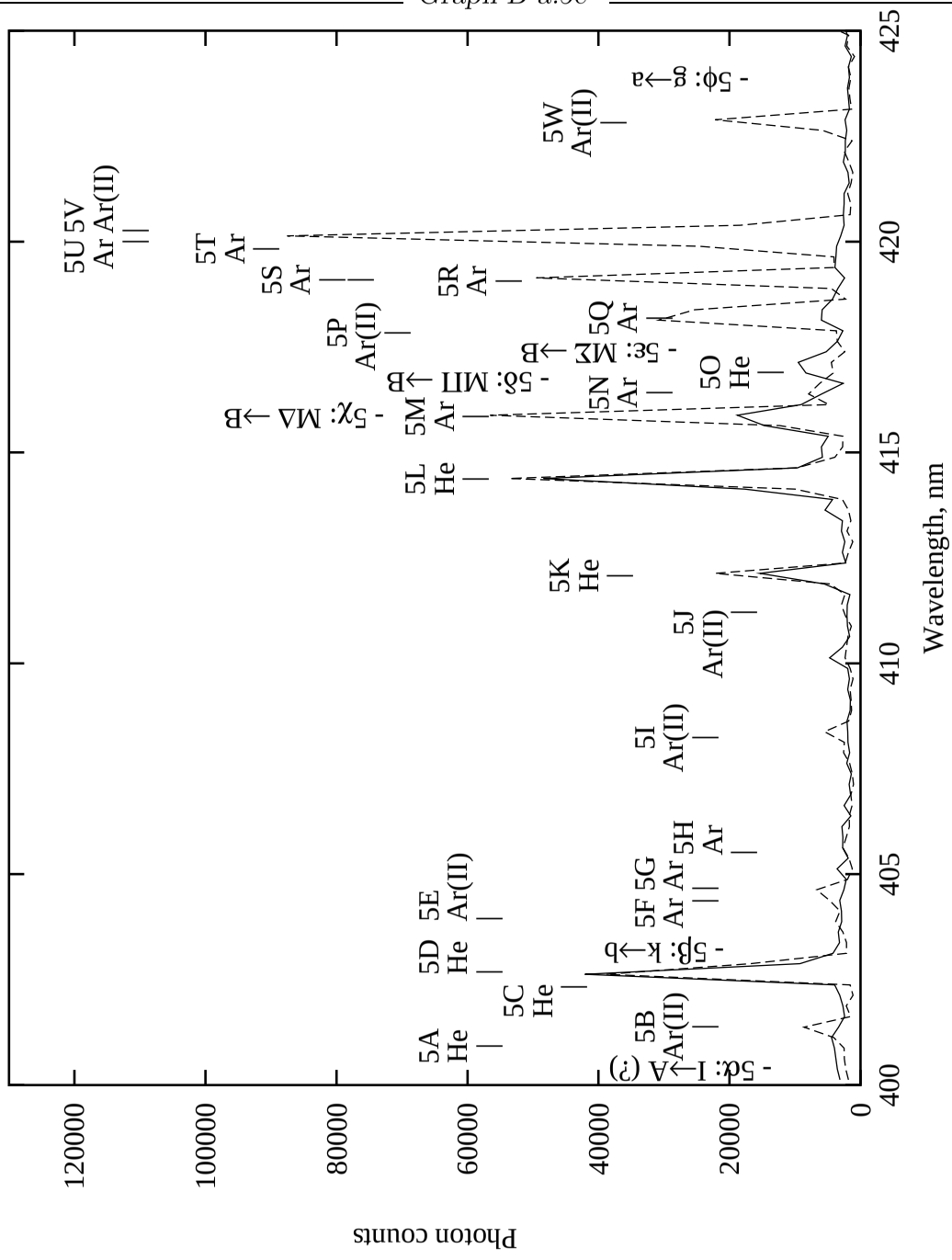
Graph B-a.5a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.5b



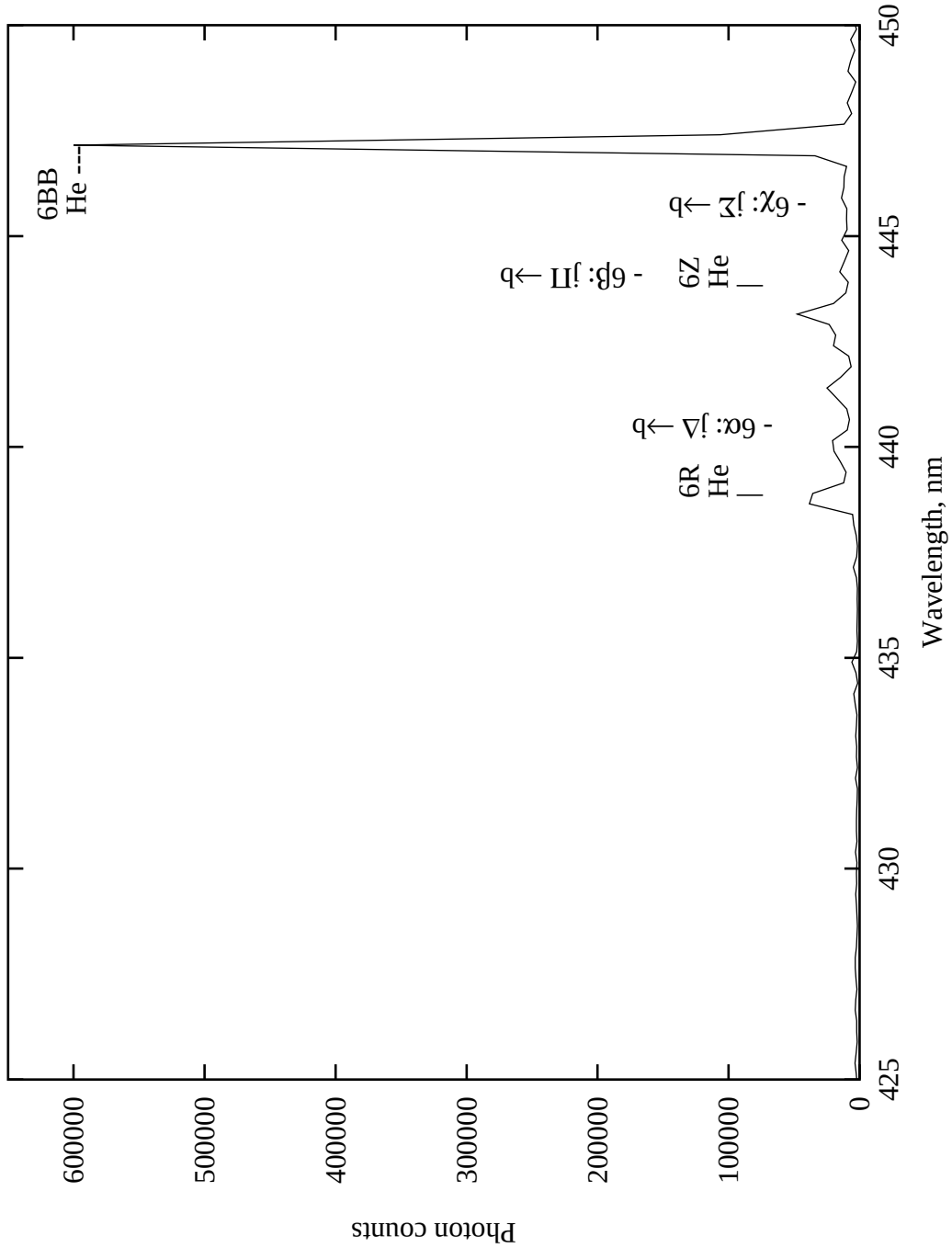
Graph B-a.5b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.5c



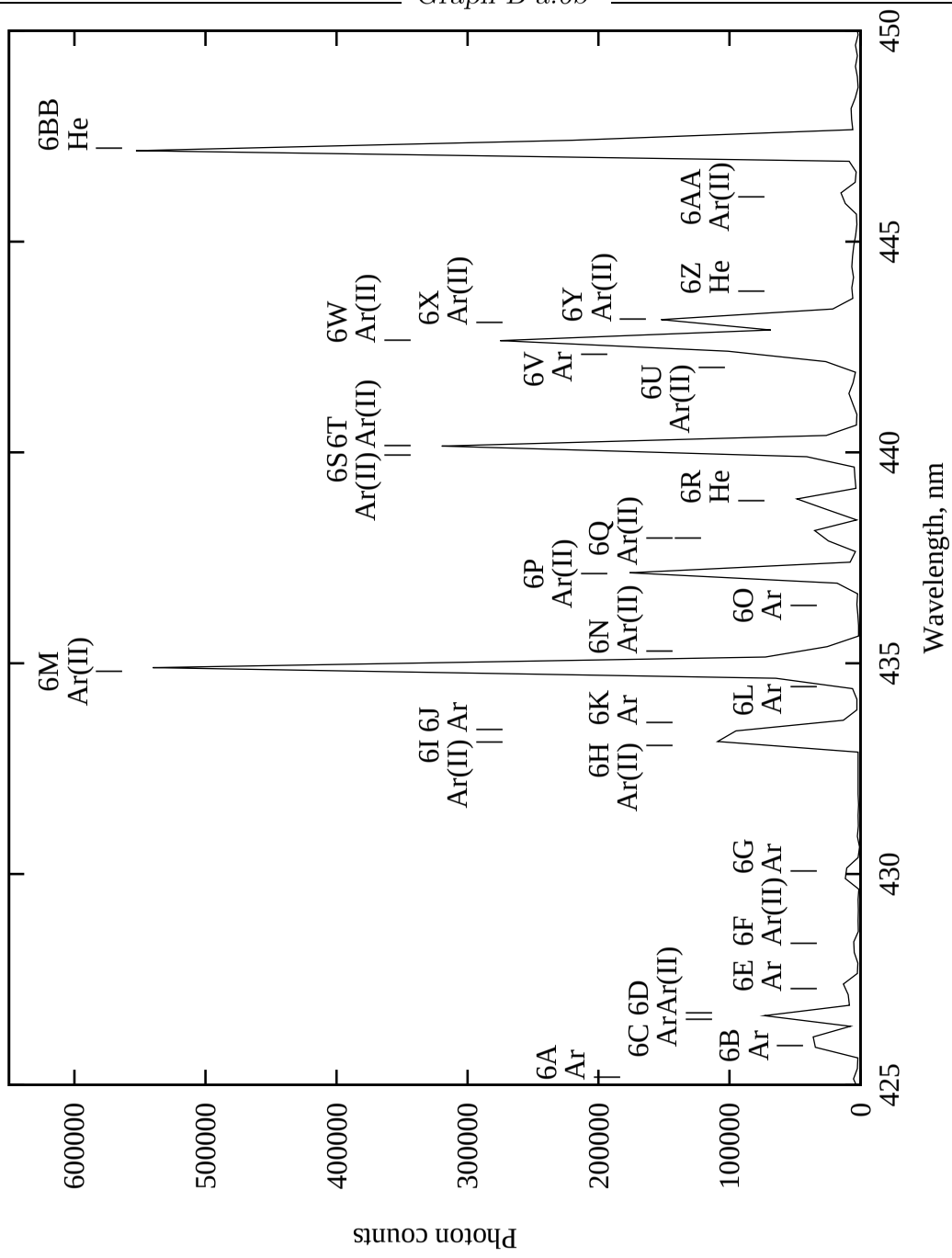
Graph B-a.5c: Direct comparison of spectra taken before (solid line) and after (dashed line) the addition of argon. The emissions are labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.6a



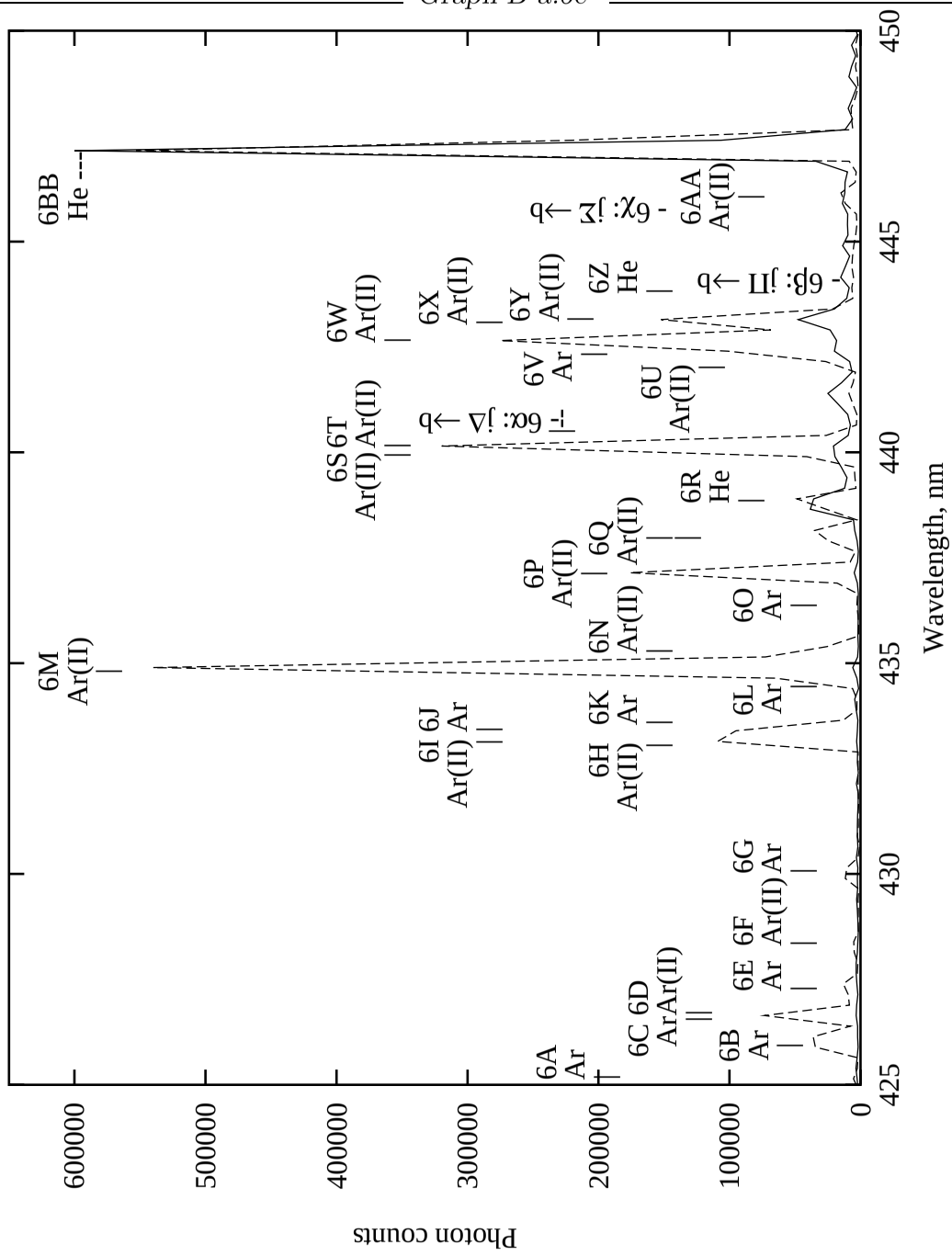
Graph B-a.6a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.6b



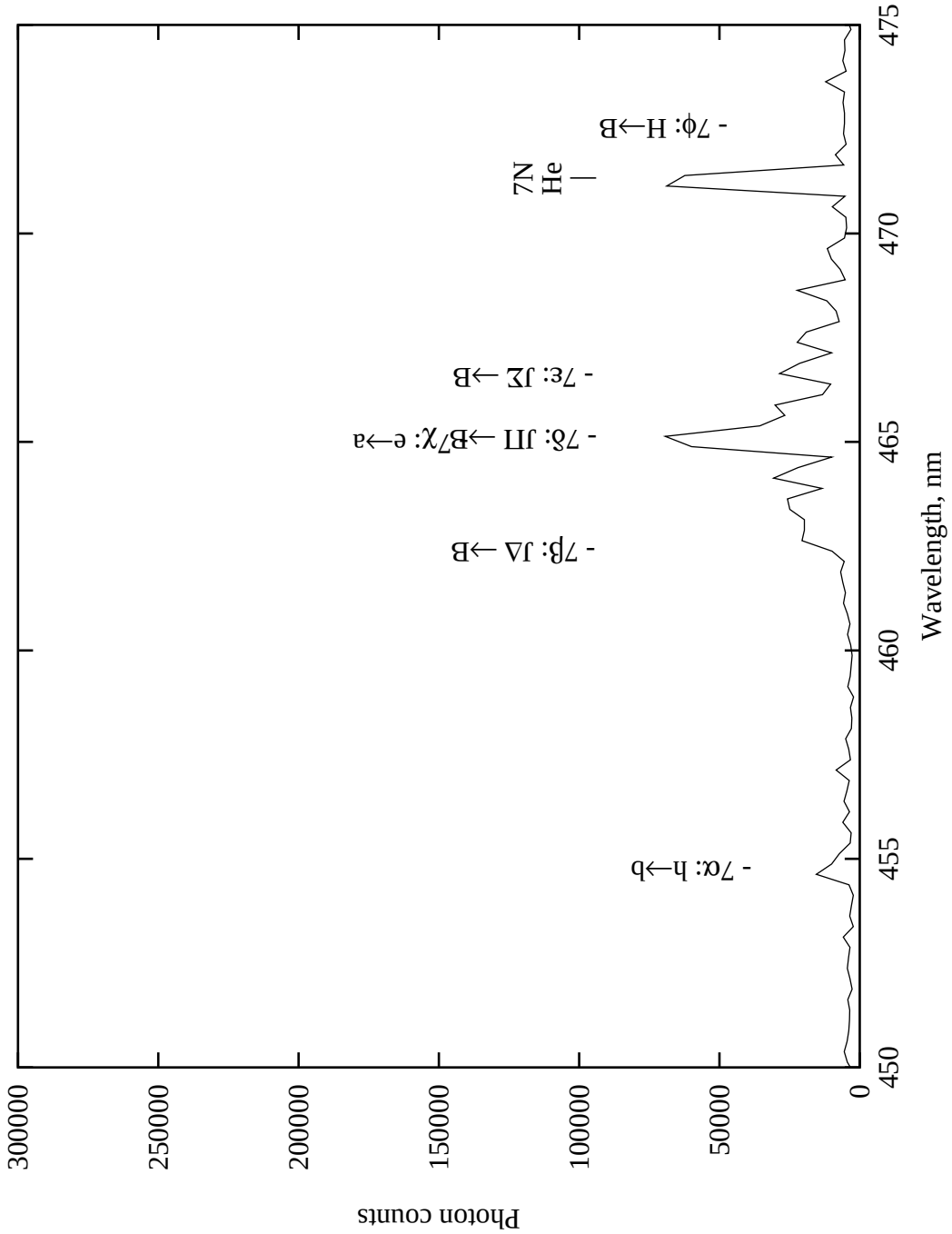
Graph B-a.6b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.6c



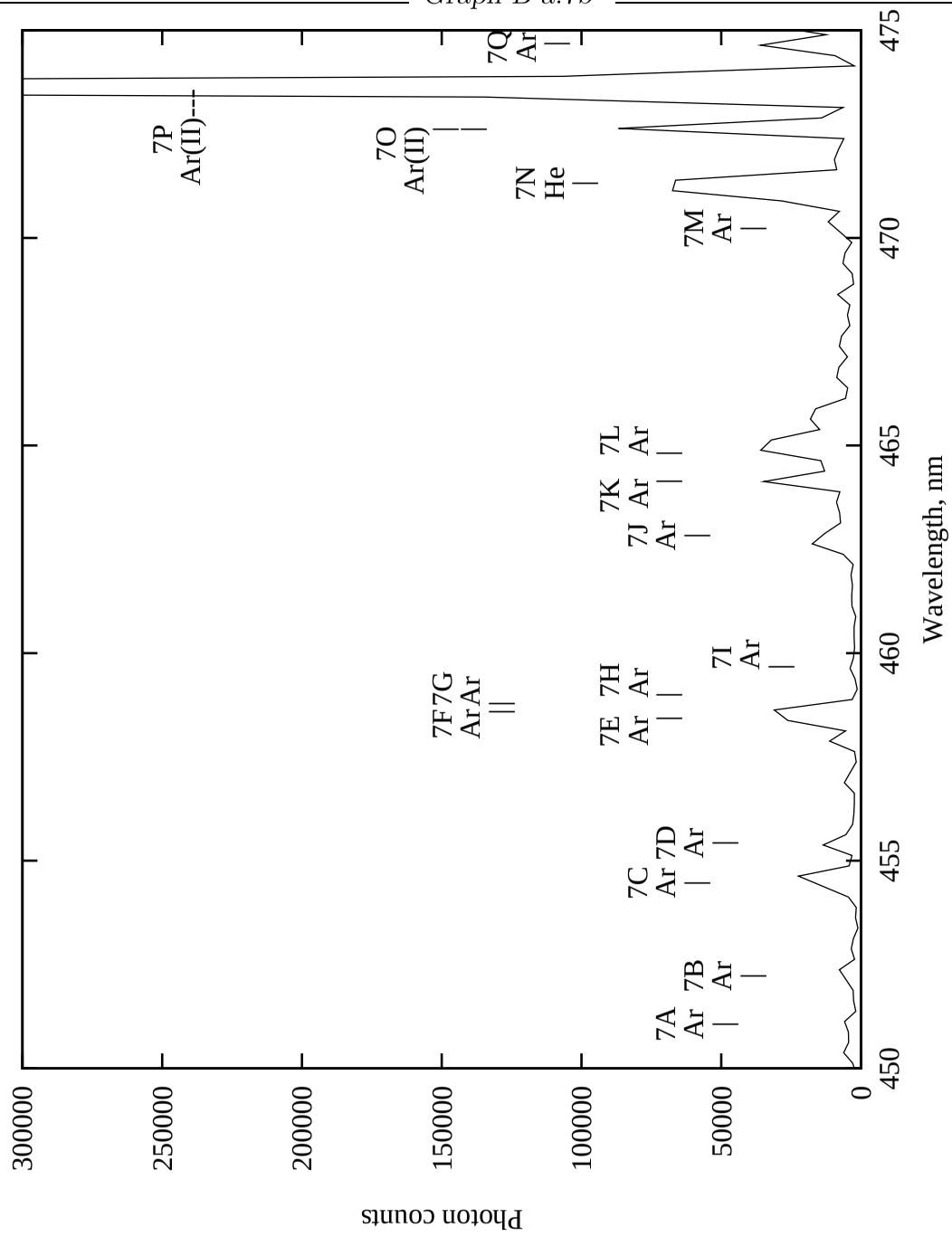
Graph B-a.6c: Direct comparison of spectra taken before (solid line) and after (dashed line) the addition of argon. The emissions are labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.7a



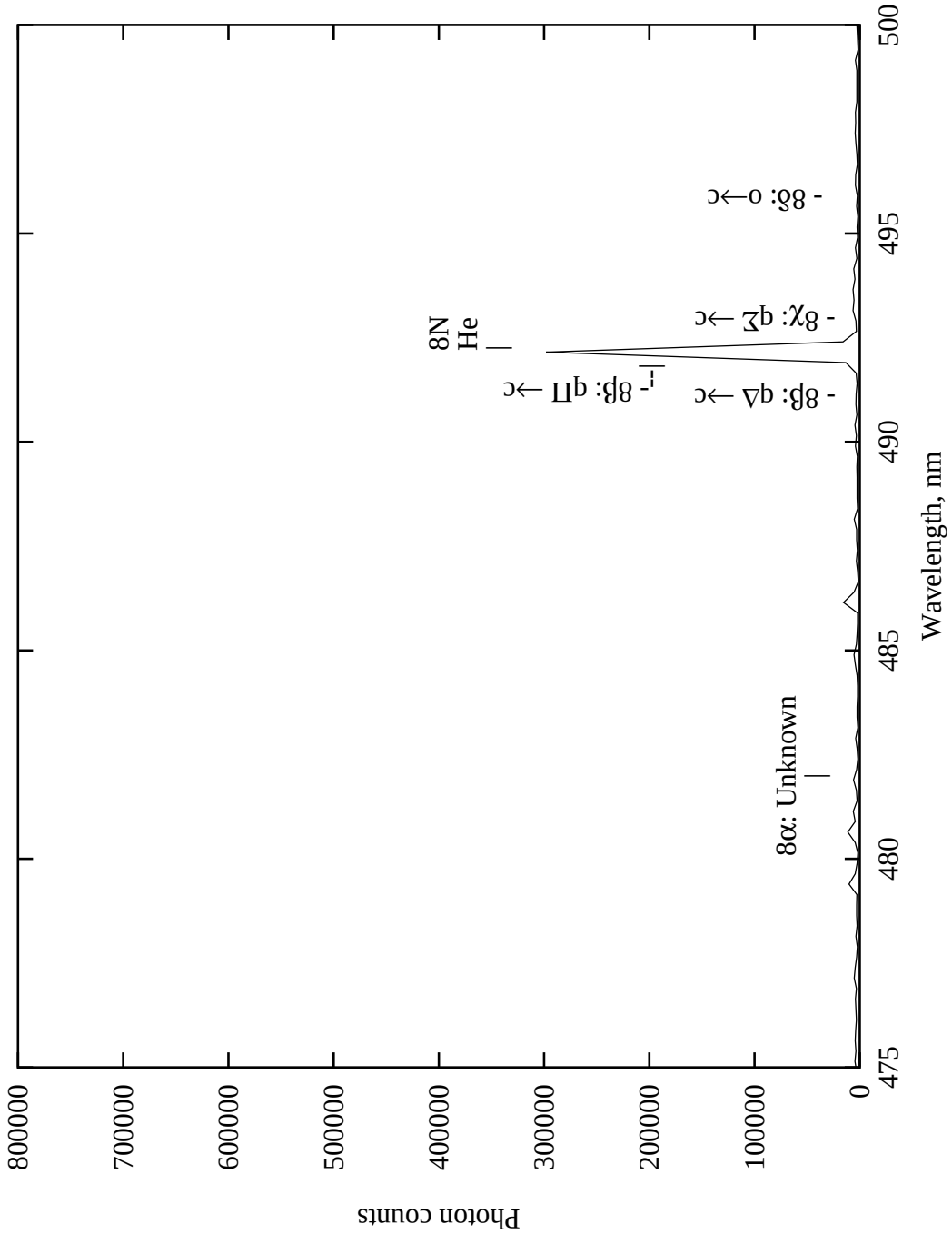
Graph B-a.7a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.7b



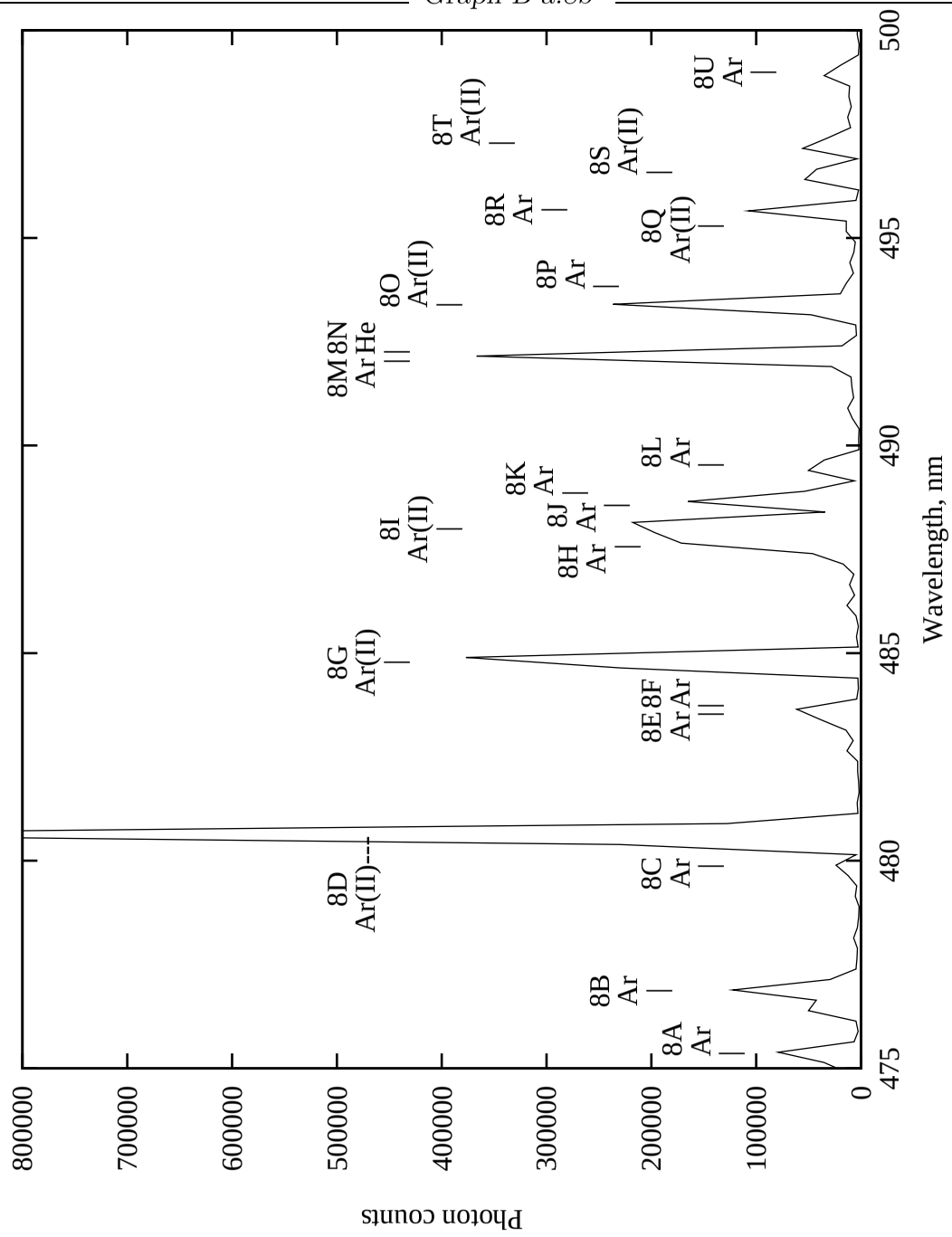
Graph B-a.7b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.8a



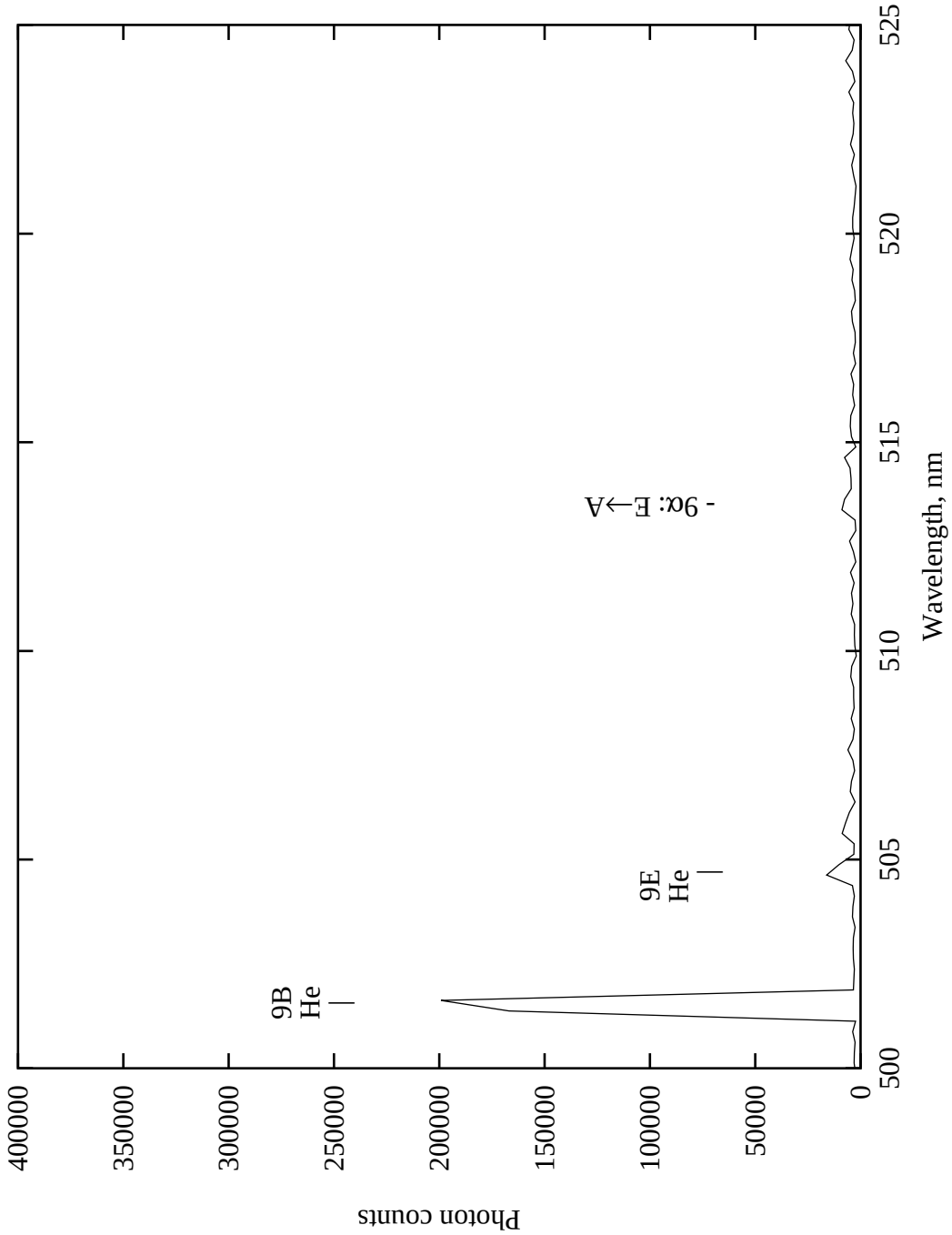
Graph B-a.8a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.8b



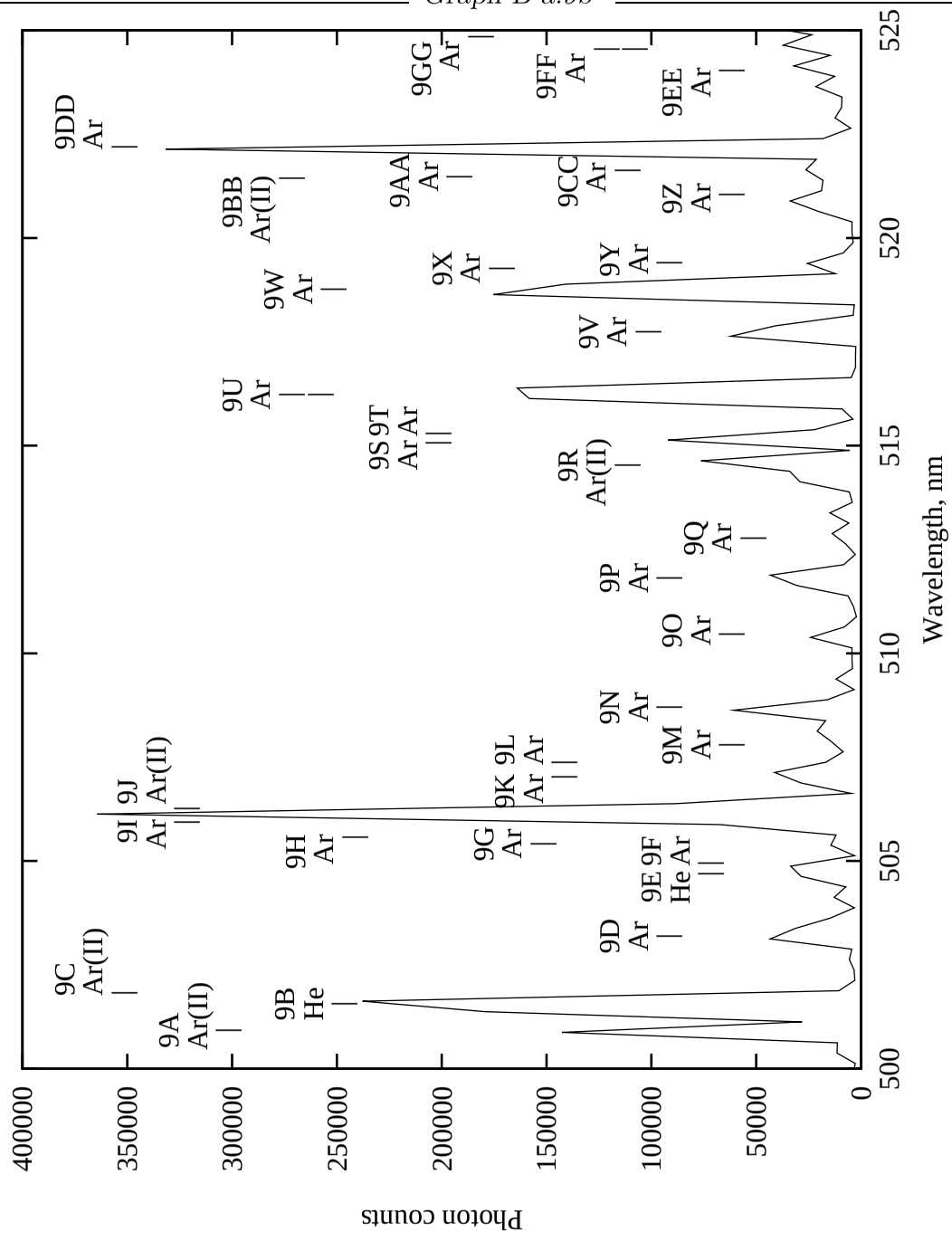
Graph B-a.8b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.9a



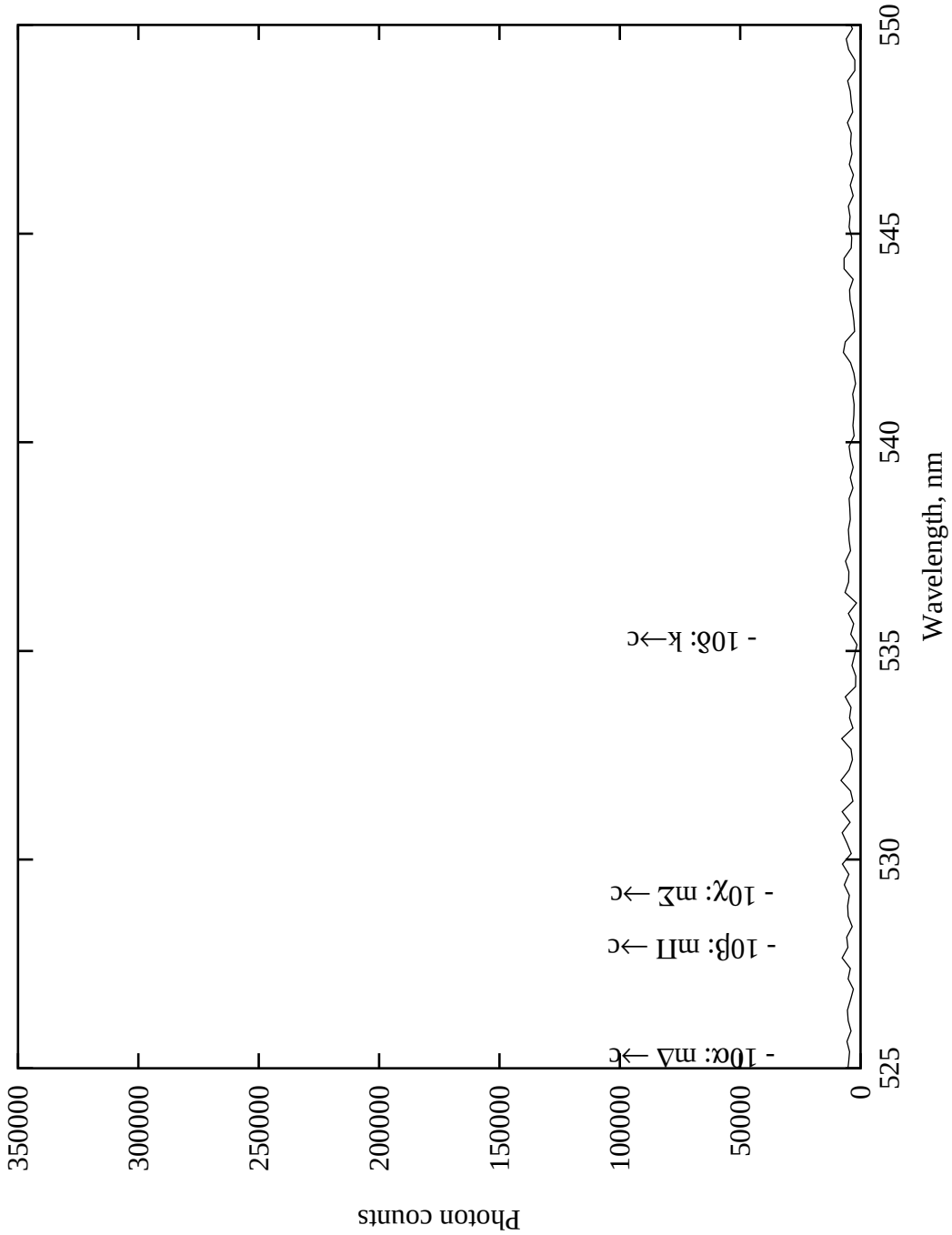
Graph B-a.9a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.9b



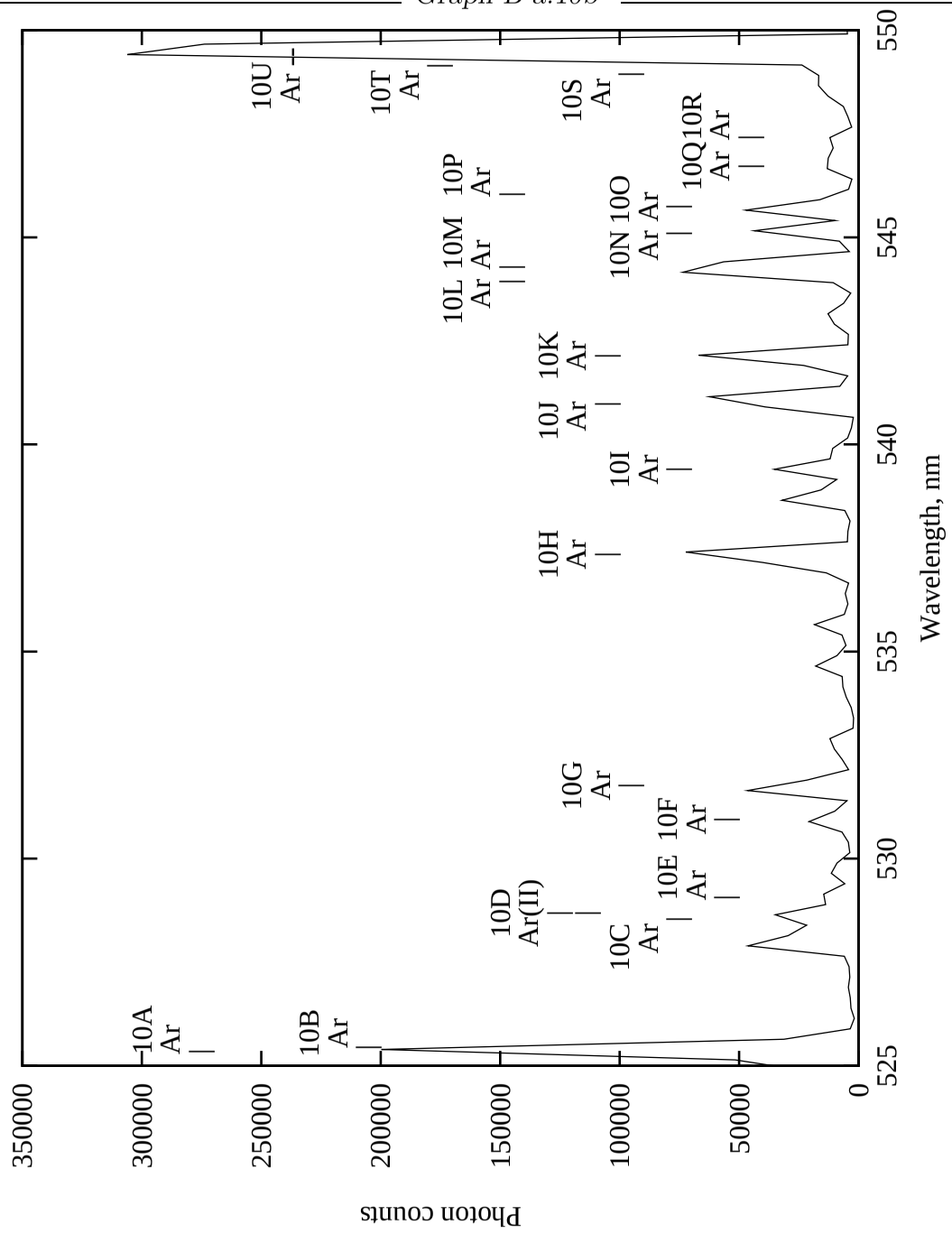
Graph B-a.9b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.10a



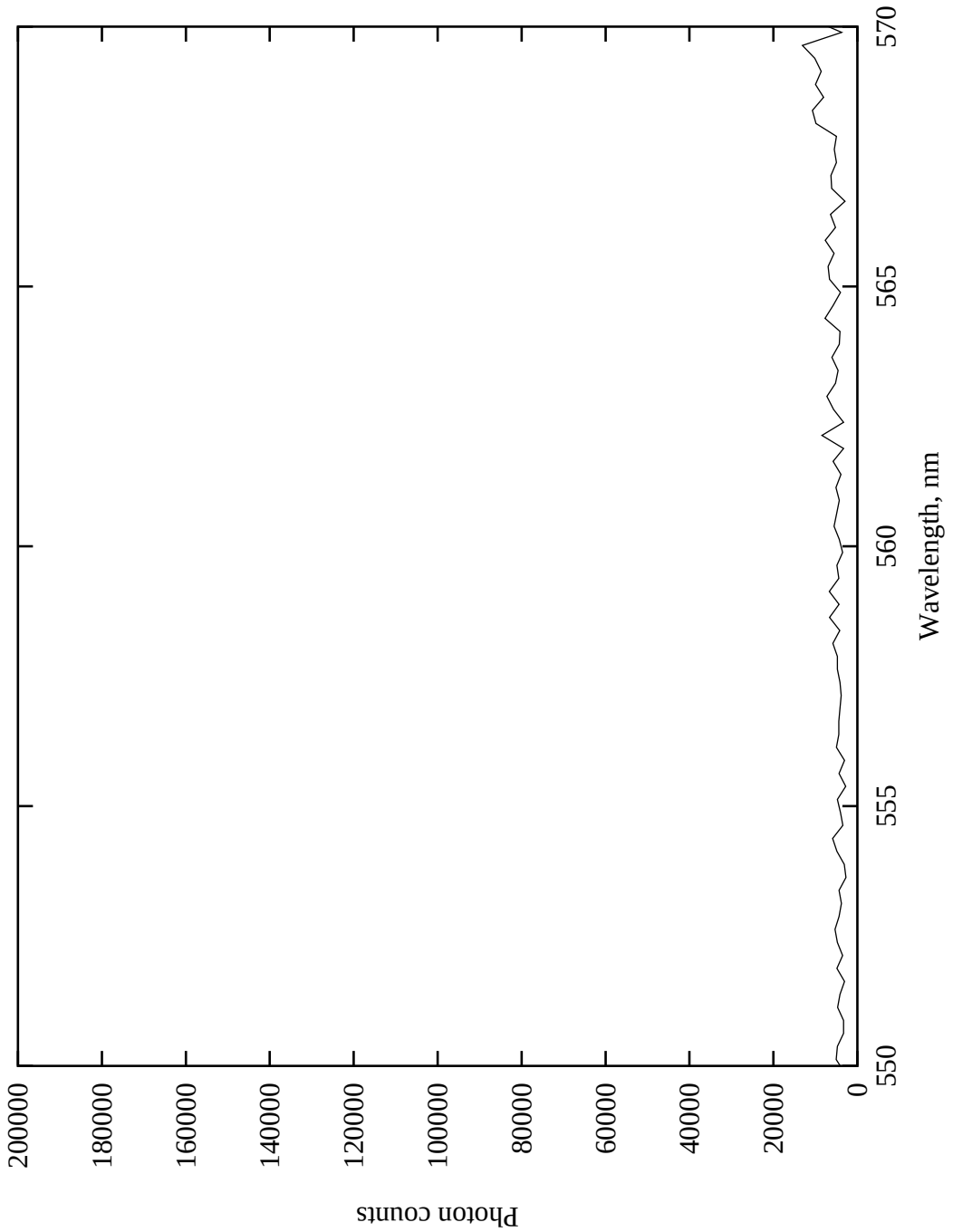
Graph B-a.10a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.10b



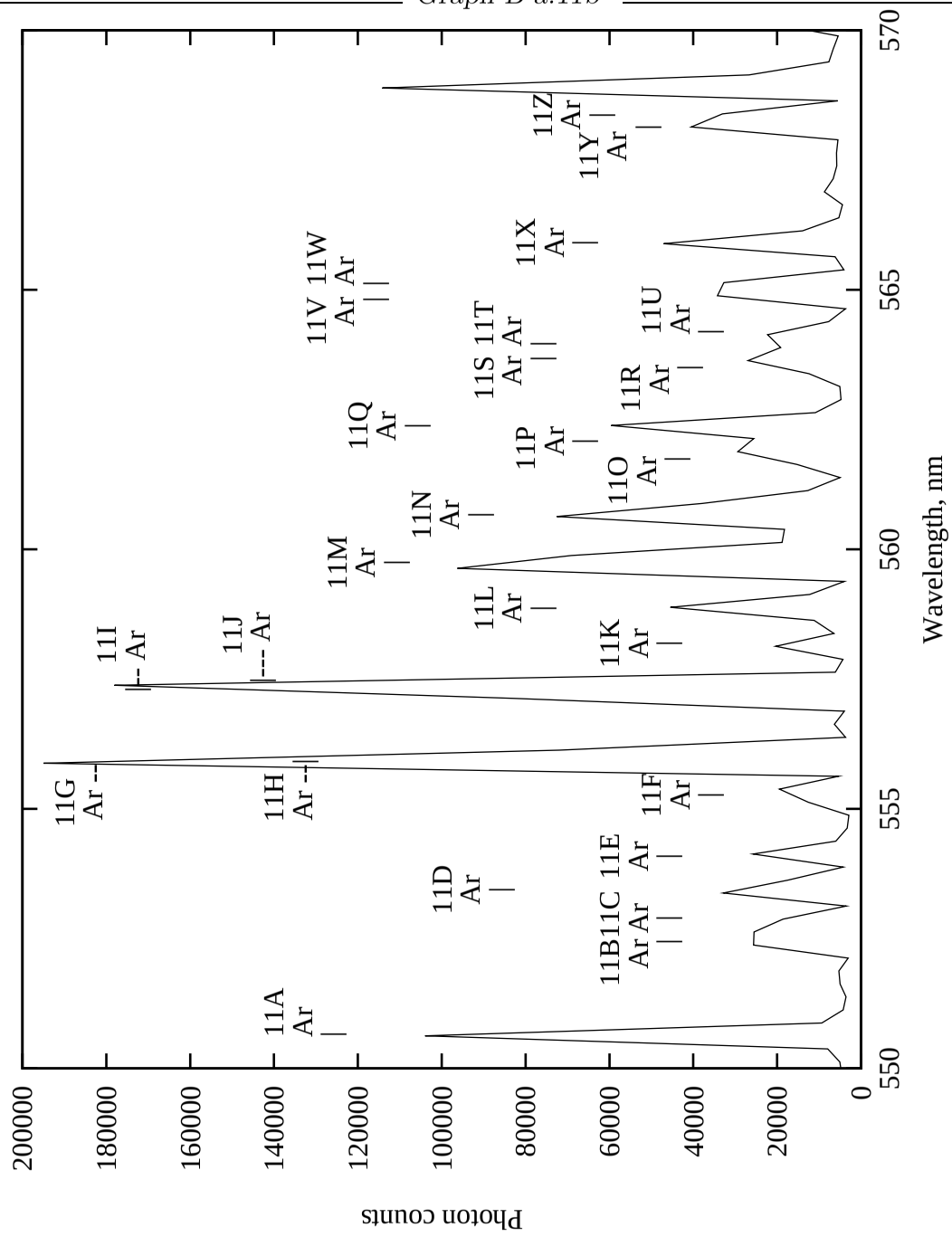
Graph B-a.10b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.11a



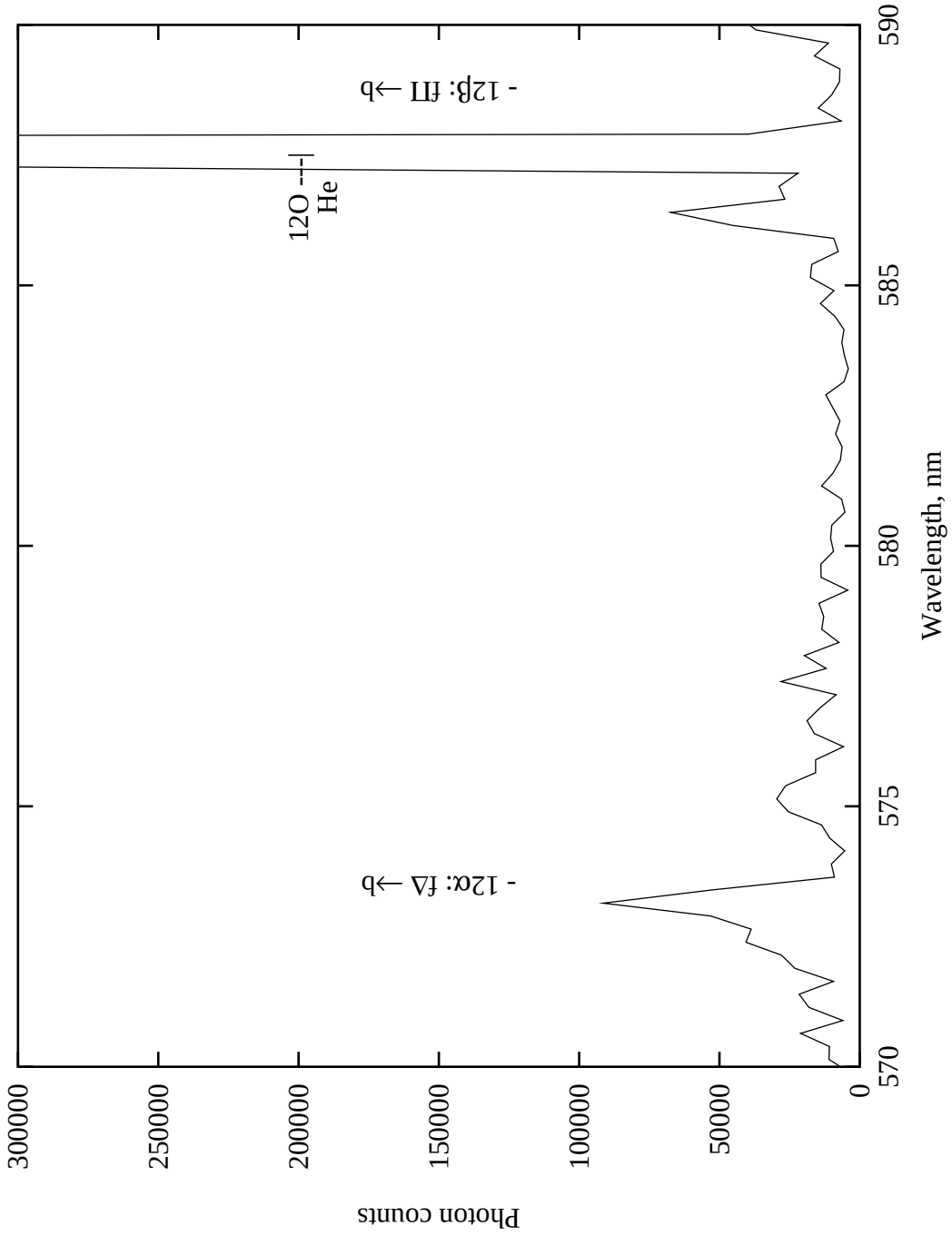
Graph B-a.11a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.11b



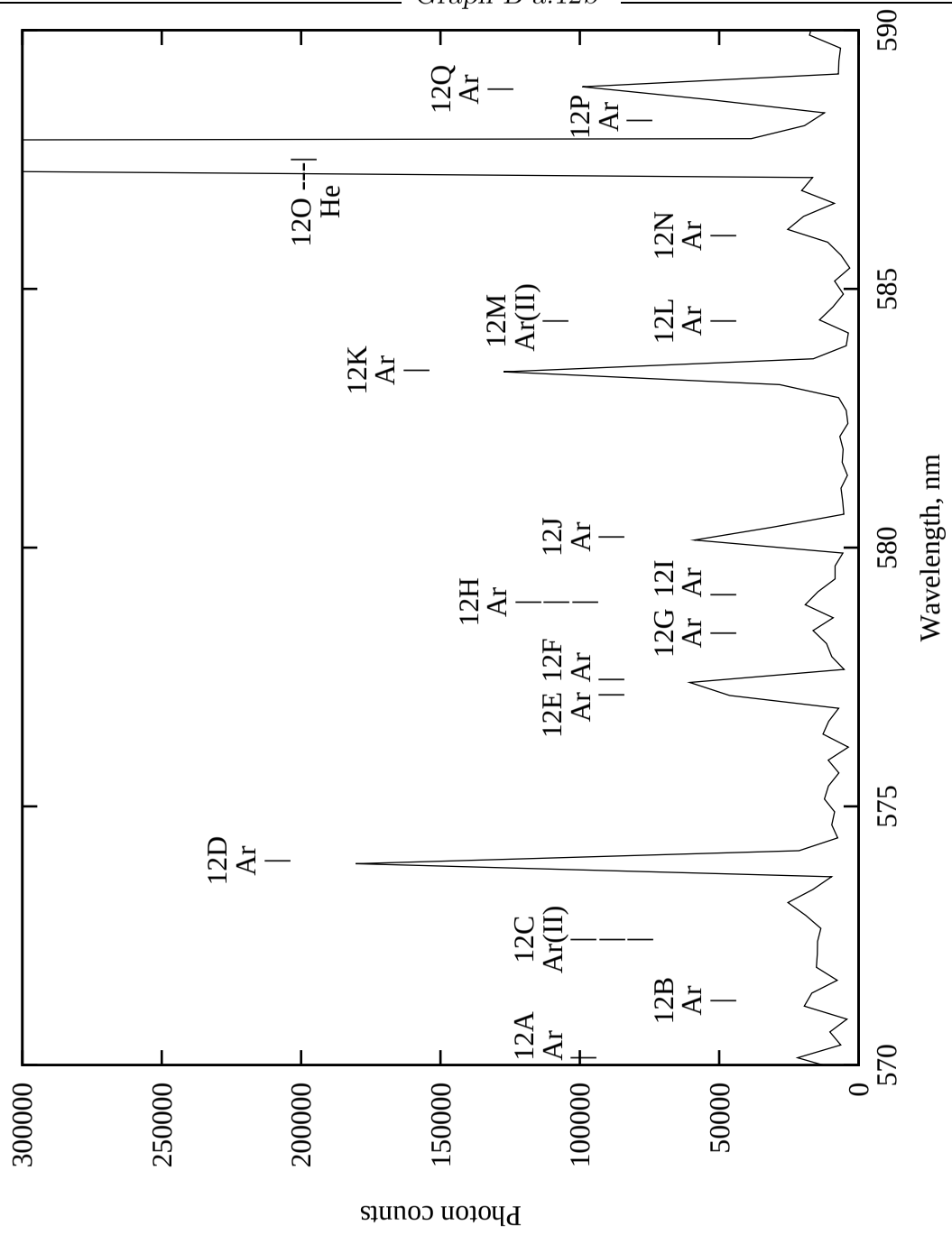
Graph B-a.11b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.12a



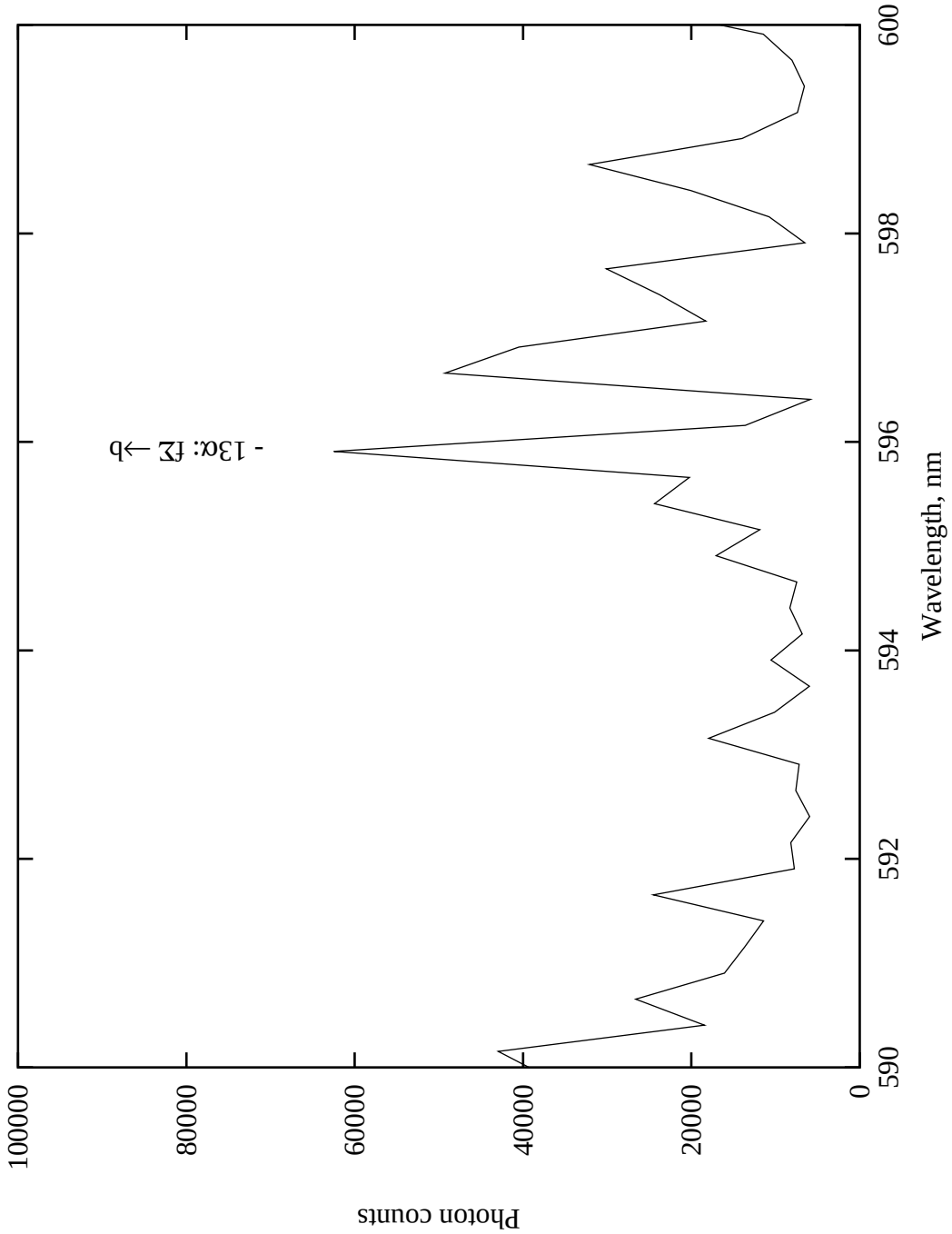
Graph B-a.12a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.12b



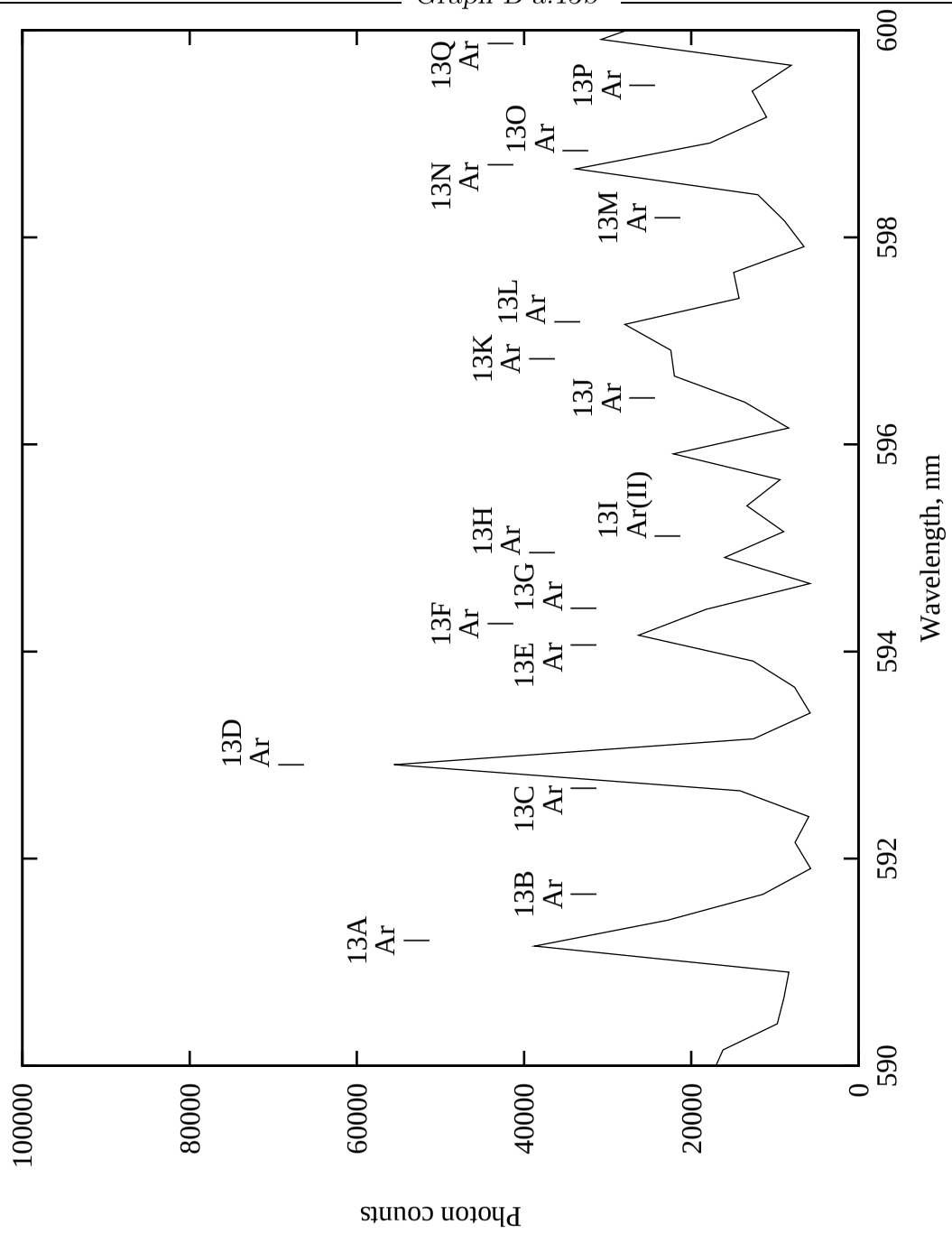
Graph B-a.12b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.13a



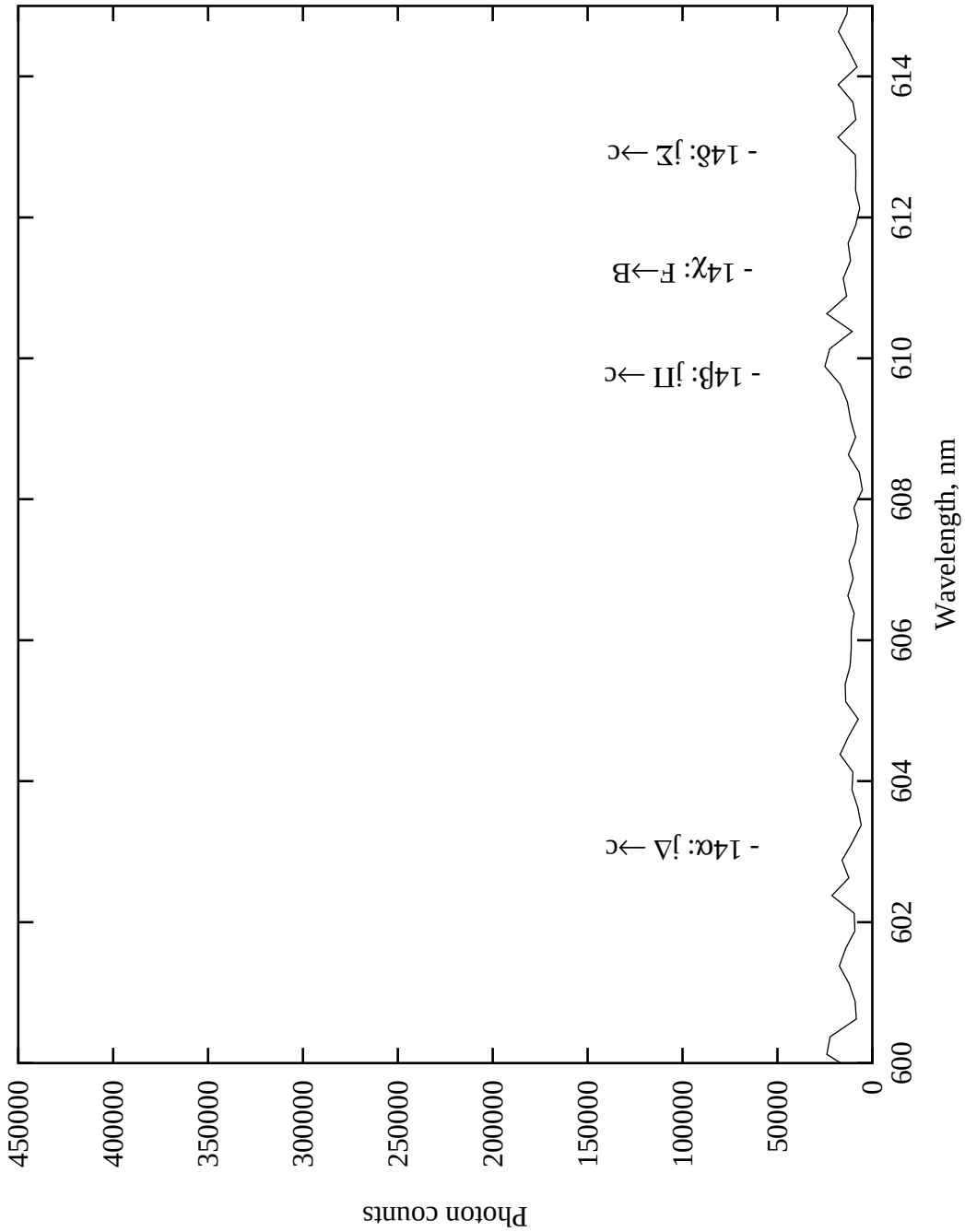
Graph B-a.13a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.13b



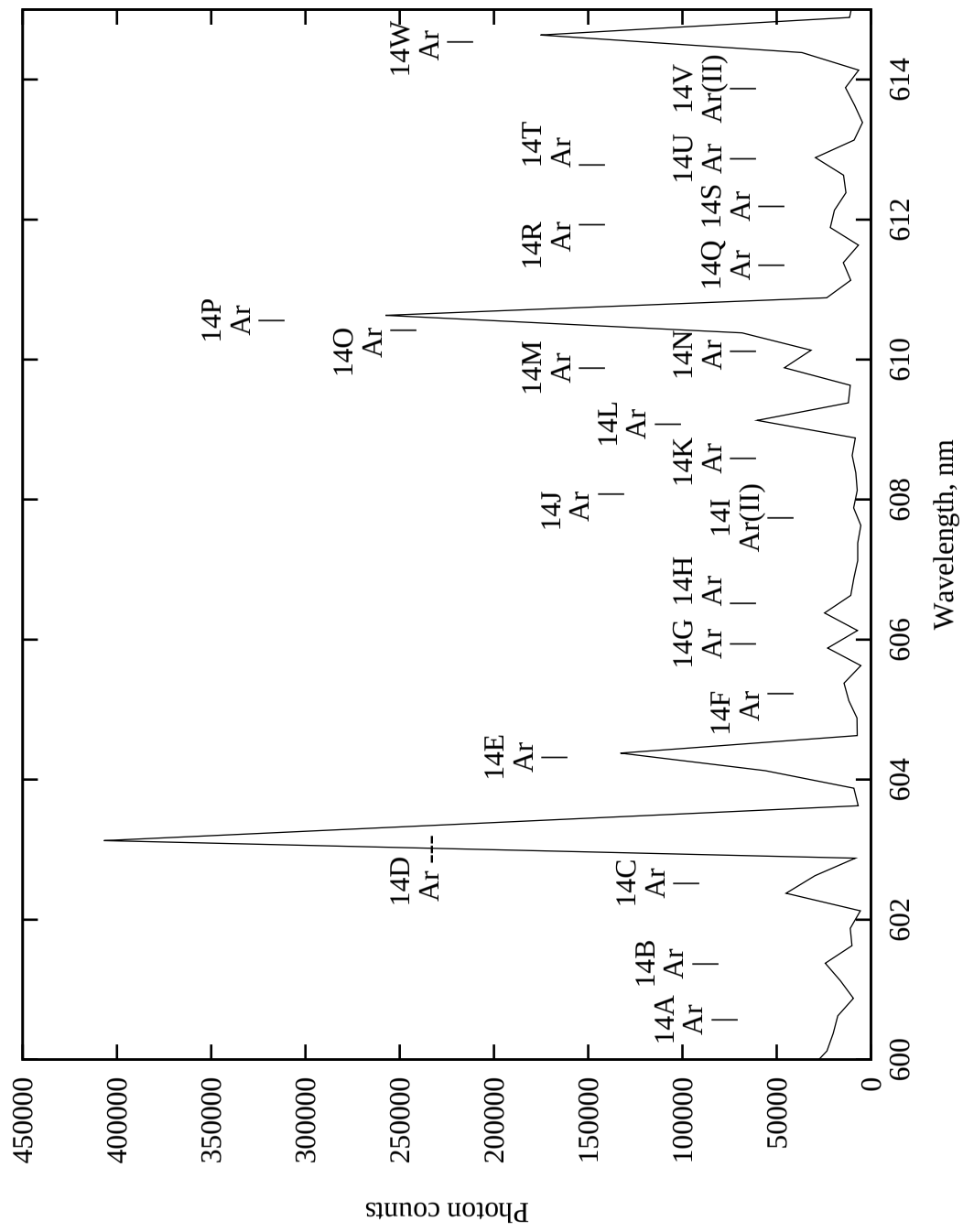
Graph B-a.13b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.14a



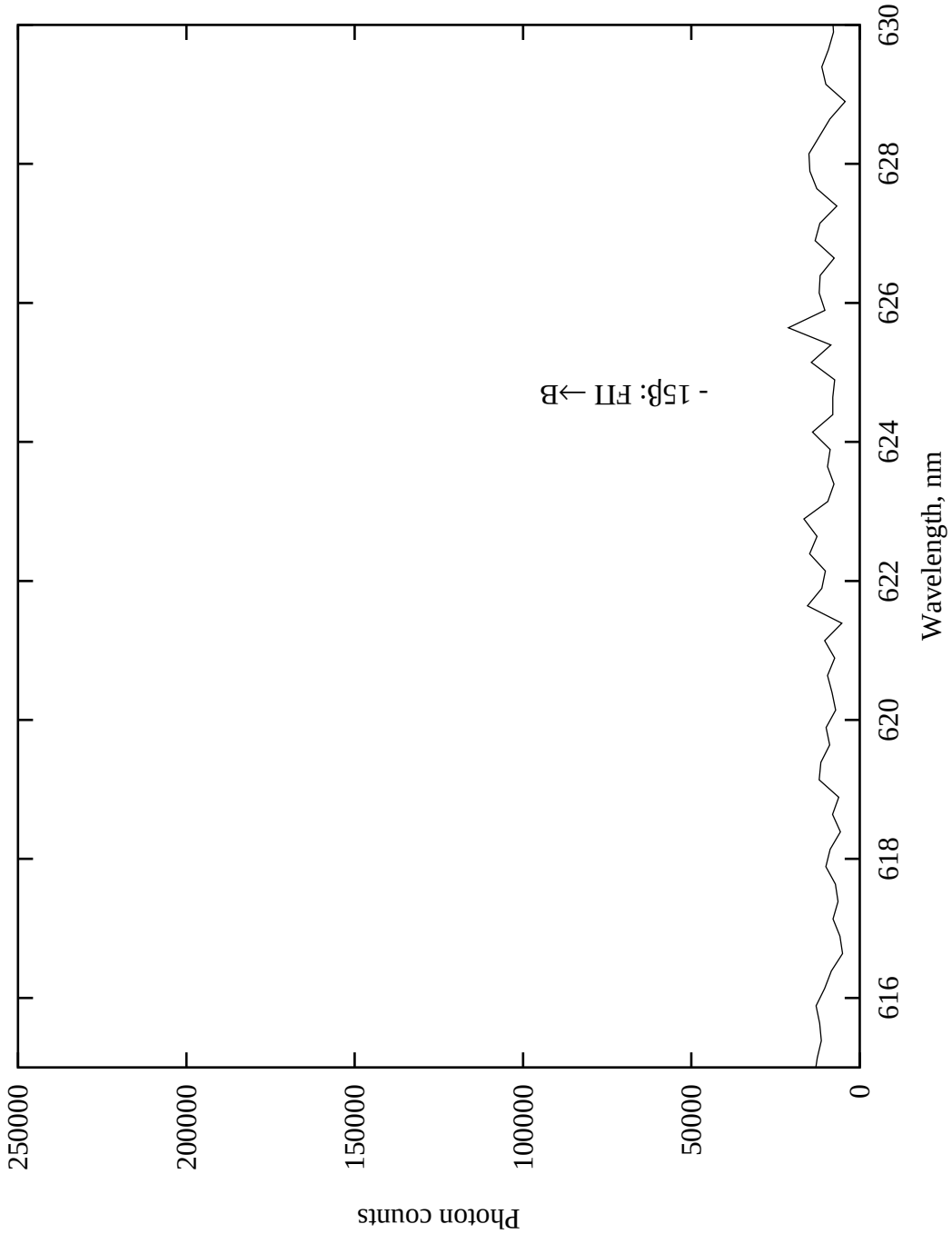
Graph B-a.14a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.14b



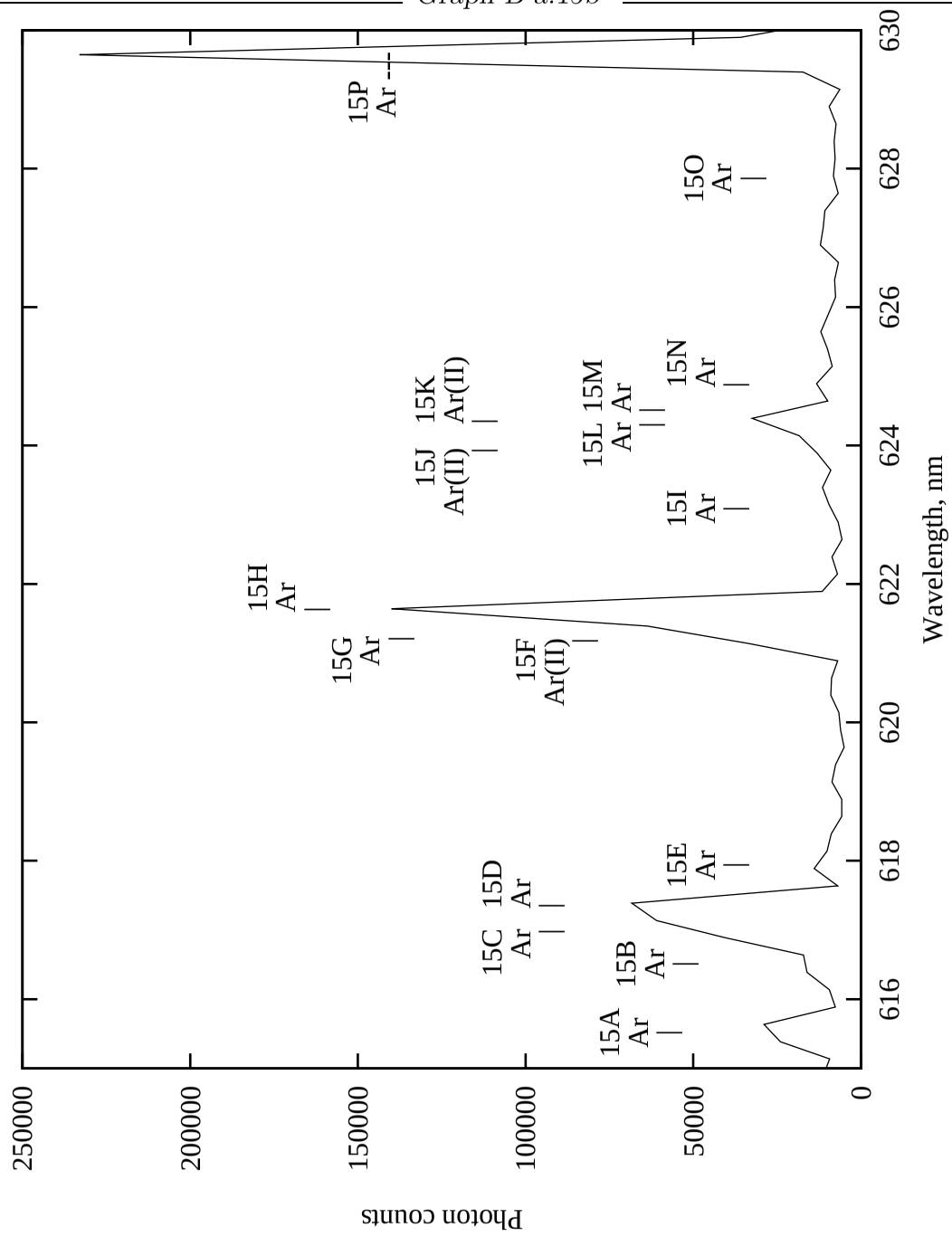
Graph B-a.14b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.15a



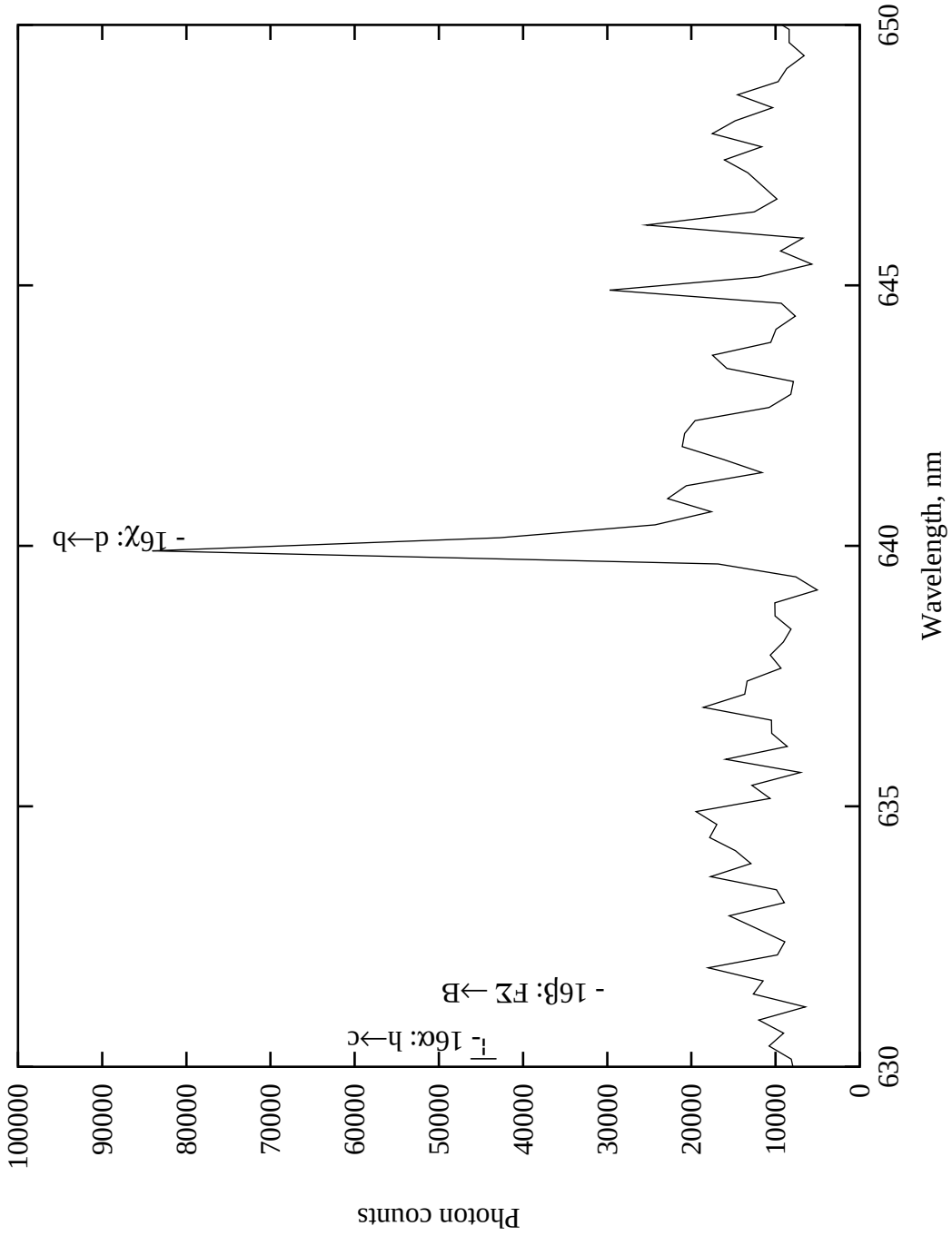
Graph B-a.15a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.15b



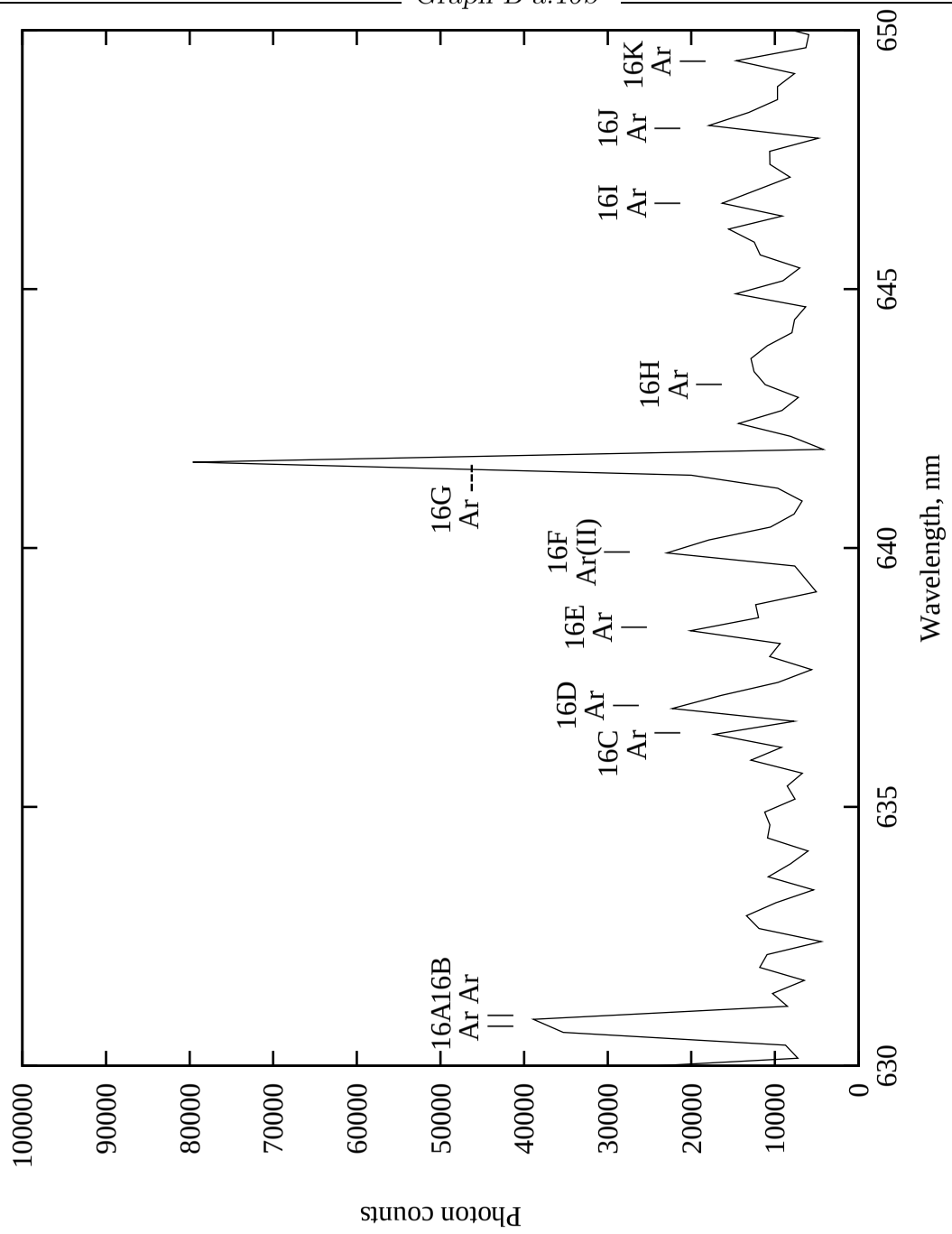
Graph B-a.15b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.16a



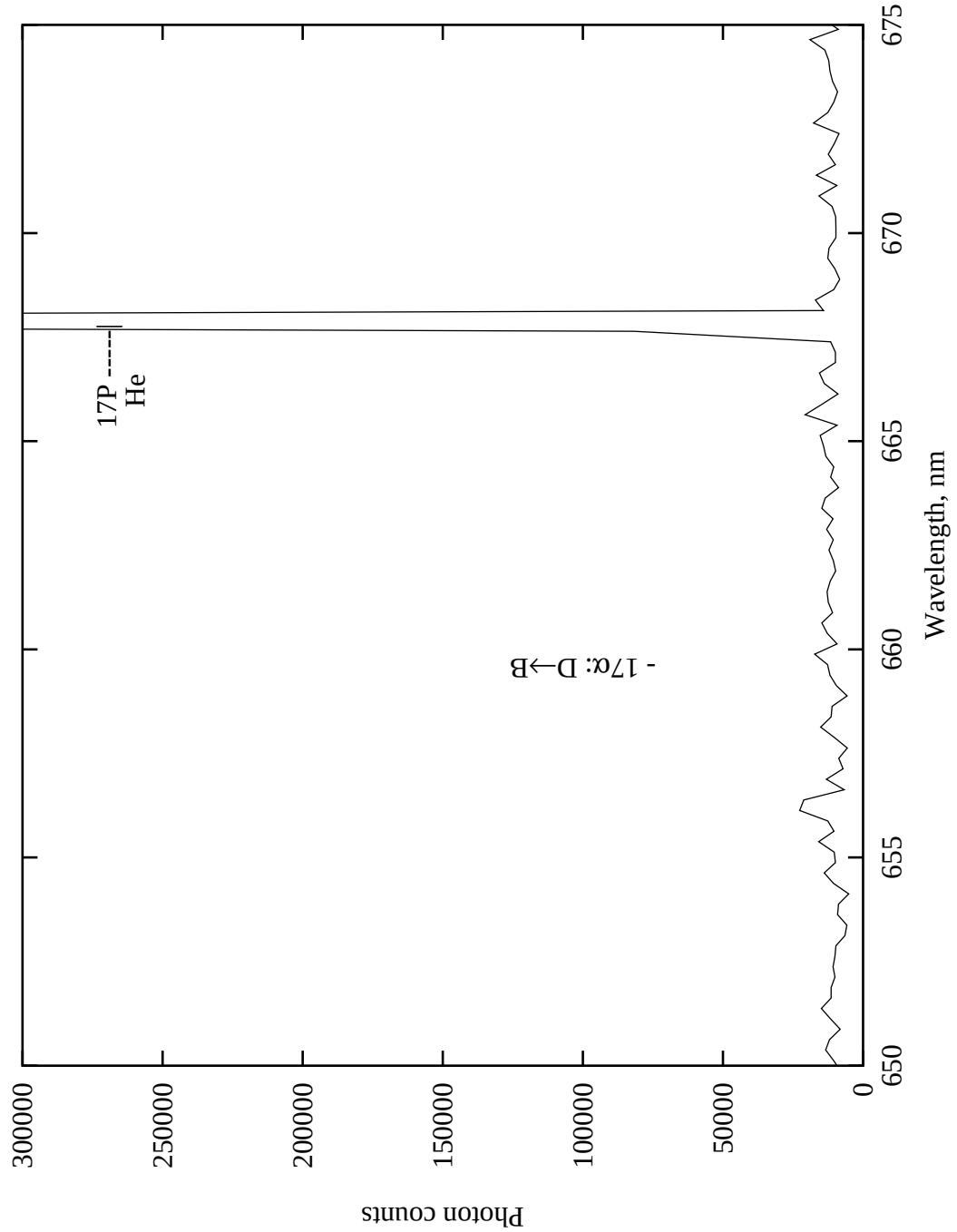
Graph B-a.16a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.16b



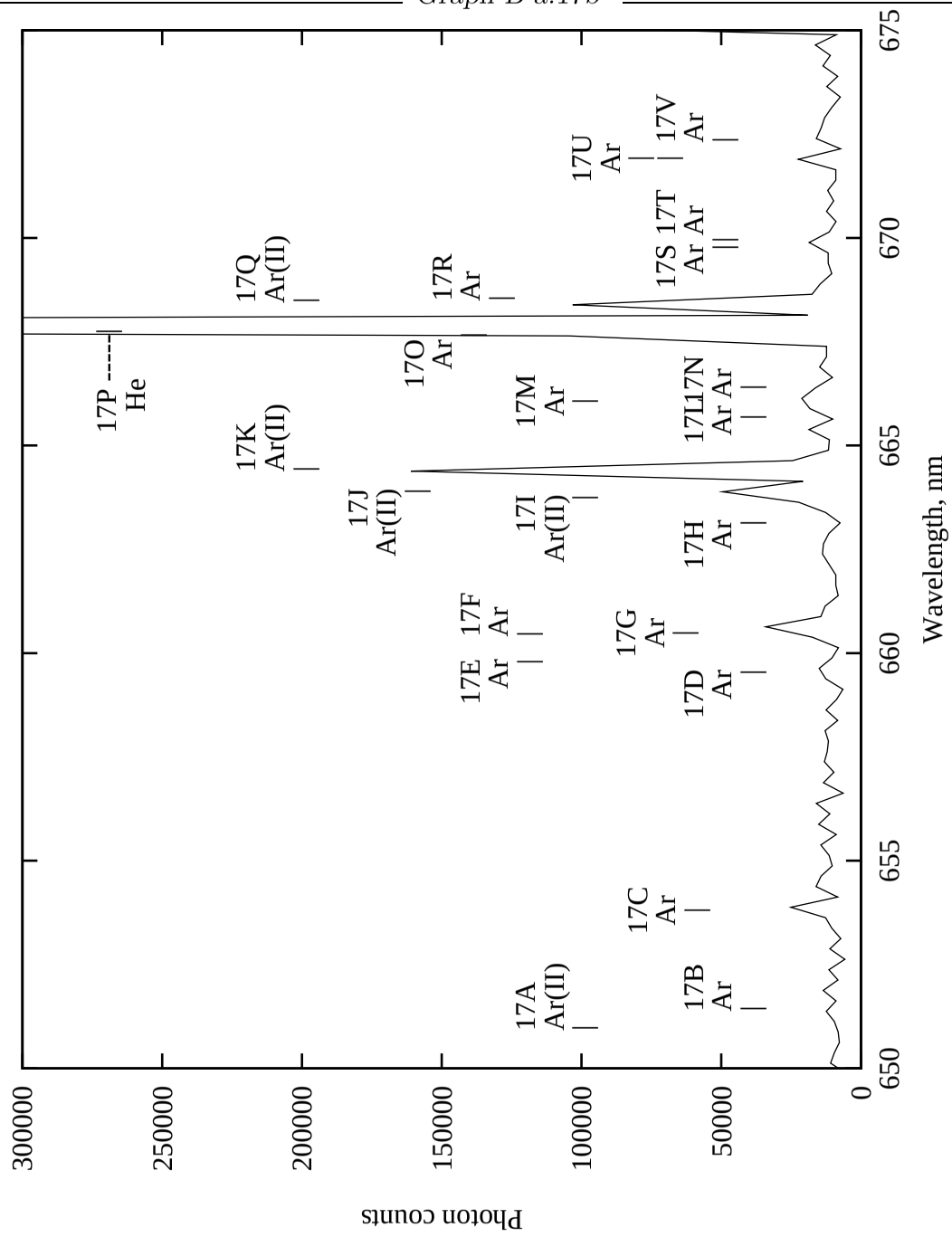
Graph B-a.16b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.17a



