Global Change Research Center
Stratospheric Ozone Model

Zhongyi Ye
B.S., Peking University, China, 1989

A Thesis submitted to the faculty of the
Oregon Graduate Institute of Science & Technology
in partial fulfillment of the
requirements for the degree
Master of Science
in
Environmental Science & Engineering

January, 1995

The thesis "Global Change Research Center Stratospheric Ozone Model" by Zhongyi Ye has been examined and approved by the following Examination Committee:

Dr. M. Aslam K. Khalil, Professor, Thesis Advisor
Oregon Graduate Institute of Science & Technology

Dr. Reinhold A. Rasmussen, Professor
Oregon Graduate Institute of Science & Technology

Dr. James J. Huntzicker, Professor
Oregon Graduate Institute of Science & Technology

Dr. Wesley M. Jarrell, Professor
Oregon Graduate Institute of Science & Technology

DEDICATION

To

Freedom, democracy, and science

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my thesis advisor, Dr. M.A.K. Khalil, for his guidance, support, and kindness. Dr. Khalil is the most brilliant and kindest professor I have ever met. Here I really appreciate his advice, help and patience.

I would like to express my appreciation to the other members of my thesis committee, Dr. Reinhold Rasmussen, Dr. James Huntzicker, and Dr. Wesley Jarrell for their time and comments.

I would like to thank Dr. Francis Moraes for his valuable help with respect to the programming, discussion, and writing. I have benefited from his GCRC-ACM model while doing this thesis. I thank Dr. Robert MacKay for his valuable comments and discussions. I thank Ms. Marty Shearer for her technical help. I thank Ms. Edie Taylor for her secretarial assistance.

I would like to thank Mr. Tom Marshall from my deep heart for his kindness, friendliness, and encouragement. I am also grateful for his careful and patient help with respect to my English and American culture. It is he who has shown me to understand the real American. I also thank his wife, Mrs. Sue Marshall, for her kindness and friendliness.

I would like to thank my parents, my parents-in-law, my brothers and sisters, my brother-in-law, and my niece and nephew. Because they have been making my life so meaningful.

Finally I am extremely grateful to my wife, Peng Xu, for her love and support. Here I give my greatest appreciation to her.

TABLE OF CONTENTS

List of Figures

List of Tables

ABSTRACT

Global Change Research Center
Stratospheric Ozone Model

Zhongyi Ye, M.S.

Oregon Graduate Institute of Science & Technology, 1994

Supervising Professor: M.A.K. Khalil

The GCRC stratospheric ozone model has been developed to study the vertical distribution of ozone in the stratosphere. It is a time-dependent, one-dimensional model which includes atmospheric transport and detailed stratospheric chemistry. A routine for actinic flux is included in the model to calculate the photodissociation rates independently. Details of the model are presented in this thesis so that it can be easily used and understood by others. Oxygen, hydrogen, nitrogen and chlorine cycles are explored and their effects and role on ozone are developed here. The model produces ozone vertical profiles with an ozone maximum at altitude around 25 kilometers. Also ozone shows different daily variations at different altitudes. These results compare favorably with other published results from similar models.

GCRC stratospheric ozone model is a valid base model and can serve as a foundation for the future study of ozone including seasonal, latitudinal and long term variations, and ozone depletion by man-made chemicals.

CHAPTER 1 INTRODUCTION

Ozone is the only atmospheric constituent which effectively absorbs ultraviolet solar radiation from a wavelength of about 250 to 310 nm, protecting plant and animal life from exposure to harmful radiation (UV-B). Moreover, since the absorbed solar energy is converted into thermal energy, the absorption of UV light by ozone constitutes the principal source of heat in the middle atmosphere and is therefore responsible for the existence of the stratosphere. The stratosphere is an atmospheric layer with a positive temperature gradient and considerable static stability, extending from about 10 kilometers to about 50 kilometers above earth's surface. The ozone layer occurs around the region where ozone abundances peak, at about 20 to 25 kilometers above the surface. Ozone is also an efficient absorber for infrared radiation.

Stratospheric Ozone is formed initially when an oxygen molecule absorbs shortwave radiation, which splits it into two oxygen atoms; each oxygen atom then combines with another oxygen molecule to form ozone. These reactions are:

$$O_2 + h\nu \rightarrow O + O \tag{1.1}$$

$$O + O_2 + M \rightarrow O_3 + M \tag{1.2}$$

Ozone destruction can happen in several ways. First, ozone reacts with an oxygen atom to create two oxygen molecules in (1.3). This is very slow process. Second, ozone absorbs UV lights and dissociates in (1.4). These are:

$$O + O_3 \rightarrow O_2 + O_2 \quad (1.3)$$

$$O_3 + h\nu \rightarrow O + O_2 \quad (1.4)$$

The four reactions (1.1)-(1.4) are called the Chapman reactions and were the first theory of the ozone layer.

Third, catalytic cycles can speed up the destruction of ozone with the net effect of $O + O_3 \rightarrow 2O_2$ (as in 1.3). These catalytic cycles are started by products from the photodissociation of naturally emitted gases as well as anthropogenic chemicals. If the catalytic cycles are not taken into account, the O_3 predicted by the Chapman scheme turns out to be much higher than observed. Four chemical families are involved in these speeded-up catalytic reactions: they are hydrogen, nitrogen, chlorine, and bromine. The catalytic reactions can be written in generic form as:

$$X + O_3 \rightarrow XO + O_2 \quad (1.5)$$

$$XO + O \rightarrow X + O_2 \quad (1.6)$$

$$\text{Net: } O + O_3 \rightarrow 2O_2 \quad (1.7)$$

where catalyst X represents H, OH, NO, Cl, or Br.

The purpose of this thesis is to develop a one-dimensional time-dependent stratospheric ozone model. The current model includes complete stratospheric chemistry, atmospheric transport and attenuation of solar radiation. It has an independent routine to calculate actinic flux at each atmospheric layer for each wavelength. The chlorine chemistry included in the model is the most complete when compared with published ozone models. The model will be useful for future global change research, especially the effects of CFCs and HCFCs on stratospheric ozone.

There are four main parts in this thesis. Chapter 2 gives a general review of ozone measurements and global distributions. Chapter 3 describes details of the stratospheric chemistry which includes oxygen, hydrogen, nitrogen and chlorine cycles; and a comparison between stratospheric chemistry and tropospheric chemistry. Chapter 4 shows the details and results of current model so that readers may reproduce the model if interested. Chapter 5 compares the current GCRC Stratospheric Ozone Model with the Lawrence Livermore National Laboratory (LLNL) model. Chapter 6 contains the concluding remarks.

CHAPTER 2 OZONE MEASUREMENT AND DISTRIBUTION

2.1 Ozone Measurements

For gas phase species, the commonly used units are parts per million by volume (ppmv), number of molecule per cubic centimeter (molecule/cm^3), micrograms per cubic meter (ug/m^3). For the ozone column, the Dobson Unit (DU) is often used. (See appendix)

Measurements of stratospheric ozone fall into two categories: total ozone observed from the ground or by means of satellites, and vertical distributions derived from balloon or rocket sondes.

2.1.1 Total Ozone Ground-Based Measurements

Three instruments are routinely used to measure total ozone from Earth's surface: the Dobson spectrophotometer, the M-83 filter ozonometer, and the Brewer grating spectrophotometer.

The Dobson Spectrophotometer is the standard instrument used for observations of total ozone. It contains a double quartz-prism monochromator that permits comparison of the radiances at two different wavelengths in the ultraviolet (UV). Since the number of stations is greater and the existing records are typically longer, the Dobson instruments constitute the backbone of the ground-based network (WMO 1988). The data are

expressed in Dobson units. It is mainly stratospheric ozone that is determined since the contributions of tropospheric and mesospheric ozone to the total column content are small.

The optical filter instrument is used chiefly in the former USSR and Eastern Europe. It is based on the same principle as the Dobson spectrophotometer in using differential absorption of UV radiation in the 300-350 nm Huggins band of ozone. The M-83 instrument, however, uses two broadband filters and measures the relative attenuation of the solar UV radiances either directly from the sun or indirectly from the zenith sky (WMO 1988).

An improved optical and electronic scheme for observations of total ozone was proposed by Brewer (1973) based in part on earlier developments by Wardle et al. (1963). The instrument has one diffraction grating (1,800 lines per mm) and five slits corresponding to five wavelengths in the 306-320 nm spectral band. The resolution is 0.6 nm as compared to 0.9-3.0 nm in the Dobson instrument.

2.1.2 Total Ozone Satellite Measurements

The Nimbus-7 satellite was launched on October 23, 1978, with two instruments on board, each capable of monitoring total ozone from all of the sunlit globe. They are the TOMS and SBUV spectrometers

The Total Ozone Mapping Spectrometer (TOMS) on the Nimbus-7 satellite was designed to provide daily global maps of the Earth's total ozone by measuring sunlight backscattered from the Earth-atmosphere system in six wavelength bands, each with a 1.0 nm bandpass: 312.5, 317.5, 331.2, 339.8, 360, and 380 nm. The two longest wavelengths are insensitive to atmospheric ozone and are used to measure surface reflectivity, while the remaining wavelengths are used in the inference of total ozone concentrations.

The Solar Backscatter Ultraviolet (SBUV) Spectrometer, also on the Nimbus-7 satellite, is similar to TOMS but is designed to measure ozone profiles as well as total ozone. SBUV measures the solar backscattered radiance every 32 seconds (i.e., as many as 2,700 data points per day) at 12 wavelengths, with a bandpass of 1.1 nm: 255.5, 273.5, 283.0, 287.6, 292.2, 298.5, 301.9, 305.8, 312.5, 317.5, 331.2, and 339.8 nm. The wavelengths from 255.5 nm to 305.8 nm are used to infer an ozone profile, while the four longer wavelengths, which are identical to those in TOMS, are used to infer total ozone. It uses a photometer set at 343 nm to measure reflectivity. The algorithms used to infer total ozone for TOMS and SBUV are identical.

2.1.3 Vertical Ozone Distribution Measurements

The standard ground based observational technique is the Umkehr method which makes use of a Dobson spectrophotometer to determine the vertical distribution of ozone. Also ozonesondes are used for taking routine observations, balloonborne devices that utilize the oxidizing capability of ozone to cause a measurable chemical change.

2.2 Profiles of Ozone Distribution

2.2.1 Total Ozone

Figure 2.1 shows the average distribution of total ozone over the globe for 1957-1975. The data were deduced from the network of Dobson stations around the world. Figure 2.1 indicates:

- A pronounced minimum over the equatorial belt, with increasing concentrations poleward to about 70 degrees, and with a secondary minimum at the poles;
- The belt of equatorial ozone minimum (240 DU) lies between 10°S and 15°N latitudes;

- The total ozone increases in a poleward direction in both hemispheres, but the north and south are not symmetrical;
- Pronounced longitudinal inhomogeneities exist;
- The ozone concentration over the Antarctic is smaller than over the Arctic, reflecting differences in the patterns and behavior of the individual circumpolar vortices.

Figure 2.2 shows the average concentration of total ozone as a function of latitude and month. The data were derived from the TOMS total ozone measurements averaged over 9 years from 1979-1987. The main features are:

- The lowest values of total ozone are found in the equatorial region, whereas the highest values occur in spring near the North pole;
- In temperate and polar latitudes, substantial changes are observed with the seasons.
- The total ozone minimum is reached near the fall equinox and the maximum occurs near the spring equinox.

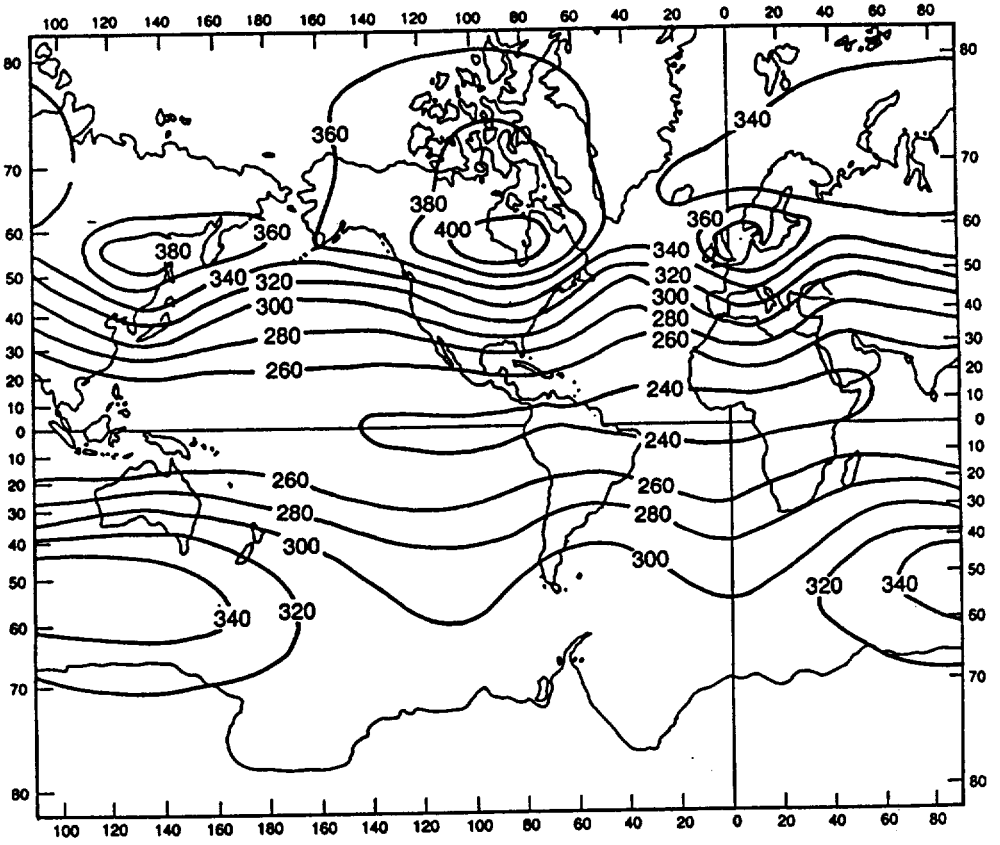

Figure 2.1 Taken from WMO, 1988. Average total ozone distribution for 1957-1975 derived from ground-based measurements.

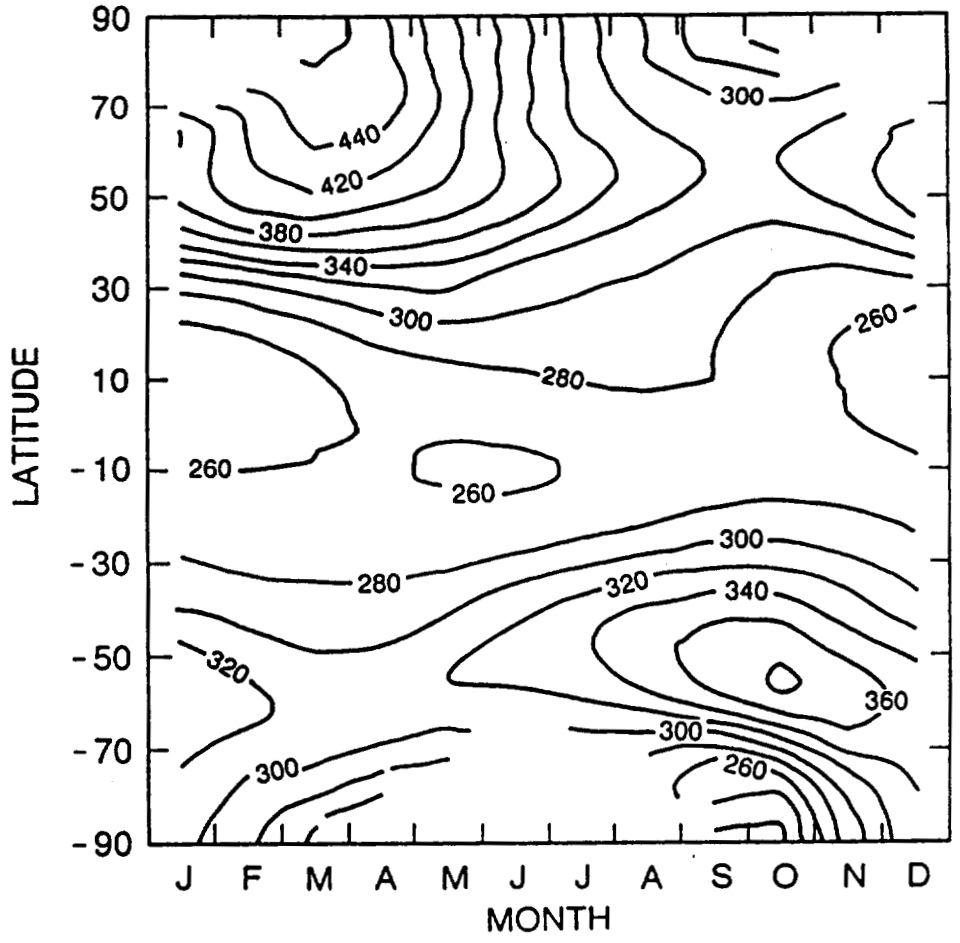

Figure 2.2 Taken from WMO, 1988. Variation of total ozone with latitude and season derived from TOMS measurements between 1979 and 1987.

2.2.2 Vertical Ozone Distribution

Figure 2.3 shows the observed height-latitude distribution of the ozone concentration. The data are from four year average of SBUV measurements. Figure 2.3 indicates:

- Ozone maximum is at about 35 km when using ppmv as its concentration unit. If PPMV is converted to Molecule/cm^3, the maximum will be around 25 km.
- The ozone concentration is evenly distributed latitudinally at the high stratosphere, about 40 km and up. More latitudinal variations are observed at low stratosphere.

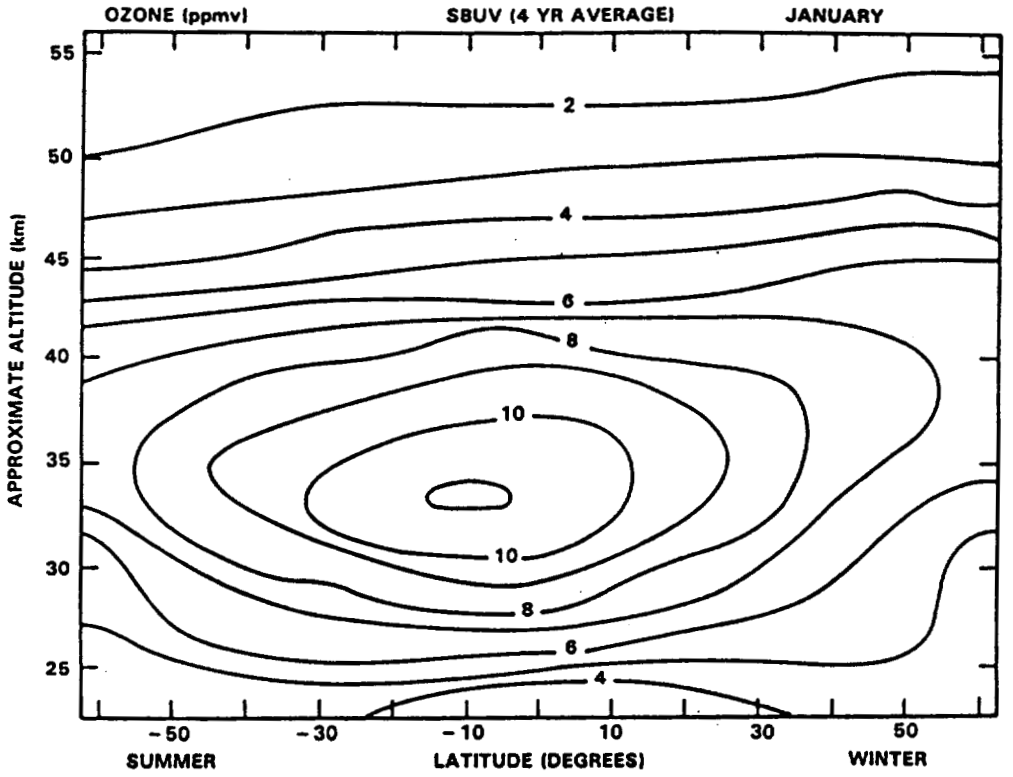

Figure 2.3 Two dimensional distribution of the ozone mixing ratio (ppmv). It is a four year average of SBUV data (taken from WMO, 1985).

Figure 2.4 indicates two examples for the vertical distribution of ozone. The data were selected to present typical concentration profiles for low and high latitudes. We can see the existence of the ozone layer in both examples. The ozone number mixing ratio (ppm) is given by the ratio of the ozone partial pressure to the atmospheric pressure at that level. The mass mixing ratio (ug/g) is 0.6 times that value. In the tropics, due to the high-lying tropopause, the layer is fairly narrow with a maximum at 26 km altitude. In the middle and high latitudes the distribution is broader, much more ragged, and the maximum occurs at heights between 20 and 23 km.

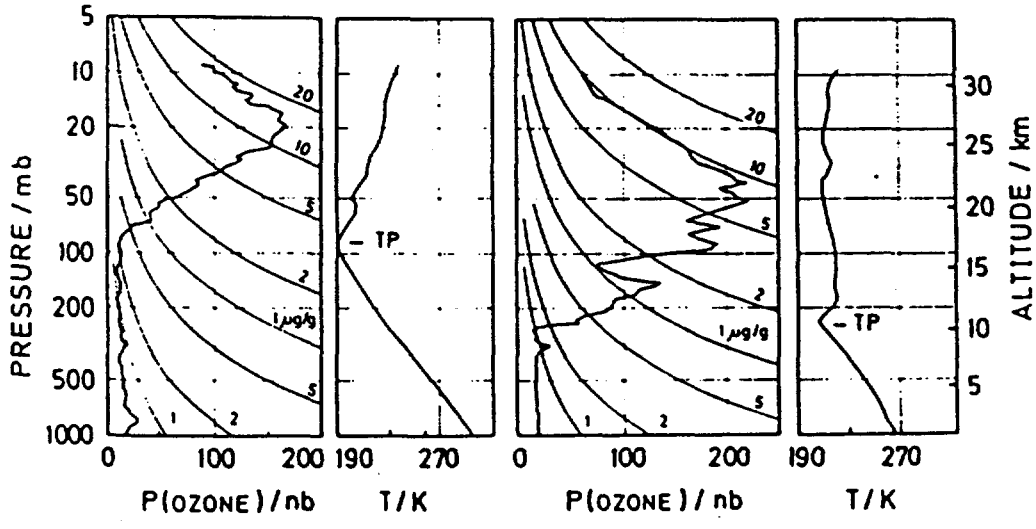

Figure 2.4 Balloon soundings of ozone (in units of partial pressure). Left: Low latitude, Right: High latitude. Vertical temperature profiles are included to indicate the tropopause levels (taken from Warneck, 1988).

CHAPTER 3 OZONE CHEMISTRY

3.1. Fundamentals of Atmospheric Chemistry

Bimolecular Reactions

There are two types of bimolecular reactions,

$$A + B \rightarrow C + D \quad (3.1)$$

$$A + A \rightarrow B + C \quad (3.2)$$

Bimolecular reactions are usually independent of pressure. If an influence of pressure is observed, it is an indication that the reaction is not truly elementary but involves more that one chemical process. The rate of a bimolecular reaction increases as the temperature is raised. In the 200-300 temperature range, the rate constant is well represented by the Arrhenius expression

$$k = A \exp(-E/RT) \quad (3.3)$$

where T is the absolute temperature, R is the gas constant, and A and E are constants. E is the activation energy and E/R is the "activation temperature". The factor A is related to the collision frequency between reactants. In the gas phase, its upper limit is around 10^{-10} cm^3/molecule s, but usually it is smaller (Warneck 1988).

Termolecular Reactions

Termolecular reactions can be expressed as

$$A + B + M \rightarrow C + M \quad (3.4)$$

or

$$A + A + M \rightarrow B + M \quad (3.5)$$

The third participant M is usually N_2 or O_2 in the atmosphere and it acts like a "catalyst". Termolecular reactions exhibit a pressure dependence. The following formula is taken from NASA/JPL 1992 data book and is commonly used for these third order reactions:

$$k(Z) = k(M,T) = \left(\frac{k_0(T)[M]}{1 + (k_0(T)[M]/k_\infty(T))} \right) 0.6^{\left\{1+[\log_{10}(k_0(T)[M]/k_\infty(T))]^2\right\}^{-1}} \quad (3.6)$$

where $k_0(T)$ represents low-pressure limiting rate constants, $k_0(T) = k_0^{300}\ (T/300)^{-n}$ $cm^6 molecule^{-2}\ s^{-1}$. n indicates temperature dependence of low-pressure limiting rate constants. $k_\infty(T)$ is high-pressure limit rate constants, $k_\infty(T) = k_\infty^{300}(T/300)^{-m}$ cm^3 $molecule^{-1}\ s^{-1}$. m indicates temperature dependence of high-pressure limit rate constants.

Photochemical Reactions

Chemical reactions in the atmosphere are initiated mainly by photochemical processes. The basic ideas underlying photochemical reactions are as follows. Light can become photochemically effective only if it is absorbed—that is, radiative energy must be incorporated into a molecule. The mere interaction of light with matter, such as in scattering, is not sufficient to initiate a chemical reaction. Moreover, energy is quantized.

A ray of light may be described as a flux of photons, each of which represents an energy quantum $E = h\nu$ proportional to the frequency ν of light and inversely proportional to the wavelength $\lambda = c/\nu$, with c being the velocity of light. In the act of optical absorption, exactly one photon is taken up by the absorbing molecule, whose internal energy, in turn, is raised by exactly the amount of energy supplied by the photon. A quantum requirement is that the energy of the photon coincides with the energy level of one of the many higher states of the molecule relative to its initial state. If the molecular states are widely spaced, the spectrum of absorption as a function of wavelength will consist of discrete lines. A dense spacing will give the appearance of a continuous absorption spectrum. The strength of the absorption is determined by the probability for such transitions between the energy states. (See Warneck, 1988)

The photodissociation rate can be expressed as:

$$J_m = \int \phi_m(\lambda,T)\sigma_m(\lambda,T)I(\lambda,\theta,z)d\lambda \qquad (3.7)$$

where λ is wavelength, z is height, θ is the solar zenith angle, σ_m is the absorption cross section, ϕ_m is the quantum yield, and I is the local photon flux at the wavelength being considered. The quantum yield is defined as the number of product molecules formed divided by the number of photons absorbed.

3.2. Stratospheric Ozone Chemistry

3.2.1. Oxygen Cycle

The following oxygen species are involved in oxygen cycle: $O(^1D)$, O, O_2, O_3. Figure 3.1 shows the reactions involved in this cycle. Source gases are shown within diamonds, reactive derivative species within boxes. The major reaction pathways are indicated by the heaviest arrows, minor pathways by dashed arrows.

Oxygen atom O is mainly from the dissociation of oxygen molecules by solar radiation in the wavelength region 180-240 nm (Herman and Mentall 1982):

$$O_2 + h\nu \rightarrow O + O \qquad (1.1)$$

The attachment of oxygen atoms to molecular oxygen leads to the formation of ozone:

$$O + O_2 + M \rightarrow O_3 + M \qquad (1.2)$$

Ozone photodissociates and produces $O(^1D)$ in the wavelength band between 200 and 300 nm. $O(^1D)$ is very active and easily converted to O by quenching with N_2, O_2.

$$O_3 + h\nu \rightarrow O_2 + O(^1D) \qquad (3.8)$$

Above 300 nm, the process produces O:

$$O_3 + h\nu \rightarrow O_2 + O \qquad (3.9)$$

At wavelength below 220 nm, N_2O decomposes and produces $O(^1D)$:

$$N_2O + h\nu \rightarrow N_2 + O(^1D) \qquad (3.10)$$

At lower altitudes and particularly in the troposphere, O atom formation from the photodissociation of NO_2 by long wavelength ultraviolet radiation is more important:

$$NO_2 + h\nu \rightarrow NO + O \qquad (3.11)$$

Ozone is removed by its reacting with oxygen atoms:

$$O + O_3 \rightarrow O_2 + O_2 \qquad (1.5)$$

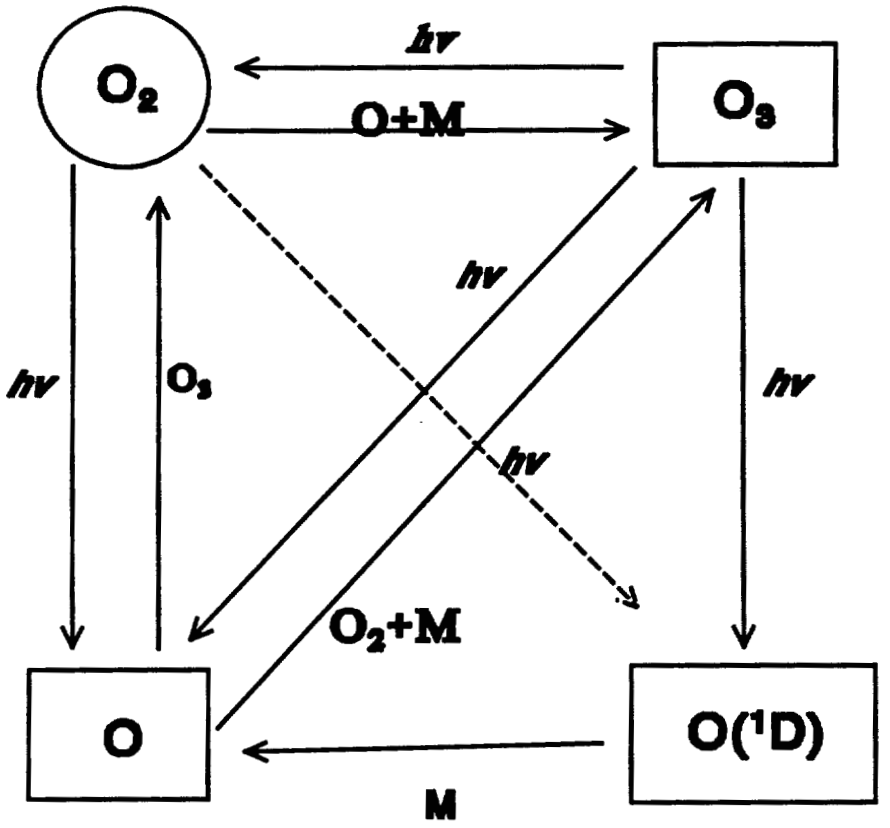

Figure 3.1 The pure oxygen photochemical cycle of the stratosphere. Source gases are shown within circles, reactive derivative species within boxes (Chapman Scheme). Minor pathways are indicated by the dashed arrows.

3.2.2. Hydrogen Cycle

Odd-hydrogen (HO_x) is defined as those species that contain a single hydrogen atom (e.g., the radicals H, OH, and HO_2), and it often includes stable compounds such as HNO_3 and H_2O_2 that are easily dissociated into odd-hydrogen radicals. In the stratosphere, odd-hydrogen is formed primarily by the reaction of metastable $O(^1D)$ atoms with water vapor or with molecular hydrogen:

$$O(^1D) + H_2O \rightarrow OH + OH \quad (3.12)$$

$$O(^1D) + H_2 \rightarrow OH + H \quad (3.13)$$

The hydrogen atoms react rapidly with the molecular oxygen to form HO_2:

$$H + O_2 + M \rightarrow HO_2 + M \quad (3.14)$$

All the odd-hydrogen radicals can recombine with one another to form H_2 and H_2O along several paths, the most important of which is initiated by the radical disproportionation:

$$HO_2 + HO_2 \rightarrow H_2O_2 + O_2 \quad (3.15)$$

The principal significance of odd-hydrogen is the catalytic destruction of ozone both directly and through its effects on odd-nitrogen and odd-chlorine. Four main reactive pathways are shown below.

$$OH + O_3 \rightarrow HO_2 + O_2 \quad (3.16)$$

$$HO_2 + O_3 \rightarrow OH + 2O_2 \quad (3.17)$$

$$\text{Net:} \quad O_3 + O_3 \rightarrow 3O_2 \quad (3.18)$$

$$OH + O \rightarrow H + O_2 \quad (3.19)$$
$$H + O_2 + M \rightarrow HO_2 + M \quad (3.14)$$
$$HO_2 + O \rightarrow OH + O_2 \quad (3.20)$$
$$\text{Net:} \quad O + O \rightarrow O_2 \quad (3.21)$$

$$OH + O \rightarrow H + O_2 \quad (3.19)$$
$$H + O_3 \rightarrow OH + O_2 \quad (3.22)$$
$$\text{Net:} \quad O + O_3 \rightarrow 2O_2 \quad (1.5)$$

and

$$HO_2 + O \rightarrow OH + O_2 \quad (3.20)$$
$$OH + O_3 \rightarrow HO_2 + O_2 \quad (3.16)$$
$$\text{Net:} \quad O + O_3 \rightarrow 2O_2 \quad (1.5)$$

The cycle of reactive HO_x species is shown in Figure 3.2.

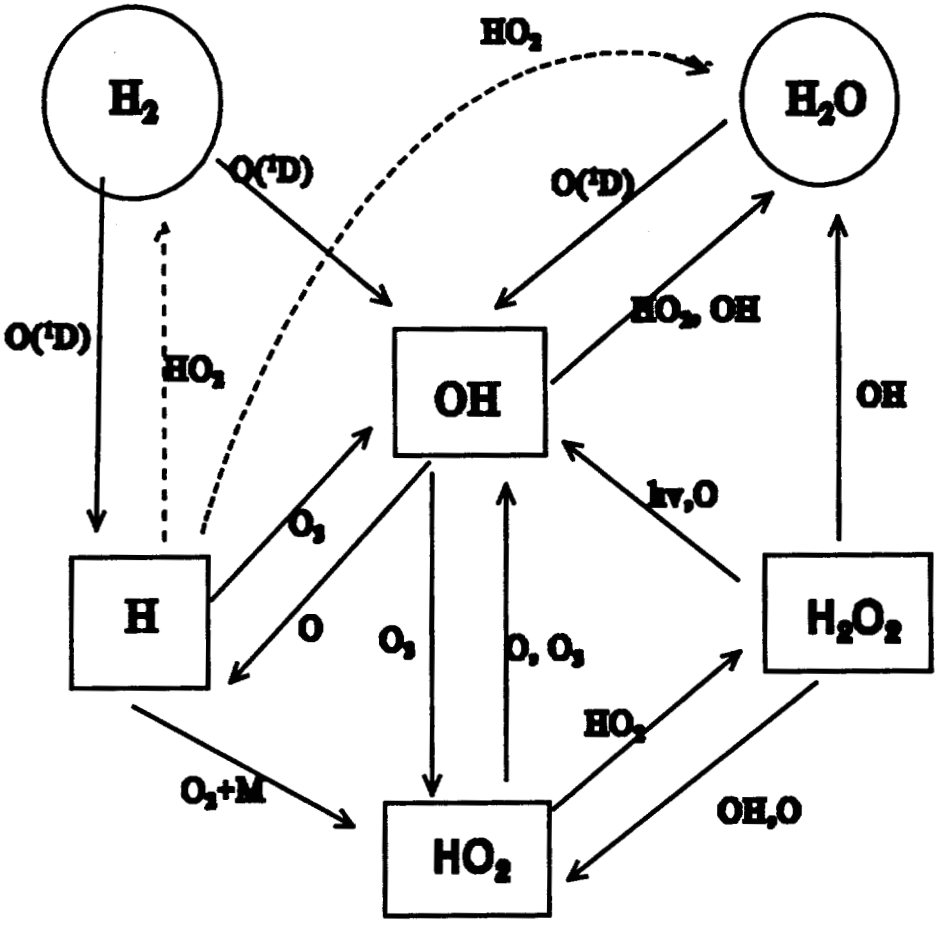

Figure 3.2 The hydrogen photochemical cycle of the stratosphere. Reactant species are shown alongside the arrows indicating reaction pathways. Minor pathways are indicated by the dashed arrows.

3.2.3. Nitrogen Cycle

Odd-nitrogen (NO_y) is defined as the family of species that contains one or more free nitrogen atoms (N, NO, NO_2, NO_3, N_2O_5, HNO_3, HO_2NO_2, and $ClONO_2$); NO_y is not easily produced in the atmosphere because nitrogen molecules are difficult to dissociate. Once formed, however, NO_y is not easily removed; stratospheric loss processes are so inefficient that NO_y must be transported to the mesosphere or ground before it can be destroyed efficiently. A subset of the odd-nitrogen family composed of NO and NO_2, and referred to as NO_x, is frequently distinguished from the other species because of its rapid reaction with ozone and atomic oxygen. (Levine 1985)

Nitrous oxide (N_2O) is by far the most important source of NO_x in the stratosphere through the reaction

$$N_2O + O(^1D) \rightarrow 2NO \qquad (3.23)$$

N_2O is released into the atmosphere as a result of natural microbiological activities and anthropological activities such as burning of fossil fuels. Likely perturbations of atmospheric N_2O by anthropogenic activities are still highly speculative (Khalil & Rasmussen, 1983b, 1992, 1994). Global distribution of nitrous oxide is quite uniform, demonstrating that its tropospheric lifetime must be very large or that the sources or sinks must be homogeneously distributed.

In addition to the formation of stratospheric NO_y by reaction (3.23), odd-nitrogen can be injected directly into the stratosphere by high altitude aircraft and by large nuclear explosions. Finally, NO_y is formed in both the stratosphere and mesosphere by charged-particle bombardment, particularly at high latitudes, and by large bodies such as meteors and space craft (Levine 1985).

Tropospheric NO_y is produced both by lightning and by combustion. However, it is removed from the atmosphere before it can be transported into the stratosphere.

Peroxynitric acid (HO_2NO_2), chlorine nitrate ($ClONO_2$), and nitrogen pentoxide (N_2O_5) are all formed by three-body association reactions:

$$HO_2 + NO_2 + M \rightarrow HO_2NO_2 + M \quad (3.24)$$

$$ClO + NO_2 + M \rightarrow ClONO_2 + M \quad (3.25)$$

$$NO_2 + NO_3 + M \rightarrow N_2O_5 + M \quad (3.26)$$

Nitrogen oxides are closely coupled to odd-oxygen, odd-hydrogen and odd-chlorine systems. The coupling of NO_y to HO_x leads to destruction of the latter. The reaction sequence is catalytic in nature since no NO_y is lost in the process. For example,

$$OH + HNO_3 \rightarrow H_2O + NO_3 \quad (3.28)$$

$$NO_3 + h\nu \rightarrow NO_2 + O \quad (3.29)$$

$$NO_2 + OH + M \rightarrow HNO_3 + M \quad (3.30)$$

$$\text{Net:} \quad OH + OH \rightarrow H_2O + O \quad (3.31)$$

and

$$OH + HO_2NO_2 \rightarrow H_2O + O_2 + NO_2 \quad (3.32)$$

$$NO_2 + HO_2 + M \rightarrow HO_2NO_2 + M \quad (3.24)$$

$$\text{Net:} \quad OH + HO_2 \rightarrow H_2O + O_2 \quad (3.33)$$

An important mechanism for ozone loss is the reactions between NO_x and odd-oxygen:

$$NO + O_3 \rightarrow NO_2 + O_2 \quad (3.34)$$

$$NO_2 + O \rightarrow NO + O_2 \quad (3.35)$$

$$\text{Net:} \quad O + O_3 \rightarrow 2O_2 \quad (1.5)$$

The cycle of reactive NO_x species is shown in Figure 3.3.

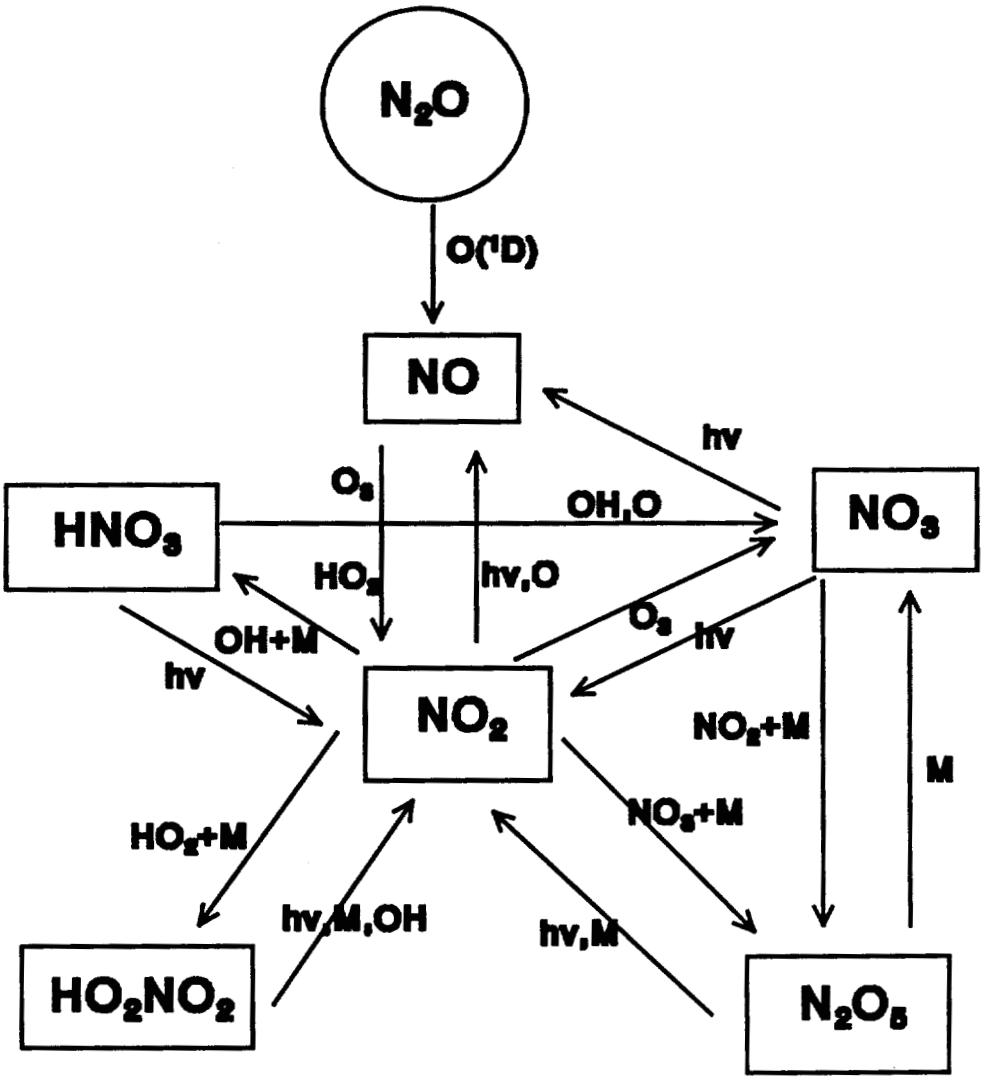

Figure 3.3 The photochemical cycle of odd-nitrogen in the present stratosphere. Sources gases are shown within circles, reactive species within boxes.

3.2.4. Chlorine Cycle

Odd-chlorine (ClO_x) includes Cl, ClO, HCl, $ClONO_2$, and HOCl. They are important because reactive Cl and ClO catalytically consume O_3 and O. Figure 3.4 shows the various source, sink, and reactive radical species involved in atmospheric chlorine chemistry and their relationship to each other. CH_3Cl (methyl chloride), CF_2Cl_2 (F-12), $CFCl_3$ (F-11), CCl_4 (carbon tetrachloride), and CH_3CCl_3 (methyl chloroform) are the main sources of odd-chlorine although in recent years other man-made compounds are also adding to the stratospheric chlorine burden. Of these compounds only CH_3Cl has major natural sources. The others are all man-made. Methyl chloride and methyl chloroform can be removed from the atmosphere by reacting with the hydroxyl radical, so that only a fraction of the initial release of these substances is able to reach the stratosphere. In contrast, no significant tropospheric sink has been identified for fully halogenated chlorofluoromethanes (except a few minor sinks of F-11 are known such soils and oceans). Almost all the chlorofluoromethane releases are therefore able to find their way into the stratosphere. Once in the stratosphere, all source gases generate atomic chlorine by photodissociation. Atomic chlorine can destroy ozone catalytically:

$$Cl + O_3 \rightarrow ClO + O_2 \tag{3.36}$$

$$ClO + O \rightarrow Cl + O_2 \tag{3.37}$$

$$\text{Net:} \quad O_3 + O \rightarrow 2O_2 \tag{1.5}$$

The Cl—ClO—Cl cycle is halted when Cl reacts with CH_4, CH_2O, or HO_2 to produce the inert HCl species:

$$Cl + CH_4 \rightarrow HCl + CH_3 \tag{3.38}$$

$$Cl + CH_2O \rightarrow HCl + CHO \tag{3.39}$$

$$Cl + HO_2 \rightarrow HCl + O_2 \tag{3.40}$$

This cycle is also terminated by reactions of ClO that lead to HOCl and $ClONO_2$ formation:

$$ClO + NO_2 + M \rightarrow ClONO_2 + M \quad (3.25)$$

$$ClO + HO_2 \rightarrow HOCl + O_2 \quad (3.41)$$

The photodissociation rates of Chlorine nitrate ($ClONO_2$) and hypochlorous acid (HOCl) are relatively high. $ClONO_2$ and HOCl are capable of driving the following two additional odd-oxygen destruction cycles:

$$Cl + O_3 \rightarrow ClO + O_2 \quad (3.36)$$

$$ClO + HO_2 \rightarrow HOCl + O_2 \quad (3.41)$$

$$OH + O_3 \rightarrow HO_2 + O_2 \quad (3.16)$$

$$HOCl + h\nu \rightarrow Cl + OH \quad (3.42)$$

$$\text{Net:} \quad 2O_3 \rightarrow 3O_2 \quad (3.18)$$

and

$$Cl + O_3 \rightarrow ClO + O_2 \quad (3.36)$$

$$ClO + NO_2 + M \rightarrow ClONO_2 + M \quad (3.25)$$

$$NO + O_3 \rightarrow NO_2 + O_2 \quad (3.34)$$

$$ClONO_2 + h\nu \rightarrow NO_3 + Cl \quad (3.43)$$

$$NO_3 + h\nu \rightarrow NO + O_2 \quad (3.44)$$

$$\text{Net:} \quad 2O_3 \rightarrow 3O_2 \quad (3.18)$$

Other chlorine species such as OClO, ClOO, Cl_2, Cl_2O_2, ClNO, $ClNO_2$ and ClONO are also important to the chlorine cycle. They are present in small quantities and dissociate very quickly. All these reactions are listed in the table in section 4.1.

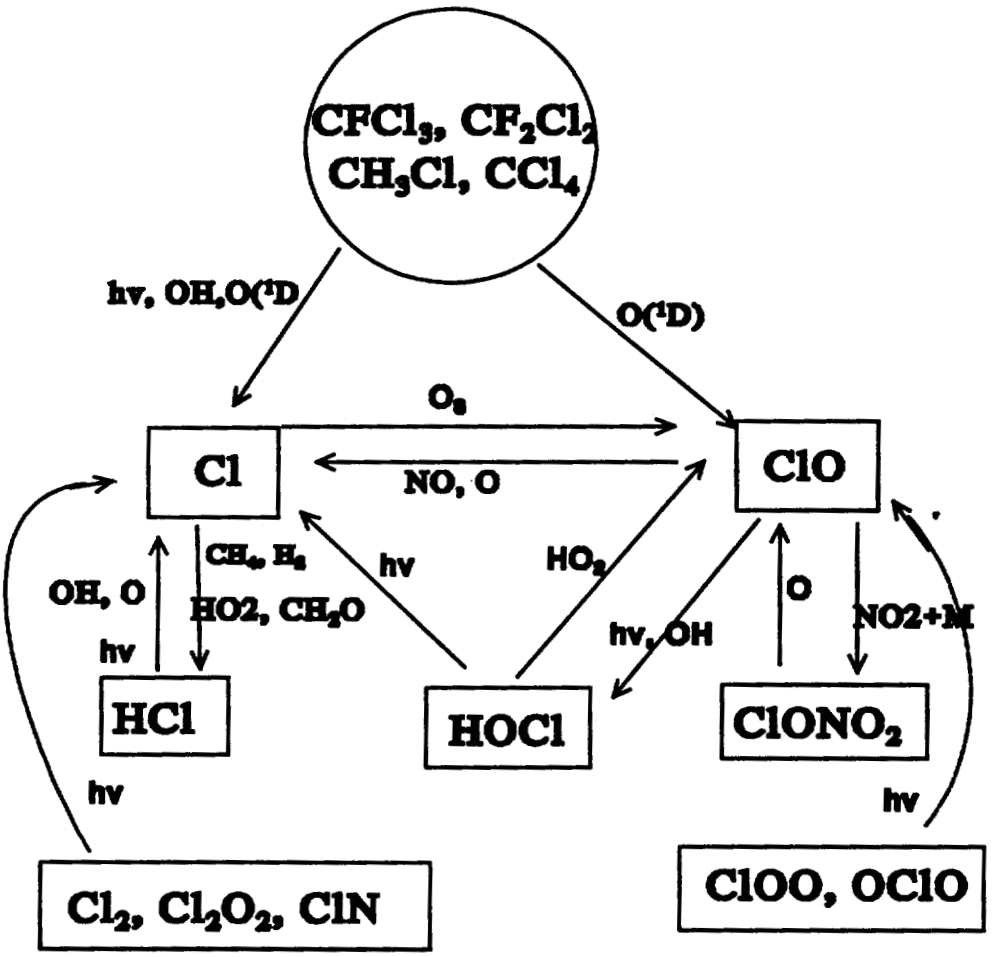

Figure 3.4 The photochemical cycle of odd-chlorine in the stratosphere. Source gases are shown within circles. Reactant species are shown alongside the arrows indicating reaction pathways. Minor pathways are indicated by the dashed arrows.

3.3 Comparison with Tropospheric Ozone Chemistry

Combustion of fossil fuels is of great importance to humans, and the byproducts of combustion are in many ways the most severe modifiers of the atmosphere. Complete combustion of hydrocarbon (C_xH_y) fuels produces carbon dioxide (CO_2) and water (H_2O):

$$C_xH_y + \tfrac{1}{4}(x+y)\, O_2 \rightarrow x\, CO_2 + \tfrac{1}{2}y\, H_2O \qquad (3.45)$$

and the high heat produced forms nitric oxide (NO) from the oxygen and nitrogen (N_2) in the air:

$$N_2 + O_2 \rightarrow 2NO \qquad (3.46)$$

In practice, combustion is never complete, and a variety of partially oxygenated products occur. These products, together with the incompletely oxidized nitrogen, are active participants in atmospheric chemistry.

Fuel burning produces high heat which makes N_2 combine with O_2 to form NO. A few percent of the NO is oxidized in the flame to form gaseous NO_2. NO_2 absorbs solar radiation and photodissociates to produce atomic oxygen:

$$NO_2 + h\nu \rightarrow NO + O \qquad \lambda < 420 \text{ nm} \qquad (3.47)$$

This reaction is rapidly followed by the reaction below to form O_3.

$$O + O_2 + M \rightarrow O_3 + M \qquad (1.2)$$

In the clean troposphere, ozone is the result of stratospheric-tropospheric exchange as well as production from chemical reactions involving the oxidation of CO, CH_4, and

possibly other biogenic hydrocarbons. The availability of NO_x determines the net production of ozone in the clean tropospheric environment from chemical oxidation reactions of CH_4 and CO. The chemical mechanism can be shown from the following reaction sequences (Warneck, 1988).

$$CH_4 + OH \rightarrow CH_3 + H_2O \quad (3.48)$$

$$CH_3 + O_2 + M \rightarrow CH_3O_2 + M \quad (3.49)$$

$$CH_3O_2 + NO \rightarrow CH_3O + NO_2 \quad (3.50)$$

$$CH_3O + O_2 \rightarrow HCHO + HO_2 \quad (3.51)$$

$$HO_2 + NO \rightarrow NO_2 + OH \quad (3.52)$$

$$NO_2 + h\nu \rightarrow NO + O \quad (x\ 2) \quad (3.53)$$

$$O + O_2 + M \rightarrow O_3 + M \quad (x\ 2) \quad (1.2)$$

$$\text{Net:} \quad CH_4 + 4O_2 \rightarrow HCHO + H_2O + 2O_3 \quad (3.54)$$

In the above CH_4 scheme two molecules of ozone are generated for each molecule of CH_4 consumed. And in the CO scheme below, one ozone molecule is formed for each CO molecule that is oxidized.

$$OH + CO \rightarrow CO_2 + H \quad (3.55)$$

$$H + O_2 + M \rightarrow HO_2 + M \quad (3.14)$$

$$HO_2 + NO \rightarrow NO_2 + OH \quad (3.52)$$

$$NO_2 + h\nu \rightarrow NO + O \quad (3.47)$$

$$O + O_2 + M \rightarrow O_3 + M \quad (1.2)$$

$$\text{Net:} \quad CO + 2O_2 \rightarrow CO_2 + O_3 \quad (3.56)$$

The CH_4-CO-NO_x reaction cycle cannot explain ozone exceedances observed in polluted atmosphere. In polluted atmospheres, unlike the clean troposphere, the rate of ozone formation is not necessarily proportional to the concentration of NO_x. It depends

upon the ratio and concentrations of the hydrocarbons and NO_x as well as the chemical composition of the hydrocarbons themselves. The hydrocarbons can be grouped as alkanes, alkenes, alkynes, and aromatics. We use the symbol R· to represent any saturated hydrocarbon chain less its terminal hydrogen atom, such as the methyl radical (CH_3·), the ethyl radical (CH_3CH_2·), etc. R· is formed when a hydrocarbon molecule reacts with a hydroxyl radical.

$$RH + HO \rightarrow R\cdot + H_2O \tag{3.57}$$

Then R· promptly adds molecular oxygen to form an peroxidic radical (ROO·):

$$R\cdot + O_2 + M \rightarrow ROO\cdot + M \tag{3.58}$$

Reaction between ROO· and nitric oxide is very effective and produces alkoxyl radical (RO·) and NO_2:

$$ROO\cdot + NO \rightarrow RO\cdot + NO_2 \tag{3.59}$$

Early in the morning the rush-hour traffic leads to high NO concentrations, which prevent appreciable amounts of ozone being present because of the reaction $O_3 + NO \rightarrow NO_2 + O_2$. Before ozone can appear, NO must first be converted to NO_2 by the degradation of hydrocarbons. As an example for such a reaction scheme, consider the oxidation of a straight-chain alkane $R_1CH_2R_2$ where R_1 and R_2 stand for suitable alkyl groups. The reaction is started by the abstraction of a hydrogen atom from the carbon skeleton, preferentially at sites of secondary hydrogen atoms. The chain is propagated by reactions of alkyl peroxy radicals.

$$R_1CH_2R_2 + HO \rightarrow R_1R_2CH + H_2O \tag{3.60}$$

$$R_1R_2CH + O_2 \rightarrow R_1R_2CHOO\cdot \tag{3.61}$$

$$R_1R_2CHOO\cdot + NO \rightarrow R_1R_2CHO\cdot + NO_2 \quad (3.62)$$

$$R_1R_2CHO\cdot + O_2 \rightarrow R_1COR_2 + HO_2 \quad (3.63)$$

$$R_1R_2CHO\cdot \rightarrow R_1\cdot + R_2CHO \quad (3.64)$$

$$R_1\cdot + O_2 \rightarrow R_1OO\cdot \quad (3.65)$$

$$R_1OO\cdot + NO \rightarrow R_1O\cdot + NO_2 \quad (3.66)$$

$$R_1O\cdot + O_2 \rightarrow R_1'CHO + HO_2 \quad (3.67)$$

$$HO_2 + NO \rightarrow NO_2 + OH \quad (3.52)$$

$$\text{Net: } R_1CH_2R_2 + 3NO + O_2 \rightarrow 3NO_2 + R_1COR_2 + R_1'CHO + H_2O \quad (3.68)$$

$$NO_2 + h\nu + O_2 \rightarrow NO + O_3 \quad (3.69)$$

Here, R_1' represents an alkyl group containing one carbon atom less than R_1 so that $R_1O\cdot = R_1'CH_2O\cdot$. The net reaction converts one molecule of n-alkane into one molecule of ketone and aldehyde each, and it oxidizes three molecules of NO to NO_2. The subsequent photodissociation of NO_2 is the source of ozone in photochemical smog. Note that the OH radical initiating the reaction sequence is regenerated so that it can continue the chain reaction.

Aldehydes are considerably more reactive toward OH radicals than alkanes, so that the aldehydes produced are subject to further oxidation. Thereby additional NO is converted to NO_2. To give an example, let R_1' stands for CH_3 so that $R_1'CHO$ represents acetaldehyde. the mechanism for acetaldehyde oxidation may be written as follows:

$$CH_3CHO + OH \rightarrow CH_3CO + H_2O \quad (3.70)$$

$$CH_3CO + O_2 \rightarrow CH_3(CO)O_2 \quad (3.71)$$

$$CH_3(CO)O_2 + NO \rightarrow CH_3 + CO_2 + NO_2 \quad (3.72)$$

$$CH_3 + O_2 \rightarrow CH_3O_2 \quad (3.49)$$

$$CH_3O_2 + NO \rightarrow CH_3O + NO_2 \quad (3.50)$$

$$CH_3O + O_2 \rightarrow HCHO + HO_2 \quad (3.51)$$

$$HO_2 + NO \rightarrow NO_2 + OH \quad (3.52)$$

$$\text{Net: } CH_3CHO + 3NO + 3O_2 \rightarrow HCHO + 3NO_2 + CO_2 + H_2O \qquad (3.73)$$

$$NO_2 + h\nu + O_2 \rightarrow NO + O_3 \qquad (3.69)$$

Here again, the OH radical initiating the reaction is regenerated. In the mechanism shown, three molecules of NO are oxidized to NO_2 and an equivalent number of ozone molecules is subsequently formed.

These processes can lead to O_3 formation in the troposphere which is a part of the total O_3 column in measurements. Generally it is a small contributor to the total ozone.

CHAPTER 4 CURRENT MODEL

4.1. Chemical Reactions

There are about 117 chemical reactions and 28 photolytic reactions in the current model. Time-dependent concentrations of thirty species are solved. These are:

Oxygen species (3): O, $O(^1D)$, O_3

Hydrogen species (4): H, HO, HO_2, H_2O_2

Nitrogen species (9): NO, NO_2, NO_3, N_2O_5, HNO_3, HNO_4, ClNO, $ClNO_2$, ClONO

Chlorine species (9): Cl, ClO, $ClONO_2$, HOCl, HCl, Cl_2, ClOO, OClO, Cl_2O_2

Methane cycle species (5): CH_3, CH_3O, CH_3O_2, CH_3OOH, H_2CO

The concentrations of eleven species are prescribed based on measurements or other models' output. They are O_2, N_2, H_2, H_2O, N_2O, CH_4, CO, CH_3Cl, CCl_4, $CFCl_3$, CF_2Cl_2.

All the chemical reactions and photodissociation reactions are listed in table 4.1. The data for rate constants are taken from NASA/JPL data book (DeMore et al., 1992). O_2, N_2 data are from standard atmosphere. H_2, H_2O, CH_3Cl, CCl_4, $CFCl_3$, CF_2Cl_2, CH_4, CO, N_2O data are taken from NASA/JPL 1992.

Table 4.1 Chemical Reactions, Rates, and Products Included in the Present Model

No.	Reactions	Rate
	A. Oxygen and Nitrogen Cycles	
1	$O + O_2 + M \rightarrow O_3 + M$	k_1 (See E. in page 35)
2	$O + O_3 \rightarrow O_2 + O_2$	$k_2 = 8.00E\text{-}12\ \exp(-2060/T)$
3	$O(^1D) + N_2 \rightarrow O + N_2$	$k_4 = 1.80E\text{-}11\ \exp(110/T)$
4	$O(^1D) + O_2 \rightarrow O + O_2$	$k_5 = 3.20E\text{-}11\ \exp(70/T)$
5	$O(^1D) + O_3 \rightarrow O_2 + O_2$	$k_6 = 1.20E\text{-}10$
6	$O(^1D) + O_3 \rightarrow O + O + O_2$	$k_7 = 1.20E\text{-}10$
7	$O(^1D) + N_2O \rightarrow N_2 + O_2$	$k_8 = 4.90E\text{-}11$
8	$O(^1D) + N_2O \rightarrow NO + NO$	$k_9 = 6.70E\text{-}11$
9	$NO + O_3 \rightarrow NO_2 + O_2$	$k_{10} = 2.30E\text{-}12\ \exp(-1450/T)$
10	$NO_2 + O \rightarrow NO + O_2$	$k_{11} = 9.30E\text{-}12$
11	$NO_2 + O_3 \rightarrow NO_3 + O_2$	$k_{14} = 1.20E\text{-}13\ \exp(-2450/T)$
12	$NO + NO_3 \rightarrow NO_2 + NO_2$	$k_{15} = 2.00E\text{-}11$
13	$NO + O + M \rightarrow NO_2 + M$	k_{16} (See E. in page 35)
14	$NO_2 + NO_3 + M \rightarrow N_2O_5 + M$	k_{17} (See E. in page 35)
15	$O(^1D) + H_2O \rightarrow OH + OH$	$k_{21} = 2.20E\text{-}10$
16	$O(^1D) + CH_4 \rightarrow OH + CH_3$	$k_{22} = 1.40E\text{-}10$
17	$O(^1D) + CH_4 \rightarrow H_2 + H_2CO$	$k_{23} = 1.40E\text{-}11$
18	$O(^1D) + H_2 \rightarrow OH + H$	$k_{24} = 9.90E\text{-}11$
	B. Hydrogen Cycle	
19	$H + O_3 \rightarrow OH + O_2$	$k_{25} = 1.40E\text{-}10\ \exp(-470/T)$
20	$H + O_2 + M \rightarrow HO_2 + M$	k_{26} (See E. in page 35)
21	$OH + O_3 \rightarrow HO_2 + O_2$	$k_{27} = 1.60E\text{-}12\ \exp(-940/T)$
22	$HO_2 + O_3 \rightarrow OH + O_2 + O_2$	$k_{28} = 1.10E\text{-}14\ \exp(-500/T)$
23	$OH + O \rightarrow H + O_2$	$k_{29} = 2.20E\text{-}11\ \exp(110/T)$
24	$HO_2 + O \rightarrow OH + O_2$	$k_{30} = 3.00E\text{-}11$
25	$H_2O_2 + O \rightarrow OH + HO_2$	$k_{31} = 1.40E\text{-}12\ \exp(-2000/T)$
26	$OH + CH_4 \rightarrow H_2O + CH_3$	$k_{32} = 2.90E\text{-}12\ \exp(-1710/T)$
27	$HO_2 + NO \rightarrow OH + NO_2$	$k_{33} = 3.50E\text{-}12\ \exp(250/T)$
28	$OH + CO \rightarrow CO_2 + H$	$k_{34} = 1.50E\text{-}13\ (1.0 + 0.6P)$
29	$OH + H_2 \rightarrow H_2O + H$	$k_{35} = 5.50E\text{-}12\ \exp(-2000/T)$
30	$OH + NO2 + M \rightarrow HNO_3 + M$	k_{36} (See E. in page 35)
31	$OH + HNO_3 \rightarrow H_2O + NO_3$	$k_{37} = 1.52E\text{-}14\ \exp(650/T)$
32	$OH + H_2O_2 \rightarrow H_2O + HO_2$	$k_{38} = 2.90E\text{-}12\ \exp(-160/T)$
33	$OH + HO_2 \rightarrow H_2O + O_2$	$k_{39} = 4.80E\text{-}11\ \exp(250/T)$
34	$OH + OH \rightarrow H_2O + O$	$k_{40} = 4.20E\text{-}12\ \exp(-240/T)$
35	$OH + H_2CO \rightarrow H_2O + HCO$	$k_{41} = 1.00E\text{-}11$
36	$HO_2 + HO_2 \rightarrow H_2O_2 + O_2$	$k_{42} = 2.30E\text{-}13\ \exp(600/T)$
37	$OH + OH + M \rightarrow H_2O_2 + M$	k_{43} (See E. in page 35)

38	$H + HO_2$	→	$H_2 + O_2$	$k_{44} = 7.40E\text{-}12$
39	$H + HO_2$	→	$H_2O + O$	$k_{45} = 2.22E\text{-}12$
40	$H + HO_2$	→	$OH + OH$	$k_{46} = 6.44E\text{-}11$

C. Methane Cycle

41	$H_2CO + O$	→	$OH + HCO$	$k_{47} = 3.40E\text{-}11\ \exp(-1600/T)$
42	$CH_3 + O_2 + M$	→	$CH_3O_2 + M$	k_{48} (See E. in page 35)
43	$CH_3O_2 + NO$	→	$CH_3O + NO_2$	$k_{49} = 4.20E\text{-}12\ \exp(180/T)$
44	$CH_3O_2 + HO_2$	→	$CH_3OOH + O_2$	$k_{50} = 3.80E\text{-}13\ \exp(800/T)$
45	$CH_3OOH + OH$	→	$CH_3O_2 + H_2O$	$k_{51} = 3.80E\text{-}12\ \exp(200/T)$
46	$CH_3O + O_2$	→	$H_2CO + HO_2$	$k_{52} = 3.90E\text{-}14\ \exp(-900/T)$

D. Chlorine Cycle

47	$O(^1D) + CF_2Cl_2$	→	$ClO + CF_2Cl$	$k_{54} = 1.40E\text{-}10$
48	$O(^1D) + CFCl_3$	→	$ClO + CFCl_2$	$k_{55} = 2.30E\text{-}10$
49	$O(^1D) + CH_3Cl$	→	$Cl + product$	$k_{56} = 1.00E\text{-}10$
50	$OH + CH_3Cl$	→	$Cl + product$	$k_{57} = 1.80E\text{-}12\ \exp(-1110/T)$
51	$Cl + O_3$	→	$ClO + O_2$	$k_{58} = 2.90E\text{-}11\ \exp(-260/T)$
52	$ClO + O$	→	$Cl + O_2$	$k_{59} = 3.00E\text{-}11\ \exp(70/T)$
53	$ClO + NO$	→	$Cl + NO_2$	$k_{60} = 6.40E\text{-}12\ \exp(290/T)$
54	$O(^1D) + HCl$	→	$Cl + OH$	$k_{61} = 1.50E\text{-}10$
55	$Cl + CH_4$	→	$HCl + CH_3$	$k_{62} = 1.10E\text{-}11\ \exp(-1400/T)$
56	$Cl + HO_2$	→	$HCl + O_2$	$k_{63} = 1.80E\text{-}11\ \exp(170/T)$
57	$Cl + H_2$	→	$HCl + H$	$k_{64} = 3.70E\text{-}11\ \exp(-2300/T)$
58	$HCl + H$	→	$Cl + H_2$	$k_{65} = 4.70E\text{-}11\ \exp(-2340/T)$
59	$Cl + H_2O_2$	→	$HCl + HO_2$	$k_{66} = 1.10E\text{-}11\ \exp(-980/T)$
60	$OH + HCl$	→	$Cl + H_2O$	$k_{67} = 2.60E\text{-}12\ \exp(-350/T)$
61	$Cl + O_2 + M$	→	$ClOO + M$	k_{68} (See E. in page 35)
62	$HO_2 + ClO$	→	$HOCl + O_2$	$k_{70} = 4.80E\text{-}13\ \exp(700/T)$
63	$H + ClO$	→	$OH + Cl$	$k_{71} = 3.00E\text{-}11$
64	$ClO + NO_2 + M$	→	$ClONO_2 + M$	k_{72} (See E. in page 35)
65	$ClONO_2 + O$	→	$ClO + NO + O_2$	$k_{73} = 2.90E\text{-}12\ \exp(-800/T)$
66	$ClONO_2 + OH$	→	$HOCl + NO_3$	$k_{74} = 1.20E\text{-}12\ \exp(-330/T)$
67	$H_2CO + Cl$	→	$HCl + HCO$	$k_{75} = 8.10E\text{-}11\ \exp(-30/T)$
68	$H_2 + O$	→	$OH + H$	$k_{76} = 8.80E\text{-}12\ \exp(-4200/T)$
69	$O(^1D) + CCl_4$	→	$ClO + CCl_3$	$k_{77} = 3.30E\text{-}10$
70	$OH + HOCl$	→	$ClO + H_2O$	$k_{78} = 3.00E\text{-}12\ \exp(-500/T)$
71	$HO_2 + NO_2 + M$	→	$HNO_4 + M$	k_{79} (See E. in page 35)
72	$HNO_4 + OH$	→	$H_2O + O_2 + NO_2$	$k_{81} = 1.30E\text{-}12\ \exp(380/T)$
73	$O + NO_2 + M$	→	$NO_3 + M$	k_{82} (See E. in page 35)
74	$O + NO_3$	→	$O_2 + NO_2$	$k_{83} = 1.00E\text{-}11$
75	$O + HNO_4$	→	$products$	$k_{84} = 7.80E\text{-}11\ \exp(-3400/T)$
76	$H + NO_2$	→	$OH + NO$	$k_{85} = 4.00E\text{-}10\ \exp(-340/T)$
77	$O + CH_3$	→	$products$	$k_{86} = 1.10E\text{-}10$

78	$CH_3 + O_3$	→	products	k_{87} = 5.40E-12 exp(-220/T)
	$CH_3O_2 + CH_3O_2$	→	products	K = 2.50E-13 exp(190/T)
79		→	$2CH_3O + O_2$	$k_{88} = k_a/K$ = 0.3 at 298K
80		→	$CH_2O + CH_3OH + O_2$	$k_{89} = k_b/K$ = 0.6 at 298K
81		→	$CH_3OOCH_3 + O_2$	$k_{90} = k_c/K$ = 0.1 at 298K
82	$Cl + CH_3Cl$	→	$CH_2Cl + HCl$	k_{91} = 3.30E-11 exp(-1250/T)
83	Cl + HOCl	→	$Cl_2 + OH$	k_{92} = 3.00E-12 exp(-130/T)
84	$Cl + NO_3$	→	$ClO + NO_2$	k_{93} = 2.60E-11
85	ClO + OH	→	products	k_{94} = 1.10E-11 exp(120/T)
86	$ClO + CH_4$	→	products	k_{95} = 1.00E-12 exp(-3700/T)
87	$ClO + H_2$	→	products	k_{96} = 1.00E-12 exp(-4800/T)
88	$HO_2 + HO_2 + M$	→	$H_2O_2 + O_2 + M$	k_{97} = 1.70E-33 [M] exp(1000/T)
89	$OH + CFCl_3$	→	products	k_{98} = 1.00E-12 exp(-3700/T)
90	$OH + CF_2Cl_2$	→	products	k_{99} = 1.00E-12 exp(-3600/T)
91	O + HCl	→	OH + Cl	k_{100} = 1.0E-11 exp(-3300/T)
92	O + HOCl	→	OH + CO	k_{101} = 1.0E-11 exp(-2200/T)
93	$O(^1D) + Cl_2$	→	products	k_{102} = 2.8E-10
94	$Cl + HO_2$	→	OH + ClO	k_{103} = 4.1E-11 exp(-450/T)
95	Cl + OClO	→	ClO + ClO	k_{105} = 3.4E-11 exp(160/T)
96	Cl + ClOO	→	$Cl_2 + O_2$	k_{106} = 2.3E-10
97		→	$Cl_2 + O_2$	k_{107} = 1.2E-11
98	$Cl + Cl_2O_2$	→	products	k_{108} = 1.0E-10
99	$Cl + ClONO_2$	→	products	k_{109} = 6.8E-12 exp(160/T)
100	Cl + ClNO	→	$NO + Cl_2$	k_{110} = 5.8E-11 exp(100/T)
101	$ClO + NO_3$	→	products	k_{111} = 4.0E-13
102	$ClO + H_2CO$	→	products	k_{112} = 1.0E-12 exp(-2100/T)
103	ClO + CO	→	products	k_{113} = 1.0E-12 exp(-3700/T)
104	$ClO + N_2O$	→	products	k_{114} = 1.0E-12 exp(-4300/T)
105	ClO + ClO	→	products	k_{115} = 8.0E-13 exp(-1250/T)
106	$ClO + O_3$	→	$OClO + O_2$	k_{116} = 1.0E-12 exp(-4000/T)
107	$OH + Cl_2$	→	HOCl + Cl	k_{117} = 1.4E-12 exp(-900/T)
108	$OH + ClNO_2$	→	$HOCl + NO_2$	k_{118} = 3.5E-14
109	OClO + O	→	$ClO + O_2$	k_{120} = 2.5E-12 exp(-950/T)
110	$OClO + O_3$	→	products	k_{121} = 2.1E-12 exp(-4700/T)
111	OClO + OH	→	$HOCl + O_2$	k_{122} = 4.5E-13 exp(800/T)
112	OClO + NO	→	$NO_2 + ClO$	k_{123} = 2.5E-12 exp(-600/T)
113	Cl + NO + M	→	ClNO + M	k_{124} (See E. in page 35)
114	$Cl + NO_2 + M$	→	ClONO + M	k_{125} (See E. in page 35)
115		→	$ClNO_2 + M$	k_{126} (See E. in page 35)
116	Cl + CO + M	→	ClCO + M	k_{127} (See E. in page 35)
117	ClO + ClO + M	→	$Cl_2O_2 + M$	k_{128} (See E. in page 35)

E. Termolecular Reactions

No.	Reactions	k_0^{300}	n	k_∞^{300}	m
1	$O + O_2 + M \rightarrow O_3 + M$	6.0E-34	2.3	-	-
13	$NO + O + M \rightarrow NO_2 + M$	9.0E-32	1.5	3.0E-11	0
14	$NO_2 + NO_3 + M \rightarrow N_2O_5 + M$	2.2E-30	3.9	1.5E-12	0.7
20	$H + O_2 + M \rightarrow HO_2 + M$	5.7E-32	1.6	-	-
30	$OH + NO2 + M \rightarrow HNO_3 + M$	2.6E-30	3.2	2.4E-11	1.3
37	$OH + OH + M \rightarrow H_2O_2 + M$	6.9E-31	0.8	1.5E-11	0
42	$CH_3 + O_2 + M \rightarrow CH_3O_2 + M$	4.5E-31	3.0	1.8E-12	1.7
61	$Cl + O_2 + M \rightarrow ClOO + M$	2.7E-33	1.5	-	-
64	$ClO + NO_2 + M \rightarrow ClONO_2 + M$	1.8E-31	3.4	1.5E-11	1.9
71	$HO_2 + NO_2 + M \rightarrow HNO_4 + M$	1.8E-31	3.2	4.7E-12	1.4
73	$O + NO_2 + M \rightarrow NO_3 + M$	9.0E-32	2.0	2.2E-11	0
113	$Cl + NO + M \rightarrow ClNO + M$	9.0E-32	1.6	-	-
114	$Cl + NO_2 + M \rightarrow ClONO + M$	1.3E-30	2.0	1.0E-10	1.0
115	$\rightarrow ClNO_2 + M$	1.8E-31	2.0	1.0E-10	1.0
116	$Cl + CO + M \rightarrow ClCO + M$	1.3E-33	3.8	-	-
117	$ClO + ClO + M \rightarrow Cl_2O_2 + M$	1.9E-32	3.9	7.0E-12	0

F. Photodissociation reactions

No.	Reactions			Rate
1	$O_2 + h\nu$	$\rightarrow$	$O + O$	J_1
2	$O_3 + h\nu$	$\rightarrow$	$O_2 + O(^3P)$	J_2
3	$O_3 + h\nu$	$\rightarrow$	$O_2 + O(^1D)$	J_3
4	$NO_2 + h\nu$	$\rightarrow$	$NO + O$	J_4
5	$N_2O + h\nu$	$\rightarrow$	$N_2 + O(^1D)$	J_5
6	$HNO_3 + h\nu$	$\rightarrow$	$OH + NO_2$	J_6
7	$H_2O_2 + h\nu$	$\rightarrow$	$OH + OH$	J_7
8	$NO_3 + h\nu$	$\rightarrow$	$NO + O_2$	J_8
9	$H_2CO + h\nu$	$\rightarrow$	$H + HCO$	J_9
10	$H_2CO + h\nu$	$\rightarrow$	$H_2 + CO$	J_{10}
11	$CF_2Cl_2 + h\nu$	$\rightarrow$	$Cl + CF_2Cl$	J_{11}
12	$CFCl_3 + h\nu$	$\rightarrow$	$Cl + CFCl_2$	J_{12}
13	$ClONO_2 + h\nu$	$\rightarrow$	$Cl + NO_3$	J_{13}
14	$HCl + h\nu$	$\rightarrow$	$H + Cl$	J_{14}
15	$HOCl + h\nu$	$\rightarrow$	$OH + Cl$	J_{15}
16	$N_2O_5 + h\nu$	$\rightarrow$	$NO_2 + NO_3$	J_{16}
17	$CH_3OOH + h\nu$	$\rightarrow$	$OH + CH_3O$	J_{17}
18	$CCl_4 + h\nu$	$\rightarrow$	$Cl + CCl_3$	J_{18}
19	$HNO_4 + h\nu$	$\rightarrow$	$HO_2 + NO_2$	J_{19}
20	$Cl_2 + h\nu$	$\rightarrow$	$Cl + Cl$	J_{21}
21	$NO_3 + h\nu$	$\rightarrow$	$NO_2 + O$	J_{22}
22	$ClOO + h\nu$	$\rightarrow$	$ClO + O$	J_{23}
23	$H_2O\ h\nu$	$\rightarrow$	$H + OH$	J_{24}
24	$OClO + h\nu$	$\rightarrow$	$O + ClO$	J_{25}
25	$Cl_2O_2 + h\nu$	$\rightarrow$	$Cl + ClOO$	J_{27}
26	$ClNO + h\nu$	$\rightarrow$	$Cl + NO$	J_{28}
27	$ClNO_2 + h\nu$	$\rightarrow$	products	J_{29}
28	$ClONO + h\nu$	$\rightarrow$	products	J_{30}

4.2. Mass Continuity Equations

In the current 1-D model, the chemical and physical processes determining the temporal variation in the concentration of the atmospheric constituent can be represented in mathematical form by the mass continuity equation,

$$\frac{\partial C_i(z,t)}{\partial t} = P_i(z) - L_i(z) \cdot C_i(z,t) - \frac{\partial \Phi_i(z,t)}{\partial z} \qquad (4.1)$$

where, $C_i(z,t)$ is the number density of *i*th species, $P_i(z)$ is the chemical production rate, $L_i(z)$ is the chemical loss rate, $\Phi_i(z,t)$ is the vertical flux of *i*th species at height z. In deriving this equation, a longitudinal and latitudinal global average of the transport flux is taken, and it is assumed that the resulting net vertical flux can be represented as a diffusive process. This equation is also valid if the horizontal transport terms are small compared to the production and loss terms in equation (4.1). And

$$-\frac{\partial \Phi_i(z,t)}{\partial z} = \frac{\partial}{\partial z}\left[K(z)N(z)\frac{\partial\left(\frac{C_i(z)}{N(z)}\right)}{\partial z}\right] \qquad (4.2)$$

Where, N(z) is the total air number density at height z, K(z) is eddy diffusion coefficient.

The diffusive treatment of transport in the 1-D model is a purely empirical representation. It does not utilize observed atmospheric motions directly but rather is based on the observations of the temporal and spatial distributions of selected tracers.

The K_z profiles used in different models have typically been based on chemical tracers such as CH_4 and N_2O and radionucleides from past nuclear tests. For such tracers, it is assumed that the value of K_z determined is a function of only the transporting motions, and within the scope of 1-D models is independent of both the details of the tracer field and the specific structure of the tracer source and sink distributions. While adopted profiles differ by as much as an order of magnitude at some altitudes, these K_z profiles tend to have similar characteristics: large values in the troposphere, much reduced values in the region near the tropopause, increasing with altitude in the stratosphere.

Generally, a single K_z profile is chosen to represent the transport of all species in the atmosphere. In practice, the representation of transport in the 1-D model has been limited by the lack of well-determined globally averaged data for the tracers and by uncertainties in understanding of the balances among sources, sinks and transport processes affecting the trace species distributions. Recent analyses suggest that it may not be possible to find a K_z at least with current chemistry that adequately fits atmospheric data for all long-lived tracers, such as N_2O and CH_4, at the same time (Wuebbles 1983b). Using a separate K_z for each species would provide additional variables in the model that would enable any observational data to be fit, but without necessarily representing any physical aspect of the atmosphere. Unless there is a well-defined basis for each species to have its own K_z it would not be reasonable to consider this (WMO 1985). According to this, a single K_z profile is chosen to represent the transport of all species in the current model. The profile is taken from Cicerone (1983) and shown in Figure 4.1.

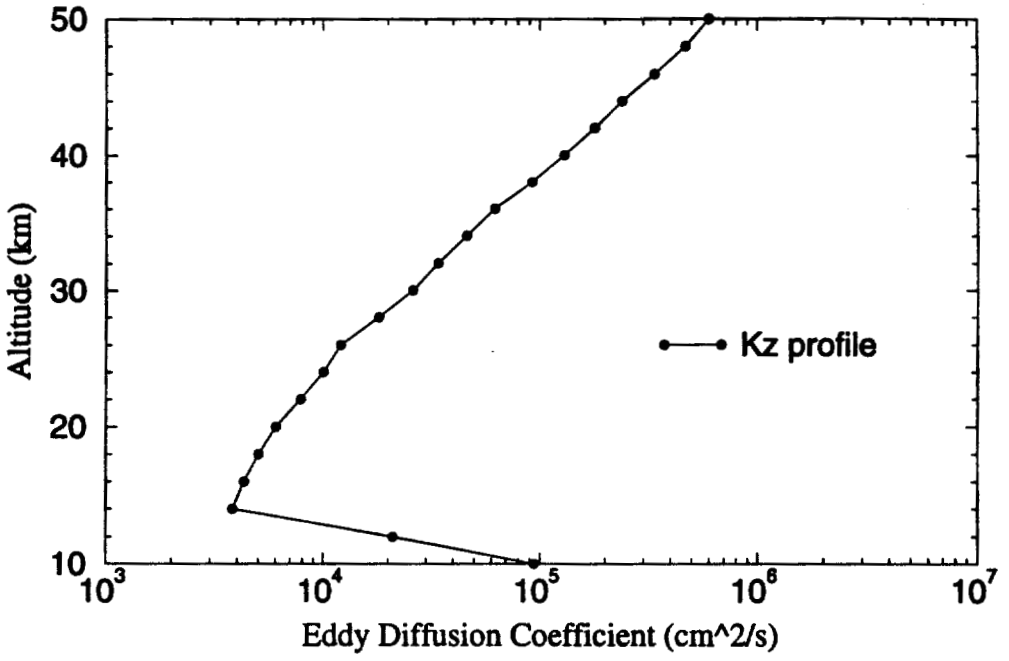

Figure 4.1 Vertical diffusion coefficient K_z profile (taken from Cicerone 1983).