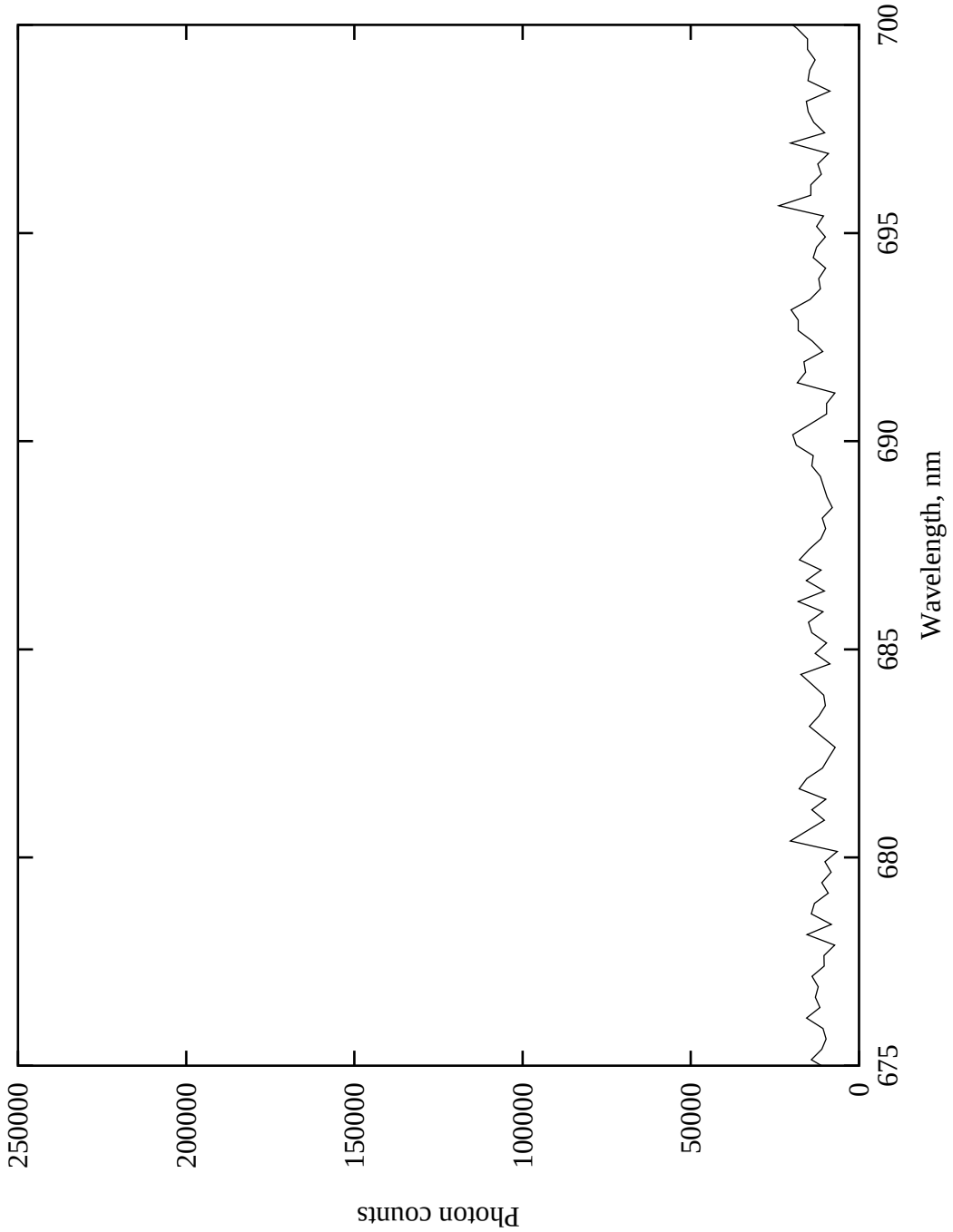
Graph B-a.17a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.17b



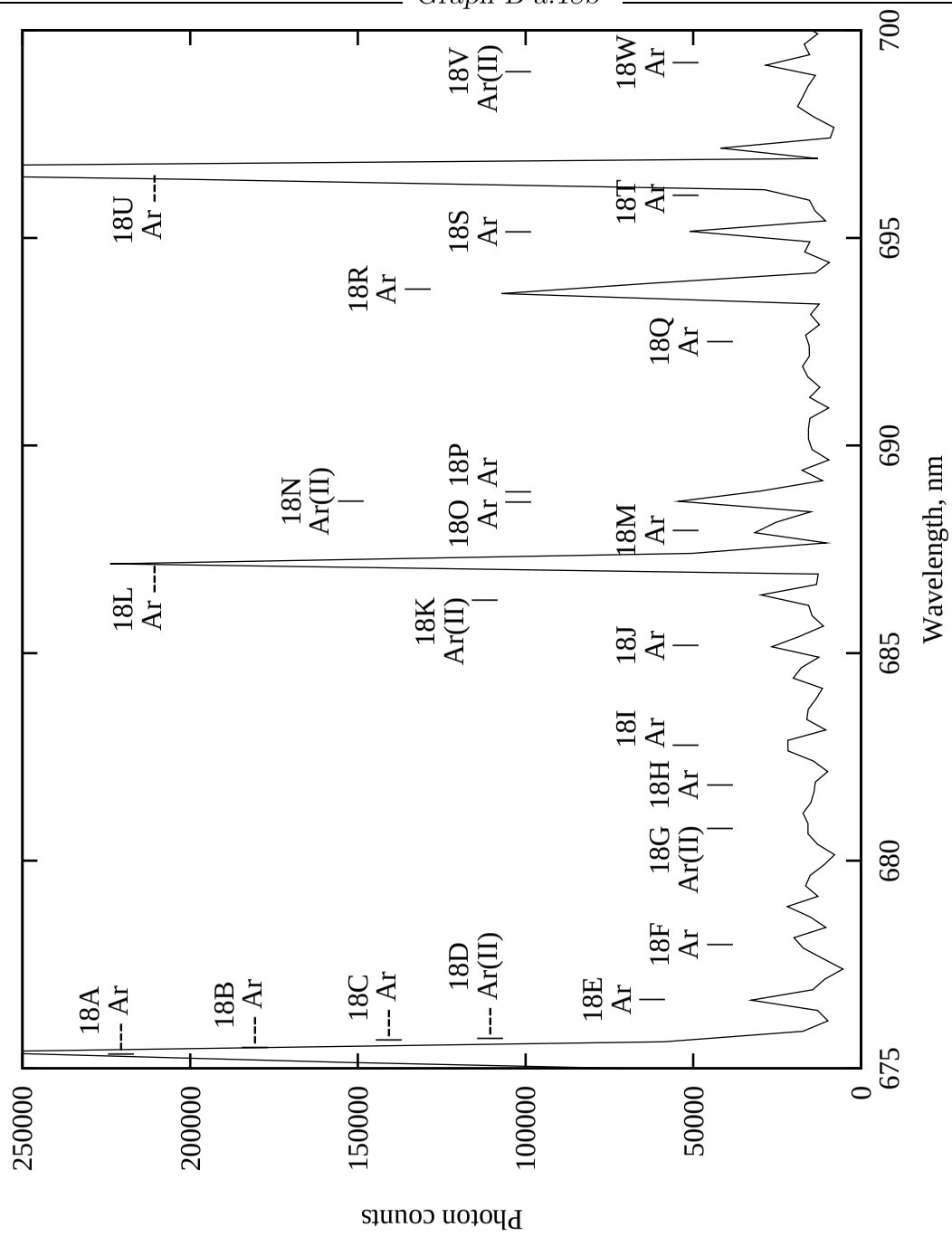
Graph B-a.17b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.18a



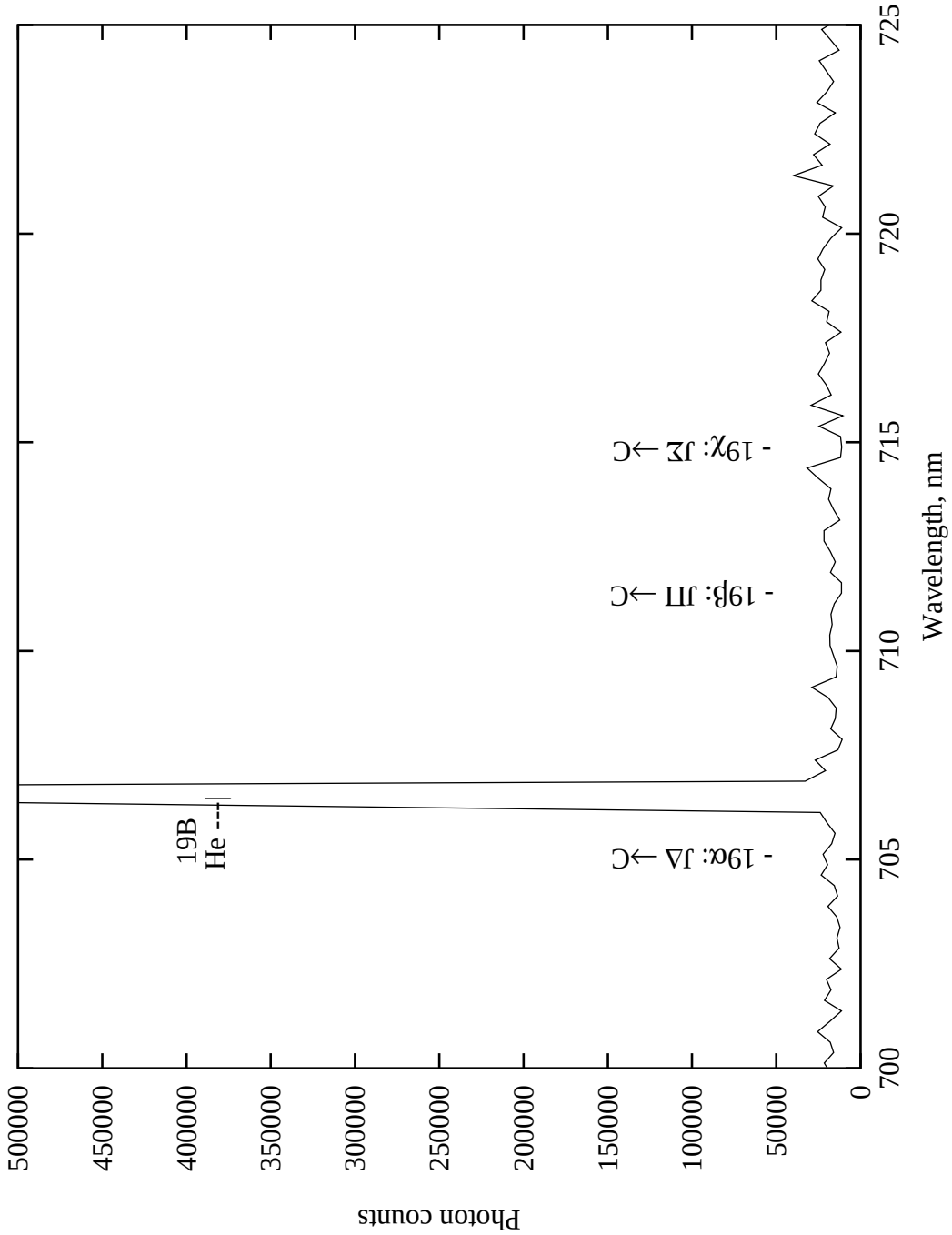
Graph B-a.18a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.18b



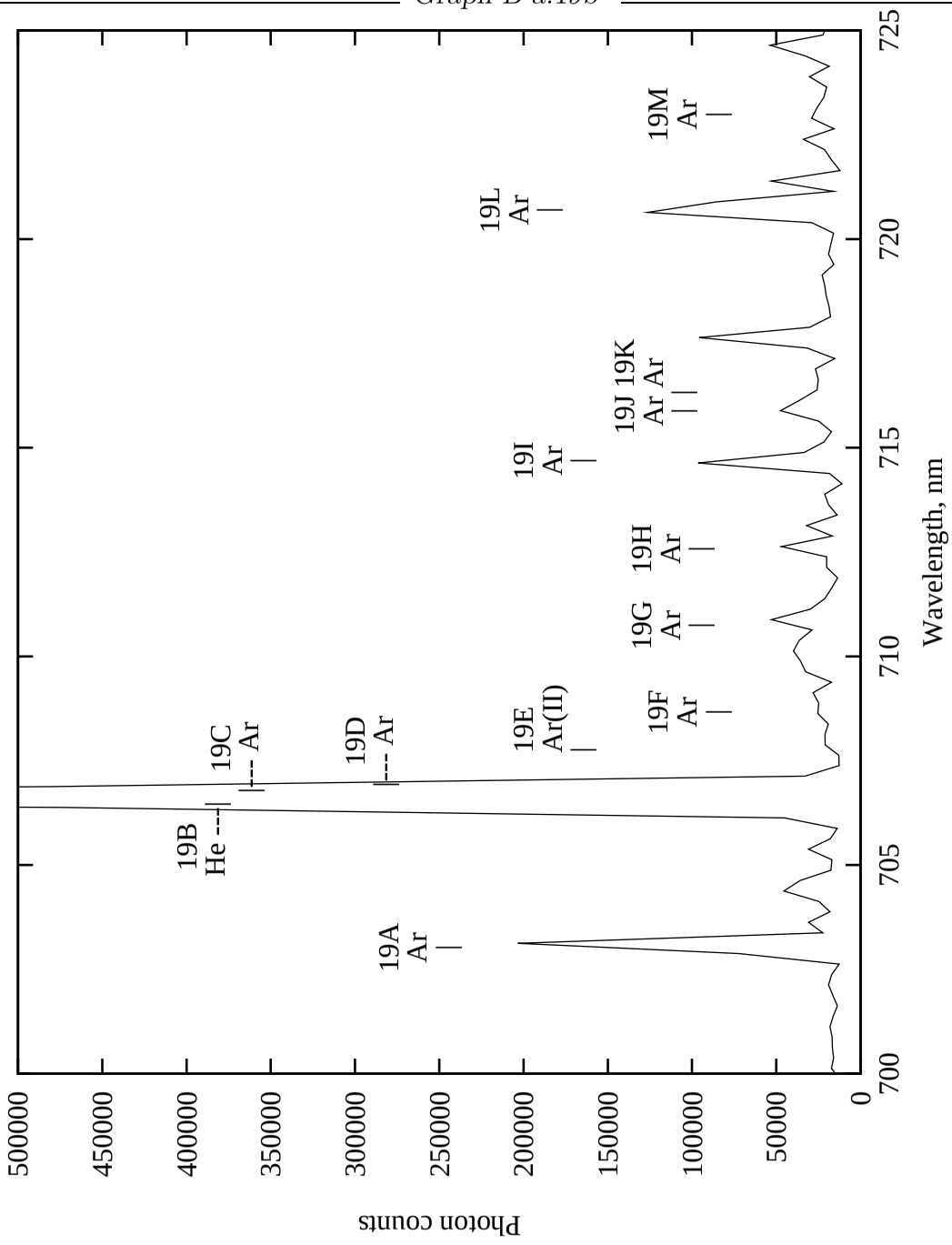
Graph B-a.18b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.19a



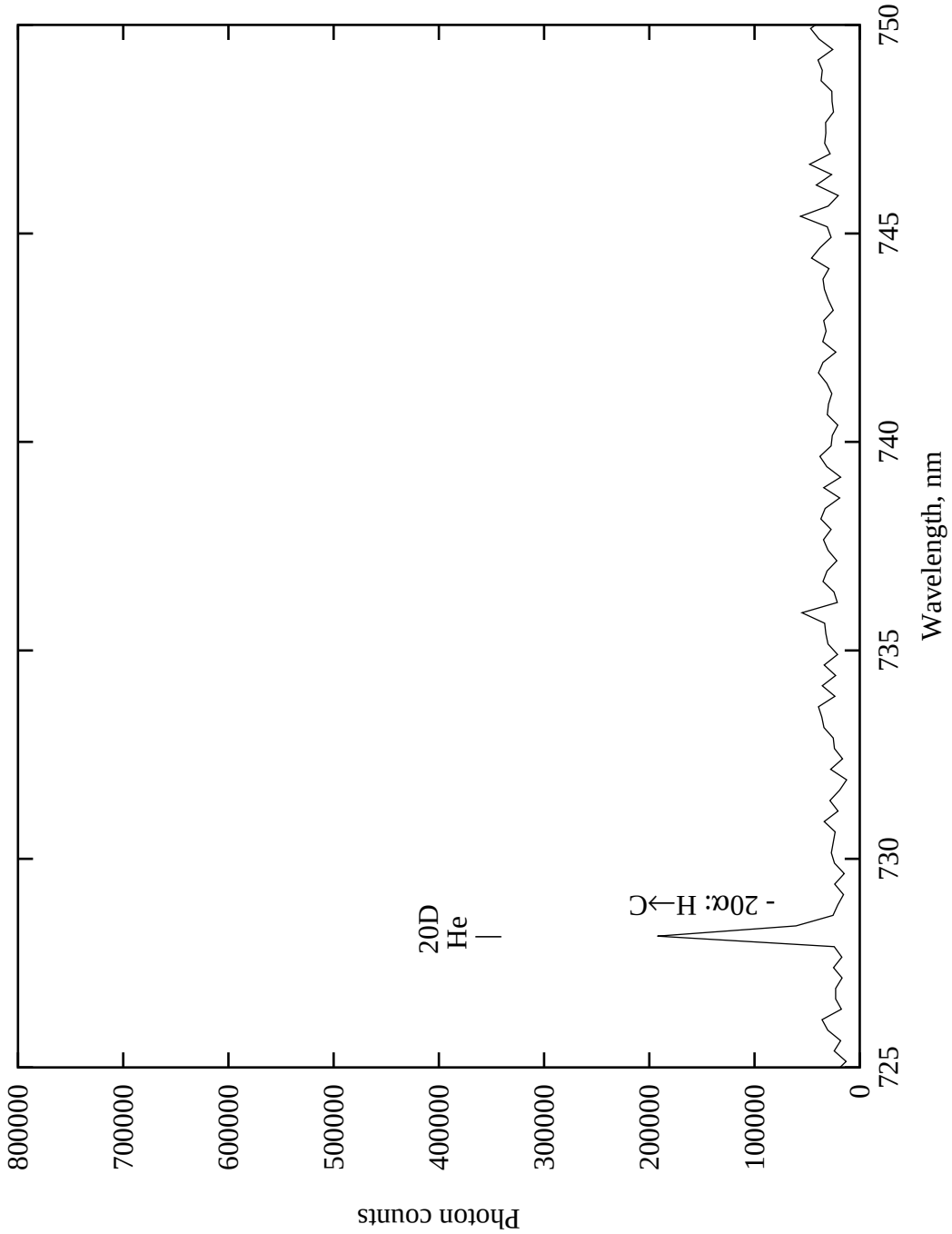
Graph B-a.19a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.19b



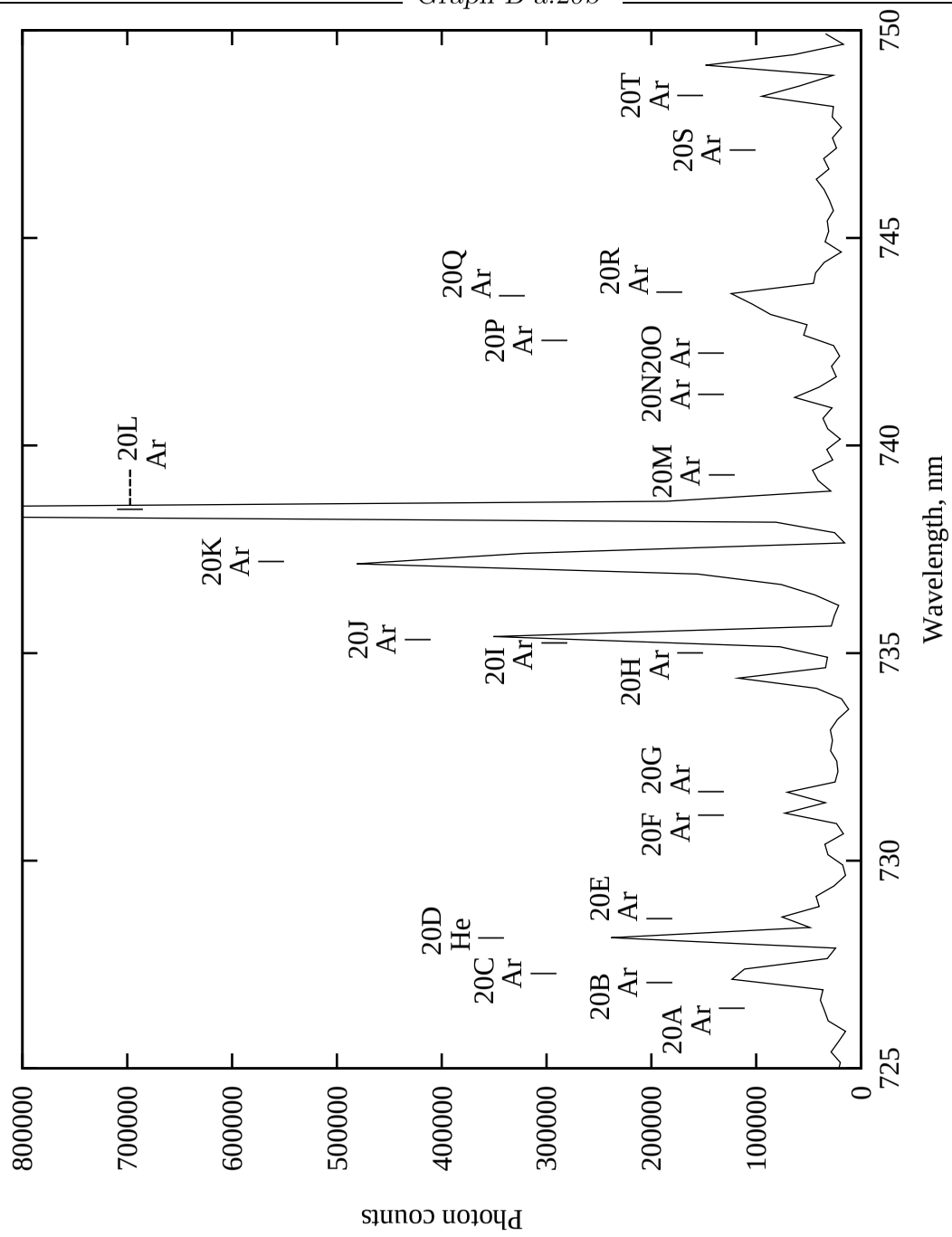
Graph B-a.19b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.20a



Graph B-a.20a: Emissions from the helium-only plasma with atomic and molecular species labeled at approximate positions. See Appendix B-b for assignments. Some low intensity peaks may be obscured by background or molecular emissions.

Graph B-a.20b



Graph B-a.20b: Emissions from the helium-argon plasma with atomic species labeled at approximate positions. See Appendix B-b for assignments. Molecular species are identified on graphs in this series labeled a and c. Some low intensity peaks may be obscured by background or molecular emissions.

B-b Key

This appendix contains the key to the alpha–numeric labels used on the graphs in Appendix B-a. See the beginning of that appendix for further information.

Key to Graphs iii & iv

Graphs iii (a) & iii (b): 100-160 nm

Label	Position (nm)	Energy Low (eV)	Term Low	Energy High (eV)	Term High
A (H)	121.6	0	$1s\ ^2S_{1/2}$	10.199	$2p\ ^2P^0_{1/2,3/2}$
B (O)	130.5	0	$2p^4\ ^3P_{0,1,2}$	9.52	$2p^3 3s\ ^3S^0_1$
C (–)	A collection of small unidentified peaks.				
D (C)	Several local peaks; the one with the largest transition probability is:				
	136.0	1.26	$2p^2\ ^1D_2$	10.41	$2p4d\ ^1F^0_3$
E (C)	156.1	0	$2s^2 2p^2\ ^3P_{0,1,2}$	7.95	$2s2p^3\ ^3D^0_{1,2,3}$

Graphs iv (a) & iv (b): 160-260 nm

Label	Position (nm)	Energy Low (eV)	Term Low	Energy High (eV)	Term High
A (C)	165.7	0	$2p^2\ ^3P_{0,1,2}$	7.48	$2p3s\ ^3P^0_{0,1,2}$
B (CO^+)	219.14	0	X $^2\Sigma^+$	5.688	B $^2\Sigma^+$

Labels for CO^+ are $v_{Hi} \rightarrow v_{Lo}$.

Graphs iii & iv: Assignments for lines in plots of spectra in the range 100-260 nm. *All atomic assignments in this table are tentative.* Molecular assignments give position at $v = 0 \rightarrow v = 0$ and energies at T_e.

Key to Graphs 1-20

Feature Label	Posiition (nm)	Energy Low (eV)	Term Low	Energy High (eV)	Term High
Graphs 1: 260-300 nm					
1A (He)	Transitions to $2s\ ^3S_1$ from $11p$, $12p$, etc., of $^3P^0_{0,1,2}$				
1B (He)	267.71	19.820	$2s\ ^3S_1$	24.450	$10p\ ^3P^0_{0,1,2}$
1C (He)	269.61	19.820	$2s\ ^3S_1$	24.417	$9p\ ^3P^0_{0,1,2}$
1D (He)	272.32	19.820	$2s\ ^3S_1$	24.371	$8p\ ^3P^0_{0,1,2}$
1E (He)	276.38	19.820	$2s\ ^3S_1$	24.304	$7p\ ^3P^0_{0,1,2}$
1F (He)	282.91	19.820	$2s\ ^3S_1$	24.201	$6p\ ^3P^0_{0,1,2}$
1G (He)	294.51	19.820	$2s\ ^3S_1$	24.028	$5p\ ^3P^0_{0,1,2}$
Graphs 2: 300-350 nm					
2α (He_2)	301.3	17.86	a $^3\Sigma^+_u$	21.98	u $^3\Pi_g$
2β (He_2)	303.71	17.86	a $^3\Sigma^+_u$	21.95	t $^3\Pi_g$
2χ (He_2)	307.16	17.86	a $^3\Sigma^+_u$	21.9	s $^3\Pi_g$
2δ (He_2)	312.34	17.86	a $^3\Sigma^+_u$	21.84	r $^3\Pi_g$
2A (He)	318.77	19.820	$2s\ ^3S_1$	23.708	$4p\ ^3P^0_{0,1,2}$
2A (He)	318.77	19.820	$2s\ ^3S_1$	23.708	$4p\ ^3P^0_{0,1,2}$
2ε (He_2)	320.72	17.86	a $^3\Sigma^+_u$	21.73	p $^3\Pi_g$
2B (He)	323.13	20.616	$2s\ ^1S_0$	24.452	$10p\ ^1P^0_1$
2C (He)	325.83	20.616	$2s\ ^1S_0$	24.420	$9p\ ^1P^0_1$
2D (He)	329.68	20.616	$2s\ ^1S_0$	24.376	$8p\ ^1P^0_1$
2γ (He_2)	330.22	17.86	a $^3\Sigma^+_u$	21.62	n $^3\Sigma^+_g$
2φ (He_2)	330.81	18.15	A $^1\Sigma^+_u$	22.01	S $^1\Pi_g$
2E (He)	335.46	20.616	$2s\ ^1S_0$	24.311	$7p\ ^1P^0_1$
2η (He_2)	335.74	17.86	a $^3\Sigma^+_u$	21.56	l $^3\Pi_g$

2ι (He_2)	336.74	18.15	A $^1\Sigma_u^+$	21.94	R $^1\Pi_g$
2F (Ar)	340.62	11.828	$4s'$ $[1/2]_1^0$	15.467	$7p'$ $[1/2]_0$
2G (He)	344.76	20.616	$2s$ $^1S_0$	24.211	$6p$ $^1P_1^0$
2H (Ar)	346.11	11.624	$4s$ $[3/2]_1^0$	15.205	$6p'$ $[3/2]_2$
2κ (He_2)	346.33	18.15	A $^1\Sigma_u^+$	21.84	P $^1\Pi_g$
2λ	Unidentified				

Graphs 3: 350-375 nm

3A (Ar)	355.43	11.548	$4s$ $[3/2]_2^0$	15.036	$6p$ $[3/2]_2$
3B (He)	355.44	20.964	$2p$ $^3P_{0,1,2}^0$	24.451	$10d$ $^3D_{1,2,3}$
3α (He_2)	355.52	17.86	a $^3\Sigma_u^+$	21.35	k' $^3\Sigma_g^+$
3C (He)	356.30	20.964	$2p$ $^3P_{0,1,2}^0$	24.443	$10s$ $^3S_1$
3D (Ar)	356.33	11.723	$4s'$ $[1/2]_0^0$	15.202	$6p'$ $[3/2]_1$
3E (Ar)	356.77	11.548	$4s$ $[3/2]_2^0$	15.023	$6p$ $[5/2]_3$
3F (Ar)	357.23	11.828	$4s'$ $[1/2]_1^0$	15.298	$7p$ $[1/2]_0$
3G (He)	358.73	20.964	$2p$ $^3P_{0,1,2}^0$	24.419	$9d$ $^3D_{1,2,3}$
3β	Unidentified				
3H (Ar)	360.65	11.624	$4s$ $[3/2]_1^0$	15.060	$6p$ $[1/2]_0$
3I (He)	361.36	20.616	$2s$ $^1S_0$	24.046	$5p$ $^1P_1^0$
3J (Ar)	363.27	11.624	$4s$ $[3/2]_1^0$	15.036	$6p$ $[3/2]_2$
3K (He)	363.42	20.964	$2p$ $^3P_{0,1,2}^0$	24.375	$8d$ $^3D_{1,2,3}$
3L (Ar)	363.45	11.624	$4s$ $[3/2]_1^0$	15.034	$6p$ $[3/2]_1$
3χ (He_2)	363.53	18.15	A $^1\Sigma_u^+$	21.67	L $^1\Pi_g$
3M (Ar)	364.31	11.624	$4s$ $[3/2]_1^0$	15.026	$6p$ $[5/2]_2$
3N (Ar)	364.98	11.828	$4s'$ $[1/2]_1^0$	15.224	$6p'$ $[1/2]_0$
3O (He)	365.20	20.964	$2p$ $^3P_{0,1,2}^0$	24.358	$8s$ $^3S_1$
3P (Ar)	365.95	11.624	$4s$ $[3/2]_1^0$	15.011	$6p$ $[1/2]_1$

3Q (Ar)	367.07	11.828	$4s'\ [1/2]^0_1$	15.205	$6p'\ [3/2]_2$
3R (Ar)	367.52	11.828	$4s'\ [1/2]^0_1$	15.201	$6p'\ [1/2]_1$
3δ (He_2)	367.74	17.86	a $^3\Sigma^+_u$	21.24	i $^3\Pi_g$
3S (He)	370.50	20.964	$2p\ ^3P^0_{0,1,2}$	24.310	$7d\ ^3D_{1,2,3}$
3T (Ar^+)	371.47	16.43	$^3P3d\ ^4D_{5/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
3U (Ar^+)	373.55	16.44	$^3P3d\ ^4D_{3/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$

Graphs 4: 375-400 nm

4A (Ar^+)	375.05	16.46	$^3P3d\ ^4D_{1/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
4B (Ar)	377.04	11.723	$4s'\ [1/2]^0_0$	15.011	$6p\ [1/2]_1$
4α (He_2)	377.6	18.45	b $^3\Pi_g$	21.85	q $^3\Delta_u$
4β (He_2)	377.84	18.45	b $^3\Pi_g$	21.84	q $^3\Pi_u$
4C (Ar^+)	378.64	16.41	$^3P3d\ ^4D_{7/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
4χ (He_2)	378.66	18.45	b $^3\Pi_g$	21.84	q $^3\Sigma^+_u$
4δ (He_2)	380.37	18.45	b $^3\Pi_g$	21.82	o $^3\Sigma^+_u$
4D (Ar^+)	380.86	16.43	$^3P3d\ ^4D_{5/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
4E (He)	381.96	20.964	$2p\ ^3P^0_{0,1,2}$	24.209	$6d\ ^3D_{1,2,3}$
4F (Ar^+)	383.04	16.44	$^3P3d\ ^4D_{3/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
4G (He)	383.37	21.218	$2p\ ^1P^0_1$	24.451	$10d\ ^1D_2$
4H (Ar)	383.47	11.828	$4s'\ [1/2]^0_1$	15.060	$6p\ [1/2]_0$
4I (He)	386.75	20.964	$2p\ ^3P^0_{0,1,2}$	24.169	$6s\ ^3S_1$
4J (He)	387.18	21.218	$2p\ ^1P^0_1$	24.420	$9d\ ^1D_2$
4K (Ar^+)	387.53	16.44	$^3P3d\ ^4D_{3/2}$	19.64	$^3P4p\ 4D^0_{1/2}$
4ε (He_2)	397.58	18.45	b $^3\Pi_g$	21.68	m $^3\Delta_u$
4L (He)	388.86	19.820	$2s\ ^3S_{0,1,2}$	23.007	$3p\ ^3P^0_1$
4M (Ar^+)	389.14	16.46	$^3P3d\ ^4D_{1/2}$	19.64	$^3P4p\ 4D^0_{1/2}$
4N (Ar^+)	389.20	16.43	$^3P3d\ ^4D_{5/2}$	19.61	$^3P4p\ 4D^0_{3/2}$

4O (Ar)	389.47	11.828	$4s'\ [1/2]^0_1$	15.011	$6p\ [1/2]_1$
4P (Ar^+)	391.48	16.44	$^3P3d\ ^4D_{3/2}$	19.61	$^3P4p\ 4D^0_{3/2}$
4Q (He)	392.65	21.218	$2p\ ^1P^0_1$	24.375	$8d\ ^1D_2$
4R (Ar^+)	393.12	16.46	$^3P3d\ ^4D_{1/2}$	19.61	$^3P4p\ 4D^0_{3/2}$
4S (He)	393.59	21.218	$2p\ ^1P^0_1$	24.367	$8s\ ^1S_0$
4T (Ar^+)	394.43	16.41	$^3P3d\ ^4D_{7/2}$	19.55	$^3P4p\ 4D^0_{5/2}$
4U (Ar)	394.75	11.548	$4s\ [3/2]^0_2$	14.688	$5p'\ [3/2]_2$
4V (Ar)	394.90	11.548	$4s\ [3/2]^0_2$	14.687	$5p'\ [1/2]_1$
4W (He)	396.47	20.616	$2s\ ^1S_0$	23.742	$4p\ ^1P^0_1$
4X (Ar^+)	396.84	16.43	$^3P3d\ ^4D_{5/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
4Y (Ar^+)	397.48	16.64	$^3P4s\ ^4P_{5/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
4φ (He_2)	398.88	18.45	b $^3\Pi_g$	21.67	m $^3\Pi_u$
4Z (Ar^+)	399.21	16.44	$^3P3d\ ^4D_{3/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
4γ (He_2)	399.7	18.45	b $^3\Pi_g$	21.66	m $^3\Sigma^+_u$

Graphs 5: 400-425 nm

5α (He_2)	400.33	18.15	A $^1\Sigma^+_u$	21.36	I $^1\Pi_g$
5A (He)	400.93	21.218	$2p\ ^1P^0_1$	24.310	$7d\ ^1D_2$
5B (Ar^+)	401.39	16.41	$^3P3d\ ^4D_{7/2}$	19.49	$^3P4p\ ^4D^0_{7/2}$
5C (He)	402.40	21.218	$2p\ ^1P^0_1$	24.298	$7s\ ^1S_0$
5D (He)	402.62	20.964	$2p\ ^3P^0_{0,1,2}$	24.043	$5d\ ^3D_{1,2,3}$
5β (He_2)	403.15	18.45	b $^3\Pi_g$	21.54	k $^3\Sigma^+_u$
5E (Ar^+)	403.88	16.43	$^3P3d\ ^4D_{5/2}$	19.49	$^3P4p\ ^4D^0_{7/2}$
5F (Ar)	404.44	11.624	$4s\ [3/2]^0_1$	14.688	$5p'\ [3/2]_2$
5G (Ar)	404.60	11.624	$4s\ [3/2]^0_1$	14.687	$5p'\ [1/2]_1$
5H (Ar)	405.45	11.624	$4s\ [3/2]^0_1$	14.681	$5p'\ [3/2]_1$
5I (Ar^+)	408.24	16.64	$^3P4s\ ^4P_{5/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$

5J (Ar^+)	411.28	16.75	$^3P4s\ ^4P_{3/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
5K (He)	412.08	20.964	$2p\ ^3P^0_{0,1,2}$	23.972	$5s\ ^3S_1$
5L (He)	414.38	21.218	$2p\ ^1P^0_1$	24.209	$6d\ ^1D_2$
5χ (He_2)	415.8	18.59	B $^1\Pi_g$	21.68	M $^1\Delta_u$
5M (Ar)	415.86	11.548	$4s\ [3/2]^0_2$	14.529	$5p\ [3/2]_2$
5N (Ar)	416.42	11.548	$4s\ [3/2]^0_2$	14.525	$5p\ [3/2]_1$
5δ (He_2)	416.67	18.59	B $^1\Pi_g$	21.67	M $^1\Pi_u$
5O (He)	416.90	21.218	$2p\ ^1P^0_1$	24.191	$6s\ ^1S_0$
5ε (He_2)	417.36	18.59	B $^1\Pi_g$	Not Avail.	M $^1\Sigma^+_u$
5P (Ar^+)	417.84	16.64	$^3P4s\ ^4P_{5/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
5Q (Ar)	418.19	11.723	$4s'\ [1/2]^0_0$	14.687	$5p'\ [1/2]_1$
5R (Ar)	419.07	11.548	$4s\ [3/2]^0_2$	14.506	$5p\ [5/2]_2$
5S (Ar)	419.10	11.723	$4s'\ [1/2]^0_0$	14.681	$5p'\ [3/2]_1$
5T (Ar)	419.83	11.624	$4s\ [3/2]^0_1$	14.576	$5p\ [1/2]_0$
5U (Ar)	420.07	11.548	$4s\ [3/2]^0_2$	14.499	$5p\ [5/2]_3$
5V (Ar^+)	420.20	16.81	$^3P4s\ ^4P_{1/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
5W (Ar^+)	422.82	16.75	$^3P4s\ ^4P_{3/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
5φ (He_2)	423.78	17.86	a $^3\Sigma^+_u$	20.79	g $^3\Sigma^+_g$

Graphs 6: 425-450 nm

6A (Ar)	425.12	11.548	$4s\ [3/2]^0_2$	14.464	$5p\ [1/2]_1$
6B (Ar)	425.94	11.828	$4s'\ [1/2]^0_1$	14.738	$5p'\ [1/2]_0$
6C (Ar)	426.63	11.624	$4s\ [3/2]^0_1$	14.529	$5p\ [3/2]_2$
6D (Ar^+)	426.65	16.64	$^3P4s\ ^4P_{5/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
6E (Ar)	427.22	11.624	$4s\ [3/2]^0_1$	14.525	$5p\ [3/2]_1$
6F (Ar^+)	428.29	16.75	$^3P4s\ ^4P_{3/2}$	19.64	$^3P4p\ ^4D^0_{1/2}$
6G (Ar)	430.01	11.624	$4s\ [3/2]^0_1$	14.506	$5p\ [5/2]_2$

6H (Ar^+)	433.12	16.75	$^3P4s\ ^4P_{3/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
6I (Ar^+)	433.20	16.44	$^3P3d\ ^4D_{3/2}$	19.31	$^3P4p\ ^4P^0_{1/2}$
6J (Ar)	433.36	11.828	$4s'\ [1/2]^0_1$	14.688	$5p'\ [3/2]_2$
6K (Ar)	433.53	11.828	$4s'\ [1/2]^0_1$	14.687	$5p'\ [1/2]_1$
6L (Ar)	434.52	11.828	$4s'\ [1/2]^0_1$	14.681	$5p'\ [3/2]_1$
6M (Ar^+)	434.81	16.64	$^3P4s\ ^4P_{5/2}$	19.49	$^3P4p\ ^4D^0_{7/2}$
6N (Ar^+)	435.22	16.46	$^3P3d\ ^4D_{1/2}$	19.31	$^3P4p\ ^4P^0_{1/2}$
6O (Ar)	436.38	11.624	$4s\ [3/2]^0_1$	14.464	$5p\ [1/2]_1$
6P (Ar^+)	437.13	16.43	$^3P3d\ ^4D_{5/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
6Q (Ar^+)	437.97	16.81	$^3P4s\ ^4P_{1/2}$	19.64	$^3P4p\ ^4D^0_{1/2}$
6R (He)	438.79	21.218	$2p\ ^1P^0_1$	24.043	$5d\ ^1D_2$
6S (Ar^+)	440.01	16.44	$^3P3d\ ^4D_{3/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
6T (Ar^+)	440.10	16.41	$^3P3d\ ^4D_{7/2}$	19.22	$^3P4p\ ^4P^0_{5/2}$
6α (He_2)	440.44	18.45	b $^3\Pi_g$	21.27	j $^3\Delta_u$
6U (Ar^+)	442.09	16.46	$^3P3d\ ^4D_{1/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
6V (Ar)	442.40	11.723	$4s'\ [1/2]^0_0$	14.525	$5p\ [3/2]_1$
6W (Ar^+)	442.60	16.75	$^3P4s\ ^4P_{3/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
6X (Ar^+)	443.02	16.81	$^3P4s\ ^4P_{1/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
6Y (Ar^+)	443.10	16.43	$^3P3d\ ^4D_{5/2}$	19.22	$^3P4p\ ^4P^0_{5/2}$
6Z (He)	443.76	21.218	$2p\ ^1P^0_1$	24.011	$5s\ ^1S_0$
6β (He_2)	444.01	18.45	b $^3\Pi_g$	21.25	j $^3\Pi_u$
6χ (He_2)	445.68	18.45	b $^3\Pi_g$	21.24	j $^3\Sigma^+_u$
6AA (Ar^+)	446.06	16.44	$^3P3d\ ^4D_{3/2}$	19.22	$^3P4p\ ^4P^0_{5/2}$
6BB (He)	447.15	20.964	$2p\ ^3P^0_{0,1,2}$	23.736	$4d\ ^3D_{1,2,3}$

Graphs 7: 450-475 nm

7A (Ar)	451.07	11.828	$4s'\ [1/2]^0_1$	14.576	$5p\ [1/2]_0$
7B (Ar)	452.23	11.723	$4s'\ [1/2]^0_0$	14.464	$5p\ [1/2]_1$
7C (Ar)	454.47	12.907	$4p\ [1/2]_1$	15.634	$11d\ [1/2]^0_1$
7α (He_2)	454.71	18.45	b $^3\Pi_g$	21.19	h $^3\Sigma^+_u$
7D (Ar)	455.43	12.907	$4p\ [1/2]_1$	15.629	$7d'\ [3/2]^0_2$
7E (Ar)	458.50	12.907	$4p\ [1/2]_1$	15.610	$10d\ [3/2]^0_2$
7F (Ar)	458.66	12.907	$4p\ [1/2]_1$	15.610	$10d\ [1/2]^0_1$
7G (Ar)	458.72	12.907	$4p\ [1/2]_1$	15.609	$10d\ [1/2]^0_0$
7H (Ar)	458.93	11.828	$4s'\ [1/2]^0_1$	14.529	$5p\ [3/2]_2$
7I (Ar)	459.61	11.828	$4s'\ [1/2]^0_1$	14.525	$5p\ [3/2]_1$
7β (He_2)	462.37	18.59	B $^1\Pi_g$	21.38	J $^1\Delta_u$
7J (Ar)	462.84	11.828	$4s'\ [1/2]^0_1$	14.506	$5p\ [5/2]_2$
7K (Ar)	464.21	12.907	$4p\ [1/2]_1$	15.577	$9d\ [3/2]^0_2$
7L (Ar)	464.75	12.907	$4p\ [1/2]_1$	15.574	$9d\ [1/2]^0_1$
7χ (He_2)	464.96	17.86	a $^3\Sigma^+_u$	20.53	e $^3\Pi_g$
7δ (He_2)	465.06	18.59	B $^1\Pi_g$	21.36	J $^1\Pi_u$
7ε (He_2)	466.54	18.59	B $^1\Pi_g$	21.35	J $^1\Sigma^+_u$
7M (Ar)	470.23	11.828	$4s'\ [1/2]^0_1$	14.464	$5p\ [1/2]_1$
7N (He)	471.32	20.964	$2p\ ^3P^0_{0,1,2}$	23.594	$4s\ ^3S_1$
7φ (He_2)	472.51	18.59	B $^1\Pi_g$	21.32	H $^1\Sigma^+_u$
7O (Ar^+)	472.69	17.14	$^3P4s\ ^2P3/2$	19.76	$^3P4p\ ^2D^0_{3/2}$
7P (Ar^+)	473.59	16.64	$^3P4s\ ^4P5/2$	19.26	$^3P4p\ ^4P^0_{3/2}$
7Q (Ar)	474.68	12.907	$4p\ [1/2]_1$	15.518	$8d\ [1/2]^0_0$

Graphs 8: 475-500 nm

8A (Ar)	475.29	12.907	$4p\ [1/2]_1$	15.515	$8d\ [1/2]^0_1$
8B (Ar)	476.87	12.907	$4p\ [1/2]_1$	15.506	$6d'\ [3/2]^0_2$
8C (Ar)	479.87	13.076	$4p\ [5/2]_3$	15.659	$12d\ [7/2]^0_4$
8D (Ar^+)	480.60	16.64	$^3P4s\ ^4P_{5/2}$	19.22	$^3P4p\ ^4P^0_{5/2}$
8α	Unidentified				
8E (Ar)	483.60	13.076	$4p\ [5/2]_3$	15.639	$11d\ [7/2]^0_4$
8F (Ar)	483.67	12.907	$4p\ [1/2]_1$	15.470	$9s\ [3/2]^0_2$
8G (Ar^+)	484.78	16.75	$^3P4s\ ^4P_{3/2}$	19.31	$^3P4p\ ^4P^0_{1/2}$
8H (Ar)	487.63	12.907	$4p\ [1/2]_1$	15.449	$7d\ [3/2]^0_2$
8I (Ar^+)	487.99	17.14	$^3P4s\ ^2P_{3/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
8J (Ar)	488.63	13.076	$4p\ [5/2]_3$	15.612	$10d\ [7/2]^0_4$
8K (Ar)	488.79	12.907	$4p\ [1/2]_1$	15.443	$7d\ [1/2]^0_1$
8L (Ar)	489.47	12.907	$4p\ [1/2]_1$	15.439	$7d\ [1/2]^0_0$
8β (He_2)	491.04	19.22	c $^3\Sigma^+_g$	21.85	q $^3\Delta_u$
8χ (He_2)	491.88	19.22	c $^3\Sigma^+_g$	21.84	q $^3\Pi_u$
8M (Ar)	492.10	13.095	$4p\ [5/2]_2$	15.614	$10d\ [7/2]^0_3$
8N (He)	492.19	21.218	$2p\ ^1P^0_1$	23.736	$4d\ ^1D_2$
8δ (He_2)	492.9	19.22	c $^3\Sigma^+_g$	21.84	q $^3\Sigma^+_u$
8O (Ar^+)	493.32	16.75	$^3P4s\ ^4P_{3/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
8P (Ar)	493.77	13.076	$4p\ [5/2]_3$	15.586	$11s\ [3/2]^0_2$
8Q (Ar^+)	495.29	17.14	$^3P4s\ ^2P_{3/2}$	19.64	$^3P4p\ ^4D^0_{1/2}$
8R (Ar)	495.68	13.076	$4p\ [5/2]_3$	15.576	$9d\ [7/2]^0_4$
8ε (He_2)	495.82	19.22	c $^3\Sigma^+_g$	21.82	o $^3\Sigma^+_u$
8S (Ar^+)	496.51	17.27	$^3P4s\ ^2P_{1/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
8T (Ar^+)	497.22	16.81	$^3P4s\ ^4P_{1/2}$	19.31	$^3P4p\ ^4P^0_{1/2}$
8U (Ar)	498.99	13.095	$4p\ [5/2]_2$	15.579	$9d\ [7/2]^0_3$

Graphs 9: 500-525 nm

9A (Ar^+)	500.93	16.75	$^3P4s\ ^4P_{3/2}$	19.22	$^3P4p\ ^4P^0_{5/2}$
9B (He)	501.57	20.616	$2s\ ^1S_0$	23.087	$3p\ ^1P^0_1$
9C (Ar^+)	501.76	17.14	$^3P4s\ ^2P_{3/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
9D (Ar)	503.20	13.076	$4p\ [5/2]_3$	15.539	$10s\ [3/2]^0_2$
9E (He)	504.77	21.218	$2p\ ^1P^0_1$	23.674	$4s\ ^1S_0$
9F (Ar)	504.88	12.907	$4p\ [1/2]_1$	15.362	$8s\ [3/2]^0_2$
9G (Ar)	505.42	12.907	$4p\ [1/2]_1$	15.360	$7s'\ [1/2]^0_1$
9H (Ar)	505.65	12.907	$4p\ [1/2]_1$	15.358	$7s'\ [1/2]^0_0$
9I (Ar)	506.01	13.076	$4p\ [5/2]_3$	15.525	$8d\ [7/2]^0_4$
9J (Ar^+)	506.20	16.81	$^3P4s\ ^4P_{1/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
9K (Ar)	507.10	13.095	$4p\ [5/2]_2$	15.539	$10s\ [3/2]^0_1$
9L (Ar)	507.31	12.907	$4p\ [1/2]_1$	15.350	$6d\ [3/2]^0_2$
9M (Ar)	507.80	13.076	$4p\ [5/2]_3$	15.517	$6d'\ [5/2]^0_3$
9N (Ar)	508.71	13.095	$4p\ [5/2]_2$	15.532	$8d\ [7/2]^0_3$
9O (Ar)	510.47	13.153	$4p\ [3/2]_1$	15.581	$9d\ [5/2]^0_2$
9P (Ar)	511.82	13.095	$4p\ [5/2]_2$	15.517	$6d'\ [5/2]^0_3$
9Q (Ar)	512.78	13.095	$4p\ [5/2]_2$	15.512	$6d'\ [5/2]^0_2$
9α (He_2)	513.44	18.15	A $^1\Sigma^+_u$	20.57	E $^1\Pi_g$
9R (Ar^+)	514.53	17.14	$^3P4s\ ^2P_{3/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
9S (Ar)	515.14	12.907	$4p\ [1/2]_1$	15.313	$6d\ [1/2]^0_0$
9T (Ar)	515.23	12.907	$4p\ [1/2]_1$	15.313	$5d'\ [5/2]^0_2$
9U (Ar)	516.23	12.907	$4p\ [1/2]_1$	15.308	$6d\ [1/2]^0_1$
9V (Ar)	517.75	13.076	$4p\ [5/2]_3$	15.470	$9s\ [3/2]^0_2$
9W (Ar)	518.77	12.907	$4p\ [1/2]_1$	15.296	$5d'\ [3/2]^0_2$
9X (Ar)	519.27	13.076	$4p\ [5/2]_3$	15.463	$7d\ [5/2]^0_3$
9Y (Ar)	519.41	13.153	$4p\ [3/2]_1$	15.540	$8s'\ [1/2]^0_0$

9Z (Ar)	521.05	13.076	$4p\ [5/2]_3$	15.455	$7d\ [7/2]^0_3$
9AA (Ar)	521.48	13.095	$4p\ [5/2]_2$	15.472	$7d\ [3/2]^0_1$
9BB (Ar^+)	521.51	17.27	$^3P4s\ ^2P_{1/2}$	19.64	$^3P4p\ ^4D^0_{1/2}$
9CC (Ar)	521.63	13.095	$4p\ [5/2]_2$	15.471	$9s\ [3/2]^0_1$
9DD (Ar)	522.13	13.076	$4p\ [5/2]_3$	15.450	$7d\ [7/2]^0_4$
9EE (Ar)	524.11	13.095	$4p\ [5/2]_2$	15.460	$7d\ [5/2]^0_2$
9FF (Ar)	524.62	13.172	$4p\ [3/2]_2$	15.535	$8d\ [5/2]^0_3$
9GG (Ar)	524.92	13.172	$4p\ [3/2]_2$	15.533	$8d\ [3/2]^0_2$

Graphs 10: 525-550 nm

10α (He_2)	525.24	19.22	c $^3\Sigma^+_g$	21.68	m $^3\Delta_u$
10A (Ar)	525.28	13.095	$4p\ [5/2]_2$	15.455	$7d\ [7/2]^0_3$
10B (Ar)	525.45	13.153	$4p\ [3/2]_1$	15.512	$6d'\ [5/2]^0_2$
10β (He_2)	527.87	19.22	c $^3\Sigma^+_g$	21.67	m $^3\Pi_u$
10C (Ar)	528.61	13.172	$4p\ [3/2]_2$	15.517	$6d'\ [5/2]^0_3$
10D (Ar^+)	528.69	17.27	$^3P4s\ ^2P_{1/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
10E (Ar)	529.00	13.172	$4p\ [3/2]_2$	15.515	$8d\ [1/2]^0_1$
10χ (He_2)	529.13	19.22	c $^3\Sigma^+_g$	21.66	m $^3\Sigma^+_u$
10F (Ar)	530.95	13.172	$4p\ [3/2]_2$	15.506	$6d'\ [3/2]^0_2$
10G (Ar)	531.77	13.302	$4p'\ [3/2]_2$	15.633	$7d'\ [5/2]^0_3$
10δ (He_2)	535.23	19.22	c $^3\Sigma^+_g$	21.54	k $^3\Sigma^+_u$
10H (Ar)	537.35	13.153	$4p\ [3/2]_1$	15.460	$7d\ [5/2]^0_2$
10I (Ar)	539.40	13.172	$4p\ [3/2]_2$	15.470	$9s\ [3/2]^0_2$
10J (Ar)	541.05	13.172	$4p\ [3/2]_2$	15.463	$7d\ [5/2]^0_3$
10K (Ar)	542.14	13.076	$4p\ [5/2]_3$	15.362	$8s\ [3/2]^0_2$
10L (Ar)	544.00	12.907	$4p\ [1/2]_1$	15.186	$7s\ [3/2]^0_1$
10M (Ar)	544.22	13.076	$4p\ [5/2]_3$	15.353	$6d\ [5/2]^0_3$

10N (Ar)	545.17	12.907	$4p\ [1/2]_1$	15.181	$7s\ [3/2]^0_2$
10O (Ar)	545.74	13.095	$4p\ [5/2]_2$	15.366	$8s\ [3/2]^0_1$
10P (Ar)	545.97	13.076	$4p\ [5/2]_3$	15.346	$6d\ [7/2]^0_3$
10Q (Ar)	546.72	13.095	$4p\ [5/2]_2$	15.362	$8s\ [3/2]^0_2$
10R (Ar)	547.35	13.095	$4p\ [5/2]_2$	15.360	$7s'\ [1/2]^0_1$
10S (Ar)	549.01	13.095	$4p\ [5/2]_2$	15.353	$6d\ [5/2]^0_2$
10T (Ar)	549.21	13.283	$4p'\ [3/2]_1$	15.540	$8s'\ [1/2]^0_0$
10U (Ar)	549.59	13.076	$4p\ [5/2]_3$	15.331	$6d\ [7/2]^0_4$

Graphs 11: 550-570 nm

11A (Ar)	550.61	13.095	$4p\ [5/2]_2$	15.346	$6d\ [7/2]^0_3$
11B (Ar)	552.50	13.076	$4p\ [5/2]_3$	15.319	$5d'\ [5/2]^0_3$
11C (Ar)	552.90	13.273	$4p\ [1/2]_0$	15.515	$8d\ [1/2]^0_1$
11D (Ar)	553.45	13.302	$4p'\ [3/2]_2$	15.542	$8s'\ [1/2]^0_1$
11E (Ar)	554.09	13.076	$4p\ [5/2]_3$	15.313	$5d'\ [5/2]^0_2$
11F (Ar)	555.27	13.283	$4p'\ [3/2]_1$	15.515	$8d\ [1/2]^0_1$
11G (Ar)	555.87	12.907	$4p\ [1/2]_1$	15.137	$5d\ [3/2]^0_2$
11H (Ar)	555.97	13.283	$4p'\ [3/2]_1$	15.512	$6d'\ [5/2]^0_2$
11I (Ar)	557.25	13.095	$4p\ [5/2]_2$	15.319	$5d'\ [5/2]^0_3$
11J (Ar)	557.42	13.283	$4p'\ [3/2]_1$	15.506	$6d'\ [3/2]^0_2$
11K (Ar)	558.19	13.076	$4p\ [5/2]_3$	15.296	$5d'\ [3/2]^0_2$
11L (Ar)	558.87	13.095	$4p\ [5/2]_2$	15.313	$5d'\ [5/2]^0_2$
11M (Ar)	559.75	13.302	$4p'\ [3/2]_2$	15.517	$6d'\ [5/2]^0_3$
11N (Ar)	560.67	12.907	$4p\ [1/2]_1$	15.118	$5d\ [1/2]^0_1$
11O (Ar)	561.80	13.153	$4p\ [3/2]_1$	15.360	$7s'\ [1/2]^0_1$
11P (Ar)	562.09	13.153	$4p\ [3/2]_1$	15.358	$7s'\ [1/2]^0_0$
11Q (Ar)	562.38	13.302	$4p'\ [3/2]_2$	15.506	$6d'\ [3/2]^0_2$

11R (Ar)	563.56	13.153	$4p\ [3/2]_1$	15.353	$6d\ [5/2]^0_2$
11S (Ar)	563.73	13.273	$4p\ [1/2]_0$	15.472	$7d\ [3/2]^0_1$
11T (Ar)	563.91	13.273	$4p\ [1/2]_0$	15.471	$9s\ [3/2]^0_1$
11U (Ar)	564.14	13.153	$4p\ [3/2]_1$	15.350	$6d\ [3/2]^0_2$
11V (Ar)	564.87	13.172	$4p\ [3/2]_2$	15.366	$8s\ [3/2]^0_1$
11W (Ar)	565.07	12.907	$4p\ [1/2]_1$	15.101	$5d\ [1/2]^0_0$
11X (Ar)	565.91	13.172	$4p\ [3/2]_2$	15.362	$8s\ [3/2]^0_2$
11Y (Ar)	568.19	13.172	$4p\ [3/2]_2$	15.353	$6d\ [5/2]^0_3$
11Z (Ar)	568.37	13.172	$4p\ [3/2]_2$	15.353	$6d\ [5/2]^0_2$

Graphs 12: 570-590 nm

12A (Ar)	570.09	13.172	$4p\ [3/2]_2$	15.346	$6d\ [7/2]^0_3$
12B (Ar)	571.25	13.273	$4p\ [1/2]_0$	15.443	$7d\ [1/2]^0_1$
12C (Ar^+)	572.43	17.14	$^3P4s\ ^2P_{3/2}$	19.31	$^3P4p\ ^4P^0_{1/2}$
12α (He_2)	573.48	18.45	b $^3\Pi_g$	20.62	f $^3\Delta_u$
12D (Ar)	573.95	13.153	$4p\ [3/2]_1$	15.313	$5d'\ [5/2]^0_2$
12E (Ar)	577.21	13.172	$4p\ [3/2]_2$	15.319	$5d'\ [5/2]^0_3$
12F (Ar)	577.40	13.302	$4p'\ [3/2]_2$	15.449	$7d\ [3/2]^0_2$
12G (Ar)	578.35	13.153	$4p\ [3/2]_1$	15.296	$5d'\ [3/2]^0_2$
12H (Ar)	578.95	13.172	$4p\ [3/2]_2$	15.313	$5d'\ [5/2]^0_2$
12I (Ar)	579.04	13.302	$4p'\ [3/2]_2$	15.443	$7d\ [1/2]^0_1$
12J (Ar)	580.21	13.172	$4p\ [3/2]_2$	15.308	$6d\ [1/2]^0_1$
12K (Ar)	583.43	13.172	$4p\ [3/2]_2$	15.296	$5d'\ [3/2]^0_2$
12L (Ar)	584.38	13.328	$4p'\ [1/2]_1$	15.449	$7d\ [3/2]^0_2$
12M (Ar^+)	584.38	17.14	$^3P4s\ ^2P_{3/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
12N (Ar)	586.03	12.907	$4p\ [1/2]_1$	15.022	$6s'\ [1/2]^0_1$
12O (He)	587.56	20.964	$2p\ ^3P^0_{0,1,2}$	23.074	$3d\ ^3D_{1,2,3}$

12P (Ar)	588.26	12.907	$4p\ [1/2]_1$	15.014	$6s'\ [1/2]^0_0$
12β (He_2)	588.73	18.45	b $^3\Pi_g$	20.57	f $^3\Pi_u$
12Q (Ar)	588.86	13.076	$4p\ [5/2]_3$	15.181	$7s\ [3/2]^0_2$

Graphs 13: 590-600 nm

13A (Ar)	591.21	12.907	$4p\ [1/2]_1$	15.004	$4d'\ [3/2]^0_1$
13B (Ar)	591.66	13.095	$4p\ [5/2]_2$	15.190	$5d\ [3/2]^0_1$
13C (Ar)	592.71	13.076	$4p\ [5/2]_3$	15.167	$5d\ [5/2]^0_3$
13D (Ar)	592.88	13.095	$4p\ [5/2]_2$	15.186	$7s\ [3/2]^0_1$
13E (Ar)	594.09	13.273	$4p\ [1/2]_0$	15.360	$7s'\ [1/2]^0_1$
13F (Ar)	594.27	13.095	$4p\ [5/2]_2$	15.181	$7s\ [3/2]^0_2$
13G (Ar)	594.39	13.076	$4p\ [5/2]_3$	15.161	$5d\ [5/2]^0_2$
13H (Ar)	594.93	13.283	$4p'\ [3/2]_1$	15.366	$8s\ [3/2]^0_1$
13I (Ar^+)	595.09	17.14	$^3P4s\ ^2P_{3/2}$	19.22	$^3P4p\ ^4P^0_{5/2}$
13J (Ar)	596.45	13.273	$4p\ [1/2]_0$	15.351	$5d'\ [3/2]^0_1$
13K (Ar)	596.83	13.283	$4p'\ [3/2]_1$	15.360	$7s'\ [1/2]^0_1$
13L (Ar)	597.16	13.283	$4p'\ [3/2]_1$	15.358	$7s'\ [1/2]^0_0$
13M (Ar)	598.19	13.095	$4p\ [5/2]_2$	15.167	$5d\ [5/2]^0_3$
13N (Ar)	598.73	13.076	$4p\ [5/2]_3$	15.146	$5d\ [7/2]^0_3$
13O (Ar)	598.81	13.283	$4p'\ [3/2]_1$	15.353	$6d\ [5/2]^0_2$
13P (Ar)	599.47	13.283	$4p'\ [3/2]_1$	15.350	$6d\ [3/2]^0_2$
13Q (Ar)	599.90	13.095	$4p\ [5/2]_2$	15.161	$5d\ [5/2]^0_2$

Graphs 14: 600-615 nm

14A (Ar)	600.57	13.302	$4p'\ [3/2]_2$	15.366	$8s\ [3/2]^0_1$
14B (Ar)	601.37	13.076	$4p\ [5/2]_3$	15.137	$5d\ [3/2]^0_2$
14C (Ar)	602.52	13.302	$4p'\ [3/2]_2$	15.360	$7s'\ [1/2]^0_1$
14α (He_2)	603.02	19.22	c $^3\Sigma^+_g$	21.27	j $^3\Delta_u$

14D (Ar)	603.21	13.076	$4p\ [5/2]_3$	15.131	$5d\ [7/2]^0_4$
14E (Ar)	604.32	13.095	$4p\ [5/2]_2$	15.146	$5d\ [7/2]^0_3$
14F (Ar)	605.27	12.907	$4p\ [1/2]_1$	14.955	$4d'\ [5/2]^0_2$
14G (Ar)	605.94	12.907	$4p\ [1/2]_1$	14.953	$4d'\ [3/2]^0_2$
14H (Ar)	606.48	13.302	$4p'\ [3/2]_2$	15.346	$6d\ [7/2]^0_3$
14I (Ar^+)	607.74	17.27	$^3P4s\ ^2P_{1/2}$	19.31	$^3P4p\ ^4P^0_{1/2}$
14J (Ar)	608.12	13.328	$4p'\ [1/2]_1$	15.366	$8s\ [3/2]^0_1$
14K (Ar)	608.59	13.153	$4p\ [3/2]_1$	15.190	$5d\ [3/2]^0_1$
14L (Ar)	609.08	13.273	$4p\ [1/2]_0$	15.308	$6d\ [1/2]^0_1$
14β (He_2)	609.73	19.22	c $^3\Sigma^+_g$	21.25	j $^3\Pi_u$
14M (Ar)	609.88	13.153	$4p\ [3/2]_1$	15.186	$7s\ [3/2]^0_1$
14N (Ar)	610.12	13.328	$4p'\ [1/2]_1$	15.360	$7s'\ [1/2]^0_1$
14O (Ar)	610.46	13.328	$4p'\ [1/2]_1$	15.358	$7s'\ [1/2]^0_0$
14P (Ar)	610.56	13.283	$4p'\ [3/2]_1$	15.313	$5d'\ [5/2]^0_2$
14χ (He_2)	611.21	18.59	B $^1\Pi_g$	20.62	F $^1\Delta_u$
14Q (Ar)	611.35	13.153	$4p\ [3/2]_1$	15.181	$7s\ [3/2]^0_2$
14R (Ar)	611.97	13.283	$4p'\ [3/2]_1$	15.308	$6d\ [1/2]^0_1$
14S (Ar)	612.19	13.328	$4p'\ [1/2]_1$	15.353	$6d\ [5/2]^0_2$
14T (Ar)	612.74	13.095	$4p\ [5/2]_2$	15.118	$5d\ [1/2]^0_1$
14U (Ar)	612.87	13.328	$4p'\ [1/2]_1$	15.350	$6d\ [3/2]^0_2$
14δ (He_2)	612.88	19.22	c $^3\Sigma^+_g$	21.24	j $^3\Sigma^+_u$
14V (Ar^+)	613.87	17.74	$^3P3d\ ^4F_{5/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
14W (Ar)	614.54	13.302	$4p'\ [3/2]_2$	15.319	$5d'\ [5/2]^0_3$

Graphs 15: 615-630 nm

15A (Ar)	615.52	13.172	$4p\ [3/2]_2$	15.186	$7s\ [3/2]^0_1$
15B (Ar)	616.51	13.302	$4p'\ [3/2]_2$	15.313	$5d'\ [5/2]^0_2$

15C (Ar)	617.02	13.172	$4p\ [3/2]_2$	15.181	$7s\ [3/2]^0_2$
15D (Ar)	617.31	13.153	$4p\ [3/2]_1$	15.161	$5d\ [5/2]^0_2$
15E (Ar)	617.94	13.302	$4p'\ [3/2]_2$	15.308	$6d\ [1/2]^0_1$
15F (Ar^+)	621.22	17.27	$^3P4s\ ^2P_{1/2}$	19.26	$^3P4p\ ^4P^0_{3/2}$
15G (Ar)	621.25	13.172	$4p\ [3/2]_2$	15.167	$5d\ [5/2]^0_3$
15H (Ar)	621.59	13.302	$4p'\ [3/2]_2$	15.296	$5d'\ [3/2]^0_2$
15I (Ar)	623.09	13.172	$4p\ [3/2]_2$	15.161	$5d\ [5/2]^0_2$
15J (Ar^+)	623.97	17.78	$^3P3d\ ^4F_{3/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
15K (Ar^+)	624.31	17.69	$^3P3d\ ^4F_{7/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
15L (Ar)	624.34	13.328	$4p'\ [1/2]_1$	15.313	$6d\ [1/2]^0_0$
15M (Ar)	624.47	13.328	$4p'\ [1/2]_1$	15.313	$5d'\ [5/2]^0_2$
15α (He_2)	624.68	18.59	B $^1\Pi_g$	20.58	F $^1\Pi_u$
15N (Ar)	624.84	13.153	$4p\ [3/2]_1$	15.137	$5d\ [3/2]^0_2$
15O (Ar)	627.86	13.172	$4p\ [3/2]_2$	15.146	$5d\ [7/2]^0_3$
15P (Ar)	629.69	13.328	$4p'\ [1/2]_1$	15.296	$5d'\ [3/2]^0_2$

Graphs 16: 630-650 nm

16α (He_2)	630.09	19.22	c $^3\Sigma^+_g$	21.19	h $^3\Sigma^+_u$
16A (Ar)	630.77	13.172	$4p\ [3/2]_2$	15.137	$5d\ [3/2]^0_2$
16B (Ar)	630.92	13.153	$4p\ [3/2]_1$	15.118	$5d\ [1/2]^0_1$
16β (He_2)	631.41	18.59	B $^1\Pi_g$	20.56	F $^1\Sigma^+_u$
16C (Ar)	636.49	13.153	$4p\ [3/2]_1$	15.101	$5d\ [1/2]^0_0$
16D (Ar)	636.96	13.172	$4p\ [3/2]_2$	15.118	$5d\ [1/2]^0_1$
16E (Ar)	638.47	12.907	$4p\ [1/2]_1$	14.848	$6s\ [3/2]^0_1$
16F (Ar^+)	639.92	17.74	$^3P3d\ ^4F_{5/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
16χ (He_2)	640.08	18.45	b $^3\Pi_g$	20.39	d $^3\Sigma^+_u$
16G (Ar)	641.63	12.907	$4p\ [1/2]_1$	14.839	$6s\ [3/2]^0_2$

16H (Ar)	643.16	13.095	$4p\ [5/2]_2$	15.022	$6s'\ [1/2]^0_1$
16I (Ar)	646.66	13.273	$4p\ [1/2]_0$	15.190	$5d\ [3/2]^0_1$
16J (Ar)	648.11	13.273	$4p\ [1/2]_0$	15.186	$7s\ [3/2]^0_1$
16K (Ar)	649.40	13.095	$4p\ [5/2]_2$	15.004	$4d'\ [3/2]^0_1$

Graphs 17: 650-675 nm

17A (Ar^+)	650.91	17.78	$^3P3d\ ^4F_{3/2}$	19.68	$^3P4p\ ^2D^0_{5/2}$
17B (Ar)	651.38	13.283	$4p'\ [3/2]_1$	15.186	$7s\ [3/2]^0_1$
17C (Ar)	653.81	13.076	$4p\ [5/2]_3$	14.972	$4d'\ [5/2]^0_3$
17α (He_2)	659.55	18.59	B $^1\Pi_g$	20.47	D $^1\Sigma^+_u$
17D (Ar)	659.61	13.076	$4p\ [5/2]_3$	14.955	$4d'\ [5/2]^0_2$
17E (Ar)	659.87	13.302	$4p'\ [3/2]_2$	15.181	$7s\ [3/2]^0_2$
17F (Ar)	660.40	13.076	$4p\ [5/2]_3$	14.953	$4d'\ [3/2]^0_2$
17G (Ar)	660.49	13.095	$4p\ [5/2]_2$	14.972	$4d'\ [5/2]^0_3$
17H (Ar)	663.21	13.153	$4p\ [3/2]_1$	15.022	$6s'\ [1/2]^0_1$
17I (Ar^+)	663.82	17.74	$^3P3d\ ^4F_{5/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
17J (Ar^+)	663.97	17.78	$^3P3d\ ^4F_{3/2}$	19.64	$^3P4p\ ^4D^0_{1/2}$
17K (Ar^+)	664.37	17.63	$^3P3d\ ^4F_{9/2}$	19.49	$^3P4p\ ^4D^0_{7/2}$
17L (Ar)	665.69	13.328	$4p'\ [1/2]_1$	15.190	$5d\ [3/2]^0_1$
17M (Ar)	666.07	13.153	$4p\ [3/2]_1$	15.014	$6s'\ [1/2]^0_0$
17N (Ar)	666.41	13.095	$4p\ [5/2]_2$	14.955	$4d'\ [5/2]^0_2$
17O (Ar)	667.73	11.624	$4s\ [3/2]^0_1$	13.480	$4p'\ [1/2]_0$
17P (He)	667.82	21.218	$2p\ ^1P^0_1$	23.074	$3d\ ^1D_2$
17Q (Ar^+)	668.43	17.69	$^3P3d\ ^4F_{7/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
17R (Ar)	668.48	13.283	$4p'\ [3/2]_1$	15.137	$5d\ [3/2]^0_2$
17S (Ar)	669.85	13.153	$4p\ [3/2]_1$	15.004	$4d'\ [3/2]^0_1$
17T (Ar)	669.89	13.172	$4p\ [3/2]_2$	15.022	$6s'\ [1/2]^0_1$

17U (Ar)	671.92	13.273	$4p\ [1/2]_0$	15.118	$5d\ [1/2]^0_1$
17V (Ar)	672.29	13.302	$4p'\ [3/2]_2$	15.146	$5d\ [7/2]^0_3$

Graphs 18: 675-700 nm

18A (Ar)	675.28	12.907	$4p\ [1/2]_1$	14.743	$4d\ [3/2]^0_2$
18B (Ar)	675.44	13.283	$4p'\ [3/2]_1$	15.118	$5d\ [1/2]^0_1$
18C (Ar)	675.62	13.302	$4p'\ [3/2]_2$	15.137	$5d\ [3/2]^0_2$
18D (Ar^+)	675.66	17.78	$^3P3d\ ^4F_{3/2}$	19.61	$^3P4p\ ^4D^0_{3/2}$
18E (Ar)	676.66	13.172	$4p\ [3/2]_2$	15.004	$4d'\ [3/2]^0_1$
18F (Ar)	677.99	13.480	$4p'\ [1/2]_0$	15.308	$6d\ [1/2]^0_1$
18G (Ar^+)	680.85	17.94	$^3P3d\ ^2P_{1/2}$	19.76	$^3P4p\ ^2D^0_{3/2}$
18H (Ar)	681.83	13.283	$4p'\ [3/2]_1$	15.101	$5d\ [1/2]^0_0$
18I (Ar)	682.72	13.302	$4p'\ [3/2]_2$	15.118	$5d\ [1/2]^0_1$
18J (Ar)	685.19	13.328	$4p'\ [1/2]_1$	15.137	$5d\ [3/2]^0_2$
18K (Ar^+)	686.35	17.74	$^3P3d\ ^4F_{5/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
18L (Ar)	687.13	12.907	$4p\ [1/2]_1$	14.711	$4d\ [1/2]^0_1$
18M (Ar)	687.96	13.153	$4p\ [3/2]_1$	14.955	$4d'\ [5/2]^0_2$
18N (Ar^+)	688.66	17.69	$^3P3d\ ^4F_{7/2}$	19.49	$^3P4p\ ^4D^0_{7/2}$
18O (Ar)	688.71	13.172	$4p\ [3/2]_2$	14.972	$4d'\ [5/2]^0_3$
18P (Ar)	688.82	13.153	$4p\ [3/2]_1$	14.953	$4d'\ [3/2]^0_2$
18Q (Ar)	692.50	13.328	$4p'\ [1/2]_1$	15.118	$5d\ [1/2]^0_1$
18R (Ar)	693.77	12.907	$4p\ [1/2]_1$	14.694	$4d\ [1/2]^0_0$
18S (Ar)	695.15	13.172	$4p\ [3/2]_2$	14.955	$4d'\ [5/2]^0_2$
18T (Ar)	696.03	13.172	$4p\ [3/2]_2$	14.953	$4d'\ [3/2]^0_2$
18U (Ar)	696.54	11.548	$4s\ [3/2]^0_2$	13.328	$4p'\ [1/2]_1$
18V (Ar^+)	699.01	17.78	$^3P3d\ ^4F_{3/2}$	19.55	$^3P4p\ ^4D^0_{5/2}$
18W (Ar)	699.22	13.328	$4p'\ [1/2]_1$	15.101	$5d\ [1/2]^0_0$

Graphs 19: 700-725 nm

19A (Ar)	703.03	13.076	$4p\ [5/2]_3$	14.839	$6s\ [3/2]^0_2$
19α (He_2)	705.02	19.52	C $^1\Sigma^+_g$	21.38	J $^1\Delta_u$
19B (He)	706.53	20.964	$2p\ ^3P^0_{0,1,2}$	22.719	$3s\ ^3S_1$
19C (Ar)	706.72	11.548	$4s\ [3/2]^0_2$	13.302	$4p'\ [3/2]_2$
19D (Ar)	706.87	13.095	$4p\ [5/2]_2$	14.848	$6s\ [3/2]^0_1$
19E (Ar^+)	707.70	17.74	$^3P3d\ ^4F_{5/2}$	19.49	$^3P4p\ ^4D^0_{7/2}$
19F (Ar)	708.67	13.273	$4p\ [1/2]_0$	15.022	$6s'\ [1/2]^0_1$
19G (Ar)	710.75	13.095	$4p\ [5/2]_2$	14.839	$6s\ [3/2]^0_2$
19β (He_2)	711.32	19.52	C $^1\Sigma^+_g$	21.36	J $^1\Pi_u$
19H (Ar)	712.58	13.283	$4p'\ [3/2]_1$	15.022	$6s'\ [1/2]^0_1$
19I (Ar)	714.70	11.548	$4s\ [3/2]^0_2$	13.283	$4p'\ [3/2]_1$
19χ (He_2)	714.78	19.52	C $^1\Sigma^+_g$	21.35	J $^1\Sigma^+_u$
19J (Ar)	715.88	13.283	$4p'\ [3/2]_1$	15.014	$6s'\ [1/2]^0_0$
19K (Ar)	716.26	13.273	$4p\ [1/2]_0$	15.004	$4d'\ [3/2]^0_1$
19L (Ar)	720.70	13.302	$4p'\ [3/2]_2$	15.022	$6s'\ [1/2]^0_1$
19M (Ar)	722.99	13.095	$4p\ [5/2]_2$	14.809	$4d\ [5/2]^0_2$

Graphs 20: 725-750 nm

20A (Ar)	726.52	13.153	$4p\ [3/2]_1$	14.859	$4d\ [3/2]^0_1$
20B (Ar)	727.07	13.076	$4p\ [5/2]_3$	14.781	$4d\ [7/2]^0_3$
20C (Ar)	727.29	11.624	$4s\ [3/2]^0_1$	13.328	$4p'\ [1/2]_1$
20D (He)	728.14	21.218	$2p\ ^1P^0_1$	22.920	$3s\ ^1S_0$
20E (Ar)	728.54	13.302	$4p'\ [3/2]_2$	15.004	$4d'\ [3/2]^0_1$
20α (He_2)	728.89	19.52	C $^1\Sigma^+_g$	21.32	H $^1\Sigma^+_u$
20F (Ar)	731.17	13.153	$4p\ [3/2]_1$	14.848	$6s\ [3/2]^0_1$
20G (Ar)	731.60	13.328	$4p'\ [1/2]_1$	15.022	$6s'\ [1/2]^0_1$

20H (Ar)	735.08	13.328	$4p'\ [1/2]_1$	15.014	$6s'\ [1/2]^0_0$
20I (Ar)	735.32	13.153	$4p\ [3/2]_1$	14.839	$6s\ [3/2]^0_2$
20J (Ar)	735.33	13.095	$4p\ [5/2]_2$	14.781	$4d\ [7/2]^0_3$
20K (Ar)	737.21	13.076	$4p\ [5/2]_3$	14.757	$4d\ [7/2]^0_4$
20L (Ar)	738.40	11.624	$4s\ [3/2]^0_1$	13.302	$4p'\ [3/2]_2$
20M (Ar)	739.30	13.172	$4p\ [3/2]_2$	14.848	$6s\ [3/2]^0_1$
20N (Ar)	741.23	13.283	$4p'\ [3/2]_1$	14.955	$4d'\ [5/2]^0_2$
20O (Ar)	742.23	13.283	$4p'\ [3/2]_1$	14.953	$4d'\ [3/2]^0_2$
20P (Ar)	742.53	13.302	$4p'\ [3/2]_2$	14.972	$4d'\ [5/2]^0_3$
20Q (Ar)	743.54	13.172	$4p\ [3/2]_2$	14.839	$6s\ [3/2]^0_2$
20R (Ar)	743.63	13.076	$4p\ [5/2]_3$	14.743	$4d\ [3/2]^0_2$
20S (Ar)	747.12	11.624	$4s\ [3/2]^0_1$	13.283	$4p'\ [3/2]_1$
20T (Ar)	748.43	13.153	$4p\ [3/2]_1$	14.809	$4d\ [5/2]^0_2$

APPENDIX C

THE RVESIM PROGRAM

The pages that follow contain:

a A minimal set of instructions for compiling and running the program.

b Instructions for creating the input files, with samples.

c A formatted printout of the program.

A text–only, compilable version of the program and a set of sample input files are available in electronic versions of this document by clicking on the hyperlinks below. The documents linked here are "works in progress." Their appearance will reflect that status, but they can be used to run simulations. The hyperlinked files are:

- A plain–text version of the program
- A set of sample input files: main, configuration, state, transition, Franck–Condon factor (4 files: FCF1, FCF2, FCF3, FCF4) and atomic
- A "tar–ball" (archive made with the program "tar") containing all of the files.

The program will probably be modified after this document is complete. More recent versions of the program should be accessible by searching the internet on "RVESIM" (as of this date, this term is not used on the internet in any other context). In addition to being more up–to–date, the documentation at the internet location will be more extensive than what is contained here. At present, the universal resource locator (URL) for the most recent version of the program is:

- http://cathubodva.agnesscott.edu/rvesim/

The site can be accessed from electronic versions of this document by clicking on the URL for the site.

C-a Minimal Instructions for Compilation and Execution

C-a.1 System requirements

- The program will run under Linux. It will probably work for most UNIX and UNIX–like systems (e. g., Linux). It should work under other operating systems with minimal changes to the source code: editing the strings used to create directories and files so that they are compatible with the operating system might be sufficient.
- The program requires a compiler for the computer language C.
- The amount of disk space and random–access memory (RAM) required will, of course, depend on the size of the simulation. To run the sample simulation (hyperlinked files available in electronic versions of the document only) used about 4 megabytes of RAM and required a total of about 10 megabytes of disk space (including source code, executable file and all input and output files).
- The program was written and tested under Linux, kernel 2.2.13, Slackware 7.0.0 distribution, using gcc version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release).

C-a.2 Compiling the program

- You must compile the program. Using the gcc compiler, the command is:

```
gcc RVESIM.c -lm -Wall -o RVESIM
```

If all goes well, you will probably see this warning:

```
RVESIM.c:  In function ‘new_mol_ck’:
RVESIM.c:8666:  warning:  function returns address of local variable
```

Do not be concerned by the warning; the return of the local variable's address will not cause problems.

C-a.3 Running the program

- Make sure that the input files are in the proper format and that they are in the directory from which the program is called. Then, use the following command:

```
RVESIM input_file
```

where "`input_file`" is the name of your main input file (see below).

– As the program runs, you will probably see statements such as:

```
mkdir:  cannot make directory ‘sample_molecules/NH’: File exists
mkdir:  cannot make directory sample_simulations’:  File exists
rm:  cannot remove ‘.rvesim_temp’:  No such file or directory
```

These statements are expected and do not indicate a problem.

C-b Input and Output Files

See Section 6-3 for a description of the usage of the input files. Only the formats for the files will be included here. The files are listed in the order in which they are accessed by the program.

C-b.1 The main input file:

The main input file contains a list of keywords and values. The keywords are required as a form of safety–check: the program checks that the input file is in the expected form and will inform the user (and stop running) if that form is not followed. Each keyword and its corresponding entry may be entered on a separate line or not, as suits the user; "whitespaces" (tabs, new–line characters and spaces) are ignored by the program. A sample main input file is given in Figure C-b.1. The required entries are listed below in the following format:

KEYWORD *acceptable value(s)* A brief description of the entry.

OUTPREF *string that does not contain spaces and contains only characters used in filenames with a maximum length of 100 characters* This is the prefix for the names of most output files and all first–level directories created by the program.

UNITS *"cm" or "nm"* Enter "nm" for nanometers or "cm" for wavenumbers. Note: this unit choice applies to all output from the program and to the keywords RES-

Figure C-b.1

```
OUTPREF         sample
UNITS           nm
RESOLUTION      0.5
EFFICIENCY      1
DETECTION       p
SCAN            s
CUTOFF          0.001
INTERACTIVE     n
MINPOINT        400
MAXPOINT        600
POINTSTEP       0.1
MAXIMUM         1.1e6
ROVIBSIM        Y         state_file.txt        transition_file.txt
NUMAT           3         Ar-Lines.txt 0.5
                          He-Lines.txt 1.0
                          ArII-Lines.txt 0.8
NUMOT           0
NUMEX           0
```

Figure C-b.1: A sample main input file.

OLUTION, MINPOINT, MAXPOINT and POINTSTEP. Caution: the program has only been thoroughly tested for nanometers.

RESOLUTION *positive number* This is the base, in the unit specified for "UNITS," of a triangle describing a single atomic emission in the experimental spectrum being simulated.

EFFICIENCY *"1"* Currently, the only acceptable value for this keyword is "1" (one).

DETECTION *"p" or "c"* This entry specifies the detection method for the experiment being simulated: enter "p" for photon–counting or "c" for calorimetric.

SCAN *"s" or "j"* This entry specifies the manner in which data points are taken in the experiment being simulated: enter "s" (scan) for systems that collect data continuously while scanning a series of wavelengths; enter "j" (jump) for systems

that take data while stopped at one wavelength then move to the next wavelength to collect more data and so on.

CUTOFF *number between 0 and 1* This is the low–intensity cut–off fraction. When the program calculates a rotational state distribution, it will ignore states with relative populations less than this number (relative to the most populated rotational level for that vibrational state). The program will also ignore vibration → vibration transitions with Franck–Condon factors less than this number.

INTERACTIVE *"n"* Currently, the only acceptable value for this keyword is "n" (for "no").

MINPOINT *positive number* The lowest wavelength or frequency to include in your simulation. Be sure to use the units specified for "UNITS."

MAXPOINT *positive number* The highest wavelength or frequency to include in your simulation. Be sure to use the units specified for "UNITS."

POINTSTEP *positive number less than the difference between MINPOINT and MAXPOINT* This is the size of the steps between data points in the experiment being simulated. Ideally, this number is at least a third of RESOLUTION, as defined above, but should be chosen to match the experimental conditions.

MAXIMUM *any number* This number scales the overall intensity of the final simulation: the maximum intensity in the simulated spectrum will be this value. This number can be negative if negative values for intensity are desired.

ROVIBSIM *"Y" or "N"* Enter "Y" if program should perform a rovibronic simulation and "N" if it should not (this is useful, for example, if only atomic transitions will be included). If "Y" is entered, also include, with no additional keywords, the names of the state file and the transition file (in that order). These filenames cannot contain spaces.

NUMAT *zero or a positive integer* The number of atomic files to be included. For each atomic file, include, with no additional keywords, the name of each atomic

Figure C-b.2

```
PROGRAM_NAME    RVESIM
DEBUG           -1
mdebugflag      0
tdebugflag      0
mrdebugflag     0
mnadebugflag    0
mnbdebugflag    0
msdebugflag     0
sdebugflag      0
mtdebugflag     0
cacdebugflag    0
cbcdebugflag    0
aadebugflag     0
bbdebugflag     0
bbSSdebugflag   0
bbSOdebugflag   0
bbOOdebugflag   0
ddebugflag      0
rpdebugflag     0
radebugflag     0
rddebugflag     0
XYdebugflag     0
HLswitch        0
bbSSSATT        0
bbSOSATT        0
bbOOSATT        0
```

Figure C-b.2: Sample configuration file (.rvesimconfig).

file immediately followed by the relative weight for transitions in that file. These filenames cannot contain spaces.

NUMOT *"0"* Currently, the only acceptable value for this keyword is "0" (zero).

NUMEX *"0"* Currently, the only acceptable value for this keyword is "0" (zero).

C-b.2 The configuration file:

Keywords must be entered exactly as they appear here and in the order given here, but need not each be on a separate line. The name of this file must be ".rvesimconfig"

(the leading period is required). A sample configuration file is given in Figure C-b.2. The format for the descriptions below is:

KEYWORD *appropriate values* `default value` Use of value in program

PROGRAM_NAME *String with no spaces containing a maximum of 199 characters*

`RVESIM` This is the name the program calls itself in many of the output files created by the program. These files begin with some variation on the comment "File generated by PROGRAM_NAME." The name does not have to be the same as the name given to the program's executable file. This name can be changed, for example, to identify a user, e. g., "BLF_RVESIM."

The following 21 entries are intended for debugging purposes and can cause enormous files (gigabytes large) to be written to the disk. The casual user is encouraged to enter the values as they appear here.

DEBUG -1	mdebugflag 0	tdebugflag 0
mrdebugflag 0	mnadebugflag 0	mnbdebugflag 0
msdebugflag 0	sdebugflag 0	mtdebugflag 0
cacdebugflag 0	cbcdebugflag 0	aadebugflag 0
bbdebugflag 0	bbSSdebugflag 0	bbSOdebugflag 0
bbOOdebugflag 0	ddebugflag 0	rpdebugflag 0
radebugflag 0	rddebugflag 0	XYdebugflag 0

The remaining entries allow the user to alter certain calculations involving transition intensities. The default values follow the rules (such as they exist). Only a small number of such modifications are available in the current version of the program.

HLswitch *0 or 1* `0` Switch to turn off the use of Hönl–London factors. If set to 0 (zero), Hönl–London factors are used; if set to 1 (one), they are not used.

The next three entries specify the extent to which satellite bands are to be included in the simulation. Ideally, satellite intensities should vary with K or J, but at present, they are all included with equal weight. This modification is only available for transitions between two states described by Hund's Case (b).

bbSSSATT *number between 0 and 1* `0` Weight for satellite bands between two Σ electronic states that are both Hund's Case (b).

bbSOSATT *number between 0 and 1* `0` Weight for satellite bands between one Σ and one non-Σ electronic state; both are Hund's Case (b).

bbOOSATT *number between 0 and 1* `0` Weight for satellite bands between two non-Σ electronic states that are both Hund's Case (b).

C-b.3 The state file

This file may contain more entries than are used in any given simulation, allowing one state file to be prepared and used repeatedly. But, each molecule/state pair must be unique. Lines that do not contain data may be included in the file (for comments). Each data–containing line must begin with an entry for MOLECULE and STATE separated by a single space, only. All data for a given state must be contained in one line. Keywords ("MOLECULE," "WEXE," etc.) should not be included. Each line beginning with a molecule/state pair should contain:

MOLECULE STATE CASE LAMBDA (p) MULTIPLICITY

HOMONUCLEAR? (g) (i) TE WE WEXE BE AE REQ

A sample state file is given in Figure C-b.3. Descriptions of each entry are listed below in the following format:

```
All information contained in this file from:  Reference Data on
Atoms, Molecules, and Ions by Radzig and Smirnov.
Springer-Verlag.  1985.  Unles otherwise indicated
 Mol St   case Lamb p  Mult Hom? g  I Te      we      wexe  Be      ae       re

 CO A     b    1       1    N         65075.8 1518.2  19.4  1.6115  0.0232   1.235
 CO a     b    1       3    N         48686.7 1743.4  14.4  1.6912  0.019    1.2057
 CO X     b    0    1  1    N         0       2169.81 13.29 1.93128 0.0175   1.1283

 N2 B     b    1       3    Y    1  1 59618.7 1733.98 14.39 1.6379  0.0181   1.212
 N2 A     b    0    1  3    Y    -1 1 50203.6 1460.6  13.87 1.455   0.018    1.287

 NH c     b    1       1    N         43740   2551    214   14.54   0.593    1.111
 NH A     b    1       3    N         29807   3231    99    16.674  0.745    1.037
 NH a     b    2       1    N         12570   3320    70    16.44   0.66     1.034
 NH X     b    0    -1 3    N         0       3282    78.3  16.7    0.649    1.036

 OH A     b    0    1  2    N         32684   3178.9  92.9  17.36   0.00079  1.012
 OH X-0.5 b    1       2    N         126.2   3737.8  84.9  18.91   0.000724 0.97
 OH X-1.5 b    2       2    N         0       3737.8  84.9  18.91   0.000724 0.97
```

Figure C-b.3: Sample state file. Hund's cases are assigned the closest value currently usable by the program.

ENTRY_NAME *acceptable values* Brief description of entry.

MOLECULE *string that does not contain spaces and with a maximum length of 39 characters* This is the name of the molecule. It must match exactly corresponding entries in the transition file.

STATE *string that does not contain spaces and with a maximum length of 20 characters* This is the name of the electronic state. It must match exactly corresponding entries in the transition file.

CASE *"a" or "b"* Hund's Case for this state. Note that transitions between different Hund's Cases are not yet treated by the program. For example, if one state in a transition is a Σ state then both states must be specified as Case (b) (regardless whether this is the truth).

LAMBDA *zero or positive integer* This is the total orbital angular momentum quantum number for the electronic state (Σ states have $\Lambda = 0$, for example).

(p) *none, "1" or "-1"* If the entry for LAMBDA is zero, then the symmetry of the wavefunction upon reflection through the internuclear axis must be specified. In typical notation, this is the "+" or "-" superscript for Σ electronic states. If the entry for LAMBDA is non–zero, this field should be left blank (a tab or other whitespace, except new–line, is acceptable).

MULTIPLICITY *positive integer* This is $2S + 1$ where S is the vector–sum of all electron spins in the molecule.

HOMONUCLEAR? *"Y" or "N"* Enter "Y" if the molecule is homonuclear and "N" if not.

(g) *none, "1" or "-1"* If the entry for HOMONUCLEAR? is "Y," then the inversion symmetry of the wavefunction must be specified. In typical notation, this is the "g" or "u" subscript given to electronic states for homonuclear molecules. If the entry for HOMONUCLEAR? is "N," this field should be left blank (a tab or other

whitespace, except new–line, is acceptable). "1" indicates a symmetric (gerade) wavefunction; "-1" indicates an antisymmetric (ungerade) wavefunction.

(i) *none or decimal number (no fractions)* If the entry for HOMONUCLEAR? is "Y," then the nuclear spin of the atoms must be specified. This is the spin for only one of the atoms (do not attempt to take a vector sum of both). If the entry for HOMONUCLEAR? is "N," this field should be left blank (a tab or other whitespace, except new–line, is acceptable). Enter the nuclear spin as a decimal; for example, enter 1.5 rather than 3/2.

TE *zero or positive number* This is the term energy for the electronic state in wavenumbers.

WE *positive number* This is the fundamental vibration frequency for the electronic state in wavenumbers.

WEXE *positive number* This is the vibration anharmonicity for the electronic state in wavenumbers.

BE *positive number* This is the rotational constant for the electronic state in wavenumbers.

AE *positive number* This is the vibration–rotation interaction constant for the electronic state in wavenumbers.

REQ *positive number* This is the equilibrium bond length for the molecule. Currently, this number is not used in the program.

C-b.4 The transition file

The structure of the transition file is somewhat complex, but the file is easy to read and change once the structure is learned. White–spaces are ignored, so the user is free to indent or merge and break lines as desired in order to make the file easier to read. The structure of the file is illustrated and a sample file is given in Figure C-b.4. The required entries are described below using the format:

Figure C-b.4a

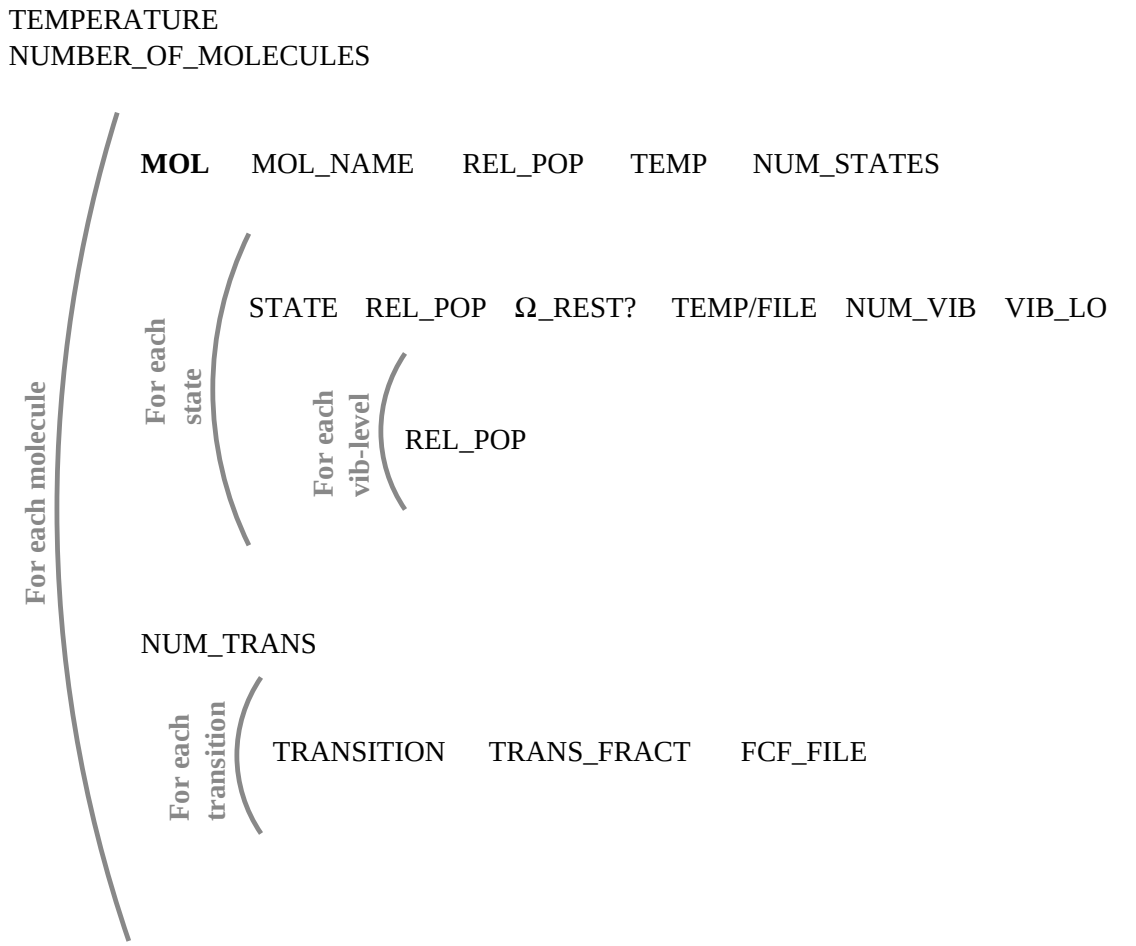


Figure C-b.4a: Graphical illustration of the structure of the transition file. See text for a description of the file and Figure C-b.4b for a sample.

ENTRY_NAME *acceptable values* Brief description of entry.

TEMPERATURE *zero or positive number, degrees Kelvin* This number represents the default rotational temperature for any species in the simulation. Temperatures for molecules or states that are specified later override this entry. This number should be non–zero unless either a temperature is specified at either the molecule or the state level for each state entered in the file. The program will not produce a meaningful simulation if the temperature is set to zero.

Figure C-b.4b

```
300
3
MOL         CO      1               300     3

a           1.00    N               0       2     0     1.0     .6
A           .2      N               800     3     5     .92     1       .3
X           0.00    N               0       1     0     1.0

2

a-X         1       __CO_sa_X.dat
A-X         1       __CO_A_X.dat

MOL         OH      .06             1000    3

A           1.0     N               0       3     0     .7      1.0     .2
X-0.5       0       N               0       1     0     1.0
X-1.5       0       N               0       1     0     1.0

2

A-X-1.5     1       __OH_A_X.dat
A-X-0.5     1       __OH_A_X.dat


MOL         NH      .35             1000    2
c           1       N               0       1     0     1
a           0       N               0       1     0     1

1

c-a         1       __NH_c_a.dat
```

Figure C-b.4b: Sample transition file.

NUMBER_OF_MOLECULES *positive integer* This is the number of molecules to be included in the simulation, and it must be less than or equal to the number of molecules in the file. If there are more molecules, the program will read the file until it has read the specified number of molecules, then stop. The program will terminate if the file ends before this number of molecules is read.