The current ozone model is evenly spaced. Here we use Tran(z) to represent the flux term in the above equation. Then the mass continuity equations for all the unknown species are listed below:

$$d[O]/dt = 2J_1[O_2] + J_2[O_3] + J_4[NO_2] + J_{20}[ClO] + J_{21}[NO] + J_{22}[NO_3] - k_1[O][O_2] - k_2[O][O_3] + k_4[O^1D][N_2] + k_5[O^1D][O_2] + 2k_7[O^1D][O_3] - k_{11}[NO_2][O] - k_{16}[NO][O] - k_{29}[OH][O] - k_{30}[HO_2][O] - k_{31}[H_2O_2][O] + k_{40}[OH]^2 + k_{45}[H][HO_2] - k_{47}[H_2CO][O] - k_{59}[ClO][O] - k_{73}[ClONO_2][O] - k_{76}[H_2][O] - k_{82}[O][NO_2] - k_{83}[O][NO_3] - k_{84}[O][HO_2NO_2] - k_{86}[O][CH_3] - k_{100}[O][HCl] - k_{101}[O][HOCl] + Tran(z) \quad (4.3)$$

$$d[O^1D]/dt = J_3[O_3] + J_5[N_2O] - k_4[O^1D][N_2] - k_5[O^1D][O_2] - (k_6+k_7)[O^1D][O_3] - (k_8+k_9)[O^1D][N_2O] - k_{21}[O^1D][H_2O] - k_{22}[O^1D][CH_4] - k_{23}[O^1D][CH_4] - k_{24}[O^1D][H_2] - k_{54}[O^1D][CF_2Cl_2] - k_{55}[O^1D][CFCl_3] - k_{56}[O^1D][CH_3Cl] - k_{61}[O^1D][HCl] - k_{77}[O^1D][CCl_4] + Tran(z) \quad (4.4)$$

$$d[O_3]/dt = -(J_2+J_3)[O_3] + k_1[O][O_2] - k_2[O][O_3] - (k_6+k_7)[O^1D][O_3] - k_{10}[NO][O_3] - k_{14}[NO_2][O_3] - k_{25}[H][O_3] - k_{27}[OH][O_3] - k_{28}[HO_2][O_3] - k_{58}[Cl][O_3] - k_{87}[CH_3][O_3] + Tran(z) \quad (4.5)$$

$$d[H]/dt = J_9[H_2CO] + J_{14}[HCl] + J_{24}[H_2O] + k_{24}[O^1D][H_2] - k_{25}[H][O^1D] - k_{26}[H][O_2] + k_{29}[OH][O] + k_{34}[OH][CO] + k_{35}[OH][H_2] - (k_{44}+k_{45}+k_{46})[H][HO_2] + k_{64}[Cl][H_2] - k_{65}[HCl][H] - k_{71}[H][ClO] + k_{76}[H_2][O] - k_{85}[H][NO_2] + Tran(z) \quad (4.6)$$

$$d[OH]/dt = J_6[HNO_3] + 2J_7[H_2O_2] + J_{15}[HOCl] + J_{17}[CH_3OOH] + J_{24}[H_2O] + 2k_{21}[O^1D][H_2O] + k_{22}[O^1D][CH_4] + k_{24}[O^1D][H_2] + k_{25}[H][O_3] - k_{27}[OH][O_3] + k_{28}[HO_2][O_3] - k_{29}[OH][O] + k_{30}[HO_2][O] + k_{31}[H_2O_2][O] - k_{32}[OH][CH_4] + k_{33}[HO_2][NO] + k_{34}[OH][CO] - k_{35}[OH][H_2] - k_{36}[OH][NO_2] - k_{37}[OH][HNO_3] - k_{38}[OH][H_2O_2] - k_{39}[OH][HO_2] - k_{40}[OH]^2 - k_{41}[OH][H_2CO] - k_{43}[OH]^2 + 2k_{46}[H][HO_2] + k_{47}[H_2CO][O] - k_{51}[CH_3OOH][OH] - k_{57}[OH][CH_3Cl] + k_{61}[O^1D][HCl] - k_{67}[OH][HCl] + k_{71}[H][ClO] - k_{74}[ClONO_2][OH] + k_{76}[H_2][O] - k_{78}[OH][HOCl] - k_{81}[HO_2NO_2][OH] + k_{85}[H][NO_2] + k_{92}[Cl][HOCl] - k_{94}[ClO][OH] - k_{98}[OH][CFCl_3] - k_{99}[OH][CF_2Cl_2] + k_{100}[O][HCl] + k_{101}[O][HOCl] + Tran(z) \quad (4.7)$$

$$d[HO_2]/dt = J_{19}[HO_2NO_2] + k_{26}[H][O_2] + k_{27}[OH][O_3] - k_{28}[HO_2][O_3] - k_{30}[HO_2][O] + k_{31}[H_2O_2][O] - k_{33}[HO_2][NO] + k_{38}[OH][H_2O_2] - k_{39}[OH][HO_2] - k_{42}[HO_2]^2 - (k_{44}+k_{45}+k_{46})[H][HO_2] - k_{50}[CH_3O_2][HO_2] + k_{52}[CH_3O][O_2] + k_{53}[HCO][O_2] - k_{63}[Cl][HO_2] + k_{66}[Cl][H_2O_2] - k_{70}[HO_2][ClO] - k_{79}[HO_2][NO_2] - k_{97}[HO_2]^2 + Tran(z) \quad (4.8)$$

$$d[H_2O_2]/dt = -J_7[H_2O_2] - k_{31}[H_2O_2][O] - k_{38}[OH][H_2O_2] + k_{42}[HO_2]^2 + k_{43}[OH]^2 - k_{66}[Cl][H_2O_2] + k_{97}[HO_2]^2 + Tran(z) \quad (4.9)$$

$$d[NO]/dt = J_4[NO_2] + J_8[NO_3] + 2k_9[O^1D][N_2O] - k_{10}[NO][O_3] + k_{11}[NO_2][O] - k_{15}[NO][NO_3] - k_{16}[NO][O] - k_{33}[HO_2][NO] - k_{49}[CH_3O_2][NO] - k_{60}[ClO][NO] + k_{73}[ClONO_2][O] + k_{85}[H][NO_2] + Tran(z) \quad (4.10)$$

$$d[NO_2]/dt = -J_4[NO_2] + J_6[HNO_3] + J_{16}[N_2O_5] + J_{19}[HO_2NO_2] + J_{22}[NO_3] + k_{10}[NO][O_3] - k_{11}[NO_2][O] - k_{14}[NO_2][O_3] + 2k_{15}[NO][NO_3] + k_{16}[NO][O] - k_{17}[NO_2][NO_3] + k_{33}[HO_2][NO] - k_{36}[OH][NO_2] + k_{49}[CH_3O_2][NO] + k_{60}[ClO][NO] - k_{72}[ClO][NO_2] - k_{79}[HO_2][NO_2] + k_{81}[HO_2NO_2][OH] - k_{82}[O][NO_2] + k_{83}[O][NO_3] - k_{85}[H][NO_2] + k_{93}[Cl][NO_3] + Tran(z) \quad (4.11)$$

$$d[NO_3]/dt = -J_8[NO_3] + J_{16}[N_2O_5] - J_{22}[NO_3] + k_{14}[NO_2][O_3] = k_{15}[NO][NO_3] - k_{17}[NO_2][NO_3] + k_{37}[OH][HNO_3] + k_{74}[ClONO_2][OH] + k_{82}[O][NO_2] - k_{83}[O][NO_3] - k_{93}[Cl][NO_3] + Tran(z) \quad (4.12)$$

$$d[N_2O_5]/dt = -J_{16}[N_2O_5] + k_{17}[NO_2][NO_3] + Tran(z) \quad (4.13)$$

$$d[HNO_3]/dt = -J_6[HNO_3] + k_{36}[OH][NO_2] - k_{37}[OH][HNO_3] + Tran(z) \quad (4.14)$$

$$d[HO_2NO_2]/dt = -J_{19}[HO_2NO_2] + k_{79}[HO_2][NO_2] - k_{81}[HO_2NO_2][OH] - k_{84}[O][HO_2NO_2] + Tran(z) \quad (4.15)$$

$$d[Cl]/dt = J_{11}[CF_2Cl_2] + J_{12}[CFCl_3] + J_{13}[ClONO_2] + J_{14}[HCl] + J_{15}[HOCl] + J_{18}[CCl_4] + J_{20}[ClO] + 2J_{21}[Cl_2] + J_{27}[Cl_2O_2] + J_{28}[ClNO] + k_{56}[O^1D][CH_3Cl] + k_{57}[OH][CH_3Cl] - k_{58}[Cl][O_3] + k_{59}[ClO][O] + k_{60}[ClO][NO] + k_{61}[O^1D][HCl] + k_{62}[Cl][CH_4] - k_{63}[Cl][HO_2] - k_{64}[Cl][H_2] + k_{65}[HCl][H] - k_{66}[Cl][H_2O_2] + k_{67}[OH][HCl] - k_{68}[Cl][O_2] + k_{71}[H][ClO] - k_{75}[H_2CO][Cl] - k_{91}[Cl][CH_3Cl] - k_{92}[Cl][HOCl] - k_{93}[Cl][NO_3] + k_{100}[O][HCl] - k_{103}[Cl][HO_2] - k_{105}[Cl][OClO] - (k_{106} + k_{107})[Cl][ClOO] - k_{108}[Cl][Cl_2O_2] - k_{109}[Cl][ClONO_2]\ k_{110}[Cl][ClNO] + k_{117}[OH][Cl_2] - k_{124}[Cl][NO] - (k_{125} + k_{126})[Cl][NO_2] - k_{127}[Cl][CO] + Tran(z) \quad (4.16)$$

$$d[ClO]/dt = -J_{20}[ClO] + J_{23}[ClOO] + J_{25}[OClO] + k_{54}[O^1D][CF_2Cl_2] + k_{55}[O^1D][CFCl_3] + k_{58}[Cl][O_3] - k_{59}[ClO][O] - k_{60}[ClO][NO] - k_{70}[HO_2][ClO] - k_{71}[H][ClO] - k_{72}[ClO][NO_2] + k_{73}[ClONO_2][O] + k_{77}[O^1D][CCl_4] + k_{78}[OH][HOCl] + k_{93}[Cl][NO_3] - k_{94}[ClO][OH] - k_{95}[ClO][CH_4] - k_{96}[ClO][H_2] + k_{103}[Cl][HO_2] + 2k_{105}[Cl][OClO] + 2k_{107}[Cl][ClOO] - k_{111}[ClO][NO_3] - k_{112}[ClO][H_2CO] - k_{113}[ClO][CO] - k_{114}[ClO][N_2O] - k_{115}[ClO]^2 - k_{116}[ClO][O_3] + k_{120}[OClO][O] + k_{123}[OClO][NO] - k_{128}[ClO]^2 + Tran(z) \quad (4.17)$$

$$d[HOCl]/dt = -J_{15}[HOCl] + k_{70}[HO_2][ClO] + k_{74}[ClONO_2][OH] - k_{78}[OH][HOCl] - k_{101}[O][HOCl] + Tran(z) \quad (4.18)$$

$$d[ClONO_2]/dt = -J_{13}[ClONO_2] + k_{72}[ClO][NO_2] - k_{73}[ClONO_2][O] - k_{74}[ClONO_2][OH] - k_{109}[Cl][ClONO_2] + Tran(z) \quad (4.19)$$

$$d[CH_3]/dt = J_{20}[CH_3O_2] + k_{22}[O^1D][CH_4] + k_{32}[OH][CH_4] - k_{48}[CH_3][O_2] + k_{62}[Cl][CH_4] - k_{86}[O][CH_3] - k_{87}[O_3][CH_3] + Tran(z) \quad (4.20)$$

$$d[CH_3O]/dt = J_{17}[CH_3OOH] + k_{49}[CH_3O_2][NO] - k_{52}[CH_3O][O_2] + 2k_{88}[CH_3O_2]^2 + Tran(z) \quad (4.21)$$

$$d[CH_3O_2]/dt = k_{48}[CH_3][O_2] - k_{49}[CH_3O_2][NO] - k_{50}[CH_3O_2][HO_2] + k_{51}[CH_3OOH][OH] - (k_{88}+k_{89}+k_{90})[CH_3O_2]^2 + Tran(z) \quad (4.22)$$

$$d[CH_3OOH]/dt = -J_{17}[CH_3OOH] + k_{50}[CH_3O_2][HO_2] - k_{51}[CH_3OOH][OH] + Tran(z) \quad (4.23)$$

$$d[H_2CO]/dt = -(J_9+J_{10})[H_2CO] + k_{23}[O^1D][CH_4] - k_{41}[OH][H_2CO] - k_{47}[H_2CO][O] + k_{52}[CH_3O][O_2] - k_{75}[H_2CO][Cl] + k_{89}[CH_3O_2]^2 + Tran(z) \quad (4.24)$$

$$d[HCl]/dt = -J_{14}[HCl] - k_{61}[O^1D][HCl] + k_{62}[Cl][CH_4] + k_{63}[Cl][HO_2] + k_{64}[Cl][H_2] - k_{65}[HCl][H] + k_{66}[Cl][H_2O_2] - k_{67}[OH][HCl] + k_{75}[H_2CO][Cl] + k_{91}[Cl][CH_3Cl] - k_{100}[O][HCl] + Tran(z) \quad (4.25)$$

$$d[Cl_2]/dt = -J_{21}[Cl_2] - k_{102}[O^1D][Cl_2] + k_{110}[Cl][ClNO] - k_{117}[OH][Cl_2] + k_{92}[Cl][HOCl] + Tran(z) \quad (4.26)$$

$$d[OClO]/dt = -J_{25}[OClO] - k_{105}[Cl][OClO] + k_{116}[ClO][O_3] - k_{120}[OClO][O] - k_{121}[OClO][O_3] - k_{122}[OClO][OH] - k_{123}[OClO][NO] + Tran(z) \quad (4.27)$$

$$d[ClOO]/dt = -J_{23}[ClOO] - (k_{106} + k_{107})[Cl][ClOO] + k_{68}[Cl][O_2] + Tran(z) \quad (4.28)$$

$$d[Cl_2O_2]/dt = -J_{27}[Cl_2O_2] - k_{108}[Cl][Cl_2O_2] + k_{128}[ClO]^2 + Tran(z) \quad (4.29)$$

$$d[ClNO]/dt = -J_{28}[ClNO] - k_{110}[Cl][ClNO] + k_{124}[Cl][NO] + Tran(z) \quad (4.30)$$

$$d[ClNO_2]/dt = -J_{29}[ClNO_2] - k_{118}[OH][ClNO_2] + k_{126}[Cl][NO_2] + Tran(z) \quad (4.31)$$

$$d[ClONO]/dt = -J_{30}[ClONO] + k_{125}[Cl][NO_2] + Tran(z) \quad (4.32)$$

The flux term Tran(z) can be expanded as: (where k represents space point)

$$Tran(z) = \frac{\left[K(z)N(z)\dfrac{\partial\left(\dfrac{C_i(z)}{N(z)}\right)}{\partial z}\right]_{+\frac{1}{2}} - \left[K(z)N(z)\dfrac{\partial\left(\dfrac{C_i(z)}{N(z)}\right)}{\partial z}\right]_{-\frac{1}{2}}}{\Delta z} \tag{4.33}$$

$$Tran(z) = \frac{K_{k+\frac{1}{2}}(z)N_{k+\frac{1}{2}}(z)\left(\left(\dfrac{C_i(z)}{N(z)}\right)_{k+1} - \left(\dfrac{C_i(z)}{N(z)}\right)_{k}\right)}{\Delta z^2} - \frac{K_{k-\frac{1}{2}}(z)N_{k-\frac{1}{2}}(z)\left(\left(\dfrac{C_i(z)}{N(z)}\right)_{k} - \left(\dfrac{C_i(z)}{N(z)}\right)_{k-1}\right)}{\Delta z^2} \tag{4.34}$$

$$Tran(z) = \frac{K_{k+\frac{1}{2}}(z)N_{k+\frac{1}{2}}(z)\left(\dfrac{C_{i,k+1}(z)}{N_{k+1}(z)} - \dfrac{C_{i,k}(z)}{N_k(z)}\right)}{\Delta z^2} - \frac{K_{k-\frac{1}{2}}(z)N_{k-\frac{1}{2}}(z)\left(\dfrac{C_{i,k}(z)}{N_k(z)} - \dfrac{C_{i,k-1}(z)}{N_{k-1}(z)}\right)}{\Delta z^2} \tag{4.35}$$

$$Tran(z) = \left[\frac{K_{k+\frac{1}{2}}(z)}{\Delta z^2}\frac{N_{k+\frac{1}{2}}(z)}{N_{k+1}(z)}\right]C_{i,k+1}(z)$$

$$-\left[\frac{K_{k+\frac{1}{2}}(z)}{\Delta z^2}\frac{N_{k+\frac{1}{2}}(z)}{N_k(z)} - \frac{K_{k-\frac{1}{2}}(z)}{\Delta z^2}\frac{N_{k-\frac{1}{2}}(z)}{N_k(z)}\right]C_{i,k}(z) + \left[\frac{K_{k-\frac{1}{2}}(z)}{\Delta z^2}\frac{N_{k-\frac{1}{2}}(z)}{N_{k-1}(z)}\right]C_{i,k-1}(z) \quad (4.36)$$

$$= \alpha C_{i,k+1}(z) - \beta C_{i,k}(z) + \gamma C_{i,k-1}(z)$$

According to Equations (4.1) and (4.36), the generic finite difference equation can be derived.

$$\frac{C_{i,k,j+1} - C_{i,k,j}}{\Delta t} = P - L C_{i,k,j+1} + (\alpha C_{i,k+1,j} - \beta C_{i,k,j+1} + \gamma C_{i,k-1,j+1}) \quad (4.37)$$

$$C_{i,k,j+1} = \frac{C_{i,k,j} + \Delta t(P + \alpha C_{i,k+1,j} + \gamma C_{i,k-1,j+1})}{1 + \Delta t(\beta + L)}$$

where i represents species, k represents space point, j for time step.

4.3. Absorption Bands for oxygen and ozone

Oxygen and ozone are the principal absorbers of solar radiation in the ultraviolet spectral region. An understanding of these spectral regions is essential in the modeling of stratospheric chemistry.

Molecular oxygen strongly absorbs the ultraviolet light at wavelengths below 220 nm. Figure 4.2 shows the absorption features of oxygen, i.e. the Herzberg continuum, the Schumann-Runge bands, and the Schumann continuum.

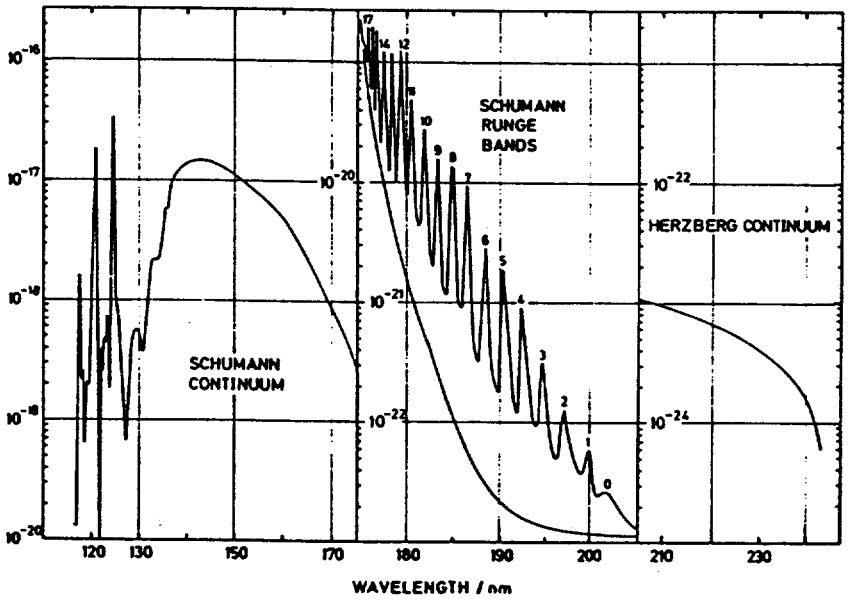

Figure 4.2 Ultraviolet absorption spectrum of molecular oxygen, with cross sections given in units of cm^2/molecule (taken from Warneck, 1988).

The absorption spectrum of O_3 can be categorized as Hartley bands, Huggins bands, and Chappius bands. The Hartley bands are from 200 to 300 nm. The absorption by O_3 in the Hartley bands is very strong in the stratosphere and it controls the short-wavelength limit of light that reaches the troposphere. The absorption region 300-360 nm is called the Huggins bands. And the Chappius bands is in the region 440-850 nm. Figure 4.3, and Figure 4.4 show these bands.

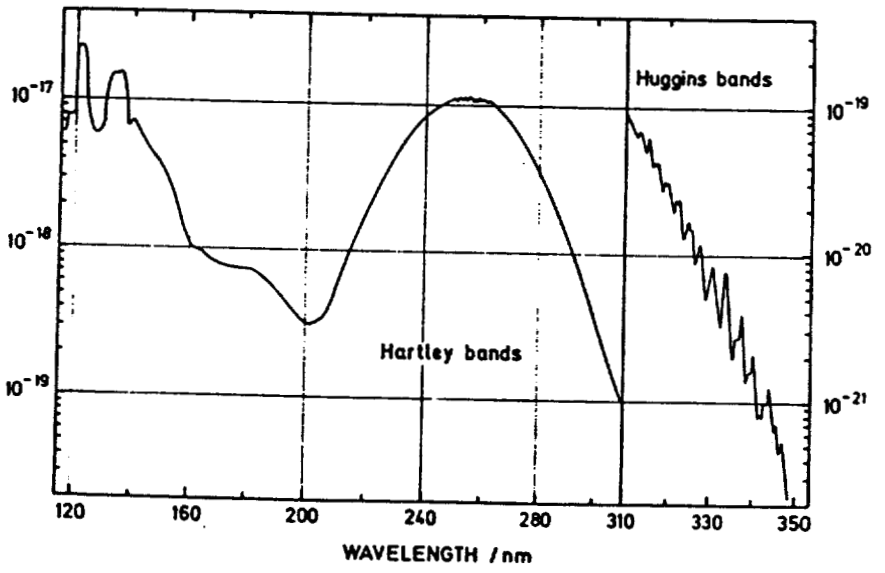

Figure 4.3 Absorption spectrum of O_3 in the wavelength region 115-350 nm, with cross sections given in units of cm^2/molecule (taken from Warneck, 1988).

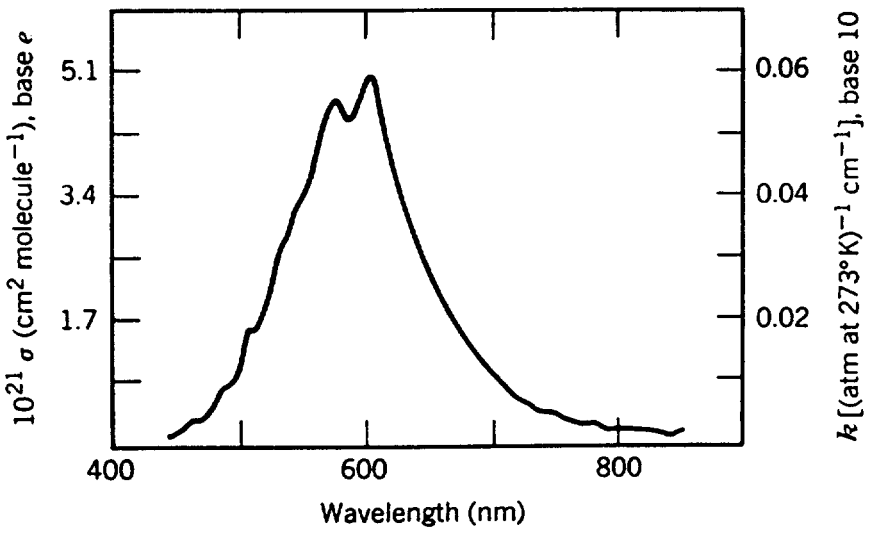

Figure 4.4 Absorption spectrum of O_3 known as the Chappius bands (taken from Finlayson-Pitts and Pitts, 1986).

4.4. Actinic Flux

Solar radiation is the driving force for the photodissociation processes and initiates the chemical reactions in the atmosphere. The sun is located 1.5 x 10^8 km from the earth's surface. It can be considered a spherical light source of diameter 1.4 million kilometer. The solar constant is the total intensity of sunlight outside the earth's atmosphere and is defined as the total amount of light received per unit area normal to the direction of propagation of the light. The value is 1340 watts/m^2 or 1.92 cal/cm^2-min. Outside the atmosphere, the solar flux approximates blackbody emission at 6000°K. But in the atmosphere, light absorption and scattering by atmospheric gases and particles attenuates the solar radiation. For atmospheric photochemistry, the solar flux per unit interval of wavelength is of direct interest. Figure 4.6 shows the solar flux as a function of wavelength outside the atmosphere and at sea level for a solar zenith angle of zero.

Actinic flux is the integrated radiation from all directions to a point. It includes direct radiation and diffuse radiation. In the current model, we use the two-stream approximation (Thompson, 1984; Luther, 1980) for the diffuse flux and consider the attenuation of direct flux from absorption of O_2 and O_3.

The angle of the sun relative to a fixed point on the surface of the earth directly effects the pathlength through which the light passes. The angle is characterized by the solar zenith angle θ, which is defined as the angle between the direction of the sun and the vertical. The figure below shows the definition of the solar zenith angle. It is a function of time of day, latitude, and season. The function can be expressed as:

$$\cos\theta = \sin i \sin l + \cos i \cos l \cos h \qquad (4.37)$$

where i is the solar inclination, l is the latitude, h is the hour angle. The inclination i varies from 23°27' on June 21 to -23°27' on December 22 and is independent of location. It is only a function of the day of year. The hour angle h is zero at solar noon and increases by 15° for every hour. Let's use t to represent the particular time of a day at a location. We can express t as:

$$t=12-\frac{1}{15}\cos^{-1}\left(\frac{\cos\theta-\sin i\sin l}{\cos i\cos l}\right), \quad t\le 12 \tag{4.38}$$

$$t=12+\frac{1}{15}\cos^{-1}\left(\frac{\cos\theta-\sin i\sin l}{\cos i\cos l}\right), \quad t>12 \tag{4.39}$$

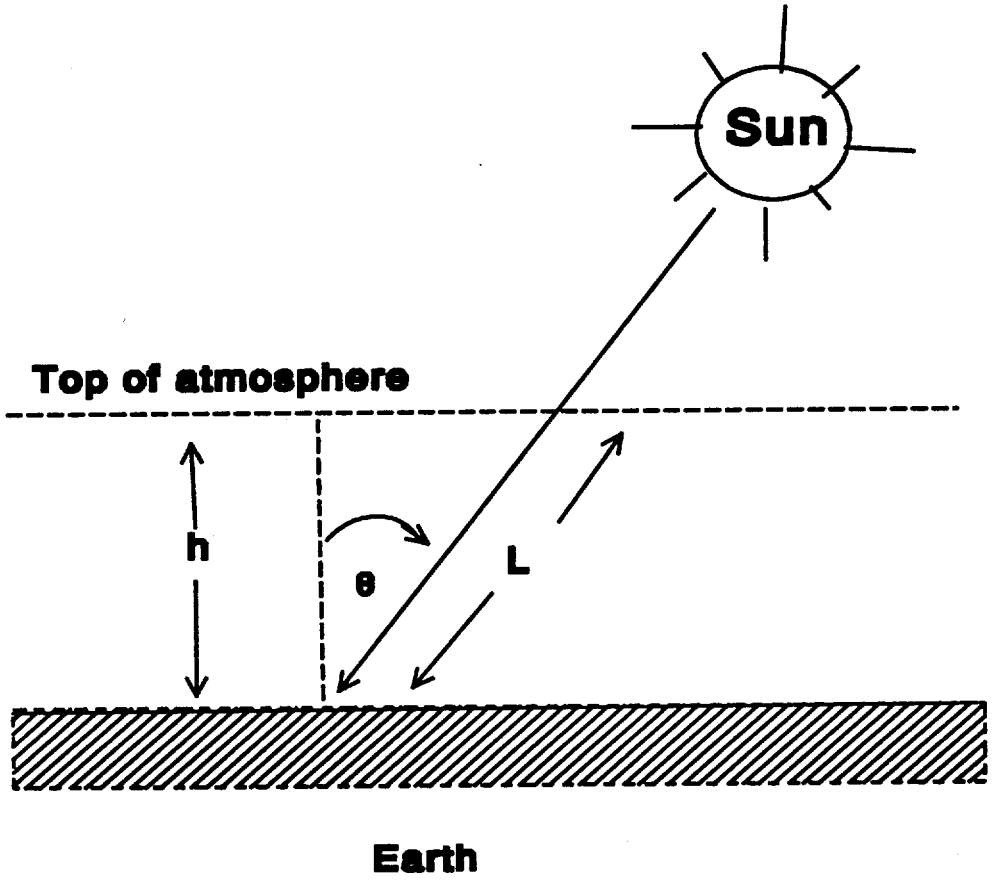

Figure 4.5 Definition of solar zenith angle θ.

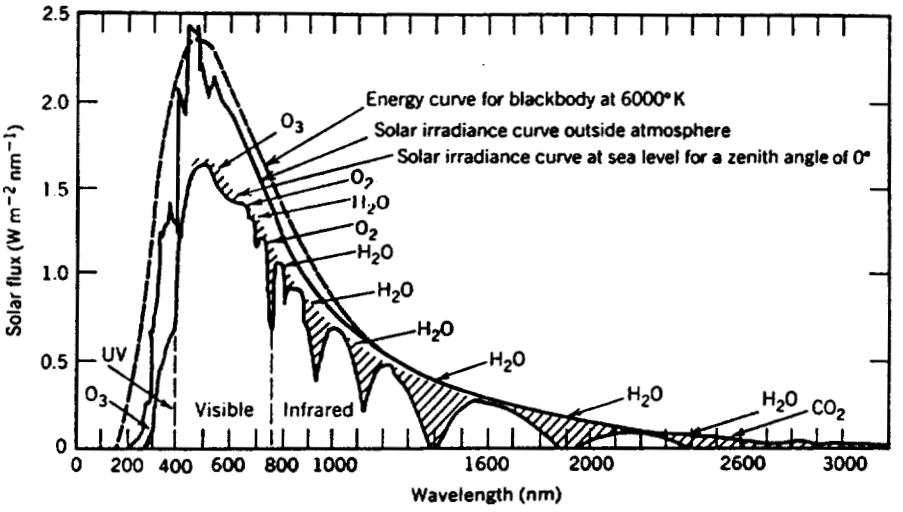

Figure 4.6 Solar flux outside the atmosphere and at sea level, respectively. The emission of a blackbody at 6000°K is also shown for comparison (taken from Finlayson-Pitts and Pitts, 1986).

4.5. Two-stream Method Solving for Radiation Field

The Beer-Lambert law states that the decrease of the radiant intensity traversing a homogeneous medium with a certain path length dz is proportional to the amount of substance in the path and radiant intensity:

$$dI_\lambda = -\sigma_\lambda M I_\lambda dz \tag{4.40}$$

where I_λ is the radiation intensity at wavelength λ, M is the number density, and σ is the absorption cross section.

The optical depth τ is an optical property which is defined as:

$$d\tau = -\sigma M dz \tag{4.41}$$

The method to calculate the radiative field is modified from Thompson (1984). Luther (1980) developed a simple two-stream algorithm for computing radiation with ozone absorption and Rayleigh scattering that gives photodissociation rates in good agreement with those obtained from highly accurate calculations, such as Demerjian et al., 1980; Logan et al., 1981; Augustsson and Levine, 1982. This paper uses the Luther's scheme; but Thompson's symbols are adopted here.

The total solar flux density ($cm^{-2}s^{-1}$) can be expressed as a sum of direct and diffuse radiation:

$$F_{total} = F_{direct} + F_{diffuse} \tag{4.42}$$

The direct flux F_{direct} at a point in the atmosphere is calculated from the top atmospheric solar flux $F_0(\lambda)$ reduced by ozone absorption throughout an entire overhead column. Figure 4.7 shows the radiation scheme. The atmosphere is treated as plane-parallel. Between grid points k and k+1 the direct beam is attenuated by absorbing gases and molecular extinction. The direct solar flux can be expressed as:

$$F_{dir}^{k}(\lambda) = F_{dir}^{k+1}(\lambda) \exp(-\sigma_{O3}(\lambda,T)[O_3]_{k,k+1}^{eff} - \sigma_{Ray}(\lambda)[M]_{k,k+1}^{eff}) \tag{4.43}$$

where $\sigma_{O3}(\lambda,T)$ is ozone absorption cross section, $\sigma_{Ray}(\lambda)$ is Rayleigh scattering cross section, $[O_3]_{k,k+1}^{eff}$ and $[M]_{k,k+1}^{eff}$) is the effective ozone and molecular column depths (cm^{-2}) along the beam path in the space between k and k+1, which are equal to the column depths times secθ.

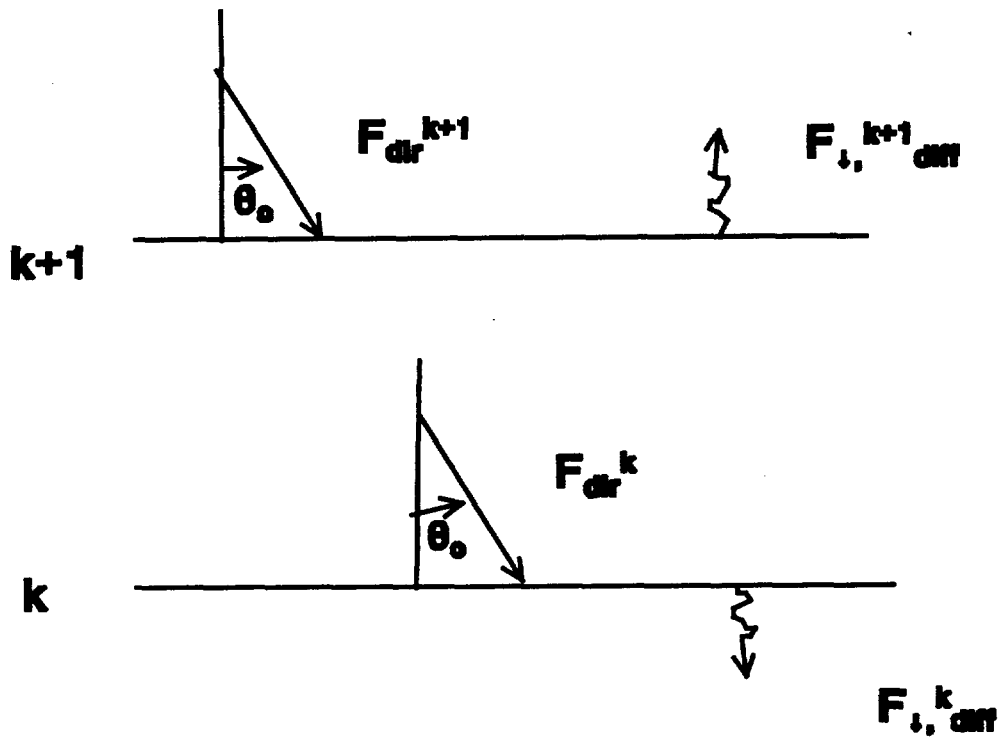

Figure 4.7 Transmission of direct solar flux (F_{dir}, incident angle θ_0) and upward and downward diffuse fluxes between several levels of plane-parallel atmosphere.

The diffuse flux includes contributions from molecular and aerosol scattering and radiation reflected from the surface and clouds. It is strongly altitude dependent. Luther's scheme is based on the conservation of radiant energy with an assumption of two-stream isotropic scattering. Consider the layer k-1, k, k+1, as given in Figure 4.7. The downward diffuse flux density at level k is the sum of diffuse radiation transmitted from above plus radiation scattered downward out of the downward direct (F_{dir}) and diffuse beams as they pass through the layer:

$$F_{\downarrow,diff}^{\ k} = f_{t,diff;k+1,k}\, F_{\downarrow,diff}^{\ k+1} + f_{s,dir;k+1,k}\, F_{dir}^{\ k+1} \cos\theta + 0.5 f_{s,diff;k+1,k}(F_{\uparrow,diff}^{\ k} + F_{\downarrow,diff}^{\ k}) \tag{4.44}$$

We have the factor of 0.5 in the above equation since Rayleigh scattering diffuse beams are divided equally between upward (or backward, $\uparrow$) and downward (or forward, $\downarrow$) directions. For the upward diffuse flux density at level k, we have:

$$F_{\uparrow,diff}^{\ k} = f_{t,diff;k-1,k}\, F_{\downarrow,diff}^{\ k-1} + f_{s,dir;k+1,k}\, F_{dir}^{\ k+1} \cos\theta + 0.5 f_{s,diff;k-1,k}(F_{\uparrow,diff}^{\ k-1} + F_{\downarrow,diff}^{\ k}) \tag{4.45}$$

In the above two equations, The symbols $f_{t,diff}$, $f_{s,dir}$, and $f_{s,diff}$ are the fractional transmission (t) and scattering (s) of each flux.

$$f_{t,diff;k+1,k} = \exp[-(\Delta\tau_{k+1,k}^{\ O3} + \Delta\tau_{k+1,k}^{\ Ray})AM_{diff}] \tag{4.46}$$

is the fraction transmitted through layer k, k+1 with ozone and Rayleigh normal optical depths $\Delta\tau_{k+1,k}^{\ O3}$ and $\Delta\tau_{k+1,k}^{\ Ray}$, respectively; AM_{diff} is the air mass through which diffuse radiation travels. The optical depth is equal to cross section times column depth.

$$f_{t,dir;k+1,k} = \exp[-(\Delta\tau_{k+1,k}^{\ O3} AM_{dir}) \cdot [1 - \exp(-\Delta\tau_{k+1,k}^{\ Ray} AM_{dir})] \tag{4.47}$$

is the fraction of flux scattered out of the direct beam and AM_{dir} is the air mass traveled by the direct beam. Here we use

$$AM_{dir} = 35/[1224(\cos\theta)^2 + 1]^{1/2} \tag{4.48}$$

$$f_{t,diff;k+1,k} = \exp[-(\Delta\tau_{k+1,k}^{O3} AM_{diff}) \bullet [1 - \exp(-\Delta\tau_{k+1,k}^{Ray} AM_{diff})] \quad (4.49)$$

is the fraction of flux scattered out of the diffuse beam.

At the surface, downward flux has the same expression. But upward flux is different because of ground albedo.

$$F_{\uparrow,diff}^{surf} = Alb\ (2F_{dir}^{surf} \cos\theta + F_{\downarrow,diff}^{surf}) \quad (4.50)$$

where Alb is the surface albedo.

The table 4.2 below illustrates the fraction coefficients.

Table 4.2 Fraction coefficients

fraction	Direct flux	Diffuse flux, ↓	Diffuse flux, ↑
Transmitted		$f_{t,diff;k+1,k}$	
Scattered	$f_{s,dir;k+1,k}$	$f_{s,diff;k+1,k}$	

Rearranging the above equations for downward and upward flux, we have:

$$F_{\downarrow,diff}^{k} - 0.5\ f_{s,diff;k+1,k}\ F_{\uparrow,diff}^{k} - (f_{t,diff;k+1,k} + 0.5\ f_{s,diff;k+1,k}) F_{\downarrow,diff}^{k+1} = f_{s,dir;k+1,k}\ F_{dir}^{k+1} \cos\theta \quad (4.51)$$

and

$$F_{\downarrow,diff}^{k} - 0.5\ f_{s,diff;k+1,k}\ F_{\uparrow,diff}^{k} - (f_{t,diff;k+1,k} + 0.5\ f_{s,diff;k+1,k}) F_{\downarrow,diff}^{k+1} = f_{s,dir;k+1,k}\ F_{dir}^{k+1} \cos\theta \quad (5.52)$$

4.6. Model Structure and Results

This model calculates the vertical distribution of stratospheric ozone and related atmospheric trace species. It includes chemical interactions, atmospheric attenuation of solar radiation and the effect of atmospheric transport. Figure 4.8 shows the general flow structure for the current ozone model. First, we change all the equations from mass balance conservation into finite difference expressions. Here an implicit finite difference approximation is used. After substituting all the reaction rate constants K and photodissociation rates J and solving the equations iteratively, we will get the output of all the trace species of interest.

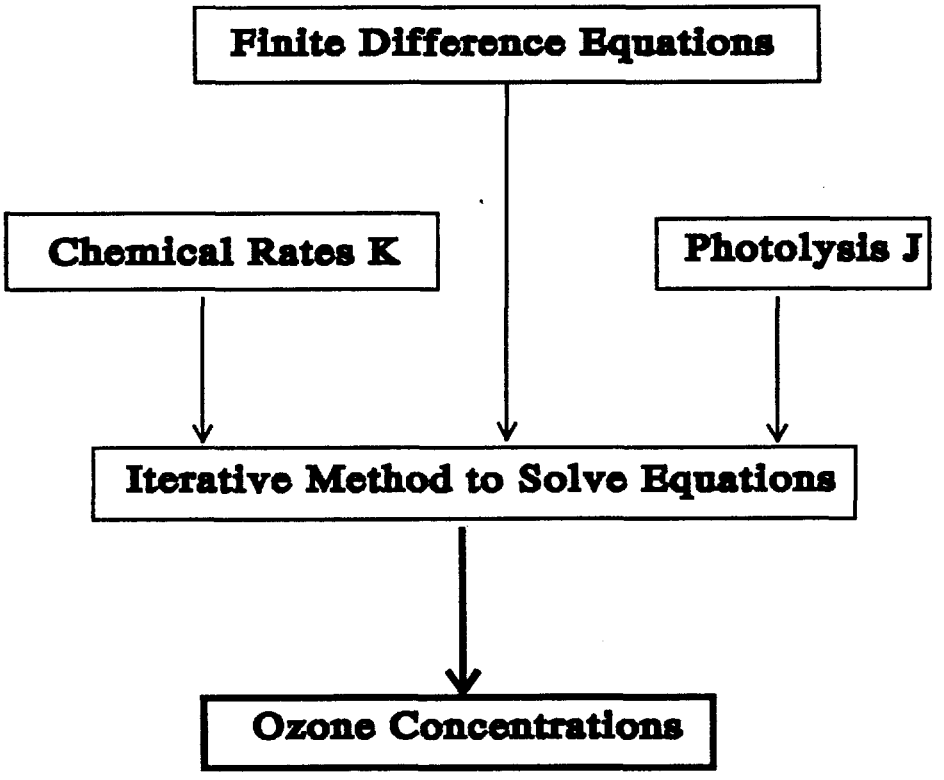

Figure 4.8 Flow chart for the current ozone model.

As for the K values, all the data are listed in the table of chemical reactions. Most of the calculation task is to get J values. Figure 4.9 shows the process of computing J and elements of calculating actinic flux. Absorption cross sections, quantum yields, and actinic flux are the three elements needed to know photodissociation rate. Among these, absorption cross sections and quantum yields are measured in the laboratory but it is impossible to measure the actinic flux for each wavelength and each space grid. The actinic flux is calculated by the two-stream method described in the previous section. Molecular scattering, absorptions by oxygen and ozone molecules, and ground albedo are taken into account. Clouds and aerosols are important to lower atmospheric chemistry mainly troposphere and are not considered here, since this model focuses on the stratospheric chemistry.

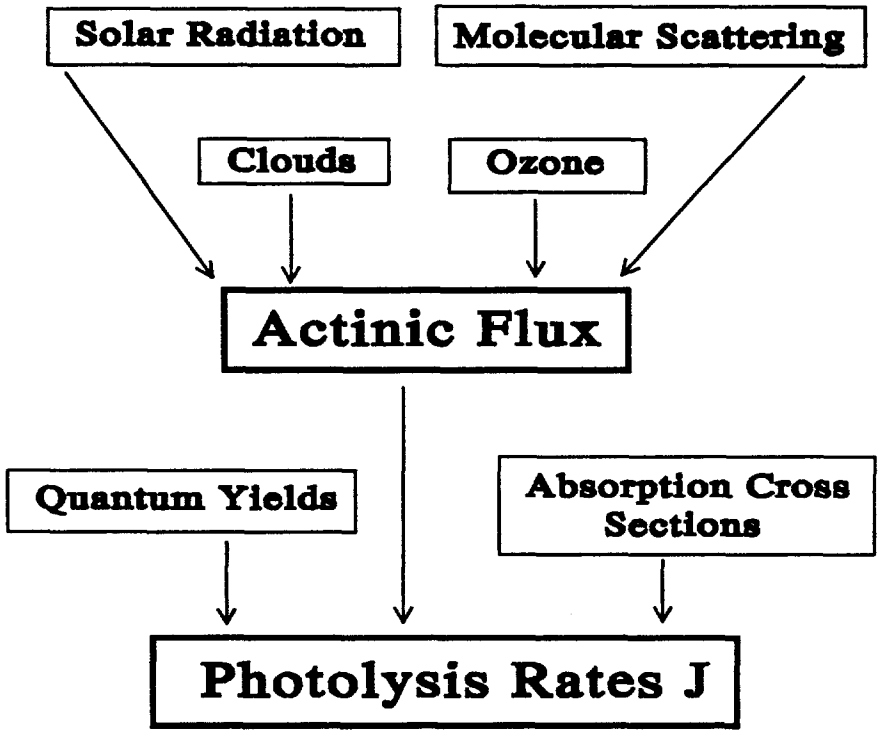

Figure 4.9 Flow chart for photodissociation rates and actinic flux.

The current model has a vertical span of 10 kilometer to 50 kilometer with 2 kilometer step. It produces good vertical profiles of ozone and other trace species. Oxygen cycle is the base of ozone model but it reveals only the correct height of the ozone layer. Based on this, the Hydrogen, Nitrogen and Chlorine cycles are added step by step. Through this process, several interesting results can be seen. Figure 4.10 shows the comparison of Oxygen cycle and Hydrogen cycle. We can see that hydrogen species decrease ozone in every atmospheric layer. Adding Hydrogen cycle (HOx) to the Oxygen cycle (Ox), the total column decrease of ozone is about 45 percent.

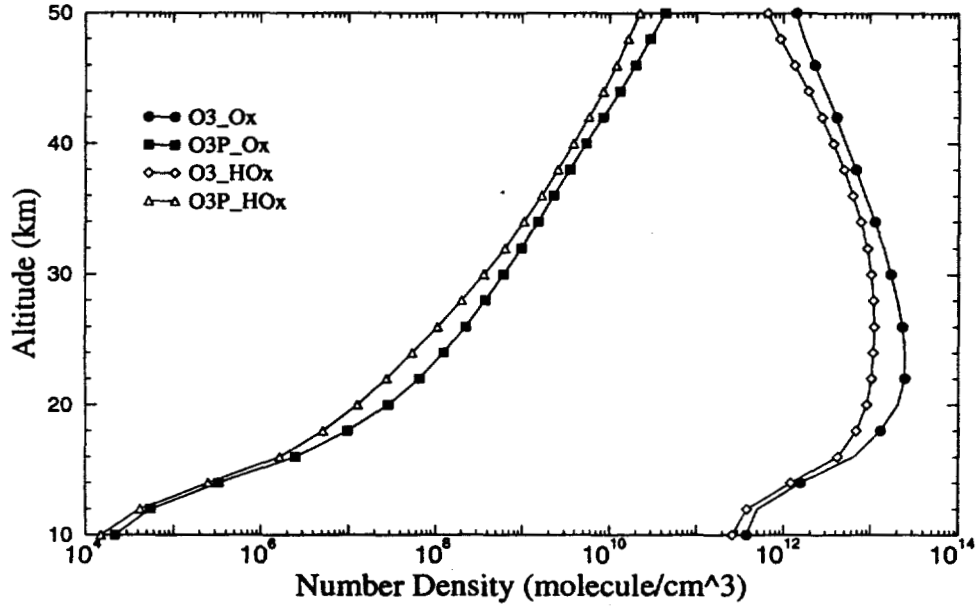

Figure 4.10 Comparison of Oxygen cycle with Hydrogen cycle. Hydrogen decreases ozone in every layer and the total column decrease is about 45 percent. Ox represents the oxygen cycle and HOx is for the hydrogen cycle.

Adding nitrogen species (NOx) causes ozone depletion from about 15 km to 40km. Figure 4.11 shows the effect of nitrogen cycle added on Ox + HOx cycle. The graph also shows a little increase of ozone below about 15 km. At lower altitudes, nitrogen dioxide becomes an important species which dissociates and produces oxygen atom and it causes such an increase in O_3. The total column decrease of ozone for adding nitrogen cycle on hydrogen cycle is about 48 percent relative to the oxygen and hydrogen cycles together.

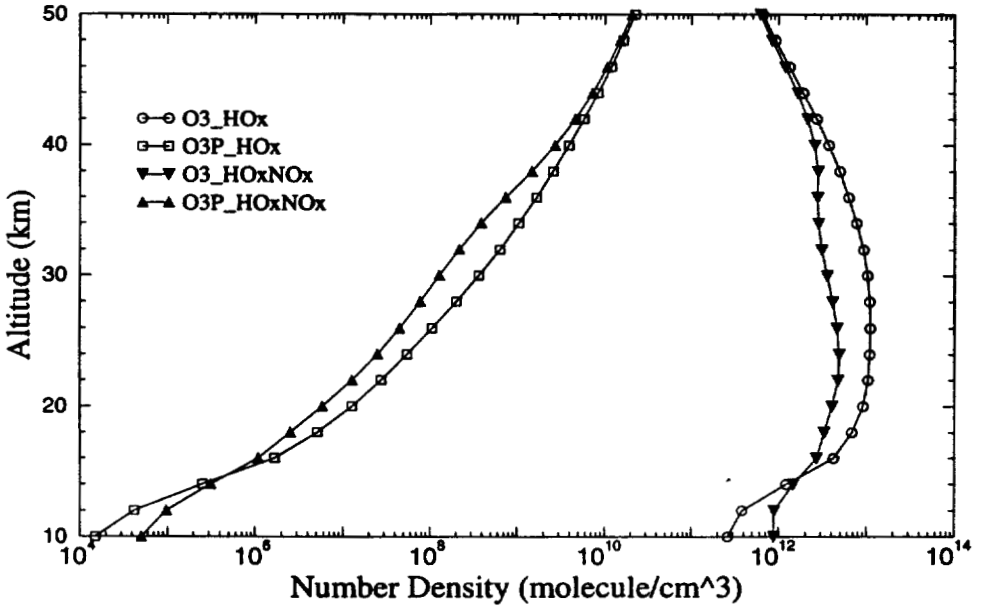

Figure 4.11 Effect of nitrogen cycle. Nitrogen cycle is added on hydrogen cycle and the total column decrease of ozone is about 48 percent. Here HOxNOx represents nitrogen cycle together with hydrogen cycle.

Then chlorine cycle is added on the previous nitrogen cycle. Figure 4.12 shows the chlorine cycle effect. The concentration values here and previous graphs are daily averages. Figure 4.12 shows that chlorine cycle only depletes ozone above 35 kilometer. ClOx depletes ozone down at 25 km as well but such depletion is not as significant as above 35 km. The ozone profile shows that the ozone maximum occurs at about 25 kilometers. The total column decrease of ozone from the addition of chlorine cycle is about 10 percent relative to the oxygen, hydrogen, nitrogen cycles together.

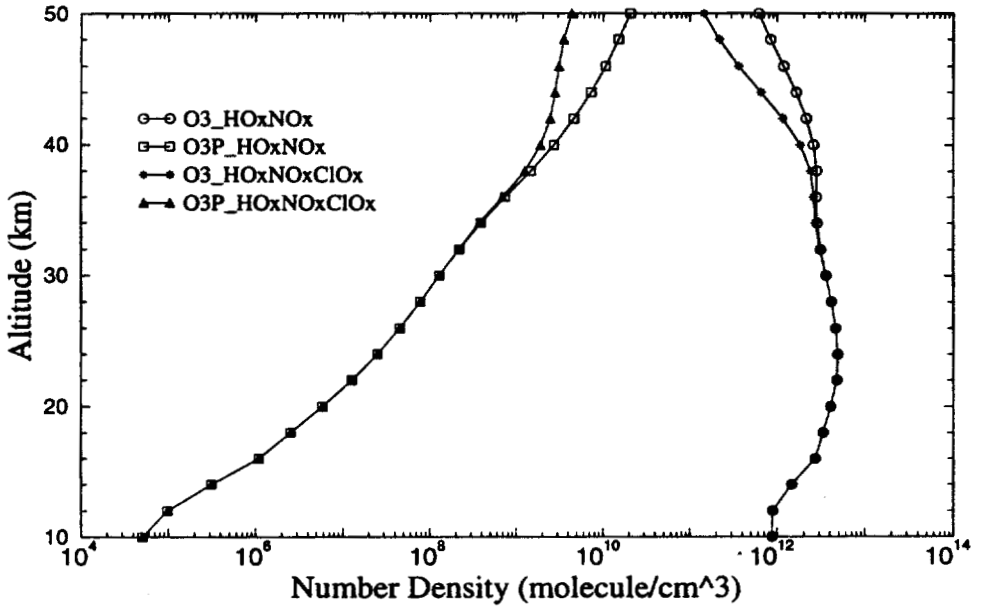

Figure 4.12 Effect of chlorine cycle. Based on the nitrogen cycle, the total column decrease of ozone the chlorine cycle is about 10 percent. Chlorine species only deplete ozone above 35 kilometer. HOxNOxClOx means all the three cycles.

An oxygen atom combines with an oxygen molecule and thus produces an ozone molecule. This is the only way to produce ozone. Oxygen molecules are abundant in the atmosphere and so ozone concentration is closely related to the concentration of oxygen atoms; More oxygen atoms, more ozone. Figure 4.13 includes all the graphs from Figure 4.10 to Figure 4.12. We can see the same pattern of variation for oxygen atom and for ozone.

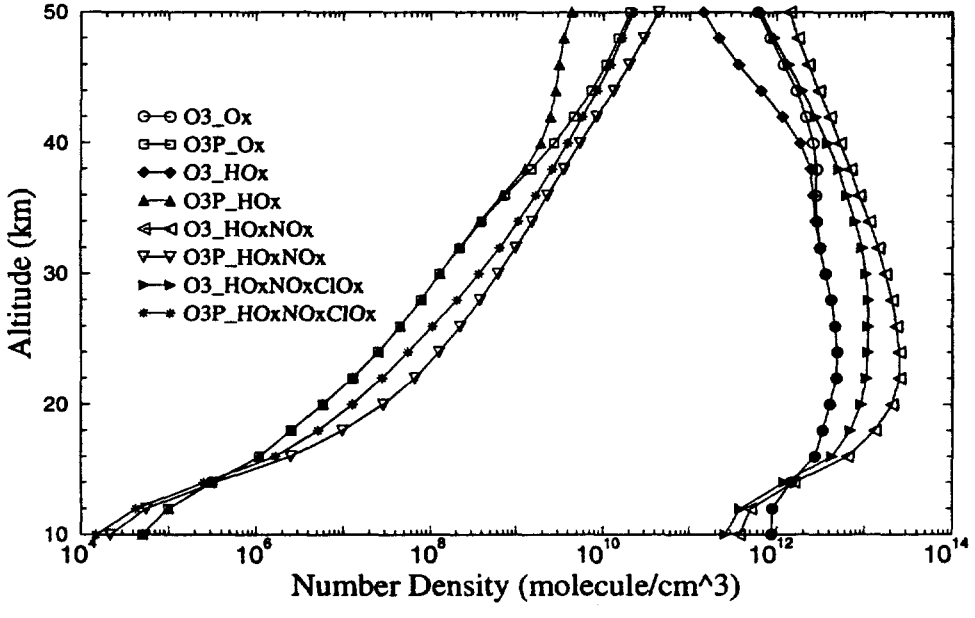

Figure 4.13 All the ozone and oxygen atom profiles from the basic oxygen cycle to the complete chlorine cycle. It shows the same pattern of variation for the oxygen atom and for ozone.

Ozone's variation with time varies with height. Figure 4.14 and 4.15 show the different features of ozone variation. Below 30 kilometer, ozone concentration is almost constant through each hour of the day. Above 30 kilometer, ozone shows daily variation, i.e. it reaches minimum at noon and daytime value is lower than nighttime value. That is because actinic flux is stronger at higher altitudes and chemical equilibrium condition is always maintained at those altitudes. But at altitudes lower than 30 km, the lifetime of ozone is longer and atmospheric transport is important. Moreover, the actinic flux in the lower stratosphere is not as strong as that at higher altitudes and thus ozone dissociation is slower.

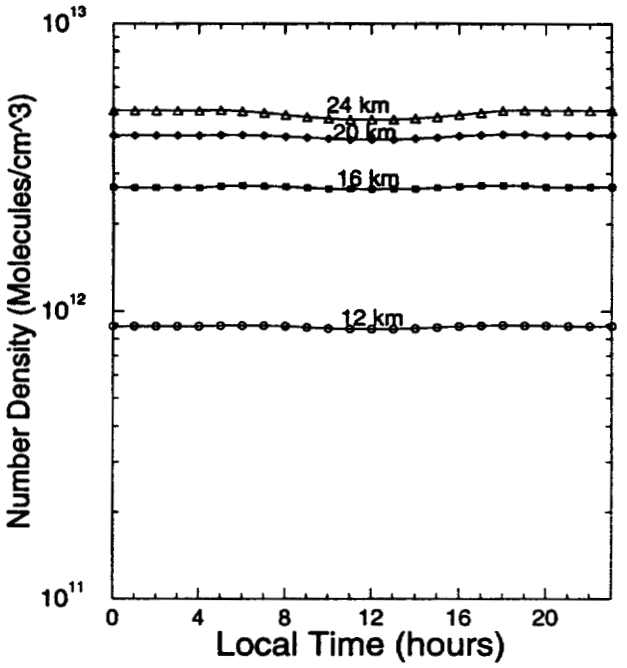

Figure 4.14 Daily ozone variation at different altitude (i.e. 12 km, 16 km, 20 km, 24 km). It shows almost no difference between day and night.

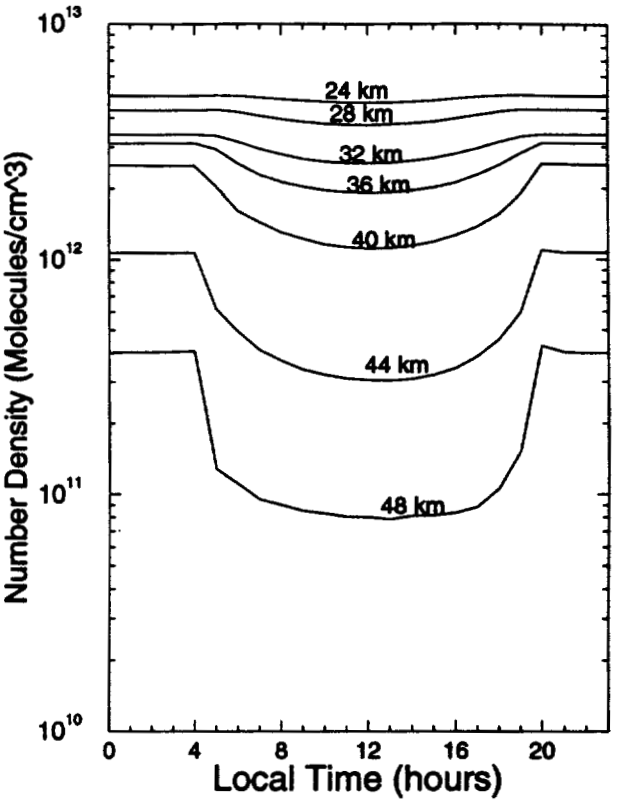

Figure 4.15 Daily ozone variation at different altitudes (i.e. 24 km, 28 km, 32 km, 36 km, 40 km, 44 km, 48 km). Significant daily variations occur at altitudes above 30 km.

Figure 4.16 shows ozone variation at midnight, noon time and daily average.

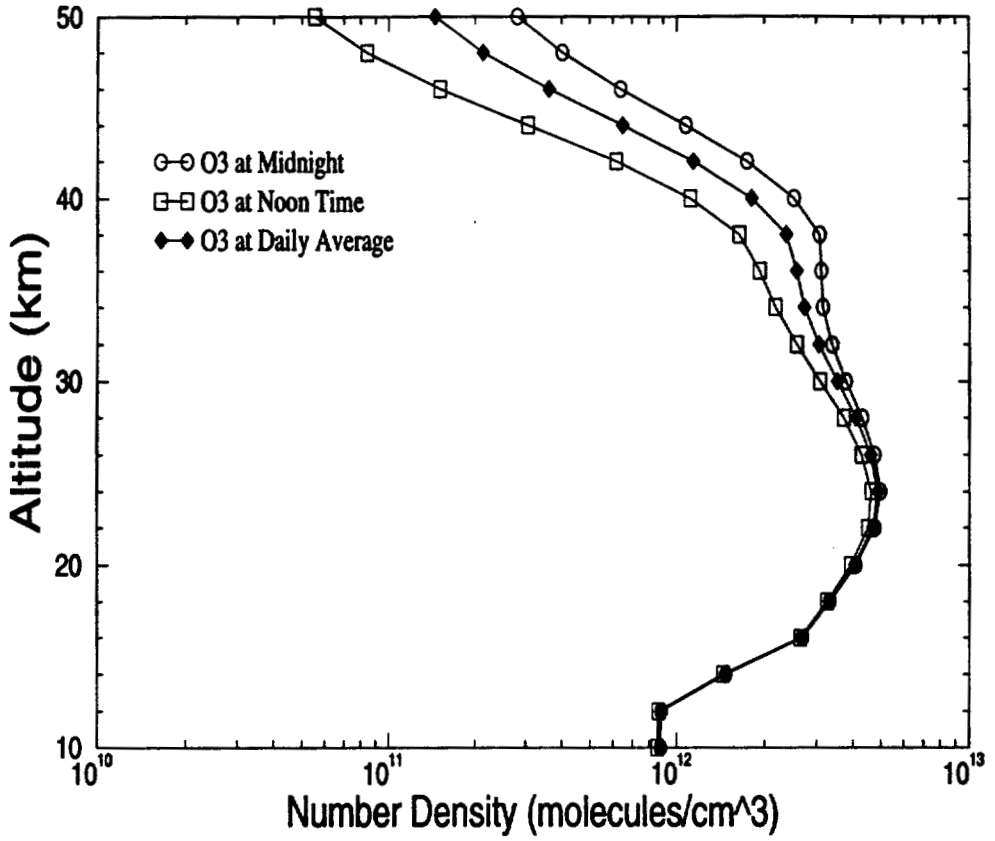

Figure 4.16 Ozone variation at midnight, noon time and daily average.

CHAPTER 5 COMPARISONS WITH OTHER MODELS

5.1 Ozone Concentrations

GCRC 1-D ozone model calculates ozone concentrations every hour from 10 km to 50 km. It compares well with other model results. Figure 5.1 shows the daily average O_3 and $O(^3P)$ concentrations from current model and LLNL model.

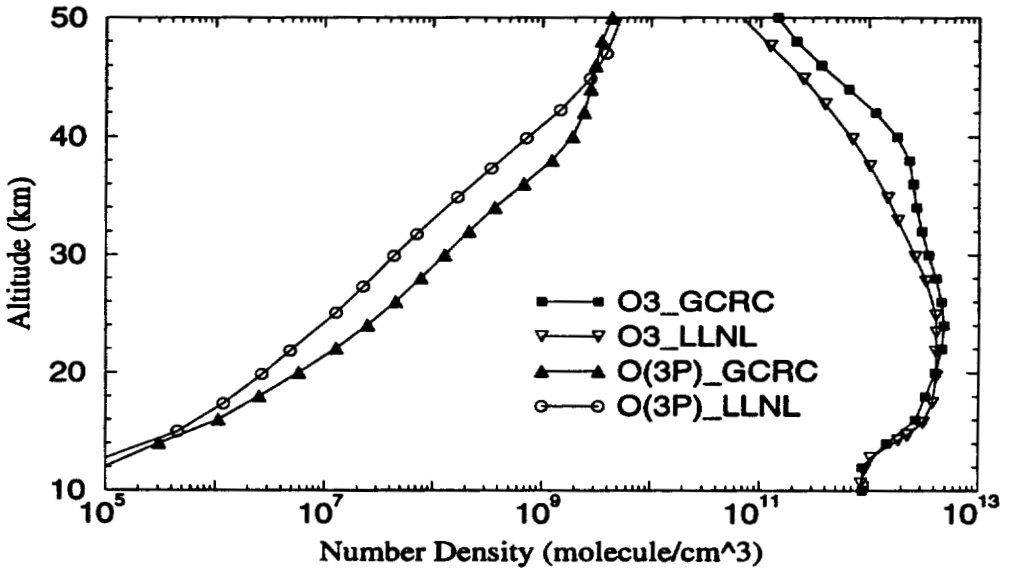

Figure 5.1 Oxygen species concentrations from GCRC and LLNL models.

5.2 Photodissociation Rates

Figure 5.2, 5.3, 5.4 compare the photolysis rates for O_2, O_3, NO_2, N_2O, and CFCs between GCRC and LLNL models. These species are very important for the distribution of ozone in the stratosphere. The graphs show that they are consistent. This means that actinic flux field from GCRC's own routine is justified.

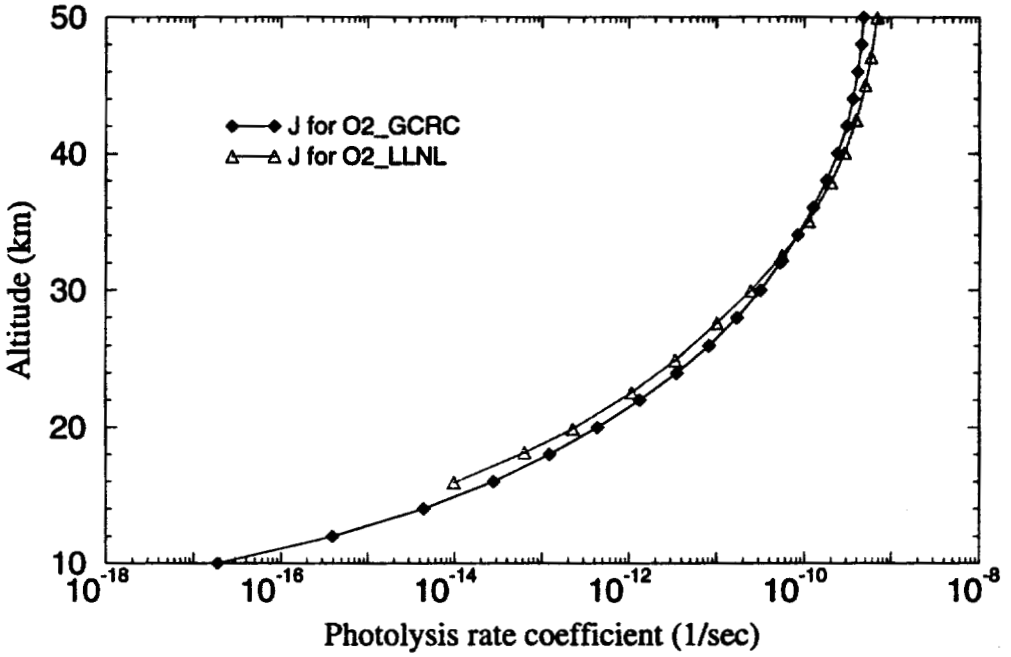

Figure 5.2 Photolysis rates for O_2 in the stratosphere from GCRC and LLNL model.

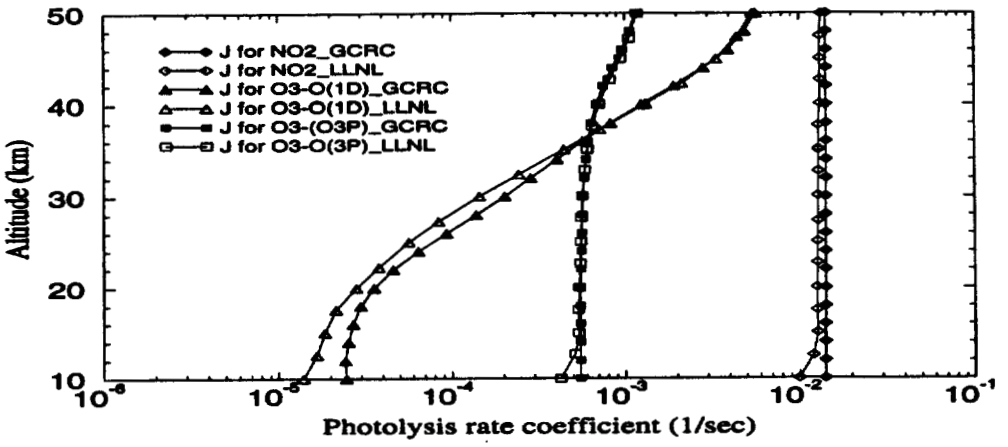

Figure 5.3 Photolysis rates for NO_2, O_3-$O(^1D)$, O_3-$O(^3P)$ in the stratosphere from GCRC and LLNL models.

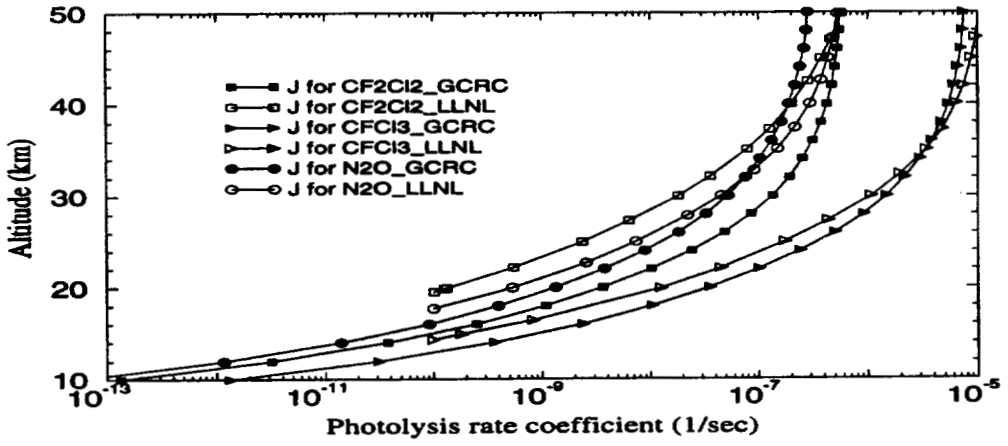

Figure 5.4 Photolysis rates for CF_2Cl_2, $CFCl_3$, and N_2O in the stratosphere from GCRC and LLNL models.

5.3 Hydrogen, Nitrogen and Chlorine Species

Figure 5.5 compares hydrogen cycle (HOx) between the two models. It indicates that the HOx concentrations above 25 km from GCRC ozone model are higher than those of LLNL model but lower while below about 25 km.

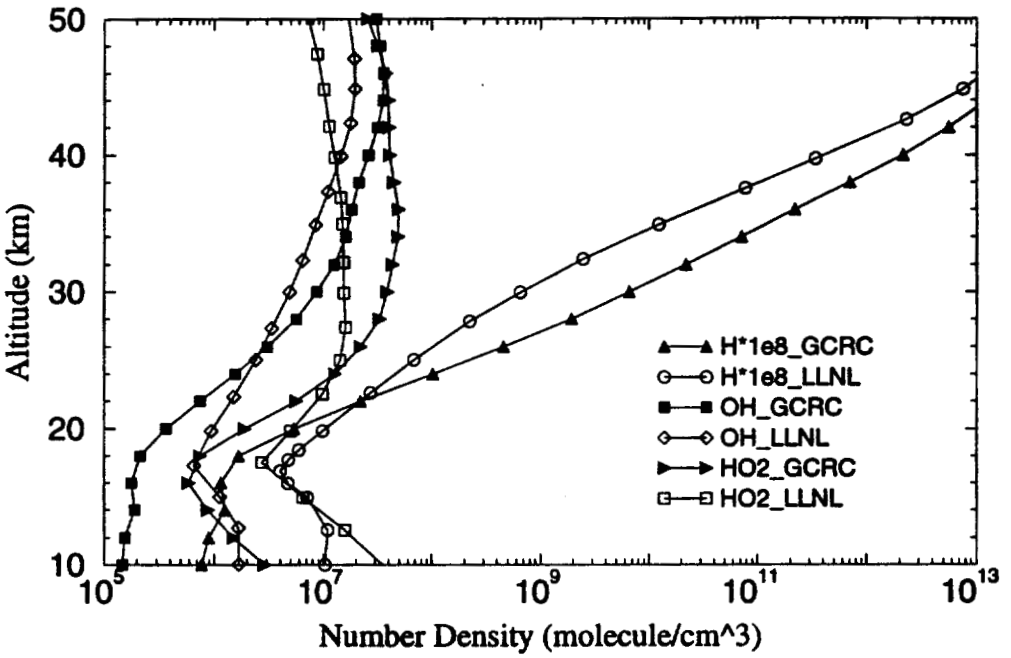

Figure 5.5 Hydrogen cycle species (HOx) from GCRC and LLNL models.

Figure 5.6 gives nitrogen species (NOx) comparison which show similar patterns for NO and NO_2.

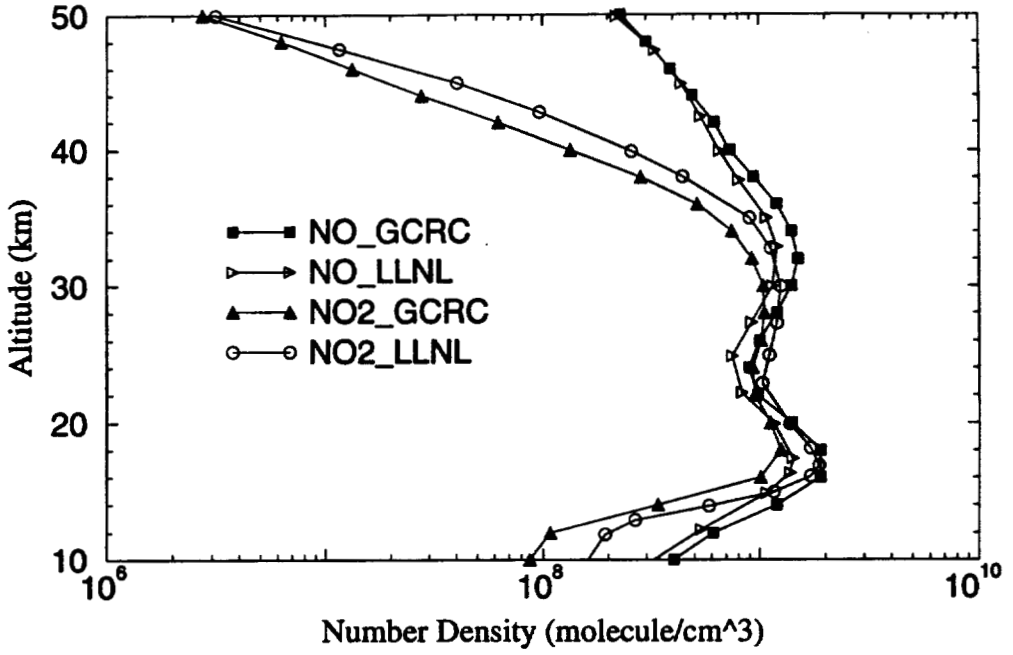

Figure 5.6 Nitrogen species (NO and NO_2) from GCRC and LLNL model. It shows the same pattern of change with altitude for both species.

Figure 5.7 gives Cl and ClO concentrations from the two models. There are big differences. But it still shows the same tendency of increase. Despite this inconsistency, GCRC ozone model produces correct ozone profiles and consistent photolysis rate coefficients and other species concentrations. It will be used as a base model for future CFC research.

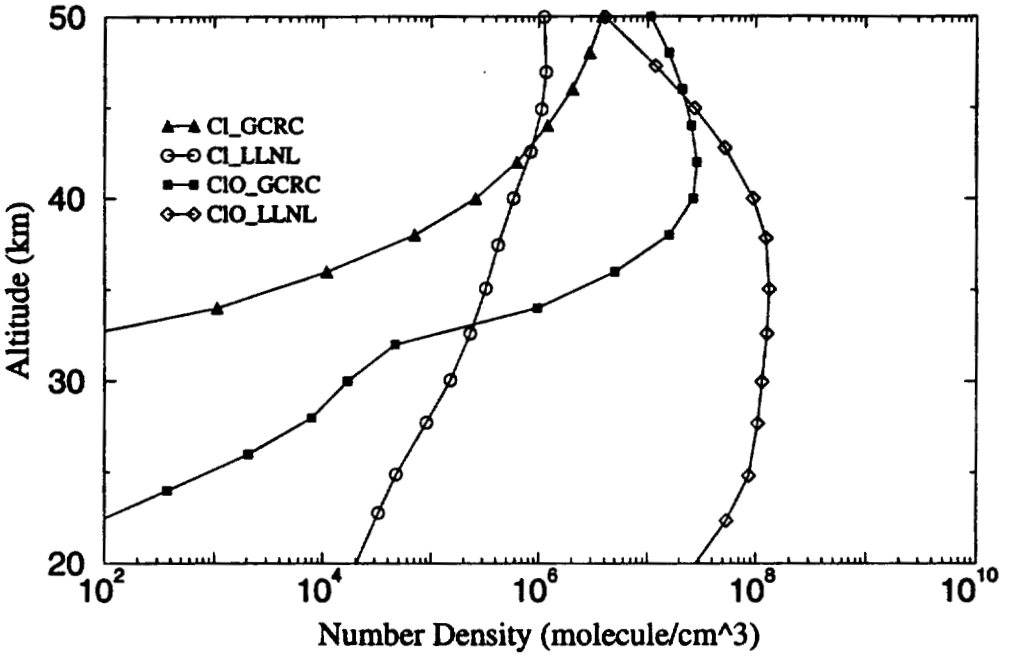

Figure 5.7 Comparison of Cl and ClO concentrations between GCRC and LLNL models.

CHAPTER 6 CONCLUSIONS

In this thesis a one dimensional stratospheric ozone model has been presented in detail. The model includes transport, actinic flux solver and stratospheric chemistry. Chlorine species studies are more complete than any other work. All the photodissociation rate values are in good agreements with those listed in NASA/JPL (1992) appendix. The model reproduces the main ozone features and other details and useful results. The ozone maximum is at about 25 kilometers. Night time ozone is more abundant than daytime ozone especially at altitudes above 30 kilometer. Chlorine species deplete ozone at altitude above 25 kilometer, specially above 35 km.

The GCRC stratospheric ozone model is a base program for further research on stratospheric ozone issues. The program is constructed that further improvements can be made easily. The next phase of this work is to improve the flexibility of the model in order to study various important aspects of ozone such as seasonal variations, latitudinal variations, and long term trends. In order to conduct such study, a two dimensional model is being developed.

REFERENCES

Augustsson, T.R., and J.S. Levine, The effects of isotropic multiple scattering and surface albedo on the photochemistry of the troposphere, *Atmos. Environ., 16,* 1373-1380, 1982.

Cicerone, R. J., and S. Walters, Nonlinear response of stratospheric ozone column to chlorine injections, *J. Geophys. Res., 88,* 3647-3661, 1983.

DeMore, W. B., S. P. Sander, D. M. Golden, R. F. Hampson, M. J. Kurylo, C. J. Howard, A. R. Ravishankara, C. E. Kolb, M. J. Molina, Chemical kinetics and photochemical data for use in stratospheric modeling, *JPL Publication 92-20,* NASA, Jet Propulsion Laboratory, 1992.

Demerjian, K.L., K.L. Schere, and J.T. Peterson, Theoretical estimates of actinic (spherically averaged) flux and photolytic rate constants of atmospheric species in the lower troposphere, in *Advances in Environmental Science and Technology,* vol. 10, edited by J.N. Pitts, Jr., and R.L. Metcalf, pp. 369-459, John Wiley, New York, 1980.

Finlayson-Pitts, B. J. and J. N. Pitts, Jr., *Atmospheric Chemistry: Fundamentals and Experimental Techniques*, John Wiley & Sons, Inc., New York, 1986.

Herman, J. R., and J. E. Mentall, O_2 absorption cross sections (187-225 nm) from stratospheric solar flux measurements, *J. Geophys. Res., 87,* 8967-8975, 1982.

Herman, J. R., and C. J. McQuillan, Atmospheric chlorine and stratospheric ozone nonlinearities and trend detection, *J. Geophys. Res., 90,* 5721-5732, 1985.

Khalil, M. A. K., and R. A. Rasmussen, Sources, sinks, and seasonal cycles of atmospheric methane, *J. Geophys. Res. 88,* 5131-5144, 1983a.

Khalil, M. A. K., and R. A. Rasmussen, Increase and seasonal cycles of nitrous oxide in the earth's atmosphere, *Tellus, 35B,* 161-169, 1983b.

Khalil, M. A. K., and R. A. Rasmussen, Modeling chemical transport and mass balances in the atmosphere, Environmental Exposure from Chemicals, Editors: W. B. Neely and G. E. Blau, CRC Press, Inc., Boca Raton, Florida, 22-54, 1984b.

Khalil, M. A. K., and R. A. Rasmussen, The global sources of nitrous oxide, *J. Geophys. Res. 97*, 14,651-14,660, 1992.

Khalil, M. A. K., A statistical method for estimating uncertainties in the total global budgets of atmospheric trace gases, *J. Environ. Sci. Health, A27(3)*, 755-770, 1992.

Khalil, M. A. K., Atmospheric trace gases, anthropogenic influences and global change, *Encyclopedia of Earth System Science, Volume 1*, 285-293, 1992.

Khalil, M. A. K., and R. A. Rasmussen, The environmental history and probable future of Fluorocarbon 11, *J. Geophys. Res., 98*, 23,091-23,106, 1993.

Khalil, M. A. K., *Physics and Chemistry of the Atmosphere: Theory and Modeling of Global Change*, Global Change Research Center, Oregon Graduate Institute of Science & Technology, Portland, Oregon, 1994.

Krueger, A. J., and R. A. Minzner, A mid-latitude ozone model for the 1976 U.S. Standard Atmosphere, *J. Geophys. Res., 81*, 4477-4481, 1976.

Levine, J. S., *The Photochemistry of Atmospheres: Earth, the Other Planets, and Comets*, Academic Press, Inc., New York, 1985.

Logan, J. A., M. J. Prather, S. C. Wofsy, and M. B. McElroy, Tropospheric Chemistry: a global perspective, *J. Geophys. Res. 86*, 7210-7254, 1981.

Luther, F. M., and R. J. Gelinas, Effect of molecular multiple scattering and surface albedo on atmospheric photodissociation rates, *J. Geophy. Res., 81*, 1125-1132, 1976.

Luther, F. M., Annual report of Lawrence Livermore National Laboratory to the FAA on the High Altitude Pollution Program-1980, *Publ. UCRL-50042-80*, Lawrence Livermore Lab., Univ. of Calif., Livermore, California, 1980.

Meador, W. E., and W. R. Weaver, Two-stream approximations to radiative transfer in planetary atmosphere: A unified description of existing methods and a new improvement, *J. Atmos. Sci., 37*, 630-643, 1980.

Miller, C., P. Meakin, R. G. E. Franks, and J. P. Jesson, The fluorocarbon-ozone theory-V. One-dimensional modeling of the atmosphere: the base case, *Atmospheric Environment 12*, 2481-2500, 1978.