MOL *"MOL"* This is a safety–check. At the beginning of the set of information for each molecule, type "MOL" (exactly like that, without quotation marks and in all–capital letters). The program continually scans for this entry. If the entry is found before a molecule is finished, the program will write a descriptive error message and terminate.

MOL_NAME *string that does not include spaces and has a maximum length of 39 characters* This string must match exactly the corresponding entry for MOLECULE in the state file.

REL–POP *zero or positive number* This is the relative population for this molecule. Ideally, it should be a number between 0 and 1, numbers greater than one are permissible.

TEMP *zero or positive number, degrees Kelvin* This is the default rotational temperature assigned to this molecule. If this entry is zero, then the previously defined temperature is the default.

NUM_STATES *positive integer* This is the number of states to be considered for the current molecule. This number must exactly match the number of states described below.

STATE *string that does not include spaces and has a maximum length of 20 characters* This is the name given to the electronic state to be considered. This string must match exactly the corresponding entry in the state file.

REL–POP *zero or positive number* This is the relative population for this state.

Ω–REST? *for this description "N"* The program is written so that electronic states with multiple possible values of Ω (the total angular momentum quantum number, including S, the electronic spin vector) can be treated as separate states. But, the description of this feature is complex, and it has not been thoroughly tested, so the only value for this entry that will be considered here is "N."

TEMP/FILE *zero, positive number (degrees, Kelvin) or file–name* This is the rotational temperature to use for this state. If this number is zero, then the temperature specified for the molecule is used. If the temperature for the molecule is also zero, then the default temperature for the simulation is used. If all three numbers are zero, the program will not produce a meaningful simulation. If a file name is entered here, the program will attempt to read a distribution of rotational populations from the indicated file. Caution: reading rotational distributions from a file has been only minimally tested. A description of the contents of the rotational distribution file will not be included here; details of its structure can be determined by inspecting the source code (Section C-c).

NUM_VIB *positive integer* This is the total number of vibrational levels to consider for this state. It is not permissible to skip vibrational levels in the transition file. To omit a vibrational level in the middle of a range of levels, assign that level a relative population of zero (see REL_POP, below).

VIB_LO *zero or positive integer* This is the lowest vibrational level to consider.

REL_POP *zero or positive number* The number of REL_POP entries must match exactly the number given for NUM_VIB. Each REL_POP describes the relative population of a vibrational level, beginning with the level specified in VIB_LO.

NUMTRANS *positive integer* The number of transitions to be considered in the simulation. This number must match exactly the number of transitions listed below.

TRANSITION *valid string representing a STATE for the current MOLECULE followed by a single dash and the followed by a valid string representing some other STATE for the current MOLECULE; both states must be properly entered in the state file* The transitions listed here must involve states listed for this molecule. If a requested transition contains a state not already described (or not

described for the current molecule), the program will issue a message similar to the following and terminate:

```
Missing definition for at least one state in transition b-X. Exiting.
Please edit transition file and restart program
```

TRANS–FRACT *zero or positive number* This number describes the fraction of the upper state whose emissions are observed. For example, if an upper state has a radiative lifetime of 8 ms, and data will be collected for the first 2 ms after the state is produced, then only about 0.22 of the states present will have radiated during the data–collection period. In this case, the entry here should be 0.22. See Chapter 5 for a more detailed discussion of radiative lifetimes and fraction of emissions observed.

FCF–FILE *name of the Franck–Condon file for the transition (maximum of 200 characters)* This string must exactly match the name of the file containing Franck–Condon factors for this transition. See below for the structure of the Franck–Condon factor file.

C-b.5 Franck–Condon factor files

The file containing Franck–Condon factors can contain lines that do not contain data (for comments, etc.). Lines that contain data must begin with an integer followed by a single space, a dash and a greater–than sign (>); leading whitespace is permissible in these lines. If other lines in this file contain this combination of characters, the program might behave unexpectedly. A sample Franck–Condon factor file is given in Figure C-b.5. The format for a line containing data is:

VLO -> VHI , WAVENUMBERS , FCF , NM , PEC

Only VLO, VHI and FCF are used by the program. The entries are described below using the format:

Figure C-b.5

```
Morse potential parameters:
De = 10.981 eV
alpha = 2.3256 /Angst

Morse potential parameters:
De = 3.6827 eV
alpha = 2.8098 /Angst
Beta:  2.32562416 2.80981623 /Angst
K: 163.266371 78.257731
Re:  1.12829995 1.23500001 Angst
Nmax = 19 19
      0  ->   0  ,  64748.47  ,  0.11172      ,  154.44  ,  3.03273E+13
      0  ->   1  ,  66227.87  ,  0.21104      ,  150.99  ,  6.13047E+13
      0  ->   2  ,  67668.47  ,  0.22453      ,  147.78  ,  6.95717E+13
      0  ->   3  ,  69070.27  ,  0.17888      ,  144.78  ,  5.89430E+13
      0  ->   4  ,  70433.27  ,  0.11972      ,  141.98  ,  4.18321E+13
      0  ->   5  ,  71757.47  ,  7.16097E-02  ,  139.36  ,  2.64590E+13
      0  ->   6  ,  73042.87  ,  3.97653E-02  ,  136.91  ,  1.54967E+13
      0  ->   7  ,  74289.47  ,  2.10304E-02  ,  134.61  ,  8.62243E+12
      0  ->   8  ,  75497.27  ,  1.07845E-02  ,  132.46  ,  4.64081E+12
      0  ->   9  ,  76666.27  ,  5.43310E-03  ,  130.44  ,  2.44828E+12
      0  ->  10  ,  77796.47  ,  2.71544E-03  ,  128.54  ,  1.27856E+12
      0  ->  11  ,  78887.87  ,  1.35644E-03  ,  126.76  ,  6.65936E+11
      0  ->  12  ,  79940.47  ,  6.81096E-04  ,  125.09  ,  3.47943E+11
      0  ->  13  ,  80954.27  ,  3.45278E-04  ,  123.53  ,  1.83184E+11
```

Figure C-b.5: Portion of a Franck–Condon factor file. This file was generated using the modified version of Kent Ervin's program [Ervin 1993] that is used with CASI (computer assisted spectral identification, see Section 3-2).

ENTRY_NAME *acceptable values* Brief description of entry.

VLO *zero or positive integer* This is the vibrational quantum number for the lower electronic state.

-> "->" This combination of characters must be entered between vibrational quantum numbers. Spaces must surround this character combination.

VHI *zero or positive integer* This is the vibrational quantum number for the higher electronic state.

Figure C-b.6

```
3554.413    3.64E-04   .8
3554.413    3.27E-03   .8
3554.413    1.31E-02   .8
3554.422    5.46E-03   .8
3554.422    9.83E-03   .8
3554.547    7.28E-03   .8

3587.268    5.69E-04   .5
3587.268    5.13E-03   .5
3587.268    2.05E-02   .5
3587.278    8.54E-03   .5
3587.278    1.54E-02   .5
3587.405    1.14E-02   .5

3634.2308   7.25E-04   .7
3634.2311   2.61E-02   .7
3634.2311   6.52E-03   .7
3634.2409   1.09E-02   .7
3634.2412   1.96E-02   .7
3634.3714   1.45E-02   .7
```

Figure C-b.6: Sample atomic file. Here, transitions from a common upper electronic state (of helium) are grouped together and assigned a common relative population (RELATIVE-WEIGHT).

, *","* Commas must appear, separated by spaces, in the locations indicated above.

WAVENUMBERS *any positive number or zero* This value is not used by the program, but a number must be entered here. The recommended value is either the frequency, in wavenumbers, for this transition, or zero. If this number is negative, the program will assign a value of zero to the Franck–Condon factor for the current transition.

FCF *positive number between 0 and 1* This is the Franck–Condon factor for this transition. The program will continue to run if this number is larger than one, but the results will not have a predictable meaning.

NM *any number* The program ignores this entry, but it must be a number. The recommended value is either the wavelength, in nanometers, for the current transition, or zero.

PEC *any number* The program ignores this entry, but it must be a number. The recommended value is either a variation of the Einstein–A coefficient for the current transition, or zero.

C-b.6 Atomic files

All lines in atomic files must either be blank or contain only information for a single transition. Although the file is called "atomic," this file can be used to include any single–line transition. For example, if frequencies and intensities for a series of rotational transitions are available from an external source (experiment or calculation, for example), they can be entered using this file. The format for non–blank lines is:

TRANSITION–POSITION TRANSITION–PROBABILITY RELATIVE–WEIGHT

The TRANSITION–POSITION must be entered in Ångstroms. This unit was chosen because it is the wavelength unit used in many references for atomic transitions. The units used for the TRANSITION–PROBABILITY can be any that are convenient to the user, but should be uniform. This entry can be used in a manner similar to TRANS_FRACT in the transition file. If the transition probability is not useful in the simulation, set all values equal to one. The RELATIVE–WEIGHT is the relative population for that state.