Miller, C., D. L. Filkin, and J. P. Jesson, The fluorocarbon-ozone theory-VI. Atmospheric modeling: calculation of the diurnal steady-state, *Atmospheric Environment 13,* 381-394, 1978.

Owens, A. J., C. H. Hales, D. L. Filkin, C. Miller, J. M. Steed, and J. P. Jesson, A coupled one-dimensional radiative-convective, chemistry-transport model of the atmosphere: 1. Model structure and steady state perturbation calculations, *J. Geophys. Res., 90,* 2283-2311, 1985.

Pitari, G. and G. Visconti, A simple method to account for Rayleigh effects on photodissociation rates, *J. Atmos. Sci., 36,* 1803-1811, 1979.

Prather, M. J., Ozone in the upper stratosphere and mesosphere, *J. Geophys. Res., 86,* 5325-5338, 1981.

Prather, M. J., M. B. McElroy, and S. C. Wofsy, Reductions in ozone at high concentrations of stratospheric halogens, *Nature, 312,* 227-231, 1984.

Rundel, R. D., D. M. Butler, and R. S. Stolarski, Uncertainty propagation in a stratospheric model: 1, Development of a concise stratospheric model, *J. Geophys. Res., 83,* 3063-3073, 1978.

Stolarski, R. S., D. M. Butler, and R. D. Rundel, Uncertainty propagation in a stratospheric model: 2, Monte Carlo analysis of imprecisions due to reaction rates, *J. Geophys. Res., 83,* 3074-3078, 1978.

Stolarski, R. S., and A. R. Douglass, Parameterization of the photochemistry of stratospheric ozone including catalytic loss processes, *J. Geophys. Res., 90,* 10,709-10,718, 1985.

Thompson, A. M., The effects of clouds on photolysis rates and ozone formation rates in the unpolluted troposphere, *J. Geophys. Res., 89,* 1341-1349, 1984.

Warneck, P., *Chemistry of the Natural Atmosphere,* Volume 41 of *International Geophysics Series,* Academic Press, Inc., New York, 1988.

Whitten, R. C., and S. S. Prasad, *Ozone in the Free Atmosphere,* Van Nostrand Reinhold Company Inc., New York, 1985.

Wuebbles, D. J., Chlorocarbon emission scenarios: potential impact on stratospheric ozone, *J. Geophys. Res., 88,* 1433-1443, 1983.

Wuebbles, D. J., F. M. Luther, and J. E. Penner, Effect of coupled anthropogenic perturbations on stratospheric ozone, *J. Geophys. Res., 88,* 1444-1456, 1983.

World Meteorological Organization (WMO), *Atmospheric Ozone 1985, WMO Global Ozone Research and Monitoring Project - Report No. 16,* NASA, Washington, D.C., 1985.

World Meteorological Organization (WMO), *Report of the International Ozone Trends Panel 1988, WMO Global Ozone Research and Monitoring Project - Report No. 18,* NASA, Washington, D.C., 1985.

APPENDIX A UNITS OF CONCENTRATION

PPMV, PPBV, PPTV: parts per million by volume (ppmv), parts per billion by volume (ppbv), parts per trillion by volume (pptv). These units express the number of molecules of gas species found in a million (10^6), a billion (10^9), or a trillion (10^{12}) molecules of air, respectively. Alternatively, according to the ideal gas law (PV = nRT) numbers of molecules are proportional to their volumes, these units may be thought of as the number of volumes of the gas found in 10^6, 10^9, or 10^{12} volumes of air, respectively, at standard pressure and temperature.

Molecule/cm^3: This concentration unit is the number of molecules, atoms, or free radicals present in a given volume of air, usually a cubic centimeter (cm^3). This unit is used in the current model for all the concentrations.

ug/m^3: A third unit of measurement for gaseous species is mass per unit volume, usually 10^{-6} gram per cubic meter ($ug\ m^{-3}$).

DU: The Dobson Unit describes the total column density of ozone in the atmosphere. It expresses the total ozone as an equivalent column height at standard pressure and temperature, in units of 10^{-2} mm. 1DU = 10^{-2} mm.

Using the ideal gas law PV = nRT, we can calculate conversions factors between the above units. Some useful expressions are listed below.

$$PV = nRT = (m/M)RT,\ V = (m/M)RT/P$$

where m is actual mass (g), M is molecular weight, P is pressure, V volume, T absolute temperature and R the gas constant.

$$1\ ug/m^3 = 10^{-6}\ RT/PM\ (m^3/m^3) = RT/PM\ (ppmv) = RT/PM \times 10^3\ (ppbv) = RT/PM \times 10^6\ (pptv)$$

where P, V, T, and R are using metric system, the unit of M is g.

$$1\ ug/m^3 = 10^{-12} \times N_0/M\ (molecule/cm^3) = 6.02E11/M\ (molecule/cm^3)$$

where N_0 is Avogadro's number, $N_0 = 6.02 \times 10^{23}$.

$$1\ ppmv = 10^{-12}\ (m^3/cm^3) = 10^{-12} \times PN_0/RT\ (molecule/cm^3)$$

$$1\ DU = 10^{-3}\ cm = 10^{-9}\ m^3/cm^2 = 10^{-9}\ P_0N_0/RT_0\ (molecule/cm^2) = 2.46 \times 10^{16}\ (molecule/cm^2)$$

where $P_0 = 1.01 \times 10^5$ Pa, R = 8.31 m^{3}Pa/mol K, T_0 = 298K.

APPENDIX B CHEMICAL FORMULAE AND NOMENCLATURE

Symbol	Name
O	Atomic oxygen
O_2	Molecular oxygen
O_3	Ozone
O_x	Odd oxygen (O, $O(^1D)$, O_3)
N_2	Molecular nitrogen
N_2O	Nitrous oxide
NO	Nitric oxide
NO_2	Nitrogen dioxide
NO_3	Nitrogen trioxide, nitrate radical
NO_y	Odd nitrogen (NO, NO_2, NO_3, N_2O_5, $ClONO_2$, HNO_4, HNO_3)
NO_x	Oxides of nitrogen (NO, NO_2, NO_3)
N_2O_5	Dinitrogen pentoxide
HNO_2, HONO	Nitrous acid
HNO_3, $HONO_2$	Nitric acid
HNO_4, HO_2NO_2	Peroxynitric acid
H_2O	Water vapor
H_2O_2	Hydrogen peroxide
H	Hydrogen atom
HO, OH	Hydroxyl radical
HO_2	Hydroperoxyl radical
HO_x	Odd hydrogen (H, OH, HO_2, H_2O_2)
CO	Carbon monoxide
HCl	Hydrogen chloride
HOCl	Hypochlorous acid
Cl	Chlorine atom
ClO	Chlorine monoxide
$ClONO_2$,$ClNO_3$	Chlorine nitrate
Cl_x	Odd chlorine, inorganic chlorine
CH_4	Methane
H_2CO, CH_2O	Formaldehyde
CH_3OOH	Methyl hydroperoxide
CH_3Cl	Methyl chloride
CFC	Chlorofluorocarbon
HC	Hydrocarbon
CH_3CCl_3	Methyl Chloroform
CCl_4	Carbon tetrachloride
CCl_3F	Trichlorofluoromethane (F-11)
CCl_2F_2	Dichlorodifluoromethane

Br	Bromine atom
BrO	Bromine monoxide
Br_x	Odd bromine, inorganic bromine

APPENDIX C SOURCE CODE FOR GCRC STRATOSPHERIC OZONE MODEL

1. Main program

```
/* File: O3ozone.c
 *
 * Copyright 1994 Zhongyi Ye and M.A.K. Khalil
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appear in all copies and that both that
 * copyright notice and this permission notice appear in supporting
 * documentation. No representation is made about the suitability of this
 * software for any purpose.  It is provided "as is" without express or
 * implied warranty.
 *
 */

#include <stdio.h>
#include <math.h>
#include "O3model.h"
#include "O3util.h"

main()
{
  int i,j,k;
  double totalcon,column;

  crit = 0.01;
  /* Latitude is 45 degrees North
   * Inclination is Northern Hemisphere Winter
   */

  latitude = 45.0*PI/180.0;              /* change degrees to radians */
  inclination = 23.27;
  inclination = inclination*PI/180.0;

  /* set output and solve parameters */
  for (i=0;i<=6;i++)
    {
      solve[i]=1;
      converge[i]=1;
      output[i]=12;
    }
  for (i=7;i<31;i++)
    {
```

```
      solve[i] = +1;
      converge[i] = +1;
      output[i] =-1;
    }
   outputtype = 4;

  /* Caculate the temperature, pressure, and number density of the
   * atmosphere and input the concentrations of the gases that the
   * model does not solve for (NOX, N2O, etc).
   */
  get_std_atmos(T,M,P,Z);

  /* Calculate the rate constants for the chemical (K) and photochemical
   * (J) reactions.
   */
  get_K(T,M,P);                          /* Calculate the K values */
  get_acs();                        /* absorption cross sections */
  get_actinic(T,M,Z,O3_column);       /* Calculate the actinic flux */
  get_J(T,M,P);                          /* Calculate the J values */

/* Main program loop */
  get_initial_cons(solve);        /* Calculate initial concentrations */
  get_cons(solve, crit);          /* Iteratively solve concentrations */
  output_data(outputtype,output,Z,M); /* Output data */

return 0;
}
```

2. /* === File: O3acs.c

```
 * includes: void get_acs()
 *     double getacsO2(double,double),getacsO3(double,double).....
 */

/* ==== getacs.c ====  */
#include <stdio.h>
#include <math.h>
#include "O3util.h"
#include "O3model.h"
void get_acs()
{
  int i, j;

  /* Input the absorption cross section data. The first number
   * is the beginning wavelength (in nm) and the second is the absorption
   * cross section for the wavelengths between the beginning wavelength
   * and the next beginning wavelength. This is the procedure for all of
   * the gases except for NO2.
   *
```

```
    * Input the NO2 absorption cross section data. The first number
    * is the beginning wavelength (in nm), the second is the absorption
    * cross section for the wavelengths between the beginning wavelength
    * and the next beginning wavelength, and the third is the change in
    * ACS per unit (K) change of temperature.
    */

    acsO2_NUM = get_2_data(acsO2_FILE,acsO2);
    acsO3_NUM = get_2_data(acsO3_FILE,acsO3);
    acsNO2_NUM = get_3_data(acsNO2_FILE,acsNO2);
    /* acsN2O_NUM = temperature dependent */
    acsHNO3_NUM = get_2_data(acsHNO3_FILE,acsHNO3);
    /* acsH2O2_NUM = temperature dependent */
    acsNO3_NUM = get_2_data(acsNO3_FILE,acsNO3);
    acsH2CO_NUM = get_2_data(acsH2CO_FILE,acsH2CO);
    acsCF2Cl2_NUM = get_2_data(acsCF2Cl2_FILE,acsCF2Cl2);
    acsCFCl3_NUM = get_2_data(acsCFCl3_FILE,acsCFCl3);
    acsClONO2_NUM = get_2_data(acsClONO2_FILE,acsClONO2);
    acsHCl_NUM = get_2_data(acsHCl_FILE,acsHCl);
    acsHOCl_NUM = get_2_data(acsHOCl_FILE,acsHOCl);
    acsN2O5_NUM = get_2_data(acsN2O5_FILE,acsN2O5);
    acsCH3OOH_NUM = get_2_data(acsCH3OOH_FILE,acsCH3OOH);
    acsCCl4_NUM = get_2_data(acsCCl4_FILE,acsCCl4);
    acsHNO4_NUM = get_2_data(acsHNO4_FILE,acsHNO4);
    acsH2O_NUM = get_2_data(acsH2O_FILE,acsH2O);
    acsClO_NUM = get_2_data(acsClO_FILE,acsClO);
    acsCl2_NUM = get_2_data(acsCl2_FILE,acsCl2);
    acsClOO_NUM = get_2_data(acsClOO_FILE,acsClOO);
    acsOClO_NUM = get_2_data(acsOClO_FILE,acsOClO);
    acsCl2O2_NUM = get_2_data(acsCl2O2_FILE,acsCl2O2);
    acsClNO_NUM = get_2_data(acsClNO_FILE,acsClNO);
    acsClNO2_NUM = get_2_data(acsClNO2_FILE,acsClNO2);
    acsClONO_NUM = get_2_data(acsClONO_FILE,acsClONO);
}


/* === acs.c ===
 *
 * Contains: functions which determine the absorption cross section (ACS)
 * for the photodissociation in the O3 model.
 *
 * Implementation: Two procedures are used for determining the ACS. The
 * first involves searching through a list of data for the proper value
 * and the second involves pluging the wavelength and temperature into
 * a formula. The second procedure is self explainatory but the first
 * requires more detail. The main() section of the program reads in a
 * starting wavelength, the ACS for the wavelengths starting at given
 * wavelength and ending at the next starting wavelength, and sometimes
```

```
 * a temperature parameter that modifies the ACS given. The functions
 * that use this aproach then search the wavelengths until they find one
 * that is larger than the wavelength of interest and use the ACS of
 * the previous wavelength (the wavelengths run from lowest to highest).
 */
#include <stdio.h>
#include <math.h>
#include "O3util.h"
#include "O3model.h"

/* ---- O2 ---- */
double getacsO2(double wavelength)
{
  extern double acsO2[][2];
  extern int acsO2_NUM;
  int i;

  if (wavelength < acsO2[0][0])
  return acsO2[0][1];
  if (wavelength > acsO2[acsO2_NUM-1][0])
  return 0;

  for (i=1;wavelength>=acsO2[i][0] && i<acsO2_NUM;i++)
    ;
  return acsO2[i-1][1];
}

/* ---- O3 ---- */
double getacsO3(double wavelength)
{
  int i;

  if (wavelength < acsO3[0][0])
  return acsO3[0][1];
  if (wavelength > acsO3[acsO3_NUM-1][0])
  return 0;
  for (i=1;wavelength>=acsO3[i][0] && i<acsO3_NUM;i++)
    ;
  return acsO3[i-1][1];
}

/* ---- NO2 ---- */
double getacsNO2(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsNO2[i][0] && i<acsNO2_NUM;i++)
    ;
```

```
  return (acsNO2[i-1][1] + acsNO2[i-1][2]*(temperature-273.15));
}

/* ---- N2O ---- */
#define A0 68.21023
#define A1 -4.071805
#define A2 4.301146E-2
#define A3 -1.777846E-4
#define A4 2.520672E-7
#define B0 123.4014
#define B1 -2.116255
#define B2 1.111572E-2
#define B3 -1.881058E-5

double getacsN2O(double wavelength, double temperature)
{
  double A, B;

  if (wavelength < 173 || wavelength > 240 ||
      temperature < 194 || temperature > 320)
  {
    return 0;
  }
  A = A0 + A1*wavelength + A2*pow(wavelength,2.0) +
        A3*pow(wavelength,3.0) + A4*pow(wavelength,4.0);
  B = B0 + B1*wavelength + B2*pow(wavelength,2.0) +
      B3*pow(wavelength,3.0);

  return (exp(A + (temperature-300.0) * exp(B)));
}

/* ---- HNO3 ---- */
double getacsHNO3(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsHNO3[i][0] && i<acsHNO3_NUM;i++)
    ;
  return (acsHNO3[i-1][1]);
}

/* ---- H2O2 ---- */
#undef  A0
#define A0 6.4761E4
#undef  A1
#define A1 -9.2170972E2
#undef  A2
#define A2 4.535649
```

```
#undef  A3
#define A3 -4.4589016E-3
#undef  A4
#define A4 -4.035101E-5
#define A5 1.6878206E-7
#define A6 -2.652014E-10
#define A7 1.5534675E-13
#undef  B0
#define B0 6.8123E3
#undef  B1
#define B1 -5.1351E1
#undef  B2
#define B2 1.1522E-1
#undef  B3
#define B3 -3.0493E-5
#define B4 -1.0924E-7

double getacsH2O2(double wavelength, double temperature)
{
  double chi, A, B;

  if (wavelength < 260 || wavelength > 350 ||
      temperature < 200 || temperature > 400)
  {
    return 0;
  }
  A = A0 + A1*wavelength + A2*pow(wavelength,2.0) + A3*pow(wavelength,3.0) +
      A4*pow(wavelength,4.0) + A5*pow(wavelength,5.0) +
      A6*pow(wavelength,6.0) + A7*pow(wavelength,7.0);
  B = B0 + B1*wavelength + B2*pow(wavelength,2.0) +
      B3*pow(wavelength,3.0) + B4*pow(wavelength,4.0);
  chi = 1.0 / ( 1.0 + exp(-1265.0/temperature));
  return (1.0E-21 * (chi*A + (1.0-chi)*B));
}

/* ---- NO3 ---- */
double getacsNO3(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsNO3[i][0] && i<acsNO3_NUM;i++)
    ;
  return (acsNO3[i-1][1]);
}

/* ---- H2CO ---- */
double getacsH2CO(double wavelength, double temperature)
{
```

```
  int i;

  for (i=1;wavelength>=acsH2CO[i][0] && i<acsH2CO_NUM;i++)
    ;
  return (acsH2CO[i-1][1]);
}

/* ---- CF2Cl2 ---- */
double getacsCF2Cl2(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsCF2Cl2[i][0] && i<acsCF2Cl2_NUM;i++)
    ;
  return (acsCF2Cl2[i-1][1]);
}

/* ---- CFCl3 ---- */
double getacsCFCl3(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsCFCl3[i][0] && i<acsCFCl3_NUM;i++)
    ;
  return (acsCFCl3[i-1][1]);
}

/* ---- ClONO2 ---- */
double getacsClONO2(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsClONO2[i][0] && i<acsClONO2_NUM;i++)
    ;
  return (acsClONO2[i-1][1]);
}

/* ---- HCl ---- */
double getacsHCl(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsHCl[i][0] && i<acsHCl_NUM;i++)
    ;
  return (acsHCl[i-1][1]);
}

/* ---- HOCl ---- */
```

```
double getacsHOCl(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsHOCl[i][0] && i<acsHOCl_NUM;i++)
    ;
  return (acsHOCl[i-1][1]);
}

/* ---- N2O5 ---- */
double getacsN2O5(double wavelength, double temperature)
{
  int i;

  if (wavelength < 285.0)
  {
    for (i=1;wavelength>=acsN2O5[i][0] && i<acsN2O5_NUM;i++)
      ;
    return (acsN2O5[i-1][1]);
  }
  else
    return (1.0E-20*exp(2.735+((4728.5-17.127*wavelength)/temperature)));
}

/* ---- CH3OOH ---- */
double getacsCH3OOH(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsCH3OOH[i][0] && i<acsCH3OOH_NUM;i++)
    ;
  return (acsCH3OOH[i-1][1]);
}
/*
 * #undef A0
 * #define A0 8.7756960795E-15
 * #undef A1
 * #define A1 -0.0490747133
 *
 * double getacsCH3OOH(double wavelength, double temperature)
 * {
 *   if (wavelength < 210.0)
 *     return 0.0;
 *   else
 *     return (A0*exp(A1*wavelength));
 * }
 */
```

```
/* ---- CCl4 ---- */
double getacsCCl4(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsCCl4[i][0] && i<acsCCl4_NUM;i++)
    ;
  return (acsCCl4[i-1][1]);
}

/* ---- HNO4 ---- */
double getacsHNO4(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsHNO4[i][0] && i<acsHNO4_NUM;i++)
    ;
  return (acsHNO4[i-1][1]);
}

/* ---- ClO ---- */
double getacsClO(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsClO[i][0] && i<acsClO_NUM;i++)
    ;
  return (acsClO[i-1][1]);
}

/* ---- H2O ---- */
double getacsH2O(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsH2O[i][0] && i<acsH2O_NUM;i++)
    ;
  return (acsH2O[i-1][1]);
}

/* ---- Cl2 ---- */
double getacsCl2(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsCl2[i][0] && i<acsCl2_NUM;i++)
    ;
  return (acsCl2[i-1][1]);
```

```
}

/* ---- ClOO ---- */
double getacsClOO(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsClOO[i][0] && i<acsClOO_NUM;i++)
    ;
  return (acsClOO[i-1][1]);
}

/* ---- OClO ---- */
double getacsOClO(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsOClO[i][0] && i<acsOClO_NUM;i++)
    ;
  return (acsOClO[i-1][1]);
}

/* ---- Cl2O2 ---- */
double getacsCl2O2(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsCl2O2[i][0] && i<acsCl2O2_NUM;i++)
    ;
  return (acsCl2O2[i-1][1]);
}

/* ---- ClNO ---- */
double getacsClNO(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsClNO[i][0] && i<acsClNO_NUM;i++)
    ;
  return (acsClNO[i-1][1]);
}

/* ---- ClNO2 ---- */
double getacsClNO2(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsClNO2[i][0] && i<acsClNO2_NUM;i++)
```

```
    ;
  return (acsClNO2[i-1][1]);
}

/* ---- ClONO ---- */
double getacsClONO(double wavelength, double temperature)
{
  int i;

  for (i=1;wavelength>=acsClONO[i][0] && i<acsClONO_NUM;i++)
    ;
  return (acsClONO[i-1][1]);
}
/* the end */
```

3. /* File: O3actinic.c

```
 * This program calculate actinic flux from 0km t0 74km
 * 38 layers, actinic flux from 0-52km will be used in subsequent
 * files.
 *
 */

#include <stdio.h>
#include <math.h>
#include "O3model.h"
#include "O3util.h"
#define RAY_CONST (1.044E-11)

get_actinic(double *T, double *M, double Z[][2],
            double *O3_column)
{
  /* ** SPACE_POINTS =27   ** */
  double **Fop, Fconst[2*(11+SPACE_POINTS)],
         Fdir[(11+SPACE_POINTS)],
         Fup[(11+SPACE_POINTS)],
         Fdown[(11+SPACE_POINTS)],
        Ftrans_diff[(11+SPACE_POINTS)],
         Fscat_diff[(11+SPACE_POINTS)],
         Fscat_dir[(11+SPACE_POINTS)],
         a[(11+SPACE_POINTS)*2],
         b[(11+SPACE_POINTS)*2],
         c[(11+SPACE_POINTS)*2],
         r[(11+SPACE_POINTS)*2],
         u[(11+SPACE_POINTS)*2],
        solar_flux[MAX],
        refraction[MAX],
        deltaM[SPACE_POINTS+11],
        O3_optical_depth,
```

```
        O2_optical_depth,
        Ray_optical_depth,
        Base_Ray_optical_depth,
        angle;

int wavelength_bin,
    angle_bin,
    i, j, k;

/* Fill ActAngle with the 'standard' angles */
initialize_angles(ActAngle);
/* Input the solar data */
num_wavelength_bins = get_solar_data(ActBin,solar_flux,refraction);
 /* Allocate space for use with matrix routines */
Fop = dmatrix(0,2*(11+SPACE_POINTS)-1,1,3);
for (i=0;i<2*(11+SPACE_POINTS)-1;i++)
   for (j=1;j<4;j++)
      Fop[i][j] = 0;
/* Calculate the layer density of the atmosphere */
get_layer_density(M,Z,deltaM);

for (wavelength_bin=0;wavelength_bin<num_wavelength_bins;wavelength_bin++)
{
   /* Rayleigh Scattering Optical Depth of the entire atmosphere */
   Base_Ray_optical_depth = RAY_CONST*refraction[wavelength_bin]*
                     pow(1.0e-7*ActBin[wavelength_bin],-4.0);

 for (angle_bin=0;angle_bin<ANGLE_POINTS;angle_bin++)
   {
      /* zenith angle in radians */
       angle = PI*ActAngle[angle_bin]/180.0;

       for (k=SPACE_POINTS+10;k>=0;k--)
       {
         /* O3 and O2 Optical Depth */
         O3_optical_depth = get_O3_optical_depth(k,O3_column,
                                 ActBin[wavelength_bin]);
         O2_optical_depth = get_O2_optical_depth(k,O2_column,
                           ActBin[wavelength_bin]);

          /* Rayleigh Scattering Optical Depth. */
 Ray_optical_depth = Base_Ray_optical_depth * deltaM[k];

 /* Fractional diffuse radiation transmission */
 Ftrans_diff[k] = exp(-(O3_optical_depth + O2_optical_depth +
                                 Ray_optical_depth)*AMdiff);

/* Fractional diffuse radiation scattering */
```

```
    Fscat_diff[k] =  exp(-O2_optical_depth*AMdiff)*
            exp(-O3_optical_depth*AMdiff)*
              (1.0 - exp(-Ray_optical_depth*AMdiff));

            /* Fractional direct radiation scattering */
    Fscat_dir[k] =  exp(-O2_optical_depth*AMdir(angle))*
       exp(-O3_optical_depth*AMdir(angle))*
              (1.0 - exp(-Ray_optical_depth*AMdir(angle)));

            /* Calculate the direct radiation flux */
    if (k == SPACE_POINTS+10)
              Fdir[k] = solar_flux[wavelength_bin]*exp(-(O3_optical_depth +
                  O2_optical_depth + Ray_optical_depth)*AMdir(angle));
    else
              Fdir[k] = Fdir[k+1]*exp(-(O3_optical_depth +
                    O2_optical_depth + Ray_optical_depth)*AMdir(angle));
      }

         fill_mat(a,b,c,r,Fdir,Ftrans_diff, Fscat_diff, Fscat_dir,
                          sur_albedo, angle);
        /* solve the matrix equation using Thomas method */
        tridag(a-1,b-1,c-1,r-1,u-1,2*(11+SPACE_POINTS));

         for (k=0;k<SPACE_POINTS+11;k++)
         {
           ActFlux[wavelength_bin][angle_bin][k] =
             Fdir[k] + /*Fconst[2*k] + Fconst[2*k+1];*/
             u[2*k] + u[2*k+1];
         }
      }
    }

void initialize_angles(double *ActAngle)
{
   ActAngle[0] = 0.0;
   ActAngle[1] = 10.0;
   ActAngle[2] = 20.0;
   ActAngle[3] = 30.0;
   ActAngle[4] = 40.0;
   ActAngle[5] = 50.0;
   ActAngle[6] = 60.0;
   ActAngle[7] = 70.0;
   ActAngle[8] = 78.0;
   ActAngle[9] = 86.0;
}

int get_solar_data(double *ActBin, double *solar_flux, double *refraction)
{
```

```
  int k;
  FILE *fp;

  /* Input the solar flux at the top of the atmosphere and the
   * refraction of the atmosphere.
   *
   * The solar flux is given as a 1 nm wavelength band centered on
   * the given wavelength. Thus, when integrating, the solar flux
   * would be solar_flux[i]*(ActBin[i+1]-ActBin[i]).
   *
   * Note: the refraction array is input as (1 - refraction). This
   * value is then squared so that refraction[] holds the values
   * of (1 - refraction)^2.
   */
  if ( (fp=fopen(TOP_RAD_FILE,"r")) == NULL)
  {
    perror(TOP_RAD_FILE);
    exit(0);
  }

  for(k=0;fscanf(fp,"%lf %lf %lf",&(ActBin[k]), &(solar_flux[k]),
                                  &(refraction[k])) != EOF;k++)
    refraction[k] = pow(refraction[k],2.0);
  fclose(fp);
  /* Return the number of wavelength bins the data are divided into */
  return k;
}

double AMdir(double angle)
{
  return (35.0/sqrt( (1224.0 * pow(cos(angle),2.0) + 1.0)));
}

void fill_mat(double *a, double *b, double *c, double *r,
        double *Fdir,double *Ftrans_diff, double *Fscat_diff,
        double *Fscat_dir,double albedo,  double angle)
{
  int i, k;
  double avg(double *, int, int);
/* here we try to use tridiagonal methods */
  /* Fill the surface layer */
/* downward */
  a[1] = 1.0;
  b[1] = -0.5*avg(Fscat_diff,0,1);
  c[1] = -(avg(Ftrans_diff,0,1) + 0.5*avg(Fscat_diff,0,1));
  r[1] = avg(Fscat_dir,0,1)*Fdir[1]*cos(angle);
/* upward */
  a[0] = 0.0;
```

```
  b[0] = -albedo - 0.5*avg(Fscat_diff,0,1);
  c[0] = 1.0;
  r[0] = (Fscat_dir[0] + 2.0*albedo)*cos(angle)*Fdir[0];
  /* Fill the matrix for all levels between the top atmospheric
     level and the surface layer */
  for (k=1;k<SPACE_POINTS+10;k++)
     {
        /* downward */
     a[2*k+1] = 1.0;
     b[2*k+1] = -0.5*avg(Fscat_diff,k,k+1);
     c[2*k+1] = -(avg(Ftrans_diff,k,k+1) +
                  0.5*avg(Fscat_diff,k,k+1));
     r[2*k+1] = avg(Fscat_dir,k,k+1)*Fdir[k+1]*cos(angle);
        /* upword */
     a[2*k] = -(avg(Ftrans_diff,k,k-1) +
                   0.5*avg(Fscat_diff,k,k-1));
     b[2*k] = -0.5*avg(Fscat_diff,k,k-1);
     c[2*k] = 1.0;
     r[2*k] = avg(Fscat_dir,k,k-1)*Fdir[k]*cos(angle);
    }
  /* Fill the matrix for the top atmospheric level */
  k = SPACE_POINTS+10;

  a[2*k+1] = 1.0;
  b[2*k+1] = -0.5*Fscat_diff[k];
  c[2*k+1] = 0.0;
  r[2*k+1] = Fscat_dir[k]*Fdir[k]*cos(angle);

  a[2*k] = -avg(Ftrans_diff,k,k-1) - 0.5*avg(Fscat_diff,k,k-1);
  b[2*k] = -0.5*avg(Fscat_diff,k,k-1);
  c[2*k] = 1.0;
  r[2*k] = avg(Fscat_dir,k,k-1)*Fdir[k]*cos(angle);
  }

double avg(double arr[], int i, int j)
{
  return ((arr[i]+arr[j])/2.0);
}

void get_layer_density(double *M, double Z[][2], double *deltaM)
{
  int k;

 /* since layer starts at 10km, not ground level, input totalM
  * calculated by hand. Data from Warneck (p.718) 0-48km
  */

/* add 11 layers from 54km to 74km, i.e. at 54,56,58,60,62,64,66,
```

```
 * 68,70,72,74km.
 * for deltaM[27..37], I use the O2_column[27..37].
 * deltaM[i]=(O2_column[i]-O2_column[i+1])/0.21/totalM.
 *
 * deltaM for the top layer is different from other layers since it takes
 * account the air mass above
 * the total air column above is O2_column[SPACE_POINTS-1]/0.21 (1/cm^2)
 *
 * deltaM[37] = O2_column[37]/0.21/totalM.
 * SPACE_POINTS = 27
 */

  deltaM[37] = O2_column[37]/0.21/totalM;
  for(k=27;k<37;k++)
  deltaM[k]=(O2_column[k]-O2_column[k+1])/0.21/totalM;

  for (k=0;k<SPACE_POINTS;k++)
     deltaM[k] = M[k]*Z[k][1]/totalM;
}

double get_O3_optical_depth(int k, double O3_column[],
                    double actinic_flux_bin)
{
   double O3_optical_depth;
   extern double getacsO3(double);

   if (k == SPACE_POINTS+10)
     O3_optical_depth = O3_column[k] * getacsO3(actinic_flux_bin);
   else
     O3_optical_depth = (O3_column[k]-O3_column[k+1]) *
                    getacsO3(actinic_flux_bin);
   return O3_optical_depth;
}
/*  I changed the structure of deltaM[] in "get_layer_density",
   and changed the file O3column.dat.  I added the lower 5 layers
   column of O3 into the file and changed the variable.
   O3_column[SPACE_POINTS+2] becomes O3_column[SPACE_POINTS+7].
*/
double get_O2_optical_depth(int k, double O2_column[],
                    double actinic_flux_bin)
{
   double O2_optical_depth;
   extern double getacsO2(double);

   if (k == SPACE_POINTS+10)
     O2_optical_depth = O2_column[k] * getacsO2(actinic_flux_bin);
   else
     O2_optical_depth = (O2_column[k]-O2_column[k+1]) *
```

```
                    getacsO2(actinic_flux_bin);
 return O2_optical_depth;
}

4. /*=== File: O3cons.c
 *
 * Purpose: Given initial values for the 'reactive' species in the model,
 *          this function iteratively solves for the species using the
 *          complete chemistry (unlike the intial values which were
 *          based upon a subset of the chemistry affecting the species).
 *
 *       Transport is included
 *       with NOx, NO + NO2 = NOx
 */
#include <stdio.h>
#include <math.h>
#include "O3model.h"
#include "O3util.h"
#define DELTAT (86400.0/TIME_POINTS)

void get_cons(int *solve, double crit)
{
  int i, j, convergence = 0,iteration=0;
  double a, b, c, oldcon, rho, rhop, rhom;
  extern double tras(int, double [][], int, int, double, double, double *);
  extern double change(double,double);

  while (convergence == 0 && iteration <10000)
  {
   fprintf(stderr,"%d\r", iteration++);
    convergence = 1;
    for (j=0;j<TIME_POINTS;j++)
    {
      for (i=0;i<SPACE_POINTS;i++)
      {
       /* zero flux boundary for every species */
        rho = DELTAT/ (Z[i][1]*Z[i][1]);
        if (i==0)
        {
          rhop =DELTAT/(Z[i][1]*Z[i][1])*getKz(i+1,Kz)*avgM(i+1,M);
          rhom =0;
       }
        else if (i== SPACE_POINTS-1)
        {
          rhop = 0;
          rhom = DELTAT /(Z[i][1]*Z[i][1])*getKz(i,Kz)*avgM(i,M);
        }
        else
```

```
        {
          rhop = DELTAT/(Z[i][1]*Z[i][1])*getKz(i+1,Kz)*avgM(i+1,M);
          rhom = DELTAT/(Z[i][1]*Z[i][1])*getKz(i,Kz)*avgM(i,M);
        }

/* -1. --- O --- */
          if (solve[IO] == 1)
          {
            oldcon = O[i][j+1];
            O[i][j+1] =(trans(0,O,i,j,rhop,rhom,M)+
                        O[i][j]+DELTAT*(2.0 * J1[i][j+1]*O2[i]
                         +J2[i][j+1]*O3[i][j]+
                         J4[i][j+1]*NO2[i][j]
                         +J20[i][j+1]*ClO[i][j]
                        +J22[i][j+1]*NO3[i][j]
                        +k4[i]*O1D[i][j]*N2[i]+
                        k5[i]*O1D[i][j]*O2[i]
                        +k40[i]*OH[i][j]*OH[i][j]+
                        k45[i]*H[i][j]*HO2[i][j]))
                        /
                        ( trans(1,O,i,j,rhop,rhom,M)+
                        1.0+DELTAT*(k1[i]*O2[i]+k2[i]*O3[i][j]
                        +k11[i]*NO2[i][j]+k16[i]*NO[i][j]+k29[i]*OH[i][j]
                        +k30[i]*HO2[i][j]+k31[i]*H2O2[i][j]+k47[i]*H2CO[i][j]+
                        k59[i]*ClO[i][j]+k73[i]*ClONO2[i][j]+k76[i]*H2[i]
                        +k82[i]*NO2[i][j]+k83[i]*NO3[i][j]+k84[i]*HNO4[i][j]+
                        k86[i]*CH3[i][j]+k100[i]*HCl[i][j]+
                        k101[i]*HOCl[i][j] ));
            if (converge[IO] == 1)
            {
                if (change(oldcon,O[i][j+1]) > crit)
              convergence = 0;
            }
          }

/* -2. --- O1D --- */
            if (solve[IO1D] == 1)
          {
            oldcon = O1D[i][j+1];
            O1D[i][j+1] = (trans(0,O1D,i,j,rhop,rhom,M)+
                        O1D[i][j]+DELTAT*(J3[i][j+1]*O3[i][j]+J5[i][j+1]*
                        N2O[i]))
                        /
                    ( trans(1,O1D,i,j,rhop,rhom,M)+
                        1.0+DELTAT*(k4[i]*N2[i] + k5[i]*O2[i] +k6[i]*O3[i][j]
                         +k7[i]*O3[i][j]+k8[i]*N2O[i]+k9[i]*N2O[i]+
                    k21[i]*H2O[i] +(k22[i]+k23[i])*CH4[i]+k24[i]*H2[i]
                         +k54[i]*CF2Cl2[i]+k55[i]*CFCl3[i]+k56[i]*CH3Cl[i]+
```

```
                    k61[i]*HCl[i][j] +  k77[i]*CCl4[i]));
            if (converge[IO1D] == 1)
            {
             if (change(oldcon,O1D[i][j+1]) > crit)
                convergence = 0;
             }
           }
/* -3. --- O3 --- */
            if (solve[IO3] == 1)
            {
                oldcon = O3[i][j+1];
                O3[i][j+1] = ( trans(0,O3,i,j,rhop,rhom,M)+
                  O3[i][j]+DELTAT* (k1[i]*O2[i]*O[i][j+1]))
                  /
                   ( trans(1,O3,i,j,rhop,rhom,M)+
                   1.0 + DELTAT* (J2[i][j+1]+J3[i][j+1]+k2[i]*O[i][j+1]+
                    (k6[i]+k7[i])*O1D[i][j+1]+  k10[i]*NO[i][j]
                    +k14[i]*NO2[i][j]  +k25[i]*H[i][j]+k27[i]*OH[i][j]+
                   k28[i]*HO2[i][j]+k58[i]*Cl[i][j] +k87[i]*CH3[i][j] ));
            if (converge[IO3] == 1)
            {
             if (change(oldcon,O3[i][j+1]) > crit)
                convergence = 0;
              }
           }

/*  OH first, then H */
/* -5. --- OH --- */

                /* short lifetime, no transport considered */
            if (solve[IOH] == 1)
            {
                oldcon = OH[i][j+1];
                a = DELTAT*(k40[i]+k43[i]);
                b = 1.0+DELTAT*(k27[i]*O3[i][j+1]+k29[i]*O[i][j+1]+
                        k32[i]*CH4[i]+ k34[i]*CO[i]+ k35[i]*H2[i]+
                        k36[i]*NO2[i][j]+k37[i]*HNO3[i][j]+
                        k38[i]*H2O2[i][j]+k39[i]*HO2[i][j]+k41[i]*H2CO[i][j]+
                        k51[i]*CH3OOH[i][j]+k57[i]*CH3Cl[i]+k67[i]*HCl[i][j]+
                        k74[i]*ClONO2[i][j]+k78[i]*HOCl[i][j]+
                        k81[i]*HNO4[i][j]+k94[i]*ClO[i][j]+k98[i]*CFCl3[i]+
                        k99[i]*CF2Cl2[i]);
                c = OH[i][j]+DELTAT*(J6[i][j+1]*HNO3[i][j]+2.0*J7[i][j+1]*
                        H2O2[i][j]+J15[i][j+1]*HOCl[i][j]+J17[i][j+1]*
                        CH3OOH[i][j]+J24[i][j+1]*H2O[i]+2.0*k21[i]*O1D[i][j+1]
                        *H2O[i]+k22[i]*O1D[i][j+1]*CH4[i]+k24[i]*O1D[i][j+1]*
                        H2[i]+k25[i]*H[i][j]*O3[i][j+1]+k28[i]*HO2[i][j]*
                        O3[i][j+1]+k30[i]*HO2[i][j]*O[i][j+1]+k31[i]*
```

```
                    H2O2[i][j]*O[i][j+1]+k33[i]*HO2[i][j]*NO[i][j]+
                    2.0*k46[i]*H[i][j]*HO2[i][j]+k47[i]*H2CO[i][j]*
                    O[i][j+1]+k61[i]*O1D[i][j+1]*HCl[i][j]+k71[i]*
                    H[i][j]*ClO[i][j]+k76[i]*H2[i]*O[i][j+1]+
                    k85[i]*H[i][j]*NO2[i][j]+k92[i]*Cl[i][j]*
                    HOCl[i][j]+k100[i]*O[i][j+1]*HCl[i][j]+
                    k101[i]*O[i][j+1]*HOCl[i][j]);

            OH[i][j+1] = 2.0 * c / ( b + sqrt(b*b + 4*a*c) );
        if (converge[IOH] == 1)
        {
            if (change(oldcon,OH[i][j+1]) > crit)
              convergence = 0;
         }
      }

/* -4. --- H --- */
         if (solve[IH] == 1)
         {
            oldcon = H[i][j+1];
            H[i][j+1] = (trans(0,H,i,j,rhop,rhom,M)+
              H[i][j]+DELTAT*(J9[i][j+1]*H2CO[i][j]+
              J14[i][j+1]*HCl[i][j]+
               J24[i][j+1]*H2O[i]+
                    k24[i]*O1D[i][j+1]*H2[i]+k29[i]*OH[i][j+1]*O[i][j+1]+
                    k34[i]*OH[i][j+1]*CO[i]+ k35[i]*OH[i][j+1]*H2[i]+
                    k64[i]*Cl[i][j]*H2[i]+k76[i]*H2[i]*O[i][j+1]))
                    /
                    (trans(1,H,i,j,rhop,rhom,M)+
                    1.0+DELTAT*(k25[i]*O3[i][j+1]+k26[i]*O2[i]+
                     (k44[i]+k45[i]+k46[i])*HO2[i][j]+k65[i]*HCl[i][j]+
                     k71[i]*ClO[i][j]+k85[i]*NO2[i][j]));
        if (converge[IH] == 1)
        {
            if (change(oldcon,H[i][j+1]) > crit)
              convergence = 0;
          }
        }
/* -6. --- HO2 --- */
         if (solve[IHO2] == 1)
         {
            oldcon = HO2[i][j+1];
            a = DELTAT*(k42[i]+k97[i]);
            b = 1.0 + DELTAT* (k28[i]*O3[i][j+1]+k30[i]*O[i][j+1]+
               k33[i]*NO[i][j]+k39[i]*OH[i][j+1]+(k44[i]+k45[i]+k46[i])*
               H[i][j+1]+k50[i]*CH3O2[i][j]+k63[i]*Cl[i][j]+k70[i]*
               ClO[i][j]+k79[i]*NO2[i][j]);
            c = HO2[i][j]+DELTAT*(J19[i][j+1]*HNO4[i][j]+k26[i]*H[i][j+1]*
```

```
                O2[i]+k27[i]*OH[i][j+1]*O3[i][j+1]+k31[i]*H2O2[i][j]*
                O[i][j+1]+k38[i]*OH[i][j+1]*H2O2[i][j]+k52[i]*CH3O[i][j]*
                O2[i]+k66[i]*Cl[i][j]*H2O2[i][j]);

            HO2[i][j+1] = 2.0 * c / ( b + sqrt(b*b + 4*a*c) );
        if (converge[IHO2] == 1)
        {
            if (change(oldcon,HO2[i][j+1]) > crit)
              convergence = 0;
            }
        }

/* -7. --- H2O2 --- */
         if (solve[IH2O2] == 1)
         {
            oldcon = H2O2[i][j+1];
            H2O2[i][j+1] = (/* trans(0,H2O2,i,j,rhop,rhom,M)+ */
                    H2O2[i][j]+DELTAT*(k42[i]*HO2[i][j+1]*
                    HO2[i][j+1]+k43[i]*OH[i][j+1]*OH[i][j+1]+
                    k97[i]*HO2[i][j+1]*HO2[i][j+1]))
                    /
                    (/* trans(1,H2O2,i,j,rhop,rhom,M)+ */
                    1.0+DELTAT* (J7[i][j+1]+k31[i]*O[i][j+1]+k38[i]*
                    OH[i][j+1]+k66[i]*Cl[i][j]));
        if (converge[IH2O2] == 1)
        {
            if (change(oldcon,H2O2[i][j+1]) > crit)
              convergence = 0;
          }
        }

/* 2) with NOx, and NO + NO2 = NOx  ======  */
/* -9. --- NO --- */
       if (solve[INO] == 1)
          {
            oldcon = NO[i][j+1];
            NO[i][j+1] = (/* trans(0,NO,i,j,rhop,rhom,M)+ */
                    NO[i][j]+DELTAT* (J4[i][j+1]*NOx[i]+
                    J8[i][j+1]*NO3[i][j]+2.0*k9[i]*O1D[i][j+1]*N2O[i]+
                    k11[i]*NOx[i]*O[i][j+1]+
                    k73[i]*ClONO2[i][j]*O[i][j+1]+
                    k85[i]*H[i][j+1]*NOx[i]))
                    /
                    (/* trans(1,NO,i,j,rhop,rhom,M)+ */
                    1.0+DELTAT*(J4[i][j+1] +
                    k10[i]*O3[i][j+1]+k11[i]*O[i][j+1]+
                    k15[i]*NO3[i][j]+k16[i]*O[i][j+1]+
                    k33[i]*HO2[i][j+1]+k49[i]*CH3O2[i][j]+
```

```
                    k60[i]*ClO[i][j]+k85[i]*H[i][j+1]));
            if (converge[INO] == 1)
            {
                if (change(oldcon,NO[i][j+1]) > crit)
                  convergence = 0;
              }
        }

/* -10. --- NO2 --- */
            if (solve[INO2] == 1)
            {
                oldcon = NO2[i][j+1];
                NO2[i][j+1] = (/* trans(0,NO2,i,j,rhop,rhom,M)+ */
                        NO2[i][j]+DELTAT*(J6[i][j+1]*HNO3[i][j]+
                        J16[i][j+1]*N2O5[i][j]+J19[i][j+1]*HNO4[i][j]+
                        J22[i][j+1]*NO3[i][j]+k10[i]*NOx[i]*O3[i][j+1]+
                        2.0*k15[i]*NOx[i]*NO3[i][j]+k16[i]*NOx[i]*
                        O[i][j+1]+k33[i]*HO2[i][j+1]*NOx[i]+k49[i]*
                        CH3O2[i][j]*NOx[i]+k60[i]*ClO[i][j]*NOx[i]+
                        k81[i]*HNO4[i][j]*OH[i][j+1]+k83[i]*O[i][j+1]*
                        NO3[i][j]+k93[i]*Cl[i][j]*NO3[i][j]))
                        /
                        (/* trans(1,NO2,i,j,rhop,rhom,M)+ */
                        1.0+DELTAT*(J4[i][j+1]+k10[i]*O3[i][j+1]+
                        k11[i]*O[i][j+1]+k14[i]*O3[i][j+1]+
                        2.0*k15[i]*NO3[i][j]+k16[i]*O[i][j+1]+
                        k17[i]*NO3[i][j]+k33[i]*HO2[i][j+1]+
                        k36[i]*OH[i][j+1]+k49[i]*CH3O2[i][j]+
                        k60[i]*ClO[i][j]+k72[i]*ClO[i][j]+
                        k79[i]*HO2[i][j+1]+
                        k82[i]*O[i][j+1]+k85[i]*H[i][j+1]));
            if (converge[INO2] == 1)
            {
                if (change(oldcon,NO2[i][j+1]) > crit)
                  convergence = 0;
                }
        }

/* -11. --- NO3 --- */
            if (solve[INO3] == 1)
            {
              oldcon = NO3[i][j+1];
              NO3[i][j+1] = (/* trans(0,NO3,i,j,rhop,rhom,M)+ */
                        NO3[i][j]+DELTAT* (J16[i][j+1]*N2O5[i][j] +
                        k14[i]*NO2[i][j+1]*O3[i][j+1]+
                        k37[i]*OH[i][j+1]*HNO3[i][j]+k74[i]*
                        ClONO2[i][j]*OH[i][j+1]+k82[i]*O[i][j+1]*
                        NO2[i][j+1]))
```

```
                    /
                    (/* trans(1,NO3,i,j,rhop,rhom,M)+ */
                    1.0+DELTAT*(J8[i][j+1]+J22[i][j+1]+
                    k15[i]*NO[i][j+1]+k17[i]*NO2[i][j+1]+
                    k83[i]*O[i][j+1]+k93[i]*Cl[i][j]));
          if (converge[INO3] == 1)
          {
           if (change(oldcon,NO3[i][j+1]) > crit)
                convergence = 0;
           }
         }

/* -12. --- N2O5 --- */
         if (solve[IN2O5] == 1)
         {
           oldcon = N2O5[i][j+1];
           N2O5[i][j+1] = (/* trans(0,N2O5,i,j,rhop,rhom,M)+ */
                    N2O5[i][j]+DELTAT*k17[i]*NO2[i][j+1]*
                    NO3[i][j+1])
                    /
                    (/* trans(1,N2O5,i,j,rhop,rhom,M)+ */
                    1.0+DELTAT*J16[i][j+1]);
          if (converge[IN2O5] == 1)
          {
           if (change(oldcon,N2O5[i][j+1]) > crit)
                convergence = 0;
              }
        }

/* -13. --- HNO3 --- */
         if (solve[IHNO3] == 1)
         {
           oldcon = HNO3[i][j+1];
           HNO3[i][j+1] =(/* trans(0,HNO3,i,j,rhop,rhom,M)+ */
                    HNO3[i][j]+DELTAT*k36[i]*OH[i][j+1]*NO2[i][j+1])
                    /
                    (/* trans(1,HNO3,i,j,rhop,rhom,M)+ */
                    1.0+DELTAT*(J6[i][j+1]+k37[i]*OH[i][j+1]));
          if (converge[IHNO3] == 1)
          {
          if (change(oldcon,HNO3[i][j+1]) > crit)
                convergence = 0;
              }
        }

/* -14. --- HNO4 --- */
         if (solve[IHNO4] == 1)
         {
```

```
            oldcon = HNO4[i][j+1];
            HNO4[i][j+1] = (/* trans(0,HNO4,i,j,rhop,rhom,M)+ */
                        HNO4[i][j]+DELTAT*k79[i]*HO2[i][j+1]*
                        NO2[i][j+1])
                        /
                        (/* trans(1,HNO4,i,j,rhop,rhom,M)+ */
                        1.0+DELTAT*(J19[i][j+1]+k81[i]*OH[i][j+1]+
                        k84[i]*O[i][j+1]));
          if (converge[IHNO4] == 1)
          {
           if (change(oldcon,HNO4[i][j+1]) > crit)
                convergence = 0;
              }
        }

/* solve ClO first, then Cl */
/* -16. --- ClO --- */
   if (solve[IClO] == 1)
     {
      oldcon = ClO[i][j+1];
      a = DELTAT * (k115[i]+k128[i]);
      b = ( trans(1,ClO,i,j,rhop,rhom,M)+
                1.0+DELTAT*(J20[i][j+1]+
                k59[i]*O[i][j+1]+k60[i]*NO[i][j+1]
                +k70[i]*HO2[i][j+1]+k71[i]*H[i][j+1]
                +k72[i]*NO2[i][j+1]
                +k94[i]*OH[i][j+1]+
                k95[i]*CH4[i]+k96[i]*H2[i]
                +k111[i]*NO3[i][j+1] +k112[i]*H2CO[i][j]
                +k113[i]*CO[i] +k114[i]*N2O[i]
                +k116[i]*O3[i][j+1]));
        c = ( trans(0,ClO,i,j,rhop,rhom,M)+
                ClO[i][j]+DELTAT*(J23[i][j+1]*ClOO[i][j]
                +J25[i][j+1]*OClO[i][j]
                +k54[i]*O1D[i][j+1]*CF2Cl2[i]+
                k55[i]*O1D[i][j+1]*CFCl3[i]+
                k58[i]*Cl[i][j]*O3[i][j+1]+
                k73[i]*ClONO2[i][j]*O[i][j+1]
                +k77[i]*O1D[i][j+1]*CCl4[i]
                +k78[i]*OH[i][j+1]*HOCl[i][j]
                +k93[i]*Cl[i][j]*NO3[i][j+1]
                +k103[i]*Cl[i][j]*HO2[i][j+1]
                +2.0*k105[i]*Cl[i][j]*OClO[i][j]
                +2.0*k107[i]*Cl[i][j]*ClOO[i][j]
                +k120[i]*OClO[i][j]*O[i][j+1]
                +k123[i]*OClO[i][j]*NO[i][j+1]));

        ClO[i][j+1] = 2.0 * c/(b + sqrt(b*b + 4.0*a*c));
```

```
        if (converge[IClO] == 1)
        {
         if (change(oldcon,ClO[i][j+1]) > crit)
               convergence = 0;
            }
      }

/* -15. --- Cl --- */
        if (solve[ICl] == 1)
        {
          oldcon = Cl[i][j+1];
          Cl[i][j+1] = ( trans(0,Cl,i,j,rhop,rhom,M)+
                        Cl[i][j]+DELTAT*(J11[i][j+1]*CF2Cl2[i]+
                        J12[i][j+1]*CFCl3[i]+J13[i][j+1]*ClONO2[i][j]+
                        J14[i][j+1]*HCl[i][j]+J15[i][j+1]*HOCl[i][j]
                        +J18[i][j+1]*CCl4[i] + J20[i][j+1]*ClO[i][j+1]
                        +2.0*J21[i][j+1]*Cl2[i][j]
                        +J27[i][j+1]*Cl2O2[i][j]
                        +J28[i][j+1]*ClNO[i][j]
                        +k56[i]*O1D[i][j+1]*CH3Cl[i]
                        +k57[i]*OH[i][j+1]*CH3Cl[i]
                        +k59[i]*ClO[i][j+1]*O[i][j+1]
                        +k60[i]*ClO[i][j+1]*NO[i][j+1]
                        +k61[i]*O1D[i][j+1]*HCl[i][j]
                        +k65[i]*HCl[i][j]*H[i][j+1]
                        +k67[i]*OH[i][j+1]*HCl[i][j]
                        +k71[i]*H[i][j+1]*ClO[i][j+1]
                        +k100[i]*O[i][j+1]*HCl[i][j]
                        +k117[i]*OH[i][j+1]*Cl2[i][j]))
                        /
                        ( trans(1,Cl,i,j,rhop,rhom,M)+
                        1.0+DELTAT*(k58[i]*O3[i][j+1]+k62[i]*CH4[i]
                        +k63[i]*HO2[i][j+1]+k64[i]*H2[i]
                        +k66[i]*H2O2[i][j+1]+
                         k68[i]*O2[i]+
                         k75[i]*H2CO[i][j] +
                        k91[i]*CH3Cl[i]+k92[i]*HOCl[i][j]+
                        k93[i]*NO3[i][j+1]
                        +k103[i]*HO2[i][j+1]
                        +k105[i]*OClO[i][j]
                        +(k106[i]+k107[i])*ClOO[i][j]
                        +k108[i]*Cl2O2[i][j]
                        +k109[i]*ClONO2[i][j]
                        +k110[i]*ClNO[i][j]
                        +k124[i]*NO[i][j+1]
                        +(k125[i]+k126[i])*NO2[i][j+1]
                        +k127[i]*CO[i]));
```

```
          if (converge[ICl] == 1)
          {
           if (change(oldcon,Cl[i][j+1]) > crit)
                convergence = 0;
              }
        }
/* -17. --- HOCl --- */
          if (solve[IHOCl] == 1)
          {
            oldcon = HOCl[i][j+1];
            HOCl[i][j+1] = (trans(0,HOCl,i,j,rhop,rhom,M)+
                        HOCl[i][j]+DELTAT*(k70[i]*HO2[i][j+1]*
                        ClO[i][j+1]+k74[i]*ClONO2[i][j]*OH[i][j+1]))
                        /
                        ( trans(1,HOCl,i,j,rhop,rhom,M)+
                        1.0+DELTAT*(J15[i][j+1]+k78[i]*OH[i][j+1]+
                        k101[i]*O[i][j+1]));
           if (converge[IHOCl] == 1)
           {
           if (change(oldcon,HOCl[i][j+1]) > crit)
                convergence = 0;
              }
        }

/* -18. --- ClONO2 --- */
           if (solve[IClONO2] == 1)
          {
            oldcon = ClONO2[i][j+1];
            ClONO2[i][j+1] = ( trans(0,ClONO2,i,j,rhop,rhom,M)+
                        ClONO2[i][j]+DELTAT*k72[i]*ClO[i][j+1]*
                        NO2[i][j+1])
                        /
                        ( trans(1,ClONO2,i,j,rhop,rhom,M)+
                        1.0+DELTAT*(J13[i][j+1]+k73[i]*O[i][j+1]+
                        k74[i]*OH[i][j+1]));
           if (converge[IClONO2] == 1)
           {
           if (change(oldcon,ClONO2[i][j+1]) > crit)
                convergence = 0;
              }
        }

/* -19. --- CH3 --- */
           if (solve[ICH3] == 1)
          {
            oldcon = CH3[i][j+1];
            CH3[i][j+1] = (CH3[i][j]+DELTAT*( /* J20[i][j+1]*CH3O2[i][j]+
                        J23[i][j+1]*CH4[i]+ */ k22[i]*O1D[i][j+1]*CH4[i]+
```

```
                    k32[i]*OH[i][j+1]*CH4[i]+k62[i]*Cl[i][j+1]*CH4[i]))
                    /
                    (1.0+DELTAT*(k48[i]*O2[i]+k86[i]*O[i][j+1]+
                    k87[i]*O3[i][j+1]));
          if (converge[ICH3] == 1)
          {
           if (change(oldcon,CH3[i][j+1]) > crit)
                convergence = 0;
              }
        }

/* -20. --- CH3O --- */
          if (solve[ICH3O] == 1)
          {
            oldcon = CH3O[i][j+1];
            CH3O[i][j+1] = (CH3O[i][j]+DELTAT*(J17[i][j+1]*CH3OOH[i][j]+
                    k49[i]*CH3O2[i][j]*NO[i][j+1]+
                    2.0*k88[i]*CH3O2[i][j]*CH3O2[i][j]))
                    /
                    (1.0+DELTAT*k52[i]*O2[i]);
          if (converge[ICH3O] == 1)
          {
           if (change(oldcon,CH3O[i][j+1]) > crit)
              convergence = 0;
              }
        }

/* -21. --- CH3O2 --- */

         if (solve[ICH3O2] == 1)
          {
            oldcon = CH3O2[i][j+1];
                a = DELTAT * (k88[i] + k89[i] + k90[i]);
                b = 1.0 + DELTAT *(/* J20[i][j+1]+ */ k49[i]*NO[i][j+1]+
                        k50[i]*HO2[i][j+1]);
                c = DELTAT * (k48[i]*CH3[i][j+1]*O2[i]+
                        k51[i]*CH3OOH[i][j+1]*OH[i][j+1]);
                CH3O2[i][j+1] = 2.0 * c / ( b + sqrt(b*b + 4*a*c) );

           if (converge[ICH3O2] == 1)
           {
            if (change(oldcon,CH3O2[i][j+1]) > crit)
              convergence = 0;
              }
        }

/* -22. --- CH3OOH --- */
          if (solve[ICH3OOH] == 1)
```

```
        {
          oldcon = CH3OOH[i][j+1];
          CH3OOH[i][j+1] = (CH3OOH[i][j]+DELTAT*k50[i]*CH3O2[i][j+1]*
                    HO2[i][j+1])
                    /
                    (1.0+DELTAT*(J17[i][j+1]+k51[i]*OH[i][j+1]));
         if (converge[ICH3OOH] == 1)
         {
          if (change(oldcon,CH3OOH[i][j+1]) > crit)
            convergence = 0;
             }
       }

/* -23. --- H2CO --- */
         if (solve[IH2CO] == 1)
         {
          oldcon = H2CO[i][j+1];
          H2CO[i][j+1] = (H2CO[i][j]+DELTAT*(k23[i]*O1D[i][j+1]*CH4[i]+
                    k52[i]*CH3O[i][j+1]*O2[i]+
                    k89[i]*CH3O2[i][j+1]*CH3O2[i][j+1]))
                    /
                    (1.0+DELTAT*(J9[i][j+1]+J10[i][j+1]+k41[i]*OH[i][j+1]+
                    k47[i]*O[i][j+1]+k75[i]*Cl[i][j+1]));
         if (converge[IH2CO] == 1)
         {
          if (change(oldcon,H2CO[i][j+1]) > crit)
            convergence = 0;
             }
       }

/* -24. --- HCl --- */
         if (solve[IHCl] == 1)
         {
          oldcon = HCl[i][j+1];
          HCl[i][j+1] = ( trans(0,HCl,i,j,rhop,rhom,M)+
                    HCl[i][j]+DELTAT*(k62[i]*Cl[i][j+1]*CH4[i]+
                    k63[i]*Cl[i][j+1]*HO2[i][j+1]+k64[i]*Cl[i][j+1]*H2[i]+
                    k66[i]*Cl[i][j+1]*H2O2[i][j+1]+k75[i]*H2CO[i][j+1]*
                    Cl[i][j+1]+k91[i]*Cl[i][j+1]*CH3Cl[i]))
                    /
                    ( trans(1,HCl,i,j,rhop,rhom,M)+
                    1.0+DELTAT*(J14[i][j+1]+k61[i]*O1D[i][j+1]+k65[i]*
                    H[i][j+1]+k67[i]*OH[i][j+1]+k100[i]*O[i][j+1]));
         if (converge[IHCl] == 1)
         {
          if (change(oldcon,HCl[i][j+1]) > crit)
            convergence = 0;
         }
```

```
        }
/* -25. --- Cl2 --- */
        if (solve[ICl2] == 1)
        {
          oldcon = Cl2[i][j+1];
          Cl2[i][j+1] = (/* trans(0,Cl2,i,j,rhop,rhom,M)+ */
                Cl2[i][j] +DELTAT*(k92[i]*Cl[i][j+1]*HOCl[i][j+1]
                +k110[i]*Cl[i][j+1]*ClNO[i][j]))
                /
                (/*  trans(1,Cl2,i,j,rhop,rhom,M)+ */
                1.0 +DELTAT*(J21[i][j+1] +k102[i]*O1D[i][j+1]
                +k117[i]*OH[i][j+1]));
         if (converge[ICl2] == 1)
         {
          if (change(oldcon,Cl2[i][j+1]) > crit)
            convergence = 0;
         }
        }

/* -26. --- ClOO --- */
        if (solve[IClOO] == 1)
        {
          oldcon = ClOO[i][j+1];
          ClOO[i][j+1] = (/* trans(0,ClOO,i,j,rhop,rhom,M)+ */
                ClOO[i][j] +DELTAT*(k68[i]*Cl[i][j+1]*O2[i]))
                /
                (/*  trans(1,ClOO,i,j,rhop,rhom,M)+ */
                1.0 +DELTAT*(J23[i][j+1]
                +(k106[i] +k107[i])*Cl[i][j+1]));
         if (converge[IClOO] == 1)
         {
          if (change(oldcon,ClOO[i][j+1]) > crit)
            convergence = 0;
         }
        }

/* -27. --- OClO --- */
        if (solve[IOClO] == 1)
        {
          oldcon = OClO[i][j+1];
          OClO[i][j+1] = (/* trans(0,OClO,i,j,rhop,rhom,M)+ */
                OClO[i][j] +DELTAT*(k116[i]*ClO[i][j+1]*O3[i][j+1]))
                /
                (/*  trans(1,OClO,i,j,rhop,rhom,M)+ */
                1.0 +DELTAT*(J25[i][j+1] +k105[i]*Cl[i][j+1]
                +k120[i]*O[i][j+1] +k121[i]*O3[i][j+1]
                +k122[i]*OH[i][j+1] +k123[i]*NO[i][j+1]));
         if (converge[IOClO] == 1)
```

```
        {
         if (change(oldcon,OClO[i][j+1]) > crit)
            convergence = 0;
        }
      }

/* -28. --- Cl2O2--- */
      if (solve[ICl2O2] == 1)
      {
        oldcon = Cl2O2[i][j+1];
        Cl2O2[i][j+1] = (/* trans(0,Cl2O2,i,j,rhop,rhom,M)+ */
              Cl2O2[i][j] +DELTAT*(k128[i]*ClO[i][j+1]*ClO[i][j+1]))
              /
              (/*  trans(1,Cl2O2,i,j,rhop,rhom,M)+ */
              1.0 +DELTAT*(J27[i][j+1] +k108[i]*Cl[i][j+1]));
       if (converge[ICl2O2] == 1)
        {
         if (change(oldcon,Cl2O2[i][j+1]) > crit)
            convergence = 0;
        }
      }

/* -29. --- ClNO --- */
      if (solve[IClNO] == 1)
      {
        oldcon = ClNO[i][j+1];
        ClNO[i][j+1] = (/* trans(0,ClNO,i,j,rhop,rhom,M)+ */
              ClNO[i][j] +DELTAT*(k124[i]*Cl[i][j+1]*NO[i][j+1]))
              /
              (/*  trans(1,ClNO,i,j,rhop,rhom,M)+ */
              1.0 +DELTAT*(J28[i][j+1] +k110[i]*Cl[i][j+1]));
       if (converge[IClNO] == 1)
        {
         if (change(oldcon,ClNO[i][j+1]) > crit)
            convergence = 0;
        }
      }

/* -30. --- ClNO2--- */
      if (solve[IClNO2] == 1)
      {
        oldcon = ClNO2[i][j+1];
        ClNO2[i][j+1] = (/* trans(0,ClNO2,i,j,rhop,rhom,M)+ */
              ClNO2[i][j] +DELTAT*(k126[i]*Cl[i][j+1]*NO2[i][j+1]))
              /
              (/*  trans(1,ClNO2,i,j,rhop,rhom,M)+ */
              1.0 +DELTAT*(J29[i][j+1] +k109[i]*Cl[i][j+1]));
       if (converge[IClNO2] == 1)
```

```
        {
         if (change(oldcon,ClNO2[i][j+1]) > crit)
            convergence = 0;
        }
      }

/* -31. --- ClONO--- */
       if (solve[IClONO] == 1)
       {
         oldcon = ClONO[i][j+1];
         ClONO[i][j+1] = (/* trans(0,ClONO,i,j,rhop,rhom,M)+ */
              ClONO[i][j] +DELTAT*(k125[i]*Cl[i][j+1]*NO2[i][j+1]))
              /
              (/*  trans(1,ClONO,i,j,rhop,rhom,M)+ */
              1.0 +DELTAT*(J30[i][j+1]));
        if (converge[IClONO] == 1)
        {
         if (change(oldcon,ClONO[i][j+1]) > crit)
            convergence = 0;
        }
       }
     }                  /* ---- i     loop ---- */

    /* now put the last element in the first element;
       this is because the time is cyclical. */
      if (j == TIME_POINTS-1)
        for (i=0;i<SPACE_POINTS;i++)
        {
         O[i][0] = O[i][TIME_POINTS];
         O1D[i][0] = O1D[i][TIME_POINTS];
         O3[i][0] = O3[i][TIME_POINTS];
         H[i][0] = H[i][TIME_POINTS];
         OH[i][0] = OH[i][TIME_POINTS];
         HO2[i][0] = HO2[i][TIME_POINTS];
         H2O2[i][0] = H2O2[i][TIME_POINTS];
         NO[i][0] = NO[i][TIME_POINTS];
         NO2[i][0] = NO2[i][TIME_POINTS];
         NO3[i][0] = NO3[i][TIME_POINTS];
         N2O5[i][0] = N2O5[i][TIME_POINTS];
         HNO3[i][0] = HNO3[i][TIME_POINTS];
         HNO4[i][0] = HNO4[i][TIME_POINTS];
         Cl[i][0] = Cl[i][TIME_POINTS];
         ClO[i][0] = ClO[i][TIME_POINTS];
         HOCl[i][0] = HOCl[i][TIME_POINTS];
         ClONO2[i][0] = ClONO2[i][TIME_POINTS];
         CH3[i][0] = CH3[i][TIME_POINTS];
         CH3O[i][0] = CH3O[i][TIME_POINTS];
         CH3O2[i][0] = CH3O2[i][TIME_POINTS];
```

```
            CH3OOH[i][0] = CH3OOH[i][TIME_POINTS];
            H2CO[i][0] = H2CO[i][TIME_POINTS];
            HCl[i][0] = HCl[i][TIME_POINTS];
            Cl2[i][0] = Cl2[i][TIME_POINTS];
            OClO[i][0] = OClO[i][TIME_POINTS];
            ClOO[i][0] = ClOO[i][TIME_POINTS];
            Cl2O2[i][0] = Cl2O2[i][TIME_POINTS];
            ClNO[i][0] = ClNO[i][TIME_POINTS];
            ClONO[i][0] = ClONO[i][TIME_POINTS];
            ClNO2[i][0] = ClNO2[i][TIME_POINTS];
           }
   }            /* ---- j     loop ---- */
 }              /* ---- while loop ---- */
 fprintf(stderr,"\n");
}               /* ---- main loop --- */
```