C-c RVESIM.c

```
/****************** Headers ******************/
/* These are the headers for the RVESIM.c program */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define SUB (double)0.5
#define PI 3.14159265358979323846264338
#define JKm 0.0504507857956213  /* constant for an equation trick */
#define JKb 3.04614091081434  /* another constant for an equation
   trick */
#define kB 1.38066e-23/1.9864e-23  /* Boltzmann's constant in
   wavenumbers per kelvin. The first number is Boltzmann's constant
   in J/K (joules per kelvin). The second number is the conversion
   from joules to wavenumbers. */

/****************** Begin Section on States ******************/

typedef struct {
  char f[201];  /* filename */
  FILE *F;  /* file pointer */
  } fileset;

typedef struct{
  int vlo, vnum;  /* starting vibrational quantum number and number
   of quantum numbers to consider */
  int vclo, vcnum;  /* starting vib quant number for cascade popula-
   tion and number of quanta to consider -- the population will all
   be contained in the rotset, but need to know about it */
  double *p;  /* relative populations for each of the vnum levels */
  int *c;  /* change flag for the relative population: (0) not cal-
   culated, (1) calculated (-1) re-calculated or (-2) is awaiting
   recalculation. */
  } vset;

typedef struct{
  int c;  /* change flag for this set */
  int j,k,jc,kc;  /* number of J and/or K values in this set jc and kc
   are there because there might be more E entries than there are P
   entries -- there will also be more CJ/CK entries.*/
  double *J, *K;  /* values of J and/or K for each level */
  double Jdissoc,Kdissoc;  /* values of J/K where the molecule dis-
   sociates, according to the ratio of Bv to Dv. */
  double *EJ, *EK;  /* the energies of this level (wavenumbers) */
  double *PJ, *PK, Jmaxp;  /* relative populations of this level,
   not including any contribution from cascade and the value of J
   that was found to be the true maximum value -- 10 June 02: I just
   commented out all use of Jmaxp */
  double *CJ, *CK;  /* cascade population of this level */
  } rotset;
```

```
   /*** The 'State' structure ***/
30 typedef struct {
31   char Name[21]; /* The letter state designation (a', B, c, etc.).
      */
32   double pop; /* relative population for this state */
33   int cp, cC, cD, co, cT, cS, cr, cv; /* change flags for population
      (cp), Case (cC), presence of rotational input file (cD), number
      omegas (co), temperature (cT), spectroscopic information (cS),
      rotation (cr) and vibration (cv) info */
34   char Case[11]; /* Hund's Case. a or b */
35   int Ca; /* numerical position in stateinfo array if Case a */
36   int Cb; /* numerical position in stateinfo array if Case b */
37   int Cc, Cd; /* similar to Ca and Cb */
38   int Dist; /* is the distribution of states is from a file (-1),
      from the temperature (1), or not relevant (0). */
39   vset *v; /* pointer to set of vibrational information MEMORY AL-
      LOCATED in read_trans.c */
40   rotset *r; /* pointer to set of rotational distribution info
      MEMORY ALLOCATED in rotation-distribution functions (e.g.,
      multi_nonzero_JKb.c) */
41   int nr,nO,nV; /* number of rotsets, number Omegas and vnum (this
      will change if different Omegas get different vnums...) */
42   rotset *rc; /* pointer to rotsets for additional vibrational lev-
      els that are populated only by cascade -- the number of these sets
      is 30 (or Onum*30) because it's hard to know which levels will get
      populated from which states, and I'm too lazy to shuffle all the
      array entries. */
43   int pmsymm, Isymm; /* if Lambda==0: Are even numbered K's + or -
      (pmsymm); and, if this set is homonuclear, are even K's symmetric
      or antisymmetric (Isymm). */
44   int no; /* How many times is this state an origination state? */
45   int *o; /* the list of positions in the transition array where
      this state is the origination state. MEMORY ALLOCATED in
      read_trans.c */
46   int nd; /* How many times is this state a destination state? */
47   int *d; /* the list of positions in the transition array where
      this state is the destination state. MEMORY ALLOCATED in
      read_trans.c */
48   fileset *f; /* file name and pointer for J/k distribution file(s)
      if there are any for this state MEMORY ALLOCATED in read_trans.c
      */
49   double *T; /* Temperature of state. Leave zero if this is only a
      destination state or if temp is defined at a higher level. There
      is one of these for each omega. MEMORY ALLOCATED in read_trans.c
      */
50   } State;

   /*** The 'caseinfo' structures ***/
   /* For now, only cases (a) and (b) are defined here. */

51 typedef struct {
52   int cTe, cwe, cwexe, cBe, cae, cbeta, cO; /* change flags for the
      term energy (cTe), we and wexe (cwe, cwexe), the rotational con-
      stant (cBe), the vibration-rotation interaction constant (cae),
```

```
      the centrifugal distortion constant (cbeta), anything concerning
      Omegas (cO) */
53    double Te, we, wexe, Be, ae, beta; /* all units wavenumbers --
      beta currently not used in the program */
54    int L, O, g; /* Lambda (L). The number of Omegas to consider; use
      0 (zero) to have the program use all possible. If applicable (g),
      gerade (1), ungerade (-1); if not homonuclear, use 0 (zero). */
55    double *lO, I, S, sflag; /* if the user specifies certain Omegas
      to use, this is the list of desired Omegas (lO). It has to be
      float since I don't want to work out some other way to have half-
      integral values for Omega. Also, if homonuclear, the nuclear spin
      (I). The electronic spin, S, and a flag for looking up whether the
      spin is integral (=0) or half-integral (=0.5). MEMORY ALLOCATED
      in read_trans.c */
56    double *pO; /* If the user specifies populations for each Omega,
      they go here. MEMORY ALLOCATED in read_trans.c */
57    int symm; /* If homonuclear: are even-plus (+1) or even-minus (-
      1) (even J-values) symmetric (default value 0)? -- This variable
      will only be useful if someone includes the +/- splitting in the
      program one day -- SEE PROGRAM_CHANGES if you are the person want-
      ing to do that */
58    double Jmin, Jmax; /* Minimum and maximum values for J. */
59    double *Jpop; /* pointer to the array of J-state populations if
      they are entered from a file MEMORY ALLOCATED in read_rot.c */
60    int a; /* how many J-values long is the array? */
61    } Case_a_stateinfo;

62  typedef struct {
63    int cTe, cwe, cwexe, cBe, cae, cbeta; /* change flags for the term
      energy (cTe), we and wexe (cwe, cwexe), the rotational constant
      (cBe), the vibration-rotation interaction constant (cae), the
      centrifugal distortion constant (cbeta) */
64    double Te, we, wexe, Be, ae, beta; /* all units wavenumbers --
      beta is currently included in some calculations in the program,
      but since it isn't read in, the value is always zero. */
65    int L, p, g; /* Lambda (L). If L=0, indicate (p) plus (+1) or minus
      (-1); if L!=0, use zero (0). If homonuclear, indicate (g) gerade
      (+1), ungerade (-1); if not homonuclear, use zero (0). */
66    double I, S, sflag; /* If homonuclear, the nuclear spin. The elec-
      tronic spin, S and a flag for looking up whether the spin is inte-
      gral (=0) or half-integral (=0.5). */
67    /*int symm; [1] If L=0 and homonuclear: are even (+1) or odd (1)
      K-values symmetric? [2] If Lambda>0 and homonuclear are even-
      plus states (+1) or even-minus states (-1) symmetric? (default
      0) */
68    double Kmin, Kmax, Jnum, *K, *J; /* Minimum and maximum values for
      K, the number of J's, and the actual values of J and K */
69    double *Jpop; /* pointer to the array of J-state populations if
      they are entered from a file MEMORY ALLOCATED in read_rot.c */
70    int a; /* how many K-values long is the array? */
71    } Case_b_stateinfo;
```

```
    /***************** Begin Section on Transitions *****************/

    /*** General structures for simulation parts ***/
typedef struct{
  int n;  /* number of transitions here */
  double pop;  /* relative population for this set */
  double *f;  /* frequency of transition */
  double *ni;  /* native intensity at that transition */
  double *ci;  /* cascade intensity at that transition if needed */
  } simset;  /* set of information needed to make the simulation */
typedef struct{
  int n;  /* number of points */
  double *x;  /* position */
  double *y;  /* relative intensity */
  } sim;  /* a simulation */

    /*** A structure for Franck-Condon Factors ***/
typedef struct {
  double fcfn[30];  /* An array of frequency-adjusted franck-condon
   factors from one vibrational level to a series of lower levels,
   for observed transitions. */
  double fcfc[30];  /* An array of frequency-adjusted franck-condon
   factors from one vibrational level to a series of lower levels for
   transitions that contribute to cascade transitions. */
  double cm[30];  /* array of cm's for the origin of this transition
   */
  int vqn[30];  /* redundant, but just in case the vib quantum number
   isn't what's expected from the file -- these are the lower state
   quantum numbers. */
  } vfcf;

    /*** The 'Transition' structure ***/
typedef struct {
  char Nhi[21], Nlo[21];  /* The letter designations for the higher
   state (Nhi -- e.g., a', B, c, etc.), and the lower state (Nlo). */
  int Hi, Lo;  /* Numerical position in states array for info about
   the high state and the low state. */
  int Ohi,Olo;  /* Number of Omegas specified for the high and low
   states. */
  int vs,vnh,vnl,vhlo,vllo;  /* the actual number of v's found in
   the file, the number of high and low v's simulated and the lowest
   values for those sets of v's */
  int cv,*cP;  /* change flag for FCF info and transition probabil-
   ity MEMORY ALLOCATED in read_trans.c */
  vfcf *v;  /* pointer to the arrays of franck-condon factors. The
   declaration of 30 states is based on the maximum number output
   from Ervin's FCF program. These array spots are indexed by the
   upper vibrational q#. MEMORY ALLOCATED in read_FCF.c*/
  int nS, *fS;  /* number of simulations for this transition and
   flags to tell if the simulations should be included in the overall
   simulation (won't be if spin-Sigma forbidden) MEMORY ALLOCATED in
   transition functions (e.g., a-a.c)*/
  simset *S;  /* pointer to simulations for this transition MEMORY
   ALLOCATED in transition functions (e.g., a-a.c)*/
```

```
 99   double *P; /* overall electronic transition probability MEMORY
       ALLOCATED in read_trans.c */
100   fileset *f; /* filenames and pointers for FCF files MEMORY ALLO-
       CATED in read_trans.c */
101   } Trans;

     /* This is a structure made solely for the purpose of transmitting
       information between b_b() and the various sub-functions it calls
       */
102  typedef struct {
103   int hvnlo,hvnnum,hvclo,hvcnum,hivlo,hivnum; /* Re high state:
       native low vib num, number vib levels; same, but for cascade;
       overall lowest vib num and overall number vib levels */
104   int lvnlo,lvnnum,lvclo,lvcnum,lovlo,lovnum; /* like above, low
       state */
105   int nT; /* maximum number transitions per K level */
106   int nhJ,nlJ; /* 2S+1 for high and low states */
107   int hiK; /* maximum number hi K might be */
108   int m; /* position in MOL array */
109   Trans *T; /* pointer to this transition structure */
110   Case_b_stateinfo *Ch,*Cl; /* to high and low Case_b_info struc-
       tures */
111   rotset *Rh,*Rl; /* to high and low rotsets */
112   State *Sh,*Sl; /* to high and low State structures */
113   vset *Vh; /* high-state vib set */
114   } BBTinfo;

     /*** Included Atomic Transitions ***/
115  typedef struct {
116   double p; /* relative population for this file */
117   fileset f; /* name and pointer for this file */
118   double *x; /* position of the transition */
119   double *A; /* Einstein A */
120   double *pop; /* relative population for this transition */
121   int M; /* multiplicity for the origination state -- not used now
       */
122   simset S; /* the part of the simulation due to this set */
123   int c, n; /* change flag for this set of atomic transitions, num-
       ber of sets of information */
124   } Ainfo;

     /*** Included Transitions, 'Other' 'experimental', etc. ***/
125  typedef struct {
126   double p; /* relative population for this set */
127   double *x; /* position of transition */
128   double *y; /* relative intensity at that position */
129   int c, n; /* change flag for this set of data and number of x,y
       pairs in this set. */
130   fileset f; /* file where this info was found */
131   } XYlist;

     /* Structure for keeping up with molecule information */
132  typedef struct{
133   int cp, cT; /* change flags for relative population and tempera-
       ture */
```

```
134   char Mol[41]; /* name of molecule */
135   double pop; /* relative population for molecule */
136   int states ; /* number of states for this molecule */
137   State *s; /* state array MEMORY ALLOCATED in read_trans.c */
138   int trans; /* number of transitions for this molecule */
139   Trans *t; /* transition array MEMORY ALLOCATED in read_trans.c */
140   double T; /* temperature if specified at molecule level */
141   } Molecule;

    /* these are the names for files containing information about the
       run */
142 char PREF[101],TMPFILE[300];
143 FILE *INMAIN,*DBG,*OUT,*PAR,*SCR,*SYS;
144 fileset *INOT,*INV,*OTR,*INEX,*INAT,INST,INTR;

    /* these functions read in the various input files */
145 void read_state_and_transition_files(), read_FCF_file(),
    read_atomic_files();
146 void read_XY_file(XYlist*),read_rot_dist_files();
    /* these are managerial and graphical functions */
    /*void get_energies_intensities();*/
147 void manage_transitions(),manage_rotations();
148 void do_simulation(), graph_results(), ask_for_changes();
    /* these functions calculate spectra for different transition types
       */
149 void
    a_a(int,int),a_b(int,int),a_c(int,int),a_d(int,int),b_b(int,int);
150 void
    b_c(int,int),b_d(int,int),c_c(int,int),c_d(int,int),d_d(int,int);
    /* these functions calculate rotational levels and populations */
151 void singlet_JK(int,int),multiplet_zeroL_JK(int,int);
152 void
    multiplet_nonzeroL_JK_a(int,int),multiplet_nonzeroL_JK_b(int,int);
153 void case_a_cascade(rotset*,int,double,int,int,int);
154 void case_b_cascade(rotset*,int,int,int,int,int);
155 void
    bb_SigmaSigma(BBTinfo),bb_SigmaOther(BBTinfo),bb_Other(BBTinfo);
156 double HL(int,double,int,int);
157 char *new_mol_ck(char*,char*);

158 int DEBUG=0,NUMTR=0,NUMAT=0,NUMOT=0,NUMEX=0,NUMMOL=0,NUMST=0;
    /* A note on the DEBUG variable. If this is equal to -1, no functions
       will print any messages. Adjusting values in .rvesimconfig can
       cause messages about the execution of functions to print. Use
       this feature cautiously as it can produce enormous files. */
159 int Case_a_num=0,Case_b_num=0,SCANTYPE=0,DETECTTYPE=3,UNITTYPE=0;
160 int EFFTYPE=1,INTERACT=1,ROVIB=0;
161 double SIMMAX=1,*ATPOP,*OTINT,*EXINT,TEMP=0,JKCUT=0.001;
162 double MINV=100,MAXV=3500,PSTEP=0.5,MAXINT=5000,RESN=2.0;
163 Case_a_stateinfo *CA;
164 Case_b_stateinfo *CB;
165 Molecule *MOL;
166 Ainfo *AT;
167 XYlist *OT,*EX,*EF;
```

```
168 char PROGRAM_NAME[200]="RVESIM";
    /* debugging-level flags */
    /* Functions that affect many functions */
169 int mdebugflag=0; /* main */
170 int tdebugflag=0;  /* transition file */
    /* Functions for calculating rotational populations and energies */
171 int mrdebugflag=0;  /* manage rotations */
172 int mnadebugflag=0;  /* multiplet non-zero-L Case a */
173 int mnbdebugflag=0;  /* multiplet non-zero-L Case b */
174 int msdebugflag=0;  /* multiplet sigma */
175 int sdebugflag=0;  /* singlet sigma */
    /* Functions involved in calculating transitions */
176 int mtdebugflag=0;  /* manage transitions */
177 int aadebugflag=0;  /* Case a <-> case a transition */
178 int cacdebugflag=0;  /* calculate energy levels in destination Case
      a state */
179 int bbdebugflag=0;  /* manages case b <-> case b transitions */
180 int cbcdebugflag=0;  /* calculate energy levels in destination Case
      b state */
181 int bbSSdebugflag=0;  /* Case b Sigma<->Sigma transitions */
182 int bbSOdebugflag=0;  /* Case b Sigma<->Non-Sigma transitions */
183 int bbOOdebugflag=0;  /* Case b Non-Sigma<->Non-Sigma transitions
      */
    /* The function that turns lines into spectra */
184 int ddebugflag=0;  /* do_simulation */
    /* Functions that read contents of files */
185 int rpdebugflag=0;  /* read FCF files */
186 int radebugflag=0;  /* read atomic files */
187 int rddebugflag=0;  /* read rotational distribution files */
188 int XYdebugflag=0;  /* read files in x-y format */
189 int HLswitch=0;  /* use Honl-London factors? 1=no 0=yes */
190 double bbSSSATT=0,bbSOSATT=0,bbOOSATT=0;


    /************************** main ************************/

    /* main opens and reads the input file, assigning values to vari-
      ables as needed and handles other file opening/closing and func-
      tion calls */
191 int main(int argc, char *argv[]){

    /* generic counters */
192 int ma=0,mb=0,mc=0,mOnuma=0,mOnumb=0;
    /* strings for various things */
193 char mstrdum[1000],mconf[100]=".rvesimconfig",mtmp[200];
194 FILE *MCONF;
    /* for scaling the transition probabilites to one */
195 double mtrprobmax=0;

196 MCONF=fopen(mconf,"r");
197 if(MCONF==NULL){
198   printf("Configuration file .rvesimconfig not found.\n");
199   printf("Using internal defaults.\n");
200   }
```

```
201 else{
202   fscanf(MCONF,"%s %s",mstrdum,PROGRAM_NAME);
203   fscanf(MCONF,"%s %d",mstrdum,&DEBUG);
204   fscanf(MCONF,"%s %d",mstrdum,&mdebugflag);
205   fscanf(MCONF,"%s %d",mstrdum,&tdebugflag);
206   fscanf(MCONF,"%s %d",mstrdum,&mrdebugflag);
207   fscanf(MCONF,"%s %d",mstrdum,&mnadebugflag);
208   fscanf(MCONF,"%s %d",mstrdum,&mnbdebugflag);
209   fscanf(MCONF,"%s %d",mstrdum,&msdebugflag);
210   fscanf(MCONF,"%s %d",mstrdum,&sdebugflag);
211   fscanf(MCONF,"%s %d",mstrdum,&mtdebugflag);
212   fscanf(MCONF,"%s %d",mstrdum,&cacdebugflag);
213   fscanf(MCONF,"%s %d",mstrdum,&cbcdebugflag);
214   fscanf(MCONF,"%s %d",mstrdum,&aadebugflag);
215   fscanf(MCONF,"%s %d",mstrdum,&bbdebugflag);
216   fscanf(MCONF,"%s %d",mstrdum,&bbSSdebugflag);
217   fscanf(MCONF,"%s %d",mstrdum,&bbSOdebugflag);
218   fscanf(MCONF,"%s %d",mstrdum,&bbOOdebugflag);
219   fscanf(MCONF,"%s %d",mstrdum,&ddebugflag);
220   fscanf(MCONF,"%s %d",mstrdum,&rpdebugflag);
221   fscanf(MCONF,"%s %d",mstrdum,&radebugflag);
222   fscanf(MCONF,"%s %d",mstrdum,&rddebugflag);
223   fscanf(MCONF,"%s %d",mstrdum,&XYdebugflag);
224   fscanf(MCONF,"%s %d",mstrdum,&HLswitch);
225   fscanf(MCONF,"%s %lf",mstrdum,&bbSSSATT);
226   fscanf(MCONF,"%s %lf",mstrdum,&bbSOSATT);
227   fscanf(MCONF,"%s %lf",mstrdum,&bbOOSATT);
228   }

    /* inform user of usage if there is no input file on the command line
       */
229 if(argv[1]==NULL){
230   printf("Usage: %s input_file\n",PROGRAM_NAME);
231   printf("See the documentation files for more info.\n");
232   exit(1);
233   }
    /* otherwise, open the file from the command line */
234 else{
235   INMAIN=fopen(argv[1],"r");
236   if(INMAIN==NULL){
237     printf("Input file open error. Exiting. \n");
238     exit(1);
239     }
240   }
    /* read the contents of the main input file */
241 fscanf(INMAIN,"%s %s",mstrdum,PREF);
242 if(strcmp(mstrdum,"OUTPREF")!=0){
243   printf("Input file error.\n");
244   printf("Entry %s should say \"OUTPREF\". Exiting.\n",mstrdum);
245   exit(1);
246   }
247 if(DEBUG>-1){
248   sprintf(mstrdum,"%s_debug.txt",PREF);
249   DBG=fopen(mstrdum,"w");
```

```
250   if(DBG==NULL){
251     printf("Error opening debugging output file. Exiting.\n");
252     exit(1);
253     }
254   }
255 sprintf(mstrdum,"%s_parameter.txt",PREF);
256 PAR=fopen(mstrdum,"w");
257 if(PAR==NULL){
258   printf("Error opening parameter file. Exiting.\n");
259   exit(1);
260   }
261 strcpy(TMPFILE,".rvesim_temp");
262 fscanf(INMAIN,"%s %s",mstrdum,mtmp);
263 if(strcmp(mstrdum,"UNITS")!=0){
264   printf("Input file error.\n");
265   printf("Entry %s should say \"UNITS\". Exiting.\n",mstrdum);
266   exit(1);
267   }
268 else{
269   switch (mtmp[0]){
270     case 'c':
271       UNITTYPE=1;
272       break;
273     case 'n':
274       UNITTYPE=0;
275       break;
276     default:
277       printf("Unexpected entry for \"Units\". Exiting.\n");
278       exit(1);
279     }
280   }
281 fscanf(INMAIN,"%s %lf",mstrdum,&RESN);
282 if(strcmp(mstrdum,"RESOLUTION")!=0){
283   printf("Input file error.\n");
284   printf("Entry %s should say \"RESOLUTION\". Exit-
    ing.\n",mstrdum);
285   exit(1);
286   }
287 else RESN/=2;
288 fscanf(INMAIN,"%s %s",mstrdum,mtmp);
289 if(strcmp(mstrdum,"EFFICIENCY")!=0){
290   printf("Input file error.\n");
291   printf("Entry %s should say \"EFFICIENCY\". Exit-
    ing.\n",mstrdum);
292   exit(1);
293   }
294 else{
295   switch (mtmp[0]){
296     case '1':
297       EFFTYPE=1;
298       break;
299     default:
300       EFFTYPE=0;
301       EF=(XYlist*)calloc(1,sizeof(XYlist));
```

```
302       strcpy(EF[0].f.f,mtmp);
303       read_XY_file(EF);
304     }
305   }
306 fscanf(INMAIN,"%s %s",mstrdum,mtmp);
307 if(strcmp(mstrdum,"DETECTION")!=0){
308   printf("Input file error.\n");
309   printf("Entry %s should say \"DETECTION\". Exiting.\n",mstrdum);
310   exit(1);
311   }
312 else{
313   switch (mtmp[0]){
314     case 'c':
315       DETECTTYPE=4;
316       break;
317     case 'p':
318       DETECTTYPE=3;
319       break;
320     default:
321     printf("Unexpected entry for \"DETECTION\". Exiting.\n");
322       exit(1);
323     }
324   }
325 fscanf(INMAIN,"%s %s",mstrdum,mtmp);
326 if(strcmp(mstrdum,"SCAN")!=0){
327   printf("Input file error.\n");
328   printf("Entry %s should say \"SCAN\". Exiting.\n",mstrdum);
329   exit(1);
330   }
331 else{
332   switch (mtmp[0]){
333     case 's':
334       SCANTYPE=0;
335       break;
336     case 'j':
337       SCANTYPE=1;
338       break;
339     default:
340   printf("Unexpected entry for \"SCAN\". Exiting.\n");
341       exit(1);
342     }
343   }
344 fscanf(INMAIN,"%s %lf",mstrdum,&JKCUT);
345 if(strcmp(mstrdum,"CUTOFF")!=0){
346   printf("Input file error.\n");
347   printf("Entry %s should say \"CUTOFF\". Exiting.\n",mstrdum);
348   exit(1);
349   }
350 if(JKCUT<=0){
351 fprintf(PAR,"The low intensity cutoff was set to a value too low
    for\n");
352 fprintf(PAR,"\tthis program to understand. Resetting to default
    0.001.\n");
353   JKCUT=0.001;
```

```
354   }
355 fscanf(INMAIN,"%s %s",mstrdum,mtmp);
356 if(strcmp(mstrdum,"INTERACTIVE")!=0){
357   printf("Input file error.\n");
358   printf("Entry %s should say \"INTERACTIVE\". Exit-
    ing.\n",mstrdum);
359   exit(1);
360   }
361 else{
362   switch (mtmp[0]){
363     case 'i':
364       INTERACT=0;
365       break;
366     case 'n':
367       INTERACT=1;
368       break;
369     default:
370   printf("Unexpected entry for \"INTERACTIVE\". Exiting.\n");
371       exit(1);
372     }
373   }
374 fscanf(INMAIN,"%s %lf",mstrdum,&MINV);
375 if(strcmp(mstrdum,"MINPOINT")!=0){
376   printf("Input file error.\n");
377   printf("Entry %s should say \"MINPOINT\". Exiting.\n",mstrdum);
378   exit(1);
379   }
380 fscanf(INMAIN,"%s %lf",mstrdum,&MAXV);
381 if(strcmp(mstrdum,"MAXPOINT")!=0){
382   printf("Input file error.\n");
383   printf("Entry %s should say \"MAXPOINT\". Exiting.\n",mstrdum);
384   exit(1);
385   }
386 fscanf(INMAIN,"%s %lf",mstrdum,&PSTEP);
387 if(strcmp(mstrdum,"POINTSTEP")!=0){
388   printf("Input file error.\n");
389   printf("Entry %s should say \"POINTSTEP\". Exiting.\n",mstrdum);
390   exit(1);
391   }
392 MINV-=0.5*PSTEP;
393 MAXV-=0.5*PSTEP;
394 fscanf(INMAIN,"%s %lf",mstrdum,&MAXINT);
395 if(strcmp(mstrdum,"MAXIMUM")!=0){
396   printf("Input file error.\n");
397   printf("Entry %s should say \"MAXIMUM\". Exiting.\n",mstrdum);
398   exit(1);
399   }
400 fscanf(INMAIN,"%s %s",mstrdum,mtmp);
401 if(strcmp(mstrdum,"ROVIBSIM")!=0){
402   printf("Input file error.\n");
403   printf("Entry %s should say \"ROVIBSIM\". Exiting.\n",mstrdum);
404   exit(1);
405   }
406 else{
```

```
407   switch (mtmp[0]){
408     case 'Y':
409       ROVIB=0;
410       break;
411     case 'N':
412       ROVIB=1;
413       break;
414     case 'y':
415       ROVIB=0;
416       break;
417     case 'n':
418       ROVIB=1;
419       break;
420     default:
421   printf("Unexpected entry for \"ROVIBSIM\". Exiting.\n");
422       exit(1);
423     }
424   }
425 if(ROVIB==0){
426   fscanf(INMAIN,"%s %s",INST.f,INTR.f);
427   INST.F=fopen(INST.f,"r");
428   if(INST.F==NULL){
429     printf("Error opening state file. Exiting.\n");
430     exit(1);
431     }
432   fclose(INST.F);
433   INTR.F=fopen(INTR.f,"r");
434   if(INTR.F==NULL){
435     printf("Error opening transition file. Exiting.\n");
436     exit(1);
437     }
438   fclose(INTR.F);
439   }
440 fscanf(INMAIN,"%s %d",mstrdum,&NUMAT);
441 if(strcmp(mstrdum,"NUMAT")!=0){
442   printf("Input file error.\n");
443   printf("Entry %s should say \"NUMAT\". Exiting.\n",mstrdum);
444   exit(1);
445   }
446 if(NUMAT!=0){
447   AT=(Ainfo*)calloc(NUMAT,sizeof(Ainfo));
448   for(ma=0;ma<NUMAT;ma++){
449     fscanf(INMAIN,"%s %lf",AT[ma].f.f,&AT[ma].p);
450     AT[ma].f.F=fopen(AT[ma].f.f,"r");
451     if(AT[ma].f.F==NULL){
452       printf("Error opening %s. Exiting.\n",AT[ma].f.f);
453       exit(1);
454       }
455     fclose(AT[ma].f.F);
456     }
457   }
458 fscanf(INMAIN,"%s %d",mstrdum,&NUMOT);
459 if(strcmp(mstrdum,"NUMOT")!=0){
460   printf("Input file error.\n");
```

```
461   printf("Entry %s should say \"NUMOT\". Exiting.\n",mstrdum);
462   exit(1);
463   }
464 if(NUMOT!=0){
465   OT=(XYlist*)calloc(NUMOT,sizeof(XYlist));
466   for(ma=0;ma<NUMOT;ma++){
467     fscanf(INMAIN,"%s %lf",OT[ma].f.f,&OT[ma].p);
468     read_XY_file(&OT[ma]);
469     }
470   }
471 fscanf(INMAIN,"%s %d",mstrdum,&NUMEX);
472 if(strcmp(mstrdum,"NUMEX")!=0){
473   printf("Input file error.\n");
474   printf("Entry %s should say \"NUMEX\". Exiting.\n",mstrdum);
475   exit(1);
476   }
477 if(NUMEX!=0){
478   EX=(XYlist*)calloc(NUMEX,sizeof(XYlist));
479   for(ma=0;ma<NUMEX;ma++){
480     fscanf(INMAIN,"%s %lf",EX[ma].f.f,&EX[ma].p);
481     read_XY_file(&EX[ma]);
482     }
483   }
484 fclose(INMAIN);

    /* Create directory for output */
485 sprintf(mstrdum,"mkdir %s_molecules",PREF);
486 system(mstrdum);
    /* FUNCTIONS for reading some of the input files */
487 if(ROVIB==0) read_state_and_transition_files();
488 if(ROVIB==0) read_FCF_file();
489 if(NUMAT>0) read_atomic_files();
    /* Loop through all the transition probabilities and make sure that
       they are all expressed in terms of fractions of one. The only
       place where this is really necessary is in the calculation of cas-
       cade intensities. But, it won't hurt elsewhere, either. If other
       files containing transition probabilites are added to the pro-
       gram, don't forget to put them in these loops, too. First: find
       the maximum transition probability */
490 if(mdebugflag<DEBUG){
491 fprintf(DBG,"NUMMOL=%d, NUMAT=%d\n",NUMMOL,NUMTR);
492 fflush(DBG);
493   }
494 for(ma=0;ma<NUMMOL;ma++){
495   for(mb=0;mb<MOL[ma].trans;mb++){
496     mOnuma=MOL[ma].t[mb].Ohi;
497     if(mOnuma==0) mOnuma=1;
498     mOnumb=MOL[ma].t[mb].Olo;
499     if(mOnumb==0) mOnumb=1;
500     mOnuma*=mOnumb;
501 if(mdebugflag<DEBUG){
502 fprintf(DBG,"First trprob scan: ma=%d, mb=%d, mOnumb=%d, mOn-
    uma=%d\n",\
503     ma,mb,mOnumb,mOnuma);
```

```
504 fflush(DBG);
505   }
506     for(mc=0;mc<mOnuma;mc++){
507       if(MOL[ma].t[mb].P[mc]>mtrprobmax){
508         mtrprobmax=MOL[ma].t[mb].P[mc];
509 if(mdebugflag<DEBUG){
510 fprintf(DBG, "MOL[%d].t[%d].P[%d]=mtrprobmax=%f\n", ma, mb, mc,
    mtrprobmax);
511 fflush(DBG);
512   }
513         }
514       }
515     }
516   }
517 for(ma=0;ma<NUMAT;ma++){
518   for(mb=0;mb<AT[ma].n;mb++){
519     if(AT[ma].A[mb]>mtrprobmax) mtrprobmax=AT[ma].A[mb];
520 if(mdebugflag<DEBUG){
521 fprintf(DBG,"AT[%d].A[%d]=%f;
    mtrprobmax=%f\n",ma,mb,AT[ma].A[mb],mtrprobmax);
522 fflush(DBG);
523   }
524     }
525   }
     /* Now, scale all the transition probabilites to that one */
526 for(ma=0;ma<NUMMOL;ma++){
527   for(mb=0;mb<MOL[ma].trans;mb++){
528     mOnuma=MOL[ma].t[mb].Ohi;
529     if(mOnuma==0) mOnuma=1;
530     mOnumb=MOL[ma].t[mb].Olo;
531     if(mOnumb==0) mOnumb=1;
532     mOnuma*=mOnumb;
533 if(mdebugflag<DEBUG){
534 fprintf(DBG,"Second trprob scan: ma=%d, mb=%d, mOnumb=%d, mOn-
    uma=%d\n",\
535     ma,mb,mOnumb,mOnuma);
536 fflush(DBG);
537   }
538     for(mc=0;mc<mOnuma;mc++){
539 if(mdebugflag<DEBUG){
540 fprintf(DBG, "MOL[%d].t[%d].P[%d]=%f\t", ma, mb, mc,
    MOL[ma].t[mb].P[mc]);
541 fflush(DBG);
542   }
543       MOL[ma].t[mb].P[mc]/=mtrprobmax;
544 if(mdebugflag<DEBUG){
545 fprintf(DBG, "MOL[%d].t[%d].P[%d]=%f\n", ma, mb, mc,
    MOL[ma].t[mb].P[mc]);
546 fflush(DBG);
547   }
548         }
549       }
550     }
551 for(ma=0;ma<NUMAT;ma++){
```

```
552   for(mb=0;mb<AT[ma].n;mb++){
553 if(mdebugflag<DEBUG){
554 fprintf(DBG,"AT[%d].A[%d]=%f\t",ma,mb,AT[ma].A[mb]);
555 fflush(DBG);
556   }
557     AT[ma].A[mb]/=mtrprobmax;
558 if(mdebugflag<DEBUG){
559 fprintf(DBG,"AT[%d].A[%d]=%f\n",ma,mb,AT[ma].A[mb]);
560 fflush(DBG);
561   }
562     }
563   }
     /* FUNCTION to calculate energies and relative intensities for ro-
       tational states */
564 if(ROVIB==0) manage_rotations();
     /* FUNCTION to step through transitions, figure out what sort of
       transitions they are and then call the right functions to simu-
       late them */
565 if(ROVIB==0) manage_transitions();
     /* FUNCTION to add together all the individual pieces and make the
       simulation -- this function will call others that write the data
       */
566 do_simulation();
     /* FUNCTION to graph the simulated spectrum, with experimental if
       requested */
     /* graph_results(); */
     /* FUNCTION to ask user if something should be changed and to make
       the change as needed */
     /* ask_for_changes(); */

567 if(ROVIB==0) fclose(INTR.F);
568 if(NUMMOL>0) free(MOL);
569 if(Case_a_num>0) free(CA);
570 if(Case_b_num>0) free(CB);
571 if(NUMAT>0) free(AT);
572 if(NUMEX>0) free(EX);
573 if(NUMOT>0) free(OT);
574 if(EFFTYPE==0) free(EF);
575 sprintf(mstrdum,"rm %s",TMPFILE);
576 system(mstrdum);
577 if(DEBUG>-1) fclose(DBG);
578 fclose(PAR);
579 return 0;
580 }


     /***************** manage_transitions *****************/

     /* This function identifies each transition's type and calls the
       correct function to do the transition. */

581 void manage_transitions(){

582 int mta=0,mtb=0,mtc=0,mtd=0,mte=0,mtdone=1,mtcasesum=0,*mtz;
583 double *mtTeH,*mtTeL,mtEmax=0;
```

```
584 int mtcounter=0,mtsorth=0,mtsortdum=0;

585 if(DEBUG>mtdebugflag){
586 fprintf(DBG,"NUMMOL is %d\n",NUMMOL);
587 fflush(DBG);
588 }
     /* begin loop over molecules */
589 for(mta=0;mta<NUMMOL;mta++){
     /* begin loop over transitions */
590   mtz=(int*)calloc(MOL[mta].trans,sizeof(int));
591   mtTeH=(double*)calloc(MOL[mta].trans,sizeof(double));
592   mtTeL=(double*)calloc(MOL[mta].trans,sizeof(double));
     /* the following is a mechanism for sorting the transitions from
        highest energy to lowest energy -- this is necessary to get the
        cascade right. The algorithm is probably primitive and ineffi-
        cient, so if you want to improve it, go right ahead... */
593   for(mtb=0;mtb<MOL[mta].trans;mtb++){ /* save Te's to temporary
        arrays */
594     mtz[mtb]=mtb;
595     if(MOL[mta].s[MOL[mta].t[mtb].Hi].Ca>-1){
596       mtTeH[mtb]=CA[MOL[mta].s[MOL[mta].t[mtb].Hi].Ca].Te;}
597     if(MOL[mta].s[MOL[mta].t[mtb].Hi].Cb>-1){
598       mtTeH[mtb]=CB[MOL[mta].s[MOL[mta].t[mtb].Hi].Cb].Te;}
599     if(MOL[mta].s[MOL[mta].t[mtb].Lo].Ca>-1){
600       mtTeL[mtb]=CA[MOL[mta].s[MOL[mta].t[mtb].Lo].Ca].Te;}
601     if(MOL[mta].s[MOL[mta].t[mtb].Lo].Cb>-1){
602       mtTeL[mtb]=CB[MOL[mta].s[MOL[mta].t[mtb].Lo].Cb].Te;}
603 if(DEBUG>mtdebugflag){
604 fprintf(DBG,"MOL[%d].s[%d].Ca (Hi) is %d",mta,MOL[mta].t[mtb].Hi,\
605     MOL[mta].s[MOL[mta].t[mtb].Hi].Ca);
606 fprintf(DBG,"MOL[%d].s[%d].Cb (Hi) is
    %d\n",mta,MOL[mta].t[mtb].Hi,\
607     MOL[mta].s[MOL[mta].t[mtb].Hi].Cb);
608 fprintf(DBG,"MOL[%d].s[%d].Ca (Lo) is %d",mta,MOL[mta].t[mtb].Lo,\
609     MOL[mta].s[MOL[mta].t[mtb].Lo].Ca);
610 fprintf(DBG,"MOL[%d].s[%d].Cb (Lo) is
    %d\n",mta,MOL[mta].t[mtb].Lo,\
611     MOL[mta].s[MOL[mta].t[mtb].Lo].Cb);
612 fprintf(DBG,"mtTeH[%d] is %f; mtTeL[%d} is %f\n",\
613     mtb,mtTeH[mtb],mtb,mtTeL[mtb]);
614 fflush(DBG);
615 }
616     }
617   mtdone=1;
618   if(MOL[mta].trans==1){mtdone=0;}
619 if(DEBUG>mtdebugflag){
620 fprintf(DBG,"Before sort. mtdone is %d\n ",mtdone);
621 fflush(DBG);
622 }
623 mtcounter=0;
624   while(mtdone!=0){ /* this first loop orders the term energy for
       the high state */
625     for(mtb=0;mtb<(MOL[mta].trans-1);mtb++){
626 if(DEBUG>mtdebugflag){
```

```
627 fprintf(DBG,"First sort for-loop: mtb=%d, mtz[mtb]=%d, mt-
    TeH[%d]=%f\n",\
628     mtb,mtz[mtb],mtb,mtTeH[mtb]);
629 fflush(DBG);
630 }
631       mtsorth=mtsortdum=mtz[mtb];
632       mtEmax=mtTeH[mtz[mtb]];
633       for(mtc=mtb;mtc<(MOL[mta].trans);mtc++){
634         if(mtTeH[mtz[mtc]]>mtEmax) {
635           mtsorth=mtz[mtc];
636           mtEmax=mtTeH[mtz[mtc]];
637           }
638 if(DEBUG>mtdebugflag){
639 fprintf(DBG,"scan mtb=%d, mtc=%d, mtz[mtc]=%d, mtTeH[%d]=%f\n",\
640     mtb,mtc,mtz[mtc],mtz[mtc],mtTeH[mtz[mtc]]);
641 fprintf(DBG,"\t\t mtsorth=%d, mtEmax=%f\n",mtsorth,mtEmax);
642 fflush(DBG);
643 }
644         }
645       mtz[mtb]=mtsorth;

646 if(DEBUG>mtdebugflag){
647   for(mtc=0;mtc<mtb+2;mtc++){
648 fprintf(DBG,"mtc=%d, mtz[mtc]=%d, mtTeH[%d]=%f\n",\
649     mtc,mtz[mtc],mtz[mtc],mtTeH[mtz[mtc]]);
650   }
651 fflush(DBG);
652 }

653       for(mtc=(mtb+1);mtc<(MOL[mta].trans);mtc++){
654         if(mtz[mtc]==mtsorth) {
655           mtz[mtc]=mtsortdum;
656           }
657 if(DEBUG>mtdebugflag){
658 fprintf(DBG,"assign loop: mtb=%d, mtc=%d, mtz[mtb]=%d,
    mtz[mtc]=%d\n",\
659     mtb,mtc,mtz[mtb],mtz[mtc]);
660 fflush(DBG);
661 }
662         }
663       }
664     mtdone=0;
665     for(mtb=0;mtb<(MOL[mta].trans-1);mtb++){
666       if(mtTeH[mtz[mtb]]<mtTeH[mtz[mtb+1]]){
667         mtdone=1;
668         }
669 if(DEBUG>mtdebugflag){
670 fprintf(DBG, "sort check: mtz[%d]=%d, mtTeH[%d]=%f,
    mtTeH[%d]=%f\n",\
671   mtb, mtz[mtb], mtz[mtb], mtTeH[mtz[mtb]], mtz[mtb+1], mt-
    TeH[mtz[mtb+1]]);
672 fflush(DBG);
673 }
674       }
```

```
675 if(DEBUG>mtdebugflag){
676 fprintf(DBG,"sort check: mtz[%d]=%d, mtTeH[%d]=%f\n",\
677   mtb,mtz[mtb],mtz[mtb],mtTeH[mtz[mtb]]);
678 fflush(DBG);
679 }
680     }
681   for(mtd=0;mtd<MOL[mta].trans-1;mtd++){ /* this loop orders the
      term energy for the low state */
682     mtdone=1;
683     mte=mtd;
684     while(mtdone!=0){
685       if(mtTeH[mtz[mte]]!=mtTeH[mtz[mte+1]]){mtdone=0;}
686       else mte++;
687       }
688     for(mtb=mtd;mtb<mte;mtb++){
689 if(DEBUG>mtdebugflag){
690 fprintf(DBG,"low sort mtb=%d, mtz[mtb]=%d, mtTeL[%d]=%f\n",\
691     mtb,mtz[mtb],mtb,mtTeL[mtb]);
692 fprintf(DBG,"\t\t mtz[mtb+1]=%d, mtTeL[%d]=%f\n",\
693     mtz[mtb+1],mtb+1,mtTeL[mtb+1]);
694 fflush(DBG);
695 }
696       mtsorth=mtsortdum=mtz[mtb];
697       mtEmax=mtTeL[mtz[mtb]];
698       for(mtc=mtb;mtc<=mte;mtc++){
699         if(mtTeL[mtz[mtc]]>mtEmax) {
700           mtsorth=mtz[mtc];
701           mtEmax=mtTeL[mtz[mtc]];
702           }
703 if(DEBUG>mtdebugflag){
704 fprintf(DBG,"scan mtb=%d, mtc=%d, mtz[mtc]=%d, mtTeL[%d]=%f\n",\
705     mtb,mtc,mtz[mtc],mtz[mtc],mtTeL[mtz[mtc]]);
706 fprintf(DBG,"\t\t mtsorth=%d, mtEmax=%f\n",mtsorth,mtEmax);
707 fflush(DBG);
708 }
709         }
710       mtz[mtb]=mtsorth;

711 if(DEBUG>mtdebugflag){
712   for(mtc=mtd;mtc<mtb+2;mtc++){
713 fprintf(DBG,"mtc=%d, mtz[mtc]=%d, mtTeL[%d]=%f\n",\
714     mtc,mtz[mtc],mtz[mtc],mtTeL[mtz[mtc]]);
715   }
716 fflush(DBG);
717 }

718       for(mtc=(mtb+1);mtc<=mte;mtc++){
719         if(mtz[mtc]==mtsorth) {
720           mtz[mtc]=mtsortdum;
721           }
722 if(DEBUG>mtdebugflag){
723 fprintf(DBG,"assign loop: mtb=%d, mtc=%d, mtz[mtb]=%d,
    mtz[mtc]=%d\n",\
724     mtb,mtc,mtz[mtb],mtz[mtc]);
```

```
725 fflush(DBG);
726 }
727         }
728       }
729     mtdone=0;
730     for(mtb=mtd;mtb<(mte);mtb++){
731       if((mtTeH[mtz[mtb]]==mtTeH[mtz[mtb+1]])&&\
732         (mtTeL[mtz[mtb]]<mtTeL[mtz[mtb+1]])){
733         mtdone=1;
734         }
735 if(DEBUG>mtdebugflag){
736 fprintf(DBG,"sort check low mtb=%d, mtz[mtb]=%d, mtTeH[%d]=%f, ",\
737     mtb,mtz[mtb],mtz[mtb],mtTeH[mtz[mtb]]);
738 fprintf(DBG,"mtTeL[%d]=%f\n",mtz[mtb],mtTeL[mtz[mtb]]);
739 fprintf(DBG,"\t\t mtz[mtb+1]=%d, mtTeL[%d]=%f\n",\
740     mtz[mtb+1],mtz[mtb+1],mtTeL[mtz[mtb+1]]);
741 fflush(DBG);
742 }
743       }
744 if(DEBUG>mtdebugflag){
745 fprintf(DBG,"sort check low mtb=%d, mtz[mtb]=%d, mtTeH[%d]=%f, ",\
746     mtb,mtz[mtb],mtz[mtb],mtTeH[mtz[mtb]]);
747 fprintf(DBG,"mtTeL[%d]=%f\n",mtz[mtb],mtTeL[mtz[mtb]]);
748 fflush(DBG);
749 }
750     mtd=mte;
751     }
752   for(mtc=0;mtc<MOL[mta].trans;mtc++){
753     mtb=mtz[mtc];  /* do transitions in order determined in the last
      loop */
    /* decide which type of transition it is and call that one */
    /* note: lines for transitions involving Case c and d are included
      here for "forward compatibility" -- they can't be used yet. But,
      having them in the code shouldn't cause a problem. */
754     if(MOL[mta].s[MOL[mta].t[mtb].Hi].Ca>-1) mtcasesum=1;
755     if(MOL[mta].s[MOL[mta].t[mtb].Hi].Cb>-1) mtcasesum=2;
756     if(MOL[mta].s[MOL[mta].t[mtb].Hi].Cc>-1) mtcasesum=4;
757     if(MOL[mta].s[MOL[mta].t[mtb].Hi].Cd>-1) mtcasesum=8;
758 if(DEBUG>mtdebugflag){
759 fprintf(DBG,"mtcasenum for mtb=%d (mtz[%d]=%d) is %d\n",\
760     mtb,mtc,mtz[mtc],mtcasesum);
761 fflush(DBG);
762 }

763     if(MOL[mta].s[MOL[mta].t[mtb].Lo].Ca>-1) mtcasesum+=1;
764     if(MOL[mta].s[MOL[mta].t[mtb].Lo].Cb>-1) mtcasesum+=2;
765     if(MOL[mta].s[MOL[mta].t[mtb].Lo].Cc>-1) mtcasesum+=4;
766     if(MOL[mta].s[MOL[mta].t[mtb].Lo].Cd>-1) mtcasesum+=8;
767 if(DEBUG>mtdebugflag){
768 fprintf(DBG,"mtcasenum for mtb=%d (mtz[%d]=%d) is %d\n",\
769     mtb,mtc,mtz[mtc],mtcasesum);
770 fflush(DBG);
771 }
```

```
772     switch(mtcasesum){
773       case 2:
774 if(DEBUG>mtdebugflag){
775 fprintf(DBG,"a_a transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
776   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
777 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
778   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
779 fflush(DBG);
780 }
781         a_a(mta,mtb);
782         break;
783       case 3:
784 if(DEBUG>mtdebugflag){
785 fprintf(DBG,"a_b transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
786   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
787 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
788   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
789 fflush(DBG);
790 }
791         /*a_b(mta,mtb);*/
792         break;
793       case 5:
794 if(DEBUG>mtdebugflag){
795 fprintf(DBG,"a_c transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
796   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
797 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
798   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
799 fflush(DBG);
800 }
801         /*a_c(mta,mtb);*/
802         break;
803       case 9:
804 if(DEBUG>mtdebugflag){
805 fprintf(DBG,"a_d transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
806   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
807 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
808   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
809 fflush(DBG);
810 }
811         /*a_d(mta,mtb);*/
812         break;
813       case 4:
814 if(DEBUG>mtdebugflag){
815 fprintf(DBG,"b_b transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
816   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
817 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
818   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
819 fprintf(DBG,"b_b transition called for\n\t MOL[%d].t[%d].Nhi = %s
    to ",\
```

```
820   mta,mtb,MOL[mta].t[mtb].Nhi);
821 fprintf(DBG,"b_b transition called for\n\t MOL[%d].t[%d].Nlo = %s
    \n",\
822   mta,mtb,MOL[mta].t[mtb].Nlo);
823 fflush(DBG);
824 }
825         b_b(mta,mtb);
826         break;
827       case 6:
828 if(DEBUG>mtdebugflag){
829 fprintf(DBG,"b_c transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
830   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
831 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
832   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
833 fflush(DBG);
834 }
835         /*b_c(mta,mtb);*/
836         break;
837       case 10:
838 if(DEBUG>mtdebugflag){
839 fprintf(DBG,"b_d transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
840   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
841 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
842   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
843 fflush(DBG);
844 }
845         /*b_d(mta,mtb);*/
846         break;
847       case 8:
848 if(DEBUG>mtdebugflag){
849 fprintf(DBG,"c_c transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
850   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
851 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
852   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
853 fflush(DBG);
854 }
855         /*c_c(mta,mtb);*/
856         break;
857       case 12:
858 if(DEBUG>mtdebugflag){
859 fprintf(DBG,"c_d transition called for\n\t MOL[%d].s[%d].Name = %s
    to ",\
860   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
861 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
862   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
863 fflush(DBG);
864 }
865         /*c_d(mta,mtb);*/
866         break;
867       case 16:
868 if(DEBUG>mtdebugflag){
```

```
 869 fprintf(DBG,"d_d transition called for\n\t MOL[%d].s[%d].Name = %s
     to ",\
 870   mta,(MOL[mta].t[mtb].Hi),MOL[mta].s[MOL[mta].t[mtb].Hi].Name);
 871 fprintf(DBG,"MOL[%d].s[%d].Name = %s\n",\
 872   mta,(MOL[mta].t[mtb].Lo),MOL[mta].s[MOL[mta].t[mtb].Lo].Name);
 873 fflush(DBG);
 874 }
 875          /*d_d(mta,mtb);*/
 876         break;
 877       default:
 878 printf("Error in manage_transitions at mta=%d, mtb=%d. Exit-
     ing.\n",mta,mtb);
 879         exit(1);
 880       } /* close switch case */
 881     } /* close loop over transitions */
 882   free(mtz);
 883   } /* close loop over molecules */
 884 return;
 885 }


      /***************** HL *****************/
 886
      /* This function calculates the RELATIVE Holn-London factor for all
         transitions (called from all transition functions). The equa-
         tions here are from Herzberg and expressed conveniently for emis-
         sion.
     For consistency with the currently written transition functions,
the integer for P is +1, for Q is 0 and for R is -1. Since Delta-Lambda
isn't used in those functions, they will be counted from the lower
state..... confusion.....*/
 887 double HL(int HB, double HJ, int HL, int HdL){

 888 double Hfac=0;

 889 if(HdL==0){ /* if Delta-Lambda =0 */
 890   if(HB==+1){ /* P-branch */
 891     Hfac=(HJ+1+HL)*(HJ+1-HL)/(2*HJ*HJ+3*HJ+1);
 892     }
 893   if(HB==0){ /* Q-branch */
 894     if(HJ==0) Hfac=0;
 895     else Hfac=HL*HL/(HJ*(HJ+1));
 896     }
 897   if(HB==-1){ /* R-branch */
 898     if(HJ==0) Hfac=0;
 899     else Hfac=(HJ+HL)*(HJ-HL)/(HJ*(2*HJ+1));
 900     }
 901   }
 902 if(HdL==+1){ /* if the low state is one less than the high state */
 903   if(HB==+1){ /* P-branch */
 904     Hfac=0.5*(HJ+1-HL)*(HJ+2-HL)/(2*HJ*HJ+3*HJ+1);
 905     }
 906   if(HB==0){ /* Q-branch */
 907     if(HJ==0) Hfac=0;
 908     else Hfac=0.5*(HJ+HL)*(HJ+1-HL)/(HJ*(HJ+1));
```

```
909     }
910   if(HB==-1){ /* R-branch */
911     if(HJ==0) Hfac=0;
912     else Hfac=0.5*(HJ+HL)*(HJ-1+HL)/(HJ*(2*HJ+1));
913     }
914   }
915 if(HdL==-1){ /* if the low state is one more than the high state */
916   if(HB==+1){ /* P-branch */
917     Hfac=0.5*(HJ+1+HL)*(HJ+2+HL)/(2*HJ*HJ+3*HJ+1);
918     }
919   if(HB==0){ /* Q-branch */
920     if(HJ==0) Hfac=0;
921     else Hfac=0.5*(HJ-HL)*(HJ+1+HL)/(HJ*(HJ+1));
922     }
923   if(HB==-1){ /* R-branch */
924     if(HJ==0) Hfac=0;
925     else Hfac=0.5*(HJ-HL)*(HJ-1-HL)/(HJ*(2*HJ+1));
926     }
927   }
928 if(HLswitch==0) return Hfac;
929 else return 1;
930 }

    /***************** a-a *****************/
931
    /* This function calculates transitions between Hund's Cases (a)
       and (a) */

932 void a_a(int aaa, int aab){
    /* indexes and dummy variables: */
933 int aac=0,aacc=0,aad=0,aadd=0,aae=0,aaee=0,aaf=0,aaff=0,aag=0;
934 int aah=0,aaj=0,aafvl=0,aafvh=0,aafrnh=0,aafrcl=0,aafrch=0;
935 int aadum=0;
    /* for variables below: n="native"; c="cascade"; St="state";
       Ca="Case a"; v="vibrational"; num="number"; des="designation";
       hi="high"; lo="low"; O="Omega"; max="maximum"; pec="pseudo-
       Einstein coefficient"; comp="comparison"; a,b,<etc> are indexes
       */
    /* variables about vibrational levels */
936 int aahvnlo=0, aahvnnum=0, aahvclo=0, aahvcnum=0, aahivlo=0,
    aahivnum=0, aaHV=0;
937 int aalvnlo=0, aalvnnum=0, aalvclo=0, aalvcnum=0, aalovlo=0,
    aalovnum=0, aaLV=0;
    /* variables for info about the two states */
938 int aaStlo=0, aaSthi=0, aaStnlovlo=30, aaStnlovhi=0, aaSt-
    clovlo=30, aaStclovhi=0;
939 int aaCahi=0,aaCalo=0,aaJsh=0;
940 double aahipop=0, aatrprob=0, aavhpop=0, aaBvDv=0, aaDe=0,
    aavfcf=0, aavfcfr=0;
    /* variables about Omegas and J-values */
941 int aaOhnum=0, aaOlnum=0, aaOvhnum=0, aaOvlnum=0, aamaxhiJ=0;
942 double aahiOlo=0, aaloOlo=0, aamaxOmJ=0, aaOmJ=0, aaOlocurr=0, aaO-
    hicurr=0;
```

```
     /* variables related to which v-v transitions need to be included */
943  double aapecnmax=0, aapecncompb=0, aapeccmax=0, aapecccompb=0;
     /* variables for the spectroscopic constants */
944  double aawexehi=0, aawexelo=0, aaTehi=0, aaTelo=0, aawehi=0,
     aawelo=0;
     /* some local pointers to use as abbreviations for the longer ad-
        dresses */
945  Case_a_stateinfo *aaHI,*aaLO;
946  Trans *aaT;
947  rotset *aaRh,*aaRl;
948  vset *aaVh;
     /* a few flags for various purposes */
949  int aaflg=0, aafrnhf=0, aafrchf=0;
     /* variables for creating and writing to output files */
950  char aafile[1000];
951  FILE *AAFILE;
     /* variables for calculating Holn-London factors */
952  int aaL=0,aaB=0,aadL=0;
953  double aaJ=0;

     /* Check FCF's for lower-state vibrational levels. Find highest and
        lowest lower-state levels that wil be populated (within the user-
        set lower precision limit). Check to see that there are energies
        for all those levels. If not, reallocate the state info to have
        room for the new levels. Don't add "native" population -- only
        cascade. (the added population will be ignored if it's a destina-
        tion state only) */
954  aaSthi=MOL[aaa].t[aab].Hi;
955  aaStlo=MOL[aaa].t[aab].Lo;
956  aaCahi=MOL[aaa].s[aaSthi].Ca;
957  aaCalo=MOL[aaa].s[aaStlo].Ca;
958  aaLO=&CA[aaCalo];
959  aaHI=&CA[aaCahi];
960  aahipop=MOL[aaa].s[aaSthi].pop;  /* upper state relative population
        */
961  aahvnlo=MOL[aaa].s[aaSthi].v[0].vlo;  /* user-specified low vib
        number */
962  aahvnnum=MOL[aaa].s[aaSthi].v[0].vnum;  /* user-specified number
        of vib levels */
963  aahvclo=MOL[aaa].s[aaSthi].v[0].vclo;  /* low vib level from cas-
        cade */
964  aahvcnum=MOL[aaa].s[aaSthi].v[0].vcnum;  /* number of vib levels
        from cascade */
965  aalvnlo=MOL[aaa].s[aaStlo].v[0].vlo;  /* user-specified low vib
        number */
966  aalvnnum=MOL[aaa].s[aaStlo].v[0].vnum;  /* user-specified number
        of vib levels */
967  aalvclo=MOL[aaa].s[aaStlo].v[0].vclo;  /* low vib level from cas-
        cade */
968  aalvcnum=MOL[aaa].s[aaStlo].v[0].vcnum;  /* number of vib levels
        from cascade */
969  aaOhnum=CA[aaCahi].O;
970  aaOlnum=CA[aaCalo].O;
```

```
971 aaOvhnum=CA[aaCahi].O;  /* these because the number of sets of v-v
       transitions */
972 aaOvlnum=CA[aaCalo].O;  /* might not be the same as the number of O-O
       trans's */
973 aawexehi=aaHI[0].wexe;
974 aawexelo=aaLO[0].wexe;
975 aaTehi=aaHI[0].Te;
976 aaTelo=aaLO[0].Te;
977 aawehi=aaHI[0].we;
978 aawelo=aaLO[0].we;
979 if((aaHI[0].L-aaLO[0].L)==+1) aadL=+1;
980 if((aaHI[0].L==aaLO[0].L)) aadL=+0;
981 if((aaHI[0].L-aaLO[0].L)==-1) aadL=-1;
982 aaL=aaHI[0].L;

983 if(aadebugflag<DEBUG){
984 fprintf(DBG,"aaSthi=%d, aaStlo=%d, aaCahi=%d, aaCalo=%d,
    aahipop=%f\n",\
985     aaSthi,aaStlo,aaCahi,aaCalo,aahipop);
986 fprintf(DBG,"aahvnlo=%d, aahvnnum=%d, aahvclo=%d, aahvcnum=%d\n",\
987     aahvnlo,aahvnnum,aahvclo,aahvcnum);
988 fprintf(DBG,"aalvnlo=%d, aalvnnum=%d, aalvclo=%d, aalvcnum=%d\n",\
989     aalvnlo,aalvnnum,aalvclo,aalvcnum);
990 fprintf(DBG,"aaOhnum=%d, aaOlnum=%d, aaOvhnum=%d, aaOvlnum=%d\n",\
991     aaOhnum,aaOlnum,aaOvhnum,aaOvlnum);
992 fflush(DBG);
993 }
     /* There needs to be either (a) all specified Omegas or (b) all pos-
        sible Omegas. Find all possible if they weren't specified. Also,
        if they weren't specified, there will only be one set of v-v tran-
        sitions */
994 if(aaOhnum==0){
995   aaOvhnum=1;
996   aaOhnum=(int)(2*CA[aaCahi].S+1.000000001); /* hail, hail, o
       great Numera / may we not have a truncation error :-) */
997   aahiOlo=(double)CA[aaCahi].L-CA[aaCahi].S;
998 if(aadebugflag<DEBUG){
999 fprintf(DBG,"aaOhnum is zero and aaOvhnum=%d, aaOhnum=%d, aahi-
    Olo=%f\n",\
1000    aaOvhnum,aaOhnum,aahiOlo);
1001 fflush(DBG);
1002 }
1003   if(aahiOlo<0){
1004     aahiOlo*=-1;  /* make a positive number for clarity */
1005     aahiOlo-=0.0000001;  /* avoid truncation mishaps */
1006     aahiOlo=ceil(aaloOlo);  /* get the next largest integer */
1007     aaOhnum-=(int)aaloOlo;  /* subtract that from 2S+1 */
1008     aahiOlo+=(double)CA[aaCahi].L-CA[aaCahi].S;
1009 if(aadebugflag<DEBUG){
1010 fprintf(DBG,"aahiOlo<0 and aaOhnum=%d,
    aahiOlo=%f\n",aaOhnum,aahiOlo);
1011 fflush(DBG);
1012 }
1013     }
```

```
1014   }
1015 if(aaOlnum==0){
1016   aaOvlnum=1;
1017   aaOlnum=(int)(2*CA[aaCalo].S+1.000000001);
1018   aaloOlo=(double)CA[aaCalo].L-CA[aaCalo].S;
1019 if(aadebugflag<DEBUG){
1020 fprintf(DBG,"aaOlnum is zero and aaOvlnum=%d, aaOlnum=%d,
     aaloOlo=%f\n",\
1021    aaOvlnum,aaOlnum,aaloOlo);
1022 fflush(DBG);
1023 }
1024   if(aaloOlo<0){
1025     aaloOlo*=-1;  /* make a positive number for clarity */
1026     aaloOlo-=0.0000001;  /* avoid truncation mishaps */
1027     aaloOlo=ceil(aaloOlo);  /* get the next largest integer */
1028     aaOlnum-=(int)aaloOlo;  /* subtract that from 2S+1 */
1029     aaloOlo+=(double)CA[aaCalo].L-CA[aaCalo].S;
1030 if(aadebugflag<DEBUG){
1031 fprintf(DBG,"aaloOlo<0 and aaOlnum=%d,
     aaloOlo=%f\n",aaOlnum,aaloOlo);
1032 fflush(DBG);
1033 }
1034     }
1035   }

      /* find total number of sub-transitions (per v-v and also Omega-
         Omega) for this transition, then allocate memory for them */
1036 if((aahvnlo<aahvclo)||(aahvcnum==0)) aahivlo=aahvnlo;
1037 else aahivlo=aahvclo;
1038 if((aahvnlo+aahvnnum)>(aahvclo+aahvcnum)){
1039   aahivnum=aahvnlo+aahvnnum-aahivlo;
1040   }
1041 else{
1042   aahivnum=aahvclo+aahvcnum-aahivlo;
1043   }
1044 MOL[aaa].t[aab].vnh=aahivnum;
1045 MOL[aaa].t[aab].vhlo=aahivlo;
1046 if(aadebugflag<DEBUG){
1047 fprintf(DBG,"aahvnlo=%d, aahvclo=%d ",aahvnlo,aahvclo);
1048 fprintf(DBG,"(aahvnlo+aahvnnum)=%d, (aahvclo+aahvcnum)=%d\n",\
1049     (aahvnlo+aahvnnum),(aahvclo+aahvcnum));
1050 fprintf(DBG,"aahivnum=%d, aahivlo=%d\n",aahivnum,aahivlo);
1051 fprintf(DBG,"aahivnum is %d, aahivlo is
     %d\n",MOL[aaa].t[aab].vnh,\
1052     MOL[aaa].t[aab].vhlo);
1053 fflush(DBG);
1054 }

      /* loop to find maximum PEC (=FCF times FREQ^DETECTTYPE) */
1055 for(aac=0;aac<aaOvhnum;aac++){
1056 for(aad=0;aad<aaOvlnum;aad++){ /* for user-specified populations
       */
1057   aaHV=aahvnnum+aahvnlo;
1058 if(aadebugflag<DEBUG){
```

```
1059 fprintf(DBG,"aac=%d, aad=%d, aaHV=%d\n",aac,aad,aaHV);
1060 fflush(DBG);
1061 }
1062   for(aae=aahvnlo;aae<aaHV;aae++){ /* find max pec for native v's
       */
1063     for(aaf=0;aaf<30;aaf++){
1064       if(MOL[aaa].t[aab].v[aaee].fcfn[aaf]>aapecnmax){
1065         aapecnmax=MOL[aaa].t[aab].v[aaee].fcfn[aaf];
1066         }
1067 if(aadebugflag<(DEBUG-1)){
1068 fprintf(DBG,"aapecnmax=%11.4e\n",aapecnmax);
1069 fflush(DBG);
1070 }
1071       }
1072     }
1073   aaHV=aahvcnum+aahvclo;
1074 if(aadebugflag<(DEBUG-1)){
1075 fprintf(DBG,"Out of find-max-pec for native v's. aaHV=%d\n",aaHV);
1076 fflush(DBG);
1077 }
1078   for(aae=aahvclo;aae<aaHV;aae++){ /* find max pec for cascade v's
       */
1079     for(aaf=0;aaf<30;aaf++){
1080       if(MOL[aaa].t[aab].v[aaee].fcfc[aaf]>aapeccmax){
1081         aapeccmax=MOL[aaa].t[aab].v[aaee].fcfc[aaf];
1082         }
1083 if(aadebugflag<(DEBUG-1)){
1084 fprintf(DBG,"aapeccmax=%11.4e\n",aapeccmax);
1085 fflush(DBG);
1086 }
1087       }
1088     }
1089   }
1090   }
     /* find minimum and maximum lower-state v's with PEC's that are
        above the low-intensity cutoff (JKCUT) */
1091 for(aac=0;aac<aaOvhnum;aac++){
1092 for(aad=0;aad<aaOvlnum;aad++){ /* for user-specified populations
       */
1093   aaHV=aahvnnum+aahvnlo;
1094 if(aadebugflag<DEBUG){
1095 fprintf(DBG,"In cut-off loop: aac=%d, aad=%d,
    aaHV=%d\n",aac,aad,aaHV);
1096 fflush(DBG);
1097 }
1098   for(aae=aahvnlo;aae<aaHV;aae++){ /* for native populations */
1099     aaee=aac*aaOvlnum*30+aad*30+aae;
1100     aaf=0;
1101 if(aadebugflag<(DEBUG-1)){
1102 fprintf(DBG,"aaee=%d\n",aaee);
1103 fflush(DBG);
1104 }
1105     while(aaflg==0){
1106       aapecncompb=MOL[aaa].t[aab].v[aaee].fcfn[aaf]/JKCUT;
```

```
1107       if(aapecncompb<aapecnmax) aadum=aaf;
1108       else aaflg=1;
1109       aaf++;
1110       }
1111 if(aadebugflag<(DEBUG-1)){
1112 fprintf(DBG,"aapecncompb=%11.4e, aaf=%d\n",aapecncompb,aaf);
1113 fflush(DBG);
1114 }
1115     if(aadum<aaStnlovlo) aaStnlovlo=aadum;
1116     aaf=29;
1117     aaflg=0;
1118 if(aadebugflag<(DEBUG-1)){
1119 fflush(DBG);
1120 }
1121     while(aaflg==0){
1122       aapecncompb=MOL[aaa].t[aab].v[aaee].fcfn[aaf]/JKCUT;
1123       if(aapecncompb<aapecnmax) aadum=aaf;
1124       else aaflg=1;
1125       aaf--;
1126       }
1127 if(aadebugflag<(DEBUG-1)){
1128 fprintf(DBG,"aapecncompb=%11.4e, aaf=%d,
     aadum=%d\n",aapecncompb,aaf,aadum);
1129 fflush(DBG);
1130 }
1131     if(aadum>aaStnlovhi) aaStnlovhi=aadum;
1132     }
1133   aaHV=aahvcnum+aahvclo;
1134 if(aadebugflag<(DEBUG-1)){
1135 fprintf(DBG,"out of native population cutoff. aaHV=%d\n",aaHV);
1136 fflush(DBG);
1137 }
1138   for(aae=aahvclo;aae<aaHV;aae++){ /* for cascade populations */
1139     aaee=aac*aaOvlnum*30+aad*30+aae;
1140     aaf=0;
1141 if(aadebugflag<(DEBUG-1)){
1142 fflush(DBG);
1143 }
1144     while(aaflg==0){
1145       aapecccompb=MOL[aaa].t[aab].v[aaee].fcfc[aaf]/JKCUT;
1146       if(aapecccompb<aapeccmax) aadum=aaf;
1147       else aaflg=1;
1148       aaf++;
1149       }
1150 if(aadebugflag<(DEBUG-1)){
1151 fprintf(DBG,"aapecccompb=%11.4e, aaf=%d,
     aadum=%d\n",aapecccompb,aaf,aadum);
1152 fflush(DBG);
1153 }
1154     if(aadum<aaStclovlo) aaStclovlo=aadum;
1155     aaf=29;
1156     aaflg=0;
1157 if(aadebugflag<(DEBUG-1)){
1158 fflush(DBG);
```

```
1159 }
1160     while(aaflg==0){
1161       aapecccompb=MOL[aaa].t[aab].v[aaee].fcfc[aaf]/JKCUT;
1162       if(aapecccompb<aapeccmax) aadum=aaf;
1163       else aaflg=1;
1164       aaf--;
1165       }
1166 if(aadebugflag<(DEBUG-1)){
1167 fprintf(DBG,"aapecccompb=%11.4e, aaf=%d,
     aadum=%d\n",aapecccompb,aaf,aadum);
1168 fflush(DBG);
1169 }
1170     if(aadum>aaStclovhi) aaStclovhi=aadum;
1171     }
1172   }
1173   }
1174 if(aadebugflag<(DEBUG-1)){
1175 fprintf(DBG,"out of cutoff loop\n\n");
1176 fflush(DBG);
1177 }

      /* determine the lowest necessary destination-state vib level and
         the number of destination-state vibration levels */
1178 if((aaStnlovlo)<(aaStclovlo)) aalovlo=aaStnlovlo;
1179 else aalovlo=aaStclovlo;
1180 if((aaStnlovhi)>(aaStclovhi)) aalovnum=aaStnlovhi+1-aalovlo;
1181 else aalovnum=aaStclovhi+1-aalovlo;
1182 MOL[aaa].t[aab].vnl=aalovnum;
1183 MOL[aaa].t[aab].vllo=aalovlo;
      /* allocate memory for the simsets in the transition structure */
1184 MOL[aaa].t[aab].nS=aaOhnum*aaOlnum*aahivnum*aalovnum;
1185 MOL[aaa].t[aab].S = (simset*) calloc(MOL[aaa].t[aab].nS,
     sizeof(simset));
1186 MOL[aaa].t[aab].fS=(int*)calloc(MOL[aaa].t[aab].nS,sizeof(int));
1187 if(aadebugflag<DEBUG){
1188 fprintf(DBG,"MOL[%d].t[%d].vnl=%d, MOL[%d].t[%d].vllo=%d
     ",aaa,aab,\
1189     MOL[aaa].t[aab].vnl,aaa,aab,MOL[aaa].t[aab].vllo);
1190 fprintf(DBG,"MOL[%d].t[%d].nS=%d\n",aaa,aab,MOL[aaa].t[aab].nS);
1191 fflush(DBG);
1192 }

      /* get maximum J value present in any of the high-state Omegas */
1193 for(aae=0;aae<aaOhnum;aae++){
1194 if(aadebugflag<DEBUG){
1195 fprintf(DBG,"At top of J-value loop. aae=%d
     aaLO[0].O=%d\n",aae,aaLO[0].O);
1196 fflush(DBG);
1197 }
1198   aaHV=aahvnnum+aahvnlo;
1199   if(aaLO[0].O!=0){ /* if Omegas are user-defined */
1200     aaOlocurr=aaLO[0].lO[aae];
1201 if(aadebugflag<DEBUG){
1202 fprintf(DBG,"aaLO[0].O!=0; aaOlocurr=%f\n",aaLO[0].lO[aae]);
```

```
1203 fflush(DBG);
1204 }
1205     }
1206   else{
1207     aaOlocurr=aaloOlo + aae;
1208 if(aadebugflag<DEBUG){
1209 fprintf(DBG,"else, and aaOlocurr=%f\n",aaOlocurr);
1210 fflush(DBG);
1211 }
1212     }
1213   for(aaf=aahvnlo;aaf<aaHV;aaf++){ /* check native populations */
1214 if(aadebugflag<(DEBUG)){
1215 fprintf(DBG,"In native population loop \n");
1216 fflush(DBG);
1217 }
1218     aaj=MOL[aaa].s[aaSthi].r[aaf-aahvnlo].j;
1219     aaOmJ=aaOlocurr+aaj-1;
1220     if(aaj>aamaxhiJ){
1221       aamaxhiJ=aaj;
1222       }
1223     if(aaOmJ>aamaxOmJ){
1224       aamaxOmJ=aaOmJ;
1225       }
1226     }
1227 if(aadebugflag<(DEBUG)){
1228 fprintf(DBG,"Native: aae=%d, aaf=%d,
    aamaxhiJ=%d\n",aae,aaf,aamaxhiJ);
1229 fflush(DBG);
1230 }
1231   aaHV=aahvcnum+aahvclo;
1232   for(aaf=aahvclo;aaf<aaHV;aaf++){ /* check cascade populations */
1233     aaj=MOL[aaa].s[aaSthi].rc[aaf-aahvclo].j;
1234     aaOmJ=aaOlocurr+aaj-1;
1235     if(aaj>aamaxhiJ){
1236       aamaxhiJ=aaj;
1237       }
1238     if(aaOmJ>aamaxOmJ){
1239       aamaxOmJ=aaOmJ;
1240       }
1241     }
1242 if(aadebugflag<(DEBUG)){
1243 fprintf(DBG,"Cascade: aae=%d, aaf=%d,
    aamaxhiJ=%d\n",aae,aaf,aamaxhiJ);
1244 fflush(DBG);
1245 }
1246   }
```

```
/* loop through the list of low-state Omegas first. See if all the
   needed destination-state v's are calculated. Check also that
   enough J's are calculated. Calculate any states that aren't al-
   ready done. The following method isn't efficient. It will cal-
   culate more lower-state energy levels than are actually needed.
   But, this shouldn't be a problem other than having a few unneces-
   sary calculations happen. */
```

```
1247 aamaxhiJ++;  /* because of the Delta-J = +1 possibility */
1248 aamaxOmJ++;  /* '' */
1249 if(aadebugflag<(DEBUG)){
1250 fprintf(DBG,"Before new calculations loop
     aamaxhiJ=%d\n",aamaxhiJ);
1251 fflush(DBG);
1252 }
      /* Calculate any newly needed energy levels. */
      /* loop through each lower Omega */
1253 for(aad=0;aad<aaOlnum;aad++){ /* loop through each lower-state
        Omega */
1254   if(aaOvlnum==1) aaee=0;  /* Only one vset if auto-Omega */
1255   else aaee=aad;
1256   if(aaLO[0].O!=0){ /* if Omegas are user-defined */
1257     aaOlocurr=aaLO[0].lO[aad];
1258 if(aadebugflag<(DEBUG)){
1259 fprintf(DBG,"Omegas User-defined and
     aaOlocurr=%15.8e.\n",aaOlocurr);
1260 fflush(DBG);
1261 }
1262     }
1263   else{
1264     aaOlocurr=aaloOlo + aad;
1265 if(aadebugflag<(DEBUG)){
1266 fprintf(DBG,"Omegas Not user-defined and
     aaOlocurr=%15.8e.\n",aaOlocurr);
1267 fprintf(DBG,"aaloOlo=%15.8e; aaOlnum=%d\n",aaloOlo,aaOlnum);
1268 fflush(DBG);
1269 }
1270     }
1271   aaLV=aalovlo+aalovnum;
1272 if(aadebugflag<(DEBUG)){
1273 fprintf(DBG,"aad=%d, aaee=%d, aaLV=%d\n",aad,aaee,aaLV);
1274 fflush(DBG);
1275 }
1276   for(aaf=aalovlo;aaf<aaLV;aaf++){ /* check low-state v-levels */
1277     aaff=aad*30+aaf; /* Index for the array of rc rotational sets */
1278 if(aadebugflag<(DEBUG)){
1279 fprintf(DBG,"aaf=%d, aaff=%d, aadum=%d, ",aaf,aaff,aadum);
1280 fflush(DBG);
1281 }
1282     if(MOL[aaa].s[aaStlo].rc[aaff].jc==0){
      /* this v has never been calculated, so call a function to calculate
        it. */
1283     if(aaLO[0].we!=0){
1284       aaDe=4*pow(aaLO[0].Be,3)/(aaLO[0].we*aaLO[0].we);
1285       aaBvDv=(aaLO[0].Be-aaLO[0].ae*(aaf+0.5))/\
1286         (2*(aaDe+aaLO[0].beta*(aaf+0.5)));
1287       MOL[aaa].s[aaStlo].rc[aaff].Jdissoc=aaLO[0].sflag+\
1288         floor((sqrt(1+4*aaBvDv)-1)/2);
1289       }
1290     else MOL[aaa].s[aaStlo].rc[aaff].Jdissoc=RAND_MAX;
1291       if(aamaxhiJ<MOL[aaa].s[aaStlo].rc[aaff].Jdissoc){
1292         case_a_cascade(&MOL[aaa].s[aaStlo].rc[aaff],\
```

```
1293           aaCalo,aaOlocurr,aaf,0,aamaxhiJ);
1294         }
1295       else{
1296         case_a_cascade(&MOL[aaa].s[aaStlo].rc[aaff],\
1297           aaCalo,aaOlocurr,aaf,0,\
1298       (int)(MOL[aaa].s[aaStlo].rc[aaff].Jdissoc+1));
1299         }
1300 if(aadebugflag<(DEBUG)){
1301 fprintf(DBG,"just called case_a_cascade, v never calc'd\n");
1302 fprintf(DBG,"aaDe=%12.6e, aaBvDv=%12.6e, ",aaDe,aaBvDv);
1303 fprintf(DBG, "MOL[%d].s[%d].rc[%d].Jdissoc = %12.6e\n", aaa,
    aaStlo, aaff,\
1304     MOL[aaa].s[aaStlo].rc[aaff].Jdissoc);
1305 fflush(DBG);
1306 fprintf(DBG,"jwas zero; just called cascade \n");
1307 fprintf(DBG,"MOL[%d].s[%d].rc[%d].jc=%d\n",aaa,aaStlo,aaff,\
1308     MOL[aaa].s[aaStlo].rc[aaff].jc);
1309 fflush(DBG);
1310 }
1311       }
1312     else{
     /* this v has been calculated. See if enough J's are calculated */
1313       if(MOL[aaa].s[aaStlo].rc[aaff].jc<aamaxhiJ){
1314 if(aadebugflag<(DEBUG)){
1315 fprintf(DBG,"j<maxJ; about to call cascade \n");
1316 fprintf(DBG,"MOL[%d].s[%d].rc[%d].jc=%d\n",aaa,aaStlo,aaff,\
1317     MOL[aaa].s[aaStlo].rc[aaff].jc);
1318 fflush(DBG);
1319 }
1320       if(aamaxhiJ<MOL[aaa].s[aaStlo].rc[aaff].Jdissoc){
1321         case_a_cascade(&MOL[aaa].s[aaStlo].rc[aaff],\
1322           aaCalo,aaOlocurr,aaf,\
1323         MOL[aaa].s[aaStlo].rc[aaff].jc,aamaxhiJ);
1324         }
1325       else{
1326         case_a_cascade(&MOL[aaa].s[aaStlo].rc[aaff],\
1327     aaCalo,aaOlocurr,aaf,MOL[aaa].s[aaStlo].rc[aaff].jc,\
1328       (int)(MOL[aaa].s[aaStlo].rc[aaff].Jdissoc+1));
1329         }
1330 if(aadebugflag<(DEBUG)){
1331 fprintf(DBG,"j was <maxJ; just called cascade \n");
1332 fprintf(DBG,"MOL[%d].s[%d].rc[%d].jc=%d\n",aaa,aaStlo,aaff,\
1333     MOL[aaa].s[aaStlo].rc[aaff].jc);
1334 fflush(DBG);
1335 fprintf(DBG,"just called case_a_cascade for cascade only\n");
1336 fflush(DBG);
1337 }
1338         }
1339       }
1340     }
     /* Reset values of vclo and vcnum for the next transition */
1341     MOL[aaa].s[aaStlo].v[aaee].vclo=aalovlo;
1342     MOL[aaa].s[aaStlo].v[aaee].vcnum=aalovnum;
1343 if(aadebugflag<(DEBUG)){
```

```
1344 fprintf(DBG,"MOL[%d].s[%d].v[%d].vclo=%d ",aaa,aaStlo,aaee,\
1345     MOL[aaa].s[aaStlo].v[aaee].vclo);
1346 fprintf(DBG,"MOL[%d].s[%d].v[%d].vcnum=%d\n",aaa,aaStlo,aaee,\
1347     MOL[aaa].s[aaStlo].v[aaee].vcnum);
1348 fflush(DBG);
1349 }
1350   }
1351
      /* Check whether larger selection rules (Lambda, S, +/- in Sigma
         states) are * violated. If they are, write a note to that effect
         to the DBG file. */
1352 if(aadebugflag<(DEBUG-1)){
1353 fprintf(DBG,"aaHI[0].L=%d, aaLO[0].L=%d\n",aaHI[0].L,aaLO[0].L);
1354 fflush(DBG);
1355 }
1356 if((abs(aaHI[0].L-aaLO[0].L))>1){
1357   fprintf(PAR,"\nTRANSITION OMITTED!!! (see below)\n\n");
1358   fprintf(PAR,"Normal selection rule violated for Delta-");
1359   fprintf(PAR,"Lambda=0,+/-1 for\n\tTransition %s-->%s\n",\
1360       MOL[aaa].t[aab].Nhi,MOL[aaa].t[aab].Nlo);
1361   fprintf(PAR,"\tThis program doesn't understand that.\n\n");
1362 if(aadebugflag<(DEBUG)){
1363 fprintf(DBG,"Delta-L selection rule violated.\n");
1364 fflush(DBG);
1365 }
1366   return;
1367   }
1368 if((fabs(aaHI[0].S-aaLO[0].S))>0.01){ /* <<< an overkill against
        round-off */
1369   fprintf(PAR,"WARNING!!! Normal selection rule violated for
     Delta-");
1370   fprintf(PAR,"S=0 for\n\tTransition %s-->%s\n",\
1371       MOL[aaa].t[aab].Nhi,MOL[aaa].t[aab].Nlo);
1372   fprintf(PAR,"\tThis is non-fatal; only a warning.\n");
1373 if(aadebugflag<(DEBUG)){
1374 fprintf(DBG,"Delta-S selection rule violated\n");
1375 fflush(DBG);
1376 }
1377   }

      /* Start with the largest Omega of the high state (in case someone
         writes in cascade between different Omegas). Check for presence
         of lower-state Omegas to which to transit. If present, transit,
         if not move on to the next higher Omega.
     If Delta-spinSigma is not zero (true if, for example, a pi-pi tran-
sition occurs and one Omega is 2 and the other is 1), flag this set of tran-
sitions to be left out of simulation and placed in separate file. At the
moment I don't recall whether thst separate file actually gets written.
*/
1378 aaT=&MOL[aaa].t[aab];  /* an abbreviation */
1379 if(aadebugflag<DEBUG){
1380 fprintf(DBG,"aaT.Nhi=%s, aaT.Nlo=%s\n",aaT[0].Nhi,aaT[0].Nlo);
1381 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vclo=%d ",aaa,aaSthi,\
1382     MOL[aaa].s[aaSthi].v[0].vclo);
```

```
1383 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vcnum=%d\n",aaa,aaSthi,\
1384     MOL[aaa].s[aaSthi].v[0].vcnum);
1385 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vlo=%d ",aaa,aaSthi,\
1386     MOL[aaa].s[aaSthi].v[0].vlo);
1387 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vnum=%d\n",aaa,aaSthi,\
1388     MOL[aaa].s[aaSthi].v[0].vnum);
1389 fprintf(DBG,"aahivnum=%d; aalovnum=%d; aaOhnum=%d, aaOlnum=%d\n",\
1390     aahivnum,aalovnum,aaOhnum,aaOlnum);
1391 fflush(DBG);
1392 }
1393 for(aac=(aaOhnum-1);aac>=0;aac--){ /* down through high-state
       Omegas */
1394   if(aaHI[0].O!=0){ /* if Omegas are user-defined */
1395     aaOhicurr=aaHI[0].lO[aac];
1396     aafvh=aac*aahivnum;
1397     aaVh=&MOL[aaa].s[aaSthi].v[aac];
1398     aatrprob=aaT[0].P[aac];
1399     }
1400   else{
1401     aaOhicurr=aahiOlo + aac;
1402     aafvh=0;
1403     aaVh=&MOL[aaa].s[aaSthi].v[0];
1404     aatrprob=aaT[0].P[0];
1405     }
1406   aafrnh=aac*aahvnnum; /* indexes for high state rotsets */
1407   aacc=aac*aaOlnum*aahivnum*aalovnum;  /* posn. in 4 dimensions */
1408 if(aadebugflag<DEBUG){
1409 fprintf(DBG,"aac=%d, aafvh=%d, aatrprob=%f ",\
1410     aac,aafvh,aatrprob);
1411 fprintf(DBG,"aafrnh=%d, aafrch=%d,
    aacc=%d\n",aafrnh,aafrch,aacc);
1412 fprintf(DBG,"aaOhicurr=%11.4e\n",aaOhicurr);
1413 fflush(DBG);
1414 }
1415 for(aad=(aaOlnum-1);aad>=0;aad--){ /* down through low-state
       Omegas */
1416   aafrnh+=aahivnum;
1417   if(aaLO[0].O!=0){ /* if Omegas are user-defined */
1418     aaOlocurr=aaLO[0].lO[aad];
1419     aafvl=aad*aalovnum;
1420     }
1421   else{
1422     aaOlocurr=aaloOlo + aad;
1423     aafvl=0;
1424     }
1425   aaJsh=(int)(aaOhicurr-aaOlocurr+0.0000001);  /* shift in J index-
       ing due to difference in Omegas */
1426   if((fabs(aaOhicurr-aaOlocurr))>1.000000001) aaflg=1;
1427   else aaflg=0;
1428   aadd=aacc+aad*aahivnum*aalovnum;  /* posn. in remaining 3 dimen-
       sions */
1429 if(aadebugflag<DEBUG){
1430 fprintf(DBG,"aad=%d, aafrcl=%d, aadd=%d\n",aad,aafrcl,aadd);
1431 fprintf(DBG,"aaOlocurr=%11.3e\n",aaOlocurr);
```

```
fflush(DBG);
}
  for(aae=(aahivlo+aahivnum-1);aae>=aahivlo;aae--){ /*high-state viblev*/
    aaee=aadd+(aae-aahivlo)*aalovnum;  /* posn. in remaining 2 dimensions */
    aafrnh--;
 /* these flags (aafrnhf and aafrchf) tell if this high vib state is natively populated, populated by cascade, or both. They will be used later */
    if((aae>=aahvnlo)&&(aae<(aahvnlo+aahvnnum))){
      aafrnhf=0;
      aavhpop=aaVh[0].p[aae-aahvnlo];
      }
    else aafrnhf=-1;
    if((aae>=aahvclo)&&(aae<(aahvclo+aahvcnum))){
      aafrchf=0;
      }
    else aafrchf=-1;
if(aadebugflag<DEBUG){
fprintf(DBG,"aafrnhf=%d, aafrnh=%d, aavhpop=%f, aafrchf=%d\n",\
    aafrnhf,aafrnh,aavhpop,aafrchf);
fprintf(DBG,"aae=%d, aaee=%d\n",aae,aaee);
fprintf(DBG,"aahivlo=%d, aahivnum=%d, aafrnh=%d, aafrch=%d\n",\
    aahivlo,aahivnum,aafrnh,aafrch);
fflush(DBG);
}
  aafrcl=aad*30+aalovnum;  /* indexes for low state rotsets */
  for(aaf=(aalovlo+aalovnum-1);aaf>=aalovlo;aaf--){ /*low-state viblev*/
if(aadebugflag<DEBUG){
fprintf(DBG,"aaf=%d, aalovlo=%d, aalovnum=%d
\n",aaf,aalovlo,aalovnum);
fflush(DBG);
}
 /* Right now, all Omegas get the same set of FCF's. */
    aaff=aaee+aaf-aalovlo;  /* posn. in final dimension */
    aafrcl--;  /* position in low-state cascade rotset */
    aaT[0].fS[aaff]=aaflg;
    aavfcf=aaT[0].v[aae].fcfn[aaf];
    if(aavfcf!=0){aavfcfr=aaT[0].v[aae].fcfc[aaf]/aavfcf;}
    if((aaOhicurr==0)&&(aaOlocurr==0))
aaT[0].S[aaff].n=2*aamaxhiJ;
    else aaT[0].S[aaff].n=3*aamaxhiJ;
    aaT[0].S[aaff].f=\
      (double*)calloc(aaT[0].S[aaff].n,sizeof(double));
    aaT[0].S[aaff].ni=\
      (double*)calloc(aaT[0].S[aaff].n,sizeof(double));
    aaT[0].S[aaff].ci=\
      (double*)calloc(aaT[0].S[aaff].n,sizeof(double));
    aaRl=&MOL[aaa].s[aaStlo].rc[aafrcl];
if(aadebugflag<DEBUG){
fprintf(DBG,"aaff=%d, aafrcl=%d, aaT[0].S[aaff].n=%d\n",\
    aaff,aafrcl,aaT[0].S[aaff].n);
```

```
fflush(DBG);
}
 /* start with highest J value (same reason as before), and loop down
    looking for lower J's to which to transit. assign intensities.
    If both states are Omega=0, then assign zero intensity to transi-
    tions for J=0<->J=0. */
  if(aafrnhf==0){ /* if this v is natively populated.*/
sprintf(aafile,"%s_molecules/%s/%s-%.1f--%s-%.1f_v%d--
v%d_NAT.dat",\
  PREF,MOL[aaa].Mol,aaT[0].Nhi,aaOhicurr,\
    aaT[0].Nlo,aaOlocurr,aae,aaf);
      AAFILE=fopen(aafile,"w");
      if(AAFILE==NULL){
printf("Error opening transition sub-file %s. Exiting.\n",aafile);
        exit(1);
        }
fprintf(AAFILE,"# File created by %s.\n# File contains ",PRO-
GRAM_NAME);
if((aaOhicurr==0)&&(aaOlocurr==0)) fprintf(AAFILE,"P and R");
else fprintf(AAFILE,"P, Q and R");
fprintf(AAFILE," branches from the simulation titled %s\n",PREF);
fprintf(AAFILE,"# THESE INTENSITIES ARE DUE TO USER-SPECIFIED POPU-
LATIONS");
fprintf(AAFILE," ONLY -- NO CASCADE\n");
fprintf(AAFILE,"# Transition is Hund's Case (a) to Hund's Case
(a).\n");
fprintf(AAFILE,"# Molecule %s, \tHigh state: %s, Omega=%.1f,
v=%d\n",\
    MOL[aaa].Mol,aaT[0].Nhi,aaOhicurr,aae);
fprintf(AAFILE,"#\t\tLow state: %s, Omega=%.1f, v=%d\n# Columns
are: ",\
    aaT[0].Nlo,aaOlocurr,aaf);
if((aaOhicurr==0)&&(aaOlocurr==0)){
fprintf(AAFILE,"J_hi P(J_hi+1)_v P_i R(J_hi-1)_v R_i\n#\n");
  }
else fprintf(AAFILE,"J_hi P(Jhi+1)_v P_i Q(J_hi+0)_v Q_i R(J_hi-1)_v
R_i\n#\n");

      aaRh=&MOL[aaa].s[aaSthi].r[aafrnh];
      if((aamaxhiJ)<MOL[aaa].s[aaSthi].r[aafrnh].j){
        aaj=aamaxhiJ-1;
        }
      else aaj=MOL[aaa].s[aaSthi].r[aafrnh].j-1;
      if((aaRl[0].Jdissoc-aaOlocurr)<(aaj-1)){
fprintf(PAR,"\nWARNING!! State %s(%.1f) of molecule %s has signifi-
cant ",\
    MOL[aaa].s[aaSthi].Name,aaOhicurr,MOL[aaa].Mol);
fprintf(PAR,"rotational population at level %.1f\n",aaRh[0].J[aaj-
1]);
fprintf(PAR,"\tBut lower state %s(%.1f) dissociates at level
%.1f\n",\
    MOL[aaa].s[aaStlo].Name,aaOlocurr,aaRl[0].Jdissoc);
fprintf(PAR,"It is likely that the bound upper state is transiting
to (an)");
```

```
1515 fprintf(PAR," unbound lower state(s).\n");
1516 fprintf(PAR,"Look for continuum emissions trailing off to the low-
     energy ");
1517 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real
     ",aae,aaf);
1518 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
     gram will");
1519 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
1520         aaj=(int)(aaRl[0].Jdissoc-aaOlocurr);
1521         }
1522 if(aadebugflag<DEBUG){
1523 fprintf(DBG,"aafrnh=%d, aaj=%d\n",aafrnh,aaj);
1524 fprintf(DBG,"MOL[%d].s[%d].r[%d].j=%d\n",aaa,aaSthi,aafrnh,\
1525     MOL[aaa].s[aaSthi].r[aafrnh].j);
1526 fflush(DBG);
1527 }
1528       for(aag=(aaj-1);aag>=0;aag--){
1529         aah=aag+aaJsh;
1530         aaJ=aaRh[0].J[aag];
1531 if(aadebugflag<DEBUG){
1532 fprintf(DBG,"aah=%d, aag=%d, aaJsh=%d, aaJ=%.1f
     \n",aah,aag,aaJsh,aaJ);
1533 fflush(DBG);
1534 }
1535         if((aaOhicurr==0)&&(aaOlocurr==0)){
1536         if(((aah+1)>=0)&&((aah+1)<=aaj)){
1537           aaB=+1;
1538           aaT[0].S[aaff].f[aag*2+1]=\
1539             aaRh[0].EJ[aag]-\
1540             aaRl[0].EJ[aah+1];
1541           aaT[0].S[aaff].ni[aag*2+1]=\
1542             aahipop*aatrprob*aavhpop*\
1543             aaRh[0].PJ[aag]*aavfcf*\
1544             HL(aaB,aaJ,aaL,aadL);
1545           aaRl[0].CJ[aah+1]+=aavfcfr*\
1546             aaT[0].S[aaff].ni[aag*2+1];
1547 if(aadebugflag<DEBUG){
1548 fprintf(DBG,"aaRh[0].J[%d]=%.1f ",aag,aaRh[0].J[aag]);
1549 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1550 fprintf(DBG,"aaRl[0].J[%d]=%.1f ",aah+1,aaRl[0].J[aah+1]);
1551 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e\n",aah+1,aaRl[0].EJ[aah+1]);
1552 fprintf(DBG,"aaRh[0].PJ[%d]=%13.6e ",aag,aaRh[0].PJ[aag]);
1553 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah+1,aaRl[0].CJ[aah+1]);
1554 fprintf(DBG,"aahipop=%13.6e aatrprob=%13.6e aavfcf=%13.6e aavh-
     pop=%13.6e\n",\
1555     aahipop,aatrprob,aavfcf,aavhpop);
1556 fflush(DBG);
1557 }
1558           }
1559         if(((aah-1)>=0)&&((aah-1)<=aaj)){
1560           aaB=-1;
1561           aaT[0].S[aaff].f[aaf*2+0]=\
1562             aaRh[0].EJ[aag]-\
1563             aaRl[0].EJ[aah-1];
```

```
1564            aaT[0].S[aaff].ni[aag*2+0]=\
1565              aahipop*aatrprob*aavhpop*\
1566              aaRh[0].PJ[aag]*aavfcf*\
1567              HL(aaB,aaJ,aaL,aadL);
1568            aaRl[0].CJ[aah-1]+=aavfcfr*\
1569              aaT[0].S[aaff].ni[aag*2+0];
1570 if(aadebugflag<DEBUG){
1571 fprintf(DBG,"aaRh[0].J[%d]=%.1f ",aag,aaRh[0].J[aag]);
1572 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1573 fprintf(DBG,"aaRl[0].J[%d]=%.1f ",aah+1,aaRl[0].J[aah-1]);
1574 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e\n",aah-1,aaRl[0].EJ[aah-1]);
1575 fprintf(DBG,"aaRh[0].PJ[%d]=%13.6e ",aag,aaRh[0].PJ[aag]);
1576 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah-1,aaRl[0].CJ[aah-1]);
1577 fprintf(DBG,"aahipop=%13.6e aatrprob=%13.6e aavfcf=%13.6e aavh-
     pop=%13.6e\n",\
1578     aahipop,aatrprob,aavfcf,aavhpop);
1579 fflush(DBG);
1580 }
1581            }
1582 if(aadebugflag<DEBUG){
1583 fprintf(DBG,"aag=%d, aaT[0].S[aaff].f[aag*2+1]=%13.6e, ",aag,\
1584     aaT[0].S[aaff].f[aag*2+1]);
1585 fprintf(DBG, "aaT[0].S[aaff].f[aag*2+0] = %13.6e\n",
     aaT[0].S[aaff].f[aag*2+0]);
1586 fprintf(DBG, "aaT[0].S[aaff].ni[aag*2+1]=%13.6e, ",
     aaT[0].S[aaff].ni[aag*2+1]);
1587 fprintf(DBG, "aaRl[0].CJ[aah+1] =
     %13.6e\naaT[0].S[aaff].ni[aag*2+0] = %13.6e, ",\
1588     aaRl[0].CJ[aah+1],aaT[0].S[aaff].ni[aag*2+0]);
1589 fprintf(DBG,"aaRl[0].CJ[aah-1]=%13.6e\n",aaRl[0].CJ[aah-1]);
1590 fflush(DBG);
1591 }
1592 fprintf(AAFILE, "%5.1f\t %18.10e\t %18.10e\t %18.10e\t %18.10e\n",
     aaRh[0].J[aag],\
1593   aaT[0].S[aaff].f[aag*2+1],aaT[0].S[aaff].ni[aag*2+1],\
1594   aaT[0].S[aaff].f[aag*2+0],aaT[0].S[aaff].ni[aag*2+0]);
1595            }
1596          else{
1597          if(((aah+1)>=0)&&((aah+1)<=aaj)){
1598            aaB=+1;
1599            aaT[0].S[aaff].f[aag*3+2]=\
1600              aaRh[0].EJ[aag]-\
1601              aaRl[0].EJ[aah+1];
1602            aaT[0].S[aaff].ni[aag*3+2]=\
1603              aahipop*aatrprob*aavhpop*\
1604              aaRh[0].PJ[aag]*aavfcf*\
1605              HL(aaB,aaJ,aaL,aadL);
1606            aaRl[0].CJ[aah+1]+=aavfcfr*\
1607              aaT[0].S[aaff].ni[aag*3+2];
1608 if(aadebugflag<DEBUG){
1609 fprintf(DBG,"aaRh[0].J[%d]=%.1f ",aag,aaRh[0].J[aag]);
1610 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1611 fprintf(DBG,"aaRl[0].J[%d]=%.1f ",aah+1,aaRl[0].J[aah+1]);
1612 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e\n",aah+1,aaRl[0].EJ[aah+1]);
```

```
1613 fprintf(DBG,"aaRh[0].PJ[%d]=%13.6e ",aag,aaRh[0].PJ[aag]);
1614 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah+1,aaRl[0].CJ[aah+1]);
1615 fprintf(DBG,"aahipop=%13.6e aatrprob=%13.6e aavfcf=%13.6e aavh-
     pop=%13.6e\n",\
1616     aahipop,aatrprob,aavfcf,aavhpop);
1617 fflush(DBG);
1618 }
1619           }
1620         if(((aah+0)>=0)&&((aah+0)<=aaj)){
1621           aaB=0;
1622           aaT[0].S[aaff].f[aag*3+1]=\
1623             aaRh[0].EJ[aag]-\
1624             aaRl[0].EJ[aah+0];
1625           aaT[0].S[aaff].ni[aag*3+1]=\
1626             aahipop*aatrprob*aavhpop*\
1627             aaRh[0].PJ[aag]*aavfcf*\
1628             HL(aaB,aaJ,aaL,aadL);
1629           aaRl[0].CJ[aah+0]+=aavfcfr*\
1630             aaT[0].S[aaff].ni[aag*3+1];
1631 if(aadebugflag<DEBUG){
1632 fprintf(DBG,"aaRh[0].J[%d]=%.1f ",aag,aaRh[0].J[aag]);
1633 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1634 fprintf(DBG,"aaRl[0].J[%d]=%.1f ",aah,aaRl[0].J[aah+0]);
1635 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e\n",aah,aaRl[0].EJ[aah+0]);
1636 fprintf(DBG,"aaRh[0].PJ[%d]=%13.6e ",aag,aaRh[0].PJ[aag]);
1637 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah,aaRl[0].CJ[aah+0]);
1638 fprintf(DBG,"aahipop=%13.6e aatrprob=%13.6e aavfcf=%13.6e aavh-
     pop=%13.6e\n",\
1639     aahipop,aatrprob,aavfcf,aavhpop);
1640 fflush(DBG);
1641 }
1642           }
1643         if(((aah-1)>=0)&&((aah-1)<=aaj)){
1644           aaB=-1;
1645           aaT[0].S[aaff].f[aag*3+0]=\
1646             aaRh[0].EJ[aag]-\
1647             aaRl[0].EJ[aah-1];
1648           aaT[0].S[aaff].ni[aag*3+0]=\
1649             aahipop*aatrprob*aavhpop*\
1650             aaRh[0].PJ[aag]*aavfcf*\
1651             HL(aaB,aaJ,aaL,aadL);
1652           aaRl[0].CJ[aah-1]+=aavfcfr*\
1653             aaT[0].S[aaff].ni[aag*3+0];
1654 if(aadebugflag<DEBUG){
1655 fprintf(DBG,"aaRh[0].J[%d]=%.1f ",aag,aaRh[0].J[aag]);
1656 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1657 fprintf(DBG,"aaRl[0].J[%d]=%.1f ",aah-1,aaRl[0].J[aah-1]);
1658 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e\n",aah-1,aaRl[0].EJ[aah-1]);
1659 fprintf(DBG,"aaRh[0].PJ[%d]=%13.6e ",aag,aaRh[0].PJ[aag]);
1660 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah-1,aaRl[0].CJ[aah-1]);
1661 fprintf(DBG,"aahipop=%13.6e aatrprob=%13.6e aavfcf=%13.6e aavh-
     pop=%13.6e\n",\
1662     aahipop,aatrprob,aavfcf,aavhpop);
1663 fflush(DBG);
```

```
1664 }
1665           }
1666 if(aadebugflag<DEBUG){
1667 fprintf(DBG, "aaT[0].S[aaff].f[aag*3+2]=%13.6e ",
     aaT[0].S[aaff].f[aag*3+2]);
1668 fprintf(DBG, "aaT[0].S[aaff].f[aag*3+1]=%13.6e ",
     aaT[0].S[aaff].f[aag*3+1]);
1669 fprintf(DBG, "aaT[0].S[aaff].f[aag*3+0]=%13.6e\n",
     aaT[0].S[aaff].f[aag*3+0]);
1670 fprintf(DBG, "aaT[0].S[aaff].ni[aag*3+2]=%13.6e ",
     aaT[0].S[aaff].ni[aag*3+2]);
1671 fprintf(DBG, "aaT[0].S[aaff].ni[aag*3+1]=%13.6e ",
     aaT[0].S[aaff].ni[aag*3+1]);
1672 fprintf(DBG, "aaT[0].S[aaff].ni[aag*3+0]=%13.6e\n",
     aaT[0].S[aaff].ni[aag*3+0]);
1673 fprintf(DBG,"aaRl[0].CJ[aah+1]=%13.6e ",aaRl[0].CJ[aah+1]);
1674 fprintf(DBG,"aaRl[0].CJ[aah+0]=%13.6e ",aaRl[0].CJ[aah+0]);
1675 fprintf(DBG,"aaRl[0].CJ[aah-1] =%13.6e\n",aaRl[0].CJ[aah-1]);
1676 fflush(DBG);
1677 }
1678 fprintf(AAFILE,\
1679   "%5.1f\t %18.10e\t %18.10e\t %18.10e\t %18.10e\t %18.10e\t
     %18.10e\n",\
1680   aaRh[0].J[aag],\
1681   aaT[0].S[aaff].f[aag*3+2],aaT[0].S[aaff].ni[aag*3+2],\
1682   aaT[0].S[aaff].f[aag*3+1],aaT[0].S[aaff].ni[aag*3+1],\
1683   aaT[0].S[aaff].f[aag*3+0],aaT[0].S[aaff].ni[aag*3+0]);
1684           }
1685         }
1686       fclose(AAFILE);
1687       }

1688     if(aafrchf==0){ /* if this v is cascade populated */
1689 sprintf(aafile,"%s_molecules/%s/%s-%.1f--%s-%.1f_v%d--
     v%d_CAS.dat",\
1690   PREF,MOL[aaa].Mol,aaT[0].Nhi,aaOhicurr,\
1691     aaT[0].Nlo,aaOlocurr,aae,aaf);
1692       AAFILE=fopen(aafile,"w");
1693       if(AAFILE==NULL){
1694 printf("Error opening transition sub-file %s. Exiting.\n",aafile);
1695         exit(1);
1696         }
1697 fprintf(AAFILE,"# File created by %s.\n# File contains ",PRO-
     GRAM_NAME);
1698 if((aaOhicurr==0)&&(aaOlocurr==0)) fprintf(AAFILE,"P and R");
1699 else fprintf(AAFILE,"P, Q and R");
1700 fprintf(AAFILE," branches from the simulation titled %s\n",PREF);
1701 fprintf(AAFILE,"# THESE INTENSITIES ARE DUE TO CASCADE ");
1702 fprintf(AAFILE," ONLY -- NO USER-SPECIFIED POPULATIONS\n");
1703 fprintf(AAFILE,"# Transition is Hund's Case (a) to Hund's Case
     (a).\n");
1704 fprintf(AAFILE,"# Molecule %s, \tHigh state: %s, Omega=%.1f,
     v=%d\n",\
1705     MOL[aaa].Mol,aaT[0].Nhi,aaOhicurr,aae);
```

```
1706 fprintf(AAFILE,"#\t\tLow state: %s, Omega=%.1f, v=%d\n# Columns
     are: ",\
1707     aaT[0].Nlo,aaOlocurr,aaf);
1708 if((aaOhicurr==0)&&(aaOlocurr==0)){
1709 fprintf(AAFILE,"J_hi P(J_hi+1)_v P_i R(J_hi-1)_v R_i\n#\n");
1710   }
1711 else fprintf(AAFILE,"J_hi P(Jhi+1)_v P_i Q(J_hi+0)_v Q_i R(J_hi-1)_v
     R_i\n#\n");
1712       aaRh=&MOL[aaa].s[aaSthi].rc[aae];
1713       if((aamaxhiJ-1)<MOL[aaa].s[aaSthi].rc[aae].jc){
1714         aaj=aamaxhiJ-1;
1715         }
1716       else aaj=MOL[aaa].s[aaSthi].rc[aae].jc;
1717       if((aaRl[0].Jdissoc-aaOlocurr)<(aaj-1)){
1718 fprintf(PAR,"\nWARNING!! State %s(%.1f) of molecule %s has signifi-
     cant ",\
1719     MOL[aaa].s[aaSthi].Name,aaOhicurr,MOL[aaa].Mol);
1720 fprintf(PAR,"CASCADE rotational population at level
     %.1f\n",aaRh[0].J[aaj-1]);
1721 fprintf(PAR,"\tBut lower state %s(%.1f) dissociates at level
     %.1f\n",\
1722     MOL[aaa].s[aaStlo].Name,aaOlocurr,aaRl[0].Jdissoc);
1723 fprintf(PAR,"It is likely that the bound upper state is transiting
     to (an)");
1724 fprintf(PAR," unbound lower state(s).\n");
1725 fprintf(PAR,"Look for continuum emissions trailing off to the low-
     energy ");
1726 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real
     ",aae,aaf);
1727 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
     gram will");
1728 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
1729         aaj=(int)(aaRl[0].Jdissoc-aaOlocurr);
1730         }
1731       for(aag=(aaj-1);aag>=0;aag--){
1732         aah=aag+aaJsh;
1733         aaJ=aaRh[0].J[aag];
1734         if((aaOhicurr==0)&&(aaOlocurr==0)){
1735         if(((aah+1)>=0)&&((aah+1)<=aaj)){
1736           aaB=+1;
1737           aaT[0].S[aaff].f[aag*2+1]=\
1738             aaRh[0].EJ[aag]-\
1739             aaRl[0].EJ[aah+1];
1740           aaT[0].S[aaff].ci[aag*2+1]=aatrprob*\
1741             aaRh[0].CJ[aag]*aavfcf*\
1742             HL(aaB,aaJ,aaL,aadL);
1743           aaRl[0].CJ[aah+1]+=aavfcfr*\
1744             aaT[0].S[aaff].ci[aag*2+1];
1745 if(aadebugflag<DEBUG){
1746 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1747 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e ",aah+1,aaRl[0].EJ[aah+1]);
1748 fprintf(DBG,"aaRh[0].CJ[%d]=%13.6e ",aag,aaRh[0].CJ[aag]);
1749 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah+1,aaRl[0].CJ[aah+1]);
1750 fprintf(DBG,"aavfcf=%13.6e aatrprob=%13.6e \n",\
```

```
1751     aavfcf,aatrprob);
1752 fflush(DBG);
1753 }
1754           }
1755         if(((aah-1)>=0)&&((aah-1)<=aaj)){
1756           aaB=-1;
1757           aaT[0].S[aaff].f[aag*2+0]=\
1758             aaRh[0].EJ[aag]-\
1759             aaRl[0].EJ[aah-1];
1760           aaT[0].S[aaff].ci[aag*2+0]=aatrprob*\
1761             aaRh[0].CJ[aag]*aavfcf*\
1762             HL(aaB,aaJ,aaL,aadL);
1763           aaRl[0].CJ[aah-1]+=aavfcfr*\
1764             aaT[0].S[aaff].ci[aag*2+0];
1765 if(aadebugflag<DEBUG){
1766 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1767 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e ",aah-1,aaRl[0].EJ[aah-1]);
1768 fprintf(DBG,"aaRh[0].CJ[%d]=%13.6e ",aag,aaRh[0].CJ[aag]);
1769 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah-1,aaRl[0].CJ[aah-1]);
1770 fprintf(DBG,"aavfcf=%13.6e aatrprob=%13.6e \n",\
1771     aavfcf,aatrprob);
1772 fflush(DBG);
1773 }
1774           }
1775 if(aadebugflag<DEBUG){
1776 fprintf(DBG,"aag=%d, aaT[0].S[aaff].f[aag*2+1]=%13.6e, ",aag,\
1777     aaT[0].S[aaff].f[aag*2+1]);
1778 fprintf(DBG, "aaT[0].S[aaff].f[aag*2+0] = %13.6e\n",
     aaT[0].S[aaff].f[aag*2+0]);
1779 fprintf(DBG, "aaT[0].S[aaff].ci[aag*2+1] = %13.6e, ",
     aaT[0].S[aaff].ci[aag*2+1]);
1780 fprintf(DBG, "aaRl[0].CJ[aah+1] = %13.6e\n
     aaT[0].S[aaff].ci[aag*2+0] = %13.6e, ",\
1781     aaRl[0].CJ[aah+1],aaT[0].S[aaff].ci[aag*2+0]);
1782 fprintf(DBG,"aaRl[0].CJ[aah-1]=%13.6e\n",aaRl[0].CJ[aah-1]);
1783 fflush(DBG);
1784 }
1785 fprintf(AAFILE,"%5.1f\t%18.10e\t%18.10e\t%18.10e\t%18.10e\n",\
1786   aaRh[0].J[aag],\
1787   aaT[0].S[aaff].f[aag*2+1],aaT[0].S[aaff].ci[aag*2+1],\
1788   aaT[0].S[aaff].f[aag*2+0],aaT[0].S[aaff].ci[aag*2+0]);
1789           }
1790         else{
1791         if(((aah+1)>=0)&&((aah+1)<=aaj)){
1792           aaB=+1;
1793           aaT[0].S[aaff].f[aag*3+2]=\
1794           aaRh[0].EJ[aag]-aaRl[0].EJ[aah+1];
1795           aaT[0].S[aaff].ci[aag*3+2]=aatrprob*\
1796             aaRh[0].CJ[aag]*aavfcf*\
1797             HL(aaB,aaJ,aaL,aadL);
1798           aaRl[0].CJ[aah+1]+=aavfcfr*\
1799             aaT[0].S[aaff].ci[aag*3+2];
1800 if(aadebugflag<DEBUG){
1801 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
```

```
1802 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e ",aah+1,aaRl[0].EJ[aah+1]);
1803 fprintf(DBG,"aaRh[0].CJ[%d]=%13.6e ",aag,aaRh[0].CJ[aag]);
1804 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah+1,aaRl[0].CJ[aah+1]);
1805 fprintf(DBG,"aavfcf=%13.6e aatrprob=%13.6e \n",\
1806     aavfcf,aatrprob);
1807 fflush(DBG);
1808 }
1809           }
1810         if(((aah+0)>=0)&&((aah+0)<=aaj)){
1811           aaB=0;
1812           aaT[0].S[aaff].f[aag*3+1]=\
1813           aaRh[0].EJ[aag]-aaRl[0].EJ[aah+0];
1814           aaT[0].S[aaff].ci[aag*3+1]=aatrprob\
1815             *aavfcf*aaRh[0].CJ[aag]*\
1816             HL(aaB,aaJ,aaL,aadL);
1817           aaRl[0].CJ[aah+0]+=aavfcfr*\
1818             aaT[0].S[aaff].ci[aag*3+1];
1819 if(aadebugflag<DEBUG){
1820 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1821 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e ",aah,aaRl[0].EJ[aah+0]);
1822 fprintf(DBG,"aaRh[0].CJ[%d]=%13.6e ",aag,aaRh[0].CJ[aag]);
1823 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah,aaRl[0].CJ[aah+0]);
1824 fprintf(DBG,"Here 20\n");
1825 fprintf(DBG,"aavfcf=%13.6e aatrprob=%13.6e\n",\
1826     aavfcf,aatrprob);
1827 fflush(DBG);
1828 }
1829           }
1830         if(((aah-1)>=0)&&((aah-1)<=aaj)){
1831           aaB=-1;
1832           aaT[0].S[aaff].f[aag*3+0]=\
1833           aaRh[0].EJ[aag]-aaRl[0].EJ[aah-1];
1834           aaT[0].S[aaff].ci[aag*3+0]=aatrprob\
1835             *aavfcf*aaRh[0].CJ[aag]*\
1836             HL(aaB,aaJ,aaL,aadL);
1837           aaRl[0].CJ[aah-1]+=aavfcfr*\
1838             aaT[0].S[aaff].ci[aag*3+0];
1839 if(aadebugflag<DEBUG){
1840 fprintf(DBG,"aaRh[0].EJ[%d]=%13.6e ",aag,aaRh[0].EJ[aag]);
1841 fprintf(DBG,"aaRl[0].EJ[%d]=%13.6e ",aah-1,aaRl[0].EJ[aah-1]);
1842 fprintf(DBG,"aaRh[0].CJ[%d]=%13.6e ",aag,aaRh[0].CJ[aag]);
1843 fprintf(DBG,"aaRl[0].CJ[%d]=%13.6e\n",aah-1,aaRl[0].CJ[aah-1]);
1844 fprintf(DBG,"aavfcf=%13.6e aatrprob=%13.6e\n",\
1845     aavfcf,aatrprob);
1846 fflush(DBG);
1847 }
1848           }
1849 if(aadebugflag<DEBUG){
1850 fprintf(DBG, "aaT[0].S[aaff].f[aag*3+2] = %13.6e ",
     aaT[0].S[aaff].f[aag*3+2]);
1851 fprintf(DBG, "aaT[0].S[aaff].f[aag*3+1] = %13.6e ",
     aaT[0].S[aaff].f[aag*3+1]);
1852 fprintf(DBG, "aaT[0].S[aaff].f[aag*3+0] = %13.6e\n",
     aaT[0].S[aaff].f[aag*3+0]);
```

```
1853 fprintf(DBG, "aaT[0].S[aaff].ci[aag*3+2] = %13.6e ",
     aaT[0].S[aaff].ci[aag*3+2]);
1854 fprintf(DBG, "aaT[0].S[aaff].ci[aag*3+1] = %13.6e ",
     aaT[0].S[aaff].ci[aag*3+1]);
1855 fprintf(DBG, "aaT[0].S[aaff].ci[aag*3+0] = %13.6e\n",
     aaT[0].S[aaff].ci[aag*3+0]);
1856 fprintf(DBG,"aaRl[0].CJ[aah+1]=%13.6e ",aaRl[0].CJ[aah+1]);
1857 fprintf(DBG,"aaRl[0].CJ[aah+0]=%13.6e ",aaRl[0].CJ[aah+0]);
1858 fprintf(DBG,"aaRl[0].CJ[aah-1] =%13.6e\n",aaRl[0].CJ[aah-1]);
1859 fflush(DBG);
1860 }
1861 fprintf(AAFILE,\
1862   "%5.1f\t %18.10e\t %18.10e\t %18.10e\t %18.10e\t %18.10e\t
     %18.10e\n",\
1863   aaRh[0].J[aag],\
1864   aaT[0].S[aaff].f[aag*3+2],aaT[0].S[aaff].ci[aag*3+2],\
1865   aaT[0].S[aaff].f[aag*3+1],aaT[0].S[aaff].ci[aag*3+1],\
1866   aaT[0].S[aaff].f[aag*3+0],aaT[0].S[aaff].ci[aag*3+0]);
1867            }
1868         }
1869       }
1870     }
1871     }
1872   }
1873   }
1874 return;
1875 }

      /***************** b-b *****************/
1876
      /* This function calls other functions that calculate transitions
        between Hund's Case (b) and Hund's Case (b) */

1877 void b_b(int bba, int bbb){
      /* indexes and dummy variables: */
1878 int bbc=0,bbd=0,bbe=0,bbf=0,bbff=0,bbk=0,bbdum=0;
      /* for variables below: n="native"; c="cascade"; St="state";
        Cb="Case b"; v="vibrational"; num="number"; des="designation";
        hi="high"; lo="low"; max="maximum"; pec="pseudo-Einstein coef-
        ficient"; comp="comparison"; a,b,<etc> are indexes */
      /* variables about vibrational levels */
1879 int bbhvnlo=0, bbhvnnum=0, bbhvclo=0, bbhvcnum=0, bbhivlo=0, bb-
     hivnum=0, bbHV=0;
1880 int bblvnlo=0, bblvnnum=0, bblvclo=0, bblvcnum=0, bblovlo=0,
     bblovnum=0, bbLV=0;
      /* variables for info about the two states */
1881 int bbStlo=0, bbSthi=0, bbStnlovlo=30, bbStnlovhi=0, bbSt-
     clovlo=30, bbStclovhi=0;
1882 int bbCbhi=0,bbCblo=0;
1883 double bbhipop=0,bbDe=0,bbBvDv=0;
      /* variables related to which v-v transitions need to be included */
1884 double bbpecnmax=0,bbpecncompb=0,bbpeccmax=0,bbpecccompb=0;
      /* variables to use for relating J and K */
1885 int bbHiJnum=0,bbLoJnum=0,bbmaxhiK=0,bbtransperK=0;
```

```
1886 double bbSh=0,bbSl=0;
     /* variables for the spectrscopic constants */
1887 double bbwexehi=0,bbwexelo=0,bbTehi=0,bbTelo=0,bbwehi=0,bbwelo=0;
     /* some local pointers to use as abbreviations for the longer ad-
        dresses */
1888 Case_b_stateinfo *bbHI,*bbLO;
1889 Trans *bbT;
     /* a few flags for various purposes */
1890 int bbflg=0;
     /* a set of info for transferring to the sub-transition functions */
1891 BBTinfo bbinfo;

     /* Check FCF's for lower-state vibrational levels. Find highest and
        lowest lower-state levels that wil be populated (within the user-
        set lower precision limit). Check to see that there are energies
        for all those levels. If not, reallocate the state info to have
        room for the new levels. Don't add "native" population -- only
        cascade. (the added population will be ignored if it's a destina-
        tion state only) */

     /* a whole lot of the stuff below is redundant. That's because I
        copied this function from a-a.c and it takes less time to be re-
        dundant than to make sure I changed everything correctly */
1892 bbSthi=MOL[bba].t[bbb].Hi;
1893 bbStlo=MOL[bba].t[bbb].Lo;
1894 bbinfo.T=&MOL[bba].t[bbb];
1895 bbCbhi=MOL[bba].s[bbSthi].Cb;
1896 bbCblo=MOL[bba].s[bbStlo].Cb;
1897 bbinfo.Sh=&MOL[bba].s[bbSthi];
1898 bbinfo.Sl=&MOL[bba].s[bbStlo];
1899 bbLO=&CB[bbCblo];
1900 bbHI=&CB[bbCbhi];
1901 bbinfo.Cl=bbLO;
1902 bbinfo.Ch=bbHI;
1903 bbinfo.m=bba;
1904 bbhipop=MOL[bba].s[bbSthi].pop;  /* upper state relative population
        */
1905 bbhvnlo=MOL[bba].s[bbSthi].v[0].vlo;  /* user-specified low vib
        number */
1906 bbhvnnum=MOL[bba].s[bbSthi].v[0].vnum;  /* user-specified number
        of vib levels */
1907 bbhvclo=MOL[bba].s[bbSthi].v[0].vclo;  /* low vib level from cas-
        cade */
1908 bbhvcnum=MOL[bba].s[bbSthi].v[0].vcnum;  /* number of vib levels
        from cascade */
1909 bblvnlo=MOL[bba].s[bbStlo].v[0].vlo;  /* user-specified low vib
        number */
1910 bblvnnum=MOL[bba].s[bbStlo].v[0].vnum;  /* user-specified number
        of vib levels */
1911 bblvclo=MOL[bba].s[bbStlo].v[0].vclo;  /* low vib level from cas-
        cade */
1912 bblvcnum=MOL[bba].s[bbStlo].v[0].vcnum;  /* number of vib levels
        from cascade */
1913 bbinfo.Vh=MOL[bba].s[bbSthi].v;
```

```
1914 bbinfo.hvnlo=bbhvnlo;
1915 bbinfo.hvnnum=bbhvnnum;
1916 bbinfo.hvclo=bbhvclo;
1917 bbinfo.hvcnum=bbhvcnum;
1918 bbinfo.lvnlo=bblvnlo;
1919 bbinfo.lvnnum=bblvnnum;
1920 bbinfo.lvclo=bblvclo;
1921 bbinfo.lvcnum=bblvcnum;
1922 bbwexehi=bbHI[0].wexe;
1923 bbwexelo=bbLO[0].wexe;
1924 bbTehi=bbHI[0].Te;
1925 bbTelo=bbLO[0].Te;
1926 bbwehi=bbHI[0].we;
1927 bbwelo=bbLO[0].we;
1928 if(bbdebugflag<DEBUG){
1929 fprintf(DBG,"bbSthi=%d, bbStlo=%d, bbCbhi=%d, bbCblo=%d, bb-
     hipop=%f\n",bbSthi,\
1930     bbStlo,bbCbhi,bbCblo,bbhipop);
1931 fprintf(DBG,"bbhvnlo=%d, bbhvnnum=%d, bbhvclo=%d, bbhvc-
     num=%d\n",bbhvnlo,\
1932     bbhvnnum,bbhvclo,bbhvcnum);
1933 fprintf(DBG,"bblvnlo=%d, bblvnnum=%d, bblvclo=%d, bblvc-
     num=%d\n",bblvnlo,\
1934     bblvnnum,bblvclo,bblvcnum);
1935 fprintf(DBG,"bbinfo.Sh[0].Cb=%d, bbinfo.Sl[0].Cb=%d\n",\
1936     bbinfo.Sh[0].Cb,bbinfo.Sl[0].Cb);
1937 fprintf(DBG,"Molecule is %s; Transition %s --> %s\n",MOL[bba].Mol,\
1938     MOL[bba].s[bbSthi].Name,MOL[bba].s[bbStlo].Name);
1939 fprintf(DBG,"MOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
1940     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
1941 fflush(DBG);
1942 }

      /* Find the number of J's per (most) K(s) */
1943 bbHiJnum=(int)(2*CB[bbCbhi].S+1.000000001); /* hail, hail, o great
       Numera / may we not have a truncation error :-) */
1944 bbLoJnum=(int)(2*CB[bbCblo].S+1.000000001);
1945 bbinfo.nhJ=bbHiJnum;
1946 bbinfo.nlJ=bbLoJnum;
1947 bbSh=CB[bbCbhi].S;
1948 bbSl=CB[bbCblo].S;
1949 if(bbdebugflag<DEBUG){
1950 fprintf(DBG,"bbSh=%.1f, bbsL=%.1f\n",bbSh,bbSl);
1951 fflush(DBG);
1952 }
      /* Find the maximum number of transitions to expect for each value
         of K. There must be some general form to use for this, but I spent
         20 minutes working on it and decided I could just let the program
         calculate it by brute force before I could work out the general
         form. This is necessary to get the relative intensities right,
         but will be crucial if anyone ever adds any fine splitting ef-
         fects. See the documentation on transitions for more info. */
```

```
      /* NOTE::: This is a neat idea, but I'm running low on time and need
         to get this done. So, I'll be being lazy a little bit further
         down. Scan for "Laziness:". */
1953 bbtransperK=0;
1954 for(bbc=0;bbc<bbHiJnum;bbc++){
      /* for K to K-1 transitions: */
1955   if(((-1-bbSl)<(-bbSh+bbc))&&((bbSl-1)>(-bbSh+bbc))) bb-
     transperK+=3;
1956   if(((-1-bbSl)==(-bbSh+bbc))&&((bbSl-1)>(-bbSh+bbc))) bb-
     transperK+=2;
1957   if(((-1-bbSl)<(-bbSh+bbc))&&((bbSl-1)==(-bbSh+bbc))) bb-
     transperK+=2;
1958   if((bbSl==0)&&((-bbSh+bbc)>=-2)&&((-bbSh+bbc)<=0)) bb-
     transperK+=1;
1959 if(bbdebugflag<DEBUG){
1960 fprintf(DBG,"after Delta-K=-1 for bbc=%d,
     bbtransperK=%d\n",bbc,bbtransperK);
1961 fflush(DBG);
1962 }
      /* for K to K transitions: */
1963 if((bbHI[0].L!=0)||(bbLO[0].L!=0)){ /* only if not Sigma-Sigma */
1964   if(((-bbSl)<(-bbSh+bbc))&&((bbSl)>(-bbSh+bbc))) bbtransperK+=3;
1965   if(((-bbSl)==(-bbSh+bbc))&&((bbSl)>(-bbSh+bbc))) bb-
     transperK+=2;
1966   if(((-bbSl)<(-bbSh+bbc))&&((bbSl)==(-bbSh+bbc))) bb-
     transperK+=2;
1967   if((bbSl==0)&&((-bbSh+bbc)>=-1)&&((-bbSh+bbc)<=1)) bb-
     transperK+=1;
1968   }
1969 if(bbdebugflag<DEBUG){
1970 fprintf(DBG,"after Delta-K=0 for bbc=%d,
     bbtransperK=%d\n",bbc,bbtransperK);
1971 fflush(DBG);
1972 }
      /* for K to K+1 transitions: */
1973   if(((+1-bbSl)<(-bbSh+bbc))&&((bbSl+1)>(-bbSh+bbc))) bb-
     transperK+=3;
1974   if(((+1-bbSl)==(-bbSh+bbc))&&((bbSl+1)>(-bbSh+bbc))) bb-
     transperK+=2;
1975   if(((+1-bbSl)<(-bbSh+bbc))&&((bbSl+1)==(-bbSh+bbc))) bb-
     transperK+=2;
1976   if((bbSl==0)&&((-bbSh+bbc)>=0)&&((-bbSh+bbc)<=2)) bb-
     transperK+=1;
1977 if(bbdebugflag<DEBUG){
1978 fprintf(DBG,"after Delta-K=+1 for bbc=%d,
     bbtransperK=%d\n",bbc,bbtransperK);
1979 fflush(DBG);
1980 }
1981   }
1982 bbinfo.nT=bbtransperK;
      /* Laziness: If the business above worked in all cases, the fol-
         lowing line wouldn't be necessary. But, this way, there is def-
         initely enough space in the array. In some cases, there will be
         far too much. */
```

```
1983 bbinfo.nT=bbtransperK=(int)((2*bbSh+1)*3*(2*bbSl+1));
1984 if(bbdebugflag<DEBUG){
1985 fprintf(DBG,"The Lazy bbtransperK (bbinfo.nT) is
     %d\n",bbtransperK);
1986 fflush(DBG);
1987 }

      /* find total number of sub-transitions (v-v) for this transition.
         This will take some doing. After, allocate memory for them */
1988 if((bbhvnlo<bbhvclo)||(bbhvcnum==0)) bbhivlo=bbhvnlo;
1989 else bbhivlo=bbhvclo;
1990 if((bbhvnlo+bbhvnnum)>(bbhvclo+bbhvcnum)){
1991   bbhivnum=bbhvnlo+bbhvnnum-bbhivlo;
1992   }
1993 else{
1994   bbhivnum=bbhvclo+bbhvcnum-bbhivlo;
1995   }
1996 MOL[bba].t[bbb].vnh=bbhivnum;
1997 MOL[bba].t[bbb].vhlo=bbhivlo;
1998 bbinfo.hivnum=bbhivnum;
1999 bbinfo.hivlo=bbhivlo;
2000 if(bbdebugflag<DEBUG){
2001 fprintf(DBG,"bbhvnlo=%d, bbhvclo=%d ",bbhvnlo,bbhvclo);
2002 fprintf(DBG,"(bbhvnlo+bbhvnnum)=%d, (bbhvclo+bbhvcnum)=%d\n",\
2003     (bbhvnlo+bbhvnnum),(bbhvclo+bbhvcnum));
2004 fprintf(DBG,"bbhivnum=%d, bbhivlo=%d\n",bbhivnum,bbhivlo);
2005 fprintf(DBG,"bbhivnum is %d, bbhivlo is
     %d\n",MOL[bba].t[bbb].vnh,\
2006     MOL[bba].t[bbb].vhlo);
2007 fprintf(DBG,"MOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2008     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2009 fflush(DBG);
2010 }
      /* loop to find maximum PEC (=FCF times FREQ^DETECTTYPE) */
2011 bbHV=bbhvnnum+bbhvnlo;
2012 if(bbdebugflag<DEBUG){
2013 fprintf(DBG,"bbc=%d, bbd=%d, bbHV=%d\n",bbc,bbd,bbHV);
2014 fflush(DBG);
2015 }
2016 for(bbe=bbhvnlo;bbe<bbHV;bbe++){ /* find max pec for native v's */
2017   for(bbf=0;bbf<30;bbf++){
2018     if(MOL[bba].t[bbb].v[bbe].fcfn[bbf]>bbpecnmax){
2019       bbpecnmax=MOL[bba].t[bbb].v[bbe].fcfn[bbf];
2020       }
2021 if(bbdebugflag<(DEBUG-1)){
2022 fprintf(DBG,"bbpecnmax=%11.4e\n",bbpecnmax);
2023 fflush(DBG);
2024 }
2025     }
2026   }
2027 bbHV=bbhvcnum+bbhvclo;
2028 if(bbdebugflag<(DEBUG-1)){
2029 fprintf(DBG,"Out of find-max-pec for native v's. bbHV=%d\n",bbHV);
```

```
2030 fflush(DBG);
2031 }
2032 for(bbe=bbhvclo;bbe<bbHV;bbe++){ /* find max pec for cascade v's */
2033   for(bbf=0;bbf<30;bbf++){
2034     if(MOL[bba].t[bbb].v[bbe].fcfc[bbf]>bbpeccmax){
2035       bbpeccmax=MOL[bba].t[bbb].v[bbe].fcfc[bbf];
2036       }
2037 if(bbdebugflag<(DEBUG-1)){
2038 fprintf(DBG,"bbpeccmax=%11.4e\n",bbpeccmax);
2039 fflush(DBG);
2040 }
2041     }
2042   }
      /* find minimum and maximum lower-state v's with PEC's that are
         above the low-intensity cutoff (JKCUT) */
2043 bbHV=bbhvnnum+bbhvnlo;
2044 if(bbdebugflag<DEBUG){
2045 fprintf(DBG,"In cut-off loop: bbc=%d, bbd=%d,
     bbHV=%d\n",bbc,bbd,bbHV);
2046 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2047     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2048 fflush(DBG);
2049 }
2050 for(bbe=bbhvnlo;bbe<bbHV;bbe++){ /* for native populations */
2051   bbf=0;
2052 if(bbdebugflag<(DEBUG-1)){
2053 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2054     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2055 fflush(DBG);
2056 }
2057   while(bbflg==0){
2058     bbpecncompb=MOL[bba].t[bbb].v[bbe].fcfn[bbf]/JKCUT;
2059     if(bbpecncompb<bbpecnmax) bbdum=bbf;
2060     else bbflg=1;
2061     bbf++;
2062     }
2063   if(bbdum<bbStnlovlo) bbStnlovlo=bbdum;
2064 if(bbdebugflag<(DEBUG-1)){
2065 fprintf(DBG,"bbpecncompb=%11.4e, bbf=%d\n",bbpecncompb,bbf);
2066 fprintf(DBG,"bbStnlovlo=%d\n",bbStnlovlo);
2067 fflush(DBG);
2068 }
2069   bbf=29;
2070   bbflg=0;
2071 if(bbdebugflag<(DEBUG-1)){
2072 fflush(DBG);
2073 }
2074   while(bbflg==0){
2075     bbpecncompb=MOL[bba].t[bbb].v[bbe].fcfn[bbf]/JKCUT;
2076     if(bbpecncompb<bbpecnmax) bbdum=bbf;
2077     else bbflg=1;
2078 if(bbdebugflag<(DEBUG-1)){
```

```
2079 fprintf(DBG,"bbStnlovhi loop: bbpecncompb=%11.4e , ",bbpecncompb);
2080 fprintf(DBG,"bbf=%d , bbdum=%d , bbflg=%d\n",bbf,bbdum,bbflg);
2081 fflush(DBG);
2082 }
2083     bbf--;
2084     }
2085   if(bbdum>bbStnlovhi) bbStnlovhi=bbdum;
2086 if(bbdebugflag<(DEBUG-1)){
2087 fprintf(DBG,"bbpecncompb=%11.4e, bbf=%d,
     bbdum=%d\n",bbpecncompb,bbf,bbdum);
2088 fprintf(DBG,"bbStnlovhi=%d\n",bbStnlovhi);
2089 fflush(DBG);
2090 }
2091   }
2092 bbHV=bbhvcnum+bbhvclo;
2093 if(bbdebugflag<(DEBUG-1)){
2094 fprintf(DBG,"out of native population cutoff. bbHV=%d\n",bbHV);
2095 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2096     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2097 fflush(DBG);
2098 }
2099 for(bbe=bbhvclo;bbe<bbHV;bbe++){ /* for cascade populations */
2100   bbf=0;
2101 if(bbdebugflag<(DEBUG-1)){
2102 fflush(DBG);
2103 }
2104   while(bbflg==0){
2105     bbpecccompb=MOL[bba].t[bbb].v[bbe].fcfc[bbf]/JKCUT;
2106     if(bbpecccompb<bbpeccmax) bbdum=bbf;
2107     else bbflg=1;
2108     bbf++;
2109     }
2110 if(bbdebugflag<(DEBUG-1)){
2111 fprintf(DBG,"bbpecccompb=%11.4e, bbf=%d,
     bbdum=%d\n",bbpecccompb,bbf,bbdum);
2112 fflush(DBG);
2113 }
2114   if(bbdum<bbStclovlo) bbStclovlo=bbdum;
2115   bbf=29;
2116   bbflg=0;
2117   while(bbflg==0){
2118     bbpecccompb=MOL[bba].t[bbb].v[bbe].fcfc[bbf]/JKCUT;
2119     if(bbpecccompb<bbpeccmax) bbdum=bbf;
2120     else bbflg=1;
2121     bbf--;
2122     }
2123 if(bbdebugflag<(DEBUG-1)){
2124 fprintf(DBG,"bbpecccompb=%11.4e, bbf=%d,
     bbdum=%d\n",bbpecccompb,bbf,bbdum);
2125 fflush(DBG);
2126 }
2127   if(bbdum>bbStclovhi) bbStclovhi=bbdum;
```

```
2128   }
2129 if(bbdebugflag<(DEBUG-1)){
2130 fprintf(DBG,"out of cutoff loop\n\n");
2131 fprintf(DBG,"MOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2132     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2133 fflush(DBG);
2134 }

      /* determine the lowest necessary destination-state vib level and
         the number of destination-state vibration levels */
2135 if((bbStnlovlo)<(bbStclovlo)) bblovlo=bbStnlovlo;
2136 else bblovlo=bbStclovlo;
2137 if((bbStnlovhi)>(bbStclovhi)) bblovnum=bbStnlovhi+1-bblovlo;
2138 else bblovnum=bbStclovhi+1-bblovlo;
2139 MOL[bba].t[bbb].vnl=bblovnum;
2140 MOL[bba].t[bbb].vllo=bblovlo;
2141 bbinfo.lovnum=bblovnum;
2142 bbinfo.lovlo=bblovlo;
      /* allocate memory for the simsets in the transition structure */
2143 MOL[bba].t[bbb].nS=bbhivnum*bblovnum;
2144 MOL[bba].t[bbb].S = (simset*) calloc(MOL[bba].t[bbb].nS,
     sizeof(simset));
2145 MOL[bba].t[bbb].fS=(int*)calloc(MOL[bba].t[bbb].nS,sizeof(int));
2146 if(bbdebugflag<DEBUG){
2147 fprintf(DBG,"MOL[%d].t[%d].vnl=%d, MOL[%d].t[%d].vllo=%d
     ",bba,bbb,\
2148     MOL[bba].t[bbb].vnl,bba,bbb,MOL[bba].t[bbb].vllo);
2149 fprintf(DBG,"MOL[%d].t[%d].nS=%d\n",bba,bbb,MOL[bba].t[bbb].nS);
2150 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2151     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2152 fflush(DBG);
2153 }

2154 bbHV=bbhvnnum+bbhvnlo;
      /* get maximum K value present */
2155 for(bbf=bbhvnlo;bbf<bbHV;bbf++){ /* check native populations */
2156 if(bbdebugflag<(DEBUG)){
2157 fprintf(DBG,"In native population loop \n");