5. /*=== File: O3init.c

```
 *
 * Purpose: solve for the initial value of various species in the O3 model.
 * They are the starting points in the file "O3_cons.c".
 *
 */

#include <math.h>
#include <stdio.h>
#include "O3model.h"
#include "O3util.h"
#define DELTAT (86400.0/TIME_POINTS)

void get_initial_cons(int *solve)
{
  int i, j;
  double a, b, c;

  for (j= 4;j<TIME_POINTS-5;j++)
     {
       for (i=0;i<SPACE_POINTS;i++)
       {
/* -1. --- O --- */
            if (solve[IO] == 1)
              O[i][j+1] =(2.0 * J1[i][j+1]*O2[i] +J2[i][j+1]*O3[i][j+1])
                        /
                        (k1[i]*O2[i]+k2[i]*O3[i][j+1]+k76[i]*H2[i]);
          else
                O[i][j+1]=0.0;

 /* -2. --- O1D --- */
            if (solve[IO1D] == 1)
```

```
            O1D[i][j+1] = (J3[i][j+1]*O3[i][j+1]+J5[i][j+1]*N2O[i])
                          /
                    (k4[i]*N2[i] + k5[i]*O2[i]+(k6[i]+k7[i])*O3[i][j+1]+
                        k8[i]*N2O[i]+ k9[i]*N2O[i]+k21[i]*H2O[i] +
                        (k22[i]+k23[i])*CH4[i]+k24[i]*H2[i]);
          else
            O1D[i][j+1] =0.0;

/* -3. --- O3 --- */
           if (solve[IO3] == 1)
              O3[i][j+1] = (k1[i]*O2[i]*O[i][j+1])
                           /
                 (J2[i][j+1]+J3[i][j+1]+k2[i]*O[i][j+1]+
                 (k6[i]+k7[i])*O1D[i][j+1]);
           else
              O3[i][j+1]= 0.0;

/* -5. --- OH --- */
           if (solve[IOH] == 1)
            {
              a = (k40[i]+k43[i]);
              b = (k27[i]*O3[i][j+1]+k29[i]*O[i][j+1]+
                        k32[i]*CH4[i]+ k34[i]*CO[i] +k35[i]*H2[i]);
              c = (J24[i][j+1]*H2O[i]+2.0*k21[i]*O1D[i][j+1]
                        *H2O[i]+k22[i]*O1D[i][j+1]*CH4[i]+k24[i]*O1D[i][j+1]*
                        H2[i]+k25[i]*H[i][j]*O3[i][j+1]
                        +k76[i]*H2[i]*O[i][j+1]);
              OH[i][j+1] = 2.0 * c / ( b + sqrt(b*b + 4*a*c));
            }
           else
              OH[i][j+1] = 0.0;

/* -4. --- H --- */
           if (solve[IH] == 1)
              H[i][j+1] = (J24[i][j+1]*H2O[i]+k24[i]*O1D[i][j+1]*H2[i]+
                        k29[i]*OH[i][j+1]*O[i][j+1]+k34[i]*OH[i][j+1]*CO[i]+
                        k35[i]*OH[i][j+1]*H2[i]+k76[i]*H2[i]*O[i][j+1])
                        /
                        (k25[i]*O3[i][j+1]+k26[i]*O2[i]);
            else
              H[i][j+1]=0.0;

/* -6. --- HO2 --- */
           if (solve[IHO2] == 1)
              {
              a =  k42[i]+k97[i];
              b = (k28[i]*O3[i][j+1]+k30[i]*O[i][j+1]+
                 k39[i]*OH[i][j+1]+(k44[i]+k45[i]+k46[i])*H[i][j+1]);
```

```
                c = k26[i]*H[i][j+1]*O2[i]+k27[i]*OH[i][j+1]*O3[i][j+1];
                HO2[i][j+1] = 2.0 * c / ( b + sqrt(b*b + 4*a*c) );
                }
            else
                HO2[i][j+1] = 0.0;

/* -7. --- H2O2 --- */
            if (solve[IH2O2] == 1)
                H2O2[i][j+1] = (k42[i]*HO2[i][j+1]*
                        HO2[i][j+1]+k43[i]*OH[i][j+1]*OH[i][j+1]+
                        k97[i]*HO2[i][j+1]*HO2[i][j+1])
                        /
                        (J7[i][j+1]+k31[i]*O[i][j+1]+k38[i]*OH[i][j+1]);
             else
                H2O2[i][j+1] = 0.0;

/* initial NO, NO2 with NOx, NO + NO2 = NOx  */
/* -9. --- NO --- */
            if (solve[INO] == 1)
                NO[i][j+1]=(J4[i][j+1]*NOx[i]+
                        2.0*k9[i]*O1D[i][j+1]*N2O[i]+k11[i]*NOx[i]*O[i][j+1]
                        +k85[i]*H[i][j+1]*NOx[i])
                        /
                        (J4[i][j+1]+ k10[i]*O3[i][j+1]+k11[i]*O[i][j+1]+
                        k16[i]*O[i][j+1]+ k33[i]*HO2[i][j+1]+
                         +k85[i]*H[i][j+1]);
            else
                NO[i][j+1] = 0.0;

/* -10. --- NO2 --- */
            if (solve[INO2] == 1)
                NO2[i][j+1] = (k10[i]*NOx[i]*O3[i][j+1]+
                        k16[i]*NOx[i]*O[i][j+1]+
                        k33[i]*HO2[i][j+1]*NOx[i])
                        /
                        (J4[i][j+1]+ k10[i]*O3[i][j+1]+
                        k11[i]*O[i][j+1]+ k14[i]*O3[i][j+1]+
                        k16[i]*O[i][j+1]+k33[i]*HO2[i][j+1]+
                        k36[i]*OH[i][j+1]+
                        k79[i]*HO2[i][j+1]+k82[i]*O[i][j+1]+
                        k85[i]*H[i][j+1]);
            else
                NO2[i][j+1] = 0.0;

/* -11. --- NO3 --- */
           if (solve[INO3] == 1)
              NO3[i][j+1] = (k14[i]*NO2[i][j+1]*O3[i][j+1]+
                        k82[i]*O[i][j+1]*NO2[i][j+1])
```

```
                    /
                    (J8[i][j+1]+J22[i][j+1]+
                    k15[i]*NO[i][j+1]+k17[i]*NO2[i][j+1]+
                    k83[i]*O[i][j+1]);
            else
                NO3[i][j+1] = 0.0;

/* -12. --- N2O5 --- */
          if (solve[IN2O5] == 1)
            N2O5[i][j+1] = k17[i]*NO2[i][j+1]*NO3[i][j+1]
                    /
                    J16[i][j+1];
            else
                N2O5[i][j+1] = 0.0;

/* -13. --- HNO3 --- */
          if (solve[IHNO3] == 1)
            HNO3[i][j+1] =k36[i]*OH[i][j+1]*NO2[i][j+1]
                    /
                    (J6[i][j+1]+k37[i]*OH[i][j+1]);
            else
                HNO3[i][j+1] = 0.0;

/* -14. --- HNO4 --- */
          if (solve[IHNO4] == 1)
            HNO4[i][j+1] = k79[i]*HO2[i][j+1]*NO2[i][j+1]
                    /
                    (J19[i][j+1]+k81[i]*OH[i][j+1]+k84[i]*O[i][j+1]);
            else
                HNO4[i][j+1] = 0.0;

/* -16. --- ClO --- */
          if (solve[IClO] == 1)
            ClO[i][j+1] = (k54[i]*O1D[i][j+1]*CF2Cl2[i]+
                 k55[i]*O1D[i][j+1]*CFCl3[i]+k58[i]*Cl[i][j]*O3[i][j+1]
                  +k77[i]*O1D[i][j+1]*CCl4[i])
                    /
                    (J20[i][j+1]+k59[i]*O[i][j+1]+k60[i]*NO[i][j+1]+
                    k70[i]*HO2[i][j+1]+k71[i]*H[i][j+1]+
                    k72[i]*k72[i]*NO2[i][j+1]
                    +k94[i]*OH[i][j+1]+k95[i]*CH4[i]+k96[i]*H2[i]);
            else
                ClO[i][j+1] = 0.0;

/* -15. --- Cl --- */
           if (solve[ICl] == 1)
            Cl[i][j+1] = (J11[i][j+1]*CF2Cl2[i]+J12[i][j+1]*CFCl3[i]
                 +J18[i][j+1]*CCl4[i]+ J20[i][j+1]*ClO[i][j+1]+
```

```
                k56[i]*O1D[i][j+1]*CH3Cl[i]+
                    k57[i]*OH[i][j+1]*CH3Cl[i]+
                    k59[i]*ClO[i][j+1]*O[i][j+1]+
                    k60[i]*ClO[i][j+1]*NO[i][j+1]+
                    k71[i]*ClO[i][j+1]*H[i][j+1])
                    /
                    (k58[i]*O3[i][j+1]+k62[i]*CH4[i]+k63[i]*
                    HO2[i][j+1]+k64[i]*H2[i]+k66[i]*H2O2[i][j+1]
                    +k91[i]*CH3Cl[i]+k93[i]*NO3[i][j+1]);
            else
                Cl[i][j+1] = 0.0;

/* -17. --- HOCl --- */
          if (solve[IHOCl] == 1)
            HOCl[i][j+1] = (k70[i]*HO2[i][j+1]*ClO[i][j+1])
                    /
                    (J15[i][j+1]+k78[i]*OH[i][j+1]);
            else
                HOCl[i][j+1] = 0.0;

/* -18. --- ClONO2 --- */
           if (solve[IClONO2] == 1)
            ClONO2[i][j+1] = k72[i]*ClO[i][j+1]*NO2[i][j+1]
                    /
                    (J13[i][j+1]+k73[i]*O[i][j+1]+
                    k74[i]*OH[i][j+1]);
            else
                ClONO2[i][j+1] = 0.0;

/* -19. --- CH3 --- */
           if (solve[ICH3] == 1)
            CH3[i][j+1] = (/* J23[i][j+1]*CH4[i]+ */
                    k22[i]*O1D[i][j+1]*CH4[i]+
                    k32[i]*OH[i][j+1]*CH4[i]+k62[i]*Cl[i][j+1]*CH4[i])
                    /
                    (k48[i]*O2[i]+k86[i]*O[i][j+1]+k87[i]*O3[i][j+1]);
            else
                CH3[i][j+1] = 0.0;

/* -21. --- CH3O2 --- */
          if (solve[ICH3O2] == 1)
                {
                a = k88[i] + k89[i] + k90[i];
                b = /* J20[i][j+1] */ +k49[i]*NO[i][j+1]+
                        k50[i]*HO2[i][j+1];
                c = (k48[i]*CH3[i][j+1]*O2[i]);
                CH3O2[i][j+1] = 2.0 * c / (b + sqrt(b*b + 4*a*c));
                }
```

```
            else
                CH3O2[i][j+1] = 0.0;

/* -20. --- CH3O --- */
            if (solve[ICH3O] == 1)
              CH3O[i][j+1] = (k49[i]*CH3O2[i][j+1]*NO[i][j+1]+
                              2.0*k88[i]*CH3O2[i][j+1]*CH3O2[i][j+1])
                             /
                             (k52[i]*O2[i]);
              else
                CH3O[i][j+1] = 0.0;

/* -22. --- CH3OOH --- */
            if (solve[ICH3OOH] == 1)
              CH3OOH[i][j+1] = (k50[i]*CH3O2[i][j+1]*HO2[i][j+1])
                             /
                             (J17[i][j+1]+k51[i]*OH[i][j+1]);
              else
                CH3OOH[i][j+1] = 0.0;

/* -23. --- H2CO --- */
            if (solve[IH2CO] == 1)
              H2CO[i][j+1] = (k23[i]*O1D[i][j+1]*CH4[i]+
                              k52[i]*CH3O[i][j+1]*O2[i]+
                              k89[i]*CH3O2[i][j+1]*CH3O2[i][j+1])
                             /
                             (J9[i][j+1]+J10[i][j+1]+k41[i]*OH[i][j+1]+
                             k47[i]*O[i][j+1]+k75[i]*Cl[i][j+1]);
              else
                H2CO[i][j+1] = 0.0;

/* -24. --- HCl --- */
            if (solve[IHCl] == 1)
              HCl[i][j+1] = (k62[i]*Cl[i][j+1]*CH4[i]+
                             k63[i]*Cl[i][j+1]*HO2[i][j+1]+k64[i]*Cl[i][j+1]*H2[i]+
                             k66[i]*Cl[i][j+1]*H2O2[i][j+1]+k75[i]*H2CO[i][j+1]*
                             Cl[i][j+1])
                             /
                             (J14[i][j+1]+k61[i]*O1D[i][j+1]+k65[i]*
                             H[i][j+1]+k67[i]*OH[i][j+1]+k100[i]*O[i][j+1]);
              else
                HCl[i][j+1] = 0.0;

      }          /* ---- i     loop ---- */
    }            /* ---- j     loop ---- */
}       /* ---- main loop ---- */
```

```
6. /* === File: O3j.c
 *
 * Include functions:
 *              1. == double Actinic(int,double,int)
 *     double Actinic(int wavelength_bin, double angle, int height_bin)
 *              2. == File: j.c including all --getJ*-- functions:
 *                 double getJ*(double temperature,int timestep,int k)
 *              3. == void get_J(double *T,double *M, double *P)
 *
 * Purpose: Determine the J values for all of the reactions in the model
 *       for each of the atmospheric layers at each time step,i.e.,
 *              J[s][t].
 */
#include <stdio.h>
#include <math.h>
#include "O3model.h"
#include "O3util.h"

/* 1. == Function: Actinic(int,double,int)
 *   double Actinic(int wavelength_bin, double angle, int height_bin)
 */

/* Function: Actinic
 *
 * Purpose: Find the actinic flux given a wavelength, zenith angle, and
 *          height.
 */

double Actinic(int wavelength_bin, double angle, int height_bin)
{
  double slope, Flux, binsize;
  int /*wavelength_bin,*/ angle_bin;

       /* ActBin[int] is an array containing the center wavelength (nm)
        * for each wavelength bin. Generally,the wavelength bin size is
        * taken to be (ActBin[i+1] - ActBin[i-1])/2 .
        */
  /* convert the angle from radians to degrees... */
  angle = 180.0 * angle / PI;
  if (ActBin[wavelength_bin] < LOW_WAVELENGTH)
    return 0.0;
  else if (ActBin[wavelength_bin] > HIGH_WAVELENGTH)
    return 0.0;
  angle_bin = (int) (angle/10.0);
  if (wavelength_bin == 0)
    binsize = ActBin[1] - ActBin[0];
  else if (wavelength_bin == num_wavelength_bins-1)
    binsize = ActBin[wavelength_bin] - ActBin[wavelength_bin-1];
```

```
  else
    binsize = (ActBin[wavelength_bin+1] - ActBin[wavelength_bin]);
  if (angle < 86.0)
  {
    slope = (ActFlux[wavelength_bin][angle_bin][height_bin] -
             ActFlux[wavelength_bin][angle_bin+1][height_bin])
               /
           (ActAngle[angle_bin] - ActAngle[angle_bin+1]);
    Flux = binsize*(slope*(angle - ActAngle[angle_bin]) +
          ActFlux[wavelength_bin][angle_bin][height_bin]);
    return (Flux);
  }
  return 0.0;
}

/* 2. == File: includes all -- getJ* -- functions
 *                          + get_zenith(timestep)          */
/* Contains: functions which determine the photodissociation constant
 * for various gases. These functions are dependent upon the functions
 * in the files QY.c and ACS.c and the function Actinic.
 *
 * Impelmentation: All of the functions are the same. They calculate the
 * zenith angle for use in the Actinic() function then they integrate
 * over all wavelengths (in 10 nm increments) the quntum yield times (QY)
 * the absoption cross section (ACS) times the actinic flux.
 *
 * inclination: the angle between the sun and the equator.
 * hour: zero at solar noon, increases 15 degrees per hour.
 */

#define MAX_ZENITH (PI*86.0/180.0)

/* -- J1 -        JO2[s][t] = getJO2(T[s],t,s);              */
double getJO2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyO2(double,double),
                getacsO2(double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;
```

```
  /* ACS:       205 - 240 nm
   * QY:          0 - 240 nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 205 - 240 nm
   */
  for (i=0;ActBin[i]<205.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=240.0;i++)
    J += (qyO2(ActBin[i],temperature)*
          getacsO2(ActBin[i])*
          Actinic(i,zenith,k));
  return (J);
}

/* -- J2 -        JO3_J2[s][t] = getJO3_J2(T[s-1],t,s-1);          */
double getJO3_J2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyO3_J2(double,double),
                getacsO3(double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:       170 - 725 nm
   * QY:          0 - 725 nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 201 - 725 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<725.0;i++)
    J += (qyO3_J2(ActBin[i],temperature)*
          getacsO3(ActBin[i])*
          Actinic(i,zenith,k));
  return (J);
}
```

```
/* -- J3 -        JO3_J3[s][t] = getJO3_J3(T[s-1],t,s-1);          */

double getJO3_J3(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyO3_J3(double,double),
                getacsO3(double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;
  /* ACS:      170 - 725 nm
   * QY:         0 - 320 nm
   * Actinic:  201 - 852.5 nm
   * Thus we integrate over lambda 201 - 320 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<320.0;i++)
    J += (qyO3_J3(ActBin[i],temperature)*
          getacsO3(ActBin[i])*
          Actinic(i,zenith,k));
  return (J);
}

/* -- J4 -        JNO2[s][t] = getJNO2(T[s-1],t,s-1);              */
double getJNO2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyNO2(double,double),
                getacsNO2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;
```

```
  /* ACS:      200 - 430 nm
   * QY:         0 - 430 nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 201 - 430 nm
   */

  for (i=0;ActBin[i]<201.0;i++)
     ;
  if (i>0)
     i--;
  for (;ActBin[i]<430.0;i++)

     J += (qyNO2(ActBin[i],temperature)*
          getacsNO2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J5 -        JN2O[s][t] = getJN2O(T[s-1],t,s-1);              */

double getJN2O(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyN2O(double,double),
                getacsN2O(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
     return 0.0;

  /* ACS:      173 - 240 nm (NASA/JPL 92,p.116)
   * QY:         0 - 240    ( 1.0 )
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 201 - 240 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
     ;
  if (i>0)
     i--;
  for (;ActBin[i]<240.0;i++)
     J += (qyN2O(ActBin[i],temperature)*
          getacsN2O(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* -- J6 -        JHNO3[s][t] = getJHNO3(T[s-1],t,s-1);          */
double getJHNO3(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyHNO3(double,double),
                getacsHNO3(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;
  /* ACS:       190 - 340 nm
   * QY:          0 - INF nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 190 - 340 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<340.0;i++)
    J += (qyHNO3(ActBin[i],temperature)*
          getacsHNO3(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J7 -        JH2O2[s][t] = getJH2O2(T[s-1],t,s-1);          */
double getJH2O2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyH2O2(double,double),
                getacsH2O2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
```

```
    return 0.0;

  /* ACS:      260 - 350 nm
   * QY:         0 - INF nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 260 - 350 nm
   */
  for (i=0;ActBin[i]<260.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<350.0;i++)
    J += (qyH2O2(ActBin[i],temperature)*
          getacsH2O2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J8 -        JNO3_J8[s][t] = getJNO3_J8(T[s-1],t,s-1);                */
double getJNO3_J8(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyNO3_J8(double,double),
                getacsNO3(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      400 - 720 nm
   * QY:         0 - ??? nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 400 - 720 nm
   */

  for (i=0;ActBin[i]<400.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<720.0;i++)
    J += (qyNO3_J8(ActBin[i],temperature)*
          getacsNO3(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
    return J;
}

/* -- J9 -        JH2CO_J9[s][t] = getJH2CO_J9(T[s-1],t,s-1);   */
double getJH2CO_J9(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyH2CO_J9(double,double),
                getacsH2CO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      240 - 310 nm
   * QY:         0 - 340 nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 240 - 340 nm
   */
  for (i=0;ActBin[i]<240.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<310.0;i++)
    J += (qyH2CO_J9(ActBin[i],temperature)*
          getacsH2CO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J10 - JH2CO_J10[s][t] = getJH2CO_J10(T[s-1],t,P[s-1],s-1);   */
double getJH2CO_J10(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyH2CO_J10(double,double),
                getacsH2CO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
```

```
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      240 - 310 nm <== Unsure of this
   * QY:         0 - INF nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 240 - 730 nm
   */
  for (i=0;ActBin[i]<240.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<310.0;i++)
    J += (qyH2CO_J10(ActBin[i],temperature)*
          getacsH2CO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J11 -      JCF2Cl2[s][t] = getJCF2Cl2(T[s-1],t,s-1);      */
double getJCF2Cl2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyCF2Cl2(double,double),
                getacsCF2Cl2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      170 - 240 nm  (NASA/JPL 92,p. 138)
   * QY:         0 - 240 nm  ( 1.0 )
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 201 - 240 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<240.0;i++)
    J += (qyCF2Cl2(ActBin[i],temperature)*
          getacsCF2Cl2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* -- J12 -       JCFCl3[s][t] = getJCFCl3(T[s-1],t,s-1);          */
double getJCFCl3(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyCFCl3(double,double),
                getacsCFCl3(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      170 - 260 nm  (NASA/JPL 92,p. 138)
   * QY:         0 - 260 nm  (  1.0  )
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 201 - 260 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<260.0;i++)
    J += (qyCFCl3(ActBin[i],temperature)*
          getacsCFCl3(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J13 -       JClONO2[s][t] = getJClONO2(T[s-1],t,s-1);     */
double getJClONO2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyClONO2(double,double),
                getacsClONO2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
```

```
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      250 - 400 nm  (NASA/JPL 92,p. 135)
   * QY:         0 - 400 nm  ( 0.9 )
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 250 - 400 nm
   */
  for (i=0;ActBin[i]<250.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<400.0;i++)
    J += (qyClONO2(ActBin[i],temperature)*
          getacsClONO2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J14 -      JHCl[s][t] = getJHCl(T[s-1],t,s-1);                */
double getJHCl(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyHCl(double,double),
                getacsHCl(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      140 - 220 nm  (NASA/JPL 92, p. 130)
   * QY:         0 - 220 nm  ( assume 1.0 )
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 201 - 220 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<220.0;i++)
    J += (qyHCl(ActBin[i],temperature)*
          getacsHCl(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* -- J15 -      JHOCl[s][t] = getJHOCl(T[s-1],t,s-1);          */
double getJHOCl(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyHOCl(double,double),
                getacsHOCl(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      215 - 375 nm  (NASA/JPL 92,p. 131)
   * QY:         0 - 375 nm  ( 1.0 )
   * Actinic:  201 - 852.5 nm
   * Thus we integrate over lambda 215 - 375 nm
   */
  for (i=0;ActBin[i]<215.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<375.0;i++)
    J += (qyHOCl(ActBin[i],temperature)*
          getacsHOCl(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J16 -      JN2O5[s][t] = getJN2O5(T[s-1],t,s-1);          */
double getJN2O5(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyN2O5(double,double),
                getacsN2O5(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
```

```
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      200 - 852 nm
   * QY:         0 - INF nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 200 - 280 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<852.5;i++)
    J += (qyN2O5(ActBin[i],temperature)*
          getacsN2O5(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J17 -      JCH3OOH[s][t] = getJCH3OOH(T[s-1],t,s-1);  */
double getJCH3OOH(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyCH3OOH(double,double),
                getacsCH3OOH(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      210 - INF nm
   * QY:         0 - INF nm
   * Actinic:   201 - 852.5 nm
   * Thus we integrate over lambda 210 - 730 nm
   */
  for (i=0;ActBin[i]<210.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<730.0;i++)
    J += (qyCH3OOH(ActBin[i],temperature)*
          getacsCH3OOH(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}
/* -- J18 -       JCCl4[s][t] = getJCCl4(T[s-1],t,s-1);          */
double getJCCl4(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyCCl4(double,double),
                getacsCCl4(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      174 - 275 nm  (NASA/JPL 92,p. 138)
   * QY:         0 - 275 nm  ( 1.0 )
   * Actinic:  201 - 852.5 nm
   * Thus we integrate over lambda 201 - 275 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<240.0;i++)
    J += (qyCCl4(ActBin[i],temperature)*
          getacsCCl4(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J19 -       JHNO4[s][t] = getJHNO4(T[s-1],t,s-1);          */
double getJHNO4(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyHNO4(double,double),
                getacsHNO4(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
```

```
    return 0.0;

  /* ACS:      190 - 330 nm
   * QY:        0 - INF nm
   * Actinic:  201 - 852.5 nm
   * Thus we integrate over lambda 190 - 330 nm
   */
  for (i=0;ActBin[i]<201.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<330.0;i++)
    J += (qyHNO4(ActBin[i],temperature)*
          getacsHNO4(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J20 -      JClO[s][t] = getJClO(T[s-1],t,s-1);                    */
double getJClO(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyClO(double,double),
                getacsClO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      225.0 - 309.9 nm  (NASA/JPL 92,p. 108)
   * QY:        0 - ?? nm  ( 1.0 )
   * Actinic:  201 - 852.5 nm
   * Thus we integrate over lambda 225 - 310.0 nm
   */

  for (i=0;ActBin[i]<225.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<310.0;i++)
    J += (qyClO(ActBin[i],temperature)*
          getacsClO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* -- J22 -      JNO3_J22[s][t] = getJNO3_J22(T[s-1],t,s-1);          */
double getJNO3_J22(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyNO3_J22(double,double),
                getacsNO3(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      400 - 720 nm
   * QY:         0 -  ??nm
   * Actinic:  201 - 852.5 nm
   * Thus we integrate over lambda 400 - 720 nm
   */
  for (i=0;ActBin[i]<400.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<720.0;i++)
    J += (qyNO3_J22(ActBin[i],temperature)*
          getacsNO3(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J24 -      JH2O[s][t] = getJH2O(T[s-1],t,s-1);          */
double getJH2O(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyH2O(double,double),
                getacsH2O(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
```

```
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:       175.5 - 189.3 nm  (NASA/JPL 92,p. 108)
   * QY:          0 - 189.3 nm  (  1.0  )
   * Actinic:   175.44 - 852.5 nm
   * Thus we integrate over lambda 175.5 - 189.4 nm
   */
  for (i=0;ActBin[i]<175.5;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<189.4;i++)
    J += (qyH2O(ActBin[i],temperature)*
          getacsH2O(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J21 -       JCl2[s][t] = getJCl2(T[s],t,s);  */
double getJCl2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyCl2(double,double),
                getacsCl2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:       240.0 - 450.0 nm  (NASA/JPL 92,p. )
   * QY:          0 - /// nm  (  1.0  )
   * Actinic:   175.44 - 852.5 nm
   * Thus we integrate over lambda 240- 450 nm
   */
  for (i=0;ActBin[i]<240.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=450.0;i++)
    J += (qyCl2(ActBin[i],temperature)*
          getacsCl2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* -- J23 -       JClOO[s][t] = getJClOO(T[s],t,s); */
double getJClOO(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyClOO(double,double),
                getacsClOO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      220.0 - 280.0 nm  (NASA/JPL 92,p. )
   * QY:         0 - /// nm  (  1.0  )
   * Actinic:   175.44 - 852.5 nm
   * Thus we integrate over lambda 220- 280 nm
   */
  for (i=0;ActBin[i]<220.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=280.0;i++)
    J += (qyClOO(ActBin[i],temperature)*
          getacsClOO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J25 -       JOClO[s][t] = getJOClO(T[s],t,s); */
double getJOClO(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyOClO(double,double),
                getacsOClO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
```

```
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      272.93-475.53 nm  (NASA/JPL 92,p. )
   * QY:         0 - /// nm  (  1.0  )
   * Actinic:   175.44 - 852.5 nm
   * Thus we integrate over lambda 272.93-475.53nm
   */
  for (i=0;ActBin[i]<272.93;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=475.53;i++)
    J += (qyOClO(ActBin[i],temperature)*
          getacsOClO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J27 -       JCl2O2[s][t] = getJCl2O2(T[s],t,s); */
double getJCl2O2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyCl2O2(double,double),
                getacsCl2O2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      200 - 360  nm  (NASA/JPL 92,p. )
   * QY:         0 - /// nm  (  1.0  )
   * Actinic:   175.44 - 852.5 nm
   * Thus we integrate over lambda 200 - 360 nm
   */
  for (i=0;ActBin[i]<200.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=360.0;i++)
    J += (qyCl2O2(ActBin[i],temperature)*
          getacsCl2O2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* -- J28 -       JClNO[s][t] = getJClNO(T[s],t,s);  */
double getJClNO(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyClNO(double,double),
                getacsClNO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:       190.0 - 350.0 nm  (NASA/JPL 92,p. )
   * QY:          0 - /// nm  (  1.0  )
   * Actinic:   175.44 - 852.5 nm
   * Thus we integrate over lambda 190- 350 nm
   */
  for (i=0;ActBin[i]<190.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=350.0;i++)
    J += (qyClNO(ActBin[i],temperature)*
          getacsClNO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J29 -       JClNO2[s][t] = getJClNO2(T[s],t,s);  */
double getJClNO2(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyClNO2(double,double),
                getacsClNO2(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
```

```
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      190 - 370  nm  (NASA/JPL 92,p. )
   * QY:        0 - /// nm  (  1.0  )
   * Actinic:  175.44 - 852.5 nm
   * Thus we integrate over lambda 190 - 370 nm
   */
  for (i=0;ActBin[i]<190.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=370.0;i++)
    J += (qyClNO2(ActBin[i],temperature)*
          getacsClNO2(ActBin[i],temperature)*
          Actinic(i,zenith,k));
  return J;
}

/* -- J30 -      JClONO[s][t] = getJClONO(T[s],t,s);  */
double getJClONO(double temperature, int timestep, int k)
{
  double J = 0.0, lambda, zenith;
  int i;
  extern int num_wavelength_bins;
  extern double latitude, inclination;
  extern double get_zenith(int),
                qyClONO(double,double),
                getacsClONO(double,double),
                Actinic(int,double,int);

  zenith = get_zenith(timestep);
  if (zenith > MAX_ZENITH)
    return 0.0;

  /* ACS:      235 - 400  nm  (NASA/JPL 92,p. )
   * QY:        0 - /// nm  (  1.0  )
   * Actinic:  175.44 - 852.5 nm
   * Thus we integrate over lambda 235 - 400 nm
   */
  for (i=0;ActBin[i]<235.0;i++)
    ;
  if (i>0)
    i--;
  for (;ActBin[i]<=400.0;i++)
    J += (qyClONO(ActBin[i],temperature)*
          getacsClONO(ActBin[i],temperature)*
          Actinic(i,zenith,k));
```

```
  return J;
}

/* 3. == Function: get_J  */
void get_J(double *T, double *M, double *P)
{
  int s,t,i,j;
  extern double
        getJO2(double,int,int),
        getJO3_J2(double,int,int),
        getJO3_J3(double,int,int),
        getJNO2(double,int,int),
        getJN2O(double,int,int),
        getJHNO3(double,int,int),
        getJH2O2(double,int,int),
        getJNO3_J8(double,int,int),
        getJH2CO_J9(double,int,int),
        getJH2CO_J10(double,int,int),
        getJCF2Cl2(double,int,int),
        getJCFCl3(double,int,int),
        getJClONO2(double,int,int),
        getJHCl(double,int,int),
        getJHOCl(double,int,int),
        getJN2O5(double,int,int),
        getJCH3OOH(double,int,int),
        getJCCl4(double,int,int),
        getJHNO4(double,int,int),
        getJNO3_J22(double,int,int),
        getJH2O(double,int,int);

  for (s=0;s< SPACE_POINTS;s++)
    {
        for (t=0;t< TIME_POINTS;t++)
          {
/* -- J1 - */    JO2[s][t] = getJO2(T[s],t,s);
/* -- J2 - */    JO3_J2[s][t] = getJO3_J2(T[s],t,s);
/* -- J3 - */    JO3_J3[s][t] = getJO3_J3(T[s],t,s);
/* -- J4 - */    JNO2[s][t] = getJNO2(T[s],t,s);
/* -- J5 - */    JN2O[s][t] = getJN2O(T[s],t,s);
/* -- J6 - */    JHNO3[s][t] = getJHNO3(T[s],t,s);
/* -- J7 - */    JH2O2[s][t] = getJH2O2(T[s],t,s);
/* -- J8 - */    JNO3_J8[s][t] = getJNO3_J8(T[s],t,s);
/* -- J9 - */    JH2CO_J9[s][t] = getJH2CO_J9(T[s],t,s);
/* -- J10 - */   JH2CO_J10[s][t] = getJH2CO_J10(T[s],t,s);
/* -- J11 - */   JCF2Cl2[s][t] = getJCF2Cl2(T[s],t,s);
/* -- J12 - */   JCFCl3[s][t] = getJCFCl3(T[s],t,s);
/* -- J13 - */   JClONO2[s][t] = getJClONO2(T[s],t,s);
/* -- J14 - */   JHCl[s][t] = getJHCl(T[s],t,s);
```

```
/* -- J15 - */   JHOCl[s][t] = getJHOCl(T[s],t,s);
/* -- J16 - */   JN2O5[s][t] = getJN2O5(T[s],t,s);
/* -- J17 - */   JCH3OOH[s][t] = getJCH3OOH(T[s],t,s);
/* -- J18 - */   JCCl4[s][t] = getJCCl4(T[s],t,s);
/* -- J19 - */   JHNO4[s][t] = getJHNO4(T[s],t,s);
/* -- J20 - */   JClO[s][t] = getJClO(T[s],t,s);
/* -- J22 - */   JNO3_J22[s][t] = getJNO3_J22(T[s],t,s);
/* -- J24 - */   JH2O[s][t] = getJH2O(T[s],t,s);
/* -- J21 - */   JCl2[s][t] = getJCl2(T[s],t,s);
/* -- J23 - */   JClOO[s][t] = getJClOO(T[s],t,s);
/* -- J25 - */   JOClO[s][t] = getJOClO(T[s],t,s);
/* -- J27 - */   JCl2O2[s][t] = getJCl2O2(T[s],t,s);
/* -- J28 - */   JClNO[s][t] = getJClNO(T[s],t,s);
/* -- J29 - */   JClNO2[s][t] = getJClNO2(T[s],t,s);
/* -- J30 - */   JClONO[s][t] = getJClONO(T[s],t,s);
        }   /* --- t loop --- */
    }       /* --- s loop --- */
}       /* --- main loop --- */
```

7. /* === File: O3k.c

```
 * Purpose: Determine the K values for all of the reactions in the model
 *          for each of the atmospheric layers.
 */

#include <math.h>
#include <stdio.h>
#include "O3model.h"
#include "O3util.h"

void get_K(double T[], double M[], double P[])
{
  int s;

  for (s=0;s<SPACE_POINTS;s++)
    {
    /* binary reactions */
                                                        /* k1 */
      KO_O3[s] = getKO_O3(T[s]);                        /* k2 */
                                                        /* k3 */
      KO1D_N2[s] = getKO1D_N2(T[s]);                    /* k4 */
      KO1D_O2[s] = getKO1D_O2(T[s]);                    /* k5 */
      KO1D_O3_1[s] = getKO1D_O3_1;                            /* k6  */
      KO1D_O3_2[s] = getKO1D_O3_2;                            /* k7  */
      KO1D_N2O_1[s] = getKO1D_N2O_1;                          /* k8  */
      KO1D_N2O_2[s] = getKO1D_N2O_2;                          /* k9  */
      KNO_O3[s] = getKNO_O3(T[s]);                            /* k10 */
      KNO2_O[s] = getKNO2_O;                                  /* k11 */
/*    KN_O3[s] = getKN_O3;        */                        /* k12 */
```

```
/*      KN_O2[s] = getKN_O2(T[s]); */                    /* k13 */
        KNO2_O3[s] = getKNO2_O3(T[s]);                   /* k14 */
        KNO_NO3[s] = getKNO_NO3;                                 /* k15 */
                                                         /* k16 */
                                                         /* k17 */
                                                         /* k18 */
/*      KN_NO[s] = getKN_NO;                */                   /* k19 */
/*      KN_NO2[s] = getKN_NO2(T[s]);        */                   /* k20 */
        KO1D_H2O[s] = getKO1D_H2O;                               /* k21 */
        KO1D_CH4_1[s] = getKO1D_CH4_1;                           /* k22 */
        KO1D_CH4_2[s] = getKO1D_CH4_2;                           /* k23 */
        KO1D_H2[s] = getKO1D_H2;                         /* k24 */
        KH_O3[s] = getKH_O3(T[s]);                       /* k25 */
                                                         /* k26 */
        KOH_O3[s] = getKOH_O3(T[s]);                             /* k27 */
        KHO2_O3[s] = getKHO2_O3(T[s]);                   /* k28 */
        KOH_O[s] = getKOH_O(T[s]);                               /* k29 */
        KHO2_O[s] = getKHO2_O;                                   /* k30 */
        KH2O2_O[s] = getKH2O2_O(T[s]);                   /* k31 */
        KOH_CH4[s] = getKOH_CH4(T[s]);                   /* k32 */
        KHO2_NO[s] = getKHO2_NO(T[s]);                   /* k33 */
        KOH_CO[s] =  getKOH_CO(P[s]);           /* k34 *
        KOH_H2[s] = getKOH_H2(T[s]);                             /* k35 */
                                                         /* k36 */
        KOH_HNO3[s] = getKOH_HNO3(T[s]);                         /* k37 */
        KOH_H2O2[s] = getKOH_H2O2(T[s]);                         /* k38 */
        KOH_HO2[s] = getKOH_HO2(T[s]);                   /* k39 */
        KOH_OH[s] = getKOH_OH(T[s]);                             /* k40 */
        KOH_H2CO[s] = getKOH_H2CO;                               /* k41 */
        KHO2_HO2[s] = getKHO2_HO2(T[s]);                         /* k42 */
                                                         /* k43 */
        KH_HO2_1[s] = getKH_HO2_1;                               /* k44 */
        KH_HO2_2[s] = getKH_HO2_2;                               /* k45 */
        KH_HO2_3[s] = getKH_HO2_3;                               /* k46 */
        KH2CO_O[s] = getKH2CO_O(T[s]);                   /* k47 */
                                                         /* k48 */
        KCH3O2_NO[s] = getKCH3O2_NO(T[s]);                       /* k49 */
        KCH3O2_HO2[s] = getKCH3O2_HO2(T[s]);                     /* k50 */
        KCH3OOH_OH[s] = getKCH3OOH_OH(T[s]);                             /* k51 */
        KCH3O_O2[s] = getKCH3O_O2(T[s]);                         /* k52 */
        KHCO_O2[s] = getKHCO_O2(T[s]);                   /* k53 */
        KO1D_CF2Cl2[s] = getKO1D_CF2Cl2;                         /* k54 */
        KO1D_CFCl3[s] = getKO1D_CFCl3;                           /* k55 */
        KO1D_CH3Cl[s] = getKO1D_CH3Cl;                           /* k56 */
        KOH_CH3Cl[s] = getKOH_CH3Cl(T[s]);                       /* k57 */
        KCl_O3[s] = getKCl_O3(T[s]);                             /* k58 */
        KClO_O[s] = getKClO_O(T[s]);                             /* k59 */
        KClO_NO[s] = getKClO_NO(T[s]);                   /* k60 */
```

```
KO1D_HCl[s] = getKO1D_HCl;                              /* k61 */
KCl_CH4[s] = getKCl_CH4(T[s]);                  /* k62 */
KCl_HO2[s] = getKCl_HO2(T[s]);                  /* k63 */
KCl_H2[s] = getKCl_H2(T[s]);                            /* k64 */
KHCl_H[s] = getKHCl_H(T[s]);                            /* k65 */
KCl_H2O2[s] = getKCl_H2O2(T[s]);                /* k66 */
KOH_HCl[s] = getKOH_HCl(T[s]);                  /* k67 */
                                                        /* k68 */
                                                        /* k69 */
KHO2_ClO[s] = getKHO2_ClO(T[s]);                /* k70 */
KH_ClO[s] = getKH_ClO;                                  /* k71 */
                                                        /* k72 */
KClONO2_O[s] = getKClONO2_O(T[s]);                      /* k73 */
KClONO2_OH[s] = getKClONO2_OH(T[s]);                    /* k74 */
KH2CO_Cl[s] = getKH2CO_Cl(T[s]);                /* k75 */
KH2_O[s] = getKH2_O(T[s]);                      /* k76 */
KO1D_CCl4[s] = getKO1D_CCl4;                            /* k77 */
KOH_HOCl[s] = getKOH_HOCl(T[s]);                        /* k78 */
                                                        /* k79 */
                                                        /* k80 */
KHNO4_OH[s] = getKHNO4_OH(T[s]);                        /* k81 */

                                                        /* k82 */
KO_NO3[s] = getKO_NO3;                                  /* k83 */
KO_HNO4[s] = getKO_HNO4(T[s]);                  /* k84 */
KH_NO2[s] = getKH_NO2(T[s]);                            /* k85 */
KO_CH3[s] = getKO_CH3;                                  /* k86 */
KO3_CH3[s] = getKO3_CH3(T[s]);                  /* k87 */
KCH3O2_CH3O2_1[s] = getKCH3O2_CH3O2_1(T[s]);                    /* k88 */
KCH3O2_CH3O2_2[s] = getKCH3O2_CH3O2_2(T[s]);                    /* k89 */
KCH3O2_CH3O2_3[s] = getKCH3O2_CH3O2_3(T[s]);                    /* k90 */
KCl_CH3Cl[s] = getKCl_CH3Cl(T[s]);                      /* k91 */
KCl_HOCl[s] = getKCl_HOCl(T[s]);                /* k92 */
KCl_NO3[s] = getKCl_NO3;                        /* k93 */
KClO_OH[s] = getKClO_OH(T[s]);                  /* k94 */
KClO_CH4[s] = getKClO_CH4(T[s]);                /* k95 */
KClO_H2[s] = getKClO_H2(T[s]);                  /* k96 */
                                                        /* k97 */
KOH_CFCl3[s] = getKOH_CFCl3(T[s]);                      /* k98 */
KOH_CF2Cl2[s] = getKOH_CF2Cl2(T[s]);                    /* k99 */
KO_HCl[s] = getKO_HCl(T[s]);                            /* k100 */
KO_HOCl[s] = getKO_HOCl(T[s]);                  /* k101 */
KCl_HO2_2[s] = getKCl_HO2_2(T[s]);              /* k103 */
KCl_OClO[s] = getKCl_OClO(T[s]);        /* k105 */
KCl_ClOO_1[s] = getKCl_ClOO_1;          /* k106 */
KCl_ClOO_2[s] = getKCl_ClOO_2;          /* k107 */
KCl_Cl2O2[s] = getKCl_Cl2O2;            /* k108 */
KCl_ClONO2[s] = getKCl_ClONO2(T[s]);        /* k109 */
```

```
        KCl_ClNO[s] = getKCl_ClNO(T[s]);          /* k110 */
        KClO_NO3[s] = getKClO_NO3;                /* k111 */
        KClO_H2CO[s] = getKClO_H2CO(T[s]);        /* k112 */
        KClO_CO[s] = getKClO_CO(T[s]);            /* k113 */
        KClO_N2O[s] = getKClO_N2O(T[s]);          /* k114 */
        KClO_ClO[s] = getKClO_ClO(T[s]);          /* k115 */
        KClO_O3[s] = getKClO_O3(T[s]);            /* k116 */
        KOClO_NO[s] = getKOH_Cl2(T[s]);           /* k117 */
        KOClO_NO[s] = getKOH_ClNO2;               /* k118 */
        KOClO_NO[s] = getKOClO_O(T[s]);           /* k120 */
        KOClO_NO[s] = getKOClO_O3(T[s]);          /* k121 */
        KOClO_NO[s] = getKOClO_OH(T[s]);          /* k122 */
        KOClO_NO[s] = getKOClO_NO(T[s]);          /* k123 */

      /* termolecular reactions */
        KO_O2_M[s] = getKO_O2_M(T[s],M[s]);                    /* k1 */
        KNO_O_M[s] = getKNO_O_M(T[s],M[s]);                    /* k16 */
        KNO2_NO3_M[s] = getKNO2_NO3_M(T[s],M[s]);              /* 17 */
        KH_O2_M[s] = getKH_O2_M(T[s],M[s]);                    /* 26 */
        KOH_NO2_M[s] = getKOH_NO2_M(T[s],M[s]);                /* 36 */
        KOH_OH_M[s] = getKOH_OH_M(T[s],M[s]);                  /* 43 */
        KCH3_O2_M[s] = getKCH3_O2_M(T[s],M[s]);                /* 48 */
        KCl_O2_M[s] = getKCl_O2_M(T[s],M[s]);                  /* k68 */
        KClO_NO2_M[s] = getKClO_NO2_M(T[s],M[s]);              /* 72 */
        KHO2_NO2_M[s] = getKHO2_NO2_M(T[s],M[s]);              /* 79 */
        KO_NO2_M[s] = getKO_NO2_M(T[s],M[s]);                  /* 82 */
        KHO2_HO2_M[s] = getKHO2_HO2_M(T[s],M[s]);              /* 97 */
        KCl_NO_M[s] = getKCl_NO_M(T[s],M[s]);                  /* k124 */
        KCl_NO2_M1[s] = getKCl_NO2_M1(T[s],M[s]);              /* k125 */
        KCl_NO2_M2[s] = getKCl_NO2_M2(T[s],M[s]);              /* k126 */
        KCl_CO_M[s] = getKCl_CO_M(T[s],M[s]);                  /* k127 */
        KClO_ClO_M[s] = getKClO_ClO_M(T[s],M[s]);              /* k128 */
        }
    for (s=0;s<SPACE_POINTS;s++)
    KOH_H2[s] = 5.5E-12 * exp(-2000.0/(T[s]));
}
```