2158 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2159     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2160 fflush(DBG);
2161 }
2162   bbk=MOL[bba].s[bbSthi].r[bbf-bbhvnlo].k;
2163   if(bbk>bbmaxhiK){
2164     bbmaxhiK=bbk;
2165     }
2166   }
2167 if(bbdebugflag<(DEBUG)){
2168 fprintf(DBG,"Native: bbe=%d, bbf=%d,
     bbmaxhiK=%d\n",bbe,bbf,bbmaxhiK);
```

```
2169 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2170     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2171 fflush(DBG);
2172 }
2173 bbHV=bbhvcnum+bbhvclo;
2174 for(bbf=bbhvclo;bbf<bbHV;bbf++){ /* check cascade populations */
2175   bbk=MOL[bba].s[bbSthi].rc[bbf-bbhvclo].k;
2176   if(bbk>bbmaxhiK){
2177     bbmaxhiK=bbk;
2178     }
2179   }
2180 if(bbdebugflag<(DEBUG)){
2181 fprintf(DBG,"Cascade: bbe=%d, bbf=%d,
     bbmaxhiK=%d\n",bbe,bbf,bbmaxhiK);
2182 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2183     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2184 fflush(DBG);
2185 }

      /* See if all the needed destination-state v's are calculated.
         Check also that enough K/J's are calculated. Calculate any states
         that aren't already done. The following method isn't efficient.
         It will calculate more lower-state energy levels than are actu-
         ally needed. But, this shouldn't be a problem other than doing a
         few unnecessary calculations. */
2186 bbinfo.hiK=bbmaxhiK;
2187 bbmaxhiK++;  /* because of the Delta-K = +1 possibility */
2188 if(bbdebugflag<(DEBUG)){
2189 fprintf(DBG,"Before new calculations loop
     bbmaxhiK=%d\n",bbmaxhiK);
2190 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2191     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2192 fflush(DBG);
2193 }
      /* Calculate any newly needed energy levels. */
      /* loop through each lower Omega */
2194 bbLV=bblovlo+bblovnum;
2195 if(bbdebugflag<(DEBUG)){
2196 fprintf(DBG,"bbd=%d, bbLV=%d\n",bbd,bbLV);
2197 fflush(DBG);
2198 }
2199 for(bbf=bblovlo;bbf<bbLV;bbf++){ /* check low-state v-levels */
2200   bbff=bbd*30+bbf; /* Index for the array of rc rotational sets */
2201 if(bbdebugflag<(DEBUG)){
2202 fprintf(DBG,"bbf=%d, bbff=%d, bbdum=%d\n",bbf,bbff,bbdum);
2203 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2204     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2205 fflush(DBG);
2206 }
2207   if(MOL[bba].s[bbStlo].rc[bbff].kc==0){
```

```
      /* this v has never been calculated, so call a function to calculate
        it. */
2208   if(CB[bbCblo].we!=0){
2209     bbDe=4*pow(CB[bbCblo].Be,3)/(CB[bbCblo].we*CB[bbCblo].we);
2210     bbBvDv=(CB[bbCblo].Be-CB[bbCblo].ae*(bbf+0.5))/\
2211       (2*(bbDe+CB[bbCblo].beta*(bbf+0.5)));
2212     MOL[bba].s[bbStlo].rc[bbff].Kdissoc=\
2213       floor((sqrt(1+4*bbBvDv)-1)/2);
2214     }
2215   else MOL[bba].s[bbStlo].rc[bbff].Kdissoc=RAND_MAX;
2216     if(bbmaxhiK<MOL[bba].s[bbStlo].rc[bbff].Kdissoc){
2217       case_b_cascade(&MOL[bba].s[bbStlo].rc[bbff],\
2218         bbCblo,bbf,0,bbmaxhiK,bbinfo.nlJ);
2219       }
2220     else{
2221       case_b_cascade(&MOL[bba].s[bbStlo].rc[bbff],bbCblo,\
2222       bbf,0,(int)(MOL[bba].s[bbStlo].rc[bbff].Kdissoc+1),\
2223       bbinfo.nlJ);
2224       }
2225 if(bbdebugflag<(DEBUG)){
2226 fprintf(DBG,"just called case_b_cascade, v never calc'd\n");
2227 fprintf(DBG,"CB[%d].Be=%12.6e; CB[%d].we=%12.6e;
     CB[%d].ae=%12.6e\n",\
2228
     bbCblo,CB[bbCblo].Be,bbCblo,CB[bbCblo].we,bbCblo,CB[bbCblo].ae);
2229 fprintf(DBG,"bbf=%d, bbDe=%12.6e, bbBvDv=%12.6e,
     ",bbf,bbDe,bbBvDv);
2230 fprintf(DBG, "MOL[%d].s[%d].rc[%d].Jdissoc = %12.6e\n", bba, bb-
     Stlo, bbff,\
2231     MOL[bba].s[bbStlo].rc[bbff].Kdissoc);
2232 fflush(DBG);
2233 fprintf(DBG,"j was zero; just called cascade \n");
2234 fprintf(DBG,"MOL[%d].s[%d].rc[%d].kc=%d\n",bba,bbStlo,bbff,\
2235     MOL[bba].s[bbStlo].rc[bbff].kc);
2236 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s, MOL[%d].t[%d].Nlo = %s\n",
     bba, bbb,\
2237     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2238 fflush(DBG);
2239 }
2240     }
2241   else{
      /* This v has been calculated. See if enough J's are calculated */
2242     if(MOL[bba].s[bbStlo].rc[bbff].kc<bbmaxhiK){
2243     if(bbmaxhiK<MOL[bba].s[bbStlo].rc[bbff].Kdissoc){
2244 if(bbdebugflag<(DEBUG)){
2245 fprintf(DBG,"j<maxJ; bbmaxhiK < Kdissoc; about to call cascade
     \n");
2246 fprintf(DBG,"bbmaxhiK=%d;
     MOL[%d].s[%d].rc[%d].Kdissoc=%12.6e\n",\
2247   bbmaxhiK,bba,bbStlo,bbff,MOL[bba].s[bbStlo].rc[bbff].Kdissoc);
2248 fprintf(DBG,"MOL[%d].s[%d].rc[%d].kc=%d\n",bba,bbStlo,bbff,\
2249     MOL[bba].s[bbStlo].rc[bbff].kc);
2250 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
     MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
```

```
2251     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2252 fflush(DBG);
2253 }
2254       case_b_cascade(&MOL[bba].s[bbStlo].rc[bbff],bbCblo,\
2255         bbf,MOL[bba].s[bbStlo].rc[bbff].kc,\
2256         bbmaxhiK,bbinfo.nlJ);
2257       }
2258     else{
2259 if(bbdebugflag<(DEBUG)){
2260 fprintf(DBG,"j<maxJ; about to call cascade \n");
2261 fprintf(DBG,"j<maxJ; bbmaxhiK > Kdissoc; about to call cascade
    \n");
2262 fprintf(DBG,"bbmaxhiK=%d;
    MOL[%d].s[%d].rc[%d].Kdissoc=%12.6e\n",\
2263   bbmaxhiK,bba,bbStlo,bbff,MOL[bba].s[bbStlo].rc[bbff].Kdissoc);
2264 fprintf(DBG,"MOL[%d].s[%d].rc[%d].kc=%d\n",bba,bbStlo,bbff,\
2265     MOL[bba].s[bbStlo].rc[bbff].kc);
2266 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
    MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2267     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2268 fflush(DBG);
2269 }
2270       case_b_cascade(&MOL[bba].s[bbStlo].rc[bbff],bbCblo,\
2271       bbf,MOL[bba].s[bbStlo].rc[bbff].kc,\
2272     (int)(MOL[bba].s[bbStlo].rc[bbff].Kdissoc+1),bbinfo.nlJ);
2273       }
2274 if(bbdebugflag<(DEBUG)){
2275 fprintf(DBG,"j was <maxJ; just called cascade \n");
2276 fprintf(DBG,"MOL[%d].s[%d].rc[%d].kc=%d\n",bba,bbStlo,bbff,\
2277     MOL[bba].s[bbStlo].rc[bbff].kc);
2278 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
    MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2279     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2280 fflush(DBG);
2281 fprintf(DBG,"just called case_b_cascade for cascade only\n");
2282 fflush(DBG);
2283 }
2284       }
2285     }
2286   }
     /* Reset values of vclo and vcnum for the next transition */
2287   MOL[bba].s[bbStlo].v[0].vclo=bblovlo;
2288   MOL[bba].s[bbStlo].v[0].vcnum=bblovnum;
2289 if(bbdebugflag<(DEBUG)){
2290 fprintf(DBG,"MOL[%d].s[%d].v[%d].vclo=%d ",bba,bbStlo,0,\
2291     MOL[bba].s[bbStlo].v[0].vclo);
2292 fprintf(DBG,"MOL[%d].s[%d].v[%d].vcnum=%d\n",bba,bbStlo,0,\
2293     MOL[bba].s[bbStlo].v[0].vcnum);
2294 fprintf(DBG,"\tMOL[%d].t[%d].Nhi=%s,
    MOL[%d].t[%d].Nlo=%s\n",bba,bbb,\
2295     MOL[bba].t[bbb].Nhi,bba,bbb,MOL[bba].t[bbb].Nlo);
2296 fflush(DBG);
2297 }
```

```
      /* Check whether larger selection rules (Lambda, S, +/- in Sigma
         states) are violated. If they are, write a note to that effect to
         the DBG file. */
2298 if(bbdebugflag<(DEBUG)){
2299 fprintf(DBG,"bbHI[0].L=%d, bbLO[0].L=%d\n",bbHI[0].L,bbLO[0].L);
2300 fflush(DBG);
2301 }
2302 if((abs(bbHI[0].L-bbLO[0].L))>1){
2303   fprintf(PAR,"WARNING!!! Normal selection rule violated for
     Delta-");
2304   fprintf(PAR,"Lambda=0,+/-1 for\n\tTransition %s-->%s\n",\
2305       MOL[bba].t[bbb].Nhi,MOL[bba].t[bbb].Nlo);
2306   fprintf(PAR,"\tThis is non-fatal; only a warning.\n");
2307 if(bbdebugflag<(DEBUG)){
2308 fprintf(DBG,"Delta-L selection rule violated.\n");
2309 fflush(DBG);
2310 }
2311   }
2312 if((fabs(bbHI[0].S-bbLO[0].S))>0.01){ /* <<< an overkill against
       round-off */
2313   fprintf(PAR,"WARNING!!! Normal selection rule violated for
     Delta-");
2314   fprintf(PAR,"S=0 for\n\tTransition %s-->%s\n",\
2315       MOL[bba].t[bbb].Nhi,MOL[bba].t[bbb].Nlo);
2316   fprintf(PAR,"\tThis is non-fatal; only a warning.\n");
2317 if(bbdebugflag<(DEBUG)){
2318 fprintf(DBG,"Delta-S selection rule violated\n");
2319 fflush(DBG);
2320 }
2321   }
      /* loop through vibration levels starting with the highest in
         case anyone wants to include cascade between v-levels one day.
         Same for J's and K's later. Certainly *someone* will want to do
         that... */
2322 bbT=&MOL[bba].t[bbb];  /* an abbreviation */
2323 if(bbdebugflag<DEBUG){
2324 fprintf(DBG,"bba=%d, bbb=%d,
     MOL[bba].t[bbb].Nlo=%s\n",bba,bbb,bbT[0].Nlo);
2325 fprintf(DBG,"bbT.Nhi=%s, bbT.Nlo=%s\n",bbT[0].Nhi,bbT[0].Nlo);
2326 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vclo=%d ",bba,bbSthi,\
2327     MOL[bba].s[bbSthi].v[0].vclo);
2328 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vcnum=%d\n",bba,bbSthi,\
2329     MOL[bba].s[bbSthi].v[0].vcnum);
2330 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vlo=%d ",bba,bbSthi,\
2331     MOL[bba].s[bbSthi].v[0].vlo);
2332 fprintf(DBG,"High: MOL[%d].s[%d].v[0].vnum=%d\n",bba,bbSthi,\
2333     MOL[bba].s[bbSthi].v[0].vnum);
2334 fprintf(DBG,"bbhivnum=%d; bblovnum=%d\n",bbhivnum,bblovnum);
2335 fprintf(DBG,"bbHI[0].L=%d, bbLO[0].L=%d\n",bbHI[0].L,bbLO[0].L);
2336 fflush(DBG);
2337 }
      /* call other functions to calculate transitions for various sce-
         narios */
2338 if((bbHI[0].L==0)&&(bbLO[0].L==0)){ /* If Sigma to Sigma */
```

```
2339   if(bbHI[0].p!=bbLO[0].p){ /* For the +<->+, -<->- rule */
2340 fprintf(PAR,"\n\nTRANSITION OMITTED!!!!!!!!! (see below)\n\n");
2341 fprintf(PAR,"Transition called for Sigma(%d)-->Sigma(%d).\n",\
2342     bbHI[0].p,bbLO[0].p);
2343 fprintf(PAR,"Molecule is %s; Transition %s-->%s\n",MOL[bba].Mol,\
2344     bbT[0].Nhi,bbT[0].Nlo);
2345 fprintf(PAR,"This program doesn't know how to break that selection
     rule.\n");
2346 fprintf(PAR,"The program will continue with this transition omit-
     ted.\n\n");
2347 fflush(PAR);
2348     }
2349   else{ /* check g-u rule if homonuclear and call function */
2350     if((bbHI[0].g!=0)&&(bbHI[0].g==bbLO[0].g)){
2351 fprintf(PAR,"\n\nTRANSITION OMITTED!!!!!!!!! (see below)\n\n");
2352 fprintf(PAR,"Transition called for two states that are both either
     ");
2353 fprintf(PAR,"gerade or ungerade.\n");
2354 fprintf(PAR,"Molecule is %s; Transition %s-->%s\n",MOL[bba].Mol,\
2355     bbT[0].Nhi,bbT[0].Nlo);
2356 fprintf(PAR,"This program doesn't know how to break that selection
     rule.\n");
2357 fprintf(PAR,"The program will continue with this transition omit-
     ted.\n\n");
2358 fflush(PAR);
2359       }
2360     else bb_SigmaSigma(bbinfo);
2361     }
2362   }
2363 else{
2364   if((bbHI[0].L==0)||(bbLO[0].L==0)) bb_SigmaOther(bbinfo);  /* if
       one or the other state is Sigma, but not both */
2365   else bb_Other(bbinfo);  /* neither state is Sigma */
2366   }
2367 return;
2368 }

     /*************** b-b_SigmaSigma *****************/
2369
     /*This function calculates transitions between Hund's Cases (b) and
       (b) when both the upper and lower states are sigma states. */

2370 void bb_SigmaSigma(BBTinfo bbSS){
     /* see parent function and header file for key to variable names */
     /* indexes and dummy variables: */
2371 int bbSSe=0,bbSSee=0,bbSSf=0,bbSSff=0;
2372 int bbSSh=0,bbSSk=0,bbSSfrnh=0,bbSSfrcl=0,bbSSfrch=0;
2373 int bbSSeh=0,bbSSi=0,bbSSehh=0,bbSSel=0,bbSSell=0,bbSSjii=0;
     /* variables for info about the two states */
2374 double
    bbSSJ=0,bbSSSh=0,bbSSSl=0,bbSStrprob=0,bbSSvhpop=0,bbSShipop=0;
     /* Intensity multiplier for satellite bands and FCF dummy variable
       */
2375 double bbSSSATT=0,bbSSfcf=0,bbSSfcfr=0;
```

```
      /* variables for calculating Holn-London factors */
2376  int bbSSL=0,bbSSB=0,bbSSdL=0;
      /* variables for spin stats if the state is homonuclear */
2377  int bbSSg=0,bbSSisym=0,bbSSpsym=0;
2378  double bbSSIa=0,bbSScascadetemp=0;
      /* a few flags for various purposes */
2379  int bbSSflga=0,bbSSflgb=0,bbSSfrnhf=0,bbSSfrchf=0;
      /* variables for creating and writing to output files */
2380  char bbSSfile[1000];
2381  FILE *BBSSFILE;
2382  Case_b_stateinfo *bbSSHI,*bbSSLO;

2383  if(bbSSdebugflag<DEBUG){
2384  fprintf(DBG,"bbSS.hvnlo=%d, bbSS.hvnnum=%d\n",\
2385      bbSS.hvnlo,bbSS.hvnnum);
2386  fprintf(DBG,"bbSS.hvclo=%d, bbSS.hvcnum=%d\n",\
2387      bbSS.hvclo,bbSS.hvcnum);
2388  fprintf(DBG,"bbSS.lvnlo=%d, bbSS.lvnnum=%d,\n",\
2389      bbSS.lvnlo,bbSS.lvnnum);
2390  fprintf(DBG,"bbSS.lvclo=%d, bbSS.lvcnum=%d\n",\
2391      bbSS.lvclo,bbSS.lvcnum);
2392  fprintf(DBG,"bbSS.hivlo=%d, bbSS.hivnum=%d,\n",\
2393      bbSS.hivlo,bbSS.hivnum);
2394  fprintf(DBG,"bbSS.lovlo=%d, bbSS.lovnum=%d\n",\
2395      bbSS.lovlo,bbSS.lovnum);
2396  fflush(DBG);
2397  }
2398  bbSSHI=bbSS.Ch;
2399  bbSSLO=bbSS.Cl;
2400  bbSSSh=bbSSHI[0].S;
2401  bbSSSl=bbSSLO[0].S;
2402  bbSSisym=bbSS.Sh[0].Isymm;
2403  bbSSpsym=bbSS.Sh[0].pmsymm;
2404  bbSShipop=bbSS.Sh[0].pop;
2405  bbSStrprob=bbSS.T[0].P[0];
      /* check for homonuclear information */
2406  if(bbSSHI[0].g!=0){
2407    bbSSg=bbSSHI[0].g;
2408    bbSSIa=bbSSHI[0].I/(bbSSHI[0].I+1);
2409    }
      /* Just to be explicit about it... */
2410  bbSSL=bbSSdL=0;  /* Upper-state Lambda is zero and Delta-Lambda is
        zero. */

      /* start loop down through high vib levels */
2411  for(bbSSe=(bbSS.hivlo+bbSS.hivnum-1);bbSSe>=bbSS.hivlo;bbSSe--){
2412    bbSSee=(bbSSe-bbSS.hivlo)*bbSS.lovnum;  /* simset posn. in 2D */
2413    bbSSfrnh=(bbSSe-bbSS.hvnlo);  /* high state native vib position
         */
2414    bbSSfrch=bbSSe;  /* high state cascade vib position */
      /* these flags (bbSSfrnhf and bbSSfrchf) tell if this high vib state
        is populated natively, by cascade, or both. They will be used
        later */
2415    if((bbSSe>=bbSS.hvnlo)&&(bbSSe<(bbSS.hvnlo+bbSS.hvnnum))){
```

```
2416      bbSSfrnhf=0;
2417      bbSSvhpop=bbSS.Vh[0].p[bbSSe-bbSS.hvnlo];
2418      }
2419    else bbSSfrnhf=-1;
2420    if((bbSSe>=bbSS.hvclo)&&(bbSSe<(bbSS.hvclo+bbSS.hvcnum))){
2421      bbSSfrchf=0;
2422      }
2423    else bbSSfrchf=-1;
2424 if(bbSSdebugflag<DEBUG){
2425 fprintf(DBG,"bbSSfrnhf=%d, bbSSvhpop=%f, bbSSfrchf=%d, ",\
2426      bbSSfrnhf,bbSSvhpop,bbSSfrchf);
2427 fprintf(DBG,"bbSSe=%d, bbSSee=%d\nbbSSfrnh=%d, bbSSfrch=%d, ",\
2428      bbSSe,bbSSee,bbSSfrnh,bbSSfrch);
2429 fflush(DBG);
2430 }
      /* start loop down through low vib levels */
2431 for(bbSSf=(bbSS.lovlo+bbSS.lovnum-1);bbSSf>=bbSS.lovlo;bbSSf--){
2432   bbSSfcf=bbSS.T[0].v[bbSSe].fcfn[bbSSf];
2433
     if(bbSSfcf!=0){bbSSfcfr=bbSS.T[0].v[bbSSe].fcfc[bbSSf]/bbSSfcf;}
2434   bbSSff=bbSSee+bbSSf-bbSS.lovlo;  /* position in last dimension */
2435   bbSSfrcl=bbSSf;
2436 if(bbSSdebugflag<DEBUG){
2437 fprintf(DBG,"bbSSf=%d, bbSS.lovlo=%d, bbSS.lovnum=%d \n",\
2438      bbSSf,bbSS.lovlo,bbSS.lovnum);
2439 fprintf(DBG,"bbSS.nT=%d, bbSS.hiK=%d \n",bbSS.nT,bbSS.hiK);
2440 fflush(DBG);
2441 }
2442   bbSS.T[0].S[bbSSff].n=bbSS.nT*bbSS.hiK;
2443   bbSS.T[0].S[bbSSff].f=\
2444     (double*)calloc(bbSS.T[0].S[bbSSff].n,sizeof(double));
2445   bbSS.T[0].S[bbSSff].ni=\
2446     (double*)calloc(bbSS.T[0].S[bbSSff].n,sizeof(double));
2447   bbSS.T[0].S[bbSSff].ci=\
2448     (double*)calloc(bbSS.T[0].S[bbSSff].n,sizeof(double));

2449 bbSS.Rl=&bbSS.Sl[0].rc[bbSSfrcl];
2450 if(bbSSdebugflag<DEBUG){
2451 fprintf(DBG,"bbSSff=%d, bbSSfrcl=%d,
     bbSS.T[0].S[bbSSff].n=%d\n",\
2452      bbSSff,bbSSfrcl,bbSS.T[0].S[bbSSff].n);
2453 fflush(DBG);
2454 }
      /* start with highest J value (same reason as before), and loop down
         looking for lower J's to which to transit. assign intensities.
         If both states are Omega=0, then assign zero intensity to transi-
         tions for J=0<->J=0. */
2455   if(bbSSfrnhf==0){ /* if this v is natively populated.*/
2456 sprintf(bbSSfile,"%s_molecules/%s/%s--%s_v%d--v%d_NAT.dat",\
2457   PREF,MOL[bbSS.m].Mol,bbSS.T[0].Nhi,bbSS.T[0].Nlo,bbSSe,bbSSf);
2458      BBSSFILE=fopen(bbSSfile,"w");
2459      if(BBSSFILE==NULL){
2460 printf("Error opening transition sub-file %s. Exit-
     ing.\n",bbSSfile);
```

```
2461       exit(1);
2462       }
2463 fprintf(BBSSFILE,"# File created by %s.\n# File contains ",PRO-
     GRAM_NAME);
2464 fprintf(BBSSFILE,"P and R");
2465 fprintf(BBSSFILE," branches from the simulation titled %s\n",PREF);
2466 fprintf(BBSSFILE,"# THESE INTENSITIES ARE DUE TO USER-SPECIFIED
     POPULATIONS");
2467 fprintf(BBSSFILE," ONLY -- NO CASCADE\n");
2468 fprintf(BBSSFILE,"# Transition is Hund's Case (b) to Hund's Case
     (b).\n");
2469 fprintf(BBSSFILE,"# Molecule %s, \tHigh state: %s, v=%d\n",\
2470     MOL[bbSS.m].Mol,bbSS.T[0].Nhi,bbSSe);
2471 fprintf(BBSSFILE,"#\t\tLow state: %s, v=%d\n# Columns are: ",\
2472     bbSS.T[0].Nlo,bbSSf);
2473 fprintf(BBSSFILE,"K_hi K_lo J_hi P(J_hi+1)_v P_i R(J_hi-1)_v
     R_i\n#\n");
2474 fflush(BBSSFILE);
2475     bbSS.Rh=&bbSS.Sh[0].r[bbSSfrnh];
2476     if((bbSS.hiK)<bbSS.Sh[0].r[bbSSfrnh].k){
2477       bbSSk=bbSS.hiK-1;
2478       }
2479     else bbSSk=bbSS.Sh[0].r[bbSSfrnh].k-1;
2480     if(bbSS.Rl[0].Kdissoc<(bbSSk-1)){
2481 fprintf(PAR,"\nWARNING!! State %s of molecule %s has significant
     ",\
2482     bbSS.T[0].Nhi,MOL[bbSS.m].Mol);
2483 fprintf(PAR,"rotational population at level %d\n",bbSSk);
2484 fprintf(PAR,"\tBut lower state %s dissociates at level %.1f\n",\
2485     bbSS.T[0].Nlo,bbSS.Rl[0].Kdissoc);
2486 fprintf(PAR,"It is likely that the bound upper state is transiting
     to (an)");
2487 fprintf(PAR," unbound lower state(s).\n");
2488 fprintf(PAR,"Look for continuum emissions trailing off to the low-
     energy ");
2489 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real ",bb-
     SSe,bbSSf);
2490 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
     gram will");
2491 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
2492       bbSSk=(int)(bbSS.Rl[0].Kdissoc);
2493       }
2494 if(bbSSdebugflag<DEBUG){
2495 fprintf(DBG,"bbSSfrnh=%d, bbSSk=%d, ",bbSSfrnh,bbSSk);
2496 fprintf(DBG, "bbSS.Sh[0].r[%d].k = %d\n", bbSSfrnh,
     bbSS.Sh[0].r[bbSSfrnh].k);
2497 fflush(DBG);
2498 }
     /* To make the output file easy to write, I'm cycling by high-state
        K first, then by high-state J. From that J/K(hi) pair, the pro-
        gram will look for a P transition and an R transition to K+1 then
        to K-1. If all the necessary states exist, it will check to see
        if there are any symmetry or nuclear spin effects to be concerned
```

```
        about. Taking these into account, it will calculate the transi-
        tion. */
      /* loop first by upper-state K-value */
2499  for(bbSSh=bbSSk;bbSSh>=0;bbSSh--){
2500    bbSSjii=(bbSSh+1)*bbSS.nT-1;  /* position in simset array */
2501    bbSSeh=bbSSh*bbSS.nhJ;  /* position in 2D for high EJ & PJ */
      /* then loop by upper-state J for this K */
2502    for(bbSSi=(bbSS.nhJ-1);bbSSi>=0;bbSSi--){
2503      bbSSJ=bbSSh+bbSSi-bbSSSh;
2504      bbSSehh=bbSSeh+bbSSi;
2505      bbSSflga=0;
2506      if(bbSSh<bbSSHI[0].L) bbSSflga=1;  /* if this K is less than
        Lambda for the high state, don't proceed */
2507      if(bbSSJ<0) bbSSflga=1;  /* if J is less than zero, don't */
2508  if(bbSSdebugflag<DEBUG){
2509  fprintf(DBG,"bbSSh=%d, bbSSi=%d, bbSSSh=%.1f, bbSSHI[0].L=%d, bb-
     SSflga=%d\n",\
2510      bbSSh,bbSSi,bbSSSh,bbSSHI[0].L,bbSSflga);
2511  fprintf(DBG,"bbSSjii=%d\n",bbSSjii);
2512  fflush(DBG);
2513  }
2514      if(bbSSflga==0){
2515        bbSSflgb=0;
      /* Check Delta-J=+1 (P) and then Delta-J=-1 (R) for Delta-K=+1.
        First, check to see if the target K exists: */
2516        if((bbSSh+1)<(bbSSLO[0].L)) bbSSflgb=1;
      /* Relative position in the low state E/P array */
2517        bbSSel=(bbSSh+1)*bbSS.nlJ + bbSS.nlJ-bbSS.nhJ + \
2518          (int)(bbSSSh-bbSSSl) -1 + bbSSi;
2519  if(bbSSdebugflag<DEBUG){
2520  fprintf(DBG,"\t(bbSSh+1)*bbSS.nlJ=%d, bbSS.nlJ-bbSS.nhJ=%d\n",\
2521      (bbSSh+1)*bbSS.nlJ,bbSS.nlJ-bbSS.nhJ);
2522  fprintf(DBG,"\t(int)(bbSSSh-bbSSSl)=%d, bbSSi=%d, bbSSel=%d\n",\
2523      (int)(bbSSSh-bbSSSl),bbSSi,bbSSel);
2524  fflush(DBG);
2525  }
2526        if((bbSSel+1)>=(bbSS.Rl[0].kc*bbSS.nlJ)) bbSSflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
2527        if((bbSSJ+1)>(bbSSh+1+bbSSSl)) bbSSflgb=1;
2528        if((bbSSJ+1)<(bbSSh+1-bbSSSl)) bbSSflgb=1;
      /* Still OK? Calculate transition */
2529        if(bbSSflgb==0){
2530          bbSSB=+1;
2531          bbSSell=bbSSel+1;
2532          bbSS.T[0].S[bbSSff].f[bbSSjii]=\
2533            bbSS.Rh[0].EJ[bbSSehh]-\
2534            bbSS.Rl[0].EJ[bbSSell];
2535  if(bbSSdebugflag<DEBUG){
2536  fprintf(DBG,"bbSS.T[0].S[%d].f[%d]=%13.5e ",bbSSff,bbSSjii,\
2537      bbSS.T[0].S[bbSSff].f[bbSSjii]);
2538  fprintf(DBG, "bbSS.Rh[0].EJ[%d] = %13.5e ", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
```

```
2539 fprintf(DBG, "bbSS.Rl[0].EJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
2540 fflush(DBG);
2541 }
2542         bbSS.T[0].S[bbSSff].ni[bbSSjii]=\
2543           bbSShipop*bbSStrprob*bbSSvhpop*\
2544           bbSS.Rh[0].PJ[bbSSehh]*bbSSfcf*\
2545           HL(bbSSB,bbSSJ,bbSSL,bbSSdL);
2546 if(bbSSdebugflag<DEBUG){
2547 fprintf(DBG,"bbSShipop=%13.5e bbSStrprob=%13.5e ",bbS-
     Shipop,bbSStrprob);
2548 fprintf(DBG,"bbSSvhpop=%13.5e
     bbSSfcf=%13.6e\n",bbSSvhpop,bbSSfcf);
2549 fprintf(DBG,"bbSS.T[0].S[%d].ni[%d]=%13.5e ",\
2550     bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2551 fprintf(DBG, "bbSS.Rh[0].PJ[%d] = %13.5e\n", bbSSehh,
     bbSS.Rh[0].PJ[bbSSehh]);
2552 fprintf(DBG, "HL(%d,%.1f,%d,%d) = %12.6e\n", bbSSB, bbSSJ, bbSSL,
     bbSSdL,\
2553     HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2554 fflush(DBG);
2555 }
2556         bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
2557           bbSS.T[0].S[bbSSff].ni[bbSSjii];
2558 if(bbSSdebugflag<DEBUG){
2559 fprintf(DBG, "bbSS.Rl[0].CJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].CJ[bbSSell]);
2560 fflush(DBG);
2561 }
2562         bbSSjii--;
2563         }
2564 if(bbSSdebugflag<DEBUG){
2565 fprintf(DBG,"%d ",bbSSh);
2566 fflush(DBG);
2567 if(bbSSflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSSh+1),bbSSJ);
2568 else{
2569 fprintf(DBG,"%d %.1f %18.12e %18.12e (1a)\n",(bbSSh+1),\
2570   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2571   bbSS.T[0].S[bbSSff].ni[bbSSjii+1]);
2572   }
2573 fflush(DBG);
2574 }
2575 fprintf(BBSSFILE,"%d ",bbSSh);
2576 if(bbSSflgb!=0) fprintf(BBSSFILE," %d %.1f M M ",(bbSSh+1),bbSSJ);
2577 else{
2578 fprintf(BBSSFILE,"%d %.1f %18.12e %18.12e ",(bbSSh+1),\
2579   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2580   bbSS.T[0].S[bbSSff].ni[bbSSjii+1]);
2581   }
2582 fflush(BBSSFILE);
     /* Now, Delta-J=-1 (R) for Delta-K=+1. */
2583       bbSSflgb=0;
2584       if((bbSSh+1)<(bbSSLO[0].L)) bbSSflgb=1;
2585       if((bbSSJ-1)>(bbSSh+1+bbSSSl)) bbSSflgb=1;
```

```
2586       if((bbSSJ-1)<(bbSSh+1-bbSSSl)) bbSSflgb=1;
2587       if((bbSSJ-1)<0) bbSSflgb=1;
2588       if((bbSSel-1)<0) bbSSflgb=1;
2589       if(bbSSflgb==0){
2590         bbSSB=-1;
2591         bbSSell=bbSSel-1;
2592         bbSS.T[0].S[bbSSff].f[bbSSjii]=\
2593           bbSS.Rh[0].EJ[bbSSehh]-\
2594           bbSS.Rl[0].EJ[bbSSell];
2595 if(bbSSdebugflag<DEBUG){
2596 fprintf(DBG,"bbSS.T[0].S[%d].f[%d]=%13.5e ",bbSSff,bbSSjii,\
2597     bbSS.T[0].S[bbSSff].f[bbSSjii]);
2598 fprintf(DBG, "bbSS.Rh[0].EJ[%d] = %13.5e ", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
2599 fprintf(DBG, "bbSS.Rl[0].EJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
2600 fflush(DBG);
2601 }
2602         bbSS.T[0].S[bbSSff].ni[bbSSjii]=\
2603           bbSShipop*bbSStrprob*bbSSvhpop*\
2604           bbSS.Rh[0].PJ[bbSSehh]*bbSSSATT*\
2605           HL(bbSSB,bbSSJ,bbSSL,bbSSdL)*bbSSfcf;
2606 if(bbSSdebugflag<DEBUG){
2607 fprintf(DBG,"bbSShipop=%13.5e bbSStrprob=%13.5e bbSSvhpop=%13.5e
     ",\
2608     bbSShipop,bbSStrprob,bbSSvhpop);
2609 fprintf(DBG,"bbSSfcf=%13.6e\n",bbSSfcf);
2610 fprintf(DBG,"bbSS.T[0].S[%d].ni[%d]=%13.5e ",\
2611     bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2612 fprintf(DBG, "bbSS.Rh[0].PJ[%d] = %13.5e\n", bbSSehh,
     bbSS.Rh[0].PJ[bbSSehh]);
2613 fprintf(DBG, "HL(%d,%.1f,%d,%d) = %12.6e\n", bbSSB, bbSSJ, bbSSL,
     bbSSdL,\
2614     HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2615 fflush(DBG);
2616 }
2617         bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
2618           bbSS.T[0].S[bbSSff].ni[bbSSjii];
2619 if(bbSSdebugflag<DEBUG){
2620 fprintf(DBG, "bbSS.Rl[0].CJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].CJ[bbSSell]);
2621 }
2622         bbSSjii--;
2623         }
2624 if(bbSSdebugflag<DEBUG){
2625 if(bbSSflgb!=0) fprintf(DBG,"M M (Nat) (1b)\n");
2626 else{
2627 fprintf(DBG, "%18.12e (Z) %18.12e (1b)\n",
     bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2628   bbSS.T[0].S[bbSSff].ni[bbSSjii+1]);
2629   }
2630 }
2631 if(bbSSflgb!=0) fprintf(BBSSFILE,"M M\n");
2632 else{
```

```
fprintf(BBSSFILE, "%18.12e %18.12e\n",
bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
  bbSS.T[0].S[bbSSff].ni[bbSSjii+1]);
  }
      bbSSflgb=0;
/* Check Delta-J=+1 (P) and then Delta-J=-1 (R) for Delta-K=-1.
   First, check to see if the target K exists: */
      if((bbSSh-1)<(bbSSLO[0].L)) bbSSflgb=1;
/* Relative position in the low state E/P array */
      bbSSel=(bbSSh-1)*bbSS.nlJ + bbSS.nlJ-bbSS.nhJ + \
        (int)(bbSSSh-bbSSSl) + 1 + bbSSi;
if(bbSSdebugflag<DEBUG){
fprintf(DBG, "\t(bbSSh+1)*bbSS.nlJ = %d, bbSS.nlJ-bbSS.nhJ=%d\n",\
    (bbSSh+1)*bbSS.nlJ,bbSS.nlJ-bbSS.nhJ);
fprintf(DBG,"\t(int)(bbSSSh-bbSSSl)=%d, bbSSi=%d, bbSSel=%d\n",\
    (int)(bbSSSh-bbSSSl),bbSSi,bbSSel);
fflush(DBG);
}
/* Then check if the target value for J+1 is less than or greater
   than the allowed J's for K-1. */
      if((bbSSJ+1)>(bbSSh-1+bbSSSl)) bbSSflgb=1;
      if((bbSSJ+1)<(bbSSh-1-bbSSSl)) bbSSflgb=1;
      if((bbSSel+1)>=(bbSS.Rl[0].kc*bbSS.nlJ)) bbSSflgb=1;
/* Still OK? Calculate transition */
      if(bbSSflgb==0){
        bbSSB=+1;
        bbSSell=bbSSel+1;
        bbSS.T[0].S[bbSSff].f[bbSSjii]=\
          bbSS.Rh[0].EJ[bbSSehh]-\
          bbSS.Rl[0].EJ[bbSSell];
        bbSS.T[0].S[bbSSff].ni[bbSSjii]=\
          bbSShipop*bbSStrprob*bbSSvhpop*\
          bbSS.Rh[0].PJ[bbSSehh]*bbSSSATT*\
          HL(bbSSB,bbSSJ,bbSSL,bbSSdL)*bbSSfcf;
        bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
          bbSS.T[0].S[bbSSff].ni[bbSSjii];
if(bbSSdebugflag<DEBUG){
fprintf(DBG,"bbSSel=%d, bbSSi-1=%d, bbSSell=%d\n",bbSSel,bbSSi-
1,bbSSell);
fprintf(DBG,"\tbbSS.T[0].S[%d].f[%d]=%12.6e\n",\
    bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].f[bbSSjii]);
fprintf(DBG, "\tbbSS.Rh[0].EJ[%d] = %12.6e\n", bbSSehh,
bbSS.Rh[0].EJ[bbSSehh]);
fprintf(DBG, "\tbbSS.Rl[0].EJ[%d] = %12.6e\n", bbSSell,
bbSS.Rl[0].EJ[bbSSell]);
fprintf(DBG,"\tbbSS.T[0].S[%d].ni[%d]=%12.6e\n",\
    bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ni[bbSSjii]);
fprintf(DBG,"\tbbSShipop=%12.6e, bbSStrprob=%12.6e, bbSSvh-
pop=%12.6e ",\
    bbSShipop,bbSStrprob,bbSSvhpop);
fprintf(DBG,"bbSSfcf=%13.6e\n",bbSSfcf);
fprintf(DBG, "\tbbSS.Rh[0].PJ[%d] = %12.6e\n", bbSSehh,
bbSS.Rh[0].PJ[bbSSehh]);
fprintf(DBG,"\tHL(%d,%.1f,%d,%d)=%12.6e\n",bbSSB,bbSSJ,\
```

```
2675     bbSSL,bbSSdL,HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2676 fprintf(DBG, "\tbbSS.Rl[0].CJ[%d] = %12.6e\n", bbSSell,
     bbSS.Rl[0].CJ[bbSSell]);
2677 fprintf(DBG,"\tbbSS.T[0].S[%d].ni[%d]=%12.6e\n",\
2678     bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2679 }
2680         bbSSjii--;
2681         }
2682 if(bbSSdebugflag<DEBUG){
2683 fprintf(DBG,"%d ",bbSSh);
2684 if(bbSSflgb!=0) fprintf(DBG," %d %.1f M M (here)",(bbSSh-1),bbSSJ);
2685 else{
2686 fprintf(DBG,"%d %.1f %18.12e %18.12e (here) ",(bbSSh-1),\
2687   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2688   bbSS.T[0].S[bbSSff].ni[bbSSjii+1]);
2689   }
2690 }
2691 fprintf(BBSSFILE,"%d ",bbSSh);
2692 if(bbSSflgb!=0) fprintf(BBSSFILE," %d %.1f M M ",(bbSSh-1),bbSSJ);
2693 else{
2694 fprintf(BBSSFILE,"%d %.1f %18.12e %18.12e ",(bbSSh-1),\
2695   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2696   bbSS.T[0].S[bbSSff].ni[bbSSjii+1]);
2697   }
     /* Now, Delta-J=-1 (R) for Delta-K=+1. */
2698       bbSSflgb=0;
2699       if((bbSSh-1)<(bbSSLO[0].L)) bbSSflgb=1;
2700       if((bbSSJ-1)>(bbSSh-1+bbSSSl)) bbSSflgb=1;
2701       if((bbSSJ-1)<(bbSSh-1-bbSSSl)) bbSSflgb=1;
2702       if((bbSSJ-1)<0) bbSSflgb=1;
2703       if((bbSSel-1)<0) bbSSflgb=1;
2704 if(bbSSdebugflag<DEBUG){
2705 fprintf(DBG,"bbSSh-1=%d bbSSLO[0].L=%d\n",bbSSh-1,bbSSLO[0].L);
2706 fprintf(DBG,"bbSSh-1=%d, bbSS.nlJ-1=%d, bbSSel=%d\n",\
2707     bbSSh-1,bbSS.nlJ-1,bbSSel);
2708 fprintf(DBG,"bbSSJ-1=%.1f bbSSh-1=%d, bbSSSl=%.1f\n",bbSSJ-
     1,bbSSh-1,bbSSSl);
2709 fprintf(DBG,"bbSSJ-1=%.1f, bbSSh-1=%d, bbSSSl=%.1f\n",bbSSJ-
     1,bbSSh-1,bbSSSl);
2710 fprintf(DBG,"bbSSflgb=%d\n",bbSSflgb);
2711 }
2712       if(bbSSflgb==0){
2713         bbSSB=-1;
2714         bbSSell=bbSSel-1;
2715         bbSS.T[0].S[bbSSff].f[bbSSjii]=\
2716           bbSS.Rh[0].EJ[bbSSehh]-\
2717           bbSS.Rl[0].EJ[bbSSell];
2718         bbSS.T[0].S[bbSSff].ni[bbSSjii]=\
2719           bbSShipop*bbSStrprob*bbSSvhpop*\
2720           bbSS.Rh[0].PJ[bbSSehh]*bbSSfcf*\
2721           HL(bbSSB,bbSSJ,bbSSL,bbSSdL);
2722         bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
2723           bbSS.T[0].S[bbSSff].ni[bbSSjii];
2724 if(bbSSdebugflag<DEBUG){
```

```
2725 fprintf(DBG,"bbSSel=%d, bbSSi-1=%d, bbSSell=%d\n",bbSSel,bbSSi-
     1,bbSSell);
2726 fprintf(DBG,"\tbbSS.T[0].S[%d].f[%d]=%12.6e\n",\
2727     bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].f[bbSSjii]);
2728 fprintf(DBG, "\tbbSS.Rh[0].EJ[%d] = %12.6e\n", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
2729 fprintf(DBG, "\tbbSS.Rl[0].EJ[%d] = %12.6e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
2730 fprintf(DBG,"\tbbSS.T[0].S[%d].ni[%d]=%12.6e\n",\
2731     bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2732 fprintf(DBG,"\tbbSShipop=%12.6e, bbSStrprob=%12.6e, bbSSvh-
     pop=%12.6e ",\
2733     bbSShipop,bbSStrprob,bbSSvhpop);
2734 fprintf(DBG,"bbSSfcf=%13.6e\n",bbSSfcf);
2735 fprintf(DBG, "\tbbSS.Rh[0].PJ[%d] = %12.6e\n", bbSSehh,
     bbSS.Rh[0].PJ[bbSSehh]);
2736 fprintf(DBG,"\tHL(%d,%.1f,%d,%d)=%12.6e\n",bbSSB,bbSSJ,\
2737     bbSSL,bbSSdL,HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2738 fprintf(DBG, "\tbbSS.Rl[0].CJ[%d] = %12.6e\n", bbSSell,
     bbSS.Rl[0].CJ[bbSSell]);
2739 fprintf(DBG,"\tbbSS.T[0].S[%d].ni[%d]=%12.6e\n",\
2740     bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2741 }
2742         }
2743 if(bbSSdebugflag<DEBUG){
2744 if(bbSSflgb!=0) fprintf(DBG,"M M\n");
2745 else{
2746 fprintf(DBG,"%18.12e %18.12e\n",bbSS.T[0].S[bbSSff].f[bbSSjii],\
2747   bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2748   }
2749 }
2750 if(bbSSflgb!=0) fprintf(BBSSFILE,"M M\n");
2751 else{
2752 fprintf(BBSSFILE,"%18.12e
     %18.12e\n",bbSS.T[0].S[bbSSff].f[bbSSjii],\
2753   bbSS.T[0].S[bbSSff].ni[bbSSjii]);
2754   }
2755       } /* close if bbSSflga is zero. */
2756     } /* close upper-state J-of-K loop */
2757   } /* close upper-state K loop */
2758   fclose(BBSSFILE);
2759     } /* close if high-state v is natively populated */

2760   if(bbSSfrchf==0){ /* if this v is cascade populated.*/
2761 sprintf(bbSSfile,"%s_molecules/%s/%s--%s_v%d--v%d_CAS.dat",\
2762   PREF,MOL[bbSS.m].Mol,bbSS.T[0].Nhi,bbSS.T[0].Nlo,bbSSe,bbSSf);
2763     BBSSFILE=fopen(bbSSfile,"w");
2764     if(BBSSFILE==NULL){
2765 printf("Error opening transition sub-file %s. Exit-
     ing.\n",bbSSfile);
2766       exit(1);
2767       }
2768 fprintf(BBSSFILE,"# File created by %s.\n# File contains ",PRO-
     GRAM_NAME);
```

```
2769 fprintf(BBSSFILE,"P and R");
2770 fprintf(BBSSFILE," branches from the simulation titled %s\n",PREF);
2771 fprintf(BBSSFILE,"# THESE INTENSITIES ARE DUE TO CASCADE ONLY -- ");
2772 fprintf(BBSSFILE,"NO USER-DEFINED POPULATIONS\n");
2773 fprintf(BBSSFILE,"# Transition is Hund's Case (b) to Hund's Case
     (b).\n");
2774 fprintf(BBSSFILE,"# Molecule %s, \tHigh state: %s, v=%d\n",\
2775     MOL[bbSS.m].Mol,bbSS.T[0].Nhi,bbSSe);
2776 fprintf(BBSSFILE,"#\t\tLow state: %s, v=%d\n# Columns are: ",\
2777     bbSS.T[0].Nlo,bbSSf);
2778 fprintf(BBSSFILE,"K_hi K_lo J_hi P(J_hi+1)_v P_i R(J_hi-1)_v
     R_i\n#\n");

2779     bbSS.Rh=&bbSS.Sh[0].rc[bbSSfrch];
2780     if((bbSS.hiK)<bbSS.Sh[0].rc[bbSSfrch].kc){
2781       bbSSk=bbSS.hiK-1;
2782       }
2783     else bbSSk=bbSS.Sh[0].rc[bbSSfrch].kc-1;
2784     if(bbSS.Rl[0].Kdissoc<(bbSSk-1)){
2785 fprintf(PAR,"\nWARNING!! State %s of molecule %s has significant
     ",\
2786     bbSS.T[0].Nhi,MOL[bbSS.m].Mol);
2787 fprintf(PAR,"CASCADE rotational population at level %d\n",bbSSk);
2788 fprintf(PAR,"\tBut lower state %s dissociates at level %.1f\n",\
2789     bbSS.T[0].Nlo,bbSS.Rl[0].Kdissoc);
2790 fprintf(PAR,"It is likely that the bound upper state is transiting
     to (an)");
2791 fprintf(PAR," unbound lower state(s).\n");
2792 fprintf(PAR,"Look for continuum emissions trailing off to the low-
     energy ");
2793 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real ",bb-
     SSe,bbSSf);
2794 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
     gram will");
2795 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
2796       bbSSk=(int)(bbSS.Rl[0].Kdissoc);
2797       }
2798 if(bbSSdebugflag<DEBUG){
2799 fprintf(DBG,"bbSSfrch=%d, bbSSk=%d, ",bbSSfrch,bbSSk);
2800 fprintf(DBG, "bbSS.Sh[0].rc[%d].kc = %d\n", bbSSfrch,
     bbSS.Sh[0].rc[bbSSfrch].kc);
2801 fflush(DBG);
2802 }
     /* To make the output file easy to write, I'm cycling by high-state
        K first, then by high-state J. From that J/K(hi) pair, the pro-
        gram will look for a P transition and an R transition to K+1 then
        to K-1. If all the necessary states exist, it will check to see
        if there are any symmetry or nuclear spin effects to be concerned
        about. Taking these into account, it will calculate the transi-
        tion. */
     /* loop first by upper-state K-value */
2803 for(bbSSh=bbSSk;bbSSh>=0;bbSSh--){
2804   bbSSjii=(bbSSh+1)*bbSS.nT-1;  /* position in simset array */
2805   bbSSeh=bbSSh*bbSS.nhJ;  /* position in 2D for high EJ & PJ */
```

```
     /* then loop by upper-state J for this K */
2806   for(bbSSi=(bbSS.nhJ-1);bbSSi>=0;bbSSi--){
2807     bbSSJ=bbSSh+bbSSi-bbSSSh;
2808     bbSSehh=bbSSeh+bbSSi;
2809     bbSSflga=0;
2810     if(bbSSh<bbSSHI[0].L) bbSSflga=1;  /* if this K is less than
       Lambda for the high state, don't proceed */
2811     if(bbSSJ<0) bbSSflga=1;  /* if J is less than zero, don't */
2812 if(bbSSdebugflag<DEBUG){
2813 fprintf(DBG,"bbSSh=%d, bbSSi=%d, bbSSSh=%.1f, bbSSHI[0].L=%d, bb-
     SSflga=%d\n",\
2814     bbSSh,bbSSi,bbSSSh,bbSSHI[0].L,bbSSflga);
2815 fflush(DBG);
2816 }
2817     if(bbSSflga==0){
2818       bbSSflgb=0;
     /* Check Delta-J=+1 (P) and then Delta-J=-1 (R) for Delta-K=+1.
       First, check to see if the target K exists: */
2819       if((bbSSh+1)<(bbSSLO[0].L)) bbSSflgb=1;
     /* Relative position in the low state E/P array */
2820       bbSSel=(bbSSh+1)*bbSS.nlJ + bbSS.nlJ-bbSS.nhJ + \
2821         (int)(bbSSSh-bbSSSl) -1 + bbSSi;
2822 if(bbSSdebugflag<DEBUG){
2823 fprintf(DBG,"\t(bbSSh+1)*bbSS.nlJ=%d, bbSS.nlJ-bbSS.nhJ=%d\n",\
2824     (bbSSh+1)*bbSS.nlJ,bbSS.nlJ-bbSS.nhJ);
2825 fprintf(DBG,"\t(int)(bbSSSh-bbSSSl)=%d, bbSSi=%d, bbSSel=%d\n",\
2826     (int)(bbSSSh-bbSSSl),bbSSi,bbSSel);
2827 fflush(DBG);
2828 }
2829       if((bbSSel+1)>=(bbSS.Rl[0].kc*bbSS.nlJ)) bbSSflgb=1;
     /* Then check if the target value for J+1 is less than or greater
       than the allowed J's for K+1. */
2830       if((bbSSJ+1)>(bbSSh+1+bbSSSl)) bbSSflgb=1;
2831       if((bbSSJ+1)<(bbSSh+1-bbSSSl)) bbSSflgb=1;
     /* Still OK? Calculate transition */
2832       if(bbSSflgb==0){
2833         bbSSB=+1;
2834         bbSSell=bbSSel+1;
2835         bbSS.T[0].S[bbSSff].f[bbSSjii]=\
2836           bbSS.Rh[0].EJ[bbSSehh]-\
2837           bbSS.Rl[0].EJ[bbSSell];
2838         bbSScascadetemp=bbSStrprob*bbSSfcf*\
2839           bbSS.Rh[0].CJ[bbSSehh]*\
2840           HL(bbSSB,bbSSJ,bbSSL,bbSSdL);
2841 if(bbSSdebugflag<DEBUG){
2842 fprintf(DBG,"bbSS.T[0].S[%d].f[%d]=%13.5e ",bbSSff,bbSSjii,\
2843     bbSS.T[0].S[bbSSff].f[bbSSjii]);
2844 fprintf(DBG, "bbSS.Rh[0].EJ[%d]=%13.5e ", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
2845 fprintf(DBG, "bbSS.Rl[0].EJ[%d]=%13.5e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
2846 fprintf(DBG,"bbSStrprob=%13.5e,
     bbSSfcf=%13.6e",bbSStrprob,bbSSfcf);
2847 fprintf(DBG,"bbSScascadetemp=%13.5e\n",bbSScascadetemp);
```

```
2848 fprintf(DBG, "bbSS.Rh[0].CJ[%d] = %13.5e, ", bbSSehh,
     bbSS.Rh[0].CJ[bbSSehh]);
2849 fprintf(DBG, "HL(%d,%.1f,%d,%d) = %13.5e\n", bbSSB, bbSSJ, bbSSL,
     bbSSdL,\
2850     HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2851 fflush(DBG);
2852 }
2853         bbSS.T[0].S[bbSSff].ci[bbSSjii]+=\
2854           bbSScascadetemp;
2855         bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
2856           bbSScascadetemp;
2857 if(bbSSdebugflag<DEBUG){
2858 fprintf(DBG,"bbSS.T[0].S[%d].ci[%d]=%12.6e
     bbSS.Rl[0].CJ[%d]=%12.6e\n",\
2859   bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ci[bbSSjii],\
2860   bbSSell,bbSS.Rl[0].CJ[bbSSell]);
2861 }
2862         bbSSjii--;
2863         }
2864 if(bbSSdebugflag<DEBUG){
2865 fprintf(DBG,"%d ",bbSSh);
2866 if(bbSSflgb!=0) fprintf(DBG," %d %.1f (1c) M M\n",(bbSSh+1),bbSSJ);
2867 else{
2868 fprintf(DBG,"%d %.1f %18.12e %18.12e (1c)\n",(bbSSh+1),\
2869   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2870   bbSS.T[0].S[bbSSff].ci[bbSSjii+1]);
2871   }
2872 }
2873 fprintf(BBSSFILE,"%d ",bbSSh);
2874 if(bbSSflgb!=0) fprintf(BBSSFILE," %d %.1f M M ",(bbSSh+1),bbSSJ);
2875 else{
2876 fprintf(BBSSFILE,"%d %.1f %18.12e %18.12e ",(bbSSh+1),\
2877   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2878   bbSS.T[0].S[bbSSff].ci[bbSSjii+1]);
2879   }
     /* Now, Delta-J=-1 (R) for Delta-K=+1. */
2880       bbSSflgb=0;
2881       if((bbSSh+1)<(bbSSLO[0].L)) bbSSflgb=1;
2882       if((bbSSJ-1)>(bbSSh+1+bbSSSl)) bbSSflgb=1;
2883       if((bbSSJ-1)<(bbSSh+1-bbSSSl)) bbSSflgb=1;
2884       if((bbSSJ-1)<0) bbSSflgb=1;
2885       if((bbSSel-1)<0) bbSSflgb=1;
2886       if(bbSSflgb==0){
2887         bbSSB=-1;
2888         bbSSell=bbSSel-1;
2889         bbSS.T[0].S[bbSSff].f[bbSSjii]=\
2890           bbSS.Rh[0].EJ[bbSSehh]-\
2891           bbSS.Rl[0].EJ[bbSSell];
2892         bbSScascadetemp=bbSStrprob*bbSSfcf*\
2893           bbSS.Rh[0].CJ[bbSSehh]*bbSSSATT*\
2894           HL(bbSSB,bbSSJ,bbSSL,bbSSdL);
2895 if(bbSSdebugflag<DEBUG){
2896 fprintf(DBG,"bbSS.T[0].S[%d].f[%d]=%13.5e ",bbSSff,bbSSjii,\
2897     bbSS.T[0].S[bbSSff].f[bbSSjii]);
```

```
2898 fprintf(DBG, "bbSS.Rh[0].EJ[%d] = %13.5e ", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
2899 fprintf(DBG, "bbSS.Rl[0].EJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
2900 fprintf(DBG,"bbSStrprob=%13.5e,
     bbSSfcf=%13.6e",bbSStrprob,bbSSfcf);
2901 fprintf(DBG,"bbSScascadetemp=%13.5e\n",bbSScascadetemp);
2902 fprintf(DBG, "bbSS.Rh[0].CJ[%d] = %13.5e, ", bbSSehh,
     bbSS.Rh[0].CJ[bbSSehh]);
2903 fprintf(DBG, "HL(%d,%.1f,%d,%d) = %13.5e\n", bbSSB, bbSSJ, bbSSL,
     bbSSdL,\
2904     HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2905 fflush(DBG);
2906 }
2907         bbSS.T[0].S[bbSSff].ci[bbSSjii]+=\
2908           bbSScascadetemp;
2909         bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
2910           bbSScascadetemp;
2911 if(bbSSdebugflag<DEBUG){
2912 fprintf(DBG, "bbSS.T[0].S[%d].ci[%d] = %12.6e bbSS.Rl[0].CJ[%d] =
     %12.6e\n",\
2913   bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ci[bbSSjii],\
2914   bbSSell,bbSS.Rl[0].CJ[bbSSell]);
2915 }
2916         bbSSjii--;
2917         }
2918 if(bbSSdebugflag<DEBUG){
2919 if(bbSSflgb!=0) fprintf(DBG,"M M\n");
2920 else{
2921 fprintf(DBG,"%18.12e %18.12e (1d)\n",
     bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2922   bbSS.T[0].S[bbSSff].ci[bbSSjii+1]);
2923   }
2924 }
2925 if(bbSSflgb!=0) fprintf(BBSSFILE,"M M\n");
2926 else{
2927 fprintf(BBSSFILE, "%18.12e %18.12e\n",
     bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2928   bbSS.T[0].S[bbSSff].ci[bbSSjii+1]);
2929   }
2930       bbSSflgb=0;
      /* Check Delta-J=+1 (P) and then Delta-J=-1 (R) for Delta-K=-1.
         First, check to see if the target K exists: */
2931       if((bbSSh-1)<(bbSSLO[0].L)) bbSSflgb=1;
      /* Relative position in the low state E/P array */
2932       bbSSel=(bbSSh-1)*bbSS.nlJ + bbSS.nlJ-bbSS.nhJ + \
2933         (int)(bbSSSh-bbSSSl) +1 + bbSSi;
2934 if(bbSSdebugflag<DEBUG){
2935 fprintf(DBG,"\t(bbSSh+1)*bbSS.nlJ=%d, bbSS.nlJ-bbSS.nhJ=%d\n",\
2936     (bbSSh+1)*bbSS.nlJ,bbSS.nlJ-bbSS.nhJ);
2937 fprintf(DBG,"\t(int)(bbSSSh-bbSSSl)=%d, bbSSi=%d, bbSSel=%d\n",\
2938     (int)(bbSSSh-bbSSSl),bbSSi,bbSSel);
2939 fflush(DBG);
2940 }
```

```
2941        if((bbSSel+1)>=(bbSS.Rl[0].kc*bbSS.nlJ)) bbSSflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K-1. */
2942        if((bbSSJ+1)>(bbSSh-1+bbSSSl)) bbSSflgb=1;
2943        if((bbSSJ+1)<(bbSSh-1-bbSSSl)) bbSSflgb=1;
      /* Still OK? Calculate transition */
2944        if(bbSSflgb==0){
2945          bbSSB=+1;
2946          bbSSell=bbSSel+1;
2947          bbSS.T[0].S[bbSSff].f[bbSSjii]=\
2948            bbSS.Rh[0].EJ[bbSSehh]-\
2949            bbSS.Rl[0].EJ[bbSSell];
2950          bbSScascadetemp=bbSStrprob*bbSSfcf*\
2951            bbSS.Rh[0].CJ[bbSSehh]*bbSSSATT*\
2952            HL(bbSSB,bbSSJ,bbSSL,bbSSdL);
2953 if(bbSSdebugflag<DEBUG){
2954 fprintf(DBG,"bbSS.T[0].S[%d].f[%d]=%13.5e ",bbSSff,bbSSjii,\
2955     bbSS.T[0].S[bbSSff].f[bbSSjii]);
2956 fprintf(DBG, "bbSS.Rh[0].EJ[%d] = %13.5e ", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
2957 fprintf(DBG, "bbSS.Rl[0].EJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
2958 fprintf(DBG, "bbSStrprob=%13.5e, bbSSfcf = %13.6e ", bbSStrprob,
     bbSSfcf);
2959 fprintf(DBG, "bbSScascadetemp = %13.5e\n", bbSScascadetemp);
2960 fprintf(DBG, "bbSS.Rh[0].CJ[%d] = %13.5e, ", bbSSehh,
     bbSS.Rh[0].CJ[bbSSehh]);
2961 fprintf(DBG, "HL(%d,%.1f,%d,%d)=%13.5e\n", bbSSB, bbSSJ, bbSSL,
     bbSSdL,\
2962     HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
2963 fflush(DBG);
2964 }
2965          bbSS.T[0].S[bbSSff].ci[bbSSjii]+=\
2966            bbSScascadetemp;
2967          bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
2968            bbSScascadetemp;
2969 if(bbSSdebugflag<DEBUG){
2970 fprintf(DBG, "bbSS.T[0].S[%d].ci[%d] = %12.6e bbSS.Rl[0].CJ[%d] =
     %12.6e\n",\
2971   bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ci[bbSSjii],\
2972   bbSSell,bbSS.Rl[0].CJ[bbSSell]);
2973 }
2974          bbSSjii--;
2975          }
2976 if(bbSSdebugflag<DEBUG){
2977 fprintf(DBG,"%d ",bbSSh);
2978 if(bbSSflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSSh-1),bbSSJ);
2979 else{
2980 fprintf(DBG,"%d %.1f %18.12e %18.12e (1e)\n",(bbSSh-1),\
2981   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2982   bbSS.T[0].S[bbSSff].ci[bbSSjii+1]);
2983   }
2984 }
2985 fprintf(BBSSFILE,"%d ",bbSSh);
```

```
2986 if(bbSSflgb!=0) fprintf(BBSSFILE," %d %.1f M M ",(bbSSh-1),bbSSJ);
2987 else{
2988 fprintf(BBSSFILE,"%d %.1f %18.12e %18.12e ",(bbSSh-1),\
2989   bbSSJ,bbSS.T[0].S[bbSSff].f[bbSSjii+1],\
2990   bbSS.T[0].S[bbSSff].ci[bbSSjii+1]);
2991   }
     /* Now, Delta-J=-1 (R) for Delta-K=-1. */
2992       bbSSflgb=0;
2993       if((bbSSh-1)<(bbSSLO[0].L)) bbSSflgb=1;
2994       if((bbSSJ-1)>(bbSSh-1+bbSSSl)) bbSSflgb=1;
2995       if((bbSSJ-1)<(bbSSh-1-bbSSSl)) bbSSflgb=1;
2996       if((bbSSJ-1)<0) bbSSflgb=1;
2997       if((bbSSel-1)<0) bbSSflgb=1;
2998       if(bbSSflgb==0){
2999         bbSSB=-1;
3000         bbSSell=bbSSel-1;
3001         bbSS.T[0].S[bbSSff].f[bbSSjii]=\
3002           bbSS.Rh[0].EJ[bbSSehh]-\
3003           bbSS.Rl[0].EJ[bbSSell];
3004         bbSScascadetemp=bbSStrprob*bbSSfcf*\
3005           bbSS.Rh[0].CJ[bbSSehh]*\
3006           HL(bbSSB,bbSSJ,bbSSL,bbSSdL);
3007 if(bbSSdebugflag<DEBUG){
3008 fprintf(DBG,"bbSS.T[0].S[%d].f[%d]=%13.5e ",bbSSff,bbSSjii,\
3009     bbSS.T[0].S[bbSSff].f[bbSSjii]);
3010 fprintf(DBG, "bbSS.Rh[0].EJ[%d] = %13.5e ", bbSSehh,
     bbSS.Rh[0].EJ[bbSSehh]);
3011 fprintf(DBG, "bbSS.Rl[0].EJ[%d] = %13.5e\n", bbSSell,
     bbSS.Rl[0].EJ[bbSSell]);
3012 fprintf(DBG,"bbSStrprob=%13.5e, bbSSfcf=%13.6e ",bbSStr-
     prob,bbSSfcf);
3013 fprintf(DBG,"bbSScascadetemp=%13.5e\n",bbSScascadetemp);
3014 fprintf(DBG, "bbSS.Rh[0].CJ[%d] = %13.5e, ", bbSSehh,
     bbSS.Rh[0].CJ[bbSSehh]);
3015 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%13.5e\n", bbSSB, bbSSJ, bbSSL, bb-
     SSdL,\
3016     HL(bbSSB,bbSSJ,bbSSL,bbSSdL));
3017 fflush(DBG);
3018 }
3019         bbSS.T[0].S[bbSSff].ci[bbSSjii]+=\
3020           bbSScascadetemp;
3021         bbSS.Rl[0].CJ[bbSSell]+=bbSSfcfr*\
3022           bbSScascadetemp;
3023 if(bbSSdebugflag<DEBUG){
3024 fprintf(DBG, "bbSS.T[0].S[%d].ci[%d] = %12.6e bbSS.Rl[0].CJ[%d] =
     %12.6e\n",\
3025   bbSSff,bbSSjii,bbSS.T[0].S[bbSSff].ci[bbSSjii],\
3026   bbSSell,bbSS.Rl[0].CJ[bbSSell]);
3027 }
3028         }
3029 if(bbSSdebugflag<DEBUG){
3030 if(bbSSflgb!=0) fprintf(DBG,"M M\n");
3031 else{
```

```
3032 fprintf(DBG, "%18.12e %18.12e (1f)\n",
     bbSS.T[0].S[bbSSff].f[bbSSjii],\
3033   bbSS.T[0].S[bbSSff].ci[bbSSjii]);
3034   }
3035 }
3036 if(bbSSflgb!=0) fprintf(BBSSFILE,"M M\n");
3037 else{
3038 fprintf(BBSSFILE,"%18.12e
     %18.12e\n",bbSS.T[0].S[bbSSff].f[bbSSjii],\
3039   bbSS.T[0].S[bbSSff].ci[bbSSjii]);
3040   }
3041       } /* close if bbSSflga is zero. */
3042     } /* close upper-state J-of-K loop */
3043   } /* close upper-state K loop */
3044   fclose(BBSSFILE);
3045     } /* close if high-state v is cascade populated */
3046   } /* close low-state v */
3047   } /* close high-state v */
      /* check to see if this is all to do... */
3048 return;
3049 }

      /*************** b-b_SigmaOther *****************/
3050
      /* This function calculates transitions between Hund's Cases (b)
         and (b) when both the upper and lower states are sigma states. */

3051 void bb_SigmaOther(BBTinfo bbSO){
      /* see parent function and header file for key to variable names */
      /* indexes and dummy variables: */
3052 int bbSOe=0,bbSOee=0,bbSOf=0,bbSOff=0;
3053 int bbSOh=0,bbSOk=0,bbSOfrnh=0,bbSOfrcl=0,bbSOfrch=0;
3054 int bbSOeh=0,bbSOehh=0,bbSOi=0,bbSOel=0,bbSOell=0,bbSOjii=0;
      /* variables for info about the two states */
3055 double
     bbSOJ=0,bbSOSh=0,bbSOSl=0,bbSOtrprob=0,bbSOvhpop=0,bbSOhipop=0;
      /* Intensity multiplier for satellite bands and FCF dummy vari-
         able*/
3056 double bbSOSATT=0,bbSOfcf=0,bbSOfcfr=0;
      /* variables for calculating Holn-London factors */
3057 int bbSOL=0,bbSOB=0,bbSOdL=0;
      /* variables for spin stats if the state is homonuclear */
3058 int bbSOg=0,bbSOisym=0,bbSOpsym=0;
3059 double bbSOIa=0,bbSOcascadetemp=0;
      /* a few flags for various purposes */
3060 int bbSOflga=0,bbSOflgb=0,bbSOfrnhf=0,bbSOfrchf=0;
      /* variables for creating and writing to output files */
3061 char bbSOfile[1000];
3062 FILE *BBSOFILE;
3063 Case_b_stateinfo *bbSOHI,*bbSOLO;

3064 if(bbSOdebugflag<DEBUG){
3065 fprintf(DBG,"bbSO.hvnlo=%d, bbSO.hvnnum=%d\n",\
3066     bbSO.hvnlo,bbSO.hvnnum);
```

```
3067 fprintf(DBG,"bbSO.hvclo=%d, bbSO.hvcnum=%d\n",\
3068     bbSO.hvclo,bbSO.hvcnum);
3069 fprintf(DBG,"bbSO.lvnlo=%d, bbSO.lvnnum=%d,\n",\
3070     bbSO.lvnlo,bbSO.lvnnum);
3071 fprintf(DBG,"bbSO.lvclo=%d, bbSO.lvcnum=%d\n",\
3072     bbSO.lvclo,bbSO.lvcnum);
3073 fprintf(DBG,"bbSO.hivlo=%d, bbSO.hivnum=%d,\n",\
3074     bbSO.hivlo,bbSO.hivnum);
3075 fprintf(DBG,"bbSO.lovlo=%d, bbSO.lovnum=%d\n",\
3076     bbSO.lovlo,bbSO.lovnum);
3077 fflush(DBG);
3078 }
3079 bbSOHI=bbSO.Ch;
3080 bbSOLO=bbSO.Cl;
3081 bbSOSh=bbSOHI[0].S;
3082 bbSOSl=bbSOLO[0].S;
3083 bbSOhipop=bbSO.Sh[0].pop;
3084 bbSOtrprob=bbSO.T[0].P[0];
3085 if(bbSOdebugflag<DEBUG){
3086 fprintf(DBG,"bbSOSh=%.1f, bbSOSl=%.1f,\n",bbSOSh,bbSOSl);
3087 fprintf(DBG,"bbSOhipop=%12.6e,
     bbSOtrprob=%12.6e\n",bbSOhipop,bbSOtrprob);
3088 fflush(DBG);
3089 }

3090 if(bbSOHI[0].L==0){
3091   bbSOisym=bbSO.Sh[0].Isymm;
3092   bbSOpsym=bbSO.Sh[0].pmsymm;
3093   if(bbSOHI[0].g!=0){
3094     bbSOg=bbSOHI[0].g;
3095     bbSOIa=bbSOHI[0].I/(bbSOHI[0].I+1);
3096     }
3097   }
3098 if(bbSOLO[0].L==0){
3099   bbSOisym=bbSO.Sl[0].Isymm;
3100   bbSOpsym=bbSO.Sl[0].pmsymm;
3101   if(bbSOLO[0].g!=0){
3102     bbSOg=bbSOLO[0].g;
3103     bbSOIa=bbSOLO[0].I/(bbSOLO[0].I+1);
3104     }
3105   }
3106 if((bbSOHI[0].L-bbSOLO[0].L)==+1) bbSOdL=+1;
3107 if((bbSOHI[0].L-bbSOLO[0].L)==-1) bbSOdL=-1;
3108 bbSOL=bbSOHI[0].L;

      /* start loop down through high vib levels */
3109 for(bbSOe=(bbSO.hivlo+bbSO.hivnum-1);bbSOe>=bbSO.hivlo;bbSOe--){
3110   bbSOee=(bbSOe-bbSO.hivlo)*bbSO.lovnum;  /* simset posn. in 2D */
3111   bbSOfrnh=(bbSOe-bbSO.hvnlo);  /* high state native vib position
        */
3112   bbSOfrch=bbSOe;  /* high state cascade vib position */
      /* these flags (bbSOfrnhf and bbSOfrchf) tell if this high vib state
         is populated natively, by cascade, or both. They will be used
         later */
```

```
3113   if((bbSOe>=bbSO.hvnlo)&&(bbSOe<(bbSO.hvnlo+bbSO.hvnnum))){
3114     bbSOfrnhf=0;
3115     bbSOvhpop=bbSO.Vh[0].p[bbSOe-bbSO.hvnlo];
3116     }
3117   else bbSOfrnhf=-1;
3118   if((bbSOe>=bbSO.hvclo)&&(bbSOe<(bbSO.hvclo+bbSO.hvcnum))){
3119     bbSOfrchf=0;
3120     }
3121   else bbSOfrchf=-1;
3122 if(bbSOdebugflag<DEBUG){
3123 fprintf(DBG,"bbSOfrnhf=%d, bbSOvhpop=%12.6e, bbSOfrchf=%d, ",\
3124     bbSOfrnhf,bbSOvhpop,bbSOfrchf);
3125 fprintf(DBG,"bbSOe=%d, bbSOee=%d\nbbSOfrnh=%d, bbSOfrch=%d,\n",\
3126     bbSOe,bbSOee,bbSOfrnh,bbSOfrch);
3127 fprintf(DBG,"bbSO.Vh[0].p[%d-%d]=%12.6e\n",bbSOe,bbSO.hvnlo,\
3128      bbSO.Vh[0].p[bbSOe-bbSO.hvnlo]);
3129 fflush(DBG);
3130 }
     /* start loop down through low vib levels */
3131 for(bbSOf=(bbSO.lovlo+bbSO.lovnum-1);bbSOf>=bbSO.lovlo;bbSOf--){
3132   bbSOfcf=bbSO.T[0].v[bbSOe].fcfn[bbSOf];
3133
     if(bbSOfcf!=0){bbSOfcfr=bbSO.T[0].v[bbSOe].fcfc[bbSOf]/bbSOfcf;}
3134   bbSOff=bbSOee+bbSOf-bbSO.lovlo;  /* position in last dimension */
3135   bbSOfrcl=bbSOf;

3136 if(bbSOdebugflag<DEBUG){
3137 fprintf(DBG,"bbSOf=%d, bbSO.lovlo=%d, bbSO.lovnum=%d, bb-
     SOfcf=%12.6e \n",\
3138     bbSOf,bbSO.lovlo,bbSO.lovnum,bbSOfcf);
3139 fflush(DBG);
3140 }
3141 bbSOff=bbSOee+bbSOf-bbSO.lovlo;  /* simset posn. in final dimension
       */
3142   bbSO.T[0].S[bbSOff].n=bbSO.nT*bbSO.hiK;
3143   bbSO.T[0].S[bbSOff].f=\
3144     (double*)calloc(bbSO.T[0].S[bbSOff].n,sizeof(double));
3145   bbSO.T[0].S[bbSOff].ni=\
3146     (double*)calloc(bbSO.T[0].S[bbSOff].n,sizeof(double));
3147   bbSO.T[0].S[bbSOff].ci=\
3148     (double*)calloc(bbSO.T[0].S[bbSOff].n,sizeof(double));

3149 bbSO.Rl=&bbSO.Sl[0].rc[bbSOfrcl];
3150 if(bbSOdebugflag<DEBUG){
3151 fprintf(DBG,"bbSOff=%d, bbSOfrcl=%d,
     bbSO.T[0].S[bbSOff].n=%d\n",\
3152     bbSOff,bbSOfrcl,bbSO.T[0].S[bbSOff].n);
3153 fprintf(DBG,"bbSO.nT=%d, bbSO.hiK=%d\n",bbSO.nT,bbSO.hiK);
3154 fflush(DBG);
3155 }
     /* start with highest J value (same reason as before), and loop down
        looking for lower J's to which to transit. Assign intensities.
        If both states are Omega=0, then assign zero intensity to transi-
        tions for J=0<->J=0. */
```

```
3156   if(bbSOfrnhf==0){ /* if this v is natively populated.*/
3157 sprintf(bbSOfile,"%s_molecules/%s/%s--%s_v%d--v%d_NAT.dat",\
3158   PREF,MOL[bbSO.m].Mol,bbSO.T[0].Nhi,bbSO.T[0].Nlo,bbSOe,bbSOf);
3159     BBSOFILE=fopen(bbSOfile,"w");
3160     if(BBSOFILE==NULL){
3161 printf("Error opening transition sub-file %s. Exit-
    ing.\n",bbSOfile);
3162       exit(1);
3163       }
3164 fprintf(BBSOFILE,"# File created by %s.\n# File contains ",PRO-
    GRAM_NAME);
3165 fprintf(BBSOFILE,"P, Q and R");
3166 fprintf(BBSOFILE," branches from the simulation titled %s\n",PREF);
3167 fprintf(BBSOFILE,"# THESE INTENSITIES ARE DUE TO USER-SPECIFIED
    POPULATIONS");
3168 fprintf(BBSOFILE," ONLY -- NO CASCADE\n");
3169 fprintf(BBSOFILE,"# Transition is Hund's Case (b) to Hund's Case
    (b).\n");
3170 fprintf(BBSOFILE,"# Molecule %s, \tHigh state: %s, v=%d\n",\
3171     MOL[bbSO.m].Mol,bbSO.T[0].Nhi,bbSOe);
3172 fprintf(BBSOFILE,"#\t\tLow state: %s, v=%d\n# Columns are: ",\
3173     bbSO.T[0].Nlo,bbSOf);
3174 fprintf(BBSOFILE,"K_hi K_lo J_hi P(J_hi+1)_v P_i ");
3175 fprintf(BBSOFILE,"Q(J_hi+0)_v Q_i R(J_hi-1)_v R_i\n#\n");

3176     bbSO.Rh=&bbSO.Sh[0].r[bbSOfrnh];
3177     if((bbSO.hiK)<bbSO.Sh[0].r[bbSOfrnh].k){
3178       bbSOk=bbSO.hiK-1;
3179       }
3180     else bbSOk=bbSO.Sh[0].r[bbSOfrnh].k-1;
3181     if(bbSO.Rl[0].Kdissoc<(bbSOk-1)){
3182 fprintf(PAR,"\nWARNING!! State %s of molecule %s has significant
    ",\
3183     bbSO.T[0].Nhi,MOL[bbSO.m].Mol);
3184 fprintf(PAR,"rotational population at level %d\n",bbSOk);
3185 fprintf(PAR,"\tBut lower state %s dissociates at level %.1f\n",\
3186     bbSO.T[0].Nlo,bbSO.Rl[0].Kdissoc);
3187 fprintf(PAR,"It is likely that the bound upper state is transiting
    to (an)");
3188 fprintf(PAR," unbound lower state(s).\n");
3189 fprintf(PAR,"Look for continuum emissions trailing off to the low-
    energy ");
3190 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real ",bb-
    SOe,bbSOf);
3191 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
    gram will");
3192 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
3193       bbSOk=(int)(bbSO.Rl[0].Kdissoc);
3194       }
3195 if(bbSOdebugflag<DEBUG){
3196 fprintf(DBG,"bbSOfrnh=%d, bbSOk=%d, ",bbSOfrnh,bbSOk);
3197 fprintf(DBG, "bbSO.Sh[0].r[%d].k = %d\n", bbSOfrnh,
    bbSO.Sh[0].r[bbSOfrnh].k);
3198 fflush(DBG);
```

```
3199 }
     /* To make the output file easy to write, I'm cycling by high-state
        K first, then by high-state J. From that J/K(hi) pair, the pro-
        gram will look for a P transition and an R transition to K+1 then
        to K-1. If all the necessary states exist, it will check to see
        if there are any symmetry or nuclear spin effects to be concerned
        about. Taking these into account, it will calculate the transi-
        tion. */
     /* loop first by upper-state K-value */
3200 for(bbSOh=bbSOk;bbSOh>=0;bbSOh--){
3201   bbSOjii=(bbSOh+1)*bbSO.nT-1;  /* position in simset array */
3202   bbSOeh=bbSOh*bbSO.nhJ;  /* position in 2D for high EJ & PJ */
     /* then loop by upper-state J for this K */
3203   for(bbSOi=(bbSO.nhJ-1);bbSOi>=0;bbSOi--){
3204     bbSOJ=bbSOh+bbSOi-bbSOSh;
3205     bbSOehh=bbSOeh+bbSOi;
3206     bbSOflga=0;
3207     if(bbSOh<bbSOHI[0].L) bbSOflga=1;  /* if this K is less than
        Lambda for the high state, don't proceed */
3208     if(bbSOJ<0) bbSOflga=1;  /* if J is less than zero, don't */
3209 if(bbSOdebugflag<DEBUG){
3210 fprintf(DBG,"bbSOh=%d, bbSOi=%d, bbSOSh=%.1f, bbSOHI[0].L=%d, bb-
    SOflga=%d\n",\
3211     bbSOh,bbSOi,bbSOSh,bbSOHI[0].L,bbSOflga);
3212 fflush(DBG);
3213 }
3214     if(bbSOflga==0){
     /* Check Delta-J=+1 (P), then Delta-J=0 (Q), and then Delta-J=-1
        (R) for Delta-K=+1. First, check to see if the target K exists:
        */
3215       bbSOflgb=0;
3216       if((bbSOh+1)<(bbSOLO[0].L)) bbSOflgb=1;
     /* Relative position in the low state E/P array */
3217       bbSOel=(bbSOh+1)*bbSO.nlJ + bbSO.nlJ-bbSO.nhJ + \
3218         (int)(bbSOSh-bbSOSl) -1 + bbSOi;
3219 if(bbSOdebugflag<DEBUG){
3220 fprintf(DBG,"\t(bbSOh+1)*bbSO.nlJ=%d, bbSO.nlJ-bbSO.nhJ=%d\n",\
3221     (bbSOh+1)*bbSO.nlJ,bbSO.nlJ-bbSO.nhJ);
3222 fprintf(DBG,"\t(int)(bbSOSh-bbSOSl)=%d, bbSOi=%d, bbSOel=%d\n",\
3223     (int)(bbSOSh-bbSOSl),bbSOi,bbSOel);
3224 fflush(DBG);
3225 }
3226       if((bbSOel+1)>=(bbSO.Rl[0].kc*bbSO.nlJ)) bbSOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
3227       if((bbSOJ+1)>(bbSOh+1+bbSOSl)) bbSOflgb=1;
3228       if((bbSOJ+1)<(bbSOh+1-bbSOSl)) bbSOflgb=1;
     /* Still OK? Calculate transition */
3229       if(bbSOflgb==0){
3230         bbSOB=+1;
3231         bbSOell=bbSOel+1;
3232         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3233           bbSO.Rh[0].EJ[bbSOehh]-\
3234           bbSO.Rl[0].EJ[bbSOell];
```

```
3235          bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3236            bbSOhipop*bbSOtrprob*bbSOvhpop*\
3237            bbSO.Rh[0].PJ[bbSOehh]*bbSOfcf*\
3238            HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
3239 if(bbSOdebugflag<DEBUG){
3240 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3241 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3242 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3243     bbSO.T[0].S[bbSOff].f[bbSOjii]);
3244 fprintf(DBG,"bbSOhipop%12.6e, bbSOtrprob%12.6e, bbSOvh-
     pop=%12.6e\n",\
3245     bbSOhipop,bbSOtrprob,bbSOvhpop);
3246 fprintf(DBG,"bbSOfcf=%13.6e ",bbSOfcf);
3247 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3248 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3249   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3250 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3251   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3252 }
      /* Check for nuclear effects and modify intensity if necessary. No
         need to do this if the high state is Sigma. */
3253      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3254        if((((bbSOh+1)%2)==0)&&(bbSOisym==-1)){
3255          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3256 if(bbSOdebugflag<DEBUG){
3257 fprintf(DBG,"Even low state is Ia intensity\n");
3258 fflush(DBG);
3259 }
3260          }
3261        if((((bbSOh+1)%2)==1)&&(bbSOisym==+1)){
3262          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3263 if(bbSOdebugflag<DEBUG){
3264 fprintf(DBG,"Even low state is Ia intensity\n");
3265 fflush(DBG);
3266 }
3267          }
3268        }
      /* set low-state population for cascade */
3269          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3270            bbSO.T[0].S[bbSOff].ni[bbSOjii];
3271 if(bbSOdebugflag<DEBUG){
3272 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3273   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3274   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3275 }
3276          bbSOjii--;
3277          }
3278 if(bbSOdebugflag<DEBUG){
3279 fprintf(DBG,"bbSO, K+1, J+1 native\n");
```

```
3280 fprintf(DBG,"%d ",bbSOh);
3281 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh+1),bbSOJ);
3282 else{
3283 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbSOh+1),\
3284   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3285   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3286   }
3287 }
3288 fprintf(BBSOFILE,"%d ",bbSOh);
3289 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh+1),bbSOJ);
3290 else{
3291 fprintf(BBSOFILE,"%d %.1f %18.12e %18.12e ",(bbSOh+1),\
3292   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3293   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3294   }
     /* Now Delta-J=0 (Q) (for Delta-K=+1) */
3295       bbSOflgb=0;
3296       if((bbSOh+1)<(bbSOLO[0].L)) bbSOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
3297       if((bbSOJ+0)>(bbSOh+1+bbSOSl)) bbSOflgb=1;
3298       if((bbSOJ+0)<(bbSOh+1-bbSOSl)) bbSOflgb=1;
3299       if((bbSOJ+0)<0) bbSOflgb=1;
     /* Still OK? Calculate transition */
3300       if(bbSOflgb==0){
3301         bbSOB=+0;
3302         bbSOell=bbSOel+0;
3303         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3304           bbSO.Rh[0].EJ[bbSOehh]-\
3305           bbSO.Rl[0].EJ[bbSOell];
3306         bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3307           bbSOhipop*bbSOtrprob*bbSOvhpop*\
3308           bbSO.Rh[0].PJ[bbSOehh]*bbSOSATT*\
3309           HL(bbSOB,bbSOJ,bbSOL,bbSOdL)*bbSOfcf;
3310 if(bbSOdebugflag<DEBUG){
3311 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3312 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3313 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3314     bbSO.T[0].S[bbSOff].f[bbSOjii]);
3315 fprintf(DBG,"bbSOhipop %12.6e, bbSOtrprob %12.6e, bbSOvhpop=
     %12.6e\n",\
3316     bbSOhipop,bbSOtrprob,bbSOvhpop);
3317 fprintf(DBG,"bbSOfcf=%13.6e ",bbSOfcf);
3318 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3319 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3320   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3321 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3322   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3323 }
     /* Check for nuclear effects and modify intensity if necessary. */
```

```
3324     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3325       if((((bbSOh+1)%2)==0)&&(bbSOisym==-1)){
3326         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3327         }
3328       if((((bbSOh+1)%2)==1)&&(bbSOisym==+1)){
3329         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3330         }
3331       }
     /* set low-state population for cascade */
3332         bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3333           bbSO.T[0].S[bbSOff].ni[bbSOjii];
3334 if(bbSOdebugflag<DEBUG){
3335 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3336   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3337   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3338 }
3339         bbSOjii--;
3340         }
3341 if(bbSOdebugflag<DEBUG){
3342 fprintf(DBG,"bbSO, K+1, J+0 native\n");
3343 fprintf(DBG,"%d ",bbSOh);
3344 if(bbSOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSOh+1),bbSOJ);
3345 else{
3346 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbSOh+1),\
3347   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3348   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3349   }
3350 }
3351 if(bbSOflgb!=0) fprintf(BBSOFILE," M M ");
3352 else{
3353 fprintf(BBSOFILE," %18.12e %18.12e
     ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3354   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3355   }
     /* Now, Delta-J=-1 (R) for Delta-K=+1. */
3356       bbSOflgb=0;
3357       if((bbSOh+1)<(bbSOLO[0].L)) bbSOflgb=1;
3358       if((bbSOJ-1)>(bbSOh+1+bbSOSl)) bbSOflgb=1;
3359       if((bbSOJ-1)<(bbSOh+1-bbSOSl)) bbSOflgb=1;
3360       if((bbSOJ-1)<0) bbSOflgb=1;
3361       if(bbSOflgb==0){
3362         bbSOB=-1;
3363         bbSOell=bbSOel-1;
3364         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3365           bbSO.Rh[0].EJ[bbSOehh]-\
3366           bbSO.Rl[0].EJ[bbSOell];
3367         bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3368           bbSOhipop*bbSOtrprob*bbSOvhpop*\
3369           bbSO.Rh[0].PJ[bbSOehh]*bbSOSATT*\
3370           HL(bbSOB,bbSOJ,bbSOL,bbSOdL)*bbSOfcf;
3371 if(bbSOdebugflag<DEBUG){
3372 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
```

```
3373 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3374 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3375     bbSO.T[0].S[bbSOff].f[bbSOjii]);
3376 fprintf(DBG,"bbSOhipop%12.6e, bbSOtrprob%12.6e, bbSOvh-
     pop=%12.6e\n",\
3377     bbSOhipop,bbSOtrprob,bbSOvhpop);
3378 fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3379 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3380 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3381   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3382 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3383   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3384 }
3385     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3386       if((((bbSOh+1)%2)==0)&&(bbSOisym==-1)){
3387         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3388         }
3389       if((((bbSOh+1)%2)==1)&&(bbSOisym==+1)){
3390         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3391         }
3392       }
3393         bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3394           bbSO.T[0].S[bbSOff].ni[bbSOjii];
3395 if(bbSOdebugflag<DEBUG){
3396 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3397   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3398   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3399 }
3400         bbSOjii--;
3401         }
3402 if(bbSOdebugflag<DEBUG){
3403 fprintf(DBG,"bbSO, K+1, J-1 native\n");
3404 if(bbSOflgb!=0) fprintf(DBG,"M M\n");
3405 else{
3406 fprintf(DBG,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3407   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3408   }
3409 }
3410 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M\n");
3411 else{
3412 fprintf(BBSOFILE,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3413   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3414   }

      /* Check Delta-J=+1 (P), then Delta-J=0 (Q), and then Delta-J=-1
         (R) for Delta-K=+0. First, check to see if the target K exists:
         */
3415       bbSOflgb=0;
```

```
3416        if((bbSOh+0)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Relative position in the low state E/P array */
3417        bbSOel=(bbSOh+0)*bbSO.nlJ + bbSO.nlJ-bbSO.nhJ + \
3418          (int)(bbSOSh-bbSOSl) +0 + bbSOi;
3419  if(bbSOdebugflag<DEBUG){
3420  fprintf(DBG,"\t(bbSOh+0)*bbSO.nlJ=%d, bbSO.nlJ-bbSO.nhJ=%d\n",\
3421      (bbSOh+0)*bbSO.nlJ,bbSO.nlJ-bbSO.nhJ);
3422  fprintf(DBG,"\t(int)(bbSOSh-bbSOSl)=%d, bbSOi=%d, bbSOel=%d\n",\
3423      (int)(bbSOSh-bbSOSl),bbSOi,bbSOel);
3424  fflush(DBG);
3425  }
3426        if((bbSOel+1)>=(bbSO.Rl[0].kc*bbSO.nlJ)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+0. */
3427        if((bbSOJ+1)>(bbSOh+0+bbSOSl)) bbSOflgb=1;
3428        if((bbSOJ+1)<(bbSOh+0-bbSOSl)) bbSOflgb=1;
      /* Still OK? Calculate transition */
3429        if(bbSOflgb==0){
3430          bbSOB=+1;
3431          bbSOell=bbSOel+1;
3432          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3433            bbSO.Rh[0].EJ[bbSOehh]-\
3434            bbSO.Rl[0].EJ[bbSOell];
3435          bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3436            bbSOhipop*bbSOtrprob*bbSOvhpop*\
3437            bbSO.Rh[0].PJ[bbSOehh]*bbSOSATT*\
3438            HL(bbSOB,bbSOJ,bbSOL,bbSOdL)*bbSOfcf;
3439  if(bbSOdebugflag<DEBUG){
3440  fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
      ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3441  fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
      ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3442  fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3443      bbSO.T[0].S[bbSOff].f[bbSOjii]);
3444  fprintf(DBG,"bbSOhipop=%12.6e, bbSOtrprob=%12.6e, bbSOvh-
      pop=%12.6e\n",\
3445      bbSOhipop,bbSOtrprob,bbSOvhpop);
3446  fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3447  fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
      ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3448  fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
      ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3449    HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3450  fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3451    bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3452  }
      /* Check for nuclear effects and modify intensity if necessary. */
3453      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3454        if((((bbSOh+0)%2)==0)&&(bbSOisym==-1)){
3455          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3456          }
3457        if((((bbSOh+0)%2)==1)&&(bbSOisym==+1)){
3458          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3459          }
```

```
3460        }
      /* set low-state population for cascade */
3461          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3462            bbSO.T[0].S[bbSOff].ni[bbSOjii];
3463 if(bbSOdebugflag<DEBUG){
3464 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3465   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3466   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3467 }
3468          bbSOjii--;
3469          }
3470 if(bbSOdebugflag<DEBUG){
3471 fprintf(DBG,"bbSO, K+0, J+1 native\n");
3472 fprintf(DBG,"%d ",bbSOh);
3473 if(bbSOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSOh+0),bbSOJ);
3474 else{
3475 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbSOh+0),\
3476   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3477   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3478   }
3479 }
3480 fprintf(BBSOFILE,"%d ",bbSOh);
3481 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh+0),bbSOJ);
3482 else{
3483 fprintf(BBSOFILE,"%d %.1f %18.12e %18.12e ",(bbSOh+0),\
3484   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3485   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3486   }

      /* Now Delta-J=0 (Q) (for Delta-K=+0) */
3487       bbSOflgb=0;
3488       if((bbSOh+0)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
         than the allowed J's for K+0. */
3489       if((bbSOJ+0)>(bbSOh+0+bbSOSl)) bbSOflgb=1;
3490       if((bbSOJ+0)<(bbSOh+0-bbSOSl)) bbSOflgb=1;
      /* Still OK? Calculate transition */
3491       if(bbSOflgb==0){
3492         bbSOB=0;
3493         bbSOell=bbSOel+0;
3494         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3495           bbSO.Rh[0].EJ[bbSOehh]-\
3496           bbSO.Rl[0].EJ[bbSOell];
3497         bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3498           bbSOhipop*bbSOtrprob*bbSOvhpop*\
3499           bbSO.Rh[0].PJ[bbSOehh]*bbSOfcf*\
3500           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
3501 if(bbSOdebugflag<DEBUG){
3502 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3503 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3504 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
```

```
3505      bbSO.T[0].S[bbSOff].f[bbSOjii]);
3506 fprintf(DBG,"bbSOhipop%12.6e, bbSOtrprob%12.6e, bbSOvh-
     pop=%12.6e\n",\
3507      bbSOhipop,bbSOtrprob,bbSOvhpop);
3508 fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3509 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3510 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3511    HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3512 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3513    bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3514 }
      /* Check for nuclear effects and modify intensity if necessary. */
3515      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3516        if((((bbSOh+0)%2)==0)&&(bbSOisym==-1)){
3517          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3518          }
3519        if((((bbSOh+0)%2)==1)&&(bbSOisym==+1)){
3520          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3521          }
3522        }
      /* set low-state population for cascade */
3523          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3524            bbSO.T[0].S[bbSOff].ni[bbSOjii];
3525 if(bbSOdebugflag<DEBUG){
3526 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3527    bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3528    bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3529 }
3530          bbSOjii--;
3531          }
3532 if(bbSOdebugflag<DEBUG){
3533 fprintf(DBG,"bbSO, K+0, J+0 native\n");
3534 fprintf(DBG,"%d ",bbSOh);
3535 if(bbSOflgb!=0) fprintf(DBG," M M ");
3536 else{
3537 fprintf(DBG," %18.12e %18.12e ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3538    bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3539    }
3540 }
3541 if(bbSOflgb!=0) fprintf(BBSOFILE," M M ");
3542 else{
3543 fprintf(BBSOFILE,"%18.12e %18.12e
     ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3544    bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3545    }
      /* Now, Delta-J=-1 (R) for Delta-K=+0. */
3546        bbSOflgb=0;
3547        if((bbSOh+0)<(bbSOLO[0].L)) bbSOflgb=1;
3548        if((bbSOJ-1)>(bbSOh+0+bbSOSl)) bbSOflgb=1;
3549        if((bbSOJ-1)<(bbSOh+0-bbSOSl)) bbSOflgb=1;
3550        if((bbSOJ-1)<0) bbSOflgb=1;
```

```
3551       if(bbSOflgb==0){
3552         bbSOB=-1;
3553         bbSOell=bbSOel-1;
3554         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3555           bbSO.Rh[0].EJ[bbSOehh]-\
3556           bbSO.Rl[0].EJ[bbSOell];
3557         bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3558           bbSOhipop*bbSOtrprob*bbSOvhpop*\
3559           bbSO.Rh[0].PJ[bbSOehh]*bbSOSATT*\
3560           HL(bbSOB,bbSOJ,bbSOL,bbSOdL)*bbSOfcf;
3561 if(bbSOdebugflag<DEBUG){
3562 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3563 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3564 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3565     bbSO.T[0].S[bbSOff].f[bbSOjii]);
3566 fprintf(DBG,"bbSOhipop=%12.6e, bbSOtrprob=%12.6e, bbSOvh-
     pop=%12.6e\n",\
3567     bbSOhipop,bbSOtrprob,bbSOvhpop);
3568 fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3569 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3570 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3571   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3572 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3573   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3574 }
3575     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3576       if((((bbSOh+0)%2)==0)&&(bbSOisym==-1)){
3577         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3578         }
3579       if((((bbSOh+0)%2)==1)&&(bbSOisym==+1)){
3580         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3581         }
3582       }
3583         bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3584           bbSO.T[0].S[bbSOff].ni[bbSOjii];
3585 if(bbSOdebugflag<DEBUG){
3586 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3587   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3588   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3589 }
3590         bbSOjii--;
3591         }
3592 if(bbSOdebugflag<DEBUG){
3593 fprintf(DBG,"bbSO, K+0, J-1 native\n");
3594 if(bbSOflgb!=0) fprintf(DBG,"M M\n");
3595 else{
3596 fprintf(DBG,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3597   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
```

```
3598   }
3599 }
3600 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M\n");
3601 else{
3602 fprintf(BBSOFILE,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3603   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3604   }
      /* Check Delta-J=+1 (P), then Delta-J=0 and Delta-J=-1 (R) for
         Delta-K=-1. First, check to see if the target K exists: */
3605       bbSOflgb=0;
3606       if((bbSOh-1)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Relative position in the low state E/P array */
3607       bbSOel=(bbSOh-1)*bbSO.nlJ + bbSO.nlJ-bbSO.nhJ + \
3608         (int)(bbSOSh-bbSOSl) + 1 + bbSOi;
3609 if(bbSOdebugflag<DEBUG){
3610 fprintf(DBG,"\t(bbSOh-1)*bbSO.nlJ=%d, bbSO.nlJ-bbSO.nhJ=%d\n",\
3611     (bbSOh-1)*bbSO.nlJ,bbSO.nlJ-bbSO.nhJ);
3612 fprintf(DBG,"\t(int)(bbSOSh-bbSOSl)=%d, bbSOi=%d, bbSOel=%d\n",\
3613     (int)(bbSOSh-bbSOSl),bbSOi,bbSOel);
3614 fflush(DBG);
3615 }
3616       if((bbSOel+1)>=(bbSO.Rl[0].kc*bbSO.nlJ)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
         than the allowed J's for K-1. */
3617       if((bbSOJ+1)>(bbSOh-1+bbSOSl)) bbSOflgb=1;
3618       if((bbSOJ+1)<(bbSOh-1-bbSOSl)) bbSOflgb=1;
      /* Still OK? Calculate transition */
3619       if(bbSOflgb==0){
3620         bbSOB=+1;
3621         bbSOell=bbSOel+1;
3622         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3623           bbSO.Rh[0].EJ[bbSOehh]-\
3624           bbSO.Rl[0].EJ[bbSOell];
3625         bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3626           bbSOhipop*bbSOtrprob*bbSOvhpop*\
3627           bbSO.Rh[0].PJ[bbSOehh]*bbSOSATT*\
3628           HL(bbSOB,bbSOJ,bbSOL,bbSOdL)*bbSOfcf;
3629 if(bbSOdebugflag<DEBUG){
3630 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3631 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3632 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3633     bbSO.T[0].S[bbSOff].f[bbSOjii]);
3634 fprintf(DBG,"bbSOhipop%12.6e, bbSOtrprob%12.6e, bbSOvh-
     pop=%12.6e\n",\
3635     bbSOhipop,bbSOtrprob,bbSOvhpop);
3636 fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3637 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3638 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3639   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
```

```
3640 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3641   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3642 }
     /* Check for nuclear effects and modify intensity if necessary. */
3643     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3644       if((((bbSOh-1)%2)==0)&&(bbSOisym==-1)){
3645         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3646         }
3647       if((((bbSOh-1)%2)==1)&&(bbSOisym==+1)){
3648         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3649         }
3650       }
3651         bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3652           bbSO.T[0].S[bbSOff].ni[bbSOjii];
3653 if(bbSOdebugflag<DEBUG){
3654 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3655   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3656   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3657 }
3658         bbSOjii--;
3659         }
3660 if(bbSOdebugflag<DEBUG){
3661 fprintf(DBG,"bbSO, K-1, J+1 native\n");
3662 fprintf(DBG,"%d ",bbSOh);
3663 if(bbSOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSOh-1),bbSOJ);
3664 else{
3665 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbSOh-1),\
3666   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3667   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3668   }
3669 }
3670 fprintf(BBSOFILE,"%d ",bbSOh);
3671 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh-1),bbSOJ);
3672 else{
3673 fprintf(BBSOFILE,"%d %.1f %18.12e %18.12e ",(bbSOh-1),\
3674   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3675   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3676   }
     /* Now, Delta-J=+0 (Q) for Delta-K=-1. */
3677       bbSOflgb=0;
3678       if((bbSOh-1)<(bbSOLO[0].L)) bbSOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K-1. */
3679       if((bbSOJ+0)>(bbSOh-1+bbSOSl)) bbSOflgb=1;
3680       if((bbSOJ+0)<(bbSOh-1-bbSOSl)) bbSOflgb=1;
     /* Still OK? Calculate transition */
3681       if(bbSOflgb==0){
3682         bbSOB=+0;
3683         bbSOell=bbSOel+0;
3684         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3685           bbSO.Rh[0].EJ[bbSOehh]-\
3686           bbSO.Rl[0].EJ[bbSOell];
3687         bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
```

```
3688            bbSOhipop*bbSOtrprob*bbSOvhpop*\
3689            bbSO.Rh[0].PJ[bbSOehh]*bbSOSATT*\
3690            HL(bbSOB,bbSOJ,bbSOL,bbSOdL)*bbSOfcf;
3691 if(bbSOdebugflag<DEBUG){
3692 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3693 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3694 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3695     bbSO.T[0].S[bbSOff].f[bbSOjii]);
3696 fprintf(DBG,"bbSOhipop=%12.6e, bbSOtrprob=%12.6e, bbSOvh-
     pop=%12.6e\n",\
3697     bbSOhipop,bbSOtrprob,bbSOvhpop);
3698 fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3699 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3700 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3701   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3702 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3703   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3704 }
      /* Check for nuclear effects and modify intensity if necessary. */
3705     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3706       if((((bbSOh-1)%2)==0)&&(bbSOisym==-1)){
3707         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3708         }
3709       if((((bbSOh-1)%2)==1)&&(bbSOisym==+1)){
3710         bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3711         }
3712       }
3713         bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3714           bbSO.T[0].S[bbSOff].ni[bbSOjii];
3715 if(bbSOdebugflag<DEBUG){
3716 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3717   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3718   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3719 }
3720         bbSOjii--;
3721         }
3722 if(bbSOdebugflag<DEBUG){
3723 fprintf(DBG,"bbSO, K-1, J+0 native\n");
3724 fprintf(DBG,"%d ",bbSOh);
3725 if(bbSOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSOh-1),bbSOJ);
3726 else{
3727 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbSOh-1),\
3728   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3729   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3730   }
3731 }
3732 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M ");
3733 else{
```

```
3734 fprintf(BBSOFILE," %18.12e %18.12e
     ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3735   bbSO.T[0].S[bbSOff].ni[bbSOjii+1]);
3736   }
      /* Now, Delta-J=-1 (R) for Delta-K=-1. */
3737        bbSOflgb=0;
3738        if((bbSOh-1)<(bbSOLO[0].L)) bbSOflgb=1;
3739        if((bbSOJ-1)>(bbSOh-1+bbSOSl)) bbSOflgb=1;
3740        if((bbSOJ-1)<(bbSOh-1-bbSOSl)) bbSOflgb=1;
3741        if((bbSOJ-1)<0) bbSOflgb=1;
3742        if(bbSOflgb==0){
3743          bbSOB=-1;
3744          bbSOell=bbSOel-1;
3745          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3746            bbSO.Rh[0].EJ[bbSOehh]-\
3747            bbSO.Rl[0].EJ[bbSOell];
3748          bbSO.T[0].S[bbSOff].ni[bbSOjii]=\
3749            bbSOhipop*bbSOtrprob*bbSOvhpop*\
3750            bbSO.Rh[0].PJ[bbSOehh]*bbSOfcf*\
3751           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
3752 if(bbSOdebugflag<DEBUG){
3753 fprintf(DBG,"bbSO.Rh[0].EJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].EJ[bbSOehh]);
3754 fprintf(DBG,"bbSO.Rl[0].EJ[%d]=%12.6e
     ",bbSOell,bbSO.Rl[0].EJ[bbSOell]);
3755 fprintf(DBG,"bbSO.T[0].S[%d].f[%d]=%12.6e ",bbSOff,bbSOjii,\
3756      bbSO.T[0].S[bbSOff].f[bbSOjii]);
3757 fprintf(DBG,"bbSOhipop=%12.6e, bbSOtrprob=%12.6e, bbSOvh-
     pop=%12.6e\n",\
3758      bbSOhipop,bbSOtrprob,bbSOvhpop);
3759 fprintf(DBG,"bbSOfcf=%12.6e ",bbSOfcf);
3760 fprintf(DBG,"bbSO.Rh[0].PJ[%d]=%12.6e
     ",bbSOehh,bbSO.Rh[0].PJ[bbSOehh]);
3761 fprintf(DBG,"HL(%d,%.1f,%d,%d)=%12.6e
     ",bbSOB,bbSOJ,bbSOL,bbSOdL,\
3762   HL(bbSOB,bbSOJ,bbSOL,bbSOdL));
3763 fprintf(DBG,"bbSO.T[0].S[%d].ni[%d]=%12.6e\n",bbSOff,bbSOjii,\
3764   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3765 }
3766      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3767        if((((bbSOh-1)%2)==0)&&(bbSOisym==-1)){
3768          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3769          }
3770        if((((bbSOh-1)%2)==1)&&(bbSOisym==+1)){
3771          bbSO.T[0].S[bbSOff].ni[bbSOjii]*=bbSOIa;
3772          }
3773        }
3774          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3775            bbSO.T[0].S[bbSOff].ni[bbSOjii];
3776 if(bbSOdebugflag<DEBUG){
3777 fprintf(DBG,"bbSO.Rl[0].CJ[%d]=%12.6e,
     bbSO.T[0].S[%d].ni[%d]=%12.6e\n",\
3778   bbSOell,bbSO.Rl[0].CJ[bbSOell],bbSOff,\
3779   bbSOjii,bbSO.T[0].S[bbSOff].ni[bbSOjii]);
```

```
3780 }
3781         }
3782 if(bbSOdebugflag<DEBUG){
3783 fprintf(DBG,"bbSO, K-1, J-1 native\n");
3784 if(bbSOflgb!=0) fprintf(DBG,"M M\n");
3785 else{
3786 fprintf(DBG,"%18.12e %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii],\
3787   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3788   }
3789 }
3790 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M\n");
3791 else{
3792 fprintf(BBSOFILE,"%18.12e
    %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii],\
3793   bbSO.T[0].S[bbSOff].ni[bbSOjii]);
3794   }
3795       } /* close if bbSOflga is zero. */
3796     } /* close upper-state J-of-K loop */
3797   } /* close upper-state K loop */
3798   fclose(BBSOFILE);
3799     } /* close if high-state v is natively populated */

3800   if(bbSOfrchf==0){ /* if this v is cascade populated.*/
3801 sprintf(bbSOfile,"%s_molecules/%s/%s--%s_v%d--v%d_CAS.dat",\
3802   PREF,MOL[bbSO.m].Mol,bbSO.T[0].Nhi,bbSO.T[0].Nlo,bbSOe,bbSOf);
3803     BBSOFILE=fopen(bbSOfile,"w");
3804     if(BBSOFILE==NULL){
3805 printf("Error opening transition sub-file %s. Exit-
    ing.\n",bbSOfile);
3806       exit(1);
3807       }
3808 fprintf(BBSOFILE,"# File created by %s.\n# File contains ",PRO-
    GRAM_NAME);
3809 fprintf(BBSOFILE,"P and R");
3810 fprintf(BBSOFILE," branches from the simulation titled %s\n",PREF);
3811 fprintf(BBSOFILE,"# THESE INTENSITIES ARE DUE TO CASCADE ONLY -- ");
3812 fprintf(BBSOFILE,"NO USER-DEFINED POPULATIONS\n");
3813 fprintf(BBSOFILE,"# Transition is Hund's Case (b) to Hund's Case
    (b).\n");
3814 fprintf(BBSOFILE,"# Molecule %s, \tHigh state: %s, v=%d\n",\
3815     MOL[bbSO.m].Mol,bbSO.T[0].Nhi,bbSOe);
3816 fprintf(BBSOFILE,"#\t\tLow state: %s, v=%d\n# Columns are: ",\
3817     bbSO.T[0].Nlo,bbSOf);
3818 fprintf(BBSOFILE,"K_hi K_lo J_hi P(J_hi+1)_v P_i R(J_hi-1)_v
    R_i\n#\n");

3819     bbSO.Rh=&bbSO.Sh[0].rc[bbSOfrch];
3820     if((bbSO.hiK)<bbSO.Sh[0].rc[bbSOfrch].kc){
3821       bbSOk=bbSO.hiK-1;
3822       }
3823     else bbSOk=bbSO.Sh[0].rc[bbSOfrch].kc-1;
3824     if(bbSO.Rl[0].Kdissoc<(bbSOk-1)){
3825 fprintf(PAR,"\nWARNING!! State %s of molecule %s has significant
    ",\
```

```
3826      bbSO.T[0].Nhi,MOL[bbSO.m].Mol);
3827 fprintf(PAR,"CASCADE rotational population at level %d\n",bbSOk);
3828 fprintf(PAR,"\tBut lower state %s dissociates at level %.1f\n",\
3829      bbSO.T[0].Nlo,bbSO.Rl[0].Kdissoc);
3830 fprintf(PAR,"It is likely that the bound upper state is transiting
     to (an)");
3831 fprintf(PAR," unbound lower state(s).\n");
3832 fprintf(PAR,"Look for continuum emissions trailing off to the low-
     energy ");
3833 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real ",bb-
     SOe,bbSOf);
3834 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
     gram will");
3835 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
3836       bbSOk=(int)(bbSO.Rl[0].Kdissoc);
3837       }
3838 if(bbSOdebugflag<DEBUG){
3839 fprintf(DBG,"bbSOfrch=%d, bbSOk=%d, ",bbSOfrch,bbSOk);
3840 fprintf(DBG, "bbSO.Sh[0].rc[%d].kc = %d\n", bbSOfrch,
     bbSO.Sh[0].rc[bbSOfrch].kc);
3841 fflush(DBG);
3842 }
     /* To make the output file easy to write, I'm cycling by high-state
        K first, then by high-state J. From that J/K(hi) pair, the pro-
        gram will look for a P transition and an R transition to K+1 then
        to K-1. If all the necessary states exist, it will check to see
        if there are any symmetry or nuclear spin effects to be concerned
        about. Taking these into account, it will calculate the transi-
        tion. */
     /* loop first by upper-state K-value */
3843 for(bbSOh=bbSOk;bbSOh>=0;bbSOh--){
3844   bbSOjii=(bbSOh+1)*bbSO.nT-1;  /* position in simset array */
3845   bbSOeh=bbSOh*bbSO.nhJ;  /* position in 2D for high EJ & PJ */
     /* then loop by upper-state J for this K */
3846   for(bbSOi=(bbSO.nhJ-1);bbSOi>=0;bbSOi--){
3847     bbSOJ=bbSOh+bbSOi-bbSOSh;
3848     bbSOehh=bbSOeh+bbSOi;
3849     bbSOflga=0;
3850     if(bbSOh<bbSOHI[0].L) bbSOflga=1;  /* if this K is less than
        Lambda for the high state, don't proceed */
3851     if(bbSOJ<0) bbSOflga=1;  /* if J is less than zero, don't */
3852 if(bbSOdebugflag<DEBUG){
3853 fprintf(DBG,"bbSOh=%d, bbSOi=%d, bbSOSh=%.1f, bbSOHI[0].L=%d, bb-
     SOflga=%d\n",\
3854      bbSOh,bbSOi,bbSOSh,bbSOHI[0].L,bbSOflga);
3855 fflush(DBG);
3856 }
3857      if(bbSOflga==0){
     /* Check Delta-J=+1 (P), then Delta-J=0 (Q) and Delta-J=-1 (R) for
        Delta-K=+1. First, check to see if the target K exists: */
3858        bbSOflgb=0;
3859        if((bbSOh+1)<(bbSOLO[0].L)) bbSOflgb=1;
     /* Relative position in the low state E/P array */
3860        bbSOel=(bbSOh+1)*bbSO.nlJ + bbSO.nlJ-bbSO.nhJ + \
```

```
3861          (int)(bbSOSh-bbSOSl) -1 + bbSOi;
3862 if(bbSOdebugflag<DEBUG){
3863 fprintf(DBG,"\t(bbSOh+1)*bbSO.nlJ=%d, bbSO.nlJ-bbSO.nhJ=%d\n",\
3864     (bbSOh+1)*bbSO.nlJ,bbSO.nlJ-bbSO.nhJ);
3865 fprintf(DBG,"\t(int)(bbSOSh-bbSOSl)=%d, bbSOi=%d, bbSOel=%d\n",\
3866     (int)(bbSOSh-bbSOSl),bbSOi,bbSOel);
3867 fflush(DBG);
3868 }
3869       if((bbSOel+1)>=(bbSO.Rl[0].kc*bbSO.nlJ)) bbSOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
3870       if((bbSOJ+1)>(bbSOh+1+bbSOSl)) bbSOflgb=1;
3871       if((bbSOJ+1)<(bbSOh+1-bbSOSl)) bbSOflgb=1;
     /* Still OK? Calculate transition */
3872       if(bbSOflgb==0){
3873         bbSOB=+1;
3874         bbSOell=bbSOel+1;
3875         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3876           bbSO.Rh[0].EJ[bbSOehh]-\
3877           bbSO.Rl[0].EJ[bbSOell];
3878         bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
3879           bbSO.Rh[0].CJ[bbSOehh]*\
3880           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
     /* Check for nuclear effects and modify intensity if necessary. */
3881      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3882         if((((bbSOh+1)%2)==0)&&(bbSOisym==-1)){
3883           bbSOcascadetemp*=bbSOIa;
3884           }
3885         if((((bbSOh+1)%2)==1)&&(bbSOisym==+1)){
3886           bbSOcascadetemp*=bbSOIa;
3887           }
3888         }
3889           bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
3890             bbSOcascadetemp;
3891           bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3892             bbSOcascadetemp;
3893           bbSOjii--;
3894           }
3895 if(bbSOdebugflag<DEBUG){
3896 fprintf(DBG,"%d ",bbSOh);
3897 if(bbSOflgb!=0) fprintf(DBG," %d %.1f (1c) M M\n",(bbSOh+1),bbSOJ);
3898 else{
3899 fprintf(DBG,"%d %.1f %18.12e %18.12e (1c)\n",(bbSOh+1),\
3900   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3901   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
3902   }
3903 }
3904 fprintf(BBSOFILE,"%d ",bbSOh);
3905 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh+1),bbSOJ);
3906 else{
3907 fprintf(BBSOFILE,"%d %.1f %18.12e %18.12e ",(bbSOh+1),\
3908   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3909   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
3910   }
```

```
      /* Now, Delta-J=0 (Q), Delta-K=+1 */
3911        bbSOflgb=0;
3912        if((bbSOh+1)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
3913        if((bbSOJ+0)>(bbSOh+1+bbSOSl)) bbSOflgb=1;
3914        if((bbSOJ+0)<(bbSOh+1-bbSOSl)) bbSOflgb=1;
3915        if((bbSOJ+0)<0) bbSOflgb=1;
      /* Still OK? Calculate transition */
3916        if(bbSOflgb==0){
3917          bbSOB=+0;
3918          bbSOell=bbSOel+0;
3919          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3920            bbSO.Rh[0].EJ[bbSOehh]-\
3921            bbSO.Rl[0].EJ[bbSOell];
3922          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
3923            bbSO.Rh[0].CJ[bbSOehh]*bbSOSATT*\
3924            HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
      /* Check for nuclear effects and modify intensity if necessary. */
3925       if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3926          if((((bbSOh+1)%2)==0)&&(bbSOisym==-1)){
3927            bbSOcascadetemp*=bbSOIa;
3928            }
3929          if((((bbSOh+1)%2)==1)&&(bbSOisym==+1)){
3930            bbSOcascadetemp*=bbSOIa;
3931            }
3932          }
3933          bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
3934            bbSOcascadetemp;
3935          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3936            bbSOcascadetemp;
3937          bbSOjii--;
3938          }
3939 if(bbSOdebugflag<DEBUG){
3940 if(bbSOflgb!=0) fprintf(DBG," M M\n");
3941 else{
3942 fprintf(DBG,"%18.12e %18.12e
      (1c)\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3943   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
3944   }
3945 }
3946 if(bbSOflgb!=0) fprintf(BBSOFILE," M M ");
3947 else{
3948 fprintf(BBSOFILE," %18.12e %18.12e
      ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3949   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
3950   }
      /* Now, Delta-J=-1 (R) for Delta-K=+1. */
3951        bbSOflgb=0;
3952        if((bbSOh+1)<(bbSOLO[0].L)) bbSOflgb=1;
3953        if((bbSOJ-1)>(bbSOh+1+bbSOSl)) bbSOflgb=1;
3954        if((bbSOJ-1)<(bbSOh+1-bbSOSl)) bbSOflgb=1;
3955        if((bbSOJ-1)<0) bbSOflgb=1;
3956        if(bbSOflgb==0){
```

```
3957          bbSOB=-1;
3958          bbSOell=bbSOel-1;
3959          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
3960            bbSO.Rh[0].EJ[bbSOehh]-\
3961            bbSO.Rl[0].EJ[bbSOell];
3962          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
3963            bbSO.Rh[0].CJ[bbSOehh]*bbSOSATT*\
3964           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
3965      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
3966        if((((bbSOh+1)%2)==0)&&(bbSOisym==-1)){
3967          bbSOcascadetemp*=bbSOIa;
3968          }
3969        if((((bbSOh+1)%2)==1)&&(bbSOisym==+1)){
3970          bbSOcascadetemp*=bbSOIa;
3971          }
3972        }
3973          bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
3974            bbSOcascadetemp;
3975          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
3976            bbSOcascadetemp;
3977          bbSOjii--;
3978          }
3979 if(bbSOdebugflag<DEBUG){
3980 if(bbSOflgb!=0) fprintf(DBG,"M M\n");
3981 else{
3982 fprintf(DBG,"%18.12e %18.12e
     (1d)\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3983   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
3984   }
3985 }
3986 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M\n");
3987 else{
3988 fprintf(BBSOFILE,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
3989   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
3990   }

      /* Now, again for Delta-K=+0. Delta-J=+1 (P), then (Q) and (R) */
3991        bbSOflgb=0;
3992        if((bbSOh+0)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Relative position in the low state E/P array */
3993        bbSOel=(bbSOh+0)*bbSO.nlJ + bbSO.nlJ-bbSO.nhJ + \
3994          (int)(bbSOSh-bbSOSl) + 0 + bbSOi;
3995 if(bbSOdebugflag<DEBUG){
3996 fprintf(DBG,"\t(bbSOh+0)*bbSO.nlJ=%d, bbSO.nlJ-bbSO.nhJ=%d\n",\
3997     (bbSOh+0)*bbSO.nlJ,bbSO.nlJ-bbSO.nhJ);
3998 fprintf(DBG,"\t(int)(bbSOSh-bbSOSl)=%d, bbSOi=%d, bbSOel=%d\n",\
3999     (int)(bbSOSh-bbSOSl),bbSOi,bbSOel);
4000 fflush(DBG);
4001 }
4002        if((bbSOel+1)>=(bbSO.Rl[0].kc*bbSO.nlJ)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
         than the allowed J's for K+0. */
4003        if((bbSOJ+1)>(bbSOh+0+bbSOSl)) bbSOflgb=1;
```

```
4004        if((bbSOJ+1)<(bbSOh+0-bbSOSl)) bbSOflgb=1;
      /* Still OK? Calculate transition */
4005        if(bbSOflgb==0){
4006          bbSOB=+1;
4007          bbSOell=bbSOel+1;
4008          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
4009            bbSO.Rh[0].EJ[bbSOehh]-\
4010            bbSO.Rl[0].EJ[bbSOell];
4011          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
4012            bbSO.Rh[0].CJ[bbSOehh]*bbSOSATT*\
4013            HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
      /* Check for nuclear effects and modify intensity if necessary. */
4014       if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
4015          if((((bbSOh+0)%2)==0)&&(bbSOisym==-1)){
4016            bbSOcascadetemp*=bbSOIa;
4017            }
4018          if((((bbSOh+0)%2)==1)&&(bbSOisym==+1)){
4019            bbSOcascadetemp*=bbSOIa;
4020            }
4021          }
4022            bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
4023              bbSOcascadetemp;
4024            bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
4025              bbSOcascadetemp;
4026            bbSOjii--;
4027            }
4028 if(bbSOdebugflag<DEBUG){
4029 fprintf(DBG,"%d ",bbSOh);
4030 if(bbSOflgb!=0) fprintf(DBG," %d %.1f (1c) M M\n",(bbSOh+0),bbSOJ);
4031 else{
4032 fprintf(DBG,"%d %.1f %18.12e %18.12e (1c)\n",(bbSOh+0),\
4033   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4034   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4035   }
4036 }
4037 fprintf(BBSOFILE,"%d ",bbSOh);
4038 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh+0),bbSOJ);
4039 else{
4040 fprintf(BBSOFILE,"%d %.1f %18.12e %18.12e ",(bbSOh+0),\
4041   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4042   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4043   }
      /* Now, Delta-J=0 (Q), Delta-K=+0 */
4044        bbSOflgb=0;
4045        if((bbSOh+0)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+0. */
4046        if((bbSOJ+0)>(bbSOh+0+bbSOSl)) bbSOflgb=1;
4047        if((bbSOJ+0)<(bbSOh+0-bbSOSl)) bbSOflgb=1;
      /* Still OK? Calculate transition */
4048        if(bbSOflgb==0){
4049          bbSOB=+0;
4050          bbSOell=bbSOel+0;
4051          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
```

```
4052            bbSO.Rh[0].EJ[bbSOehh]-\
4053            bbSO.Rl[0].EJ[bbSOell];
4054          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
4055            bbSO.Rh[0].CJ[bbSOehh]*\
4056            HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
      /* Check for nuclear effects and modify intensity if necessary. */
4057      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
4058        if((((bbSOh+0)%2)==0)&&(bbSOisym==-1)){
4059          bbSOcascadetemp*=bbSOIa;
4060          }
4061        if((((bbSOh+0)%2)==1)&&(bbSOisym==+1)){
4062          bbSOcascadetemp*=bbSOIa;
4063          }
4064        }
4065          bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
4066            bbSOcascadetemp;
4067          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
4068            bbSOcascadetemp;
4069          bbSOjii--;
4070          }
4071 if(bbSOdebugflag<DEBUG){
4072 if(bbSOflgb!=0) fprintf(DBG," M M\n");
4073 else{
4074 fprintf(DBG,"%18.12e %18.12e
     (1c)\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4075   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4076   }
4077 }
4078 if(bbSOflgb!=0) fprintf(BBSOFILE," M M ");
4079 else{
4080 fprintf(BBSOFILE," %18.12e %18.12e
     ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4081   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4082   }
      /* Now, Delta-J=-1 (R) for Delta-K=+0. */
4083        bbSOflgb=0;
4084        if((bbSOh+0)<(bbSOLO[0].L)) bbSOflgb=1;
4085        if((bbSOJ-1)>(bbSOh+0+bbSOSl)) bbSOflgb=1;
4086        if((bbSOJ-1)<(bbSOh+0-bbSOSl)) bbSOflgb=1;
4087        if((bbSOJ-1)<0) bbSOflgb=1;
4088        if(bbSOflgb==0){
4089          bbSOB=-1;
4090          bbSOell=bbSOel-1;
4091          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
4092            bbSO.Rh[0].EJ[bbSOehh]-\
4093            bbSO.Rl[0].EJ[bbSOell];
4094          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
4095            bbSO.Rh[0].CJ[bbSOehh]*bbSOSATT*\
4096           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
4097      if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
4098        if((((bbSOh+0)%2)==0)&&(bbSOisym==-1)){
4099          bbSOcascadetemp*=bbSOIa;
4100          }
4101        if((((bbSOh+0)%2)==1)&&(bbSOisym==+1)){
```

```
4102          bbSOcascadetemp*=bbSOIa;
4103          }
4104        }
4105          bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
4106            bbSOcascadetemp;
4107          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
4108            bbSOcascadetemp;
4109          bbSOjii--;
4110          }
4111 if(bbSOdebugflag<DEBUG){
4112 if(bbSOflgb!=0) fprintf(DBG,"M M\n");
4113 else{
4114 fprintf(DBG,"%18.12e %18.12e
     (1d)\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4115   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4116   }
4117 }
4118 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M\n");
4119 else{
4120 fprintf(BBSOFILE,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4121   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4122   }

      /* Check Delta-J=+1 (P), then Delta-J=0 (Q) and Delta-J=-1 (R) for
        Delta-K=-1. First, check to see if the target K exists: */
4123        bbSOflgb=0;
4124        if((bbSOh-1)<(bbSOLO[0].L)) bbSOflgb=1;
      /* Relative position in the low state E/P array */
4125        bbSOel=(bbSOh-1)*bbSO.nlJ + bbSO.nlJ-bbSO.nhJ + \
4126          (int)(bbSOSh-bbSOSl) + 1 + bbSOi;
4127 if(bbSOdebugflag<DEBUG){
4128 fprintf(DBG,"\t(bbSOh-1)*bbSO.nlJ=%d, bbSO.nlJ-bbSO.nhJ=%d\n",\
4129     (bbSOh-1)*bbSO.nlJ,bbSO.nlJ-bbSO.nhJ);
4130 fprintf(DBG,"\t(int)(bbSOSh-bbSOSl)=%d, bbSOi=%d, bbSOel=%d\n",\
4131     (int)(bbSOSh-bbSOSl),bbSOi,bbSOel);
4132 fflush(DBG);
4133 }
4134        if((bbSOel+1)>=(bbSO.Rl[0].kc*bbSO.nlJ)) bbSOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
4135        if((bbSOJ+1)>(bbSOh-1+bbSOSl)) bbSOflgb=1;
4136        if((bbSOJ+1)<(bbSOh-1-bbSOSl)) bbSOflgb=1;
      /* Still OK? Calculate transition */
4137        if(bbSOflgb==0){
4138          bbSOB=+1;
4139          bbSOell=bbSOel+1;
4140          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
4141            bbSO.Rh[0].EJ[bbSOehh]-\
4142            bbSO.Rl[0].EJ[bbSOell];
4143          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
4144            bbSO.Rh[0].CJ[bbSOehh]*bbSOSATT*\
4145            HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
      /* Check for nuclear effects and modify intensity if necessary. */
```

```
4146     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
4147       if((((bbSOh-1)%2)==0)&&(bbSOisym==-1)){
4148         bbSOcascadetemp*=bbSOIa;
4149         }
4150       if((((bbSOh-1)%2)==1)&&(bbSOisym==+1)){
4151         bbSOcascadetemp*=bbSOIa;
4152         }
4153       }
4154         bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
4155           bbSOcascadetemp;
4156         bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
4157           bbSOcascadetemp;
4158         bbSOjii--;
4159         }
4160 if(bbSOdebugflag<DEBUG){
4161 fprintf(DBG,"%d ",bbSOh);
4162 if(bbSOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbSOh-1),bbSOJ);
4163 else{
4164 fprintf(DBG,"%d %.1f %18.12e %18.12e (1e)\n",(bbSOh-1),\
4165   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4166   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4167   }
4168 }
4169 fprintf(BBSOFILE,"%d ",bbSOh);
4170 if(bbSOflgb!=0) fprintf(BBSOFILE," %d %.1f M M ",(bbSOh-1),bbSOJ);
4171 else{
4172 fprintf(BBSOFILE,"%d %.1f %18.12e %18.12e ",(bbSOh-1),\
4173   bbSOJ,bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4174   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4175   }
     /* Now, Delta-J=+0 (Q), Delta-K=-1. */
4176       bbSOflgb=0;
4177       if((bbSOh-1)<(bbSOLO[0].L)) bbSOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
4178       if((bbSOJ+0)>(bbSOh-1+bbSOSl)) bbSOflgb=1;
4179       if((bbSOJ+0)<(bbSOh-1-bbSOSl)) bbSOflgb=1;
     /* Still OK? Calculate transition */
4180       if(bbSOflgb==0){
4181         bbSOB=+0;
4182         bbSOell=bbSOel+0;
4183         bbSO.T[0].S[bbSOff].f[bbSOjii]=\
4184           bbSO.Rh[0].EJ[bbSOehh]-\
4185           bbSO.Rl[0].EJ[bbSOell];
4186         bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
4187           bbSO.Rh[0].CJ[bbSOehh]*bbSOSATT*\
4188           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
     /* Check for nuclear effects and modify intensity if necessary. */
4189     if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
4190       if((((bbSOh-1)%2)==0)&&(bbSOisym==-1)){
4191         bbSOcascadetemp*=bbSOIa;
4192         }
4193       if((((bbSOh-1)%2)==1)&&(bbSOisym==+1)){
4194         bbSOcascadetemp*=bbSOIa;
```

```
4195          }
4196        }
4197          bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
4198            bbSOcascadetemp;
4199          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
4200            bbSOcascadetemp;
4201          bbSOjii--;
4202          }
4203 if(bbSOdebugflag<DEBUG){
4204 if(bbSOflgb!=0) fprintf(DBG," M M ");
4205 else{
4206 fprintf(DBG,"%18.12e %18.12e
     \n",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4207   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4208   }
4209 }
4210 if(bbSOflgb!=0) fprintf(BBSOFILE," M M ");
4211 else{
4212 fprintf(BBSOFILE,"%18.12e %18.12e
     ",bbSO.T[0].S[bbSOff].f[bbSOjii+1],\
4213   bbSO.T[0].S[bbSOff].ci[bbSOjii+1]);
4214   }
      /* Now, Delta-J=-1 (R) for Delta-K=-1. */
4215        bbSOflgb=0;
4216        if((bbSOh-1)<(bbSOLO[0].L)) bbSOflgb=1;
4217        if((bbSOJ-1)>(bbSOh-1+bbSOSl)) bbSOflgb=1;
4218        if((bbSOJ-1)<(bbSOh-1-bbSOSl)) bbSOflgb=1;
4219        if((bbSOJ-1)<0) bbSOflgb=1;
4220        if(bbSOflgb==0){
4221          bbSOB=-1;
4222          bbSOell=bbSOel-1;
4223          bbSO.T[0].S[bbSOff].f[bbSOjii]=\
4224            bbSO.Rh[0].EJ[bbSOehh]-\
4225            bbSO.Rl[0].EJ[bbSOell];
4226          bbSOcascadetemp=bbSOtrprob*bbSOfcf*\
4227            bbSO.Rh[0].CJ[bbSOehh]*\
4228           HL(bbSOB,bbSOJ,bbSOL,bbSOdL);
      /* If the high state is Sigma, the nuclear spin-stats (if homonu-
        clear) were taken care of in the rotational distribution func-
        tion, so no need to bother with them here. */
4229       if((bbSOg!=0)&&(bbSOLO[0].L==0)){ /* low state is Sigma */
4230         if((((bbSOh-1)%2)==0)&&(bbSOisym==-1)){
4231           bbSOcascadetemp*=bbSOIa;
4232           }
4233         if((((bbSOh-1)%2)==1)&&(bbSOisym==+1)){
4234           bbSOcascadetemp*=bbSOIa;
4235           }
4236         }
4237          bbSO.T[0].S[bbSOff].ci[bbSOjii]+=\
4238            bbSOcascadetemp;
4239          bbSO.Rl[0].CJ[bbSOell]+=bbSOfcfr*\
4240            bbSOcascadetemp;
4241          }
4242 if(bbSOdebugflag<DEBUG){
```

```
4243 if(bbSOflgb!=0) fprintf(DBG,"M M\n");
4244 else{
4245 fprintf(DBG,"%18.12e %18.12e
     (1f)\n",bbSO.T[0].S[bbSOff].f[bbSOjii],\
4246   bbSO.T[0].S[bbSOff].ci[bbSOjii]);
4247   }
4248 }
4249 if(bbSOflgb!=0) fprintf(BBSOFILE,"M M\n");
4250 else{
4251 fprintf(BBSOFILE,"%18.12e
     %18.12e\n",bbSO.T[0].S[bbSOff].f[bbSOjii],\
4252   bbSO.T[0].S[bbSOff].ci[bbSOjii]);
4253   }
4254       } /* close if bbSOflga is zero. */
4255     } /* close upper-state J-of-K loop */
4256   } /* close upper-state K loop */
4257   fclose(BBSOFILE);
4258     } /* close if high-state v is cascade populated */

4259   } /* close low-state v */
4260   } /* close high-state v */
4261 return;
4262 }

      /*************** b-b_Other *****************/
4263
      /* This function calculates transitions between Hund's Cases (b)
         and (b) when both the upper and lower states are sigma states. */

4264 void bb_Other(BBTinfo bbOO){
      /* see parent function and header file for key to variable names */
      /* indexes and dummy variables: */
4265 int bbOOe=0,bbOOee=0,bbOOf=0,bbOOff=0,bbOOi=0,bbOOjii=0;
4266 int bbOOeh=0,bbOOehh=0,bbOOel=0,bbOOell=0;
4267 int bbOOh=0,bbOOk=0,bbOOfrnh=0,bbOOfrcl=0,bbOOfrch=0;
      /* variables for info about the two states */
4268 double
     bbOOJ=0,bbOOSh=0,bbOOSl=0,bbOOtrprob=0,bbOOvhpop=0,bbOOhipop;
      /* Intensity multiplier for satellite bands, temporary variable and
         fcf var */
4269 double bbOOSATT=0,bbOOcascadetemp=0,bbOOfcf=0,bbOOfcfr=0;
      /* variables for calculating Holn-London factors */
4270 int bbOOL=0,bbOOB=0,bbOOdL=0;
      /* a few flags for various purposes */
4271 int bbOOflga=0,bbOOflgb=0,bbOOfrnhf=0,bbOOfrchf=0,bbOOLLflg=1;
      /* variables for creating and writing to output files */
4272 char bbOOfile[1000];
4273 FILE *BBOOFILE;
4274 Case_b_stateinfo *bbOOHI,*bbOOLO;

4275 if(bbOOdebugflag<DEBUG){
4276 fprintf(DBG,"bbOO.hvnlo=%d, bbOO.hvnnum=%d\n",\
4277     bbOO.hvnlo,bbOO.hvnnum);
4278 fprintf(DBG,"bbOO.hvclo=%d, bbOO.hvcnum=%d\n",\
4279     bbOO.hvclo,bbOO.hvcnum);
```

```
4280 fprintf(DBG,"bbOO.lvnlo=%d, bbOO.lvnnum=%d,\n",\
4281     bbOO.lvnlo,bbOO.lvnnum);
4282 fprintf(DBG,"bbOO.lvclo=%d, bbOO.lvcnum=%d\n",\
4283     bbOO.lvclo,bbOO.lvcnum);
4284 fprintf(DBG,"bbOO.hivlo=%d, bbOO.hivnum=%d,\n",\
4285     bbOO.hivlo,bbOO.hivnum);
4286 fprintf(DBG,"bbOO.lovlo=%d, bbOO.lovnum=%d\n",\
4287     bbOO.lovlo,bbOO.lovnum);
4288 fflush(DBG);
4289 }
4290 bbOOHI=bbOO.Ch;
4291 bbOOLO=bbOO.Cl;
4292 bbOOSh=bbOOHI[0].S;
4293 bbOOSl=bbOOLO[0].S;
4294 bbOOhipop=bbOO.Sh[0].pop;
4295 bbOOtrprob=bbOO.T[0].P[0];
4296 if(bbOOHI[0].L==bbOOLO[0].L){
4297   bbOOLLflg=0;
4298   bbOOdL=0;
4299   }
4300 if((bbOOHI[0].L-bbOOLO[0].L)==+1) bbOOdL=+1;
4301 if((bbOOHI[0].L-bbOOLO[0].L)==-1) bbOOdL=-1;
4302 bbOOL=bbOOHI[0].L;

     /* start loop down through high vib levels */
4303 for(bbOOe=(bbOO.hivlo+bbOO.hivnum-1);bbOOe>=bbOO.hivlo;bbOOe--){
4304   bbOOee=(bbOOe-bbOO.hivlo)*bbOO.lovnum;  /* simset posn. in 2D */
4305   bbOOfrnh=(bbOOe-bbOO.hvnlo);  /* high state native vib position
       */
4306   bbOOfrch=bbOOe;  /* high state cascade vib position */
     /* these flags (bbOOfrnhf and bbOOfrchf) tell if this high vib state
        is populated natively, by cascade, or both. They will be used
        later */
4307   if((bbOOe>=bbOO.hvnlo)&&(bbOOe<(bbOO.hvnlo+bbOO.hvnnum))){
4308     bbOOfrnhf=0;
4309     bbOOvhpop=bbOO.Vh[0].p[bbOOe-bbOO.hvnlo];
4310     }
4311   else bbOOfrnhf=-1;
4312   if((bbOOe>=bbOO.hvclo)&&(bbOOe<(bbOO.hvclo+bbOO.hvcnum))){
4313     bbOOfrchf=0;
4314     }
4315   else bbOOfrchf=-1;
4316 if(bbOOdebugflag<DEBUG){
4317 fprintf(DBG,"bbOOfrnhf=%d, bbOOvhpop=%f, bbOOfrchf=%d, ",\
4318     bbOOfrnhf,bbOOvhpop,bbOOfrchf);
4319 fprintf(DBG,"bbOOe=%d, bbOOee=%d\nbbOOfrnh=%d, bbOOfrch=%d, ",\
4320     bbOOe,bbOOee,bbOOfrnh,bbOOfrch);
4321 fflush(DBG);
4322 }
     /* start loop down through low vib levels */
4323 for(bbOOf=(bbOO.lovlo+bbOO.lovnum-1);bbOOf>=bbOO.lovlo;bbOOf--){
4324   bbOOfcf=bbOO.T[0].v[bbOOe].fcfn[bbOOf];
4325
     if(bbOOfcf!=0){bbOOfcfr=bbOO.T[0].v[bbOOe].fcfc[bbOOf]/bbOOfcf;}
```

```
4326   bbOOff=bbOOee+bbOOf-bbOO.lovlo;  /* position in last dimension */
4327   bbOOfrcl=bbOOf;
4328 if(bbOOdebugflag<DEBUG){
4329 fprintf(DBG,"bbOOf=%d, bbOO.lovlo=%d, bbOO.lovnum=%d \n",\
4330     bbOOf,bbOO.lovlo,bbOO.lovnum);
4331 fflush(DBG);
4332 }
4333 bbOOff=bbOOee+bbOOf-bbOO.lovlo;  /* simset posn. in final dimension
       */
4334   bbOO.T[0].S[bbOOff].n=bbOO.nT*bbOO.hiK;
4335   bbOO.T[0].S[bbOOff].f=\
4336     (double*)calloc(bbOO.T[0].S[bbOOff].n,sizeof(double));
4337   bbOO.T[0].S[bbOOff].ni=\
4338     (double*)calloc(bbOO.T[0].S[bbOOff].n,sizeof(double));
4339   bbOO.T[0].S[bbOOff].ci=\
4340     (double*)calloc(bbOO.T[0].S[bbOOff].n,sizeof(double));
4341 bbOO.Rl=&bbOO.Sl[0].rc[bbOOfrcl];
4342 if(bbOOdebugflag<DEBUG){
4343 fprintf(DBG,"bbOOff=%d, bbOOfrcl=%d,
    bbOO.T[0].S[bbOOff].n=%d\n",\
4344     bbOOff,bbOOfrcl,bbOO.T[0].S[bbOOff].n);
4345 fflush(DBG);
4346 }
     /* start with highest J value (same reason as before), and loop down
        looking for lower J's to which to transit. assign intensities.
        If both states are Omega=0, then assign zero intensity to transi-
        tions for J=0<->J=0. */
4347   if(bbOOfrnhf==0){ /* if this v is natively populated.*/
4348 sprintf(bbOOfile,"%s_molecules/%s/%s--%s_v%d--v%d_NAT.dat",\
4349   PREF,MOL[bbOO.m].Mol,bbOO.T[0].Nhi,bbOO.T[0].Nlo,bbOOe,bbOOf);
4350     BBOOFILE=fopen(bbOOfile,"w");
4351     if(BBOOFILE==NULL){
4352 printf("Error opening transition sub-file %s. Exit-
    ing.\n",bbOOfile);
4353       exit(1);
4354       }
4355 fprintf(BBOOFILE,"# File created by %s.\n# File contains ",PRO-
    GRAM_NAME);
4356 fprintf(BBOOFILE,"P, Q and R");
4357 fprintf(BBOOFILE," branches from the simulation titled %s\n",PREF);
4358 fprintf(BBOOFILE,"# THESE INTENSITIES ARE DUE TO USER-SPECIFIED
    POPULATIONS");
4359 fprintf(BBOOFILE," ONLY -- NO CASCADE\n");
4360 fprintf(BBOOFILE,"# Transition is Hund's Case (b) to Hund's Case
    (b).\n");
4361 fprintf(BBOOFILE,"# Molecule %s, \tHigh state: %s, v=%d\n",\
4362     MOL[bbOO.m].Mol,bbOO.T[0].Nhi,bbOOe);
4363 fprintf(BBOOFILE,"#\t\tLow state: %s, v=%d\n# Columns are: ",\
4364     bbOO.T[0].Nlo,bbOOf);
4365 fprintf(BBOOFILE,"K_hi K_lo J_hi P(J_hi+1)_v P_i ");
4366 fprintf(BBOOFILE,"Q(J_hi+0)_v Q_i R(J_hi-1)_v R_i\n#\n");

4367     bbOO.Rh=&bbOO.Sh[0].r[bbOOfrnh];
4368     if((bbOO.hiK)<bbOO.Sh[0].r[bbOOfrnh].k){
```

```
4369       bbOOk=bbOO.hiK-1;
4370       }
4371     else bbOOk=bbOO.Sh[0].r[bbOOfrnh].k-1;
4372     if(bbOO.Rl[0].Kdissoc<(bbOOk-1)){
4373 fprintf(PAR,"\nWARNING!! State %s of molecule %s has significant
    ",\
4374     bbOO.T[0].Nhi,MOL[bbOO.m].Mol);
4375 fprintf(PAR,"rotational population at level %d\n",bbOOk);
4376 fprintf(PAR,"\tBut lower state %s dissociates at level %.1f\n",\
4377     bbOO.T[0].Nlo,bbOO.Rl[0].Kdissoc);
4378 fprintf(PAR,"It is likely that the bound upper state is transiting
    to (an)");
4379 fprintf(PAR," unbound lower state(s).\n");
4380 fprintf(PAR,"Look for continuum emissions trailing off to the low-
    energy ");
4381 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real
    ",bbOOe,bbOOf);
4382 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
    gram will");
4383 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
4384       bbOOk=(int)(bbOO.Rl[0].Kdissoc);
4385       }
4386 if(bbOOdebugflag<DEBUG){
4387 fprintf(DBG,"bbOOfrnh=%d, bbOOk=%d, ",bbOOfrnh,bbOOk);
4388 fprintf(DBG, "bbOO.Sh[0].r[%d].k = %d\n", bbOOfrnh,
    bbOO.Sh[0].r[bbOOfrnh].k);
4389 fflush(DBG);
4390 }
     /* To make the output file easy to write, I'm cycling by high-state
        K first, then by high-state J. From that J/K(hi) pair, the pro-
        gram will look for a P transition and an R transition to K+1 then
        to K-1. If all the necessary states exist, it will check to see
        if there are any symmetry or nuclear spin effects to be concerned
        about. Taking these into account, it will calculate the transi-
        tion. */
     /* loop first by upper-state K-value */
4391 for(bbOOh=bbOOk;bbOOh>=0;bbOOh--){
4392   bbOOjii=(bbOOh+1)*bbOO.nT-1;  /* position in simset array */
4393   bbOOeh=bbOOh*bbOO.nhJ;  /* position in 2D for high EJ & PJ */
     /* then loop by upper-state J for this K */
4394   for(bbOOi=(bbOO.nhJ-1);bbOOi>=0;bbOOi--){
4395     bbOOJ=bbOOh+bbOOi-bbOOSh;
4396     bbOOehh=bbOOeh+bbOOi;
4397     bbOOflga=0;
4398     if(bbOOh<bbOOHI[0].L) bbOOflga=1;  /* if this K is less than
        Lambda for the high state, don't proceed */
4399     if(bbOOJ<0) bbOOflga=1;  /* if J is less than zero, don't */
4400 if(bbOOdebugflag<DEBUG){
4401 fprintf(DBG,"bbOOh=%d, bbOOi=%d, bbOOSh=%.1f, bbOOHI[0].L=%d,
    bbOOflga=%d\n",\
4402     bbOOh,bbOOi,bbOOSh,bbOOHI[0].L,bbOOflga);
4403 fflush(DBG);
4404 }
4405     if(bbOOflga==0){
```

```
      /* Check Delta-J=+1 (P), then Delta-J=0 (Q), and then Delta-J=-1
        (R) for Delta-K=+1. First, check to see if the target K exists:
        */
4406        bbOOflgb=0;
4407        if((bbOOh+1)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Relative position in the low state E/P array */
4408        bbOOel=(bbOOh+1)*bbOO.nlJ + bbOO.nlJ-bbOO.nhJ + \
4409          (int)(bbOOSh-bbOOSl) -1 + bbOOi;
4410  if(bbOOdebugflag<DEBUG){
4411  fprintf(DBG,"\t(bbOOh+1)*bbOO.nlJ=%d, bbOO.nlJ-bbOO.nhJ=%d\n",\
4412      (bbOOh+1)*bbOO.nlJ,bbOO.nlJ-bbOO.nhJ);
4413  fprintf(DBG,"\t(int)(bbOOSh-bbOOSl)=%d, bbOOi=%d, bbOOel=%d\n",\
4414      (int)(bbOOSh-bbOOSl),bbOOi,bbOOel);
4415  fflush(DBG);
4416  }
4417        if((bbOOel+1)>=(bbOO.Rl[0].kc*bbOO.nlJ)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
4418        if((bbOOJ+1)>(bbOOh+1+bbOOSl)) bbOOflgb=1;
4419        if((bbOOJ+1)<(bbOOh+1-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
4420        if(bbOOflgb==0){
4421          bbOOB=+1;
4422          bbOOell=bbOOel+1;
4423          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4424            bbOO.Rh[0].EJ[bbOOehh]-\
4425            bbOO.Rl[0].EJ[bbOOell];
4426          bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4427            bbOOhipop*bbOOtrprob*bbOOvhpop*\
4428            bbOO.Rh[0].PJ[bbOOehh]*bbOOfcf*\
4429            HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
      /* set low-state population for cascade */
4430          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4431            bbOO.T[0].S[bbOOff].ni[bbOOjii];
4432          bbOOjii--;
4433          }
4434  if(bbOOdebugflag<DEBUG){
4435  fprintf(DBG,"%d ",bbOOh);
4436  if(bbOOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbOOh+1),bbOOJ);
4437  else{
4438  fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbOOh+1),\
4439    bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4440    bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4441    }
4442  }
4443  fprintf(BBOOFILE,"%d ",bbOOh);
4444  if(bbOOflgb!=0) fprintf(BBOOFILE," %d %.1f M M ",(bbOOh+1),bbOOJ);
4445  else{
4446  fprintf(BBOOFILE,"%d %.1f %18.12e %18.12e ",(bbOOh+1),\
4447    bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4448    bbOO.T[0].S[bbOOff].ni[bbOOjii+1]); }

      /* Now Delta-J=0 (Q) (for Delta-K=+1) */
4449        bbOOflgb=0;
```

```
4450        if((bbOOh+1)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
         than the allowed J's for K+1. */
4451        if((bbOOJ+0)>(bbOOh+1+bbOOSl)) bbOOflgb=1;
4452        if((bbOOJ+0)<(bbOOh+1-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
4453        if(bbOOflgb==0){
4454          bbOOB=0;
4455          bbOOell=bbOOel+0;
4456          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4457            bbOO.Rh[0].EJ[bbOOehh]-\
4458            bbOO.Rl[0].EJ[bbOOell];
4459          bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4460            bbOOhipop*bbOOtrprob*bbOOvhpop*\
4461            bbOO.Rh[0].PJ[bbOOehh]*bbOOSATT*\
4462            HL(bbOOB,bbOOJ,bbOOL,bbOOdL)*bbOOfcf;
      /* set low-state population for cascade */
4463          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4464            bbOO.T[0].S[bbOOff].ni[bbOOjii];
4465          bbOOjii--;
4466          }
4467 if(bbOOdebugflag<DEBUG){
4468 fprintf(DBG,"%d ",bbOOh);
4469 if(bbOOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbOOh+1),bbOOJ);
4470 else{
4471 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbOOh+1),\
4472   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4473   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4474   }
4475 }
4476 if(bbOOflgb!=0) fprintf(BBOOFILE," M M ");
4477 else{
4478 fprintf(BBOOFILE,"%18.12e %18.12e
      ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4479   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4480   }
      /* Now, Delta-J=-1 (R) for Delta-K=+1. */
4481        bbOOflgb=0;
4482        if((bbOOh+1)<(bbOOLO[0].L)) bbOOflgb=1;
4483        if((bbOOJ-1)>(bbOOh+1+bbOOSl)) bbOOflgb=1;
4484        if((bbOOJ-1)<(bbOOh+1-bbOOSl)) bbOOflgb=1;
4485        if((bbOOJ-1)<0) bbOOflgb=1;
4486        if(bbOOflgb==0){
4487          bbOOB=-1;
4488          bbOOell=bbOOel-1;
4489          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4490            bbOO.Rh[0].EJ[bbOOehh]-\
4491            bbOO.Rl[0].EJ[bbOOell];
4492          bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4493            bbOOhipop*bbOOtrprob*bbOOvhpop*\
4494            bbOO.Rh[0].PJ[bbOOehh]*bbOOSATT*\
4495            HL(bbOOB,bbOOJ,bbOOL,bbOOdL)*bbOOfcf;
4496          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4497            bbOO.T[0].S[bbOOff].ni[bbOOjii];
```

```
4498          bbOOjii--;
4499          }
4500 if(bbOOdebugflag<DEBUG){
4501 if(bbOOflgb!=0) fprintf(DBG,"M M\n");
4502 else{
4503 fprintf(DBG,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4504   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4505   }
4506 }
4507 if(bbOOflgb!=0) fprintf(BBOOFILE,"M M\n");
4508 else{
4509 fprintf(BBOOFILE,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4510   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4511   }
      /* Check Delta-J=+1 (P), then Delta-J=0 (Q), and then Delta-J=-1
         (R) for Delta-K=+0. First, check to see if the target K exists:
         */
4512        bbOOflgb=0;
4513        if((bbOOh+0)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Relative position in the low state E/P array */
4514        bbOOel=(bbOOh+0)*bbOO.nlJ + bbOO.nlJ-bbOO.nhJ + \
4515          (int)(bbOOSh-bbOOSl) + 0 + bbOOi;
4516 if(bbOOdebugflag<DEBUG){
4517 fprintf(DBG,"\t(bbOOh+0)*bbOO.nlJ=%d, bbOO.nlJ-bbOO.nhJ=%d\n",\
4518     (bbOOh+0)*bbOO.nlJ,bbOO.nlJ-bbOO.nhJ);
4519 fprintf(DBG,"\t(int)(bbOOSh-bbOOSl)=%d, bbOOi=%d, bbOOel=%d\n",\
4520     (int)(bbOOSh-bbOOSl),bbOOi,bbOOel);
4521 fflush(DBG);
4522 }
4523        if((bbOOel+1)>=(bbOO.Rl[0].kc*bbOO.nlJ)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+0. */
4524        if((bbOOJ+1)>(bbOOh+0+bbOOSl)) bbOOflgb=1;
4525        if((bbOOJ+1)<(bbOOh+0-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
4526        if(bbOOflgb==0){
4527          bbOOB=+1;
4528          bbOOell=bbOOel+1;
4529          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4530            bbOO.Rh[0].EJ[bbOOehh]-\
4531            bbOO.Rl[0].EJ[bbOOell];
4532          bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4533            bbOOhipop*bbOOtrprob*bbOOvhpop*\
4534            bbOO.Rh[0].PJ[bbOOehh]*bbOOSATT*\
4535            HL(bbOOB,bbOOJ,bbOOL,bbOOdL)*bbOOfcf;
      /* set low-state population for cascade */
4536          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4537            bbOO.T[0].S[bbOOff].ni[bbOOjii];
4538          bbOOjii--;
4539          }
4540 if(bbOOdebugflag<DEBUG){
4541 fprintf(DBG,"%d ",bbOOh);
```

```
4542 if(bbOOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbOOh+0),bbOOJ);
4543 else{
4544 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbOOh+0),\
4545   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4546   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4547   }
4548 }
4549 fprintf(BBOOFILE,"%d ",bbOOh);
4550 if(bbOOflgb!=0) fprintf(BBOOFILE," %d %.1f M M ",(bbOOh+0),bbOOJ);
4551 else{
4552 fprintf(BBOOFILE,"%d %.1f %18.12e %18.12e ",(bbOOh+0),\
4553   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4554   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4555   }
     /* Now Delta-J=0 (Q) (for Delta-K=+0) */
4556       bbOOflgb=0;
4557       if((bbOOh+0)<(bbOOLO[0].L)) bbOOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+0. */
4558       if((bbOOJ+0)>(bbOOh+0+bbOOSl)) bbOOflgb=1;
4559       if((bbOOJ+0)<(bbOOh+0-bbOOSl)) bbOOflgb=1;
     /* Still OK? Calculate transition */
4560       if(bbOOflgb==0){
4561         bbOOB=0;
4562         bbOOell=bbOOel+0;
4563         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4564           bbOO.Rh[0].EJ[bbOOehh]-\
4565           bbOO.Rl[0].EJ[bbOOell];
4566         bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4567           bbOOhipop*bbOOtrprob*bbOOvhpop*\
4568           bbOO.Rh[0].PJ[bbOOehh]*bbOOfcf*\
4569           HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
     /* set low-state population for cascade */
4570         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4571           bbOO.T[0].S[bbOOff].ni[bbOOjii];
4572         bbOOjii--;
4573         }
4574 if(bbOOdebugflag<DEBUG){
4575 fprintf(DBG,"%d ",bbOOh);
4576 if(bbOOflgb!=0) fprintf(DBG," M M ");
4577 else{
4578 fprintf(DBG," %18.12e %18.12e ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4579   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4580   }
4581 }
4582 if(bbOOflgb!=0) fprintf(BBOOFILE," M M ");
4583 else{
4584 fprintf(BBOOFILE,"%18.12e %18.12e
     ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4585   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4586   }
     /* Now, Delta-J=-1 (R) for Delta-K=+0. */
4587       bbOOflgb=0;
4588       if((bbOOh+0)<(bbOOLO[0].L)) bbOOflgb=1;
```

```
4589        if((bbOOJ-1)>(bbOOh+0+bbOOSl)) bbOOflgb=1;
4590        if((bbOOJ-1)<(bbOOh+0-bbOOSl)) bbOOflgb=1;
4591        if((bbOOJ-1)<0) bbOOflgb=1;
4592        if(bbOOflgb==0){
4593          bbOOB=-1;
4594          bbOOell=bbOOel-1;
4595          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4596            bbOO.Rh[0].EJ[bbOOehh]-\
4597            bbOO.Rl[0].EJ[bbOOell];
4598          bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4599            bbOOhipop*bbOOtrprob*bbOOvhpop*\
4600            bbOO.Rh[0].PJ[bbOOehh]*bbOOSATT*\
4601            HL(bbOOB,bbOOJ,bbOOL,bbOOdL)*bbOOfcf;
4602          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4603            bbOO.T[0].S[bbOOff].ni[bbOOjii];
4604          bbOOjii--;
4605          }
4606 if(bbOOdebugflag<DEBUG){
4607 if(bbOOflgb!=0) fprintf(DBG,"M M\n");
4608 else{
4609 fprintf(DBG,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4610   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4611   }
4612 }
4613 if(bbOOflgb!=0) fprintf(BBOOFILE,"M M\n");
4614 else{
4615 fprintf(BBOOFILE,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4616   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4617   }
      /* Check Delta-J=+1 (P), then Delta-J=0 and Delta-J=-1 (R) for
        Delta-K=-1. First, check to see if the target K exists: */
4618        bbOOflgb=0;
4619        if((bbOOh-1)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Relative position in the low state E/P array */
4620        bbOOel=(bbOOh-1)*bbOO.nlJ + bbOO.nlJ-bbOO.nhJ + \
4621          (int)(bbOOSh-bbOOSl) + 1 + bbOOi;
4622 if(bbOOdebugflag<DEBUG){
4623 fprintf(DBG,"\t(bbOOh-1)*bbOO.nlJ=%d, bbOO.nlJ-bbOO.nhJ=%d\n",\
4624     (bbOOh-1)*bbOO.nlJ,bbOO.nlJ-bbOO.nhJ);
4625 fprintf(DBG,"\t(int)(bbOOSh-bbOOSl)=%d, bbOOi=%d, bbOOel=%d\n",\
4626     (int)(bbOOSh-bbOOSl),bbOOi,bbOOel);
4627 fflush(DBG);
4628 }
4629        if((bbOOel+1)>=(bbOO.Rl[0].kc*bbOO.nlJ)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K-1. */
4630        if((bbOOJ+1)>(bbOOh-1+bbOOSl)) bbOOflgb=1;
4631        if((bbOOJ+1)<(bbOOh-1-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
4632        if(bbOOflgb==0){
4633          bbOOB=+1;
4634          bbOOell=bbOOel+1;
```

```
4635          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4636            bbOO.Rh[0].EJ[bbOOehh]-\
4637            bbOO.Rl[0].EJ[bbOOell];
4638          bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4639            bbOOhipop*bbOOtrprob*bbOOvhpop*\
4640            bbOO.Rh[0].PJ[bbOOehh]*bbOOSATT*\
4641            HL(bbOOB,bbOOJ,bbOOL,bbOOdL)*bbOOfcf;
4642          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4643            bbOO.T[0].S[bbOOff].ni[bbOOjii];
4644          bbOOjii--;
4645          }
4646 if(bbOOdebugflag<DEBUG){
4647 fprintf(DBG,"%d ",bbOOh);
4648 if(bbOOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbOOh-1),bbOOJ);
4649 else{
4650 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbOOh-1),\
4651   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4652   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4653   }
4654 }
4655 fprintf(BBOOFILE,"%d ",bbOOh);
4656 if(bbOOflgb!=0) fprintf(BBOOFILE," %d %.1f M M ",(bbOOh-1),bbOOJ);
4657 else{
4658 fprintf(BBOOFILE,"%d %.1f %18.12e %18.12e ",(bbOOh-1),\
4659   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4660   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4661   }
     /* Now, Delta-J=+0 (Q) for Delta-K=-1. */
4662       bbOOflgb=0;
4663       if((bbOOh-1)<(bbOOLO[0].L)) bbOOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K-1. */
4664       if((bbOOJ+0)>(bbOOh-1+bbOOSl)) bbOOflgb=1;
4665       if((bbOOJ+0)<(bbOOh-1-bbOOSl)) bbOOflgb=1;
     /* Still OK? Calculate transition */
4666       if(bbOOflgb==0){
4667         bbOOB=0;
4668         bbOOell=bbOOel+0;
4669         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4670           bbOO.Rh[0].EJ[bbOOehh]-\
4671           bbOO.Rl[0].EJ[bbOOell];
4672         bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4673           bbOOhipop*bbOOtrprob*bbOOvhpop*\
4674           bbOO.Rh[0].PJ[bbOOehh]*bbOOSATT*\
4675           HL(bbOOB,bbOOJ,bbOOL,bbOOdL)*bbOOfcf;
4676         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4677           bbOO.T[0].S[bbOOff].ni[bbOOjii];
4678         bbOOjii--;
4679         }
4680 if(bbOOdebugflag<DEBUG){
4681 fprintf(DBG,"%d ",bbOOh);
4682 if(bbOOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbOOh-1),bbOOJ);
4683 else{
4684 fprintf(DBG,"%d %.1f %18.12e %18.12e ",(bbOOh-1),\
```

```
4685   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4686   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4687   }
4688 }
4689 if(bbOOflgb!=0) fprintf(BBOOFILE," M M ");
4690 else{
4691 fprintf(BBOOFILE," %18.12e %18.12e
     ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4692   bbOO.T[0].S[bbOOff].ni[bbOOjii+1]);
4693   }
     /* Now, Delta-J=-1 (R) for Delta-K=-1. */
4694       bbOOflgb=0;
4695       if((bbOOh-1)<(bbOOLO[0].L)) bbOOflgb=1;
4696       if((bbOOJ-1)>(bbOOh-1+bbOOSl)) bbOOflgb=1;
4697       if((bbOOJ-1)<(bbOOh-1-bbOOSl)) bbOOflgb=1;
4698       if((bbOOJ-1)<0) bbOOflgb=1;
4699       if(bbOOflgb==0){
4700         bbOOB=-1;
4701         bbOOell=bbOOel-1;
4702         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4703           bbOO.Rh[0].EJ[bbOOehh]-\
4704           bbOO.Rl[0].EJ[bbOOell];
4705         bbOO.T[0].S[bbOOff].ni[bbOOjii]=\
4706           bbOOhipop*bbOOtrprob*bbOOvhpop*\
4707           bbOO.Rh[0].PJ[bbOOehh]*bbOOfcf*\
4708          HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4709         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4710           bbOO.T[0].S[bbOOff].ni[bbOOjii];
4711         }
4712 if(bbOOdebugflag<DEBUG){
4713 if(bbOOflgb!=0) fprintf(DBG,"M M\n");
4714 else{
4715 fprintf(DBG,"%18.12e %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii],\
4716   bbOO.T[0].S[bbOOff].ni[bbOOjii]);
4717   }
4718 }
4719 if(bbOOflgb!=0) fprintf(BBOOFILE,"M M\n");
4720 else{
4721 fprintf(BBOOFILE,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii],\
4722   bbOO.T[0].S[bbOOff].ni[bbOOjii]);
4723   }
4724       } /* close if bbOOflga is zero. */
4725     } /* close upper-state J-of-K loop */
4726   } /* close upper-state K loop */
4727   fclose(BBOOFILE);
4728     } /* close if high-state v is natively populated */

4729   if(bbOOfrchf==0){ /* if this v is cascade populated.*/
4730 sprintf(bbOOfile,"%s_molecules/%s/%s--%s_v%d--v%d_CAS.dat",\
4731   PREF,MOL[bbOO.m].Mol,bbOO.T[0].Nhi,bbOO.T[0].Nlo,bbOOe,bbOOf);
4732     BBOOFILE=fopen(bbOOfile,"w");
4733     if(BBOOFILE==NULL){
```

```
4734 printf("Error opening transition sub-file %s. Exit-
     ing.\n",bbOOfile);
4735       exit(1);
4736       }
4737 fprintf(BBOOFILE,"# File created by %s.\n# File contains ",PRO-
     GRAM_NAME);
4738 fprintf(BBOOFILE,"P and R");
4739 fprintf(BBOOFILE," branches from the simulation titled %s\n",PREF);
4740 fprintf(BBOOFILE,"# THESE INTENSITIES ARE DUE TO CASCADE ONLY -- ");
4741 fprintf(BBOOFILE,"NO USER-DEFINED POPULATIONS\n");
4742 fprintf(BBOOFILE,"# Transition is Hund's Case (b) to Hund's Case
     (b).\n");
4743 fprintf(BBOOFILE,"# Molecule %s, \tHigh state: %s, v=%d\n",\
4744     MOL[bbOO.m].Mol,bbOO.T[0].Nhi,bbOOe);
4745 fprintf(BBOOFILE,"#\t\tLow state: %s, v=%d\n# Columns are: ",\
4746     bbOO.T[0].Nlo,bbOOf);
4747 fprintf(BBOOFILE,"K_hi K_lo J_hi P(J_hi+1)_v P_i R(J_hi-1)_v
     R_i\n#\n");

4748     bbOO.Rh=&bbOO.Sh[0].rc[bbOOfrch];
4749     if((bbOO.hiK)<bbOO.Sh[0].rc[bbOOfrch].kc){
4750       bbOOk=bbOO.hiK-1;
4751       }
4752     else bbOOk=bbOO.Sh[0].rc[bbOOfrch].kc-1;
4753     if(bbOO.Rl[0].Kdissoc<(bbOOk-1)){
4754 fprintf(PAR,"\nWARNING!! State %s of molecule %s has significant
     ",\
4755     bbOO.T[0].Nhi,MOL[bbOO.m].Mol);
4756 fprintf(PAR,"CASCADE rotational population at level %d\n",bbOOk);
4757 fprintf(PAR,"\tBut lower state %s dissociates at level %.1f\n",\
4758     bbOO.T[0].Nlo,bbOO.Rl[0].Kdissoc);
4759 fprintf(PAR,"It is likely that the bound upper state is transiting
     to (an)");
4760 fprintf(PAR," unbound lower state(s).\n");
4761 fprintf(PAR,"Look for continuum emissions trailing off to the low-
     energy ");
4762 fprintf(PAR,"end of the v(hi)=%d to v(lo)=%d\nband in the real
     ",bbOOe,bbOOf);
4763 fprintf(PAR,"spectrum (not the simulated one)\nCurrently, this pro-
     gram will");
4764 fprintf(PAR," not simulate bound-->unbound spectra.\n\n");
4765       bbOOk=(int)(bbOO.Rl[0].Kdissoc);
4766       }
4767 if(bbOOdebugflag<DEBUG){
4768 fprintf(DBG,"bbOOfrch=%d, bbOOk=%d, ",bbOOfrch,bbOOk);
4769 fprintf(DBG, "bbOO.Sh[0].rc[%d].kc = %d\n", bbOOfrch,
     bbOO.Sh[0].rc[bbOOfrch].kc);
4770 fflush(DBG);
4771 }
      /* To make the output file easy to write, I'm cycling by high-state
        K first, then by high-state J. From that J/K(hi) pair, the pro-
        gram will look for a P transition and an R transition to K+1 then
        to K-1. If all the necessary states exist, it will check to see
        if there are any symmetry or nuclear spin effects to be concerned
```

```
        about. Taking these into account, it will calculate the transi-
        tion. */
      /* loop first by upper-state K-value */
4772  for(bbOOh=(bbOOk-1);bbOOh>=0;bbOOh--){
4773    bbOOjii=(bbOOh+1)*bbOO.nT-1;  /* position in simset array */
4774    bbOOeh=bbOOh*bbOO.nhJ;  /* position in 2D for high EJ & PJ */
      /* then loop by upper-state J for this K */
4775    for(bbOOi=(bbOO.nhJ-1);bbOOi>=0;bbOOi--){
4776      bbOOJ=bbOOh+bbOOi-bbOOSh;
4777      bbOOehh=bbOOeh+bbOOi;
4778      bbOOflga=0;
4779      if(bbOOh<bbOOHI[0].L) bbOOflga=1;  /* if this K is less than
        Lambda for the high state, don't proceed */
4780      if(bbOOJ<0) bbOOflga=1;  /* if J is less than zero, don't */
4781  if(bbOOdebugflag<DEBUG){
4782  fprintf(DBG,"bbOOh=%d, bbOOi=%d, bbOOSh=%.1f, bbOOHI[0].L=%d,
     bbOOflga=%d\n",\
4783      bbOOh,bbOOi,bbOOSh,bbOOHI[0].L,bbOOflga);
4784  fflush(DBG);
4785  }
4786      if(bbOOflga==0){
      /* Check Delta-J=+1 (P), then Delta-J=0 (Q) and Delta-J=-1 (R) for
        Delta-K=+1. First, check to see if the target K exists: */
4787        bbOOflgb=0;
4788        if((bbOOh+1)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Relative position in the low state E/P array */
4789        bbOOel=(bbOOh+1)*bbOO.nlJ + bbOO.nlJ-bbOO.nhJ + \
4790          (int)(bbOOSh-bbOOSl) -1 + bbOOi;
4791  if(bbOOdebugflag<DEBUG){
4792  fprintf(DBG,"\t(bbOOh+1)*bbOO.nlJ=%d, bbOO.nlJ-bbOO.nhJ=%d\n",\
4793      (bbOOh+1)*bbOO.nlJ,bbOO.nlJ-bbOO.nhJ);
4794  fprintf(DBG,"\t(int)(bbOOSh-bbOOSl)=%d, bbOOi=%d, bbOOel=%d\n",\
4795      (int)(bbOOSh-bbOOSl),bbOOi,bbOOel);
4796  fflush(DBG);
4797  }
4798        if((bbOOel+1)>=(bbOO.Rl[0].kc*bbOO.nlJ)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
4799        if((bbOOJ+1)>(bbOOh+1+bbOOSl)) bbOOflgb=1;
4800        if((bbOOJ+1)<(bbOOh+1-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
4801        if(bbOOflgb==0){
4802          bbOOB=+1;
4803          bbOOell=bbOOel+1;
4804          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4805            bbOO.Rh[0].EJ[bbOOehh]-\
4806            bbOO.Rl[0].EJ[bbOOell];
4807          bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
4808            bbOO.Rh[0].CJ[bbOOehh]*\
4809            HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4810          bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
4811            bbOOcascadetemp;
4812          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4813            bbOOcascadetemp;
```

```
4814          bbOOjii--;
4815          }
4816 if(bbOOdebugflag<DEBUG){
4817 fprintf(DBG,"%d ",bbOOh);
4818 if(bbOOflgb!=0) fprintf(DBG," %d %.1f (1c) M M\n",(bbOOh+1),bbOOJ);
4819 else{
4820 fprintf(DBG,"%d %.1f %18.12e %18.12e (1c)\n",(bbOOh+1),\
4821   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4822   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4823   }
4824 }
4825 fprintf(BBOOFILE,"%d ",bbOOh);
4826 if(bbOOflgb!=0) fprintf(BBOOFILE," %d %.1f M M ",(bbOOh+1),bbOOJ);
4827 else{
4828 fprintf(BBOOFILE,"%d %.1f %18.12e %18.12e ",(bbOOh+1),\
4829   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4830   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4831   }
     /* Now, Delta-J=0 (Q), Delta-K=+1 */
4832       bbOOflgb=0;
4833       if((bbOOh+1)<(bbOOLO[0].L)) bbOOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+1. */
4834       if((bbOOJ+0)>(bbOOh+1+bbOOSl)) bbOOflgb=1;
4835       if((bbOOJ+0)<(bbOOh+1-bbOOSl)) bbOOflgb=1;
     /* Still OK? Calculate transition */
4836       if(bbOOflgb==0){
4837         bbOOB=0;
4838         bbOOell=bbOOel+0;
4839         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4840           bbOO.Rh[0].EJ[bbOOehh]-\
4841           bbOO.Rl[0].EJ[bbOOell];
4842         bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
4843           bbOO.Rh[0].CJ[bbOOehh]*bbOOSATT*\
4844           HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4845         bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
4846           bbOOcascadetemp;
4847         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4848           bbOOcascadetemp;
4849         bbOOjii--;
4850         }
4851 if(bbOOdebugflag<DEBUG){
4852 if(bbOOflgb!=0) fprintf(DBG," M M\n");
4853 else{
4854 fprintf(DBG,"%18.12e %18.12e
     (1c)\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4855   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4856   }
4857 }
4858 if(bbOOflgb!=0) fprintf(BBOOFILE," M M ");
4859 else{
4860 fprintf(BBOOFILE," %18.12e %18.12e
     ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4861   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
```

```
4862   }
     /* Now, Delta-J=-1 (R) for Delta-K=+1. */
4863       bbOOflgb=0;
4864       if((bbOOh+1)<(bbOOLO[0].L)) bbOOflgb=1;
4865       if((bbOOJ-1)>(bbOOh+1+bbOOSl)) bbOOflgb=1;
4866       if((bbOOJ-1)<(bbOOh+1-bbOOSl)) bbOOflgb=1;
4867       if((bbOOJ-1)<0) bbOOflgb=1;
4868       if(bbOOflgb==0){
4869         bbOOB=-1;
4870         bbOOell=bbOOel-1;
4871         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4872           bbOO.Rh[0].EJ[bbOOehh]-\
4873           bbOO.Rl[0].EJ[bbOOell];
4874         bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
4875           bbOO.Rh[0].CJ[bbOOehh]*bbOOSATT*\
4876          HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4877         bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
4878           bbOOcascadetemp;
4879         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4880           bbOOcascadetemp;
4881         bbOOjii--;
4882         }
4883 if(bbOOdebugflag<DEBUG){
4884 if(bbOOflgb!=0) fprintf(DBG,"M M\n");
4885 else{
4886 fprintf(DBG,"%18.12e %18.12e
     (1d)\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4887   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4888   }
4889 }
4890 if(bbOOflgb!=0) fprintf(BBOOFILE,"M M\n");
4891 else{
4892 fprintf(BBOOFILE,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4893   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4894   }
     /* Now, again for Delta-K=+0. Delta-J=+1 (P), then (Q) and (R) */
4895       bbOOflgb=0;
4896       if((bbOOh+0)<(bbOOLO[0].L)) bbOOflgb=1;
     /* Relative position in the low state E/P array */
4897       bbOOel=(bbOOh+0)*bbOO.nlJ + bbOO.nlJ-bbOO.nhJ + \
4898         (int)(bbOOSh-bbOOSl) + 0 + bbOOi;
4899 if(bbOOdebugflag<DEBUG){
4900 fprintf(DBG,"\t(bbOOh+0)*bbOO.nlJ=%d, bbOO.nlJ-bbOO.nhJ=%d\n",\
4901     (bbOOh+0)*bbOO.nlJ,bbOO.nlJ-bbOO.nhJ);
4902 fprintf(DBG,"\t(int)(bbOOSh-bbOOSl)=%d, bbOOi=%d, bbOOel=%d\n",\
4903     (int)(bbOOSh-bbOOSl),bbOOi,bbOOel);
4904 fflush(DBG);
4905 }
4906       if((bbOOel+1)>=(bbOO.Rl[0].kc*bbOO.nlJ)) bbOOflgb=1;
     /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+0. */
4907       if((bbOOJ+1)>(bbOOh+0+bbOOSl)) bbOOflgb=1;
4908       if((bbOOJ+1)<(bbOOh+0-bbOOSl)) bbOOflgb=1;
```

```
      /* Still OK? Calculate transition */
4909        if(bbOOflgb==0){
4910          bbOOB=+1;
4911          bbOOell=bbOOel+1;
4912          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4913            bbOO.Rh[0].EJ[bbOOehh]-\
4914            bbOO.Rl[0].EJ[bbOOell];
4915          bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
4916            bbOO.Rh[0].CJ[bbOOehh]*bbOOSATT*\
4917            HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4918          bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
4919            bbOOcascadetemp;
4920          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4921            bbOOcascadetemp;
4922          bbOOjii--;
4923          }
4924 if(bbOOdebugflag<DEBUG){
4925 fprintf(DBG,"%d ",bbOOh);
4926 if(bbOOflgb!=0) fprintf(DBG," %d %.1f (1c) M M\n",(bbOOh+0),bbOOJ);
4927 else{
4928 fprintf(DBG,"%d %.1f %18.12e %18.12e (1c)\n",(bbOOh+0),\
4929   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4930   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4931   }
4932 }
4933 fprintf(BBOOFILE,"%d ",bbOOh);
4934 if(bbOOflgb!=0) fprintf(BBOOFILE," %d %.1f M M ",(bbOOh+0),bbOOJ);
4935 else{
4936 fprintf(BBOOFILE,"%d %.1f %18.12e %18.12e ",(bbOOh+0),\
4937   bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4938   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4939   }
      /* Now, Delta-J=0 (Q), Delta-K=+0 */
4940        bbOOflgb=0;
4941        if((bbOOh+0)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K+0. */
4942        if((bbOOJ+0)>(bbOOh+0+bbOOSl)) bbOOflgb=1;
4943        if((bbOOJ+0)<(bbOOh+0-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
4944        if(bbOOflgb==0){
4945          bbOOB=0;
4946          bbOOell=bbOOel+0;
4947          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4948            bbOO.Rh[0].EJ[bbOOehh]-\
4949            bbOO.Rl[0].EJ[bbOOell];
4950          bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
4951            bbOO.Rh[0].CJ[bbOOehh]*\
4952            HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4953          bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
4954            bbOOcascadetemp;
4955          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4956            bbOOcascadetemp;
4957          bbOOjii--;
```

```
4958         }
4959 if(bbOOdebugflag<DEBUG){
4960 if(bbOOflgb!=0) fprintf(DBG," M M\n");
4961 else{
4962 fprintf(DBG,"%18.12e %18.12e
     (1c)\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4963   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4964   }
4965 }
4966 if(bbOOflgb!=0) fprintf(BBOOFILE," M M ");
4967 else{
4968 fprintf(BBOOFILE," %18.12e %18.12e
     ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4969   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4970   }
      /* Now, Delta-J=-1 (R) for Delta-K=+0. */
4971       bbOOflgb=0;
4972       if((bbOOh+0)<(bbOOLO[0].L)) bbOOflgb=1;
4973       if((bbOOJ-1)>(bbOOh+0+bbOOSl)) bbOOflgb=1;
4974       if((bbOOJ-1)<(bbOOh+0-bbOOSl)) bbOOflgb=1;
4975       if((bbOOJ-1)<0) bbOOflgb=1;
4976       if(bbOOflgb==0){
4977         bbOOB=-1;
4978         bbOOell=bbOOel-1;
4979         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
4980           bbOO.Rh[0].EJ[bbOOehh]-\
4981           bbOO.Rl[0].EJ[bbOOell];
4982         bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
4983           bbOO.Rh[0].CJ[bbOOehh]*bbOOSATT*\
4984          HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
4985         bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
4986           bbOOcascadetemp;
4987         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
4988           bbOOcascadetemp;
4989         bbOOjii--;
4990         }
4991 if(bbOOdebugflag<DEBUG){
4992 if(bbOOflgb!=0) fprintf(DBG,"M M\n");
4993 else{
4994 fprintf(DBG,"%18.12e %18.12e
     (1d)\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
4995   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
4996   }
4997 }
4998 if(bbOOflgb!=0) fprintf(BBOOFILE,"M M\n");
4999 else{
5000 fprintf(BBOOFILE,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
5001   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
5002   }
      /* Check Delta-J=+1 (P), then Delta-J=0 (Q) and Delta-J=-1 (R) for
        Delta-K=-1. First, check to see if the target K exists: */
5003       bbOOflgb=0;
5004       if((bbOOh-1)<(bbOOLO[0].L)) bbOOflgb=1;
```

```
      /* Relative position in the low state E/P array */
5005        bbOOel=(bbOOh-1)*bbOO.nlJ + bbOO.nlJ-bbOO.nhJ + \
5006          (int)(bbOOSh-bbOOSl) + 1 + bbOOi;
5007  if(bbOOdebugflag<DEBUG){
5008  fprintf(DBG,"\t(bbOOh-1)*bbOO.nlJ=%d, bbOO.nlJ-bbOO.nhJ=%d\n",\
5009      (bbOOh-1)*bbOO.nlJ,bbOO.nlJ-bbOO.nhJ);
5010  fprintf(DBG,"\t(int)(bbOOSh-bbOOSl)=%d, bbOOi=%d, bbOOel=%d\n",\
5011      (int)(bbOOSh-bbOOSl),bbOOi,bbOOel);
5012  fflush(DBG);
5013  }
5014        if((bbOOel+1)>=(bbOO.Rl[0].kc*bbOO.nlJ)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K-1. */
5015        if((bbOOJ+1)>(bbOOh-1+bbOOSl)) bbOOflgb=1;
5016        if((bbOOJ+1)<(bbOOh-1-bbOOSl)) bbOOflgb=1;
      /* Still OK? Calculate transition */
5017        if(bbOOflgb==0){
5018          bbOOB=+1;
5019          bbOOell=bbOOel+1;
5020          bbOO.T[0].S[bbOOff].f[bbOOjii]=\
5021            bbOO.Rh[0].EJ[bbOOehh]-\
5022            bbOO.Rl[0].EJ[bbOOell];
5023          bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
5024            bbOO.Rh[0].CJ[bbOOehh]*bbOOSATT*\
5025            HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
5026          bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
5027            bbOOcascadetemp;
5028          bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
5029            bbOOcascadetemp;
5030          bbOOjii--;
5031          }
5032  if(bbOOdebugflag<DEBUG){
5033  fprintf(DBG,"%d ",bbOOh);
5034  if(bbOOflgb!=0) fprintf(DBG," %d %.1f M M ",(bbOOh-1),bbOOJ);
5035  else{
5036  fprintf(DBG,"%d %.1f %18.12e %18.12e (1e)\n",(bbOOh-1),\
5037    bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
5038    bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
5039    }
5040  }
5041  fprintf(BBOOFILE,"%d ",bbOOh);
5042  if(bbOOflgb!=0) fprintf(BBOOFILE," %d %.1f M M ",(bbOOh-1),bbOOJ);
5043  else{
5044  fprintf(BBOOFILE,"%d %.1f %18.12e %18.12e ",(bbOOh-1),\
5045    bbOOJ,bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
5046    bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
5047    }
      /* Now, Delta-J=+0 (Q), Delta-K=-1. */
5048        bbOOflgb=0;
5049        if((bbOOh-1)<(bbOOLO[0].L)) bbOOflgb=1;
      /* Then check if the target value for J+1 is less than or greater
        than the allowed J's for K-1. */
5050        if((bbOOJ+0)>(bbOOh-1+bbOOSl)) bbOOflgb=1;
5051        if((bbOOJ+0)<(bbOOh-1-bbOOSl)) bbOOflgb=1;
```

```
     /* Still OK? Calculate transition */
5052       if(bbOOflgb==0){
5053         bbOOB=0;
5054         bbOOell=bbOOel+0;
5055         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
5056           bbOO.Rh[0].EJ[bbOOehh]-\
5057           bbOO.Rl[0].EJ[bbOOell];
5058         bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
5059           bbOO.Rh[0].CJ[bbOOehh]*bbOOSATT*\
5060           HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
5061         bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
5062           bbOOcascadetemp;
5063         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
5064           bbOOcascadetemp;
5065         bbOOjii--;
5066         }
5067 if(bbOOdebugflag<DEBUG){
5068 if(bbOOflgb!=0) fprintf(DBG," M M ");
5069 else{
5070 fprintf(DBG,"%18.12e %18.12e
    \n",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
5071   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
5072   }
5073 }
5074 if(bbOOflgb!=0) fprintf(BBOOFILE," M M ");
5075 else{
5076 fprintf(BBOOFILE,"%18.12e %18.12e
    ",bbOO.T[0].S[bbOOff].f[bbOOjii+1],\
5077   bbOO.T[0].S[bbOOff].ci[bbOOjii+1]);
5078   }
     /* Now, Delta-J=-1 (R) for Delta-K=-1. */
5079       bbOOflgb=0;
5080       if((bbOOh-1)<(bbOOLO[0].L)) bbOOflgb=1;
5081       if((bbOOJ-1)>(bbOOh-1+bbOOSl)) bbOOflgb=1;
5082       if((bbOOJ-1)<(bbOOh-1-bbOOSl)) bbOOflgb=1;
5083       if((bbOOJ-1)<0) bbOOflgb=1;
5084       if(bbOOflgb==0){
5085         bbOOB=-1;
5086         bbOOell=bbOOel-1;
5087         bbOO.T[0].S[bbOOff].f[bbOOjii]=\
5088           bbOO.Rh[0].EJ[bbOOehh]-\
5089           bbOO.Rl[0].EJ[bbOOell];
5090         bbOOcascadetemp=bbOOtrprob*bbOOfcf*\
5091           bbOO.Rh[0].CJ[bbOOehh]*\
5092          HL(bbOOB,bbOOJ,bbOOL,bbOOdL);
5093         bbOO.T[0].S[bbOOff].ci[bbOOjii]+=\
5094           bbOOcascadetemp;
5095         bbOO.Rl[0].CJ[bbOOell]+=bbOOfcfr*\
5096           bbOOcascadetemp;
5097         }
5098 if(bbOOdebugflag<DEBUG){
5099 if(bbOOflgb!=0) fprintf(DBG,"M M\n");
5100 else{
```

```
5101 fprintf(DBG,"%18.12e %18.12e
     (1f)\n",bbOO.T[0].S[bbOOff].f[bbOOjii],\
5102   bbOO.T[0].S[bbOOff].ci[bbOOjii]);
5103   }
5104 }
5105 if(bbOOflgb!=0) fprintf(BBOOFILE,"M M\n");
5106 else{
5107 fprintf(BBOOFILE,"%18.12e
     %18.12e\n",bbOO.T[0].S[bbOOff].f[bbOOjii],\
5108   bbOO.T[0].S[bbOOff].ci[bbOOjii]);
5109   }
5110       } /* close if bbOOflga is zero. */
5111     } /* close upper-state J-of-K loop */
5112   } /* close upper-state K loop */
5113   fclose(BBOOFILE);
5114     } /* close if high-state v is cascade populated */
5115   } /* close low-state v */
5116   } /* close high-state v */
5117 return;
5118 }

      /*************** case_a_cascade *****************/

      /* This function calculates energy levels only when needed for des-
        tination states in a transition. This function is called from a-
        a.c and a-b.c. For multiplet non-Sigma states, Hund's Case (a).

      The input variables are (1) the rotset where the energy levels
        should be stored, (2) the position in the case-a info array, (3)
        the value of Omega for this set of J's, (4) the vibrational quan-
        tum number, (5) the current highest value of J calculated, (6) the
        highest value needed */
5119 void
5120 case_a_cascade(rotset *cacR,int cacA,double cacO,int cacV,int
     cacJc,int cacJn){
5121 int cacb=0,cacL=0;
5122 double cacJ=0,cacDv=0,cacBe=0,cacAe=0,cacwe=0,cacwexe=0;
5123 double cacbeta=0,cacTe=0,cacDe=0,cacBv=0,cacTeVib=0;

5124 if(DEBUG>cacdebugflag){
5125 fprintf(DBG,"\n\nTop of case_a_cascade\n\n");\
5126 fflush(DBG);
5127 }
5128 cacBe=CA[cacA].Be;
5129 cacAe=CA[cacA].ae;
5130 cacTe=CA[cacA].Te;
5131 cacwe=CA[cacA].we;
5132 cacwexe=CA[cacA].wexe;
5133 cacL=CA[cacA].L;
5134 cacbeta=CA[cacA].beta;
5135 if(DEBUG>cacdebugflag){
5136 fprintf(DBG,"Be=%f ae=%f Te=%f we=%f ",cacBe,cacAe,cacTe,cacwe);
5137 fprintf(DBG,"wexe=%f cacL=%d,
     cacbeta=%f\n",cacwexe,cacL,cacbeta);
5138 fflush(DBG);
```

```
5139 }
5140
     /* Calculate some stuff that will be useful later */
5141 cacDe=4*pow(cacBe,3)/(cacwe*cacwe);  /* Centrifugal distortion */
5142 cacBv=cacBe-cacAe*(cacV+0.5);
5143 cacDv=cacDe+cacbeta*(cacV+0.5);
5144 cacTeVib=cacTe+(cacV+0.5)*cacwe-(cacV+0.5)*(cacV+0.5)*cacwexe;
5145 if(DEBUG>cacdebugflag){
5146 fprintf(DBG,"cacO=%f cacDe=%f cacBv=%f cacDv=%f cacTeVib=%f\n",\
5147     cacO,cacDe,cacBv,cacDv,cacTeVib);
5148 fflush(DBG);
5149 }

     /* There aren't any extra E's here. If someone ever wants to add in a
        splitting factor for the +/- wavefunctions, then double the space
        available (cacJn) and change the indexing and energy calculation
        in the loops below.*/
5150 cacR[0].J=(double*)realloc(cacR[0].J,(cacJn*sizeof(double)));
5151 cacR[0].EJ=(double*)realloc(cacR[0].EJ,(cacJn*sizeof(double)));
5152 cacR[0].CJ=(double*)realloc(cacR[0].CJ,(cacJn*sizeof(double)));
5153 for(cacb=cacJc;cacb<cacJn;cacb++){
5154   cacJ=cacb+cacO;
5155   cacR[0].J[cacb]=cacJ;
5156   cacR[0].EJ[cacb]=cacTeVib+cacBv*(cacJ*(cacJ+1)-cacO*cacO)-\
5157     cacDv*cacJ*cacJ*(cacJ+1)*(cacJ+1);
5158   cacR[0].CJ[cacb]=0;
5159 if(DEBUG>cacdebugflag){
5160 fprintf(DBG,\
5161 "cacb=%d cacR[0].J[cacb]=%f cacR[0].EJ[cacb]=%f
    cacR[0].CJ[cacb]=%f\n",\
5162     cacb,cacJ,cacR[0].EJ[cacb],cacR[0].CJ[cacb]);
5163 fflush(DBG);
5164 }
5165   }
5166 cacR[0].jc=cacJn;
5167 return;
5168 }

     /*************** case_b_cascade_b ****************/

     /* This function calculates energy levels only when needed for des-
        tination states in a transition. This function is called from b-
        b.c.

     The input variables are (1) the rotset where the energy levels
        should be stored, (2) the position in the case-b info array, (3)
        the vibrational quantum number, (4) the current highest value of
        K calculated, (5) the highest value needed, (6) the multiplicity
        (J's per K) */
5169 void
5170 case_b_cascade(rotset *cbcR,int cbcA,int cbcV,int cbcKc,int
    cbcKn,int cbcM){
5171 int cbcb=0,cbcc=0,cbcL=0;
5172 double cbcK=0,cbcDv=0,cbcBe=0,cbcAe=0,cbcwe=0,cbcwexe=0;
5173 double cbcbeta=0,cbcTe=0,cbcDe=0,cbcBv=0,cbcTeVib=0;
```

```
5174 if(DEBUG>cbcdebugflag){
5175 fprintf(DBG,"\n\nTop of case_b_cascade\n\n");\
5176 fprintf(DBG,"cbcA=%d, cbcV=%d, cbcKc=%d, cbcKn=%d\n",\
5177     cbcA,cbcV,cbcKc,cbcKn);
5178 fprintf(DBG,"cbcR[0].j=%d\n",cbcR[0].j);
5179 fflush(DBG);
5180 }
5181 cbcBe=CB[cbcA].Be;
5182 cbcAe=CB[cbcA].ae;
5183 cbcTe=CB[cbcA].Te;
5184 cbcwe=CB[cbcA].we;
5185 cbcwexe=CB[cbcA].wexe;
5186 cbcL=CB[cbcA].L;
5187 cbcbeta=CB[cbcA].beta;
5188 if(DEBUG>cbcdebugflag){
5189 fprintf(DBG,"Be=%f ae=%f Te=%f we=%f ",cbcBe,cbcAe,cbcTe,cbcwe);
5190 fprintf(DBG,"wexe=%f cbcL=%d,
     cbcbeta=%f\n",cbcwexe,cbcL,cbcbeta);
5191 fflush(DBG);
5192 }
5193
      /* Calculate some stuff that will be useful later */
5194 cbcDe=4*pow(cbcBe,3)/(cbcwe*cbcwe);  /* Centrifugal distortion */
5195 cbcBv=cbcBe-cbcAe*(cbcV+0.5);
5196 cbcDv=cbcDe+cbcbeta*(cbcV+0.5);
5197 cbcTeVib=cbcTe+(cbcV+0.5)*cbcwe-(cbcV+0.5)*(cbcV+0.5)*cbcwexe;
5198 if(DEBUG>cbcdebugflag){
5199 fprintf(DBG,"cbcDe=%f cbcBv=%f cbcDv=%f cbcTeVib=%f\n",\
5200     cbcDe,cbcBv,cbcDv,cbcTeVib);
5201 fflush(DBG);
5202 }

      /* There aren't any extra E's here. If someone ever wants to add in a
         splitting factor for the +/- wavefunctions, then double the space
         available (cbcKn) and change the indexing and energy calculation
         in the loops below.*/
5203 cbcR[0].J = (double*) realloc(cbcR[0].J,
     (cbcM*cbcKn*sizeof(double)));
5204 cbcR[0].EJ = (double*) realloc(cbcR[0].EJ,
     (cbcM*cbcKn*sizeof(double)));
5205 cbcR[0].CJ = (double*) realloc(cbcR[0].CJ,
     (cbcM*cbcKn*sizeof(double)));
5206 for(cbcb=cbcKc;cbcb<cbcKn;cbcb++){
5207   cbcK=cbcb;
5208   for(cbcc=0;cbcc<cbcM;cbcc++){
5209     cbcR[0].J[cbcb*cbcM+cbcc]=cbcK;
5210     cbcR[0].EJ[cbcb*cbcM+cbcc]=cbcTeVib+cbcBv*(cbcK*(cbcK+1))-\
5211       cbcDv*cbcK*cbcK*(cbcK+1)*(cbcK+1);
5212     cbcR[0].CJ[cbcb*cbcM+cbcc]=0;
5213 if(DEBUG>cbcdebugflag){
5214 fprintf(DBG,\
5215 "cbcb=%d cbcR[0].J[cbcb]=%f cbcR[0].EJ[cbcb]=%f
     cbcR[0].CJ[cbcb]=%f\n",\
5216     cbcb,cbcK,cbcR[0].EJ[cbcb],cbcR[0].CJ[cbcb]);
```

```
5217 fflush(DBG);
5218 }
5219     }
5220   }
5221 cbcR[0].kc=cbcKn;
5222 return;
5223 }

     /*************** do_simulation *****************/

     /* This function loops through all the line emissions (whether cal-
        culated in a transition function or read from a file) and adds
        their contributions to appropriate places in the simulation. The
        distinction between an instrument that scans or jumps from point
        to point is made here. */
5224 void do_simulation(){
     /* counting variables */
5225 int da=0,db=0,dc=0,dd=0,dn=0,dbin=0,dnn=0,dsw=0;
     /* flags */
5226 int ddoflag=0,ddoneflag=0;
     /* variables for assigning intensities */
5227 double
    dposn=0,dcslope=0,dnslope=0,ddumposn=0,ddumnint=0,ddumcint=0;
5228 double ddend=0,dwidth=0,dintmax=0;
     /* temporary simulations for writing separate atomic, molecular,
        etc, simulations to files */
5229 sim dmnsim,dmcsim,dasim,dosim,dtotsim;
     /* variables about files to create */
5230 char dnfile[2000],dcfile[2000];
5231 FILE *DNFILE,*DCFILE;

     /* Make the directory for storing simulation output */
5232 sprintf(dnfile,"mkdir %s_simulations",PREF);
5233 system(dnfile);
     /* get number of bins and allocate memory for simulations */
5234 dn=(int)(ceil((MAXV-MINV)/PSTEP));
5235 if(ddebugflag<DEBUG){
5236 fprintf(DBG,"Top of do_simulation. dn=%d\n",dn);
5237 fflush(DBG);
5238 }
5239 dtotsim.n=dmnsim.n=dmcsim.n=dasim.n=dosim.n=dn;
5240 dmnsim.x=(double*)calloc(dn,sizeof(double));
5241 dmnsim.y=(double*)calloc(dn,sizeof(double));
5242 dmcsim.x=(double*)calloc(dn,sizeof(double));
5243 dmcsim.y=(double*)calloc(dn,sizeof(double));
5244 dasim.x=(double*)calloc(dn,sizeof(double));
5245 dasim.y=(double*)calloc(dn,sizeof(double));
5246 dosim.x=(double*)calloc(dn,sizeof(double));
5247 dosim.y=(double*)calloc(dn,sizeof(double));
5248 dtotsim.x=(double*)calloc(dn,sizeof(double));
5249 dtotsim.y=(double*)calloc(dn,sizeof(double));

     /****************** Diatomic emissions *****************/
5250 for(da=0;da<NUMMOL;da++){ /* Loop over all molecules */
5251   for(db=0;db<MOL[da].trans;db++){ /* then loop over transitions */
```

```
5252 if(ddebugflag<DEBUG){
5253 fprintf(DBG,"MOL[da].t[db].Nhi=%s, MOL[%d].t[%d].Nlo=%s",\
5254     MOL[da].t[db].Nhi,da,db,MOL[da].t[db].Nlo);
5255 fflush(DBG);
5256 }
5257     for(dc=0;dc<MOL[da].t[db].nS;dc++){ /* loop through simsets */
5258 if(ddebugflag<DEBUG){
5259 fprintf(DBG,"da=%d, db=%d, dc=%d, MOL[da].t[db].fS[dc]=%d, ",\
5260     da,db,dc,MOL[da].t[db].fS[dc]);
5261 fflush(DBG);
5262 }
      /* if inclusion flag says to include this simset */
5263     if(MOL[da].t[db].fS[dc]==0){
      /* loop through transitions in simset */
5264 if(ddebugflag<DEBUG){
5265 fprintf(DBG,"MOL[%d].t[%d].S[%d].n=%d, ",\
5266     da,db,dc,MOL[da].t[db].S[dc].n);
5267 fflush(DBG);
5268 }
5269       for(dd=0;dd<MOL[da].t[db].S[dc].n;dd++){
      /* if the current frequency is not equal to zero */
5270       if(MOL[da].t[db].S[dc].f[dd]!=0){
      /* see if line falls within requested simulation min/max */
5271 if(UNITTYPE==0) dposn=1e7/MOL[da].t[db].S[dc].f[dd];
5272 else dposn=MOL[da].t[db].S[dc].f[dd];
5273 ddoflag=0;
5274 if(dposn<(MINV-RESN)) ddoflag=1;
5275 if(dposn>(MAXV+RESN)) ddoflag=1;
5276 if(ddebugflag<DEBUG){
5277 fprintf(DBG,"UNITTYPE=%d, dposn=%12.6e,
     ddoflag=%d\n",UNITTYPE,dposn,ddoflag);
5278 fprintf(DBG,"MOL[%d].t[%d].S[%d].f[%d]=%12.6e\n",da,db,dc,dd,\
5279     MOL[da].t[db].S[dc].f[dd]);
5280 fflush(DBG);
5281 }
5282 if(ddoflag==0){
      /* find bin where line originates, also get native and cascade
        slopes */
5283   dbin=(int)(floor((dposn-MINV)/PSTEP));
5284   dcslope=MOL[da].pop*MOL[da].t[db].S[dc].ci[dd]/RESN;
5285   dnslope=MOL[da].pop*MOL[da].t[db].S[dc].ni[dd]/RESN;
5286 if(ddebugflag<DEBUG){
5287 fprintf(DBG,"RESN=%12.6e,
     MOL[%d].pop=%12.6e\n",RESN,da,MOL[da].pop);
5288 fprintf(DBG,"MOL[%d].t[%d].S[%d].ci[%d]=%12.6e\n",da,db,dc,dd,\
5289     MOL[da].t[db].S[dc].ci[dd]);
5290 fprintf(DBG,"MOL[%d].t[%d].S[%d].ni[%d]=%12.6e\n",da,db,dc,dd,\
5291     MOL[da].t[db].S[dc].ni[dd]);
5292 fprintf(DBG,"dposn=%12.6e, MINV=%12.6e,
     PSTEP=%12.6e\n",dposn,MINV,PSTEP);
5293 fflush(DBG);
5294 }
      /* increment through them and calculate the intensities for each
        bin -- integrate if SCAN and point if JUMP */
```

```
5295   ddoneflag=1;
5296   ddumposn=dposn;
5297   ddumnint=MOL[da].pop*MOL[da].t[db].S[dc].ni[dd];
5298   ddumcint=MOL[da].pop*MOL[da].t[db].S[dc].ci[dd];
5299   dnn=0;
5300   if(dbin<0) ddoneflag=0;
5301 if(ddebugflag<DEBUG){
5302 fprintf(DBG,"dbin=%d, dcslope=%12.6e, dnslope=%12.6e,
     ddumnint=%12.6e, ",\
5303     dbin,dcslope,dnslope,ddumnint);
5304 fprintf(DBG, "ddumcint = %12.6e, ddoneflag = %d\n", ddumcint,
     ddoneflag);
5305 fflush(DBG);
5306 }
5307   while(ddoneflag==1){ /* fill bins at/below dbin */
5308     dsw=0;
5309     ddend=MINV+(dbin-dnn)*PSTEP;
5310     if(ddumposn==dposn) dsw+=1;
5311     if(ddend<(dposn-RESN)) dsw+=2;
5312     if(SCANTYPE==1) dsw+=4;
5313     if((dbin-dnn)>(dn-1)) dsw=8;
5314 if(ddebugflag<DEBUG){
5315 fprintf(DBG,"AT/BELOW ddumposn=%12.6e, ddend=%12.6e,
     ",ddumposn,ddend);
5316 fprintf(DBG,"SCANTYPE=%d, dsw=%d, ",SCANTYPE,dsw);
5317 fflush(DBG);
5318 }
5319     switch (dsw){
      /* the following for equipment that scans between "points" */
5320       case 0:
5321         dmnsim.y[dbin-dnn]+=\
5322           (ddumnint-0.5*dnslope*PSTEP)*PSTEP;
5323         dmcsim.y[dbin-dnn]+=\
5324           (ddumcint-0.5*dcslope*PSTEP)*PSTEP;
5325         ddumnint-=dnslope*PSTEP;
5326         ddumcint-=dcslope*PSTEP;
5327         ddumposn-=PSTEP;
5328         if(dnn==dbin) ddoneflag=0;
5329         dnn++;
5330 if(ddebugflag<DEBUG){
5331 fprintf(DBG,"Case 0. dmnsim.y[%d]=%12.6e,
     dmcsim.y[%d]=%12.6e\n",\
5332   (dbin-dnn+1),dmnsim.y[dbin-dnn+1],(dbin-dnn+1),dmcsim.y[dbin-
     dnn+1]);
5333 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5334     ddumnint,ddumcint,ddumposn,dnn);
5335 fflush(DBG);
5336 }
5337         break;
5338       case 1:
5339         dwidth=ddumposn-ddend;
5340         dmnsim.y[dbin-dnn]+=\
5341           (ddumnint-0.5*dnslope*dwidth)*dwidth;
```

```
5342          dmcsim.y[dbin-dnn]+=\
5343            (ddumcint-0.5*dcslope*dwidth)*dwidth;
5344          ddumnint-=dnslope*dwidth;
5345          ddumcint-=dcslope*dwidth;
5346          ddumposn=ddend;
5347          if(dnn==dbin) ddoneflag=0;
5348          dnn++;
5349 if(ddebugflag<DEBUG){
5350 fprintf(DBG,"Case 1. dwidth=%12.6e, dmnsim.y[%d]=%12.6e,
     ",dwidth,\
5351   (dbin-dnn+1),dmnsim.y[dbin-dnn+1]);
5352 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin-dnn+1),dmcsim.y[dbin-
     dnn+1]);
5353 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5354     ddumnint,ddumcint,ddumposn,dnn);
5355 fflush(DBG);
5356 }
5357          break;
5358        case 2:
5359          if(dnslope!=0) dwidth=ddumnint/dnslope;
5360          else{
5361            if(dcslope!=0) dwidth=ddumcint/dcslope;
5362            else dwidth=0;
5363            }
5364          dmnsim.y[dbin-dnn]+=ddumnint*0.5*dwidth;
5365          dmcsim.y[dbin-dnn]+=ddumcint*0.5*dwidth;
5366          ddoneflag=0;
5367 if(ddebugflag<DEBUG){
5368 fprintf(DBG,"Case 2. dwidth=%12.6e, dmnsim.y[%d]=%12.6e,
     ",dwidth,\
5369   (dbin-dnn),dmnsim.y[dbin-dnn]);
5370 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin-dnn),dmcsim.y[dbin-
     dnn]);
5371 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5372     ddumnint,ddumcint,ddumposn,dnn);
5373 fflush(DBG);
5374 }
5375          break;
5376        case 3:
5377          dmnsim.y[dbin-dnn]+=ddumnint*0.5*PSTEP;
5378          dmcsim.y[dbin-dnn]+=ddumcint*0.5*PSTEP;
5379          ddoneflag=0;
5380 if(ddebugflag<DEBUG){
5381 fprintf(DBG,"Case 3. dmnsim.y[%d]=%12.6e, ",(dbin-
     dnn),dmnsim.y[dbin-dnn]);
5382 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin-dnn),dmcsim.y[dbin-
     dnn]);
5383 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5384     ddumnint,ddumcint,ddumposn,dnn);
5385 fflush(DBG);
5386 }
```

```
5387          break;
      /* the following for equipment that jumps from point to point */
5388        case 4:
5389          ddumnint-=dnslope*PSTEP;
5390          ddumcint-=dcslope*PSTEP;
5391          dmnsim.y[dbin-dnn]+=ddumnint;
5392          dmcsim.y[dbin-dnn]+=ddumcint;
5393          ddumposn-=PSTEP;
5394          if(dnn==dbin) ddoneflag=0;
5395          dnn++;
5396 if(ddebugflag<DEBUG){
5397 fprintf(DBG,"Case 4. dmnsim.y[%d]=%12.6e,
     dmcsim.y[%d]=%12.6e\n",\
5398   (dbin-dnn+1),dmnsim.y[dbin-dnn+1],(dbin-dnn+1),dmcsim.y[dbin-
     dnn+1]);
5399 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5400     ddumnint,ddumcint,ddumposn,dnn);
5401 fflush(DBG);
5402 }
5403          break;
5404        case 5:
5405          dwidth=ddumposn-ddend;
5406          ddumnint-=dnslope*dwidth;
5407          ddumcint-=dcslope*dwidth;
5408          dmnsim.y[dbin-dnn]+=ddumnint;
5409          dmcsim.y[dbin-dnn]+=ddumcint;
5410          ddumposn=ddend;
5411          if(dnn==dbin) ddoneflag=0;
5412          dnn++;
5413 if(ddebugflag<DEBUG){
5414 fprintf(DBG,"Case 5. dwidth=%12.6e, dmnsim.y[%d]=%12.6e,
     ",dwidth,\
5415   (dbin-dnn+1),dmnsim.y[dbin-dnn+1]);
5416 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin-dnn+1),dmcsim.y[dbin-
     dnn+1]);
5417 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5418     ddumnint,ddumcint,ddumposn,dnn);
5419 fflush(DBG);
5420 }
5421          break;
5422        case 6:
5423          ddoneflag=0;
5424 if(ddebugflag<DEBUG){
5425 fprintf(DBG,"Case 6\n");
5426 fflush(DBG);
5427 }
5428          break;
5429        case 7:
5430          dmnsim.y[dbin-dnn]+=ddumnint;
5431          dmcsim.y[dbin-dnn]+=ddumcint;
5432          ddoneflag=0;
5433 if(ddebugflag<DEBUG){
```

```
5434 fprintf(DBG,"Case 7. dmnsim.y[%d]=%12.6e, ",(dbin-
     dnn),dmnsim.y[dbin-dnn]);
5435 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin-dnn),dmcsim.y[dbin-
     dnn]);
5436 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5437     ddumnint,ddumcint,ddumposn,dnn);
5438 fflush(DBG);
5439 }
5440         break;
      /* the following for emissions that provide only "spillover" */
5441       case 8:
5442         if(ddumposn==dposn){
5443           dwidth=ddumposn-ddend;
5444           ddumnint-=dnslope*dwidth;
5445           ddumcint-=dcslope*dwidth;
5446           ddumposn=ddend;
5447           if(dnn==dbin) ddoneflag=0;
5448           dnn++;
5449           }
5450         else{
5451           if(ddend<(dposn-RESN)){
5452             printf(\
5453         "Fix switch case in do_sim at 1a.\n");
5454             exit(1);
5455             }
5456           else{
5457             ddumnint-=dnslope*PSTEP;
5458             ddumcint-=dcslope*PSTEP;
5459             ddumposn-=PSTEP;
5460             if(dnn==dbin) ddoneflag=0;
5461             dnn++;
5462             }
5463           }
5464 if(ddebugflag<DEBUG){
5465 fprintf(DBG,"Case 8. dwidth=%12.6e, dbin=%d",dwidth,dbin);
5466 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5467     ddumnint,ddumcint,ddumposn,dnn);
5468 fflush(DBG);
5469 }
5470         break;
5471       default:
5472         printf("Go fix do_sim.\n");
5473         exit(1);
5474       } /* close switch-case */
5475     } /* close while for bins below the transition position */
5476   ddoneflag=1;
5477   ddumposn=dposn;
5478   ddumnint=MOL[da].pop*MOL[da].t[db].S[dc].ni[dd];
5479   ddumcint=MOL[da].pop*MOL[da].t[db].S[dc].ci[dd];
5480   dnn=0;
5481   if(dbin>dn-1) ddoneflag=0;
5482 if(ddebugflag<DEBUG){
```

```
5483 fprintf(DBG,"dbin=%d, dcslope=%12.6e, dnslope=%12.6e,
     ddumnint=%12.6e, ",\
5484     dbin,dcslope,dnslope,ddumnint);
5485 fprintf(DBG, "ddumcint = %12.6e, ddoneflag = %d\n", ddumcint,
     ddoneflag);
5486 fflush(DBG);
5487 }
5488   while(ddoneflag==1){ /* fill bins at/above dbin */
5489     dsw=0;
5490     ddend=MINV+(dbin+dnn+1)*PSTEP;
5491     if(dposn==ddumposn) dsw+=1;
5492     if(ddend>(dposn+RESN)) dsw+=2;
5493     if(SCANTYPE==1) dsw+=4;
5494     if((dbin+dnn)<0) dsw=8;
5495 if(ddebugflag<DEBUG){
5496 fprintf(DBG,"AT/ABOVE ddumposn=%12.6e, ddend=%12.6e,
     ",ddumposn,ddend);
5497 fprintf(DBG,"SCANTYPE=%d, dsw=%d, ",SCANTYPE,dsw);
5498 fflush(DBG);
5499 }
5500     switch (dsw){
      /* the following for equipment that scans between "points" */
5501       case 0:
5502         dmnsim.y[dbin+dnn]+=\
5503           (ddumnint-0.5*dnslope*PSTEP)*PSTEP;
5504         dmcsim.y[dbin+dnn]+=\
5505           (ddumcint-0.5*dcslope*PSTEP)*PSTEP;
5506         ddumnint-=dnslope*PSTEP;
5507         ddumcint-=dcslope*PSTEP;
5508         ddumposn+=PSTEP;
5509         dnn++;
5510         if((dnn+dbin)==dn) ddoneflag=0;
5511 if(ddebugflag<DEBUG){
5512 fprintf(DBG,"Case 0. dmnsim.y[%d]=%12.6e,
     dmcsim.y[%d]=%12.6e\n",\
5513   (dbin+dnn-1),dmnsim.y[dbin+dnn-1],(dbin+dnn-
     1),dmcsim.y[dbin+dnn-1]);
5514 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5515     ddumnint,ddumcint,ddumposn,dnn);
5516 fflush(DBG);
5517 }
5518         break;
5519       case 1:
5520         dwidth=ddend-ddumposn;
5521         dmnsim.y[dbin+dnn]+=\
5522           (ddumnint-0.5*dnslope*dwidth)*dwidth;
5523         dmcsim.y[dbin+dnn]+=\
5524           (ddumcint-0.5*dcslope*dwidth)*dwidth;
5525         ddumnint-=dnslope*dwidth;
5526         ddumcint-=dcslope*dwidth;
5527         ddumposn=ddend;
5528         dnn++;
5529         if((dnn+dbin)==dn) ddoneflag=0;
```

```
5530 if(ddebugflag<DEBUG){
5531 fprintf(DBG,"Case 1. dwidth=%12.6e, dmnsim.y[%d]=%12.6e,
     ",dwidth,\
5532   (dbin+dnn-1),dmnsim.y[dbin+dnn-1]);
5533 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin+dnn-
     1),dmcsim.y[dbin+dnn-1]);
5534 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5535     ddumnint,ddumcint,ddumposn,dnn);
5536 fflush(DBG);
5537 }
5538         break;
5539       case 2:
5540         if(dnslope!=0) dwidth=ddumnint/dnslope;
5541         else{
5542           if(dcslope!=0) dwidth=ddumcint/dcslope;
5543           else dwidth=0;
5544           }
5545         dmnsim.y[dbin+dnn]+=ddumnint*0.5*dwidth;
5546         dmcsim.y[dbin+dnn]+=ddumcint*0.5*dwidth;
5547         ddoneflag=0;
5548 if(ddebugflag<DEBUG){
5549 fprintf(DBG,"Case 2. dwidth=%12.6e, dmnsim.y[%d] = %12.6e, ",
     dwidth,\
5550   (dbin+dnn),dmnsim.y[dbin+dnn]);
5551 fprintf(DBG, "dmcsim.y[%d] = %12.6e\n", (dbin+dnn), dmc-
     sim.y[dbin+dnn]);
5552 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5553     ddumnint,ddumcint,ddumposn,dnn);
5554 fflush(DBG);
5555 }
5556         break;
5557       case 3:
5558         dmnsim.y[dbin+dnn]+=ddumnint*0.5*PSTEP;
5559         dmcsim.y[dbin+dnn]+=ddumcint*0.5*PSTEP;
5560         ddoneflag=0;
5561 if(ddebugflag<DEBUG){
5562 fprintf(DBG,"Case 3. dmnsim.y[%d]=%12.6e, ", (dbin+dnn), dmn-
     sim.y[dbin+dnn]);
5563 fprintf(DBG,"dmcsim.y[%d] = %12.6e\n", (dbin+dnn), dmc-
     sim.y[dbin+dnn]);
5564 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5565     ddumnint,ddumcint,ddumposn,dnn);
5566 fflush(DBG);
5567 }
5568         break;
     /* the following for equipment that jumps from point to point */
5569       case 4:
5570         ddumnint-=dnslope*PSTEP;
5571         ddumcint-=dcslope*PSTEP;
5572         dmnsim.y[dbin+dnn]+=ddumnint;
5573         dmcsim.y[dbin+dnn]+=ddumcint;
```

```
5574          ddumposn+=PSTEP;
5575          dnn++;
5576          if((dnn+dbin)==dn) ddoneflag=0;
5577 if(ddebugflag<DEBUG){
5578 fprintf(DBG,"Case 4. dmnsim.y[%d]=%12.6e, dmcsim.y[%d] =
     %12.6e\n",\
5579   (dbin+dnn-1), dmnsim.y[dbin+dnn-1], (dbin+dnn-1),
     dmcsim.y[dbin+dnn-1]);
5580 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5581     ddumnint,ddumcint,ddumposn,dnn);
5582 fflush(DBG);
5583 }
5584          break;
5585        case 5:
5586          dwidth=ddend-ddumposn;
5587          ddumnint-=dnslope*dwidth;
5588          ddumcint-=dcslope*dwidth;
5589          dmnsim.y[dbin+dnn]+=ddumnint;
5590          dmcsim.y[dbin+dnn]+=ddumcint;
5591          ddumposn=ddend;
5592          dnn++;
5593          if((dnn+dbin)==dn) ddoneflag=0;
5594 if(ddebugflag<DEBUG){
5595 fprintf(DBG,"Case 5. dwidth=%12.6e, dmnsim.y[%d]=%12.6e,
     ",dwidth,\
5596   (dbin+dnn-1),dmnsim.y[dbin+dnn-1]);
5597 fprintf(DBG,"dmcsim.y[%d]=%12.6e\n",(dbin+dnn-
     1),dmcsim.y[dbin+dnn-1]);
5598 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5599     ddumnint,ddumcint,ddumposn,dnn);
5600 fflush(DBG);
5601 }
5602          break;
5603        case 6:
5604          ddoneflag=0;
5605 if(ddebugflag<DEBUG){
5606 fprintf(DBG,"Case 6\n");
5607 fflush(DBG);
5608 }
5609          break;
5610        case 7:
5611          dmnsim.y[dbin+dnn]+=ddumnint;
5612          dmcsim.y[dbin+dnn]+=ddumcint;
5613          ddoneflag=0;
5614 if(ddebugflag<DEBUG){
5615 fprintf(DBG,"Case 7. dmnsim.y[%d] = %12.6e, ", (dbin+dnn), dmn-
     sim.y[dbin+dnn]);
5616 fprintf(DBG,"dmcsim.y[%d] = %12.6e\n", (dbin+dnn), dmc-
     sim.y[dbin+dnn]);
5617 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5618     ddumnint,ddumcint,ddumposn,dnn);
```

```
5619 fflush(DBG);
5620 }
5621         break;
     /* the following for emissions that provide only "spillover" */
5622       case 8:
5623         if(ddumposn==dposn){
5624           dwidth=ddend-ddumposn;
5625           ddumnint-=dnslope*dwidth;
5626           ddumcint-=dcslope*dwidth;
5627           ddumposn=ddend;
5628           dnn++;
5629           if((dnn+dbin)==dn) ddoneflag=0;
5630           }
5631         else{
5632           if(ddend>(dposn+RESN)){
5633             printf(\
5634         "Fix switch case in do_sim at 1b.\n");
5635             exit(1);
5636             }
5637           else{
5638             ddumnint-=dnslope*PSTEP;
5639             ddumcint-=dcslope*PSTEP;
5640             ddumposn+=PSTEP;
5641             dnn++;
5642             if((dnn+dbin)==dn) ddoneflag=0;
5643             }
5644           }
5645 if(ddebugflag<DEBUG){
5646 fprintf(DBG,"Case 8. dwidth=%12.6e, dbin=%d, ",dwidth,dbin);
5647 fprintf(DBG,"ddumnint=%12.6e, ddumcint=%12.6e, ddumposn=%12.6e,
     dnn=%d\n",\
5648     ddumnint,ddumcint,ddumposn,dnn);
5649 fflush(DBG);
5650 }
5651         break;
5652       default:
5653         printf("Go fix do_sim.\n");
5654         exit(1);
5655       } /* close switch-case */
5656     } /* close fill bins above the transition position */
5657   } /* close if frequency in range condition */
5658         } /* close if non-zero frequency condition */
5659         } /* close simset-transitions loop */
5660       } /* close if include condition */
5661       } /* close loop over simulations */
5662     } /* close loop over transitions */
5663   } /* close loop over molecules */

     /* start loop over previous simulation and write to file */
5664 sprintf(dnfile,"%s_simulations/Mol_native_pop",PREF);
5665 DNFILE=fopen(dnfile,"w");
5666 if(DNFILE==NULL){
5667   printf("Error opening DNFILE for native pop. Exiting.\n");
5668   exit(1);
```

```
5669   }
5670 sprintf(dcfile,"%s_simulations/Mol_cascade_pop",PREF);
5671 DCFILE=fopen(dcfile,"w");
5672 if(DCFILE==NULL){
5673   printf("Error opening DCFILE for cascade pop. Exiting.\n");
5674   exit(1);
5675   }
5676 fprintf(DNFILE,"# File created by %s. \n",PROGRAM_NAME);
5677 fprintf(DNFILE,"# Contains simulated rovibronic emissions based on
     \n");
5678 fprintf(DNFILE,"# molecular populations specified by the user
     (only).\n");
5679 fprintf(DNFILE,"# See file Mol_cascade_pop for emissions due to cas-
     cade.\n");
5680 fprintf(DNFILE,"# See file %s_Sim_All for the whole simula-
     tion.\n#\n",PREF);
5681 fprintf(DCFILE,"# File created by %s. \n",PROGRAM_NAME);
5682 fprintf(DCFILE,"# Contains simulated rovibronic emissions based on
     \n");
5683 fprintf(DCFILE,"# cascade of molecular states (only).\n");
5684 fprintf(DCFILE,"# See file Mol_native_pop for emissions from molecu-
     lar.\n");
5685 fprintf(DCFILE,"# populations specified by the user.\n");
5686 fprintf(DCFILE,"# See file %s_Sim_All for the whole simula-
     tion.\n#\n",PREF);
5687 for(da=0;da<dn;da++){
5688   dmnsim.x[da]=MINV+0.5*PSTEP+da*PSTEP;
5689   dmcsim.x[da]=MINV+0.5*PSTEP+da*PSTEP;
5690
     fprintf(DNFILE,"%18.12e\t%18.12e\n",dmnsim.x[da],dmnsim.y[da]);
5691
     fprintf(DCFILE,"%18.12e\t%18.12e\n",dmcsim.x[da],dmcsim.y[da]);
5692   }
5693 fclose(DNFILE);
5694 fclose(DCFILE);

      /**************** Atomic emissions ******************/
5695 for(da=0;da<NUMAT;da++){ /* loop over sets of atomic info */
5696 for(db=0;db<AT[da].n;db++){ /* loop through transitions in this set
       */
5697 if(AT[da].x[db]!=0){ /* if the current frequency is not equal to
       zero */
      /* see if line falls within requested simulation min/max */
5698 if(UNITTYPE==0) dposn=AT[da].x[db]/10;
5699 else dposn=1e8/AT[da].x[db];
5700 ddoflag=0;
5701 if(dposn<(MINV-RESN)) ddoflag=1;
5702 if(dposn>(MAXV+RESN)) ddoflag=1;
5703 if(ddoflag==0){
      /* find bin where line originates, also get native and cascade
       slopes */
5704   dbin=(int)(floor((dposn-MINV)/PSTEP));
5705   dnslope=AT[da].p*AT[da].A[db]*AT[da].pop[db]/RESN;
```

```
     /* increment through them and calculate the intensities for each
        bin -- integrate if SCAN and point if JUMP */
5706   ddoneflag=1;
5707   ddumposn=dposn;
5708   ddumnint=AT[da].p*AT[da].A[db]*AT[da].pop[db];
5709   dnn=0;
5710 if(ddebugflag<DEBUG){
5711 fprintf(DBG,"dbin=%d, dposn=%12.6e, dnslope=%12.6e,
     ddumnint=%12.6e\n",\
5712     dbin,dposn,dnslope,ddumnint);
5713 fprintf(DBG,"AT[%d].p=%12.6e, AT[%d].A[%d]=%12.6e,
     AT[%d].pop[%d]=%12.6e\n",\
5714   da,AT[da].p,da,db,AT[da].A[db],da,db,AT[da].pop[db]);
5715 fprintf(DBG,"RESN=%12.6e, MINV=%12.6e,
     PSTEP=%12.6e\n",RESN,MINV,PSTEP);
5716 fflush(DBG);
5717 }
5718   if(dbin<0) ddoneflag=0;
5719   while(ddoneflag==1){ /* fill bins at/below dbin */
5720     dsw=0;
5721     ddend=MINV+(dbin-dnn)*PSTEP;
5722     if(dposn==ddumposn) dsw+=1;
5723     if(ddend<(dposn-RESN)) dsw+=2;
5724     if(SCANTYPE==1) dsw+=4;
5725     if((dbin-dnn)>(dn-1)) dsw=8;
5726     switch (dsw){
     /* the following for equipment that scans between "points" */
5727       case 0:
5728         dasim.y[dbin-dnn]+=\
5729           (ddumnint-0.5*dnslope*PSTEP)*PSTEP;
5730         ddumnint-=dnslope*PSTEP;
5731         ddumposn-=PSTEP;
5732         if(dnn==dbin) ddoneflag=0;
5733         dnn++;
5734 if(ddebugflag<DEBUG){
5735 fprintf(DBG,"AT/BELOW Case 0. dasim.y[%d]=%12.6e, ",\
5736     (dbin-dnn+1),dasim.y[dbin-dnn+1]);
5737 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5738     ddumnint,ddumposn,dnn);
5739 fflush(DBG);
5740 }
5741         break;
5742       case 1:
5743         dwidth=ddumposn-ddend;
5744         dasim.y[dbin-dnn]+=\
5745           (ddumnint-0.5*dnslope*dwidth)*dwidth;
5746         ddumnint-=dnslope*dwidth;
5747         ddumposn=ddend;
5748         if(dnn==dbin) ddoneflag=0;
5749         dnn++;
5750 if(ddebugflag<DEBUG){
5751 fprintf(DBG,"AT/BELOW Case 1. dwidth=%12.6e, dasim.y[%d]=%12.6e,
     ",dwidth,\
5752   (dbin-dnn+1),dasim.y[dbin-dnn+1]);
```

```
5753 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5754     ddumnint,ddumposn,dnn);
5755 fflush(DBG);
5756 }
5757         break;
5758       case 2:
5759         if(dnslope!=0) dwidth=ddumnint/dnslope;
5760         else{
5761           printf("dammit (below)\n");
5762           exit(1);
5763           }
5764         dasim.y[dbin-dnn]+=ddumnint*0.5*dwidth;
5765         ddoneflag=0;
5766 if(ddebugflag<DEBUG){
5767 fprintf(DBG,"AT/BELOW Case 2. dwidth=%12.6e, dasim.y[%d]=%12.6e,
    ",dwidth,\
5768   (dbin-dnn),dasim.y[dbin-dnn]);
5769 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5770     ddumnint,ddumposn,dnn);
5771 fflush(DBG);
5772 }
5773         break;
5774       case 3:
5775         dasim.y[dbin-dnn]+=ddumnint*0.5*PSTEP;
5776         ddoneflag=0;
5777 if(ddebugflag<DEBUG){
5778 fprintf(DBG,"AT/BELOW Case 3. dasim.y[%d]=%12.6e, ",\
5779     (dbin-dnn),dasim.y[dbin-dnn]);
5780 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5781     ddumnint,ddumposn,dnn);
5782 fflush(DBG);
5783 }
5784         break;
     /* the following for equipment that jumps from point to point */
5785       case 4:
5786         ddumnint-=dnslope*PSTEP;
5787         dasim.y[dbin-dnn]+=ddumnint;
5788         ddumposn-=PSTEP;
5789         if(dnn==dbin) ddoneflag=0;
5790         dnn++;
5791 if(ddebugflag<DEBUG){
5792 fprintf(DBG,"AT/BELOW Case 4. dasim.y[%d]=%12.6e, ",\
5793     (dbin-dnn+1),dmnsim.y[dbin-dnn+1]);
5794 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5795     ddumnint,ddumposn,dnn);
5796 fflush(DBG);
5797 }
5798         break;
5799       case 5:
5800         dwidth=ddumposn-ddend;
5801         ddumnint-=dnslope*dwidth;
5802         dasim.y[dbin-dnn]+=ddumnint;
5803         ddumposn=ddend;
5804         if(dnn==dbin) ddoneflag=0;
```

```
5805          dnn++;
5806 if(ddebugflag<DEBUG){
5807 fprintf(DBG,"AT/BELOW Case 5. dwidth=%12.6e, dasim.y[%d]=%12.6e,
     ",dwidth,\
5808   (dbin-dnn+1),dasim.y[dbin-dnn+1]);
5809 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5810      ddumnint,ddumposn,dnn);
5811 fflush(DBG);
5812 }
5813          break;
5814        case 6:
5815          ddoneflag=0;
5816 if(ddebugflag<DEBUG){
5817 fprintf(DBG,"AT/BELOW Case 6\n");
5818 fflush(DBG);
5819 }
5820          break;
5821        case 7:
5822          dasim.y[dbin-dnn]+=ddumnint;
5823          ddoneflag=0;
5824 if(ddebugflag<DEBUG){
5825 fprintf(DBG,"AT/BELOW Case 7. dasim.y[%d]=%12.6e, ",\
5826      (dbin-dnn),dasim.y[dbin-dnn]);
5827 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5828      ddumnint,ddumposn,dnn);
5829 fflush(DBG);
5830 }
5831          break;
      /* the following for emissions that provide only "spillover" */
5832        case 8:
5833          if(ddumposn==dposn){
5834            dwidth=ddumposn-ddend;
5835            ddumnint-=dnslope*dwidth;
5836            ddumcint-=dcslope*dwidth;
5837            ddumposn=ddend;
5838            if(dnn==dbin) ddoneflag=0;
5839            dnn++;
5840            }
5841          else{
5842            if(ddend<(dposn-RESN)){
5843              printf(\
5844          "Fix switch case in do_sim at 1a.\n");
5845              exit(1);
5846              }
5847            else{
5848              ddumnint-=dnslope*PSTEP;
5849              ddumcint-=dcslope*PSTEP;
5850              ddumposn-=PSTEP;
5851              if(dnn==dbin) ddoneflag=0;
5852              dnn++;
5853              }
5854            }
5855 if(ddebugflag<DEBUG){
```

```
5856 fprintf(DBG,"AT/BELOW Case 8. dwidth=%12.6e, dbin=%d
     ",dwidth,dbin);
5857 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5858     ddumnint,ddumposn,dnn);
5859 fflush(DBG);
5860 }
5861         break;
5862       default:
5863         printf("Go fix do_sim.\n");
5864         exit(1);
5865       } /* close switch-case */
5866     } /* close while for bins below the transition position */
5867   ddoneflag=1;
5868   ddumposn=dposn;
5869   ddumnint=AT[da].p*AT[da].A[db]*AT[da].pop[db];
5870   dnn=0;
5871   if(dbin>dn-1) ddoneflag=0;
5872   while(ddoneflag==1){ /* fill bins at/above dbin */
5873     dsw=0;
5874     ddend=MINV+(dbin+dnn+1)*PSTEP;
5875     if(dposn==ddumposn) dsw+=1;
5876     if(ddend>(dposn+RESN)) dsw+=2;
5877     if(SCANTYPE==1) dsw+=4;
5878     if((dbin+dnn)<0) dsw=8;
5879     switch (dsw){
      /* the following for equipment that scans between "points" */
5880       case 0:
5881         dasim.y[dbin+dnn]+=\
5882           (ddumnint-0.5*dnslope*PSTEP)*PSTEP;
5883         ddumnint-=dnslope*PSTEP;
5884         ddumposn+=PSTEP;
5885         dnn++;
5886         if((dnn+dbin)==dn) ddoneflag=0;
5887 if(ddebugflag<DEBUG){
5888 fprintf(DBG,"AT/ABOVE Case 0. dasim.y[%d]=%12.6e ",\
5889     (dbin+dnn-1),dasim.y[dbin+dnn-1]);
5890 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5891     ddumnint,ddumposn,dnn);
5892 fflush(DBG);
5893 }
5894         break;
5895       case 1:
5896         dwidth=ddend-ddumposn;
5897         dasim.y[dbin+dnn]+=\
5898           (ddumnint-0.5*dnslope*dwidth)*dwidth;
5899         ddumnint-=dnslope*dwidth;
5900         ddumposn=ddend;
5901         dnn++;
5902         if((dnn+dbin)==dn) ddoneflag=0;
5903 if(ddebugflag<DEBUG){
5904 fprintf(DBG,"AT/ABOVE Case 1. dwidth=%12.6e, dasim.y[%d]=%12.6e,
     ",dwidth,\
5905   (dbin+dnn-1),dasim.y[dbin+dnn-1]);
5906 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
```

```
5907     ddumnint,ddumposn,dnn);
5908 fflush(DBG);
5909 }
5910         break;
5911       case 2:
5912         if(dnslope!=0) dwidth=ddumnint/dnslope;
5913         else{
5914           printf("dammit...\n");
5915           exit(1);
5916           }
5917         dasim.y[dbin+dnn]+=ddumnint*0.5*dwidth;
5918         ddoneflag=0;
5919 if(ddebugflag<DEBUG){
5920 fprintf(DBG,"AT/ABOVE Case 2. dwidth=%12.6e, dasim.y[%d]=%12.6e,
    ",dwidth,\
5921   (dbin+dnn),dasim.y[dbin+dnn]);
5922 fprintf(DBG,"ddumnint=%12.6e, dnslope=%12.6e, ddumposn=%12.6e,
    dnn=%d\n",\
5923     ddumnint,dnslope,ddumposn,dnn);
5924 fflush(DBG);
5925 }
5926         break;
5927       case 3:
5928         dasim.y[dbin+dnn]+=ddumnint*0.5*PSTEP;
5929         ddoneflag=0;
5930 if(ddebugflag<DEBUG){
5931 fprintf(DBG,"AT/ABOVE Case 3. dasim.y[%d]=%12.6e, ",\
5932     (dbin+dnn),dasim.y[dbin+dnn]);
5933 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5934     ddumnint,ddumposn,dnn);
5935 fflush(DBG);
5936 }
5937         break;
     /* the following for equipment that jumps from point to point */
5938       case 4:
5939         ddumnint-=dnslope*PSTEP;
5940         dasim.y[dbin+dnn]+=ddumnint;
5941         ddumposn+=PSTEP;
5942         dnn++;
5943         if((dnn+dbin)==dn) ddoneflag=0;
5944 if(ddebugflag<DEBUG){
5945 fprintf(DBG,"AT/ABOVE Case 4. dasim.y[%d]=%12.6e ",\
5946     (dbin+dnn-1),dasim.y[dbin+dnn-1]);
5947 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5948     ddumnint,ddumposn,dnn);
5949 fflush(DBG);
5950 }
5951         break;
5952       case 5:
5953         dwidth=ddend-ddumposn;
5954         ddumnint-=dnslope*dwidth;
5955         dasim.y[dbin+dnn]+=ddumnint;
5956         ddumposn=ddend;
5957         dnn++;
```

```
5958          if((dnn+dbin)==dn) ddoneflag=0;
5959 if(ddebugflag<DEBUG){
5960 fprintf(DBG,"AT/ABOVE Case 5. dwidth=%12.6e, dasim.y[%d]=%12.6e,
     ",dwidth,\
5961   (dbin+dnn-1),dasim.y[dbin+dnn-1]);
5962 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5963     ddumnint,ddumposn,dnn);
5964 fflush(DBG);
5965 }
5966          break;
5967        case 6:
5968          ddoneflag=0;
5969 if(ddebugflag<DEBUG){
5970 fprintf(DBG,"AT/ABOVE Case 6\n");
5971 fflush(DBG);
5972 }
5973          break;
5974        case 7:
5975          dasim.y[dbin+dnn]+=ddumnint;
5976          ddoneflag=0;
5977 if(ddebugflag<DEBUG){
5978 fprintf(DBG,"AT/ABOVE Case 7. dasim.y[%d]=%12.6e, ",\
5979     (dbin+dnn),dasim.y[dbin+dnn]);
5980 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
5981     ddumnint,ddumposn,dnn);
5982 fflush(DBG);
5983 }
5984          break;
      /* the following for emissions that provide only "spillover" */
5985        case 8:
5986          if(ddumposn==dposn){
5987            dwidth=ddend-ddumposn;
5988            ddumnint-=dnslope*dwidth;
5989            ddumposn=ddend;
5990            dnn++;
5991            if((dnn+dbin)==dn) ddoneflag=0;
5992            }
5993          else{
5994            if(ddend>(dposn+RESN)){
5995              printf(\
5996          "Fix switch case in do_sim at 1b.\n");
5997              exit(1);
5998              }
5999            else{
6000              ddumnint-=dnslope*PSTEP;
6001              ddumposn+=PSTEP;
6002              dnn++;
6003              if((dnn+dbin)==dn) ddoneflag=0;
6004              }
6005            }
6006 if(ddebugflag<DEBUG){
6007 fprintf(DBG,"AT/ABOVE Case 8. dwidth=%12.6e dbin=%d",dwidth,dbin);
6008 fprintf(DBG,"ddumnint=%12.6e, ddumposn=%12.6e, dnn=%d\n",\
6009     ddumnint,ddumposn,dnn);
```

```
6010 fflush(DBG);
6011 }
6012         break;
6013       default:
6014         printf("Go fix do_sim.\n");
6015         exit(1);
6016       } /* close switch-case */
6017     } /* close fill bins above the transition position */
6018     } /* close if frequency in range condition */
6019   } /* close if non-zero frequency condition */
6020   } /* close atomic transitions loop */
6021   } /* close atomic sets loop */
     /* start loop over previous simulation and write to file */
6022 sprintf(dnfile,"%s_simulations/Atomic_lines",PREF);
6023 DNFILE=fopen(dnfile,"w");
6024 if(DNFILE==NULL){
6025   printf("Error opening DNFILE for native pop. Exiting.\n");
6026   exit(1);
6027   }
6028 fprintf(DNFILE,"# File created by %s. \n",PROGRAM_NAME);
6029 fprintf(DNFILE,"# Contains simulated spectrum based on \n");
6030 fprintf(DNFILE,"# atomic information specified by the user
     (only).\n");
6031 fprintf(DNFILE,"# See file Mol_cascade_pop for emissions due to cas-
     cade.\n");
6032 fprintf(DNFILE,"# See file Mol_native_pop for emissions from molecu-
     lar.\n");
6033 fprintf(DNFILE,"# populations specified by the user.\n");
6034 fprintf(DNFILE,"# See file %s_Sim_All for the whole simula-
     tion.\n#\n",PREF);
6035 for(da=0;da<dn;da++){
6036   dasim.x[da]=MINV+0.5*PSTEP+da*PSTEP;
6037   fprintf(DNFILE,"%18.12e\t%18.12e\n",dasim.x[da],dasim.y[da]);
6038   }
6039 fclose(DNFILE);
     /* add all simulations together -- find maximum intensity and ad-
        just */
6040 for(da=0;da<dn;da++){
6041
     dtotsim.y[da]=dmnsim.y[da]+dmcsim.y[da]+dasim.y[da]+dosim.y[da];
6042   if(dintmax<dtotsim.y[da]) dintmax=dtotsim.y[da];
6043   }
6044 for(da=0;da<dn;da++){
6045   dtotsim.y[da]*=(MAXINT/dintmax);
6046   }
     /* write total simulation to file */
     /* start loop over previous simulation and write to file */
6047 sprintf(dnfile,"%s_simulations/%s_Sim_All",PREF,PREF);
6048 DNFILE=fopen(dnfile,"w");
6049 if(DNFILE==NULL){
6050   printf("Error opening DNFILE for native pop. Exiting.\n");
6051   exit(1);
6052   }
6053 fprintf(DNFILE,"# File created by %s. \n",PROGRAM_NAME);
```

```
6054 fprintf(DNFILE,"# Contains total simulated spectrum based on \n");
6055 fprintf(DNFILE,"# all information specified by the user as well
     as\n");
6056 fprintf(DNFILE,"# diatomic emissions due to cascade (as possi-
     ble).\n");
6057 fprintf(DNFILE,"# See documentation for information about other
     files.\n");
6058 fprintf(DNFILE,"#\n");
6059 for(da=0;da<dn;da++){
6060   dtotsim.x[da]=MINV+0.5*PSTEP+da*PSTEP;
6061   fprintf(DNFILE, "%18.12e\t %18.12e\n", dtotsim.x[da], dtot-
     sim.y[da]);
6062   }
6063 fclose(DNFILE);
6064 return;
6065 }

      /*************** manage_rotations ****************/

      /* This function determines the characteristics of the molecule as
         they relate to rotational levels and then calls another function
         to calculate energy levels and populations. */
6066 void manage_rotations(){

6067 int mra=0,mrb=0,mrstatetype=0;

6068 if(DEBUG>mrdebugflag){
6069 fprintf(DBG,"\nTop of manage rotations\n\n");
6070 fflush(DBG);
6071 }
      /* loop over molecules */
6072 for(mra=0;mra<NUMMOL;mra++){
      /* loop over states */
6073   for(mrb=0;mrb<MOL[mra].states;mrb++){
      /* determine type of state */
6074     if(MOL[mra].s[mrb].Ca>-1){
6075       if(CA[MOL[mra].s[mrb].Ca].L==0) mrstatetype=0;
6076       if(CA[MOL[mra].s[mrb].Ca].L>0) mrstatetype=2;
6077       if(CA[MOL[mra].s[mrb].Ca].S==0) mrstatetype+=0;
6078       if(CA[MOL[mra].s[mrb].Ca].S>0) mrstatetype+=1;
6079       }
6080     if(MOL[mra].s[mrb].Cb>-1){
6081       if(CB[MOL[mra].s[mrb].Cb].L==0) mrstatetype=0;
6082       if(CB[MOL[mra].s[mrb].Cb].L>0) mrstatetype=2;
6083       if(CB[MOL[mra].s[mrb].Cb].S==0) mrstatetype+=0;
6084       if(CB[MOL[mra].s[mrb].Cb].S>0) mrstatetype+=1;
6085       }
6086 if(DEBUG>mrdebugflag){
6087 fprintf(DBG,"mrstatetype is %d\n",mrstatetype);
6088 fflush(DBG);
6089 }
      /* call appropriate function */
6090     switch(mrstatetype){
6091       case 0:
6092 if(DEBUG>mrdebugflag){
```

```
6093 fprintf(DBG,"Singlet rotation function called (case 0).\n");
6094 fprintf(DBG,"Mol: %s; State:
     %s.\n",MOL[mra].Mol,MOL[mra].s[mrb].Name);
6095 fflush(DBG);
6096 }
6097          singlet_JK(mra,mrb);
6098 if(DEBUG>mrdebugflag){
6099 fprintf(DBG,"Returned from Singlet rotation function (case 0).\n");
6100 fflush(DBG);
6101 }
6102          break;
6103        case 1:
6104 if(DEBUG>mrdebugflag){
6105 fprintf(DBG,"Multiplet L=0 rotation function called.\n");
6106 fprintf(DBG,"Mol: %s; State:
     %s.\n",MOL[mra].Mol,MOL[mra].s[mrb].Name);
6107 fflush(DBG);
6108 }
6109          multiplet_zeroL_JK(mra,mrb);
6110          break;
6111        case 2:
6112 if(DEBUG>mrdebugflag){
6113 fprintf(DBG,"Singlet rotation function called. (case 2)\n");
6114 fprintf(DBG,"Mol: MOL[%d].Mol %s, ",mra,MOL[mra].Mol);
6115 fprintf(DBG,"State: MOL[%d].s[%d].Name
     %s\n",mra,mrb,MOL[mra].s[mrb].Name);
6116 fflush(DBG);
6117 }
6118          singlet_JK(mra,mrb);
6119 if(DEBUG>mrdebugflag){
6120 fprintf(DBG,"Returned from Singlet rotation function (case 0).\n");
6121 fflush(DBG);
6122 }
6123          break;
6124        case 3:
6125 if(DEBUG>mrdebugflag){
6126 fprintf(DBG,"Multiplet L>0 rotation function called.\n");
6127 fprintf(DBG,"Mol: %s; State:
     %s.\n",MOL[mra].Mol,MOL[mra].s[mrb].Name);
6128 fflush(DBG);
6129 }
6130          if(MOL[mra].s[mrb].Ca>-1){
6131 if(DEBUG>mrdebugflag){
6132 fprintf(DBG,"\tCase a \n");
6133 fflush(DBG);
6134 }
6135            multiplet_nonzeroL_JK_a(mra,mrb);
6136            }
6137          if(MOL[mra].s[mrb].Cb>-1){
6138 if(DEBUG>mrdebugflag){
6139 fprintf(DBG,"\tCase b \n");
6140 fflush(DBG);
6141 }
6142            multiplet_nonzeroL_JK_b(mra,mrb);
```

```
6143           }
6144         break;
6145       default:
6146 printf("Error in manage_rot switch case for mra=%d, mrb=%d. Exit-
     ing.\n",\
6147     mra,mrb);
6148         exit(1);
6149       } /* close switch case */
6150 if(DEBUG>mrdebugflag){
6151 fprintf(DBG,"\n switch-case just closed mra=%d,
     mrb=%d\n",mra,mrb);
6152 fflush(DBG);
6153 }
6154     } /* close loop over states */
6155 if(DEBUG>mrdebugflag){
6156 fprintf(DBG,"\nstates loop closed mra=%d, mrb=%d\n",mra,mrb);
6157 fflush(DBG);
6158 }
6159   } /* close loop over molecules */
6160 if(DEBUG>mrdebugflag){
6161 fprintf(DBG,"\n about to return in manage_rotations\n");
6162 fflush(DBG);
6163 }
6164 return;
6165 }

      /*************** multiplet_nonzero_JK_a ****************/

      /* This function calculates energy levels and rotational distribu-
        tions (if needed) for all multiplet non-Sigma states (Hund's Case
        a) */
6166 void multiplet_nonzeroL_JK_a(int mnaM, int mnaS){
6167 int mnaa=0,mnab=0,mnaJck=1,mnaL=0,mnaV=0,mnaVnum=0,mnaOnum=0;
6168 int mnaCa=0,mnaJDnum=0,mnaD=1,mnaOOD=0,mnaOODend=0,mnaf=0;
6169 double
     mnaT=0,mnaDv=0,mnaBe=0,mnaAe=0,mnaJMAX=0,mnaJMAXc=0,mnaJMAXf=0;
6170 double mnaaA=0,mnaJcut=0,mnawe=0,mnawexe=0,mnaInt=0,mnaImax=0;
6171 double mnabeta=0,mnaFv=0,mnaTe=0,mnaDe=0,mnaBv=0,mnaTeVib=0;
6172 double mnaSpin=0,mnaJcurr=0,mnaOm=0,mnaBvDv=0;
6173 int mnaJnum=0,mnaJcountnum=0,mnaAlloc=0,mnaO=0,mnaOO=0;
6174 char mnafnstr[1000];
6175 FILE *MNAROT;

6176 if(DEBUG>mnadebugflag){
6177 fprintf(DBG,"\n\nTop of multiplet_nonzeroL_JK_a. Molecule %s; State
     %s.\n\n",\
6178     MOL[mnaM].Mol,MOL[mnaM].s[mnaS].Name);
6179 fflush(DBG);
6180 }
      /* Find lowest-level temperature designation */
6181 if(MOL[mnaM].s[mnaS].T[0]!=0){
6182   mnaT=MOL[mnaM].s[mnaS].T[0];
6183 if(DEBUG>mnadebugflag){
6184 fprintf(DBG,"Temperature defined at state level: %f",mnaT);
```

```
6185 fflush(DBG);
6186 }
6187   }
6188 else{
6189   if(MOL[mnaM].T!=0){
6190     mnaT=MOL[mnaM].T;
6191 if(DEBUG>mnadebugflag){
6192 fprintf(DBG,"Temperature defined at molecule level: %f",mnaT);
6193 fflush(DBG);
6194 }
6195     }
6196   else{
6197     mnaT=TEMP;
6198 if(DEBUG>mnadebugflag){
6199 fprintf(DBG,"Temperature defined at global level: %f",mnaT);
6200 fflush(DBG);
6201 }
6202     }
6203   }
6204 if(mnaT==0){
6205   fprintf(PAR,"WARNING!! temperature defined as zero for state ");
6206   fprintf(PAR,"%s of molecule %s.\n\n",\
6207     MOL[mnaM].s[mnaS].Name,MOL[mnaM].Mol);
6208   }
      /* check for state identity sanity and complain if not sane */
6209 if(MOL[mnaM].s[mnaS].Ca==-1){ /* if somehow this isn't Hund's Case
       a */
6210   printf("Error in mutiplet_nonzeroL_JK_a. Any state this func-
     tion\n");
6211   printf("calculates should be Hund's Case A.\nState ");
6212   printf("%s of molecule %s doesn't think it's a Hund's Case a.\n",\
6213     MOL[mnaM].s[mnaS].Name,MOL[mnaM].Mol);
6214   printf("Fatal error. Exiting.\n");
6215   exit(1);
6216   }
6217   mnaCa=MOL[mnaM].s[mnaS].Ca;
6218   mnaBe=CA[mnaCa].Be;
6219   mnaAe=CA[mnaCa].ae;
6220   mnaTe=CA[mnaCa].Te;
6221   mnawe=CA[mnaCa].we;
6222   mnawexe=CA[mnaCa].wexe;
6223   mnaL=CA[mnaCa].L;
6224   mnaSpin=CA[mnaCa].S;
6225   mnaOnum=CA[mnaCa].O;
6226   if(mnaOnum==0){mnaOnum=(int)(2*mnaSpin+1);}
6227 if(DEBUG>mnadebugflag){
6228 fprintf(DBG,"State is Case (a).\nBe=%f ae=%f ",mnaBe,mnaAe);
6229 fprintf(DBG,"Te=%f; we=%f; wexe=%f;\n",mnaTe,mnawe,mnawexe);
6230 fprintf(DBG,"mnaL=%d; mnaSpin=%f; mnaOnum=%d
     ",mnaL,mnaSpin,mnaOnum);
6231 fflush(DBG);
6232 }
6233
      /* Calculate some stuff that will be useful later */
```

```
6234 if(mnawe!=0) mnaDe=4*pow(mnaBe,3)/(mnawe*mnawe);  /* Centrifugal
       distortion */
6235 if(DEBUG>mnadebugflag){
6236 fprintf(DBG,"mnaDe is %f;\n",mnaDe);
6237 fflush(DBG);
6238 }
6239 if(mnaL==0){ /* if this is a Sigma state, there's a problem */
6240 printf("How did multiplet_nonzeroL_JK_a get called if L==0? Exit-
     ing.\n");
6241   exit(1);
6242   }
     /* Print info to the parameter file */
6243 fprintf(PAR,"STATE %s is a ",MOL[mnaM].s[mnaS].Name);
6244 if(DEBUG>mnadebugflag){
6245 fprintf(DBG,"STATE %s is a ",MOL[mnaM].s[mnaS].Name);
6246 fflush(DBG);
6247 }
6248   switch(mnaL){
6249     case 1:
6250 if(DEBUG>mnadebugflag){
6251 fprintf(DBG,"Pi state -- Hund's Case (a)\n ");
6252 fflush(DBG);
6253 }
6254       fprintf(PAR,"Pi state\n");
6255       break;
6256     case 2:
6257 if(DEBUG>mnadebugflag){
6258 fprintf(DBG,"Delta state -- Hund's Case (a)\n ");
6259 fflush(DBG);
6260 }
6261       fprintf(PAR,"Delta state\n");
6262       break;
6263     case 3:
6264 if(DEBUG>mnadebugflag){
6265 fprintf(DBG,"Phi state -- Hund's Case (a)\n ");
6266 fflush(DBG);
6267 }
6268       fprintf(PAR,"Phi state\n");
6269       break;
6270     default:
6271 fprintf(PAR,"state with Lambda>3 -- an exotic state\n");
6272 fprintf(PAR,"Interpret the results from this program with
     care.\n");
6273       break;
6274     }
6275
     /* There may be multiple Omegas here */
6276 if(MOL[mnaM].s[mnaS].Dist!=-1){
     /* Estimate J max. This comes from taking the derivative of the pop-
        ulation distribution and setting it equal to zero. */
6277   mnaBv=mnaBe-mnaAe/2;
6278   mnaaA=mnaBv/(kB*mnaT);
6279   mnaJMAX=1/sqrt(2*mnaaA) - 0.5;
6280   mnaJMAXc=ceil(mnaJMAX);
```

```
6281   mnaJMAXf=floor(mnaJMAX);
6282   mnaaA=mnaBv*mnaJMAXc*(mnaJMAXc+1)/(kB*mnaT);
6283   mnaImax=(2*mnaJMAXc+1)*exp(-mnaaA);
6284   mnaaA=mnaBv*mnaJMAXf*(mnaJMAXf+1)/(kB*mnaT);
6285   mnaInt=(2*mnaJMAXf+1)*exp(-mnaaA);
6286 if(DEBUG>mnadebugflag){
6287 fprintf(DBG,"mnaBv=%f; mnaaA=%f; mnaJMAX=%f; mnaJMAXc=%f; mnaJ-
     MAXf=%f\n",\
6288     mnaBv,mnaaA,mnaJMAX,mnaJMAXc,mnaJMAXf);
6289 fprintf(DBG,"mnaImax is %f; mnaInt is %f --- ",mnaImax,mnaInt);
6290 fflush(DBG);
6291 }
6292   if((mnaImax)<(mnaInt)){mnaJMAX=mnaJMAXf;}
6293   else{mnaJMAX=mnaJMAXc;}
6294 if(DEBUG>mnadebugflag){
6295 fprintf(DBG,"mnaJMAX is now %f\n ",mnaJMAX);
6296 fflush(DBG);
6297 }
     /* Find an upper limit for J/K corresponding to the user specifica-
        tion. To find out where this equation comes from, see the docu-
        mentation, particularly the documentation found in files or di-
        rectories with the word "trick" in the name. Note that this trick
        is only good down to a cutoff intensity of about 1/10,000 of the
        maximum value. */
6298   mnaaA=mnaBv/(kB*mnaT);
6299   mnaJcut=sqrt((-2/mnaaA)/(JKm + JKb/(log(JKCUT)))) + 1/(2*mnaaA)
     - 0.5;
6300   mnaJcut=ceil(mnaJcut);
6301 if(DEBUG>mnadebugflag){
6302 fprintf(DBG,"JKCUT is %f; mnaJcut is %f\n",JKCUT,mnaJcut);
6303 fflush(DBG);
6304 }
     /* check this number and chastise programmer if not good... Also, if
        not good, find a better number using a less elegant method. */
6305   mnaaA=mnaBv*mnaJcut*(mnaJcut+1)/(kB*mnaT);
6306   mnaInt=(2*mnaJcut+1)*exp(-mnaaA);
6307   mnaaA=mnaBv*mnaJMAX*(mnaJMAX+1)/(kB*mnaT);
6308   mnaImax=(2*mnaJMAX+1)*exp(-mnaaA);
6309 if(DEBUG>mnadebugflag){
6310 fprintf(DBG,"mnaInt=%f; mnaImax=%f \n ",mnaInt,mnaImax);
6311 fflush(DBG);
6312 }
6313   if((mnaInt/mnaImax)>JKCUT){
6314     if(JKCUT>0){ /* put back to 0.0001 */
6315       printf("Dammit... Go fix the J/K business... \n");
6316       printf("The calculated mnaJcut is %5.4f ",mnaJcut);
6317       }
6318     mnaJck=1;
6319     while(mnaJck==1){
6320       mnaJcut++;
6321       mnaaA=mnaBv*mnaJcut*(mnaJcut+1)/(kB*mnaT);
6322       mnaInt=(2*mnaJcut+1)*exp(-mnaaA);
6323       if((mnaInt/mnaImax)<JKCUT){
6324         mnaJck=0;
```

```
6325          }
6326        } /*vvvvvvvvv put back to 0.0001 vvvvvvvv*/
6327      if(JKCUT>0) printf("The refined one is %5.4f \n",mnaJcut);
6328      }
6329    }
      /* Check for the distribution to be defined in a file. */
6330  if(MOL[mnaM].s[mnaS].Dist==-1){
6331    mnaD=0;
6332    mnaJDnum=(int)(CA[mnaCa].Jmax-CA[mnaCa].Jmin+1);
6333    mnaJcut=(int)CA[mnaCa].Jmax;
6334  if(DEBUG>mnadebugflag){
6335  fprintf(DBG,"Distribution is by file.\n ");
6336  fflush(DBG);
6337  }
6338    }
      /* Allocate memory for J/K array. Assign s/a and +/- flags. */
6339  mnaVnum=MOL[mnaM].s[mnaS].v[0].vnum;  /* number of vib levels */
6340  mnaAlloc=mnaVnum*mnaOnum;  /* allocate for vib levels and Omegas */
6341  MOL[mnaM].s[mnaS].nV=mnaVnum;
6342  MOL[mnaM].s[mnaS].nr=mnaAlloc;
6343  MOL[mnaM].s[mnaS].r=(rotset*)calloc(mnaAlloc, sizeof(rotset));
6344  MOL[mnaM].s[mnaS].rc=(rotset*)calloc((30*mnaOnum),
     sizeof(rotset));
6345  mnaJnum=mnaJcut+1;  /* number of J's to consider */
6346  for(mnaO=0;mnaO<mnaOnum;mnaO++){
6347    for(mnaa=0;mnaa<mnaVnum;mnaa++){
6348      mnaOO=mnaO*mnaVnum+mnaa;
6349      MOL[mnaM].s[mnaS].r[mnaOO].j=mnaJnum;
6350      MOL[mnaM].s[mnaS].r[mnaOO].J=\
6351        (double*)calloc((mnaJnum),sizeof(double));
      /* There aren't any extra E's here. If someone ever wants to add in a
         splitting factor for the +/- wavefunctions, then double the space
         available (mnaJnum) and change the indexing and energy calcula-
         tion in the loops below.*/
6352      MOL[mnaM].s[mnaS].r[mnaOO].EJ=\
6353        (double*)calloc((mnaJnum),sizeof(double));
6354      MOL[mnaM].s[mnaS].r[mnaOO].PJ=\
6355        (double*)calloc((mnaJnum),sizeof(double));
6356      MOL[mnaM].s[mnaS].r[mnaOO].CJ=\
6357        (double*)calloc((mnaJnum),sizeof(double));
6358      }
6359    }
      /* Loop through J/K values, calculating energies and populations
         (include nuclear effects if applicable). */
6360  for(mnaO=0;mnaO<mnaOnum;mnaO++){
6361    if(CA[MOL[mnaM].s[mnaS].Ca].O==0){
6362      mnaOm=mnaL-mnaSpin+mnaO;
6363      }
6364    else{
6365      mnaOm=CA[MOL[mnaM].s[mnaS].Ca].lO[mnaO];
6366      }
6367    for(mnaa=0;mnaa<mnaVnum;mnaa++){
6368      mnaOO=mnaO*mnaVnum+mnaa;
6369      mnaV=MOL[mnaM].s[mnaS].v[0].vlo+mnaa;
```

```
6370      mnaImax=0;
6371
     mnaTeVib=mnaTe+(mnaV+0.5)*mnawe-(mnaV+0.5)*(mnaV+0.5)*mnawexe;
6372 if(DEBUG>mnadebugflag){
6373 fprintf(DBG,"mnaVnum is %d; mnaTeVib=%f \n",mnaVnum,mnaTeVib);
6374 fflush(DBG);
6375 }
6376   mnaJcurr=mnaOm;
6377   if(mnawe!=0){
6378     mnaBv=mnaBe-mnaAe*(mnaV+0.5);
6379     mnaDv=mnaDe+mnabeta*(mnaV+0.5);
6380     mnaBvDv=mnaBv/(2*mnaDv);
6381     MOL[mnaM].s[mnaS].r[mnaOO].Jdissoc=\
6382       floor((sqrt(1+4*mnaBvDv)-1)/2);
6383     }
6384   else MOL[mnaM].s[mnaS].r[mnaOO].Jdissoc=RAND_MAX;
6385   if(MOL[mnaM].s[mnaS].r[mnaOO].Jdissoc>(mnaOm+mnaJnum)){
6386     mnaJcountnum=mnaJnum;
6387     }
6388   else{
6389     mnaJcountnum=(int)(MOL[mnaM].s[mnaS].r[mnaOO].Jdissoc-
     mnaOm);
6390 fprintf(PAR,"\nWARNING!! MOLECULE DISSOCIATION (see be-
     low)\nAccording to");
6391 fprintf(PAR,"the spectroscopic constants in the state file,\n
     state");
6392 fprintf(PAR,"%s(%.1f) of Molecule %s in vibration level %d dissoci-
     ates\n",\
6393     MOL[mnaM].s[mnaS].Name,mnaOm,MOL[mnaM].Mol,mnaa);
6394 fprintf(PAR,"at rotational level
     J=%.1f,",MOL[mnaM].s[mnaS].r[mnaOO].Jdissoc);
6395 fprintf(PAR," which is significantly populated at temperature
     %.1f.\n",mnaT);
6396 fprintf(PAR,"Interpret results from this simulation carefully\n");
6397     }
6398   for(mnab=0;mnab<mnaJcountnum;mnab++){
6399     MOL[mnaM].s[mnaS].r[mnaOO].J[mnab]=mnaJcurr;
6400     mnaFv=mnaBv*(mnaJcurr*(mnaJcurr+1) - mnaOm*mnaOm)-\
6401       mnaDv*mnaJcurr*mnaJcurr*(mnaJcurr+1)*(mnaJcurr+1);
6402     MOL[mnaM].s[mnaS].r[mnaOO].EJ[mnab]=mnaFv;
6403     if(mnaD!=0){
6404       mnaInt=(2*mnaJcurr+1)*exp(-mnaFv/(kB*mnaT));
6405       MOL[mnaM].s[mnaS].r[mnaOO].PJ[mnab]=mnaInt;
6406       }
6407 if(DEBUG>mnadebugflag){
6408 fprintf(DBG,"mnab is %d; mnaJcurr is %f; mnaFv=%f; mnaInt=%f\n",\
6409     mnab,mnaJcurr,mnaFv,mnaInt);
6410 fprintf(DBG,"MOL[%d].s[%d].r[%d].EJ[%d]=%f;
     MOL[%d].s[%d].r[%d].PJ[%d]=%f\n",\
6411   mnaM,mnaS,mnaa,mnab,MOL[mnaM].s[mnaS].r[mnaOO].EJ[mnab],\
6412   mnaM,mnaS,mnaa,mnab,MOL[mnaM].s[mnaS].r[mnaOO].PJ[mnab]);
6413 fflush(DBG);
6414 }
6415     mnaImax+=mnaInt;
```

```
6416 if(DEBUG>mnadebugflag){
6417 fprintf(DBG,"\n mnaImax is %f\n\n",mnaImax);
6418 fflush(DBG);
6419 }
6420     mnaJcurr++;
6421     } /* close J number loop */
6422   if(mnaD==0){ /* this means the dist is read from file */
6423     mnaOOD=(mnaO*mnaVnum+mnaa)*mnaJDnum;
6424     mnaOODend=(mnaO*mnaVnum+mnaa+1)*mnaJDnum;
6425     mnaf=(int)(CA[mnaCa].Jmin-mnaOm);
6426   for(mnab=mnaOOD;mnab<mnaOODend;mnab++){
6427     MOL[mnaM].s[mnaS].r[mnaa].PJ[mnaf]=\
6428       CA[mnaCa].Jpop[mnab];
6429     mnaInt=MOL[mnaM].s[mnaS].r[mnaa].PJ[mnaf];
6430     mnaImax+=mnaInt;
6431     mnaf++;
6432     }
6433     }
6434   mnaJcurr=mnaOm;
6435   for(mnab=0;mnab<mnaJcountnum;mnab++){
6436     MOL[mnaM].s[mnaS].r[mnaOO].EJ[mnab]+=mnaTeVib;
6437     MOL[mnaM].s[mnaS].r[mnaOO].PJ[mnab]/=mnaImax;
6438 if(DEBUG>mnadebugflag){
6439 fprintf(DBG,"%d\t%f\t%f\t%f\n",mnab,mnaJcurr,\
6440     MOL[mnaM].s[mnaS].r[mnaOO].EJ[mnab],\
6441     MOL[mnaM].s[mnaS].r[mnaOO].PJ[mnab]);
6442 fflush(DBG);
6443 }
6444     mnaJcurr++;
6445     } /* close J number loop */
6446 if(DEBUG>mnadebugflag){
6447 fprintf(DBG,"\n out of for loop \n");
6448 fflush(DBG);
6449 }
6450     } /* close vibration number loop */
6451   } /* close Omega number loop */
6452 sprintf(mnafnstr,"mkdir %s_molecules/%s",PREF,MOL[mnaM].Mol);
6453 system(mnafnstr);
6454 sprintf(mnafnstr, "%s_molecules/%s/%s_%s_rot.dat", PREF,
     MOL[mnaM].Mol,\
6455     MOL[mnaM].Mol,MOL[mnaM].s[mnaS].Name);
6456 MNAROT=fopen(mnafnstr,"w");
6457 if(MNAROT==NULL){
6458   printf("Error opening output file (singlet_JK) %s. Exiting.\n",\
6459       mnafnstr);
6460   exit(1);
6461   }
6462 fprintf(MNAROT,"## File generated by program %s.\n",PROGRAM_NAME);
6463 fprintf(MNAROT,"## This file contains ");
6464 fprintf(MNAROT,"rotational state information about\n");
6465 fprintf(MNAROT,"## state %s of molecule %s.\n",\
6466     MOL[mnaM].s[mnaS].Name,MOL[mnaM].Mol);
6467 fprintf(MNAROT,"##\tTo get the value of J add the Counter to the
     value\n");
```

```
6468 fprintf(MNAROT,"##\tof Omega for that set of columns,\n");
6469 fprintf(MNAROT,"## The columns are, in order:\n## Counter ");
6470 for(mnaO=0;mnaO<mnaOnum;mnaO++){
6471   if(CA[MOL[mnaM].s[mnaS].Ca].O==0){
6472     fprintf(MNAROT," OMEGA=%.1f<<",(mnaL-mnaSpin+mnaO));
6473     }
6474   else{
6475     fprintf(MNAROT," OMEGA=%.1f<<",\
6476         CA[MOL[mnaM].s[mnaS].Ca].lO[mnaO]);
6477     }
6478   for(mnaa=0;mnaa<mnaVnum;mnaa++){
6479     mnaV=MOL[mnaM].s[mnaS].v[0].vlo+mnaa;
6480     fprintf(MNAROT," F(v=%d) Pop(v=%d) ",mnaV,mnaV);
6481     }
6482   fprintf(MNAROT,">> ");
6483   }
6484 fprintf(MNAROT,"\n");
6485 mnaJcurr=mnaOm;
6486 for(mnab=0;mnab<mnaJcountnum;mnab++){
6487   fprintf(MNAROT,"%7d",mnab);
6488   for(mnaO=0;mnaO<mnaOnum;mnaO++){
6489     for(mnaa=0;mnaa<mnaVnum;mnaa++){
6490       mnaOO=mnaO*mnaVnum+mnaa;
6491       fprintf(MNAROT,"\t%18.10e\t%18.10e",\
6492         MOL[mnaM].s[mnaS].r[mnaOO].EJ[mnab],\
6493         MOL[mnaM].s[mnaS].r[mnaOO].PJ[mnab]);
6494       }
6495     }
6496   fprintf(MNAROT,"\n");
6497   mnaJcurr++;
6498   }
6499 fflush(MNAROT);
6500 fclose(MNAROT);
6501 if(DEBUG>mnadebugflag){
6502 fprintf(DBG,"\n out of mnaa(Vnum) loop. mnaa=%d\n",mnaa);
6503 fflush(DBG);
6504 }
6505 return;
6506 }

     /*************** multiplet_nonzero_JK_b ****************/

     /* This function calculates energy levels and rotational distribu-
       tions (if needed) for all multiplet non-Sigma states (Hund's Case
       b) */
6507 void multiplet_nonzeroL_JK_b(int mnbM, int mnbS){
6508 int mnba=0,mnbb=0,mnbc=0,mnbKck=1,mnbL=0,mnbV=0,mnbVnum=0;
6509 int mnbCb=0,mnbJDnum=0,mnbJDmin=0,mnbD=1;
6510 double
    mnbT=0,mnbDv=0,mnbBe=0,mnbAe=0,mnbKMAX=0,mnbKMAXc=0,mnbKMAXf=0;
6511 double mnbaA=0,mnbKcut=0,mnbwe=0,mnbwexe=0,mnbInt=0,mnbImax=0;
6512 double mnbbeta=0,mnbFv=0,mnbTe=0,mnbDe=0,mnbBv=0,mnbTeVib=0;
6513 double mnbSpin=0,mnbJcurr=0,mnbBvDv=0,mnbKcountnum=0;
6514 int mnbKnum=0,mnbJnum=0,mnbcc=0;
```

```
6515 char mnbfnstr[1000];
6516 FILE *MNBROT;

6517 if(DEBUG>mnbdebugflag){
6518 fprintf(DBG,"\n\nTop of multiplet_nonzeroL_JK_b. Molecule %s; State
     %s.\n\n",\
6519     MOL[mnbM].Mol,MOL[mnbM].s[mnbS].Name);
6520 fflush(DBG);
6521 }
     /* Find lowest-level temperature designation */
6522 if(MOL[mnbM].s[mnbS].T[0]!=0){
6523   mnbT=MOL[mnbM].s[mnbS].T[0];
6524 if(DEBUG>mnbdebugflag){
6525 fprintf(DBG,"Temperature defined at state level: %f",mnbT);
6526 fflush(DBG);
6527 }
6528   }
6529 else{
6530   if(MOL[mnbM].T!=0){
6531     mnbT=MOL[mnbM].T;
6532 if(DEBUG>mnbdebugflag){
6533 fprintf(DBG,"Temperature defined at molecule level: %f",mnbT);
6534 fflush(DBG);
6535 }
6536     }
6537   else{
6538     mnbT=TEMP;
6539 if(DEBUG>mnbdebugflag){
6540 fprintf(DBG,"Temperature defined at global level: %f",mnbT);
6541 fflush(DBG);
6542 }
6543     }
6544   }
6545 if(mnbT==0){
6546   fprintf(PAR,"WARNING!! temperature defined as zero for state ");
6547   fprintf(PAR,"%s of molecule %s.\n\n",\
6548     MOL[mnbM].s[mnbS].Name,MOL[mnbM].Mol);
6549   }

     /* check for state identity sanity and complain if not sane */
6550 if(MOL[mnbM].s[mnbS].Cb==-1){ /* if somehow this isn't Hund's Case
       b */
6551   printf("Error in mutiplet_nonzeroL_JK_b. Any state this func-
     tion\n");
6552   printf("calculates should be Hund's Case B.\nState ");
6553   printf("%s of molecule %s doesn't think it's a Hund's Case b.\n",\
6554     MOL[mnbM].s[mnbS].Name,MOL[mnbM].Mol);
6555   printf("Fatal error. Exiting.\n");
6556   exit(1);
6557   }
6558   mnbCb=MOL[mnbM].s[mnbS].Cb;
6559   mnbBe=CB[mnbCb].Be;
6560   mnbAe=CB[mnbCb].ae;
6561   mnbTe=CB[mnbCb].Te;
```

```
6562   mnbwe=CB[mnbCb].we;
6563   mnbwexe=CB[mnbCb].wexe;
6564   mnbL=CB[mnbCb].L;
6565   mnbSpin=CB[mnbCb].S;
6566 if(DEBUG>mnbdebugflag){
6567 fprintf(DBG,"State is Case (b).\nBe=%f ae=%f ",mnbBe,mnbAe);
6568 fprintf(DBG,"Te=%f; we=%f; wexe=%f; ",mnbTe,mnbwe,mnbwexe);
6569 fprintf(DBG," mnbL=%d; mnbSpin=%f ",mnbL,mnbSpin);
6570 fflush(DBG);
6571 }
6572
     /* Calculate some stuff that will be useful later */
6573 if(mnbwe!=0) mnbDe=4*pow(mnbBe,3)/(mnbwe*mnbwe);  /* Centrifugal
       distortion */
6574 if(DEBUG>mnbdebugflag){
6575 fprintf(DBG,"mnbDe is %f;\n",mnbDe);
6576 fflush(DBG);
6577 }
6578 if(mnbL==0){ /* if this isn't a Sigma state, there's a problem */
6579 printf("How did multiplet_nonzeroL_JK_b get called if L==0? Exit-
    ing.\n");
6580   exit(1);
6581   }
     /* Print info to the parameter file */
6582 fprintf(PAR,"STATE %s is a ",MOL[mnbM].s[mnbS].Name);
6583 if(DEBUG>mnbdebugflag){
6584 fprintf(DBG,"STATE %s is a ",MOL[mnbM].s[mnbS].Name);
6585 fflush(DBG);
6586 }
6587   switch(mnbL){
6588     case 1:
6589 if(DEBUG>mnbdebugflag){
6590 fprintf(DBG,"Pi state -- Hund's Case (b)\n ");
6591 fflush(DBG);
6592 }
6593       fprintf(PAR,"Pi state\n");
6594       break;
6595     case 2:
6596 if(DEBUG>mnbdebugflag){
6597 fprintf(DBG,"Delta state -- Hund's Case (b)\n ");
6598 fflush(DBG);
6599 }
6600       fprintf(PAR,"Delta state\n");
6601       break;
6602     case 3:
6603 if(DEBUG>mnbdebugflag){
6604 fprintf(DBG,"Phi state -- Hund's Case (b)\n ");
6605 fflush(DBG);
6606 }
6607       fprintf(PAR,"Phi state\n");
6608       break;
6609     default:
6610 fprintf(PAR,"state with Lambda>3 -- an exotic state\n");
```

```
6611 fprintf(PAR,"Interpret the results from this program with
     care.\n");
6612       break;
6613     }
6614
6615 if(MOL[mnbM].s[mnbS].Dist!=-1){
     /* Estimate J/K max. This comes from taking the derivative of the
        population distribution and setting it equal to zero. */
6616   mnbBv=mnbBe-mnbAe/2;
6617   mnbaA=mnbBv/(kB*mnbT);
6618   mnbKMAX=1/sqrt(2*mnbaA) - 0.5;
6619   mnbKMAXc=ceil(mnbKMAX);
6620   mnbKMAXf=floor(mnbKMAX);
6621   mnbaA=mnbBv*mnbKMAXc*(mnbKMAXc+1)/(kB*mnbT);
6622   mnbImax=(2*mnbKMAXc+1)*exp(-mnbaA);
6623   mnbaA=mnbBv*mnbKMAXf*(mnbKMAXf+1)/(kB*mnbT);
6624   mnbInt=(2*mnbKMAXf+1)*exp(-mnbaA);
6625 if(DEBUG>mnbdebugflag){
6626 fprintf(DBG,"mnbBv=%f; mnbaA=%f; mnbKMAX=%f; mnbKMAXc=%f; mnbJ-
     MAXf=%f\n",\
6627     mnbBv,mnbaA,mnbKMAX,mnbKMAXc,mnbKMAXf);
6628 fprintf(DBG,"mnbImax is %f; mnbInt is %f --- ",mnbImax,mnbInt);
6629 fflush(DBG);
6630 }
6631   if((mnbImax)<(mnbInt)){mnbKMAX=mnbKMAXf;}
6632   else{mnbKMAX=mnbKMAXc;}
6633 if(DEBUG>mnbdebugflag){
6634 fprintf(DBG,"mnbKMAX is now %f\n ",mnbKMAX);
6635 fflush(DBG);
6636 }
     /* Find an upper limit for J/K corresponding to the user specifica-
        tion. To find out where this equation comes from, see the docu-
        mentation, particularly the documentation found in files or di-
        rectories with the word "trick" in the name. Note that this trick
        is only good down to a cutoff intensity of about 1/10,000 of the
        maximum value. */
6637   mnbaA=mnbBv/(kB*mnbT);
6638   mnbKcut=sqrt((-2/mnbaA)/(JKm + JKb/(log(JKCUT)))) + 1/(2*mnbaA)
     - 0.5;
6639   mnbKcut=ceil(mnbKcut);
6640 if(DEBUG>mnbdebugflag){
6641 fprintf(DBG,"JKCUT is %f; mnbKcut is %f\n",JKCUT,mnbKcut);
6642 fflush(DBG);
6643 }
     /* check this number and chastise programmer if not good... Also, if
        not good, find a better number using a less elegant method. */
6644   mnbaA=mnbBv*mnbKcut*(mnbKcut+1)/(kB*mnbT);
6645   mnbInt=(2*mnbKcut+1)*exp(-mnbaA);
6646   mnbaA=mnbBv*mnbKMAX*(mnbKMAX+1)/(kB*mnbT);
6647   mnbImax=(2*mnbKMAX+1)*exp(-mnbaA);
6648 if(DEBUG>mnbdebugflag){
6649 fprintf(DBG,"mnbInt=%f; mnbImax=%f \n ",mnbInt,mnbImax);
6650 fflush(DBG);
6651 }
```

```
6652   if((mnbInt/mnbImax)>JKCUT){
6653     if(JKCUT>0){ /* put back to 0.0001 */
6654       printf("Dammit... Go fix the J/K business... \n");
6655       printf("The calculated mnbKcut is %5.4f ",mnbKcut);
6656       }
6657     mnbKck=1;
6658     while(mnbKck==1){
6659       mnbKcut++;
6660       mnbaA=mnbBv*mnbKcut*(mnbKcut+1)/(kB*mnbT);
6661       mnbInt=(2*mnbKcut+1)*exp(-mnbaA);
6662       if((mnbInt/mnbImax)<JKCUT){
6663         mnbKck=0;
6664         }
6665       } /*vvvvvvvvv put back to 0.0001 vvvvvvvv*/
6666     if(JKCUT>0) printf("The refined one is %5.4f \n",mnbKcut);
6667     }
6668   }
      /* There are no Omegas here -- check for the distribution to be de-
         fined in a file. */
6669 if(MOL[mnbM].s[mnbS].Dist==-1){
6670   mnbD=0;
6671   mnbKcut=(int)CB[mnbCb].Kmax;
6672   mnbJDmin=(int)CB[mnbCb].Kmin;
6673   mnbJDnum=(int)CB[mnbCb].Jnum;
6674 if(DEBUG>mnbdebugflag){
6675 fprintf(DBG,"Distribution is by file.\n ");
6676 fflush(DBG);
6677 }
6678   }
      /* Allocate memory for J/K array. Assign s/a and +/- flags. */
6679 mnbVnum=MOL[mnbM].s[mnbS].v[0].vnum;  /* in other functions, where
         there might be multiple Omegas, this is vnum*numomega (so far as
         memory allocation goes) */
6680 MOL[mnbM].s[mnbS].nV=mnbVnum;
6681 MOL[mnbM].s[mnbS].nr=mnbVnum;
6682 MOL[mnbM].s[mnbS].r=(rotset*)calloc(mnbVnum,sizeof(rotset));
6683 MOL[mnbM].s[mnbS].rc=(rotset*)calloc(30,sizeof(rotset));
6684 mnbKnum=mnbKcut+1;  /* number of K's to consider */
6685 mnbJnum=mnbKnum*((int)(2*(float)mnbSpin+1));  /* number of J's to
         consider -- this is the multiplcity times the number of K's con-
         sidered */
6686 for(mnba=0;mnba<mnbVnum;mnba++){
6687   MOL[mnbM].s[mnbS].r[mnba].j=mnbJnum;
6688   MOL[mnbM].s[mnbS].r[mnba].k=mnbKnum;
6689   MOL[mnbM].s[mnbS].r[mnba].J = (double*) calloc((mnbJnum),
     sizeof(double));
      /* The extra E's are redundant here, but if someone ever wants to
         add in a splitting factor for the different J's relative to the
         nearest K, then the space is available */
6690   MOL[mnbM].s[mnbS].r[mnba].EJ = (double*) calloc((mnbJnum),
     sizeof(double));
6691   MOL[mnbM].s[mnbS].r[mnba].PJ = (double*) calloc((mnbJnum),
     sizeof(double));
```

```
6692   MOL[mnbM].s[mnbS].r[mnba].CJ = (double*) calloc((mnbJnum),
     sizeof(double));
6693   }
      /* Loop through J/K values, calculating energies and populations
        (include nuclear effects if applicable). */
6694 for(mnba=0;mnba<mnbVnum;mnba++){
6695   mnbV=MOL[mnbM].s[mnbS].v[0].vlo+mnba;
6696   mnbImax=0;
6697   mnbTeVib=mnbTe+(mnbV+0.5)*mnbwe-(mnbV+0.5)*(mnbV+0.5)*mnbwexe;
6698   if(mnbwe!=0){
6699     mnbBv=mnbBe-mnbAe*(mnbV+0.5);
6700     mnbDv=mnbDe+mnbbeta*(mnbV+0.5);
6701     mnbBvDv=mnbBv/(2*mnbDv);
6702     MOL[mnbM].s[mnbS].r[mnba].Kdissoc=\
6703       floor((sqrt(1+4*mnbBvDv)-1)/2);
6704     }
6705   else MOL[mnbM].s[mnbS].r[mnba].Kdissoc=RAND_MAX;
6706   if(MOL[mnbM].s[mnbS].r[mnba].Kdissoc>((double)mnbKnum)){
6707     mnbKcountnum=mnbKnum;
6708     }
6709   else{
6710     mnbKcountnum=(int)(MOL[mnbM].s[mnbS].r[mnba].Kdissoc);
6711 fprintf(PAR,"\nWARNING!! MOLECULE DISSOCIATION (see be-
     low)\nAccording to");
6712 fprintf(PAR,"the spectroscopic constants in the state file,\n
     state");
6713 fprintf(PAR,"%s of Molecule %s in vibration level %d dissoci-
     ates\n",\
6714     MOL[mnbM].s[mnbS].Name,MOL[mnbM].Mol,mnba);
6715 fprintf(PAR,"at rotational level
     K=%.1f,",MOL[mnbM].s[mnbS].r[mnba].Kdissoc);
6716 fprintf(PAR," which is significantly populated at temperature
     %.1f.\n",mnbT);
6717 fprintf(PAR,"Interpret results from this simulation carefully\n");
6718     }
6719 if(DEBUG>mnbdebugflag){
6720 fprintf(DBG,"mnbVnum=%d; mnbTeVib=%f mnbBv=%12.6e,
     mnbDv=%12.6e\n",\
6721     mnbVnum,mnbTeVib,mnbBv,mnbDv);
6722 fflush(DBG);
6723 }
6724   for(mnbb=0;mnbb<mnbKcountnum;mnbb++){
6725     mnbFv=mnbBv*mnbb*(mnbb+1)-mnbDv*mnbb*mnbb*(mnbb+1)*(mnbb+1);
6726     for(mnbc=0;mnbc<((int)(2*mnbSpin+1));mnbc++){
6727       mnbJcurr=(double)mnbb-mnbSpin+(double)mnbc;
6728       mnbcc=mnbb*((int)(2*mnbSpin+1))+mnbc;
6729       if(mnbD!=0){
6730         mnbInt=(2*mnbJcurr+1)*exp(-mnbFv/(kB*mnbT));
6731         MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]=mnbInt;
6732         }
6733       MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc]=mnbFv;
6734       if((mnbb<mnbL)||(mnbJcurr<0)){
6735         MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc]=0;
6736         MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]=0;
```

```
6737         }
6738 if(DEBUG>mnbdebugflag){
6739 fprintf(DBG,\
6740   "mnbb is %d; mnbbeta=%12.6e, mnbJcurr is %f; mnbFv=%f; mn-
     bInt=%f\n",\
6741     mnbb,mnbbeta,mnbJcurr,mnbFv,mnbInt);
6742 fprintf(DBG,"MOL[%d].s[%d].r[%d].EJ[%d]=%f;
     MOL[%d].s[%d].r[%d].PJ[%d]=%f\n",\
6743   mnbM,mnbS,mnba,mnbcc,MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc],\
6744   mnbM,mnbS,mnba,mnbcc,MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]);
6745 fflush(DBG);
6746 }
6747       mnbImax+=mnbInt;
6748 if(DEBUG>mnbdebugflag){
6749 fprintf(DBG,"\n mnbImax is %f\n\n",mnbImax);
6750 fflush(DBG);
6751 }
6752       }
6753     }
6754   if(mnbD==0){
6755   for(mnbb=(mnba*mnbJDnum);mnbb<((mnba+1)*mnbJDnum);mnbb++){
      /* the following is algebra for: K(2S+1) + J - [K-S] */
6756
     mnbcc=(int)(mnbSpin*(2*CB[mnbCb].K[mnbb]+1)+CB[mnbCb].J[mnbb]);
6757     MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]=CB[mnbCb].Jpop[mnbb];
6758     mnbInt=MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc];
6759     mnbImax+=mnbInt;
6760     }
6761     }
6762   for(mnbb=0;mnbb<mnbKcountnum;mnbb++){
6763     for(mnbc=0;mnbc<((int)(2*mnbSpin+1));mnbc++){
6764       mnbJcurr=(double)mnbb-mnbSpin+(double)mnbc;
6765       mnbcc=mnbb*((int)(2*mnbSpin+1))+mnbc;
6766       MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc]+=mnbTeVib;
6767       MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]/=mnbImax;
6768       if((mnbb<mnbL)||(mnbJcurr<0)){
6769         MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc]=0;
6770         MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]=0;
6771         }
6772 if(DEBUG>mnbdebugflag){
6773 fprintf(DBG,"%d\t%f\t%d\t%f\t%f\n",mnbb,mnbJcurr,mnbcc,\
6774     MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc],\
6775     MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]);
6776 fflush(DBG);
6777 }
6778       }
6779     }
6780 if(DEBUG>mnbdebugflag){
6781 fprintf(DBG,"\n out of for loop \n");
6782 fflush(DBG);
6783 }
6784   }
6785 sprintf(mnbfnstr,"mkdir %s_molecules/%s",PREF,MOL[mnbM].Mol);
6786 system(mnbfnstr);
```

```
6787 sprintf(mnbfnstr, "%s_molecules/%s/%s_%s_rot.dat", PREF,
     MOL[mnbM].Mol,\
6788     MOL[mnbM].Mol,MOL[mnbM].s[mnbS].Name);
6789 MNBROT=fopen(mnbfnstr,"w");
6790 if(MNBROT==NULL){
6791   printf("Error opening output file (singlet_JK) %s. Exiting.\n",\
6792       mnbfnstr);
6793   exit(1);
6794   }
6795 fprintf(MNBROT,"## File generated by program %s.\n",PROGRAM_NAME);
6796 fprintf(MNBROT,"## This file contains ");
6797 fprintf(MNBROT,"rotational state information about\n");
6798 fprintf(MNBROT,"## state %s of molecule %s.\n",\
6799     MOL[mnbM].s[mnbS].Name,MOL[mnbM].Mol);
6800 fprintf(MNBROT,"## The columns are, in order:\n## K J");
6801 for(mnba=0;mnba<mnbVnum;mnba++){
6802   mnbV=MOL[mnbM].s[mnbS].v[0].vlo+mnba;
6803   fprintf(MNBROT,"\tF(v=%d)\tPop(v=%d)",mnbV,mnbV);
6804   }
6805 fprintf(MNBROT,"\n");
6806 for(mnbb=0;mnbb<mnbKcountnum;mnbb++){
6807   for(mnbc=0;mnbc<((int)(2*mnbSpin+1));mnbc++){
6808     mnbJcurr=(double)mnbb-mnbSpin+(double)mnbc;
6809     mnbcc=mnbb*((int)(2*mnbSpin+1))+mnbc;
6810     fprintf(MNBROT,"%d\t%f",mnbb,mnbJcurr);
6811     for(mnba=0;mnba<mnbVnum;mnba++){
6812       fprintf(MNBROT,"\t%18.10e\t%18.10e",\
6813         MOL[mnbM].s[mnbS].r[mnba].EJ[mnbcc],\
6814         MOL[mnbM].s[mnbS].r[mnba].PJ[mnbcc]);
6815       }
6816     fprintf(MNBROT,"\n");
6817     }
6818   }
6819 fflush(MNBROT);
6820 fclose(MNBROT);
6821 if(DEBUG>mnbdebugflag){
6822 fprintf(DBG,"\n out of mnba(Vnum) loop. mnba=%d\n",mnba);
6823 fflush(DBG);
6824 }
6825 return;
6826 }

     /*************** multiplet_zeroL_JK ****************/

     /* This function calculates energy levels and rotational distribu-
        tions (if needed) for all multiplet sigma states (always Hund's
        Case b) */
6827 void multiplet_zeroL_JK(int msM, int msS){
6828 int msa=0, msb=0, msc=0, msKck=1, msg=0, msL=0, msp=0, msV=0,
     msVnum=0, msItst=0;
6829 int msCb=0,msD=1,msKnum=0,msJnum=0,mscc=0,msJDmin=0,msJDnum=0;
6830 double msT=0, msDv=0, msBe=0, msAe=0, msKMAX=0, msKMAXc=0,
     msKMAXf=0, msaA=0, msKcut=0;
```

```
6831 double
     mswe=0,mswexe=0,msI=0,msInt=0,msImax=0,msbeta=0,msFv=0,msIs=0;
6832 double msIa=0, msTe=0, msDe=0, msBv=0, msTeVib=0, msSpin=0, msI-
     fac=0, msJcurr=0;
6833 double msBvDv=0;
6834 int msKcountnum=0;
6835 char msfnstr[1000];
6836 FILE *MSROT;

6837 if(DEBUG>msdebugflag){
6838 fprintf(DBG,"\n\nTop of multiplet_zeroL_JK. Molecule %s; State
     %s.\n\n",\
6839     MOL[msM].Mol,MOL[msM].s[msS].Name);
6840 fflush(DBG);
6841 }
      /* Find lowest-level temperature designation */
6842 if(MOL[msM].s[msS].T[0]!=0){
6843   msT=MOL[msM].s[msS].T[0];
6844 if(DEBUG>msdebugflag){
6845 fprintf(DBG,"Temperature defined at state level: %f",msT);
6846 fflush(DBG);
6847 }
6848   }
6849 else{
6850   if(MOL[msM].T!=0){
6851     msT=MOL[msM].T;
6852 if(DEBUG>msdebugflag){
6853 fprintf(DBG,"Temperature defined at molecule level: %f",msT);
6854 fflush(DBG);
6855 }
6856     }
6857   else{
6858     msT=TEMP;
6859 if(DEBUG>msdebugflag){
6860 fprintf(DBG,"Temperature defined at global level: %f",msT);
6861 fflush(DBG);
6862 }
6863     }
6864   }
6865 if(msT==0){
6866   fprintf(PAR,"WARNING!! temperature defined as zero for state ");
6867   fprintf(PAR,"%s of molecule %s.\n\n",\
6868     MOL[msM].s[msS].Name,MOL[msM].Mol);
6869   }

      /* check for state identity sanity and complain if not sane */
6870 if(MOL[msM].s[msS].Cb==-1){ /* if somehow this isn't Hund's Case b
       */
6871   printf("Error in mutiplet_zeroL_JK. Any state this function\n");
6872   printf("calculates should be Hund's Case B.\nState ");
6873   printf("%s of molecule %s doesn't think it's a Hund's Case b.\n",\
6874     MOL[msM].s[msS].Name,MOL[msM].Mol);
6875   printf("Fatal error. Exiting.\n");
6876   exit(1);
```

```
6877   }
6878   msCb=MOL[msM].s[msS].Cb;
6879   msBe=CB[msCb].Be;
6880   msAe=CB[msCb].ae;
6881   msTe=CB[msCb].Te;
6882   mswe=CB[msCb].we;
6883   mswexe=CB[msCb].wexe;
6884   msg=CB[msCb].g;
6885   msI=CB[msCb].I;
6886   msL=CB[msCb].L;
6887   msp=CB[msCb].p;
6888   msSpin=CB[msCb].S;
6889 if(DEBUG>msdebugflag){
6890 fprintf(DBG,"State is Case (b).\nBe=%f ae=%f ",msBe,msAe);
6891 fprintf(DBG,"Te=%f; we=%f; wexe=%f;
     \nmsg=%d",msTe,mswe,mswexe,msg);
6892 fprintf(DBG," msI=%f; msL=%d; msp=%d; msSpin=%f
     ",msI,msL,msp,msSpin);
6893 fflush(DBG);
6894 }
6895
      /* Calculate some stuff that will be useful later */
6896 if(mswe!=0) msDe=4*pow(msBe,3)/(mswe*mswe);  /* Centrifugal dis-
       tortion */
6897 if(DEBUG>msdebugflag){
6898 fprintf(DBG,"msDe is %f;\n",msDe);
6899 fflush(DBG);
6900 }
6901 if(msg!=0){
6902   msIs=1;
6903   msIa=msI/(msI+1);
6904 if(DEBUG>msdebugflag){
6905 fprintf(DBG,"msIs=%f; msIa=%f; \n",msIs,msIa);
6906 fflush(DBG);
6907 }
6908   }
6909 if(msL!=0){ /* if this isn't a Sigma state, there's a problem */
6910   printf("How did multiplet_zeroL_JK get called if L!=0? Exit-
     ing.\n");
6911   exit(1);
6912   }
6913 fprintf(PAR,"STATE %s is a Sigma",MOL[msM].s[msS].Name);
6914 if(DEBUG>msdebugflag){
6915 fprintf(DBG,"STATE %s is a Sigma",MOL[msM].s[msS].Name);
6916 fflush(DBG);
6917 }
6918 if(msg!=0){ /* if this is a homonuclear molecule */
6919   if(msp==+1){ /* and this is a Sigma+ state */
6920     fprintf(PAR,"+ ");
6921 if(DEBUG>msdebugflag){
6922 fprintf(DBG,"+ ");
6923 fflush(DBG);
6924 }
6925     if(msg==+1){ /* and the state is gerade */
```

```
6926     fprintf(PAR,"gerade state and even K's are (+), ");
6927 if(DEBUG>msdebugflag){
6928 fprintf(DBG,"gerade state and even K's are (+), ");
6929 fflush(DBG);
6930 }
6931       MOL[msM].s[msS].pmsymm=+1;
6932       if(fmod(msI,1.0)==0){
6933         MOL[msM].s[msS].Isymm=+1;
6934         fprintf(PAR,"s (boson) \n");
6935 if(DEBUG>msdebugflag){
6936 fprintf(DBG,"s (boson)\n");
6937 fflush(DBG);
6938 }
6939           }
6940       if(fmod(msI,1.0)==0.5){
6941         MOL[msM].s[msS].Isymm=-1;
6942         fprintf(PAR,"a (fermion) \n");
6943 if(DEBUG>msdebugflag){
6944 fprintf(DBG,"a (fermion)\n");
6945 fflush(DBG);
6946 }
6947         }
6948       }
6949     else{ /* and the state is ungerade */
6950     fprintf(PAR,"ungerade state and even K's are (+), ");
6951 if(DEBUG>msdebugflag){
6952 fprintf(DBG,"ungerade state and even K's are (+), ");
6953 fflush(DBG);
6954 }
6955       MOL[msM].s[msS].pmsymm=+1;
6956       if(fmod(msI,1.0)==0){
6957         MOL[msM].s[msS].Isymm=-1;
6958         fprintf(PAR,"a (boson) \n");
6959 if(DEBUG>msdebugflag){
6960 fprintf(DBG,"a (boson) \n");
6961 fflush(DBG);
6962 }
6963         }
6964       if(fmod(msI,1.0)==0.5){
6965         MOL[msM].s[msS].Isymm=+1;
6966         fprintf(PAR,"s (fermion) \n");
6967 if(DEBUG>msdebugflag){
6968 fprintf(DBG,"s (fermion)\n");
6969 fflush(DBG);
6970 }
6971         }
6972       }
6973     }
6974   else{ /* and this is a Sigma- state */
6975     fprintf(PAR,"- ");
6976 if(DEBUG>msdebugflag){
6977 fprintf(DBG,"- ");
6978 fflush(DBG);
6979 }
```

```
6980     if(msg==+1){ /* and the state is gerade */
6981     fprintf(PAR,"gerade state and even K's are (-), ");
6982 if(DEBUG>msdebugflag){
6983 fprintf(DBG,"gerade state and even K's are (-), ");
6984 fflush(DBG);
6985 }
6986       MOL[msM].s[msS].pmsymm=-1;
6987       if(fmod(msI,1.0)==0){
6988         MOL[msM].s[msS].Isymm=-1;
6989         fprintf(PAR,"a (boson) \n");
6990 if(DEBUG>msdebugflag){
6991 fprintf(DBG,"a (boson) \n");
6992 fflush(DBG);
6993 }
6994         }
6995       if(fmod(msI,1.0)==0.5){
6996         MOL[msM].s[msS].Isymm=+1;
6997         fprintf(PAR,"s (fermion) \n");
6998 if(DEBUG>msdebugflag){
6999 fprintf(DBG,"s (fermion)\n");
7000 fflush(DBG);
7001 }
7002         }
7003       }
7004     else{ /* and the state is ungerade */
7005     fprintf(PAR,"ungerade state and even K's are (-), ");
7006 if(DEBUG>msdebugflag){
7007 fprintf(DBG,"ungerade state and even K's are (-), ");
7008 fflush(DBG);
7009 }
7010       MOL[msM].s[msS].pmsymm=-1;
7011       if(fmod(msI,1.0)==0){
7012         MOL[msM].s[msS].Isymm=+1;
7013         fprintf(PAR,"s (boson) \n");
7014 if(DEBUG>msdebugflag){
7015 fprintf(DBG,"s (boson)\n");
7016 fflush(DBG);
7017 }
7018         }
7019       if(fmod(msI,1.0)==0.5){
7020         MOL[msM].s[msS].Isymm=-1;
7021         fprintf(PAR,"a (fermion) \n");
7022 if(DEBUG>msdebugflag){
7023 fprintf(DBG,"a (fermion)\n");
7024 fflush(DBG);
7025 }
7026         }
7027       }
7028     }
7029   }
7030 else{ /* and this a heteronuclear molecule */
7031   if(msp==+1){
7032     fprintf(PAR,"+ state and even K's are (+)\n");
7033     MOL[msM].s[msS].pmsymm=+1;  /* and a Sigma+ state */
```

```
7034 if(DEBUG>msdebugflag){
7035 fprintf(DBG,"+ state and even K's are (+)\n");
7036 fflush(DBG);
7037 }
7038     }
7039   else{
7040     fprintf(PAR,"- state and even K's are (-)\n");
7041     MOL[msM].s[msS].pmsymm=-1;  /* and a Sigma- state */
7042 if(DEBUG>msdebugflag){
7043 fprintf(DBG,"- state and even K's are (-)\n");
7044 fflush(DBG);
7045 }
7046     }
7047   }
7048 if(MOL[msM].s[msS].Dist!=-1){
     /* Estimate J/K max. This comes from taking the derivative of the
        population distribution and setting it equal to zero. */
7049   msBv=msBe-msAe/2;
7050   msaA=msBv/(kB*msT);
7051   msKMAX=1/sqrt(2*msaA) - 0.5;
7052   msKMAXc=ceil(msKMAX);
7053   msKMAXf=floor(msKMAX);
7054   msaA=msBv*msKMAXc*(msKMAXc+1)/(kB*msT);
7055   msImax=(2*msKMAXc+1)*exp(-msaA);
7056   msaA=msBv*msKMAXf*(msKMAXf+1)/(kB*msT);
7057   msInt=(2*msKMAXf+1)*exp(-msaA);
7058 if(DEBUG>msdebugflag){
7059 fprintf(DBG,"msBv=%f; msaA=%f; msKMAX=%f; msKMAXc=%f; msJ-
    MAXf=%f\n",\
7060     msBv,msaA,msKMAX,msKMAXc,msKMAXf);
7061 fprintf(DBG,"msImax is %f; msInt is %f --- ",msImax,msInt);
7062 fflush(DBG);
7063 }
7064   if((msImax)<(msInt)){msKMAX=msKMAXf;}
7065   else{msKMAX=msKMAXc;}
7066 if(DEBUG>msdebugflag){
7067 fprintf(DBG,"msKMAX is now %f\n ",msKMAX);
7068 fflush(DBG);
7069 }
     /* Find an upper limit for J/K corresponding to the user specifica-
        tion. To find out where this equation comes from, see the docu-
        mentation, particularly the documentation found in files or di-
        rectories with the word "trick" in the name. Note that this trick
        is only good down to a cutoff intensity of about 1/10,000 of the
        maximum value. */
7070   msaA=msBv/(kB*msT);
7071   msKcut=sqrt((-2/msaA)/(JKm + JKb/(log(JKCUT)))) + 1/(2*msaA) -
    0.5;
7072   msKcut=ceil(msKcut);
7073 if(DEBUG>msdebugflag){
7074 fprintf(DBG,"JKCUT is %f; msKcut is %f\n",JKCUT,msKcut);
7075 fflush(DBG);
7076 }
```

```
      /* check this number and chastise programmer if not good... Also, if
         not good, find a better number using a less elegant method. */
7077    msaA=msBv*msKcut*(msKcut+1)/(kB*msT);
7078    msInt=(2*msKcut+1)*exp(-msaA);
7079    msaA=msBv*msKMAX*(msKMAX+1)/(kB*msT);
7080    msImax=(2*msKMAX+1)*exp(-msaA);
7081  if(DEBUG>msdebugflag){
7082  fprintf(DBG,"msInt=%f; msImax=%f \n ",msInt,msImax);
7083  fflush(DBG);
7084  }
7085    if((msInt/msImax)>JKCUT){
7086      if(JKCUT>0){ /* put back to 0.0001 */
7087        printf("Dammit... Go fix the J/K business... \n");
7088        printf("The calculated msKcut is %5.4f ",msKcut);
7089        }
7090      msKck=1;
7091      while(msKck==1){
7092        msKcut++;
7093        msaA=msBv*msKcut*(msKcut+1)/(kB*msT);
7094        msInt=(2*msKcut+1)*exp(-msaA);
7095        if((msInt/msImax)<JKCUT){
7096          msKck=0;
7097          }
7098        } /*vvvvvvvvv put back to 0.0001 vvvvvvvv*/
7099      if(JKCUT>0) printf("The refined one is %5.4f \n",msKcut);
7100      }
7101    } /* close if distribution not in a file condition */
      /* There can't be more than one omega here -- check for the distribu-
         tion to be defined in a file. If it is, go back. */
7102  if(MOL[msM].s[msS].Dist==-1){
7103    msD=0;
7104    msKcut=(int)CB[msCb].Kmax;
7105    msJDmin=(int)CB[msCb].Kmin;
7106    msJDnum=(int)CB[msCb].Jnum;
7107  if(DEBUG>msdebugflag){
7108  fprintf(DBG,"Distribution is by file.\n ");
7109  fflush(DBG);
7110  }
7111    }
      /* Allocate memory for J/K array. Assign s/a and +/- flags. */
7112  msVnum=MOL[msM].s[msS].v[0].vnum;  /* in other functions, where
         there might be multiple Omegas, this is vnum*numomega (so far as
         memory allocation goes) */
7113  MOL[msM].s[msS].nV=msVnum;
7114  MOL[msM].s[msS].nr=msVnum;
7115  MOL[msM].s[msS].r=(rotset*)calloc(msVnum,sizeof(rotset));
7116  MOL[msM].s[msS].rc=(rotset*)calloc(30,sizeof(rotset));
7117  msKnum=msKcut+1;  /* number of K's to consider */
7118  msJnum=msKnum*((int)(2*(float)msSpin+1));  /* number of J's to con-
         sider -- this is the multiplcity times the number of K's consid-
         ered */
7119  for(msa=0;msa<msVnum;msa++){
7120    MOL[msM].s[msS].r[msa].j=msJnum;
7121    MOL[msM].s[msS].r[msa].k=msKnum;
```

```
7122   MOL[msM].s[msS].r[msa].J = (double*) calloc((msJnum),
     sizeof(double));
      /* The extra E's are redundant here, but if someone ever wants to
         add in a splitting factor for the different J's relative to the
         nearest K, then the space is available */
7123   MOL[msM].s[msS].r[msa].EJ = (double*) calloc((msJnum),
     sizeof(double));
7124   MOL[msM].s[msS].r[msa].PJ = (double*) calloc((msJnum),
     sizeof(double));
7125   MOL[msM].s[msS].r[msa].CJ = (double*) calloc((msJnum),
     sizeof(double));
7126   }
      /* Loop through J/K values, calculating energies and populations
         (include nuclear effects if applicable). */
7127 for(msa=0;msa<msVnum;msa++){
7128   msV=MOL[msM].s[msS].v[0].vlo+msa;
7129   msImax=0;
7130   msTeVib=msTe+(msV+0.5)*mswe-(msV+0.5)*(msV+0.5)*mswexe;
7131 if(DEBUG>msdebugflag){
7132 fprintf(DBG,"msVnum is %d; msTeVib=%f \n",msVnum,msTeVib);
7133 fflush(DBG);
7134 }
7135   if(mswe!=0){
7136     msBv=msBe-msAe*(msV+0.5);
7137     msDv=msDe+msbeta*(msV+0.5);
7138     msBvDv=msBv/(2*msDv);
7139     MOL[msM].s[msS].r[msa].Kdissoc=floor((sqrt(1+4*msBvDv)-
     1)/2);
7140     }
7141   else MOL[msM].s[msS].r[msa].Kdissoc=RAND_MAX;
7142 if(DEBUG>msdebugflag){
7143 fprintf(DBG,"msBv=%12.6e, msDv=%12.6e,
     msBvDv=%12.6e\n",msBv,msDv,msBvDv);
7144 fprintf(DBG,"msKcountnum=%d, msKnum=%d\n",msKcountnum,msKnum);
7145 fprintf(DBG,"MOL[%d].s[%d].r[%d].Kdissoc=%.1f\n",msM,msS,msa,\
7146     MOL[msM].s[msS].r[msa].Kdissoc);
7147 fflush(DBG);
7148 }
7149   if(MOL[msM].s[msS].r[msa].Kdissoc>((double)msKnum)){
7150     msKcountnum=msKnum;
7151     }
7152   else{
7153     msKcountnum=(int)(MOL[msM].s[msS].r[msa].Kdissoc);
7154 fprintf(PAR,"\nWARNING!! MOLECULE DISSOCIATION (see be-
     low)\nAccording to");
7155 fprintf(PAR,"the spectroscopic constants in the state file,\n
     state");
7156 fprintf(PAR,"%s of Molecule %s in vibration level %d dissoci-
     ates\n",\
7157     MOL[msM].s[msS].Name,MOL[msM].Mol,msa);
7158 fprintf(PAR,"at rotational level
     K=%.1f,",MOL[msM].s[msS].r[msa].Kdissoc);
7159 fprintf(PAR," which is significantly populated at temperature
     %.1f.\n",msT);
```

```
7160 fprintf(PAR,"Interpret results from this simulation carefully\n");
7161     }
7162 if(DEBUG>msdebugflag){
7163 fprintf(DBG,"msVnum=%d; msTeVib=%f msBv=%12.6e, msDv=%12.6e\n",\
7164     msVnum,msTeVib,msBv,msDv);
7165 fprintf(DBG,"msKcountnum=%d, msKnum=%d\n",msKcountnum,msKnum);
7166 fprintf(DBG,"MOL[%d].s[%d].r[%d].Kdissoc=%.1f\n",msM,msS,msa,\
7167     MOL[msM].s[msS].r[msa].Kdissoc);
7168 fflush(DBG);
7169 }
7170   for(msb=0;msb<msKcountnum;msb++){
7171     msFv=msBv*msb*(msb+1)-msDv*msb*msb*(msb+1)*(msb+1);
7172     msIfac=1;
7173     if((msg!=0)&&(msD!=0)){
7174       msItst=msb%2 + MOL[msM].s[msS].Isymm;
7175 if(DEBUG>msdebugflag){
7176 fprintf(DBG,"msg is %d\n",msg);
7177 fprintf(DBG,"msItst is %d\n",msItst);
7178 fflush(DBG);
7179 }
7180       switch(msItst){ /* see "switch_case.txt" in Tricks */
7181         case 0:
7182           msIfac=msIs;
7183 if(DEBUG>msdebugflag){
7184 fprintf(DBG,"msIfac=%f",msIfac);
7185 fflush(DBG);
7186 }
7187           break;
7188         case 1:
7189           msIfac=msIs;
7190 if(DEBUG>msdebugflag){
7191 fprintf(DBG,"msIfac=%f",msIfac);
7192 fflush(DBG);
7193 }
7194           break;
7195         case -1:
7196           msIfac=msIa;
7197 if(DEBUG>msdebugflag){
7198 fprintf(DBG,"msIfac=%f",msIfac);
7199 fflush(DBG);
7200 }
7201           break;
7202         case 2:
7203           msIfac=msIa;
7204 if(DEBUG>msdebugflag){
7205 fprintf(DBG,"msIfac=%f",msIfac);
7206 fflush(DBG);
7207 }
7208           break;
7209         default:
7210 printf("problem with nuclear stats in singlet_JK. Exiting. \n");
7211           exit(1);
7212         }
7213       }
```

```
7214     for(msc=0;msc<((int)(2*msSpin+1));msc++){
7215       msJcurr=(double)msb-msSpin+(double)msc;
7216       mscc=msb*((int)(2*msSpin+1))+msc;
7217       if(msD!=0){
7218         msInt=msIfac*(2*msJcurr+1)*exp(-msFv/(kB*msT));
7219 if(DEBUG>msdebugflag){
7220 fprintf(DBG,"msIfac=%.1f, msJcurr=%.1f, msFv=%.2f,
     (kB*msT)=%.1f\n",\
7221     msIfac,msJcurr,msFv,kB*msT);
7222 fprintf(DBG,"msD!=0, msInt=%f\n",msInt);
7223 fflush(DBG);
7224 }
7225         }
7226       MOL[msM].s[msS].r[msa].EJ[mscc]=msFv;
7227       if(msD!=0) MOL[msM].s[msS].r[msa].PJ[mscc]=msInt;
7228       if(msJcurr<0){
7229         MOL[msM].s[msS].r[msa].EJ[mscc]=0;
7230         MOL[msM].s[msS].r[msa].PJ[mscc]=0;
7231         }
7232 if(DEBUG>msdebugflag){
7233 fprintf(DBG,"msb is %d; msJcurr is %f; msFv=%f; msInt=%f\n",\
7234     msb,msJcurr,msFv,msInt);
7235 fprintf(DBG,"MOL[%d].s[%d].r[%d].EJ[%d]=%f;
     MOL[%d].s[%d].r[%d].PJ[%d]=%f\n",\
7236   msM, msS, msa, mscc, MOL[msM].s[msS].r[msa].EJ[mscc], msM, msS,
     msa, mscc,\
7237   MOL[msM].s[msS].r[msa].PJ[mscc]);
7238 fflush(DBG);
7239 }
7240       msImax+=msInt;
7241 if(DEBUG>msdebugflag){
7242 fprintf(DBG,"\n msImax is %f\n\n",msImax);
7243 fflush(DBG);
7244 }
7245       }
7246     }
7247   if(msD==0){
7248   for(msb=(msa*msJDnum);msb<((msa+1)*msJDnum);msb++){
      /* the following is algebra for: K(2S+1) + J - [K-S] */
7249     mscc=(int)(msSpin*(2*CB[msCb].K[msb]+1)+CB[msCb].J[msb]);
7250     MOL[msM].s[msS].r[msa].PJ[mscc]=CB[msCb].Jpop[msb];
7251     msInt=MOL[msM].s[msS].r[msa].PJ[mscc];
7252     msImax+=msInt;
7253     }
7254     }
7255   for(msb=0;msb<msKcountnum;msb++){
7256     for(msc=0;msc<((int)(2*msSpin+1));msc++){
7257       msJcurr=(double)msb-msSpin+(double)msc;
7258       mscc=msb*((int)(2*msSpin+1))+msc;
7259       MOL[msM].s[msS].r[msa].EJ[mscc]+=msTeVib;
7260       MOL[msM].s[msS].r[msa].PJ[mscc]/=msImax;
7261       if(msJcurr<0){
7262         MOL[msM].s[msS].r[msa].EJ[mscc]=0;
7263         MOL[msM].s[msS].r[msa].PJ[mscc]=0;
```

```
7264         }
7265 if(DEBUG>msdebugflag){
7266 fprintf(DBG,"%d\t%f\t%d\t%f\t%f\n",msb,msJcurr,mscc,\
7267     MOL[msM].s[msS].r[msa].EJ[mscc],\
7268     MOL[msM].s[msS].r[msa].PJ[mscc]);
7269 fflush(DBG);
7270 }
7271       }
7272     }
7273 if(DEBUG>msdebugflag){
7274 fprintf(DBG,"\n out of for loop \n");
7275 fflush(DBG);
7276 }
7277   }
7278 sprintf(msfnstr,"mkdir %s_molecules/%s",PREF,MOL[msM].Mol);
7279 system(msfnstr);
7280 sprintf(msfnstr, "%s_molecules/%s/%s_%s_rot.dat", PREF,
     MOL[msM].Mol,\
7281     MOL[msM].Mol,MOL[msM].s[msS].Name);
7282 MSROT=fopen(msfnstr,"w");
7283 if(MSROT==NULL){
7284   printf("Error opening output file (singlet_JK) %s. Exiting.\n",\
7285       msfnstr);
7286   exit(1);
7287   }
7288 fprintf(MSROT,"## File generated by program %s.\n",PROGRAM_NAME);
7289 fprintf(MSROT,"## This file contains ");
7290 fprintf(MSROT,"rotational state information about\n");
7291 fprintf(MSROT,"## state %s of molecule %s.\n",\
7292     MOL[msM].s[msS].Name,MOL[msM].Mol);
7293 fprintf(MSROT,"## The columns are, in order:\n## K J");
7294 for(msa=0;msa<msVnum;msa++){
7295   msV=MOL[msM].s[msS].v[0].vlo+msa;
7296   fprintf(MSROT,"\tF(v=%d)\tPop(v=%d)",msV,msV);
7297   }
7298 fprintf(MSROT,"\n");
7299 for(msb=0;msb<msKcountnum;msb++){
7300   for(msc=0;msc<((int)(2*msSpin+1));msc++){
7301     msJcurr=(double)msb-msSpin+(double)msc;
7302     mscc=msb*((int)(2*msSpin+1))+msc;
7303     fprintf(MSROT,"%d\t%f",msb,msJcurr);
7304     for(msa=0;msa<msVnum;msa++){
7305       fprintf(MSROT,"\t%18.10e\t%18.10e",\
7306         MOL[msM].s[msS].r[msa].EJ[mscc],\
7307         MOL[msM].s[msS].r[msa].PJ[mscc]);
7308       }
7309     fprintf(MSROT,"\n");
7310     }
7311   }
7312 fflush(MSROT);
7313 fclose(MSROT);
7314 if(DEBUG>msdebugflag){
7315 fprintf(DBG,"\n out of msa(Vnum) loop. msa=%d\n",msa);
7316 fflush(DBG);
```

```
7317 }
7318 return;
7319 }


     /*************** new_mol_ck ****************/

     /* this function checks entries to see if there is a "MOL" entry
        where the program doesn't expect one. If it finds such an entry,
        it writes a message to stdout and to the parameter file and termi-
        nates the program. */
7320 char * new_mol_ck(char *curr_mol, char *curr_st){

7321 char nmol_ck[201];
7322 fscanf(INTR.F,"%s",nmol_ck);
7323 if(strcmp("MOL",nmol_ck)==0){
7324 printf("Unexpected MOL entry in state %s of molecule %s.\n",\
7325   curr_st,curr_mol);
7326 printf("Please edit transition file and restart program. Exit-
     ing.\n");
7327 fprintf(PAR,"Unexpected MOL entry in state %s of molecule %s.\n",\
7328   curr_st,curr_mol);
7329 fprintf(PAR,"Please edit transition file and restart program. Exit-
     ing.\n");
7330   exit(1);
7331   }
7332 return nmol_ck;
7333 }


     /*************** read_FCF_file() ****************/

     /* This function reads the FCF file. */
7334 void read_FCF_file(){

7335 int rpC=0, rpa=0, rpb=0, rpc=0, rpd=0, rpz=0, rpy=0, rpw=1, rpv=1,
     rpvs=0, rpck=0, rpcc=0;
7336 char rpsys[500],rpdum[500],rpfn[400],rpstrtoss[100];
7337 FILE *RPF;
7338 double rptoss=0,rpsumf=0,rpsumfvn=0,rpsumfvc=0,rpcma=0,rpcm=0;
7339 double rpTeH=0,rpTeL=0,rpweH=0,rpweL=0,rpwexeH=0,rpwexeL=0;

7340 sprintf(rpfn,".%s.temp1",PREF);
7341 sprintf(rpdum,".%s.temp2",PREF);
7342 for(rpa=0;rpa<NUMMOL;rpa++){
7343   for(rpb=0;rpb<MOL[rpa].trans;rpb++){
     /* Get the state and determine the relevant spectroscopic con-
        stants. Also check for presence of multiple Omegas <<< This
        doesn't work at the moment. Below, only the first set of fcf's is
        assigned.*/
7344   if(MOL[rpa].s[MOL[rpa].t[rpb].Hi].Ca!=-1){
7345     rpC=MOL[rpa].s[MOL[rpa].t[rpb].Hi].Ca;
7346     rpw=CA[rpC].O;
7347     MOL[rpa].t[rpb].Ohi=rpw;
7348     if(rpw<1){rpw=1;}
7349     rpTeH=CA[rpC].Te;
```

```
7350      rpweH=CA[rpC].we;
7351      rpwexeH=CA[rpC].wexe;
7352      }
7353    if(MOL[rpa].s[MOL[rpa].t[rpb].Hi].Cb!=-1){
7354      rpC=MOL[rpa].s[MOL[rpa].t[rpb].Hi].Cb;
7355      rpw=1;
7356      rpTeH=CB[rpC].Te;
7357      rpweH=CB[rpC].we;
7358      rpwexeH=CB[rpC].wexe;
7359      }
       /* change this if support for Hund's Case C or D is added */
7360    if((MOL[rpa].s[MOL[rpa].t[rpb].Hi].Ca==-1)&&\
7361      (MOL[rpa].s[MOL[rpa].t[rpb].Hi].Cb==-1)){
7362 printf("State %s of molecule
     ",MOL[rpa].s[MOL[rpa].t[rpb].Hi].Name);
7363 printf("%s is neither Hund's Case a nor b.
     Exiting.\n",MOL[rpa].Mol);
7364      exit(1);
7365      }
7366    if(MOL[rpa].s[MOL[rpa].t[rpb].Lo].Ca!=-1){
7367      rpC=MOL[rpa].s[MOL[rpa].t[rpb].Lo].Ca;
7368      rpv=CA[rpC].O;
7369      MOL[rpa].t[rpb].Olo=rpv;
7370      if(rpv<1){rpv=1;}
7371      rpTeL=CA[rpC].Te;
7372      rpweL=CA[rpC].we;
7373      rpwexeL=CA[rpC].wexe;
7374      }
7375    if(MOL[rpa].s[MOL[rpa].t[rpb].Lo].Cb!=-1){
7376      rpC=MOL[rpa].s[MOL[rpa].t[rpb].Lo].Cb;
7377      rpw=1;
7378      rpTeH=CB[rpC].Te;
7379      rpweH=CB[rpC].we;
7380      rpwexeH=CB[rpC].wexe;
7381      }
       /* change this if support for Hund's Case C or D is added */
7382    if((MOL[rpa].s[MOL[rpa].t[rpb].Lo].Ca==-1)&&\
7383      (MOL[rpa].s[MOL[rpa].t[rpb].Lo].Cb==-1)){
7384 printf("State %s of molecule
     ",MOL[rpa].s[MOL[rpa].t[rpb].Lo].Name);
7385 printf("%s is neither Hund's Case a nor b.
     Exiting.\n",MOL[rpa].Mol);
7386      exit(1);
7387      }
7388 MOL[rpa].t[rpb].v=(vfcf*)calloc(30*rpv*rpw,sizeof(vfcf));
7389 if(DEBUG>rpdebugflag){
7390 fprintf(DBG,"rpv=%d, rpw=%d\n",rpv,rpw);
7391 }

7392 for(rpz=0;rpz<rpw;rpz++){
7393    for(rpy=0;rpy<rpv;rpy++){
7394      for(rpc=0;rpc<30;rpc++){
7395        rpcc=rpz*rpv*30+rpy*30+rpc;  /* This is the 3D part of a 4D
        data set. This (3D) set is indexed so that the primary set is the
```

```
        set of upper-level vibrational numbers. The next level is the
        number of lower-state Omegas. There are upper-state-Omega-number
        of sets of (30 upper vib levels)x(lower-state-Omega-number)
        pieces of data. later, each one of the lower vib levels gets its
        own set of 30 lower vib levels. */

      /* In output from Ervin's program, the second quantum number is that
        of the higher state. So, this grep is asking for all the entries
        that go to the high state rpc */
7396        sprintf(rpsys,"grep \"\\->%3d\" %s > %s",\
7397          rpc,MOL[rpa].t[rpb].f[rpz*rpv+rpy].f,rpfn);
7398 if(DEBUG>rpdebugflag){
7399   fprintf(DBG,"grep rpsys is %s\n",rpsys);
7400 }
7401        system(rpsys);
7402        sprintf(rpsys,"wc -l %s > %s",rpfn,rpdum);
7403 if(DEBUG>rpdebugflag){
7404   fprintf(DBG,"wc rpsys is %s\n",rpsys);
7405 }
7406        system(rpsys);
7407        RPF=fopen(rpdum,"r");
7408        if(RPF==NULL){
7409 printf("Error opening temporary file 2 in read FCF for rpa=%d,
     ",rpa);
7410 printf("rpb=%d, and rpc=%d; filename %s. Exiting.\n",\
7411     rpb,rpc,MOL[rpa].t[rpb].f[rpz*rpv+rpy].f);
7412          exit(1);
7413          }
7414        fscanf(RPF,"%d",&rpvs);
7415        fclose(RPF);
7416        RPF=fopen(rpfn,"r");
7417        if(RPF==NULL){
7418 printf("Error opening temporary file 1 in read FCF for rpa=%d,
     ",rpa);
7419 printf("rpb=%d, and rpc=%d; filename %s. Exiting.\n",\
7420     rpb,rpc,MOL[rpa].t[rpb].f[rpz*rpv+rpy].f);
7421          exit(1);
7422          }
7423 if(DEBUG>rpdebugflag){
7424   fprintf(DBG,"The following are the entries from the FCF
     file.\n");
7425   fprintf(DBG,"Entries that are saved to an array are in brack-
     ets[].\n");
7426   fprintf(DBG,"rpvs is %d\n",rpvs);
7427 }
7428   rpcma=rpTeH+rpweH*(rpc+0.5)-rpwexeH*(rpc+0.5)*(rpc+0.5)-rpTeL;
7429        rpsumf=rpsumfvn=rpsumfvc=0;
7430        for(rpd=0;rpd<rpvs;rpd++){
7431     rpcm=rpcma-rpweL*(rpd+0.5)+rpwexeL*(rpd+0.5)*(rpd+0.5);
7432     fscanf(RPF,"%d",&MOL[rpa].t[rpb].v[rpc].vqn[rpd]);
7433          fscanf(RPF,"%s",rpstrtoss);  /* comma */
7434          fscanf(RPF,"%d",&rpck);  /* vhi */
7435 if(DEBUG>rpdebugflag){
```

```
7436 fprintf(DBG,"[%d] %s %d
     ",MOL[rpa].t[rpb].v[rpc].vqn[rpd],rpstrtoss,rpck);
7437 }
7438         if(rpck!=rpc){
7439 printf("rpc not equal to rpck(%d) for rpa=%d, rpb=%d, rpc=%d,
     rpd=%d\n",\
7440             rpck,rpa,rpb,rpc,rpd);
7441         exit(1);
7442         }
7443         fscanf(RPF,"%s ",rpstrtoss);  /* comma */
7444         fscanf(RPF,"%lf",&rptoss);  /* wavenumbers */
7445 if(DEBUG>rpdebugflag){
7446 fprintf(DBG,"%s %f ",rpstrtoss,rptoss);
7447 fflush(DBG);
7448 }
7449         fscanf(RPF,"%s ",rpstrtoss);  /* comma */
7450     fscanf(RPF,"%lf",&MOL[rpa].t[rpb].v[rpc].fcfn[rpd]);
7451         if(rptoss<=0){ /* negative trans. frequency */
7452           MOL[rpa].t[rpb].v[rpc].fcfn[rpd]=0;
7453           }
7454         MOL[rpa].t[rpb].v[rpc].fcfc[rpd]=\
7455           MOL[rpa].t[rpb].v[rpc].fcfn[rpd];
7456         rpsumf+=MOL[rpa].t[rpb].v[rpc].fcfc[rpd];
7457 if(DEBUG>rpdebugflag){
7458 fprintf(DBG,"%s
     [%f](n)\n",rpstrtoss,MOL[rpa].t[rpb].v[rpc].fcfn[rpd]);
7459 fprintf(DBG, "MOL[%d].t[%d].v[%d].fcfc[%d] = %f\n", rpa, rpb, rpc,
     rpd,\
7460     MOL[rpa].t[rpb].v[rpc].fcfc[rpd]);
7461 fflush(DBG);
7462 }
7463         MOL[rpa].t[rpb].v[rpc].fcfc[rpd]*=pow(rpcm,3);
7464         MOL[rpa].t[rpb].v[rpc].fcfn[rpd]*=\
7465           pow(rpcm,DETECTTYPE);
7466         rpsumfvn+=MOL[rpa].t[rpb].v[rpc].fcfn[rpd];
7467         rpsumfvc+=MOL[rpa].t[rpb].v[rpc].fcfc[rpd];
7468 if(DEBUG>rpdebugflag){
7469 fprintf(DBG, "rpsumf=%f [%f](n)\n", rpsumf,
     MOL[rpa].t[rpb].v[rpc].fcfn[rpd]);
7470 fprintf(DBG, "MOL[%d].t[%d].v[%d].fcfc[%d] = %f\n", rpa, rpb, rpc,
     rpd,\
7471     MOL[rpa].t[rpb].v[rpc].fcfc[rpd]);
7472 fprintf(DBG,"rpsumfvn=%13.6e ;
     rpsumfvc=%13.6e\n",rpsumfvn,rpsumfvc);
7473 fflush(DBG);
7474 }
7475         fscanf(RPF,"%s ",rpstrtoss);  /* comma */
7476         fscanf(RPF,"%lf",&rptoss);
7477 if(DEBUG>rpdebugflag){
7478 fprintf(DBG,"%s %f ",rpstrtoss,rptoss);
7479 fflush(DBG);
7480 }
7481         fscanf(RPF,"%s ",rpstrtoss);  /* comma */
7482         fscanf(RPF,"%lf",&rptoss);  /* pec3 */
```

```
7483 if(DEBUG>rpdebugflag){
7484 fprintf(DBG,"%s %f\n",rpstrtoss,rptoss);
7485 fflush(DBG);
7486 }
7487         } /* close read fcf & nm for loop */
      /* This piece of code scales FCF's for the frequency factor. This
        isn't the best way to do this for two reasons. One, this should be
        done at the rot-->rot transition level. Two, if the franck-condon
        factors don't all add to one, then this becomes an approximation
        to an approximation. */
7488       if((rpsumf-1)>0.01){
7489 printf("rpsumf for state %s of
     molecule",MOL[rpa].s[MOL[rpa].t[rpb].Hi].Name);
7490 printf(" %s is %f (greater than one).
     Exiting.\n",MOL[rpa].Mol,rpsumf);
7491         exit(1);
7492         }
7493       if((1-rpsumf)>0.01){
7494 fprintf(PAR,"WARNING!!: The sum of the Franck-Condon factors (rp-
     sumf) for");
7495 fprintf(PAR,"\n\tstate %s ",MOL[rpa].s[MOL[rpa].t[rpb].Hi].Name);
7496 fprintf(PAR,"of molecule %s is %f
     (v-hi=%d).\n",MOL[rpa].Mol,rpsumf,rpc);
7497         }
7498       for(rpd=0;rpd<rpvs;rpd++){
7499       MOL[rpa].t[rpb].v[rpc].fcfc[rpd]*=rpsumf/rpsumfvc;
7500       MOL[rpa].t[rpb].v[rpc].fcfn[rpd]*=rpsumf/rpsumfvn;
7501 if(DEBUG>rpdebugflag){
7502 fprintf(DBG,"MOL[%d].t[%d].v[%d].fcfn[%d] = %f ; ", rpa, rpb, rpc,
     rpd,\
7503     MOL[rpa].t[rpb].v[rpc].fcfn[rpd]);
7504 fprintf(DBG,"MOL[%d].t[%d].v[%d].fcfc[%d] = %f\n", rpa, rpb, rpc,
     rpd,\
7505     MOL[rpa].t[rpb].v[rpc].fcfc[rpd]);
7506 fflush(DBG);
7507 }
7508         } /* close scale fcf for frequency loop */
7509       } /* close vib levels for loop */
7510     } /* close rpy (num low Omegas) loop */
7511   } /* close rpz (num high Omegas) loop */
7512     } /* close transitions for loop */
7513   } /* close NUMMOL for loop */

7514 sprintf(rpsys,"rm %s %s",rpfn,rpdum);
7515 system(rpsys);
7516 return;
7517 }


      /*************** read_atomic_file() ****************/

      /* This function reads atomic files. */
7518 void read_atomic_files(){

7519 int raa=0,rab=0,raz=0;
```

```
7520 char rtmp[500];
7521 FILE *ATF;

7522 if(DEBUG>radebugflag){
7523 fprintf(DBG,"Top of read_atomic_files. NUMAT=%d.\n",NUMAT);
7524 fflush(DBG);
7525 }
7526 for(raa=0;raa<NUMAT;raa++){
7527   sprintf(rtmp,"wc -l %s > %s",AT[raa].f.f,TMPFILE);
7528 if(DEBUG>radebugflag){
7529 fprintf(DBG,"System call string is %s.\n",rtmp);
7530 fflush(DBG);
7531 }
7532   system(rtmp);
7533   SYS=fopen(TMPFILE,"r");
7534   if(SYS==NULL){
7535 printf("Error opening atomic temporary file #%d. Exiting.\n",raa);
7536     exit(1);
7537     }
7538   fscanf(SYS,"%d",&raz);
7539 if(DEBUG>radebugflag){
7540 fprintf(DBG,"raz is %d.\n",raz);
7541 fflush(DBG);
7542 }
7543   AT[raa].n=raz;
7544   fclose(SYS);
7545   AT[raa].x=(double*)calloc(raz,sizeof(double));
7546   AT[raa].A=(double*)calloc(raz,sizeof(double));
7547   AT[raa].pop=(double*)calloc(raz,sizeof(double));
7548   ATF=fopen(AT[raa].f.f,"r");
7549   if(ATF==NULL){
7550 printf("Error opening atomic info file #%d. Exiting.\n",raa);
7551     exit(1);
7552     }
7553   for(rab=0;rab<raz;rab++){
7554     fscanf(ATF,"%lf %lf %lf",&AT[raa].x[rab],&AT[raa].A[rab],\
7555         &AT[raa].pop[rab]);
7556 if(DEBUG>radebugflag){
7557 fprintf(DBG,"xposition is %f, A is
     %f\n",AT[raa].x[rab],AT[raa].A[rab]);
7558 fflush(DBG);
7559 }
7560     }
7561   fclose(ATF);
7562   }
7563 return;
7564 }
```

```
/*************** read_rot_dist_file() ****************/

/* This function reads rotational distribution files.

A note on the J(K) array: positions in the array are given by:
```

```
      rdc*rdJK*rdv + rdd*rdv + rde

      where rdc is the current Omega number (not the value of this Omega);
        rdJK is the total number of lines in the J(K) distribution file
        as determined by "wc -l filename"; rdv is the number of vibration
        levels to be considered; rdd is the current J(K) number; rde is
        the current vibrational level number. */

7565 void read_rot_dist_files(){

7566 int rd-
     numf=1,rda=0,rdb=0,rdc=0,rdd=0,rde=0,rdv=0,rdJK=0,rdsz=0,rdcs=0;
7567 int rdcc=0,rddd=0,rdee=0,rdnum=0;
7568 double rdJmax=0,rdJprev=0,rdJdum=0;
7569 char rdsys[500];

7570 sprintf(TMPFILE,".%s.temporary",PREF);
      /* start loop over molecules and states. Reinitialize rdnumf */
7571 for(rda=0;rda<NUMMOL;rda++){
7572 if(DEBUG>rddebugflag){
7573 fprintf(DBG,"Top of rda loop for rda=%d. NUMMOL is
     %d\n",rda,NUMMOL);
7574 fflush(DBG);
7575 }
7576   for(rdb=0;rdb<MOL[rda].states;rdb++){
7577 if(DEBUG>rddebugflag){
7578 fprintf(DBG,"Top of rdb loop for rdb=%d\n",rdb);
7579 fflush(DBG);
7580 }
      /* For each state, determine if there are files to read and, if so,
        is the state Case a or Case b, and the position in the Case array
        */
7581 if(MOL[rda].s[rdb].Dist==-1){
      /* determine the number of vibrational levels to consider */
7582   rdv=MOL[rda].s[rdb].v[0].vnum;
7583 if(DEBUG>rddebugflag){
7584 fprintf(DBG,"State %s (molecule %s) has distribution file(s) ",\
7585     MOL[rda].s[rdb].Name,MOL[rda].Mol);
7586 fprintf(DBG,"and vnum is %d\n",MOL[rda].s[rdb].v[0].vnum);
7587 fflush(DBG);
7588 }
      /* if Case a, figure out what rdnumf really is */
7589   if(MOL[rda].s[rdb].Ca>-1){
7590     rdcs=MOL[rda].s[rdb].Ca;
7591     rdnumf=CA[rdcs].O;
7592     rdJK=0;
7593     for(rdc=0;rdc<rdnumf;rdc++){
7594       sprintf(rdsys,"wc -l %s > %s",\
7595         MOL[rda].s[rdb].f[rdc].f,TMPFILE);
7596       system(rdsys);
7597 if(DEBUG>rddebugflag){
7598 fprintf(DBG,"System string is %s\n",rdsys);
7599 fflush(DBG);
7600 }
7601       SYS=fopen(TMPFILE,"r");
```

```
7602       if(SYS==NULL){
7603 printf("Error opening temporary file for file %s. Exiting.\n",\
7604       MOL[rda].s[rdc].f[rdc].f);
7605         exit(1);
7606         }
7607       fscanf(SYS,"%d",&rdd);
7608       if(rdd>rdJK) rdJK=rdd;
7609       fclose(SYS);
7610       }
7611     rdsz=rdnumf*rdv*rdJK;
7612     CA[rdcs].Jpop=(double*)calloc(rdsz,sizeof(double));
7613 if(DEBUG>rddebugflag){
7614 fprintf(DBG,"MOL[%d].Mol (%s) MOL[%d].s[%d].Name (%s)
    ",rda,MOL[rda].Mol,\
7615     rda,rdb,MOL[rda].s[rdb].Name);
7616 fprintf(DBG,"is defined as Case a.\nThere are %d lines in the
    file.\n",rdJK);
7617 fflush(DBG);
7618 }
7619     }
7620   else{
7621     if(MOL[rda].s[rdb].Cb==-1){
7622 printf("Molecule Name %s State %s has no defined case. Exit-
    ing.\n",\
7623       MOL[rda].Mol,MOL[rda].s[rdb].Name);
7624       exit(1);
7625       }
7626     rdcs=MOL[rda].s[rdb].Cb;
7627     rdnum=1;
7628     sprintf(rdsys,"wc -l %s > %s",\
7629         MOL[rda].s[rdb].f[rdc].f,TMPFILE);
7630     system(rdsys);
7631     SYS=fopen(TMPFILE,"r");
7632     if(SYS==NULL){
7633 printf("Error opening temporary file for file %s. Exiting.\n",\
7634     MOL[rda].s[rdc].f[rdc].f);
7635       exit(1);
7636       }
7637     fscanf(SYS,"%d",&rdJK);
7638     fclose(SYS);
7639     rdsz=rdv*rdJK;
7640     CB[rdcs].Jnum=rdJK;
7641     CB[rdcs].Jpop=(double*)calloc(rdsz,sizeof(double));
7642     CB[rdcs].K=(double*)calloc(rdJK,sizeof(double));
7643     CB[rdcs].J=(double*)calloc(rdJK,sizeof(double));
7644 if(DEBUG>rddebugflag){
7645 fprintf(DBG,"MOL[%d].Mol (%s) MOL[%d].s[%d].Name (%s)
    ",rda,MOL[rda].Mol,\
7646     rda,rdb,MOL[rda].s[rdb].Name);
7647 fprintf(DBG,"is defined as Case b.\nThere are %d lines in the
    file.\n",rdJK);
7648 fflush(DBG);
7649 }
7650     }
```

```
      /* start loop over files [rdc] -- there will only be one file for
         Case b or for Case a where a list of Omegas aren't specified */
7651    for(rdc=0;rdc<rdnumf;rdc++){
7652      rdcc=rdc*rdJK*rdv;
7653      rdJdum=rdJprev=rdJmax=0;
7654 if(DEBUG>rddebugflag){
7655 fprintf(DBG,"rdcc is %d, rdc is %d\n",rdcc,rdc);
7656 fflush(DBG);
7657 }
7658    MOL[rda].s[rdb].f[rdc].F=fopen(MOL[rda].s[rdb].f[rdc].f,"r");
7659      if(MOL[rda].s[rdb].f[rdc].F==NULL){
7660 printf("Error opening distribution file %s. Exiting.\n",\
7661      MOL[rda].s[rdb].f[rdc].f);
7662        exit(1);
7663        }
      /* start loop over lines in file [rdd] and read in J/K values */
7664      for(rdd=0;rdd<rdJK;rdd++){
7665        rddd=rdd*rdv;
7666        fscanf(MOL[rda].s[rdb].f[rdc].F,"%lf",&rdJdum);
7667 if(DEBUG>rddebugflag){
7668 fprintf(DBG,"rddd id %d, rdd is %d, rdJdum is
      %f\n",rddd,rdd,rdJdum);
7669 fflush(DBG);
7670 }
      /* check to make sure that the J(K) values in the file are sequential
         */
7671        if(rdd==0){
7672          rdJprev=rdJdum;
7673          if(MOL[rda].s[rdb].Ca>-1){
7674            CA[rdcs].Jmin=rdJdum;
7675 if(DEBUG>rddebugflag){
7676 fprintf(DBG,"CA[%d].Jmin is %f\n",rdcs,CA[rdcs].Jmin);
7677 fflush(DBG);
7678 }
7679            }
7680          else{
7681            CB[rdcs].Kmin=rdJdum;
7682            CB[rdcs].K[rdd]=rdJdum;
7683    fscanf(MOL[rda].s[rdb].f[rdc].F,"%lf",&CB[rdcs].J[rdd]);
7684 if(DEBUG>rddebugflag){
7685 fprintf(DBG,"CB[%d].Kmin is %f\n",rdcs,CB[rdcs].Kmin);
7686 fprintf(DBG,"CB[%d].K[%d] is %f\n",rdcs,rdd,CB[rdcs].K[rdd]);
7687 fprintf(DBG,"CB[%d].J[%d] is %f\n",rdcs,rdd,CB[rdcs].J[rdd]);
7688 fflush(DBG);
7689 }
7690            }
7691          }
7692        else{
7693          if(rdJdum!=(rdJprev+1)){
7694 printf("Non-sequential J(or K) values found in file %s ",\
7695      MOL[rda].s[rdb].f[rdc].f);
7696 printf("between lines numbered %d and %d.\n",rdd,(rdd-1));
7697 printf("The two J-values read are %f and %f.\n",rdJprev,rdJdum);
```

```
7698 printf("Exiting. Please edit distribution file and restart pro-
     gram.\n");
7699 fprintf(PAR,"Non-sequential J(or K) values found in file %s ",\
7700     MOL[rda].s[rdb].f[rdc].f);
7701 fprintf(PAR,"between lines numbered %d and %d.\n",rdd,(rdd-1));
7702 fprintf(PAR,"The two J-values read are %f and
     %f.\n",rdJprev,rdJdum);
7703 fprintf(PAR,"Exiting. Please edit distribution file and restart
     program.\n");
7704          exit(1);
7705            }
7706          if(MOL[rda].s[rdb].Cb>-1){
7707            CB[rdcs].K[rdd]=rdJdum;
7708   fscanf(MOL[rda].s[rdb].f[rdc].F,"%lf",&CB[rdcs].J[rdd]);
7709 if(DEBUG>rddebugflag){
7710 fprintf(DBG,"CB[%d].K[%d] is %f\n",rdcs,rdd,CB[rdcs].K[rdd]);
7711 fprintf(DBG,"CB[%d].J[%d] is %f\n",rdcs,rdd,CB[rdcs].J[rdd]);
7712 fflush(DBG);
7713 }
7714            }
7715          rdJprev=rdJdum;
      /* check for maximum J(K) value. */
7716          if(rdJdum>rdJmax) rdJmax=rdJdum;
7717 if(DEBUG>rddebugflag){
7718 fprintf(DBG,"rdJmax is %f\n",rdJmax);
7719 fflush(DBG);
7720 }
7721          }
      /* read in the populations for each vib level [rde] */
7722        for(rde=0;rde<rdv;rde++){
7723          rdee=rdcc+rddd+rde;
7724          if(MOL[rda].s[rdb].Ca>-1){
7725            fscanf(MOL[rda].s[rdb].f[rdc].F,\
7726              "%lf",&CA[rdcs].Jpop[rdee]);
7727 if(DEBUG>rddebugflag){
7728 fprintf(DBG,"rdee is %d, rde is %d, ",rdee,rde);
7729 fprintf(DBG,"CA[%d].Jpop[%d] is
     %f\n",rdcs,rdee,CA[rdcs].Jpop[rdee]);
7730 fflush(DBG);
7731 }
7732            }
7733          if(MOL[rda].s[rdb].Cb>-1){
7734            fscanf(MOL[rda].s[rdb].f[rdc].F,\
7735              "%lf",&CB[rdcs].Jpop[rdee]);
7736 if(DEBUG>rddebugflag){
7737 fprintf(DBG,"rdee is %d, rde is %d, ",rdee,rde);
7738 fprintf(DBG,"CB[%d].Jpop[%d] is
     %f\n",rdcs,rdee,CB[rdcs].Jpop[rdee]);
7739 fflush(DBG);
7740 }
7741            }
7742          } /* close read over v-levels */
7743        } /* close loop over lines in file (J or K values) */
7744      fclose(MOL[rda].s[rdb].f[rdc].F);
```

```
7745     if(MOL[rda].s[rdb].Ca>-1){
7746       CA[rdcs].Jmax=rdJmax;
7747 if(DEBUG>rddebugflag){
7748 fprintf(DBG,"rdJmax is %f CA[%d].Jmax is
     %f\n",rdJmax,rdcs,CA[rdcs].Jmax);
7749 fflush(DBG);
7750 }       }
7751     if(MOL[rda].s[rdb].Cb>-1){
7752       CB[rdcs].Kmax=rdJmax;
7753 if(DEBUG>rddebugflag){
7754 fprintf(DBG,"rdJmax is %f CB[%d].Kmax is
     %f\n",rdJmax,rdcs,CB[rdcs].Kmax);
7755 fflush(DBG);
7756 }
7757       }
7758     } /* close loop over rdnumf (num Omegas if Case a, else 1) */
7759 } /* close if Dist==-1 condition */
7760     } /* close loop over number of states */
7761   } /* close loop over number of molecules */
7762 return;
7763 }


     /*********** read_state_and_transition_files() ************/

     /* This function reads the transition file and the state file. */
7764 void read_state_and_transition_files(){

7765 char tdum[2000],tdumm[8],tsys[2000],tstring[2000],thom[15]="N";
7766 int ta=0,tb=0,tc=0,td=0,te=0,tz=0,ty=0,tx=0,tw=0,tv=0,tu=0,tt=0;
7767 int tisomega=1,tnumomega=1,tf=0;
7768 char tmolck[201],tmoldum[201],tstdum[201],tsysdir[1000];
7769 double tmult=0,tomeglo=0,tomeghi=0,tmodO=1,tmodS=1,tvmaxpop=0;
7770 FILE *TSYSF;

7771 INTR.F=fopen(INTR.f,"r");
7772 if(INTR.F==NULL){
7773 printf("Error opening transition file. Exiting.\n");
7774 exit(1);
7775 }
7776 fscanf(INTR.F,"%lf",&TEMP);
7777 if(DEBUG>tdebugflag){
7778 fprintf(DBG,"%f\n",TEMP);
7779 fflush(DBG);
7780 }
7781 if(TEMP==0){ /* write warning to parameter file */
7782 fprintf(PAR,"WARNING!!! Temperature NOT defined globally.\n");
7783 fprintf(PAR,"If there is not a temperature designation for any mol-
     ecule or\n");
7784 fprintf(PAR,"state, that molecule or state will be at T=0
     (brrr....).\n\n");
7785   }
7786 else{ /* write global temperature to parameter file */
7787 fprintf(PAR,"Temperature defined globally as %f.\n\n",TEMP);
7788   }
```

```
7789   fflush(PAR);

7790 fscanf(INTR.F,"%d",&NUMMOL);
7791 if(DEBUG>tdebugflag){
7792 fprintf(DBG,"NUMMOL is %d\n",NUMMOL);
7793 fflush(DBG);
7794 }

      /******* BEGIN LOOP OVER MOLECULES **************/
7795 MOL=(Molecule *)calloc(NUMMOL, sizeof(Molecule));
7796 for(ta=0;ta<NUMMOL;ta++){
7797   sprintf(tmoldum,"Molecule number %d",ta);
7798   strcpy(tstdum,"No state specified yet");
7799   fscanf(INTR.F,"%s",tmolck);
7800   if(strcmp(tmolck,"MOL")!=0){
7801 printf("MOL entry not found at beginning of data for molecule");
7802 printf("number %d. Exiting. \nPlease edit transition file
     and",(ta+1));
7803 printf(" restart program.\n");
7804 fprintf(PAR,"MOL entry not found at beginning of data for mol-
     ecule");
7805 fprintf(PAR,"number %d. Exiting. \nPlease edit transition file
     and",(ta+1));
7806 fprintf(PAR," restart program.\n");
7807     exit(1);
7808     }
7809   strcpy(tmolck,new_mol_ck(tmoldum,tstdum));
7810   sscanf(tmolck,"%s",MOL[ta].Mol);
7811   strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
7812   sscanf(tmolck,"%lf",&MOL[ta].pop);
7813   strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
7814   sscanf(tmolck,"%lf",&MOL[ta].T);
7815   sprintf(tsysdir,"mkdir %s_molecules/%s",PREF,MOL[ta].Mol);
7816   system(tsysdir);
7817 if(DEBUG>tdebugflag){
7818 fprintf(DBG,"The molecule is %s with pop %f and temp
     %f\n",MOL[ta].Mol,\
7819     MOL[ta].pop,MOL[ta].T);
7820 fflush(DBG);
7821 }
7822   if(MOL[ta].T==0){
7823 fprintf(PAR,"Temperature NOT specified for molecule
     %s.\n\n",MOL[ta].Mol);
7824     }
7825   else{
7826 fprintf(PAR,"Temperature specified as %f for molecule %s.\n\n",\
7827     MOL[ta].T,MOL[ta].Mol);
7828     }
7829   fflush(PAR);
7830   strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
7831   sscanf(tmolck,"%d",&MOL[ta].states);
7832   NUMST+=MOL[ta].states;
7833 if(DEBUG>tdebugflag){
7834 fprintf(DBG,"MOL[%d].states is %d\n",ta,MOL[ta].states);
```

```
fflush(DBG);
}
  MOL[ta].s=(State *)calloc(MOL[ta].states, sizeof(State));
  for(tb=0;tb<MOL[ta].states;tb++){
    MOL[ta].s[tb].Ca=-1;
    MOL[ta].s[tb].Cb=-1;
    MOL[ta].s[tb].Cc=-1;
    MOL[ta].s[tb].Cd=-1;
    }
if(DEBUG>tdebugflag){
fprintf(DBG,"Mol. %s Rel. Pop. is %f at Temp. %f\tMOL[%d].states
is %d\n",\
    MOL[ta].Mol,MOL[ta].pop,MOL[ta].T,ta,MOL[ta].states);
fflush(DBG);
}

     /*** OPEN LOOP OVER STATES ***/
  for(tb=0;tb<MOL[ta].states;tb++){
    tisomega=1;
    tnumomega=1;
    sprintf(tstdum,"State number %d",(tb+1));
    strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
    sscanf(tmolck,"%s",MOL[ta].s[tb].Name);
    strcpy(tmolck,new_mol_ck(MOL[ta].Mol,MOL[ta].s[tb].Name));
    sscanf(tmolck,"%lf",&MOL[ta].s[tb].pop);
     /* read state info before getting to the Omega issue */
     /*sprintf(tsys,"grep
       \"%s %s[ |\t]\" %s > .%s.temporary",MOL[ta].Mol,\<<- this is an
       alternate form of the next half-line. */
sprintf(tsys,"grep \"%s %s[[:space:]]\" %s >
.%s.temporary",MOL[ta].Mol,\
  MOL[ta].s[tb].Name,INST.f,PREF);
    sprintf(tstring,".%s.temporary",PREF);
    system(tsys);
if(DEBUG>tdebugflag){
fprintf(DBG,"tsys is %s; tstring is %s\n",tsys, tstring);
fflush(DBG);
}
    TSYSF=fopen(tstring,"r");
    if(TSYSF==NULL){
printf("Error opening temporary file # %d for molecule %s in
read_trans. \
Exiting.\n",tb,MOL[ta].Mol);
      exit(1);
      }
    fscanf(TSYSF,"%s",tdumm);
    if(strcmp(tdumm,MOL[ta].Mol)!=0){
printf("Error reading MOL in temporary file for tdumm=%s and
MOL[%d].Mol=%s \
and MOL[%d].s[%d].Name=%s.
Exiting.\n",tdumm,ta,MOL[ta].Mol,ta,tb,\
    MOL[ta].s[tb].Name);
      exit(1);
      }
```

```
7878     fscanf(TSYSF,"%s",tdumm);
7879     if(strcmp(tdumm,MOL[ta].s[tb].Name)!=0){
7880 printf("Error reading state in temporary file for tdumm=%s and
     MOL[%d].Mol=%s\
7881 and MOL[%d].s[%d].Name=%s.
     Exiting.\n",tdumm,ta,MOL[ta].Mol,ta,tb,\
7882     MOL[ta].s[tb].Name);
7883       exit(1);
7884       }
7885     fscanf(TSYSF,"%s",MOL[ta].s[tb].Case);
7886     if(strcmp(MOL[ta].s[tb].Case,"a")==0){
7887       tc=Case_a_num;
7888       MOL[ta].s[tb].Ca=Case_a_num;
7889 if(DEBUG>tdebugflag){
7890 fprintf(DBG,"MOL[%d].s[%d].Ca is %d\n",ta,tb,MOL[ta].s[tb].Ca);
7891 fflush(DBG);
7892 }
7893       CA=(Case_a_stateinfo*)realloc(CA,\
7894           ((tc+1)*sizeof(Case_a_stateinfo)));
7895       fscanf(TSYSF,"%d",&CA[tc].L);
7896 if(DEBUG>tdebugflag){
7897 fprintf(DBG,"State %s of molecule %s has Lambda=%d and is \n",\
7898     MOL[ta].s[tb].Name,MOL[ta].Mol,CA[tc].L);
7899 fprintf(DBG,"defined as Case a.\n");
7900 fflush(DBG);
7901 }
7902       if(CA[tc].L==0){
7903 fprintf(PAR,"State %s of molecules %s
     ",MOL[ta].s[tb].Name,MOL[ta].Mol);
7904 fprintf(PAR,"has Lambda=0\n and is defined as Case a.\n");
7905 fprintf(PAR,"This program isn't set up to call Sigma states Case
     a.\n");
7906 fprintf(PAR,"Redefining this state as Case b.\n");
7907 fflush(PAR);
7908         strcpy(MOL[ta].s[tb].Case,"b");
7909         tc=Case_b_num;
7910         MOL[ta].s[tb].Cb=Case_b_num;
7911         MOL[ta].s[tb].Ca=-1;
7912         CB=(Case_b_stateinfo*)realloc(CB,\
7913           ((tc+1)*(sizeof(Case_b_stateinfo))));
7914         CB[tc].L=0;
7915         fscanf(TSYSF,"%d",&CB[tc].p);
7916         fscanf(TSYSF,"%lf",&tmult);
7917         CB[tc].S=(tmult-1)/2;
7918         CB[tc].sflag=fmod(CB[tc].S,1);
7919         fscanf(TSYSF,"%s",thom);
7920         if(strcmp(thom,"Y")==0){
7921           fscanf(TSYSF,"%d",&CB[tc].g);
7922           fscanf(TSYSF,"%lf",&CB[tc].I);
7923           }
7924         fscanf(TSYSF,"%lf",&CB[tc].Te);
7925         fscanf(TSYSF,"%lf",&CB[tc].we);
7926         fscanf(TSYSF,"%lf",&CB[tc].wexe);
7927         fscanf(TSYSF,"%lf",&CB[tc].Be);
```

```
7928          fscanf(TSYSF,"%lf",&CB[tc].ae);
7929          tnumomega=1;  /* Case b, tnumomega = 1 */
7930          Case_b_num++;
7931 if(DEBUG>tdebugflag){
7932 fprintf(DBG,"Redefined as Case (b): MOL[%d].s[%d].Cb is %d\n",\
7933      ta,tb,MOL[ta].s[tb].Cb);
7934 fprintf(DBG,"tc: %d; Mult: %f; S: %f\n",tc,tmult,CB[tc].S);
7935 fprintf(DBG,"Homonuclear? %s; g: %d; I: %f;
     \n",thom,CB[tc].g,CB[tc].I);
7936 fprintf(DBG,"Te=%f; we=%f; wexe=%f; Be=%f;
     ae=%f\n",CB[tc].Te,CB[tc].we,\
7937      CB[tc].wexe,CB[tc].Be,CB[tc].ae);
7938 fflush(DBG);
7939 }
7940 if(CB[tc].Be==0){
7941   printf("It isn't possible to simulate a rotational spectrum
     if\n");
7942   printf("the rotational constant (Be) is zero. Call me picky. :-
     )\n");
7943   printf("The program will exit. Either edit the state file
     for\n");
7944   printf("state %s of molecule %s or delete its transition(s).\n",\
7945     MOL[ta].s[tb].Name,MOL[ta].Mol);
7946   exit(1);
7947   }
7948          }
7949        else{
7950          fscanf(TSYSF,"%lf",&tmult);
7951          CA[tc].S=(tmult-1)/2;
7952          CA[tc].sflag=fmod(CA[tc].S,1);
7953          fscanf(TSYSF,"%s",thom);
7954          if(strcmp(thom,"Y")==0){
7955            fscanf(TSYSF,"%d",&CA[tc].g);
7956            fscanf(TSYSF,"%lf",&CA[tc].I);
7957            }
7958          fscanf(TSYSF,"%lf",&CA[tc].Te);
7959          fscanf(TSYSF,"%lf",&CA[tc].we);
7960          fscanf(TSYSF,"%lf",&CA[tc].wexe);
7961          fscanf(TSYSF,"%lf",&CA[tc].Be);
7962          fscanf(TSYSF,"%lf",&CA[tc].ae);
7963 if(DEBUG>tdebugflag){
7964 fprintf(DBG,"tc: %d; Mult: %f; S: %f\n",tc,tmult,CA[tc].S);
7965 fprintf(DBG,"Homonuclear? %s; g: %d; I: %f;
     \n",thom,CA[tc].g,CA[tc].I);
7966 fprintf(DBG,"Te=%f; we=%f; wexe=%f; Be=%f;
     ae=%f\n",CA[tc].Te,CA[tc].we,\
7967      CA[tc].wexe,CA[tc].Be,CA[tc].ae);
7968 fflush(DBG);
7969 }
7970          tnumomega=tmult;  /* Case a, tnumomega = the multiplicity
        unless the user specifies otherwise (see below) */
7971          Case_a_num++;
7972 if(CA[tc].Be==0){
```

```
7973   printf("It isn't possible to simulate a rotational spectrum
     if\n");
7974   printf("the rotational constant (Be) is zero. Call me picky. :-
     )\n");
7975   printf("The program will exit. Either edit the state file
     for\n");
7976   printf("state %s of molecule %s or delete its transition(s).\n",\
7977     MOL[ta].s[tb].Name,MOL[ta].Mol);
7978   exit(1);
7979   }
7980         }
7981       }
7982 if((strcmp(MOL[ta].s[tb].Case,"b")==0)&&(MOL[ta].s[tb].Cb==-1)){
7983       tc=Case_b_num;
7984       MOL[ta].s[tb].Cb=Case_b_num;
7985 CB = (Case_b_stateinfo*) realloc(CB,
     ((tc+1)*sizeof(Case_b_stateinfo)));
7986       fscanf(TSYSF,"%d",&CB[tc].L);
7987       if(CB[tc].L==0){
7988         fscanf(TSYSF,"%d",&CB[tc].p);
7989         }
7990 if(DEBUG>tdebugflag){
7991 fprintf(DBG,"State %s of molecules %s has Lambda=%d and is \n",\
7992     MOL[ta].s[tb].Name,MOL[ta].Mol,CB[tc].L);
7993 fprintf(DBG,"defined as Case b.\n");
7994 fflush(DBG);
7995 }
7996       fscanf(TSYSF,"%lf",&tmult);
7997       tmodS=(float)fmod(tmult,1);
7998       if(tmodS!=0){
7999 printf("Multiplicities must be integers. For state %s
     ",MOL[ta].s[tb].Name);
8000 printf("of molecule %s, ",MOL[ta].Mol);
8001 printf("the multiplicity is %6.2f. Exiting.\n",tmult);
8002 printf("Please edit transtion file or state file and restart pro-
     gram.\n");
8003 fprintf(PAR,"Multiplicities must be integers. ");
8004 fprintf(PAR,"For state %s ",MOL[ta].s[tb].Name);
8005 fprintf(PAR,"of molecule %s, ",MOL[ta].Mol);
8006 fprintf(PAR,"the multiplicity is %6.2f. Exiting.\n",tmult);
8007 fprintf(PAR,"Please edit transtion file or state file and restart
     program.\n");
8008         exit(1);
8009         }
8010       CB[tc].S=(tmult-1)/2;
8011       CB[tc].sflag=fmod(CB[tc].S,1);
8012       fscanf(TSYSF,"%s",thom);
8013       if(strcmp(thom,"Y")==0){
8014         fscanf(TSYSF,"%d",&CB[tc].g);
8015         fscanf(TSYSF,"%lf",&CB[tc].I);
8016         }
8017       fscanf(TSYSF,"%lf",&CB[tc].Te);
8018       fscanf(TSYSF,"%lf",&CB[tc].we);
8019       fscanf(TSYSF,"%lf",&CB[tc].wexe);
```

```
8020       fscanf(TSYSF,"%lf",&CB[tc].Be);
8021       fscanf(TSYSF,"%lf",&CB[tc].ae);
8022 if(DEBUG>tdebugflag){
8023 fprintf(DBG,"tc: %d; Mult: %f; S: %f\n",tc,tmult,CB[tc].S);
8024 fprintf(DBG,"Homonuclear? %s; g: %d; I: %f;
     \n",thom,CB[tc].g,CB[tc].I);
8025 fprintf(DBG,"Te=%f; we=%f; wexe=%f; Be=%f;
     ae=%f\n",CB[tc].Te,CB[tc].we,\
8026     CB[tc].wexe,CB[tc].Be,CB[tc].ae);
8027 fflush(DBG);
8028 }
8029       tnumomega=1;  /* for Case b, tnumomega is always 1 */
8030       Case_b_num++;
8031 if(CB[tc].Be==0){
8032   printf("It isn't possible to simulate a rotational spectrum
     if\n");
8033   printf("the rotational constant (Be) is zero. Call me picky. :-
     )\n");
8034   printf("The program will exit. Either edit the state file
     for\n");
8035   printf("state %s of molecule %s or delete its transition(s).\n",\
8036     MOL[ta].s[tb].Name,MOL[ta].Mol);
8037   exit(1);
8038   }
8039       }
8040     fclose(TSYSF);
8041     if((MOL[ta].s[tb].Cb==-1)&&(MOL[ta].s[tb].Ca==-1)){
8042 printf("Case neither a nor b in temporary file for
     MOL[ta].s[tb].Case=%s and \
8043 MOL[%d].Mol=%s and MOL[%d].s[%d].Name=%s.
     Exiting.\n",MOL[ta].s[tb].Case,ta,
8044 MOL[ta].Mol,ta,tb,MOL[ta].s[tb].Name);
8045       exit(1);
8046       }

      /* Read in statement about Omega Specifications. Then, read in vi-
        brational level information and other info as appropriate */
8047     strcpy(tmolck,new_mol_ck(MOL[ta].Mol,MOL[ta].s[tb].Name));
8048     sscanf(tmolck,"%s",tdum);
8049     if(strcmp(tdum,"Y")==0){
8050       if(MOL[ta].s[tb].Ca==-1){
8051 printf("Omega restriction requested for a state that isn't Case
     a\n");
8052 printf("This program doesn't understand that. Exiting.\n");
8053 printf("Please edit transtion (or state) file and restart pro-
     gram.\n\n");
8054 fprintf(PAR,"Omega restriction requested for a state that isn't
     Case a\n");
8055 fprintf(PAR,"This program doesn't understand that. Exiting.\n");
8056 fprintf(PAR,"Please edit transtion (or state) file and restart pro-
     gram.\n\n");
8057 fflush(PAR);
8058         exit(1);
8059         }
```

```
8060        tisomega=0;
8061        tv=MOL[ta].s[tb].Ca;
8062        strcpy(tmolck,new_mol_ck(MOL[ta].Mol,\
8063          MOL[ta].s[tb].Name));
8064        sscanf(tmolck,"%d",&tw);
8065          CA[tv].O=tw;
8066        if(tw<1){
8067          tw=2*CA[tv].S+1;
8068          tisomega=1;
8069 fprintf(PAR,"WARNING!!! Omega restriction requested, but with the
     default\n");
8070 fprintf(PAR,"!! number of omegas. The program will not read for a
     set of\n");
8071 fprintf(PAR,"!! user-specified omegas and populations. If the pro-
     gram\n");
8072 fprintf(PAR,"!! exits after complaining of a MOL entry in the wrong
     place,\n");
8073 fprintf(PAR,"!! this might be the reason why. \n\n");
8074          }
8075        tnumomega=tw;
8076        CA[tv].lO=(double*)calloc(tw,sizeof(double));
8077        CA[tv].pO=(double*)calloc(tw,sizeof(double));
8078        tomeglo=fabs((double)CA[tv].L-CA[tv].S);
8079        tomeghi=(double)CA[tv].L+CA[tv].S;
8080        tmodS=(float)fmod(CA[tv].S,1);
8081        }
8082 MOL[ta].s[tb].nO=tnumomega;
8083 MOL[ta].s[tb].f=(fileset *)calloc(tnumomega,sizeof(fileset));
8084 MOL[ta].s[tb].T=(double *)calloc(tnumomega,sizeof(double));
8085 MOL[ta].s[tb].v=(vset *)calloc(tnumomega,sizeof(vset));
8086 if(DEBUG>tdebugflag){
8087 fprintf(DBG,"tnumomega is %d. Just allocated f, T,
     v.\n",tnumomega);
8088 fflush(DBG);
8089   }
8090   if(tisomega!=0){
8091     tnumomega=1;
8092     }
8093     for(te=0;te<tnumomega;te++){
8094       if(tisomega==0){
8095         strcpy(tmolck,new_mol_ck(MOL[ta].Mol,\
8096           MOL[ta].s[tb].Name));
8097           sscanf(tmolck,"%lf",&CA[tv].lO[te]);
8098           tmodO=(float)fmod(CA[tv].lO[te],1);
8099   if((CA[tv].lO[te]<tomeglo)||(CA[tv].lO[te]>tomeghi)){
8100 printf("For state %s of molecule %s,
     ",MOL[ta].s[tb].Name,MOL[ta].Mol);
8101 printf("Omega values must lie between %f and
     %f.\n",tomeglo,tomeghi);
8102 printf("You requested an Omega of %f. Exiting.\n",CA[tv].lO[te]);
8103 printf("Please edit the transition file (or the state file) and ");
8104 printf("restart the program.\n\n");
8105 fprintf(PAR,"For state %s of molecule %s,
     ",MOL[ta].s[tb].Name,MOL[ta].Mol);
```

```
8106 fprintf(PAR,"Omega values must lie between %f and
     %f.\n",tomeglo,tomeghi);
8107 fprintf(PAR,"You requested an Omega of %f.
     Exiting.\n",CA[tv].lO[te]);
8108 fprintf(PAR,"Please edit the transition file (or the state file) and
     ");
8109 fprintf(PAR,"restart the program.\n\n");
8110     exit(1);
8111     }
8112   if(tmodS!=tmodO){
8113 printf("For state %s of molecule %s,
     ",MOL[ta].s[tb].Name,MOL[ta].Mol);
8114 printf("Omega values for spins equal to %3.1f ",CA[tv].S);
8115 if(tmodS==0.5){printf("must be half-integral.\n");}
8116 if(tmodS==0){printf("must be integral.\n");}
8117 printf("You requested an Omega of %f. Exiting.\n",CA[tv].lO[te]);
8118 printf("Please edit the transition file (or the state file) and ");
8119 printf("restart the program.\n\n");
8120 fprintf(PAR,"For state %s of molecule %s,
     ",MOL[ta].s[tb].Name,MOL[ta].Mol);
8121 fprintf(PAR,"Omega values for spins equal to %3.1f ",CA[tv].S);
8122 if(tmodS==0.5){fprintf(PAR,"must be half-integral.\n");}
8123 if(tmodS==0){fprintf(PAR,"must be integral.\n");}
8124 fprintf(PAR,"You requested an Omega of %f.
     Exiting.\n",CA[tv].lO[te]);
8125 fprintf(PAR,"Please edit the transition file (or the state file) and
     ");
8126 fprintf(PAR,"restart the program.\n\n");
8127     exit(1);
8128     }
8129         strcpy(tmolck,new_mol_ck(MOL[ta].Mol,\
8130           MOL[ta].s[tb].Name));
8131         sscanf(tmolck,"%lf",&CA[tv].pO[te]);
8132         }
8133 strcpy(tmolck,new_mol_ck(MOL[ta].Mol,MOL[ta].s[tb].Name));
8134     sscanf(tmolck,"%s",tdum);
8135     tz=ty=tx=0;
8136     for(td=0;td<strlen(tdum);td++){
8137       ty=isdigit((int)tdum[td]);
8138       if((int)tdum[td]==46) tx++;
8139       if((ty==0)&&((int)tdum[td]!=46)) tz=1;
8140       }
8141     if((tz==0)&&(tx<=1)){
8142       sscanf(tdum,"%lf",&MOL[ta].s[tb].T[te]);
8143       MOL[ta].s[tb].Dist=1;
8144 fprintf(PAR,"Temp for state %s of Molecule %s is
     %f\n",MOL[ta].s[tb].Name,\
8145     MOL[ta].Mol,MOL[ta].s[tb].T[te]);
8146 fflush(PAR);
8147 if(DEBUG>tdebugflag){
8148 fprintf(DBG,"Temp for state %s of Molecule %s is
     %f\n",MOL[ta].s[tb].Name,\
8149     MOL[ta].Mol,MOL[ta].s[tb].T[te]);
8150 fflush(DBG);
```

```
8151 }
8152       } /* close if tdum is a number (read temperature) */
8153     else{
8154       sscanf(tdum,"%s",MOL[ta].s[tb].f[te].f);
8155       MOL[ta].s[tb].Dist=-1;
8156       MOL[ta].s[tb].f[te].F=fopen(MOL[ta].s[tb].f[te].f,"r");
8157       if(MOL[ta].s[tb].f[te].F==NULL){
8158 printf("Cannot open input file %s.
     Exiting.\n",MOL[ta].s[tb].f[te].f);
8159         exit(1);
8160         }
8161       fclose(MOL[ta].s[tb].f[te].F);
8162 fprintf(PAR,"Distribution file for state %s of Molecule %s is
     %s\n",\
8163     MOL[ta].s[tb].Name,MOL[ta].Mol,MOL[ta].s[tb].f[te].f);
8164 fflush(PAR);
8165 if(DEBUG>tdebugflag){
8166 fprintf(DBG,"Dist (%d) file for state %s of Molecule %s is %s\n",\
8167   MOL[ta].s[tb].Dist,MOL[ta].s[tb].Name,\
8168   MOL[ta].Mol,MOL[ta].s[tb].f[te].f);
8169 fflush(DBG);
8170 }
8171       } /* close if tdum isn't a number (open file) */
8172 if(DEBUG>tdebugflag){
8173 fprintf(DBG, "%s\t %f\t %s\t (%d)\n", MOL[ta].s[tb].Name,
     MOL[ta].s[tb].pop, tdum,\
8174     MOL[ta].s[tb].Dist);
8175 fflush(DBG);
8176 }
8177     strcpy(tmolck,new_mol_ck(MOL[ta].Mol,MOL[ta].s[tb].Name));
8178     sscanf(tmolck,"%d",&MOL[ta].s[tb].v[te].vnum);
8179 if(DEBUG>tdebugflag){
8180 fprintf(DBG,"vnum is %d\n",MOL[ta].s[tb].v[te].vnum);
8181 fflush(DBG);
8182 }
8183     if(MOL[ta].s[tb].v[te].vnum>0){
8184       strcpy(tmolck,new_mol_ck(MOL[ta].Mol,\
8185             MOL[ta].s[tb].Name));
8186       sscanf(tmolck,"%d",&MOL[ta].s[tb].v[te].vlo);
8187 if(DEBUG>tdebugflag){
8188 fprintf(DBG,"vlo = %d ",MOL[ta].s[tb].v[te].vlo);
8189 fflush(DBG);
8190 }
8191 MOL[ta].s[tb].v[te].p=(double*)calloc(MOL[ta].s[tb].v[te].vnum,\
8192     sizeof(double));
8193 MOL[ta].s[tb].v[te].c = (int*) calloc(MOL[ta].s[tb].v[te].vnum,
     sizeof(int));
8194       }
8195     tt=MOL[ta].s[tb].v[te].vlo;
8196     tvmaxpop=0;
8197     for(td=0;td<MOL[ta].s[tb].v[te].vnum;td++){
8198       strcpy(tmolck,new_mol_ck(MOL[ta].Mol,\
8199             MOL[ta].s[tb].Name));
8200       sscanf(tmolck,"%lf",&MOL[ta].s[tb].v[te].p[td]);
```

```
8201       tvmaxpop+=MOL[ta].s[tb].v[te].p[td];
8202 if(DEBUG>tdebugflag){
8203 fprintf(DBG,"MOL[%d].s[%d].v[%d].p[%d]=%f\n",\
8204     ta,tb,te,td,MOL[ta].s[tb].v[te].p[td]);
8205 fflush(DBG);
8206 }
8207       }
     /* Make sure the vibrational populations sum to one in the simula-
       tion */
8208     for(td=0;td<MOL[ta].s[tb].v[te].vnum;td++){
8209       MOL[ta].s[tb].v[te].p[td]/=tvmaxpop;
8210       }
     /* Beginning of scan for file containing rotational distributions
       */
8211     if(MOL[ta].s[tb].Dist==-1){
8212       if((MOL[ta].s[tb].Ca!=-1)&&(MOL[ta].s[tb].Cb!=-1)){
8213 printf("MOL[%d].s[%d].Ca is %d\n",ta,tb,MOL[ta].s[tb].Ca);
8214 printf("MOL[%d].s[%d].Cb is %d\n",ta,tb,MOL[ta].s[tb].Cb);
8215         printf("fix Ca/Cb assignment problem\n");
8216         exit(1);
8217         }
8218       sprintf(tsys,"head -1 %s > .%s.temporary", \
8219           MOL[ta].s[tb].f[te].f,PREF);
8220       system(tsys);
8221       TSYSF=fopen(tstring,"r");
8222       if(TSYSF==NULL){
8223         printf("fix tstring 1\n");
8224         exit(1);
8225         }
8226       fscanf(TSYSF,"%lf",&tmult);
8227       if(MOL[ta].s[tb].Ca!=-1){
8228         CA[tc].Jmin=tmult;
8229         }
8230       if(MOL[ta].s[tb].Cb!=-1){
8231         CB[tc].Kmin=tmult;
8232         }
8233       fclose(TSYSF);
8234 if(DEBUG>tdebugflag){
8235 fprintf(DBG,"string for MOL[%d].s[%d].f[%d].f is %s\n",\
8236     ta,tb,te,MOL[ta].s[tb].f[te].f);
8237 if(MOL[ta].s[tb].Ca!=-1){
8238 fprintf(DBG,"Jmin is %f, ",CA[tc].Jmin);
8239 }
8240 if(MOL[ta].s[tb].Cb!=-1){
8241 fprintf(DBG,"Kmin is %f, ",CB[tc].Kmin);
8242 }
8243 fflush(DBG);
8244 }
8245       sprintf(tsys,"tail -1 %s > .%s.temporary", \
8246           MOL[ta].s[tb].f[te].f,PREF);
8247       system(tsys);
8248       TSYSF=fopen(tstring,"r");
8249       if(TSYSF==NULL){
8250         printf("fix tstring 2\n");
```

```
8251          exit(1);
8252          }
8253        fscanf(TSYSF,"%lf",&tmult);
8254        if(MOL[ta].s[tb].Ca!=-1){
8255          CA[tc].Jmax=tmult;
8256          }
8257        if(MOL[ta].s[tb].Cb!=-1){
8258          CB[tc].Kmax=tmult;
8259          }
8260        fclose(TSYSF);
8261 if(DEBUG>tdebugflag){
8262 fprintf(DBG,"string for MOL[%d].s[%d].f[%d].f is %s\n",\
8263     ta,tb,te,MOL[ta].s[tb].f[te].f);
8264 if(MOL[ta].s[tb].Ca!=-1){fprintf(DBG,"Jmax is %f, ",CA[tc].Jmax);
     }
8265 if(MOL[ta].s[tb].Cb!=-1){fprintf(DBG,"Kmax is %f, ",CB[tc].Kmax);
     }
8266 fflush(DBG);
8267 }
8268        sprintf(tsys,"wc -l %s > .%s.temporary", \
8269            MOL[ta].s[tb].f[te].f,PREF);
8270        system(tsys);
8271        TSYSF=fopen(tstring,"r");
8272        if(TSYSF==NULL){
8273          printf("fix tstring 3\n");
8274          exit(1);
8275          }
8276        fscanf(TSYSF,"%lf",&tmult);
8277        if(MOL[ta].s[tb].Ca!=-1){
8278          if(tmult!=(CA[tc].Jmax-CA[tc].Jmin+1)){
8279            printf("Jmax-Jmin range not wc -l\n");
8280            printf("This might cause trouble\n");
8281            }
8282          }
8283        if(MOL[ta].s[tb].Cb!=-1){
8284          if(tmult!=(CB[tc].Kmax-CB[tc].Kmin+1)){
8285            printf("Kmax-Kmin range not wc -l\n");
8286            printf("This might cause trouble\n");
8287            }
8288          }
8289        fclose(TSYSF);
8290 if(DEBUG>tdebugflag){
8291 fprintf(DBG,"tmult is %f\n",tmult);
8292 fflush(DBG);
8293 }
8294        } /* close read for rotation distribution file */
8295        } /* close loop over number of specified omegas -- or over the
        one set of vibrational states */
8296      } /* close states for loop */

      /* Preliminary stuff for loop over transitions for this molecule */
8297    sprintf(tstdum,"(not a state, top of transition list)");
8298    strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
8299    sscanf(tmolck,"%d",&MOL[ta].trans);
```

```
8300 if(DEBUG>tdebugflag){
8301 fprintf(DBG,"MOL[%d].trans is %d\n",ta,MOL[ta].trans);
8302 fflush(DBG);
8303 }
8304   MOL[ta].t=(Trans*)calloc(MOL[ta].trans,sizeof(Trans));
     /* Start loop over transitions for this molecule */
8305   for(tb=0;tb<MOL[ta].trans;tb++){
8306     sprintf(tstdum,"(not a state, transition number %d)",(tb+1));
8307     strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
8308     sscanf(tmolck,"%s",tdum);
     /* Separate the two states in the transition */
8309     tw=tv=0;
8310     while(tw==0){
8311       MOL[ta].t[tb].Nhi[tv]=tdum[tv];
8312       if(tdum[tv]=='-'){
8313         MOL[ta].t[tb].Nhi[tv]='\0';
8314         tw=1;
8315         }
8316       tv++;
8317       }
8318     tu=tv;
8319     tw=0;
8320     while(tw==0){
8321       MOL[ta].t[tb].Nlo[tv-tu]=tdum[tv];
8322       if(tdum[tv]=='\0') tw=1;
8323       tv++;
8324       }
     /* Find the position in the state array for each state in the transi-
        tion */
8325     tw=tv=tu=tt=0;
8326     while((tw==0)&&(tv<MOL[ta].states)){
8327       if(strcmp(MOL[ta].s[tv].Name,MOL[ta].t[tb].Nhi)==0){
8328         MOL[ta].t[tb].Hi=tv;
8329         MOL[ta].s[tv].no+=1;
8330 MOL[ta].s[tv].o = (int*) realloc(MOL[ta].s[tv].o,
     MOL[ta].s[tv].no*sizeof(int));
8331         MOL[ta].s[tv].o[MOL[ta].s[tv].no-1]=tb;
8332         tu=1;
8333 if(DEBUG>tdebugflag){
8334 fprintf(DBG,"MOL[%d].t[%d].Nhi %s matches
     ",ta,tb,MOL[ta].t[tb].Nhi);
8335 fprintf(DBG,"MOL[%d].s[%d].Name %s and
     ",ta,tv,MOL[ta].s[tv].Name);
8336 fprintf(DBG,"MOL[%d].t[%d].Hi is %d.\n",ta,tb,MOL[ta].t[tb].Hi);
8337 fprintf(DBG,"MOL[%d].s[%d].no is %d.\n",ta,tv,MOL[ta].s[tv].no);
8338 for(tf=0;tf<MOL[ta].s[tv].no;tf++){
8339 fprintf(DBG,"State %s is an origination state in
     ",MOL[ta].s[tv].Name);
8340 fprintf(DBG,"transition number %d.\n\n",tb);
8341   }
8342 fflush(DBG);
8343 }
8344         }
8345       if(strcmp(MOL[ta].s[tv].Name,MOL[ta].t[tb].Nlo)==0){
```

```
8346          MOL[ta].t[tb].Lo=tv;
8347          MOL[ta].s[tv].nd+=1;
8348 MOL[ta].s[tv].d = (int*) realloc(MOL[ta].s[tv].d,
     MOL[ta].s[tv].nd*sizeof(int));
8349          MOL[ta].s[tv].d[MOL[ta].s[tv].nd-1]=tb;
8350          tt=1;
8351 if(DEBUG>tdebugflag){
8352 fprintf(DBG,"MOL[%d].t[%d].Nlo %s matches
     ",ta,tb,MOL[ta].t[tb].Nlo);
8353 fprintf(DBG,"MOL[%d].s[%d].Name %s and
     ",ta,tv,MOL[ta].s[tv].Name);
8354 fprintf(DBG,"MOL[%d].t[%d].Lo is %d.\n",ta,tb,MOL[ta].t[tb].Lo);
8355 fprintf(DBG,"MOL[%d].s[%d].nd is %d.\n",ta,tv,MOL[ta].s[tv].nd);
8356 for(tf=0;tf<MOL[ta].s[tv].nd;tf++){
8357 fprintf(DBG,"State %s is an destination state in
     ",MOL[ta].s[tv].Name);
8358 fprintf(DBG,"transition number %d.\n\n",tb);
8359   }
8360 fflush(DBG);
8361 }
8362          }
8363        if((tu==1)&&(tt==1)){
8364          tw=1;
8365          }
8366        tv++;
8367        }
8368 if(DEBUG>tdebugflag){
8369 fprintf(DBG,"tdum is %s\n",tdum);
8370 fflush(DBG);
8371 }
8372      if(tw==0){
8373 printf("Missing definition for at least one state in transition
     %s.",tdum);
8374 printf(" Exiting.\nPlease edit transition file and restart pro-
     gram\n");
8375 fprintf(PAR,"Missing definition for at least one state in transi-
     tion %s.",tdum);
8376 fprintf(PAR," Exiting.\nPlease edit transition file and restart
     program.\n");
8377        exit(1);
8378        }
8379      if(MOL[ta].s[MOL[ta].t[tb].Hi].Ca!=-1){
8380        tw=CA[MOL[ta].s[MOL[ta].t[tb].Hi].Ca].O;
8381        MOL[ta].t[tb].Ohi=tw;
8382        if(tw==0){tw=1;}
8383        }
8384      else{
8385        tw=1;
8386        }
8387      if(MOL[ta].s[MOL[ta].t[tb].Lo].Ca!=-1){
8388        tv=CA[MOL[ta].s[MOL[ta].t[tb].Lo].Ca].O;
8389        MOL[ta].t[tb].Olo=tv;
8390        if(tv==0){tv=1;}
8391        }
```

```
8392     else{
8393       tv=1;
8394       }
8395     MOL[ta].t[tb].f=(fileset*)calloc((tw*tv),sizeof(fileset));
8396     MOL[ta].t[tb].P=(double*)calloc((tw*tv),sizeof(double));
8397     MOL[ta].t[tb].cP=(int*)calloc((tw*tv),sizeof(int));
8398     for(te=0;te<tw;te++){
8399     for(td=0;td<tv;td++){
8400       strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
8401       sscanf(tmolck,"%lf",&MOL[ta].t[tb].P[te*tv+td]);
8402       strcpy(tmolck,new_mol_ck(MOL[ta].Mol,tstdum));
8403       sscanf(tmolck,"%s",MOL[ta].t[tb].f[te*tv+td].f);
8404 if(DEBUG>tdebugflag){
8405 fprintf(DBG,"Prob is %f; file is %s\n",MOL[ta].t[tb].P[te*tv+td],\
8406     MOL[ta].t[tb].f[te*tv+td].f);
8407 fflush(DBG);
8408 }
8409 if((MOL[ta].t[tb].P[te*tv+td]==0) &&
     (MOL[ta].t[tb].f[te*tv+td].f!="NULL")){
8410 fprintf(PAR,"WARNING!!! FCF-FILE specified for zero transition ");
8411 fprintf(PAR,"probability for transtion %s-%s \n\tand specified
     Omegas ",\
8412     MOL[ta].t[tb].Nhi,MOL[ta].t[tb].Nlo);
8413 fprintf(PAR,"numbers %d (Hi) and %d (Lo).\n",(te+1),(td+1));
8414 fprintf(PAR,"\t(This is a non-fatal error, the program will con-
     tinue)\n");
8415   }
8416 if((MOL[ta].t[tb].P[te*tv+td]!=0) &&
     (MOL[ta].t[tb].f[te*tv+td].f=="NULL")){
8417 printf("FCF-FILE specified as NULL for non-zero transition ");
8418 printf("probability\nfor transtion %s-%s and specified Omegas ",\
8419     MOL[ta].t[tb].Nhi,MOL[ta].t[tb].Nlo);
8420 printf("numbers %d (Hi) and %d (Lo).\n",(te+1),(td+1));
8421 printf("Exiting. Please edit transition file and restart pro-
     gram.\n");
8422 fprintf(PAR,"FCF-FILE specified as NULL for non-zero transition ");
8423 fprintf(PAR,"probability\nfor transtion %s-%s and specified Omegas
     ",\
8424     MOL[ta].t[tb].Nhi,MOL[ta].t[tb].Nlo);
8425 fprintf(PAR,"numbers %d (Hi) and %d (Lo).\n",(te+1),(td+1));
8426 fprintf(PAR,"Exiting. Please edit transition file and restart pro-
     gram.\n");
8427   exit(1);
8428   }
8429     if(MOL[ta].t[tb].P[te*tv+td]!=0){
8430       TSYSF=fopen(MOL[ta].t[tb].f[te*tv+td].f,"r");
8431 if(DEBUG>tdebugflag){
8432 fprintf(DBG,"FCF file for MOL[%d].t[%d].f[%d].f is %s.\n\n",\
8433     ta,tb,(te*tv+td),MOL[ta].t[tb].f[te*tv+td].f);
8434 fflush(DBG);
8435   }
8436       if(TSYSF==NULL){
8437 printf("Error opening FCF file MOL[%d].t[%d].f[%d].f=%s. Exit-
     ing.\n\n",\
```

```
8438     ta,tb,(te*tv+td),MOL[ta].t[tb].f[te*tv+td].f);
8439         exit(1);
8440         }
8441       fclose(TSYSF);
8442       } /* close if trans. prob. not zero file check */
8443       } /* close td loop over omegas-in-transitions */
8444       } /* close te loop over omegas-in-transitions */
8445     } /* close transitions for loop */
8446   } /* end NUMMOL loop */
8447 fclose(INTR.F);
8448 return;
8449 }


      /*************** singlet_JK *****************/

      /* This function calculates energy levels and rotational distribu-
        tions (if needed) for all singlet states (Hund's Cases a and b) */
8450 void singlet_JK(int sM, int sS){

8451 int sa=0,sb=0,sJck=1,sg=0,sL=0,sp=0,sV=0,sVnum=0,sItst=0;
8452 int sCa=0,sCb=0,sC=0,sD=1,sf=0;
8453 double
     sT=0,sDv=0,sBe=0,sAe=0,sJMAX=0,sJMAXc=0,sJMAXf=0,saA=0,sJcut=0;
8454 double
     swe=0,swexe=0,sI=0,sInt=0,sImax=0,sOmm=0,sbeta=0,sFv=0,sBvDv=0;
8455 double
     sIs=0,sIa=0,sTe=0,sDe=0,sBv=0,sTeVib=0,sJDmin=0,sJcountnum=0;
8456 double sbJ=0;
8457 char sfnstr[1000];
8458 FILE *SROT;

8459 if(DEBUG>sdebugflag){
8460 fprintf(DBG,"\n\nTop of singlet_JK. Molecule %s; State %s.\n\n",\
8461     MOL[sM].Mol,MOL[sM].s[sS].Name);
8462 fflush(DBG);
8463 }
      /* Find lowest-level temperature designation */
8464 if(MOL[sM].s[sS].T[0]!=0){
8465   sT=MOL[sM].s[sS].T[0];
8466 if(DEBUG>sdebugflag){
8467 fprintf(DBG,"Temperature defined at state level: %f",sT);
8468 fflush(DBG);
8469 }
8470   }
8471 else{
8472   if(MOL[sM].T!=0){
8473     sT=MOL[sM].T;
8474 if(DEBUG>sdebugflag){
8475 fprintf(DBG,"Temperature defined at molecule level: %f",sT);
8476 fflush(DBG);
8477 }
8478     }
8479   else{
8480     sT=TEMP;
```

```
8481 if(DEBUG>sdebugflag){
8482 fprintf(DBG,"Temperature defined at global level: %f",sT);
8483 fflush(DBG);
8484 }
8485     }
8486   }
8487 if(sT==0){
8488   fprintf(PAR,"WARNING!! temperature defined as zero for state ");
8489   fprintf(PAR,"%s of molecule
     %s.\n\n",MOL[sM].s[sS].Name,MOL[sM].Mol);
8490   }

      /* get the necessary info -- the sOmm variable is there because for
         singlet states, the only difference in the energy functions I'm
         using here is that for Case a, Lambda occurs in Fv(J). So, I will
         always put Lambda in the energy term -- it just gets multiplied by
         zero if the case is b. */
8491 if(MOL[sM].s[sS].Ca>-1){ /* if this is Hund's Case a */
8492   sC=0;
8493   sCa=MOL[sM].s[sS].Ca;
8494   sBe=CA[sCa].Be;
8495   sAe=CA[sCa].ae;
8496   sTe=CA[sCa].Te;
8497   swe=CA[sCa].we;
8498   swexe=CA[sCa].wexe;
8499   sg=CA[sCa].g;
8500   sI=CA[sCa].I;
8501   sL=CA[sCa].L;
      /* This is the place to add in scan for the constant beta */
8502   sOmm=1;
8503 if(DEBUG>sdebugflag){
8504 fprintf(DBG,"State is Case (a).\n");
8505 fprintf(DBG,"Be=%f; ae=%f; Te=%f; we=%f; wexe=%f;
     \n",sBe,sAe,sTe,swe,swexe);
8506 fprintf(DBG,"sg=%d; sI=%f; sL=%d; sOmm=%f ",sg,sI,sL,sOmm);
8507 fflush(DBG);
8508 }
8509   }
8510 if(MOL[sM].s[sS].Cb>-1){ /* if this is Hund's Case b */
8511   sC=1;
8512   sCb=MOL[sM].s[sS].Cb;
8513   sBe=CB[sCb].Be;
8514   sAe=CB[sCb].ae;
8515   sTe=CB[sCb].Te;
8516   swe=CB[sCb].we;
8517   swexe=CB[sCb].wexe;
8518   sg=CB[sCb].g;
8519   sI=CB[sCb].I;
8520   sL=CB[sCb].L;
8521   sp=CB[sCb].p;
8522    /* add in scan for beta if that gets included */
8523   sOmm=0;
8524 if(DEBUG>sdebugflag){
8525 fprintf(DBG,"State is Case (b).\n");
```

```
8526 fprintf(DBG,"Be=%f; ae=%f; Te=%f; we=%f; wexe=%f;
     \n",sBe,sAe,sTe,swe,swexe);
8527 fprintf(DBG,"sg=%d; sI=%f; sL=%d; sp=%d; sOmm=%f
     ",sg,sI,sL,sp,sOmm);
8528 fflush(DBG);
8529 }
8530   }

      /* Calculate some stuff that will be useful later */
8531 if(swe!=0) sDe=4*pow(sBe,3)/(swe*swe);  /* Centrifugal distortion
        */
8532 if(DEBUG>sdebugflag){
8533 fprintf(DBG,"sDe is %f;\n",sDe);
8534 fflush(DBG);
8535 }
8536 if(sg!=0){
8537   sIs=1;  /* Equivalent to: sIs=(2*sI+1)*(sI+1); */
8538   sIa=sI/(sI+1);  /* Equiv. to: sIa=(2*sI+1)*sI; */
8539 if(DEBUG>sdebugflag){
8540 fprintf(DBG,"sIs=%f; sIa=%f; \n",sIs,sIa);
8541 fflush(DBG);
8542 }
8543   }
8544 if(sL==0){ /* if this is a Sigma state */
8545   fprintf(PAR,"STATE %s is a Sigma",MOL[sM].s[sS].Name);
8546 if(DEBUG>sdebugflag){
8547 fprintf(DBG,"STATE %s is a Sigma",MOL[sM].s[sS].Name);
8548 fflush(DBG);
8549 }
8550   if(sg!=0){ /* and this is a homonuclear molecule */
8551     if(sp==+1){ /* and this is a Sigma+ state */
8552       fprintf(PAR,"+ ");
8553 if(DEBUG>sdebugflag){
8554 fprintf(DBG,"+ ");
8555 fflush(DBG);
8556 }
8557       if(sg==+1){ /* and the state is gerade */
8558       fprintf(PAR,"gerade state and even K's are (+), ");
8559 if(DEBUG>sdebugflag){
8560 fprintf(DBG,"gerade state and even K's are (+), ");
8561 fflush(DBG);
8562 }
8563         MOL[sM].s[sS].pmsymm=+1;
8564         if(fmod(sI,1.0)==0){
8565           MOL[sM].s[sS].Isymm=+1;
8566           fprintf(PAR,"s (boson) \n");
8567 if(DEBUG>sdebugflag){
8568 fprintf(DBG,"s (boson)\n");
8569 fflush(DBG);
8570 }
8571           }
8572         if(fmod(sI,1.0)==0.5){
8573           MOL[sM].s[sS].Isymm=-1;
8574           fprintf(PAR,"a (fermion) \n");
```

```
8575 if(DEBUG>sdebugflag){
8576 fprintf(DBG,"a (fermion)\n");
8577 fflush(DBG);
8578 }
8579           }
8580         }
8581       else{ /* and the state is ungerade */
8582       fprintf(PAR,"ungerade state and even K's are (+), ");
8583 if(DEBUG>sdebugflag){
8584 fprintf(DBG,"ungerade state and even K's are (+), ");
8585 fflush(DBG);
8586 }
8587         MOL[sM].s[sS].pmsymm=+1;
8588         if(fmod(sI,1.0)==0){
8589           MOL[sM].s[sS].Isymm=-1;
8590           fprintf(PAR,"a (boson) \n");
8591 if(DEBUG>sdebugflag){
8592 fprintf(DBG,"a (boson) \n");
8593 fflush(DBG);
8594 }
8595           }
8596         if(fmod(sI,1.0)==0.5){
8597           MOL[sM].s[sS].Isymm=+1;
8598           fprintf(PAR,"s (fermion) \n");
8599 if(DEBUG>sdebugflag){
8600 fprintf(DBG,"s (fermion)\n");
8601 fflush(DBG);
8602 }
8603           }
8604         }
8605       }
8606     else{ /* and this is a Sigma- state */
8607       fprintf(PAR,"- ");
8608 if(DEBUG>sdebugflag){
8609 fprintf(DBG,"- ");
8610 fflush(DBG);
8611 }
8612       if(sg==+1){ /* and the state is gerade */
8613       fprintf(PAR,"gerade state and even K's are (-), ");
8614 if(DEBUG>sdebugflag){
8615 fprintf(DBG,"gerade state and even K's are (-), ");
8616 fflush(DBG);
8617 }
8618         MOL[sM].s[sS].pmsymm=-1;
8619         if(fmod(sI,1.0)==0){
8620           MOL[sM].s[sS].Isymm=-1;
8621           fprintf(PAR,"a (boson) \n");
8622 if(DEBUG>sdebugflag){
8623 fprintf(DBG,"a (boson) \n");
8624 fflush(DBG);
8625 }
8626           }
8627         if(fmod(sI,1.0)==0.5){
8628           MOL[sM].s[sS].Isymm=+1;
```

```
8629           fprintf(PAR,"s (fermion) \n");
8630 if(DEBUG>sdebugflag){
8631 fprintf(DBG,"s (fermion)\n");
8632 fflush(DBG);
8633 }
8634           }
8635         }
8636       else{ /* and the state is ungerade */
8637       fprintf(PAR,"ungerade state and even K's are (-), ");
8638 if(DEBUG>sdebugflag){
8639 fprintf(DBG,"ungerade state and even K's are (-), ");
8640 fflush(DBG);
8641 }
8642         MOL[sM].s[sS].pmsymm=-1;
8643         if(fmod(sI,1.0)==0){
8644           MOL[sM].s[sS].Isymm=+1;
8645           fprintf(PAR,"s (boson) \n");
8646 if(DEBUG>sdebugflag){
8647 fprintf(DBG,"s (boson)\n");
8648 fflush(DBG);
8649 }
8650           }
8651         if(fmod(sI,1.0)==0.5){
8652           MOL[sM].s[sS].Isymm=-1;
8653           fprintf(PAR,"a (fermion) \n");
8654 if(DEBUG>sdebugflag){
8655 fprintf(DBG,"a (fermion)\n");
8656 fflush(DBG);
8657 }
8658           }
8659         }
8660       }
8661     }
8662   else{ /* and this a heteronuclear molecule */
8663     if(sp==+1){
8664       fprintf(PAR,"+ state and even K's are (+)\n");
8665       MOL[sM].s[sS].pmsymm=+1;  /* and a Sigma+ state */
8666 if(DEBUG>sdebugflag){
8667 fprintf(DBG,"+ state and even K's are (+)\n");
8668 fflush(DBG);
8669 }
8670       }
8671     else{
8672       fprintf(PAR,"- state and even K's are (-)\n");
8673       MOL[sM].s[sS].pmsymm=-1;  /* and a Sigma- state */
8674 if(DEBUG>sdebugflag){
8675 fprintf(DBG,"- state and even K's are (-)\n");
8676 fflush(DBG);
8677 }
8678       }
8679     }
8680   }
8681 else{ /* if not a Sigma state, print info to the parameter file */
8682   fprintf(PAR,"STATE %s is a ",MOL[sM].s[sS].Name);
```

```
8683 if(DEBUG>sdebugflag){
8684 fprintf(DBG,"STATE %s is a ",MOL[sM].s[sS].Name);
8685 fflush(DBG);
8686 }
8687   switch(sL){
8688     case 1:
8689 if(DEBUG>sdebugflag){
8690 fprintf(DBG,"Pi state\n ");
8691 fflush(DBG);
8692 }
8693       fprintf(PAR,"Pi state\n");
8694       break;
8695     case 2:
8696 if(DEBUG>sdebugflag){
8697 fprintf(DBG,"Delta state\n ");
8698 fflush(DBG);
8699 }
8700       fprintf(PAR,"Delta state\n");
8701       break;
8702     case 3:
8703 if(DEBUG>sdebugflag){
8704 fprintf(DBG,"Phi state\n ");
8705 fflush(DBG);
8706 }
8707       fprintf(PAR,"Phi state\n");
8708       break;
8709     default:
8710 fprintf(PAR,"state with Lambda>3 -- an exotic state\n");
8711 fprintf(PAR,"Interpret the results from this program with
     care.\n");
8712       break;
8713     }
8714   }

8715 if(MOL[sM].s[sS].Dist!=-1){
     /* Estimate J/K max. This comes from taking the derivative of the
        population distribution and setting it equal to zero. */
8716   sBv=sBe-sAe/2;
8717   saA=sBv/(kB*sT);
8718   sJMAX=1/sqrt(2*saA) - 0.5;
8719   sJMAXc=ceil(sJMAX);
8720   sJMAXf=floor(sJMAX);
     /* Note the Omega(=Lambda for singlet states) in the next calcula-
        tion. The prefactor, called sOmm, is set to zero for Case b states
        and to one for Case a states. So, the energy should be calculated
        properly for either state */
8721   saA=(sBv*(sJMAXc*(sJMAXc+1)-sOmm*sL*sL))/(kB*sT);
8722   sImax=(2*sJMAXc+1)*exp(-saA);
8723   saA=(sBv*(sJMAXf*(sJMAXf+1)-sOmm*sL*sL))/(kB*sT);
8724   sInt=(2*sJMAXf+1)*exp(-saA);
8725 if(DEBUG>sdebugflag){
8726 fprintf(DBG,"sBv=%f; saA=%f; sJMAX=%f; sJMAXc=%f; aJMAXf=%f\n",\
8727     sBv,saA,sJMAX,sJMAXc,sJMAXf);
8728 fprintf(DBG,"sImax is %f; sInt is %f --- ",sImax,sInt);
```

```
8729 fflush(DBG);
8730 }
8731   if((sImax)<(sInt)){sJMAX=sJMAXf;}
8732   else{sJMAX=sJMAXc;}
8733 if(DEBUG>sdebugflag){
8734 fprintf(DBG,"sJMAX is now %f\n ",sJMAX);
8735 fflush(DBG);
8736 }

      /* Find an upper limit for J/K corresponding to the user specifica-
         tion. To find out where this equation comes from, see the docu-
         mentation, particularly the documentation found in files or di-
         rectories with the word "trick" in the name. Note that this trick
         is only good down to a cutoff intensity of about 1/10,000 of the
         maximum value. */
8737   saA=sBv/(kB*sT);
8738   sJcut=sqrt((-2/saA)/(JKm + JKb/(log(JKCUT)))) + 1/(2*saA) - 0.5;
8739   sJcut=ceil(sJcut);
8740 if(DEBUG>sdebugflag){
8741 fprintf(DBG,"JKCUT is %f; sJcut is %f\n",JKCUT,sJcut);
8742 fflush(DBG);
8743 }
      /* check this number and chastise programmer if not good... Also, if
         not good, find a better number using a less elegant method. */
8744   saA=(sBv*(sJcut*(sJcut+1)-sOmm*sL*sL))/(kB*sT);
8745   sInt=(2*sJcut+1)*exp(-saA);
8746   saA=(sBv*(sJMAX*(sJMAX+1)-sOmm*sL*sL))/(kB*sT);
8747   sImax=(2*sJMAX+1)*exp(-saA);
8748 if(DEBUG>sdebugflag){
8749 fprintf(DBG,"sInt=%f; sImax=%f \n ",sInt,sImax);
8750 fflush(DBG);
8751 }
8752   if((sInt/sImax)>JKCUT){
8753     if(JKCUT>0){ /* put back to 0.0001 */
8754       printf("Dammit... Go fix the J/K business... \n");
8755       printf("The calculated sJcut is %5.4f ",sJcut);
8756       }
8757     sJck=1;
8758     while(sJck==1){
8759       sJcut++;
8760       saA=(sBv*(sJcut*(sJcut+1)-sOmm*sL*sL))/(kB*sT);
8761       sInt=(2*sJcut+1)*exp(-saA);
8762       if((sInt/sImax)<JKCUT){
8763         sJck=0;
8764         }
8765       } /*vvvvvvvvv put back to 0.0001 vvvvvvvv*/
8766     if(JKCUT>0) printf("The refined one is %5.4f \n",sJcut);
8767     }
8768   }

      /* There can't be more than one omega here -- check for the distribu-
         tion to be defined in a file. */
8769 if(MOL[sM].s[sS].Dist==-1){
8770   sD=0;
```

```
8771   if(sC==0){
8772     sJcut=CA[sCa].Jmax;
8773     sJDmin=CA[sCa].Jmin;
8774     }
8775   if(sC==1){
8776     sJcut=CB[sCb].Kmax;
8777     sJDmin=CB[sCb].Kmin;
8778     }
8779 if(DEBUG>sdebugflag){
8780 fprintf(DBG,"Distribution is by file.\n ");
8781 fflush(DBG);
8782 }
8783   }
     /* Allocate memory for J/K array. Assign s/a and +/- flags. */
8784 sVnum=MOL[sM].s[sS].v[0].vnum;  /* in other functions, where there
       might be multiple Omegas, this is vnum*numomega (so far as memory
       allocation goes) */
8785 MOL[sM].s[sS].nV=sVnum;
8786 MOL[sM].s[sS].nr=sVnum;
8787 MOL[sM].s[sS].r=(rotset*)calloc(sVnum,sizeof(rotset));
8788 MOL[sM].s[sS].rc=(rotset*)calloc(30,sizeof(rotset));
8789 for(sa=0;sa<sVnum;sa++){
8790   MOL[sM].s[sS].r[sa].k=sJcut+1;
8791   MOL[sM].s[sS].r[sa].j=sJcut+1;
8792
    MOL[sM].s[sS].r[sa].J=(double*)calloc((sJcut+1),sizeof(double));
8793   MOL[sM].s[sS].r[sa].EJ = (double*) calloc((sJcut+1),
    sizeof(double));
8794   MOL[sM].s[sS].r[sa].PJ = (double*) calloc((sJcut+1),
    sizeof(double));
8795   MOL[sM].s[sS].r[sa].CJ = (double*) calloc((sJcut+1),
    sizeof(double));
     /* I'm not allocating for K, even for Case b, because for singlet
       Case b states J=K */
8796   }
     /* Loop through J/K values, calculating energies and populations
       (include nuclear effects if applicable). Only one energy calcu-
       lation is needed here since the energy equation I'm using here is
       the same for singlet states of Hund's Cases a and b.*/
8797 for(sa=0;sa<sVnum;sa++){
8798   sV=MOL[sM].s[sS].v[0].vlo+sa;
8799   sImax=0;
8800   sTeVib=sTe+(sV+0.5)*swe-(sV+0.5)*(sV+0.5)*swexe;
8801   if(swe!=0){
8802     sBv=sBe-sAe*(sV+0.5);
8803     sDv=sDe+sbeta*(sV+0.5);
8804     sBvDv=sBv/(2*sDv);
8805     MOL[sM].s[sS].r[sa].Jdissoc=floor((sqrt(1+4*sBvDv)-1)/2);
8806     MOL[sM].s[sS].r[sa].Kdissoc=MOL[sM].s[sS].r[sa].Jdissoc;
8807     }
8808   else{
8809     MOL[sM].s[sS].r[sa].Jdissoc=RAND_MAX;
8810     MOL[sM].s[sS].r[sa].Kdissoc=RAND_MAX;
8811     }
```

```
8812 if(DEBUG>sdebugflag){
8813 fprintf(DBG,"sVnum is %d; sTeVib=%f \n",sVnum,sTeVib);
8814 fprintf(DBG,"sBv=%12.6e, sDv=%12.6e, sBvDv=%12.6e,
     ",sBv,sDv,sBvDv);
8815 fprintf(DBG,"MOL[%d].s[%d].r[%d].Kdissoc=%12.6e\n",sM,sS,sa,\
8816     MOL[sM].s[sS].r[sa].Kdissoc);
8817 fflush(DBG);
8818 }
8819   if(MOL[sM].s[sS].r[sa].Jdissoc>((double)sJcut)){
8820     sJcountnum=sJcut+1;
8821     }
8822   else{
8823     sJcountnum=(int)(MOL[sM].s[sS].r[sa].Jdissoc);
8824 fprintf(PAR,"\nWARNING!! MOLECULE DISSOCIATION (see be-
     low)\nAccording to");
8825 fprintf(PAR,"the spectroscopic constants in the state file,\n
     state");
8826 fprintf(PAR,"%s of Molecule %s in vibration level %d dissoci-
     ates\n",\
8827     MOL[sM].s[sS].Name,MOL[sM].Mol,sa);
8828 fprintf(PAR,"at rotational level
     J=%.1f,",MOL[sM].s[sS].r[sa].Jdissoc);
8829 fprintf(PAR," which is significantly populated at temperature
     %.1f.\n",sT);
8830 fprintf(PAR,"Interpret results from this simulation carefully\n");
8831     }
8832 if(DEBUG>sdebugflag){
8833 fprintf(DBG,"sVnum=%d; sTeVib=%f sBv=%12.6e, sDv=%12.6e\n",\
8834     sVnum,sTeVib,sBv,sDv);
8835 fflush(DBG);
8836 }
8837   for(sb=0;sb<sJcountnum;sb++){
8838     sbJ=sb+sOmm*sL;
8839     MOL[sM].s[sS].r[sa].J[sb]=sbJ;
8840
     sFv=(sBv*(sbJ*(sbJ+1)-sOmm*sL*sL))-sDv*sbJ*sbJ*(sbJ+1)*(sbJ+1);
8841     MOL[sM].s[sS].r[sa].EJ[sb]=sFv;
8842     if(sD!=0){
8843       sInt=(2*sbJ+1)*exp(-sFv/(kB*sT));
8844 if(DEBUG>sdebugflag){
8845 fprintf(DBG,"sb is %d; sFv=%f; sInt=%f\n",sb,sFv,sInt);
8846 fflush(DBG);
8847 }
8848         }
8849     if((sg!=0)&&(sD!=0)){
      /* Note the following bit of trickery. It works even if the state in
        question has Lambda>0 and doesn't need to have alternating spin-
        stats. The reason for this is that the value of Isymm is zero for
        those states. If Isymm is -1, even K's are a. If Isymm is +1, even
        K's are s. So, there are six possibilities:

      sb%2--> 0 (K is even) 1 (K is odd)

      \Isymm/
```

```
      -1 -1 (a) 0 (s)

      +1 +1 (s) 2 (a)

      0 0 (s-equiv.) 1 (s-equiv)

      The factor for the a state is relative to the weight of the s state
        (weight with s = 1). So, only when the result of the calculation
        is -1 or 2 does sIa need to be multiplied. (but I'm lazy so the
        switch still has all the cases in it...) */
8850        sItst=sb%2 + MOL[sM].s[sS].Isymm;
8851 if(DEBUG>sdebugflag){
8852 fprintf(DBG,"sg is 0\n");
8853 fflush(DBG);
8854 }
8855       switch(sItst){
8856         case 0:
8857           sInt*=sIs;
8858 if(DEBUG>sdebugflag){
8859 fprintf(DBG,"sInt=%f",sInt);
8860 fflush(DBG);
8861 }
8862           break;
8863         case 1:
8864           sInt*=sIs;
8865 if(DEBUG>sdebugflag){
8866 fprintf(DBG,"sInt=%f",sInt);
8867 fflush(DBG);
8868 }
8869           break;
8870         case -1:
8871           sInt*=sIa;
8872 if(DEBUG>sdebugflag){
8873 fprintf(DBG,"sInt=%f",sInt);
8874 fflush(DBG);
8875 }
8876           break;
8877         case 2:
8878           sInt*=sIa;
8879 if(DEBUG>sdebugflag){
8880 fprintf(DBG,"sInt=%f",sInt);
8881 fflush(DBG);
8882 }
8883           break;
8884         default:
8885 printf("problem with nuclear stats in singlet_JK. Exiting. \n");
8886           exit(1);
8887         }
8888       }
8889     if((sD==0)&&(sb>=sJDmin)){
8890       if(sC==0){
8891         sf=(CA[sCa].Jmax-CA[sCa].Jmin+1)*sa+sb-sJDmin;
8892         MOL[sM].s[sS].r[sa].PJ[sb]=\
8893           CA[sCa].Jpop[sf];
```

```
8894          }
8895        if(sC==1){
8896          sf=(CB[sCb].Jnum)*sa+sb-sJDmin;
8897          MOL[sM].s[sS].r[sa].PJ[sb]=\
8898            CB[sCb].Jpop[sf];
8899          }
8900        sInt=MOL[sM].s[sS].r[sa].PJ[sb];
8901 if(DEBUG>sdebugflag){
8902 fprintf(DBG,"sb is %d; sFv=%f; sInt=%f\n",sb,sFv,sInt);
8903 fflush(DBG);
8904 }
8905        }
8906      if(sD!=0) MOL[sM].s[sS].r[sa].PJ[sb]=sInt;
8907      sImax+=sInt;
8908 if(DEBUG>sdebugflag){
8909 fprintf(DBG,"\n sImax is %f\n\n",sImax);
8910 fflush(DBG);
8911 }
8912      }
8913    for(sb=0;sb<sJcountnum;sb++){
8914      MOL[sM].s[sS].r[sa].EJ[sb]+=sTeVib;
8915      MOL[sM].s[sS].r[sa].PJ[sb]/=sImax;
8916 if(DEBUG>sdebugflag){
8917 fprintf(DBG,"%d\t%f\t%f\n",sb,MOL[sM].s[sS].r[sa].EJ[sb],\
8918      MOL[sM].s[sS].r[sa].PJ[sb]);
8919 fflush(DBG);
8920 }
8921      }
8922 if(DEBUG>sdebugflag){
8923 fprintf(DBG,"\n out of for loop \n");
8924 fflush(DBG);
8925 }
8926    }
8927 sprintf(sfnstr,"mkdir %s_molecules/%s",PREF,MOL[sM].Mol);
8928 system(sfnstr);
8929 sprintf(sfnstr,"%s_molecules/%s/%s_%s_rot.dat",PREF,MOL[sM].Mol,\
8930      MOL[sM].Mol,MOL[sM].s[sS].Name);
8931 SROT=fopen(sfnstr,"w");
8932 if(SROT==NULL){
8933    printf("Error opening output file (singlet_JK) %s. Exiting.\n",\
8934        sfnstr);
8935    exit(1);
8936    }
8937 fprintf(SROT,"## File generated by program %s.\n",PROGRAM_NAME);
8938 fprintf(SROT,"## This file contains ");
8939 fprintf(SROT,"rotational state information about\n");
8940 fprintf(SROT,"## state %s of molecule
     %s.\n",MOL[sM].s[sS].Name,MOL[sM].Mol);
8941 fprintf(SROT,"## The columns are, in order:\n## J");
8942 for(sa=0;sa<sVnum;sa++){
8943    sV=MOL[sM].s[sS].v[0].vlo+sa;
8944    fprintf(SROT,"\tF(v=%d)\tPop(v=%d)",sV,sV);
8945    }
8946 fprintf(SROT,"\n");
```

```
8947 for(sb=0;sb<sJcountnum;sb++){
8948   fprintf(SROT,"%d",sb);
8949   for(sa=0;sa<sVnum;sa++){
8950     fprintf(SROT,"\t%18.10e\t%18.10e",\
8951       MOL[sM].s[sS].r[sa].EJ[sb],\
8952       MOL[sM].s[sS].r[sa].PJ[sb]);
8953     }
8954   fprintf(SROT,"\n");
8955   }
8956 fflush(SROT);
8957 fclose(SROT);
8958 if(DEBUG>sdebugflag){
8959 fprintf(DBG,"\n out of sa(Vnum) loop. sa=%d\n",sa);
8960 fflush(DBG);
8961 }
8962 return;
8963 }


      /*************** read_XY_file *****************/

      /* This function can be used to read in any file whose contents con-
        sist solely of two columns of numbers. */
8964 void read_XY_file(XYlist *XY){

8965 int XYa=0;
8966 char XYsys[500];
8967 FILE *XYF;

8968 sprintf(XYsys,"wc -l %s > %s",XY[0].f.f,TMPFILE);
8969 system(XYsys);
8970 if(DEBUG>XYdebugflag){
8971 fprintf(DBG,"XYsys is %s.\n",XYsys);
8972 fflush(DBG);
8973 }
8974 SYS=fopen(TMPFILE,"r");
8975 if(SYS==NULL){
8976 printf("Error opening system temporary file. Exiting.\n");
8977 exit(1);
8978 }
8979 fscanf(SYS,"%d",&XY[0].n);
8980 fclose(SYS);

8981 if(DEBUG>XYdebugflag){
8982 fprintf(DBG,"Here 1\n");
8983 fflush(DBG);
8984 }
8985 XY[0].x=(double*)calloc(XY[0].n,sizeof(double));
8986 if(DEBUG>XYdebugflag){
8987 fprintf(DBG,"Here 2\n");
8988 fflush(DBG);
8989 }
8990 XY[0].y=(double*)calloc(XY[0].n,sizeof(double));
8991 if(DEBUG>XYdebugflag){
8992 fprintf(DBG,"Here 3\n");
```

```
8993 fflush(DBG);
8994 }
8995 XYF=fopen(XY[0].f.f,"r");
8996 if(XYF==NULL){
8997 printf("Error opening XY input file %s. Exiting.\n",XY[0].f.f);
8998 exit(1);
8999 }
9000 if(DEBUG>XYdebugflag){
9001 fprintf(DBG,"Here 4\n");
9002 fflush(DBG);
9003 }
9004 for(XYa=0;XYa<XY[0].n;XYa++){
9005   fscanf(XYF,"%lf",&XY[0].x[XYa]);
9006 if(DEBUG>XYdebugflag){
9007 fprintf(DBG,"X is %f ",XY[0].x[XYa]);
9008 fflush(DBG);
9009 }
9010   fscanf(XYF,"%lf",&XY[0].y[XYa]);
9011 if(DEBUG>XYdebugflag){
9012 fprintf(DBG,"Y is %f\n",XY[0].y[XYa]);
9013 fflush(DBG);
9014 }
9015   }
9016 fclose(XYF);
9017 return;
9018 }
```

APPENDIX D

MASS SPECTRA

These are the mass spectra, where available, taken prior to each run.

Figure 4-1.10 a

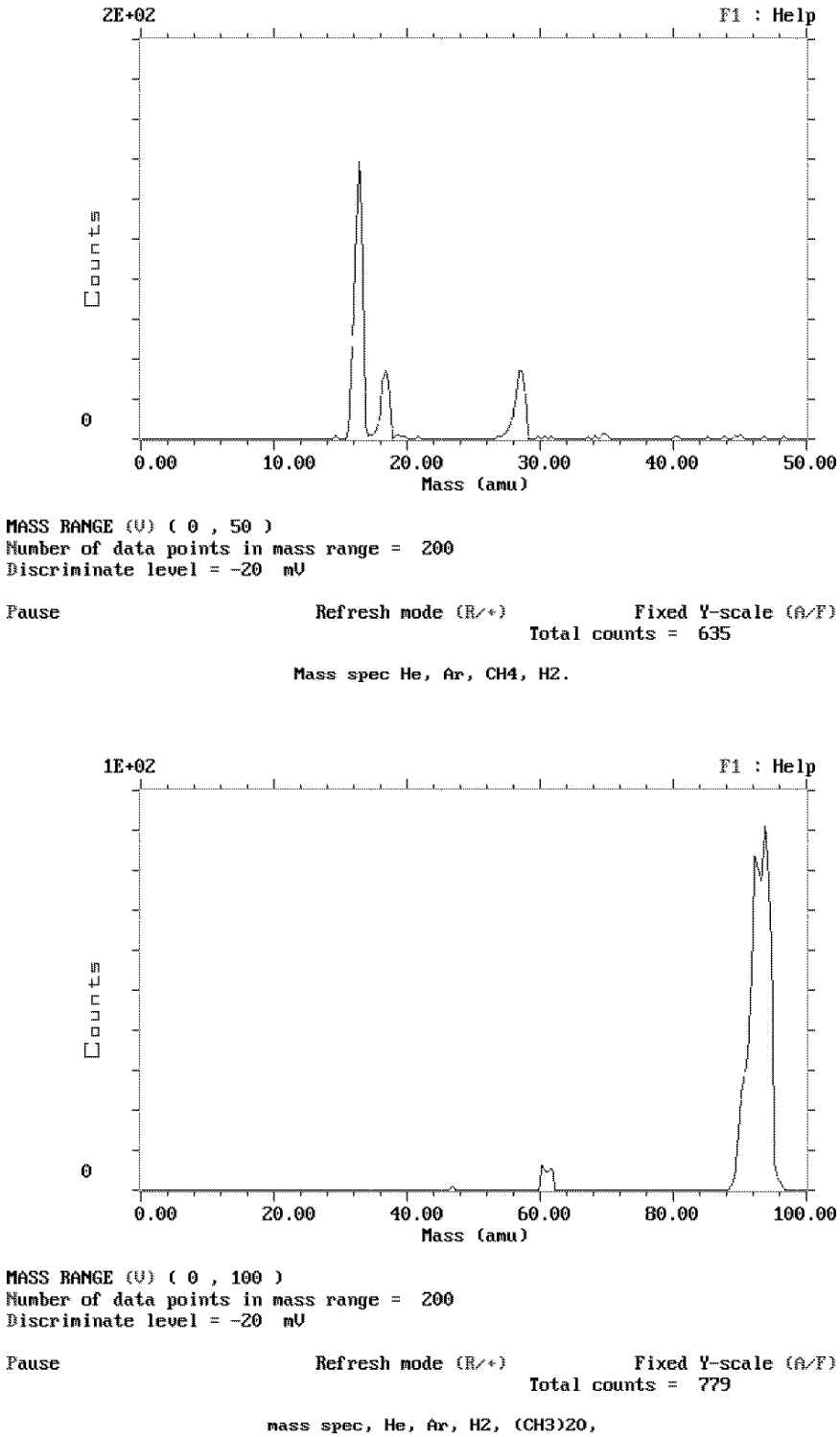


Figure 4-1.10 a: Pre-run mass spectra for CH_5^+ (upper) and $(CH_3)_2OH^+$ (lower).

Figure 4-1.10 b

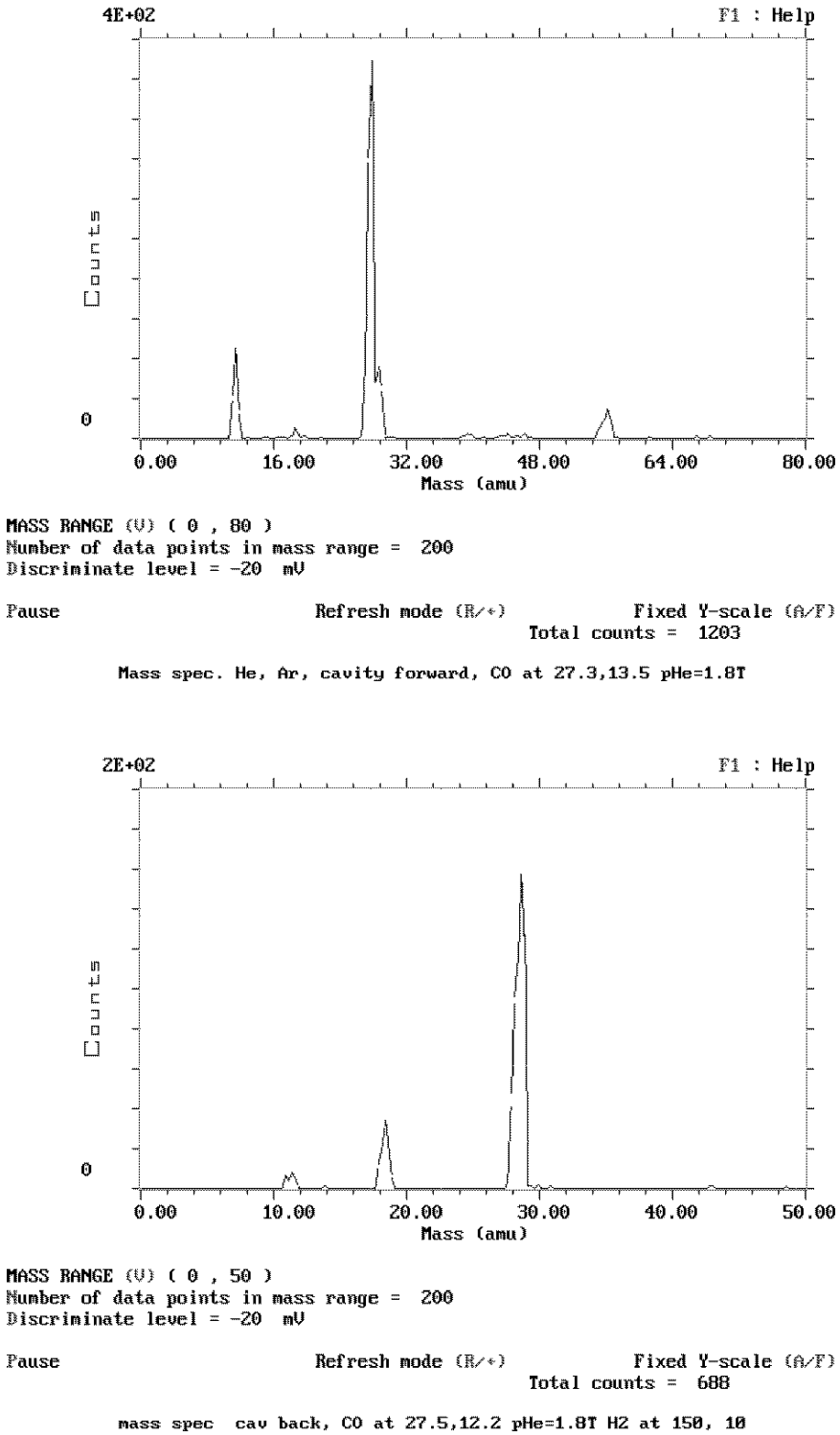


Figure 4-1.10 a: Pre-run mass spectra for CO^+ (upper) and HCO^+ (lower).

Figure 4-1.10 c

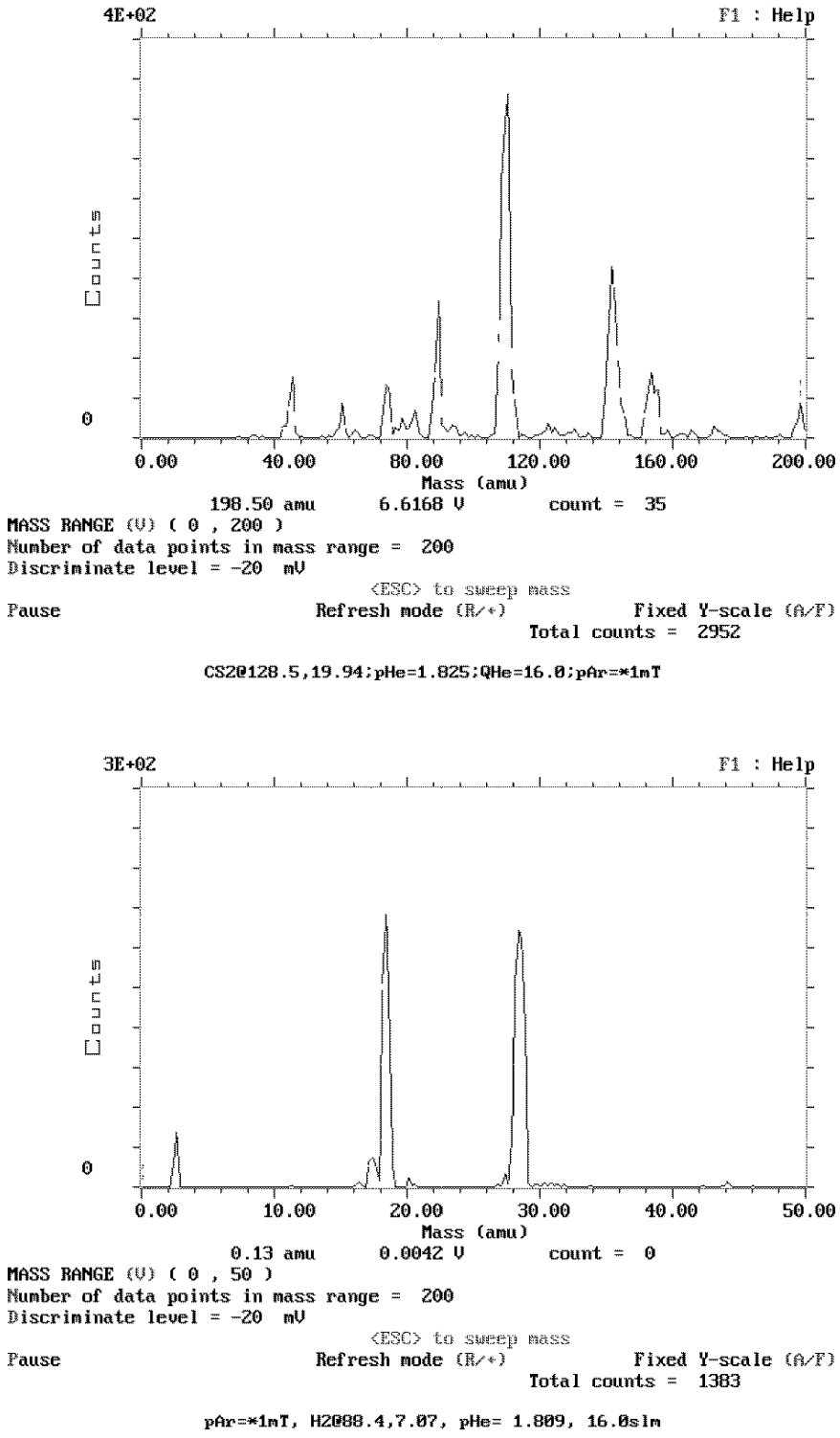
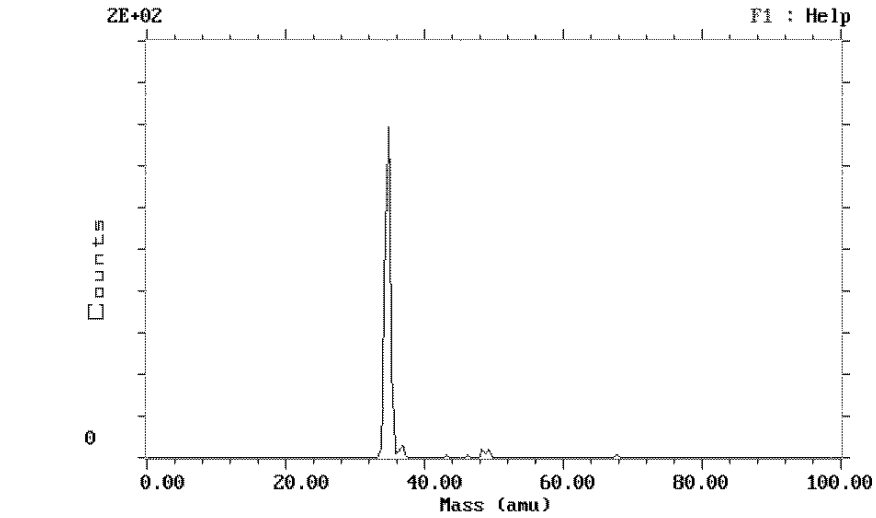


Figure 4-1.10 a: Pre-run mass spectra for CS_2^+ (upper) and H_3^+ (lower).

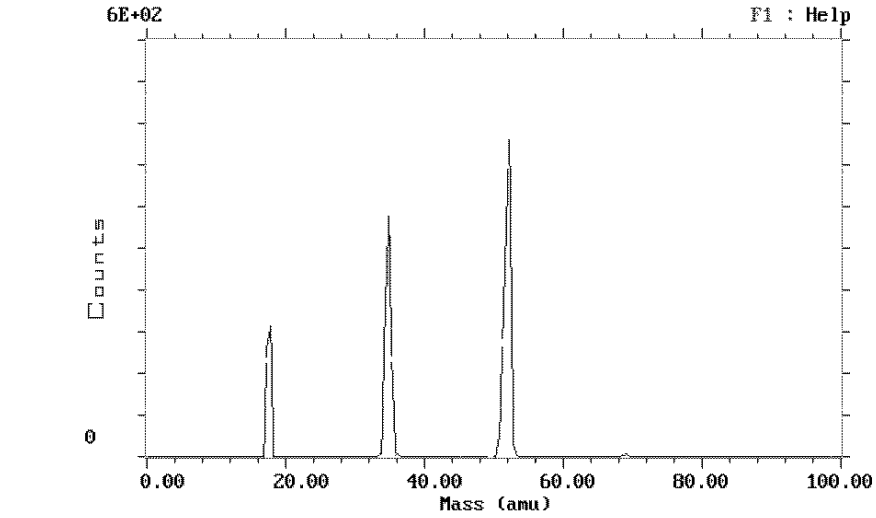
Figure 4-1.10 d



MASS RANGE (U) (0 , 100)
Number of data points in mass range = 200
Discriminate level = -20 mV

Pause Refresh mode (R/+) Auto Y-scale (A/F)
Total counts = 319

Mass spec of He, Ar, 16.0 slm pHe = 1.816 H2 at 150,10; H2S at 20,10



MASS RANGE (U) (0 , 100)
Number of data points in mass range = 200
Discriminate level = -20 mV

Pause Refresh mode (R/+) Fixed Y-scale (A/F)
Total counts = 2024

mass spec He, Ar, NH3, H2.

Figure 4-1.10 a: Pre-run mass spectra for H_3S^+ (upper) and NH_3^+ (lower).

Figure 4-1.10 e

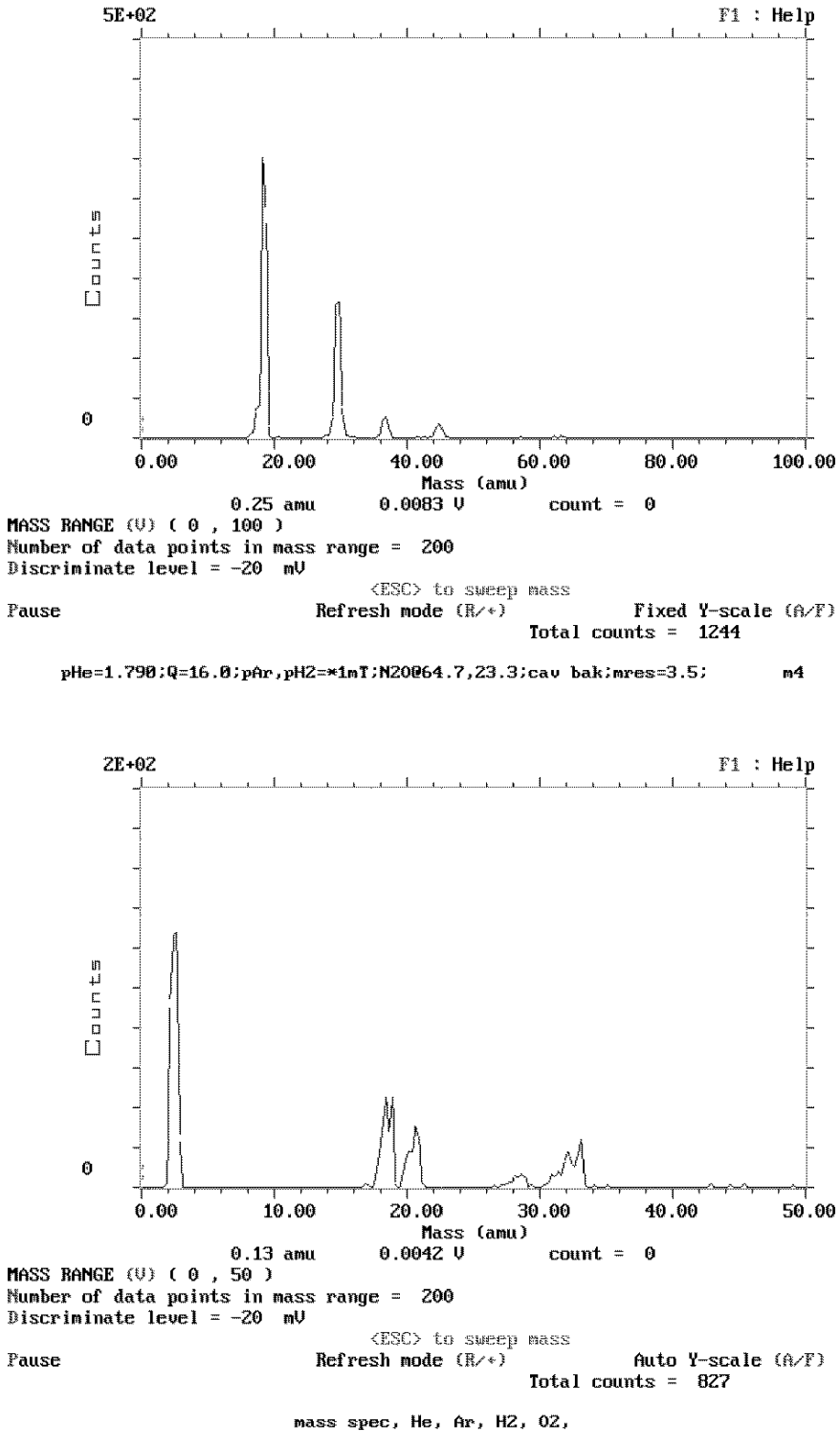


Figure 4-1.10 a: Pre-run mass spectra for N_2OH^+ (upper) and O_2H^+ (lower).

Figure 4-1.10 f

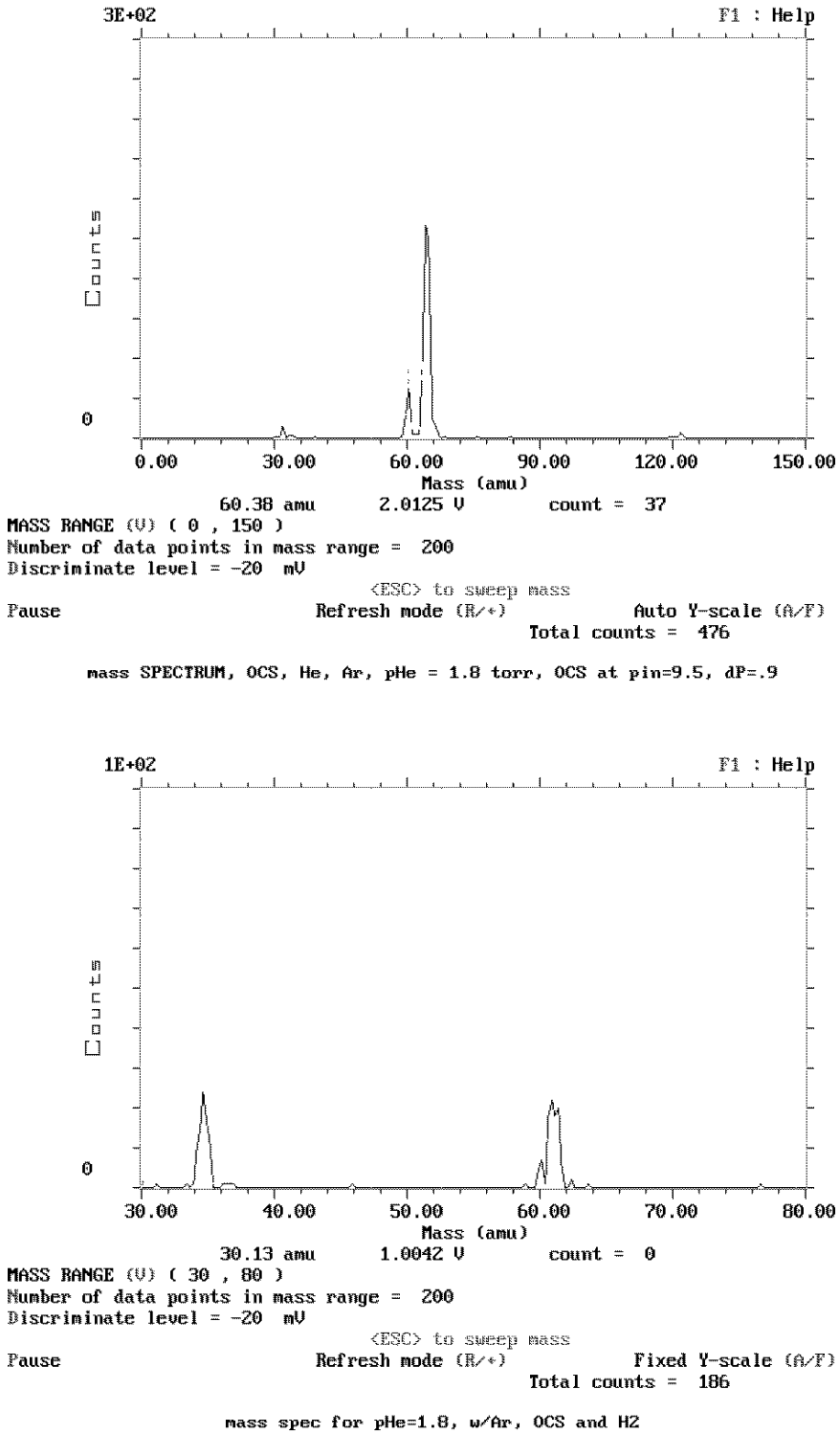


Figure 4-1.10 a: Pre-run mass spectra for OCS^+ (upper) and $HOCS^+$ (lower).

Figure 4-1.10 g

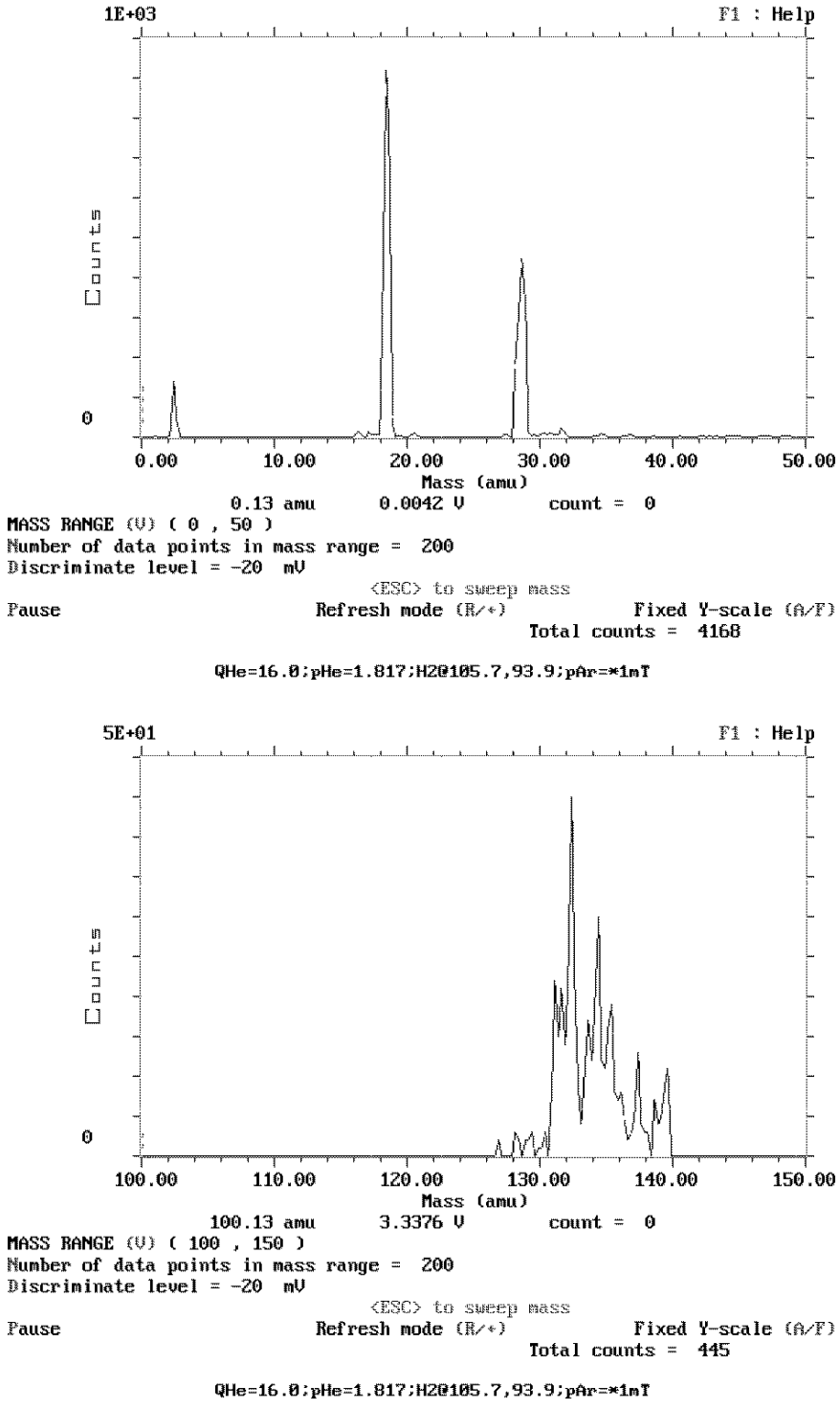


Figure 4-1.10 a: Pre-run mass spectra for N_2 and H_2O contamination in XeH^+ run (upper) and XeH^+ (lower).

APPENDIX E

OPTICAL SYSTEM TRANSMISSION EFFICIENCY FORMULAE

System Efficiency Correction Curves for the

UV–VIS and VUV Spectrometry Systems

UV–VIS:

Experimental Conditions for Discussion in Section 4-3.2

nm Range	Correction Formula
200-280	$0.023255 - 0.02634921(nm/100) - 0.000962396(nm/100)^2$ $+0.004420605(nm/100)^3 + 0.001546166(nm/100)^4$ $-0.00009261899(nm/100)^5 - 0.0001919025(nm/100)^6$
280-400	$-0.4958784 + 0.3426415(nm/100) - 0.03660233(nm/100)^2$ $+0.008210709(nm/100)^3 - 0.009085676(nm/100)^4$ $-0.0008663098(nm/100)^5 + 0.00122605(nm/100)^6$ $-0.0001590731(nm/100)^7$
400-500	$-0.190926 + -0.148401(nm/100) + 0.07299296(nm/100)^2$ $+0.007812568(nm/100)^3 - 0.003174406(nm/100)^4$ $-0.0004277098(nm/100)^5 + 0.00009378758(nm/100)^6$
500-600	$0.1477934 - 0.02398665(nm/100) - 0.003425963(nm/100)^2$ $+0.0006856264(nm/100)^3 - 0.000064683813(nm/100)^4$ $+0.00001120682(nm/100)^5 - 0.0000005838109(nm/100)^6$
600-700	$0.1244297 - 0.4202022(nm/1000) + 0.4710591(nm/1000)^2$ $-0.1728368(nm/1000)^3$
700-800	$0.007713982 + 0.00113327(nm/100) - 0.0002124426(nm/100)^2$ $-0.000009682039(nm/100)^3 - 0.000005127594(nm/100)^4$ $0.0000008066386(nm/100)^5 - 0.00000001307592(nm/100)^6$

VUV:*

nm Range	Correction Formula
140-200	$\Big\lvert 0.5507449 - 1.502763(nm/100) + 1.462712(nm/100)^2$ $-0.6013169(nm/100)^3 + 0.09019877(nm/100)^4 \Big\rvert$
200-300	$-0.3136474 + 0.2604572(nm/100) + 0.03235921(nm/100)^2$ $-0.06216563(nm/100)^3 + 0.01186695(nm/100)^4$

To correct photon counts, divide the observed number of photon counts by the formula appropriate to the wavelength in nanometers, nm. *VUV system efficiency constructed by Mark Drucker.