8. /* === header file: O3model.h */

```
/* include all the definitions */
#include <math.h>
#define MAX 300
#define LOW_WAVELENGTH 175.44
#define HIGH_WAVELENGTH 852.5
#define SPACE_POINTS 27
#define TIME_POINTS 24
#define WAVELENGTH_POINTS 234
#define ANGLE_POINTS 10
#define PI      3.1415926536
```

```
#define totalM 2.4141e25
#define AMdiff 1.66
#define sur_albedo 0.3

/* absorption cross section data files */
#define acsO2_FILE "dat/acsO2.dat"
#define acsO3_FILE "dat/acsO3.dat"
#define acsNO2_FILE "dat/acsNO2.dat"
#define acsN2O_FILE "dat/acsN2O.dat"
#define acsHNO3_FILE "dat/acsHNO3.dat"
#define acsNO3_FILE "dat/acsNO3.dat"
#define acsH2CO_FILE "dat/acsH2CO.dat"
#define acsCF2Cl2_FILE "dat/acsCF2Cl2.dat"
#define acsCFCl3_FILE "dat/acsCFCl3.dat"
#define acsClONO2_FILE "dat/acsClONO2.dat"
#define acsHCl_FILE "dat/acsHCl.dat"
#define acsHOCl_FILE "dat/acsHOCl.dat"
#define acsN2O5_FILE "dat/acsN2O5.dat"
#define acsCH3OOH_FILE "dat/acsCH3OOH.dat"
#define acsCCl4_FILE "dat/acsCCl4.dat"
#define acsHNO4_FILE "dat/acsHNO4.dat"
#define acsH2O_FILE "dat/acsH2O.dat"
#define acsClO_FILE "dat/acsClO.dat"
#define acsCl2_FILE "dat/acsCl2.dat"
#define acsClOO_FILE "dat/acsClOO.dat"
#define acsOClO_FILE "dat/acsOClO.dat"
#define acsCl2O2_FILE "dat/acsCl2O2.dat"
#define acsClNO_FILE "dat/acsClNO.dat"
#define acsClNO2_FILE "dat/acsClNO2.dat"
#define acsClONO_FILE "dat/acsClONO.dat"
#define H2O_FILE "dat/H2O.dat"
#define CH4_FILE "dat/CH4.dat"
#define CO_FILE "dat/CO.dat"
#define N2_FILE "dat/N2.dat"
#define O2_FILE "dat/O2.dat"
#define H2_FILE "dat/H2.dat"
#define N2O_FILE "dat/N2O.dat"
#define CFCl3_FILE "dat/CFCl3.dat"
#define CF2Cl2_FILE "dat/CF2Cl2.dat"
#define CCl4_FILE "dat/CCl4.dat"
#define CH3Cl_FILE "dat/CH3Cl.dat"
#define STDATM_FILE "dat/O3_stdatm.dat"
#define NOx_FILE "dat/NOx.dat"
#define SOLAR_FLUX_FILE "dat/O3_solarflux.dat"
#define Kz_FILE "dat/O3kz.dat"
#define O3_COLUMN_FILE "dat/O3column.dat"
#define O2_COLUMN_FILE "dat/O2column.dat"
#define TOP_RAD_FILE "dat/O3_topflux.dat"
```

```
/* index values for the solve and output arrays */
#define IO      0
#define IO1D    1
#define IO3     2
#define IH      3
#define IOH     4
#define IHO2    5
#define IH2O2   6
#define IN      7
#define INO     8
#define INO2    9
#define INO3    10
#define IN2O5   11
#define IHNO3           12
#define IHNO4           13
#define ICl     14
#define IClO    15
#define IHOCl   16
#define IClONO2         17
#define ICH3    18
#define ICH3O           19
#define ICH3O2          20
#define ICH3OOH         21
#define IH2CO           22
#define IHCl    23
#define ICl2    24
#define IClOO   25
#define IOClO   26
#define ICl2O2          27
#define IClNO   28
#define IClNO2          29
#define IClONO          30

/* 1. === K - bimolecular reactions ============================
 *
 * purpose: This file contains all of the binary recations that are
 * used in the model. The first letter of the definition indicated that
 * it is a K value (thermal reaction). The next set of letters and
 * numbers are the first reactent and the second (separated by an
 * underscore) is the second.
 *
 * The functions are dependent upon temperature (T), pressure (P), and
 * number density (M).
 */
#include <math.h>
#                                                       /* k1 */
#define getKO_O3(T)             (8.0E-12*exp(-2060.0/(T)))      /* k2 */
#                                                       /* k3 */
```

```
#define getKO1D_N2(T)          (1.8E-11 * exp(110.0/(T)))     /* k4 */
#define getKO1D_O2(T)          (3.2E-11 * exp(70.0/(T)))      /* k5 */
#define getKO1D_O3_1           (1.2E-10)                      /* k6  */
#define getKO1D_O3_2           (1.2E-10)                      /* k7  */
#define getKO1D_N2O_1          (4.9E-11)                      /* k8  */
#define getKO1D_N2O_2          (6.7E-11)                      /* k9  */
#define getKNO_O3(T)           (2.3E-12 * exp(-1450.0/(T)))   /* k10 */
#define getKNO2_O      (9.3E-12)                      /* k11 */
/* #define getKN_O3    (5.0E-16)                       k12 */
/* #define getKN_O2(T)          (4.4E-12 * exp(-3220.0/(T)))    k13 */
#define getKNO2_O3(T)          (1.2E-13 * exp(-2450.0/(T)))   /* k14 */
#define getKNO_NO3     (2.0E-11)                      /* k15 */
#                                                     /* k16 */
#                                                     /* k17 */
#                                                     /* k18 */
/* #define getKN_NO    (3.4E-11)                       k19 */
/* #define getKN_NO2(T)         (2.1E-11 * exp(-800.0/(T)))     k20 */
#define getKO1D_H2O            (2.2E-10)                      /* k21 */
#define getKO1D_CH4_1          (1.4E-10)                      /* k22 */
#define getKO1D_CH4_2          (1.4E-11)                      /* k23 */
#define getKO1D_H2     (1.0E-10)                      /* k24 */
#define getKH_O3(T)    (1.4E-10 * exp(-470.0/(T)))    /* k25 */
#                                                     /* k26 */
#define getKOH_O3(T)           (1.6E-12 * exp(-940.0/(T)))    /* k27 */
#define getKHO2_O3(T)          (1.1E-14 * exp(-500.0/(T)))    /* k28 */
#define getKOH_O(T)    (2.2E-11 * exp(110.0/(T)))     /* k29 */
#define getKHO2_O      (3.0E-11)                      /* k30 */
#define getKH2O2_O(T)          (1.4E-12 * exp(-2000.0/(T)))   /* k31 */
#define getKOH_CH4(T)          (2.9E-12 * exp(-1800.0/(T)))   /* k32 */
#define getKHO2_NO(T)          (3.5E-12 * exp(250.0/(T)))     /* k33 */
#define getKOH_CO(P)           (1.5E-13 * (1.0+0.6*(P)))      /* k34 */
#define getKOH_H2(T)           (5.5E-12 * exp(-2000.0/(T)))   /* k35 */
#                                                     /* k36 */
#define getKOH_HNO3(T)         (1.5E-14 * exp(650.0/(T)))     /* k37 */
#define getKOH_H2O2(T)         (2.9E-12 * exp(-160.0/(T)))    /* k38 */
#define getKOH_HO2(T)          (4.8E-11 * exp(250.0/(T)))     /* k39 */
#define getKOH_OH(T)           (4.2E-12 * exp(-240.0/(T)))    /* k40 */
#define getKOH_H2CO            (1.0E-11)                      /* k41 */
#define getKHO2_HO2(T)         (2.3E-13 * exp(600.0/(T)))     /* k42 */
#                                                     /* k43 */
#define getKH_HO2_1    (7.4E-12)                      /* k44 */
#define getKH_HO2_2    (2.2E-12)                      /* k45 */
#define getKH_HO2_3    (6.4E-11)                      /* k46 */
#define getKH2CO_O(T)          (3.4E-11 * exp(-1600.0/(T)))   /* k47 */
#                                                     /* k48 */
#define getKCH3O2_NO(T)        (4.2E-12 * exp(180.0/(T)))     /* k49 */
#define getKCH3O2_HO2(T)  (3.8E-13 * exp(800.0/(T)))      /* k50 */
#define getKCH3OOH_OH(T)  (3.8E-12 * exp(200.0/(T)))      /* k51 */
```

```
#define getKCH3O_O2(T)          (3.9E-14 * exp(-900.0/(T)))     /* k52 */
#define getKHCO_O2(T)           (3.5E-12 * exp(140.0/(T)))      /* k53 */
#define getKO1D_CF2Cl2          (1.4E-10)                       /* k54 */
#define getKO1D_CFCl3           (2.3E-10)                       /* k55 */
#define getKO1D_CH3Cl           (1.0E-10)                       /* k56 */
#define getKOH_CH3Cl(T)         (1.8E-12 * exp(-1110.0/(T)))    /* k57 */
#define getKCl_O3(T)        (2.9E-11 * exp(-260.0/(T)))     /* k58 */
#define getKClO_O(T)        (3.0E-11 * exp(70.0/(T)))       /* k59 */
#define getKClO_NO(T)           (6.4E-12 * exp(290.0/(T)))      /* k60 */
#define getKO1D_HCl         (1.5E-10)                       /* k61 */
#define getKCl_CH4(T)           (1.1E-11 * exp(-1400.0/(T)))    /* k62 */
#define getKCl_HO2(T)           (1.8E-11 * exp(170.0/(T)))      /* k63 */
#define getKCl_H2(T)        (3.7E-11 * exp(-2300.0/(T)))    /* k64 */
#define getKHCl_H(T)        (4.7E-11 * exp(-2340.0/(T)))    /* k65 */
#define getKCl_H2O2(T)          (1.1E-11 * exp(-980.0/(T)))     /* k66 */
#define getKOH_HCl(T)           (2.6E-12 * exp(-350.0/(T)))     /* k67 */
#                                                       /* k68 */
#                                                       /* k69 */
#define getKHO2_ClO(T)          (4.8E-13 * exp(700.0/(T)))      /* k70 */
#define getKH_ClO           (3.0E-11)                       /* k71 */
#                                                       /* k72 */
#define getKClONO2_O(T)         (2.9E-12 * exp(-800.0/(T)))     /* k73 */
#define getKClONO2_OH(T)    (1.2E-12 * exp(-330.0/(T)))     /* k74 */
#define getKH2CO_Cl(T)          (8.1E-11 * exp(-30.0/(T)))      /* k75 */
#define getKH2_O(T)         (8.8E-12 * exp(-4200.0/(T)))    /* k76 */
#define getKO1D_CCl4            (3.3E-10)                       /* k77 */
#define getKOH_HOCl(T)          (3.0E-12 * exp(-500.0/(T)))     /* k78 */
#                                                       /* k79 */
#                                                       /* k80 */
#define getKHNO4_OH(T)          (1.3E-12 * exp(380.0/(T)))      /* k81 */
#                                                       /* k82 */
#define getKO_NO3           (1.0E-11)                       /* k83 */
#define getKO_HNO4(T)           (7.8E-11 * exp(-3400.0/(T)))    /* k84 */
#define getKH_NO2(T)            (4.0E-10 * exp(-340.0/(T)))     /* k85 */
#define getKO_CH3           (1.1E-10)                       /* k86 */
#define getKO3_CH3(T)           (5.4E-12 * exp(-220.0/(T)))     /* k87 */
#define getKCH3O2_CH3O2(T)      (2.5E-13 * exp(-190.0/(T)))
#define getKCH3O2_CH3O2_1(T)    (0.3*2.5E-13 * exp(-190.0/(T)))         /* k88 */
#define getKCH3O2_CH3O2_2(T)    (0.6*2.5E-13 * exp(-190.0/(T)))         /* k89 */
#define getKCH3O2_CH3O2_3(T)    (0.1*2.5E-13 * exp(-190.0/(T)))         /* k90 */
#define getKCl_CH3Cl(T)         (3.3E-11 * exp(-1250.0/(T)))    /* k91 */
#define getKCl_HOCl(T)          (3.0E-12 * exp(-130.0/(T)))     /* k92 */
#define getKCl_NO3          (2.6E-11)                       /* k93 */
#define getKClO_OH(T)           (1.1E-11 * exp(-120.0/(T)))     /* k94 */
#define getKClO_CH4(T)          (1.0E-12 * exp(-3700.0/(T)))    /* k95 */
#define getKClO_H2(T)           (1.0E-12 * exp(-4800.0/(T)))    /* k96 */
#                                                       /* k97 */
#define getKOH_CFCl3(T)         (1.0E-12 * exp(-3700.0/(T)))    /* k98 */
```

```
#define getKOH_CF2Cl2(T)   (1.0E-12 * exp(-3600.0/(T)))    /* k99 */
#define getKO_HCl(T)       (1.0E-11 * exp(-3300.0/(T)))    /* k100 */
#define getKO_HOCl(T)            (1.0E-11 * exp(-2200.0/(T)))    /* k101 */
#define getKCl_HO2_2(T)          (4.1E-11 * exp(-450.0/(T)))     /* k103 */
#define getKCl_OClO(T)           (3.4E-11 * exp(160.0/(T)))      /* k105 */
#define getKCl_ClOO_1            (2.3E-10)                       /* k106 */
#define getKCl_ClOO_2            (1.2E-11)                       /* k107 */
#define getKCl_Cl2O2       (1.0E-10)                 /* k108 */
#define getKCl_ClONO2(T)   (6.8E-12 * exp(160.0/(T)))      /* k109 */
#define getKCl_ClNO(T)           (5.8E-11 * exp(110.0/(T)))      /* k110 */
#define getKClO_NO3        (4.0E-13)                       /* k111 */
#define getKClO_H2CO(T)          (1.0E-12 * exp(-2100.0/(T)))    /* k112 */
#define getKClO_CO(T)            (1.0E-12 * exp(-3700.0/(T)))    /* k113 */
#define getKClO_N2O(T)           (1.0E-12 * exp(-4300.0/(T)))    /* k114 */
#define getKClO_ClO(T)           (8.0E-13 * exp(-1250.0/(T)))    /* k115 */
#define getKClO_O3(T)            (1.0E-12 * exp(-4000.0/(T)))    /* k116 */
#define getKOH_Cl2(T)            (1.4E-12 * exp(-900.0/(T)))     /* k117 */
#define getKOH_ClNO2             (3.5E-14)                       /* k118 */
#define getKOClO_O(T)            (2.5E-12 * exp(-950.0/(T)))     /* k120 */
#define getKOClO_O3(T)           (2.1E-12 * exp(-4700.0/(T)))    /* k121 */
#define getKOClO_OH(T)           (4.5E-13 * exp(800.0/(T)))      /* k122 */
#define getKOClO_NO(T)           (2.5E-12 * exp(-600.0/(T)))     /* k123 */

/* simplify k in calculations */
#                                                          /* k1 */
#define k2      KO_O3
#                                                          /* k3 */
#define k4      KO1D_N2
#define k5      KO1D_O2
#define k6      KO1D_O3_1
#define k7      KO1D_O3_2
#define k8      KO1D_N2O_1
#define k9      KO1D_N2O_2
#define k10     KNO_O3
#define k11     KNO2_O
/* #define k12  KN_O3
 #define k13    KN_O2
*/
#define k14     KNO2_O3
#define k15     KNO_NO3
#                                                          /* k16 */
#                                                          /* k17 */
#                                                          /* k18 */
#define k21     KO1D_H2O
#define k22     KO1D_CH4_1
#define k23     KO1D_CH4_2
#define k24     KO1D_H2
#define k25     KH_O3
```

```
#                                                    /* k26 */
#define k27     KOH_O3
#define k28     KHO2_O3
#define k29     KOH_O
#define k30     KHO2_O
#define k31     KH2O2_O
#define k32     KOH_CH4
#define k33     KHO2_NO
#define k34     KOH_CO
#define k35     KOH_H2
#                                                    /* k36 */
#define k37     KOH_HNO3
#define k38     KOH_H2O2
#define k39     KOH_HO2
#define k40     KOH_OH
#define k41     KOH_H2CO
#define k42     KHO2_HO2
#                                                    /* k43 */
#define k44     KH_HO2_1
#define k45     KH_HO2_2
#define k46     KH_HO2_3
#define k47     KH2CO_O
#                                                    /* k48 */
#define k49     KCH3O2_NO
#define k50     KCH3O2_HO2
#define k51     KCH3OOH_OH
#define k52     KCH3O_O2
#define k53     KHCO_O2
#define k54     KO1D_CF2Cl2
#define k55     KO1D_CFCl3
#define k56     KO1D_CH3Cl
#define k57     KOH_CH3Cl
#define k58     KCl_O3
#define k59     KClO_O
#define k60     KClO_NO
#define k61     KO1D_HCl
#define k62     KCl_CH4
#define k63     KCl_HO2
#define k64     KCl_H2
#define k65     KHCl_H
#define k66     KCl_H2O2
#define k67     KOH_HCl
#                                                    /* k68 */
#                                                    /* k69 */
#define k70     KHO2_ClO
#define k71     KH_ClO
#                                                    /* k72 */
#define k73     KClONO2_O
```

```
#define k74     KClONO2_OH
#define k75     KH2CO_Cl
#define k76     KH2_O
#define k77     KO1D_CCl4
#define k78     KOH_HOCl
#                                                   /* k79 */
#                                                   /* k80 */
#define k81     KHNO4_OH                            /* k81 */
#                                                   /* k82 */
#define k83     KO_NO3
#define k84     KO_HNO4
#define k85     KH_NO2
#define k86     KO_CH3
#define k87     KO3_CH3
 /* define getKCH3O2_CH3O2(T)    (2.5E-13 * exp(-190.0/(T)))  */
#define k88     KCH3O2_CH3O2_1
#define k89     KCH3O2_CH3O2_2
#define k90     KCH3O2_CH3O2_3
#define k91     KCl_CH3Cl
#define k92     KCl_HOCl
#define k93     KCl_NO3
#define k94     KClO_OH
#define k95     KClO_CH4
#define k96     KClO_H2
#                                                   /* k97 */
#define k98     KOH_CFCl3
#define k99     KOH_CF2Cl2
#define k100    KO_HCl
#define k101    KO_HOCl
#define k102    KO1D_Cl2
#define k103    KCl_HO2_2
                                        /* k104 */
#define k105    KCl_OClO
#define k106    KCl_ClOO_1
#define k107    KCl_ClOO_2
#define k108    KCl_Cl2O2
#define k109    KCl_ClONO2
#define k110    KCl_ClNO
#define k111    KClO_NO3
#define k112    KClO_H2CO
#define k113    KClO_CO
#define k114    KClO_N2O
#define k115    KClO_ClO
#define k116    KClO_O3
#define k117    KOH_Cl2
#define k118    KOH_ClNO2
                                        /* k119 */
#define k120    KOClO_O
```

```
#define k121    KOClO_O3
#define k122    KOClO_OH
#define k123    KOClO_NO

/* 2. == K - termolecular reactions
 * purpose: This file contains all of the termolecular recations that are
 * used in the model. The first letter of the definition indicated that
 * it is a K value (thermal reaction). The next set of letters and
 * numbers are the first reactent and the second (separated by an
 * underscore) is the second and third M.
 *
 * The functions are dependent upon temperature (T), pressure (P), and
 * number density (M).
 */
/* having "_M " in every expression
 * function " TerRxn(T,numden,K300,N,KINF300,M)" is in file "O3util.c",
 * formula for TerRxn is from page 88 of NASA/JPL data.
 * units for Low Pressure  Limit are (cm^6/molecule^2-sec),
 *       for High Pressure Limit are (cm^3/molecule-sec).
 * We only need Low Pressure Limit for following reactions:
 * (1)  O + O2 + M --> O3 + M
 * (2)  Cl + O3 + M --> ClOO + M
 */
#define getKO_O2_M(T,M)   (6.0E-34*pow((T)/300.0,2.3)*(M))  /* k1 */
#define getKNO_O_M(T,M)      (TerRxn((T),(M),9.0E-32,1.5,3.0E-11,0.0)) /* k16 */
#define getKNO2_NO3_M(T,M) (TerRxn((T),(M),2.2E-30,3.9,1.5E-12,0.7)) /* 17 */
#define getKH_O2_M(T,M)      (TerRxn((T),(M),5.7E-32,1.6,7.5E-11,0.0))  /* 26 */
#define getKOH_NO2_M(T,M) (TerRxn((T),(M),2.6E-30,3.2,2.4E-11,1.3)) /* 36 */
#define getKOH_OH_M(T,M) (TerRxn((T),(M),6.9E-31,0.8,1.5E-11,0.0))  /* 43 */
#define getKCH3_O2_M(T,M) (TerRxn((T),(M),4.5E-31,3.0,1.8E-12,1.7)) /* 48 */
#define getKCl_O2_M(T,M)  (2.7E-33*pow((T)/300.0,-1.5)*(M))   /* k68 */
#define getKClO_NO2_M(T,M) (TerRxn((T),(M),1.8E-31,3.4,1.5E-11,1.9)) /* 72 */
#define getKHO2_NO2_M(T,M) (TerRxn((T),(M),1.8E-31,3.2,4.7E-12,1.4)) /* 79 */
#define getKO_NO2_M(T,M)  (TerRxn((T),(M),9.0E-32,2.0,2.2E-11,0.0))  /* 82 */
#define getKHO2_HO2_M(T,M) (1.7E-33 * (M) * exp(1000.0/(T)))        /* 97 */
#define getKCl_NO_M(T,M)   (9.0E-32*pow((T)/300.0,1.6)*(M))     /* k124 */
#define getKCl_NO2_M1(T,M) (TerRxn((T),(M),1.3E-30,2.0,1.0E-10,1.0)) /* k125 */
#define getKCl_NO2_M2(T,M) (TerRxn((T),(M),1.8E-31,2.0,1.0E-10,1.0)) /* k126 */
#define getKCl_CO_M(T,M)   (1.3E-33*pow((T)/300.0,3.8)*(M))     /* k127 */
#define getKClO_ClO_M(T,M) (TerRxn((T),(M),1.9E-32,3.9,7.0E-12,0.0)) /* k128 */

/* double TerRxn(double,double,double,double,double,double);
 *     TerRxn(T,numden,K300,N,KINF300,M)
 */
/* simplify k in calculations */

#define k1      KO_O2_M
#define k16     KNO_O_M
```

```
#define k17     KNO2_NO3_M
#define k26     KH_O2_M
#define k36     KOH_NO2_M
#define k43     KOH_OH_M
#define k48     KCH3_O2_M
#define k68     KCl_O2_M
#define k72     KClO_NO2_M
#define k79     KHO2_NO2_M
#define k82     KO_NO2_M
#define k97     KHO2_HO2_M
#define k124    KCl_NO_M
#define k125    KCl_NO2_M1
#define k126    KCl_NO2_M2
#define k127    KCl_CO_M
#define k128    KClO_ClO_M

/* 3. == J - photodissociation constants  */
/* simplify j in calculations */
#define J1      JO2
#define J2      JO3_J2
#define J3      JO3_J3
#define J4      JNO2
#define J5      JN2O
#define J6      JHNO3
#define J7      JH2O2
#define J8      JNO3_J8
#define J9      JH2CO_J9
#define J10     JH2CO_J10
#define J11     JCF2Cl2
#define J12     JCFCl3
#define J13     JClONO2
#define J14     JHCl
#define J15     JHOCl
#define J16     JN2O5
#define J17     JCH3OOH
#define J18     JCCl4
#define J19     JHNO4
#define J22     JNO3_J22
#define J24     JH2O
#define J20     JClO
#define J21     JCl2
#define J23     JClOO
#define J25     JOClO
#define J27     JCl2O2
#define J28     JClNO
#define J29     JClNO2
#define J30     JClONO
/* the end  of File: O3model.h  */
```

```
9. /* == File: O3output.c */
#include <stdio.h>
#include "O3model.h"
#include "O3util.h"
void output_data(int outputtype, int *output, double Z[][2], double *M)
{
  int i, j;
  double totalcon;

/* -1. ==  concentration for every timestep at a fixed space point
 *        C[i][0..TIME_SPOINTS-1], i is a output[IX], fixed value.
 */
  if (outputtype == 1)
  {
    for (j=0;j<TIME_POINTS;j++)
    {
      printf("%g", (float)(j)/(float)TIME_POINTS);
      if (output[IO] >= 0)
        printf(" %.2e", O[output[IO]][j]);
      if (output[IO1D] >= 0)
        printf(" %.2e", O1D[output[IO1D]][j]);
      if (output[IO3] >= 0)
        printf(" %.2e", O3[output[IO3]][j]);
      if (output[IH] >= 0)
        printf(" %.2e", H[output[IH]][j]);
      if (output[IOH] >= 0)
        printf(" %.2e", OH[output[IOH]][j]);
      if (output[IHO2] >= 0)
        printf(" %.2e", HO2[output[IHO2]][j]);
      if (output[IH2O2] >= 0)
        printf(" %.2e", H2O2[output[IH2O2]][j]);
      if (output[INO] >= 0)
        printf(" %.2e", NO[output[INO]][j]);
      if (output[INO2] >= 0)
        printf(" %.2e", NO2[output[INO2]][j]);
      if (output[INO3] >= 0)
        printf(" %.2e", NO3[output[INO3]][j]);
      if (output[IN2O5] >= 0)
        printf(" %.2e", N2O5[output[IN2O5]][j]);
      if (output[IHNO3] >= 0)
        printf(" %.2e", HNO3[output[IHNO3]][j]);
      if (output[IHNO4] >= 0)
        printf(" %.2e", HNO4[output[IHNO4]][j]);
      if (output[ICl] >= 0)
        printf(" %.2e", Cl[output[ICl]][j]);
      if (output[IClO] >= 0)
        printf(" %.2e", ClO[output[IClO]][j]);
      if (output[IHOCl] >= 0)
```

```
            printf(" %.2e", HOCl[output[IHOCl]][j]);
          if (output[IClONO2] >= 0)
            printf(" %.2e", ClONO2[output[IClONO2]][j]);
          if (output[ICH3] >= 0)
            printf(" %.2e", CH3[output[ICH3]][j]);
          if (output[ICH3O] >= 0)
            printf(" %.2e", CH3O[output[ICH3O]][j]);
          if (output[ICH3O2] >= 0)
            printf(" %.2e", CH3O2[output[ICH3O2]][j]);
          if (output[ICH3OOH] >= 0)
            printf(" %.2e", CH3OOH[output[ICH3OOH]][j]);
          if (output[IH2CO] >= 0)
            printf(" %.2e", H2CO[output[IH2CO]][j]);
          if (output[IHCl] >= 0)
            printf(" %.2e", HCl[output[IHCl]][j]);
          printf("\n");
        }
      }

/* -2 == concentration for every space point at a specific timestep
 *       C[0..SPACE_POINTS-1][j], j is a output[IX], fixed value
 */
      else if (outputtype == 2)
      {
        for (i=0;i<SPACE_POINTS;i++)
        {
          printf("%g", Z[i][0]/100000.0);
          if (output[IO] >= 0)
            printf(" %.2e", O[i][output[IO]]);
          if (output[IO1D] >= 0)
            printf(" %.2e", O1D[i][output[IO1D]]);
          if (output[IO3] >= 0)
            printf(" %.2e", O3[i][output[IO3]]);
          if (output[IH] >= 0)
            printf(" %.2e", H[i][output[IH]]);
          if (output[IOH] >= 0)
            printf(" %.2e", OH[i][output[IOH]]);
          if (output[IHO2] >= 0)
            printf(" %.2e", HO2[i][output[IHO2]]);
          if (output[IH2O2] >= 0)
            printf(" %.2e", H2O2[i][output[IH2O2]]);
          if (output[INO] >= 0)
            printf(" %.2e", NO[i][output[INO]]);
          if (output[INO2] >= 0)
            printf(" %.2e", NO2[i][output[INO2]]);
          if (output[INO3] >= 0)
            printf(" %.2e", NO3[i][output[INO3]]);
          if (output[IN2O5] >= 0)
```

```
            printf(" %.2e", N2O5[i][output[IN2O5]]);
          if (output[IHNO3] >= 0)
            printf(" %.2e", HNO3[i][output[IHNO3]]);
          if (output[IHNO4] >= 0)
            printf(" %.2e", HNO4[i][output[IHNO4]]);
          if (output[ICl] >= 0)
            printf(" %.2e", Cl[i][output[ICl]]);
          if (output[IClO] >= 0)
            printf(" %.2e", ClO[i][output[IClO]]);
          if (output[IHOCl] >= 0)
            printf(" %.2e", HOCl[i][output[IHOCl]]);
          if (output[IClONO2] >= 0)
            printf(" %.2e", ClONO2[i][output[IClONO2]]);
          if (output[ICH3] >= 0)
            printf(" %.2e", CH3[i][output[ICH3]]);
          if (output[ICH3O] >= 0)
            printf(" %.2e", CH3O[i][output[ICH3O]]);
          if (output[ICH3O2] >= 0)
            printf(" %.2e", CH3O2[i][output[ICH3O2]]);
          if (output[ICH3OOH] >= 0)
            printf(" %.2e", CH3OOH[i][output[ICH3OOH]]);
          if (output[IH2CO] >= 0)
            printf(" %.2e", H2CO[i][output[IH2CO]]);
          if (output[IHCl] >= 0)
            printf(" %.2e", HCl[i][output[IHCl]]);
          printf("\n");
        }
      }
/* -3. ==  total column at each timestep
 *       sum over every space point
 */

    else if (outputtype == 3)
    {
      for (j=0;j<TIME_POINTS;j++)
      {
      /*
        printf("%g", (float)(j)/(float)TIME_POINTS);
      */
        if (output[IO] >= 0)
        {
          totalcon = 0.0;
          for (i=0;i<SPACE_POINTS;i++)
            totalcon += (O[i][j]*Z[i][1]);
          printf(" %.2e", totalcon);
        }
        if (output[IO1D] >= 0)
        {
```

```
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
          totalcon += (O1D[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
      if (output[IO3] >= 0)
      {
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
          totalcon += (O3[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
      if (output[IH] >= 0)
      {
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
          totalcon += (H[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
      if (output[IOH] >= 0)
      {
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
            totalcon += (OH[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
      if (output[IHO2] >= 0)
      {
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
          totalcon += (HO2[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
      if (output[IH2O2] >= 0)
      {
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
          totalcon += (H2O2[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
/*    if (output[IN] >= 0)
      {
        totalcon = 0.0;
        for (i=0;i<SPACE_POINTS;i++)
          totalcon += (N[i][j]*Z[i][1]);
        printf(" %.2e", totalcon);
      }
*/
```

```
if (output[INO] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (NO[i][j]*Z[i][1]);
  printf(" %.2e", totalcon);
}
if (output[INO2] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (NO2[i][j]*Z[i][1]);
  printf(" %.2e", totalcon);
}
if (output[INO3] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (NO3[i][j]*Z[i][1]);
  printf(" %.2e", totalcon);
}
if (output[IN2O5] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (N2O5[i][j]*Z[i][1]);
  printf(" %.2e", totalcon);
}
if (output[IHNO3] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (HNO3[i][j]*Z[i][1]);
  printf(" %.2e", totalcon);
}
if (output[IHNO4] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (HNO4[i][j]*Z[i][1]);
  printf(" %.2e", totalcon);
}

if (output[ICl] >= 0)
{
  totalcon = 0.0;
  for (i=0;i<SPACE_POINTS;i++)
    totalcon += (Cl[i][j]*Z[i][1]);
```

```
    printf(" %.2e", totalcon);
}

if (output[IClO] >= 0)
{
    totalcon = 0.0;
    for (i=0;i<SPACE_POINTS;i++)
        totalcon += (ClO[i][j]*Z[i][1]);
    printf(" %.2e", totalcon);
}
if (output[IHOCl] >= 0)
{
    totalcon = 0.0;
    for (i=0;i<SPACE_POINTS;i++)
        totalcon += (HOCl[i][j]*Z[i][1]);
    printf(" %.2e", totalcon);
}

if (output[IClONO2] >= 0)
{
    totalcon = 0.0;
    for (i=0;i<SPACE_POINTS;i++)
        totalcon += (ClONO2[i][j]*Z[i][1]);
    printf(" %.2e", totalcon);
}
if (output[ICH3] >= 0)
{
    totalcon = 0.0;
    for (i=0;i<SPACE_POINTS;i++)
        totalcon += (CH3[i][j]*Z[i][1]);
    printf(" %.2e", totalcon);
}

if (output[ICH3O] >= 0)
{
    totalcon = 0.0;
    for (i=0;i<SPACE_POINTS;i++)
        totalcon += (CH3O[i][j]*Z[i][1]);
    printf(" %.2e", totalcon);
}

if (output[ICH3O2] >= 0)
{
    totalcon = 0.0;
    for (i=0;i<SPACE_POINTS;i++)
        totalcon += (CH3O2[i][j]*Z[i][1]);
    printf(" %.2e", totalcon);
}
```

```
        if (output[ICH3OOH] >= 0)
        {
          totalcon = 0.0;
          for (i=0;i<SPACE_POINTS;i++)
            totalcon += (CH3OOH[i][j]*Z[i][1]);
          printf(" %.2e", totalcon);
        }

        if (output[IH2CO] >= 0)
        {
          totalcon = 0.0;
          for (i=0;i<SPACE_POINTS;i++)
            totalcon += (H2CO[i][j]*Z[i][1]);
          printf(" %.2e", totalcon);
        }

        if (output[IHCl] >= 0)
        {
          totalcon = 0.0;
          for (i=0;i<SPACE_POINTS;i++)
            totalcon += (HCl[i][j]*Z[i][1]);
          printf(" %.2e", totalcon);
        }

        printf("\n");
      }
    }
/* -4. == daily average concentration for each space point
 *        totalcon[i][0..27]/TIME_POINTS
 */

    else if (outputtype == 4)
    {
      for (i=0;i<SPACE_POINTS;i++)
      {
        printf("%2d", (int)(Z[i][0]/100000.0));
        if (output[IO] >= 0)
        {
          totalcon = 0.0;
          for (j=0;j<TIME_POINTS;j++)
            totalcon += (O[i][j]);
          printf(" %.2e", totalcon/TIME_POINTS);
        }
        if (output[IO1D] >= 0)
        {
          totalcon = 0.0;
          for (j=0;j<TIME_POINTS;j++)
            totalcon += (O1D[i][j]);
```

```
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IO3] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (O3[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }

  if (output[IH] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (H[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IOH] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (OH[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IHO2] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (HO2[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IH2O2] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (H2O2[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[INO] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (NO[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[INO2] >= 0)
  {
    totalcon = 0.0;
```

```
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (NO2[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[INO3] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (NO3[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IN2O5] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (N2O5[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IHNO3] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (HNO3[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IHNO4] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (HNO4[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[ICl] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (Cl[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IClO] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (ClO[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IHOCl] >= 0)
  {
```

```
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (HOCl[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IClONO2] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (ClONO2[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[ICH3] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (CH3[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[ICH3O] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (CH3O[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[ICH3O2] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (CH3O2[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[ICH3OOH] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (CH3OOH[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[ICH3O] >= 0)
  {
    totalcon = 0.0;
    for (j=0;j<TIME_POINTS;j++)
      totalcon += (CH3O[i][j]);
    printf(" %.2e", totalcon/TIME_POINTS);
  }
  if (output[IH2CO] >= 0)
```

```
        {
          totalcon = 0.0;
          for (j=0;j<TIME_POINTS;j++)
            totalcon += (H2CO[i][j]);
          printf(" %.2e", totalcon/TIME_POINTS);
        }
        if (output[IHCl] >= 0)
        {
          totalcon = 0.0;
          for (j=0;j<TIME_POINTS;j++)
            totalcon += (HCl[i][j]);
          printf(" %.2e", totalcon/TIME_POINTS);
        }
        printf("\n");
      }
    }
}
```

10. /* === file: O3qy.c

```
 *
 * Contents: This file contains all of the functions for determining
 * the quantum yields for all of the photodissociation processes used
 * in the model.
 */

#include <stdio.h>
#include <math.h>
#include "O3model.h"
#include "O3util.h"

/* -1. --- O2 --- */
/* function: qyO2(double lambda, double temperature)
 *
 * Purpose: Calculate the quantum yield of O2.
 * need to look up WMO-NASA,1985 to get the data.
 * WMO 1985 page 368-376 have no explicit quantum yield
 * data.  I think we can assume that it's 1.0 since in page
 * 101 of NASA-JPL 92 it said "the expected photodissociation
 * quantum yields are unity,unless otherwise stated.
 */

double qyO2(double lambda,double temperature)
{
  return 1.0;
}

/* -2. --- O3 --- WMO-NASA for more data */
/* function: qyO3_J2(double lambda, double temperature)
```

```
 *
 *      O3 + hv                 ->      O2 + O                  J2
 * Purpose: Calculate the quantum yield of O3.
 * need to look up WMO-NASA,1985 to get the data.
 * assume to be 0.1 temperally
 */

double qyO3_J2(double lambda,double temperature)
{
 if (lambda<320.0)
  return 0.1;
 else
  return 0.90;
}

/* -3.  --- O3 --- WMO-NASA for more data */
/* function: qyO3_J3(double lambda, double temperature)
 *
 *      O3 + hv                 ->      O2 + O(1D)              J3
 *
 * purpose: Calculate the quantum yield for the photolysis of Ozone
 * to O(1D) in the wavelength region 305 to 320 nm. Wavelengths below
 * this will be given quantum yields of 0.95 and wavelengths above it
 * will be given quantum yields of zero. This subroutine is only valid
 * for temperatures in [220,300] Kelvin. Temperatures not in this
 * region are given either 220 K or 300 K (depending whether low or
 * high) and processed but the glob variable error_flag is set. The
 * data used for this subroutine comes from ''Chemical Kinetics and
 * Photochemical Data for Use in Stratospheric Modeling'' by DeMore
 * et al.
 */
#define A00  0.94932
#define A01 -0.00017039
#define A02  0.0000014072
#define A0(Tem) (A00 + A01*(Tem) + A02*(Tem)*(Tem))
#define A10 -0.024052
#define A11  0.0010479
#define A12 -0.000010655
#define A1(Tem) (A10 + A11*(Tem) + A12*(Tem)*(Tem))
#define A20  0.018771
#define A21 -0.00036401
#define A22 -0.000018587
#define A2(Tem) (A20 + A21*(Tem) + A22*(Tem)*(Tem))
#define A30 -0.01454
#define A31 -0.000047787
#define A32  0.0000081277
#define A3(Tem) (A30 + A31*(Tem) + A32*(Tem)*(Tem))
#define A40  0.0023287
```

```
#define A41  0.000019891
#define A42 -0.0000011801
#define A4(Tem) (A40 + A41*(Tem) + A42*(Tem)*(Tem))
#define A50 -0.00014471
#define A51 -0.0000017188
#define A52  0.000000072661
#define A5(Tem) (A50 + A51*(Tem) + A52*(Tem)*(Tem))
#define A60  0.000003183
#define A61  0.000000046209
#define A62 -0.0000000016266
#define A6(Tem) (A60 + A61*(Tem) + A62*(Tem)*(Tem))

double qyO3_J3(double lambda,double temperature)
{
  double qy;

  if (temperature > 300.0)
    temperature = 300.0;
  else if (temperature < 220.0)
    temperature = 220.0;
  if (lambda < 305.0)
    return 0.95;
  else if (lambda > 320.0)
    return 0.0;
  else
  {
  temperature = 298.0 - temperature;
  lambda = lambda - 305.0;

  qy = A0(temperature) + A1(temperature) * lambda +
       A2(temperature) * pow(lambda,2.0) +
       A3(temperature) * pow(lambda,3.0) +
       A4(temperature) * pow(lambda,4.0) +
       A5(temperature) * pow(lambda,5.0) +
       A6(temperature) * pow(lambda,6.0);
  if (qy < 0.02)
    qy = 0.0;
  return qy;
 }
}

/* -4. --- NO2 --- */
/* function: qyNO2(double lambda, double temperature)
 *
 * Purpose: Calculate the quantum yield of NO2.
 *
 * Implementation: The data in NASA 90-1 were fit using parabolic aproximation
 * for the wavelengths from 285 to 394 and a forth order polynomial for the
```

```
 * wavelengths from 395 to 423.
 */

#undef A0
#define A0  0.9955302581
#undef A1
#define A1  0.0001790401
#undef A2
#define A2 -0.0000046925

#define B0 -1510.7327103
#define B1    49.226740374
#define B2    -0.5976602682
#define B3     0.0032073379
#define B4    -0.000006424

double qyNO2(double lambda, double temperature)
{
  if (lambda < 285.0)
    return 1.0;
  else if (lambda < 395.0)
    return (A0 + A1*(lambda-285.0) + A2*pow(lambda-285.0,2.0));
  else if (lambda < 423.0)
    return ( B0 + B1*(lambda-285.0) + B2*pow(lambda-285.0,2.0) +
          B3*pow(lambda-285.0,3.0) + B4*pow(lambda-285.0,4.0) );
  else
    return 0.0;
}

/* -5. --- N2O --- */
/* function: qyN2O(double lambda, double temperature)
 *
 * Purpose: Calculate the quantum yield of N2O.
 * Accordingly to NASA/JPL 92-20, quantum yield is unity and
 * the products are  N2 and O(1D).
 */

double qyN2O(double lambda,double temperature)
{
  return 1.0;
}

/* -6. --- HNO3 --- */
/* function: qyHNO3(double lambda, double temperature)
 *
 * Purpose: Calculate the quantum yield of HNO3.
 * Accordingly to NASA/JPL 92-20, quantum yield is ~1 for the
 * OH + NO2 channel in the 200 - 315 nm range,using end product
```

```
 * analysis.
 */
double qyHNO3(double lambda, double temperature)
{
  return 1.0;
}

/* -7. --- H2O2 --- */
/* function: qyH2O2(double lambda, double temperature)
 *
 * Purpose: Calculate the quantum yield of H2O2.
 * Accordingly to NASA/JPL 92-20(page 110), quantum yield is believed
 * to be unity.
 */

double qyH2O2(double lambda,double temperature)
{
  return 1.0;
}

/* -8. + -22. --- NO3 --- */
/* NASA/JPL 92,for overhead sun at the earth's surface,
 *                    J1(NO+O2) = 0.022 s(-1)
 *                    J2(NO2+O) = 0.18  s(-1)
 */
/* function: qyNO3_J8(double lambda, double temperature)
 * function: qyNO3_J22(double lambda, double temperature)
 *
 * The functions for the NO3 quantum yields were derived from Atkinson
 * and Lloyd (1984). Note that the total quantum yield is above zero.
 */

#undef A0
#define A0 0.5
#undef A1
#define A1 30.0
#undef A2
#define A2 589.0

double qyNO3_J8(double lambda, double temperature)
{
  if (lambda < A2)
    return 0.0;
  else
    return (A0 * exp(A1*(1.0-lambda/A2)));
}

/* -8. + -22. --- NO3 --- */
```

```
#undef A0
#define A0 93.877800729
#undef A1
#define A1 -0.283519671
#undef A2
#define A2 0.0002139621

double qyNO3_J22(double lambda, double temperature)
{
  if (lambda < 592.67110079)
    return 1.0;
  else if (lambda < 648.09488675)
    return (A0 + A1*lambda + A2*pow(lambda,2.0));
  else
    return 0.0;
}

/* -9. + -10. --- H2CO --- */
/* == Not all from NASA/JPL 92.
 * function: qyH2CO_J9(double lambda, double temperature)
 * function: qyH2CO_J10(double lambda, double temperature,double pressure)
 */
#undef A0
#define A0 120.075505828
#undef A1
#define A1 -1.336996892
#undef A2
#define A2 0.0049254662
#undef A3
#define A3 -0.000005981

double qyH2CO_J9(double lambda, double temperature)
{
  if (lambda < 240.0)
    return 0.21;
  else if (lambda < 340.0)
    return (A0 + A1*lambda + A2*pow(lambda,2.0) + A3*pow(lambda,3.0));
  else
    return 0.0;
}

/* -9. + -10. --- H2CO --- */
#undef A0
#define A0 5529.8069309
#undef A1
#define A1 -98.797426552
#undef A2
#define A2 0.7025906395
```

```
#undef A3
#define A3 -0.0024853642
#undef A4
#define A4 0.0000043728
#undef A5
#define A5 -0.0000000031
#undef A6
#define A6 4.4827141E-11

double qyH2CO_J10(double lambda, double temperature)
{
  if (lambda < 240.0)
     return 0.5;
  if (lambda < 301.25)
    return 0.251;
  else if (lambda < 303.75)
    return 0.247;
  else if (lambda < 306.25)
    return 0.247;
  else if (lambda < 308.75)
    return 0.252;
  else if (lambda < 311.25)
    return 0.261;
  else if (lambda < 313.75)
    return 0.276;
  else if (lambda < 316.25)
    return 0.316;
  else if (lambda < 318.75)
    return 0.368;
  else if (lambda < 321.75)
    return 0.423;
  else if (lambda < 323.75)
    return 0.480;
  else if (lambda < 326.25)
    return 0.550;
  else if (lambda < 328.75)
    return 0.634;
  else if (lambda < 331.25)
    return 0.697;
  else if (lambda < 333.75)
    return 0.739;
  else if (lambda < 336.25)
    return 0.728;
  else if (lambda < 338.75)
    return 0.667;
  else if (lambda < 341.25)
    return 0.602;
  else if (lambda < 343.75)
```

```
    return 0.535;
  else if (lambda < 346.25)
    return 0.469;
  else if (lambda < 348.75)
    return 0.405;
  else if (lambda < 351.25)
    return 0.337;
  else if (lambda < 353.75)
    return 0.265;
  else
    return 0.197;
}

/* -11. --- CF2Cl2 --- */
/* Function: double qyCF2Cl2(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 136), quantum yield is expected
 * to be unity.
 */
double qyCF2Cl2(double lambda, double temperature)
{
  return 1.0;
}

/* -12. --- CFCl3 --- */
/* Function: double qyCFCl3(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 136), quantum yield is expected
 * to be unity.
 */
double qyCFCl3(double lambda, double temperature)
{
  return 1.0;
}

/* -13. --- ClONO2 --- */
/* Function: double qyClONO2(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 135), the preferred quantum yield
 * values are 0.9 for the Cl + NO3 channel, and a complementary value of
 * 0.1 for the O + ClONO channel.
 */

double qyClONO2(double lambda, double temperature)
{
  return 0.9;
}
```

```
/* -14. --- HCl --- */
/* Function: double qyHCl(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 130), there is no information on
 * quantum yield.  So assume to be 1.0.
 */
double qyHCl(double lambda, double temperature)
{
  return 1.0;
}

/* -15. --- HOCl --- */
/* Function: double qyHOCl(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 131),
 * quantum yield can be 1.0.
 */
double qyHOCl(double lambda, double temperature)
{
  return 1.0;
}

/* -16. --- N2O5 --- */
/* Function: double qyN2O5(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 117),
 * quantum yield can be 1.0.
 */
double qyN2O5(double lambda, double temperature)
{
  return 1.0;
}

/* -17. --- CH3OOH --- */
/* Function: double qyCH3OOH(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 156), quantum yield is 1.0.
 */
double qyCH3OOH(double lambda, double temperature)
{
  return 1.0;
}

/* -18. --- CCl4 --- */
/* Function: double qyCCl4(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 136), quantum yield is expected
 * to be unity.
```

```
 */
double qyCCl4(double lambda, double temperature)
{
  return 1.0;
}

/* -19. --- HNO4 --- */
/* Function: double qyHNO4(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 121), Macleod et al (1988) report
 * that photolysis at 248 nm yields one thrid OH + NO3(0.33) and two thirds
 * HO2 + NO2 (0.67).
 */
double qyHNO4(double lambda, double temperature)
{
  return 0.67;
}

/* -20. --- ClO --- */
/* Function: double qyClO(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 123)
 */

double qyClO(double lambda, double temperature)
{
  return 1.0;
}

/* -24. --- H2O --- */
/* Function: double qyH2O(double lambda, double temperature)
 *
 * Accordingly to NASA/JPL 92-20(page 108), quantum yield is 1.0.
 */
double qyH2O(double lambda, double temperature)
{
  return 1.0;
}

/* -21. --- Cl2 --- */
double qyCl2(double lambda, double temperature)
{
  return 1.0;
}

/* -23. --- ClOO --- */
double qyClOO(double lambda, double temperature)
{
```

```
  return 1.0;
}

/* -25. --- OClO --- */
double qyOClO(double lambda, double temperature)
{
  return 1.0;
}

/* -27. --- Cl2O2 --- */
double qyCl2O2(double lambda, double temperature)
{
  return 1.0;
}

/* -28. --- ClNO --- */
double qyClNO(double lambda, double temperature)
{
  return 1.0;
}

/* -29. --- ClNO2 --- */
double qyClNO2(double lambda, double temperature)
{
  return 1.0;
}

/* -30. --- ClONO --- */
double qyClONO(double lambda, double temperature)
{
  return 1.0;
}
/* the end of file O3qy.c */
```

11. /* === File: O3stdat.c */

```
#include <math.h>
#include <stdio.h>
#include "O3model.h"
#include "O3util.h"

void get_std_atmos(double *T, double *M, double *P, double Z[][2])
{
  int i, j;
  FILE *fp;
  extern void get_data(char *, double *, int),
              exit_error(char *);

  get_data(CH4_FILE,CH4,SPACE_POINTS);
```

```
  get_data(CO_FILE,CO,SPACE_POINTS);
  get_data(H2_FILE,H2,SPACE_POINTS);
  get_data(N2O_FILE,N2O,SPACE_POINTS);
  get_data(H2O_FILE,H2O,SPACE_POINTS);
  get_data(CCl4_FILE,CCl4,SPACE_POINTS);
  get_data(CFCl3_FILE,CFCl3,SPACE_POINTS);
  get_data(CF2Cl2_FILE,CF2Cl2,SPACE_POINTS);
  get_data(CH3Cl_FILE,CH3Cl,SPACE_POINTS);
  get_data(Kz_FILE,Kz,SPACE_POINTS+2);
  get_data(O3_COLUMN_FILE,O3_column,SPACE_POINTS+11);
  get_data(O2_COLUMN_FILE,O2_column,SPACE_POINTS+11);
  get_data(NOx_FILE,NOx,SPACE_POINTS);

  if ( (fp=fopen(STDATM_FILE,"r")) == NULL)
    exit_error(STDATM_FILE);
  for (i=0;i<SPACE_POINTS;i++)
  {
    fscanf(fp,"%lf %lf %lf %lf %lf", &Z[i][0], &Z[i][1], T+i, P+i, M+i);

/* convert concentration from mixing ratio (ppbv) to number density
 * (molecule/cm^3)
 * convert P from bar to atmosphere (used in KOH_CO)
 * 1 atm = 1.01e5 bar
 */
    P[i] = P[i]/(1.01e5);
    N2[i] = 0.79*M[i];
    O2[i] = 0.21*M[i];
    H2[i] = H2[i]*M[i];
    CH3Cl[i] = CH3Cl[i]*M[i];
    CFCl3[i] = CFCl3[i]*M[i];
    CF2Cl2[i] = CF2Cl2[i]*M[i];
    CCl4[i] = CCl4[i]*M[i];
  }
  fclose(fp);
}
```

12. /* === File: O3utils.c ==== utilities ======= */

```
#include <stdio.h>
#include <math.h>
#include "O3model.h"
#include "O3util.h"

/* ==== void exit_error(char *str)  =====
 *
 * exit_error, which prints an error message to the standard error and
 * exits the program.
 */
void exit_error(char *str)
```

```
{
  perror(str);
  exit(1);
}

/* ==== getline(FILE *fp,char s[]) ====== */
/* This is used in "void get_par_data()" and get_seasonal_data()
 */

getline(FILE *fp, char s[])
{
  int i;

  do
  {
    for (i=0;(s[i]=fgetc(fp))!='\n';i++)
      if (s[i] == EOF)
        return EOF;
  }
  while (s[0] == '#');
  s[i] = 0;
  return 0;
}

/* === O3_data.c ===
 * Purpose: This file contains four functions for inputing data from file:
 *
 *      get_data: This function reads a data file which consists of a
 *              single number per row. The number is assumed to be
 *              double precision.
 *
 *      get_2_data: This function reads a data file which consists of
 *              two numbers per row. The numbers are assumed to be double
 *              precision.
 *
 *      get_3_data: This function reads a data file which consists of
 *              three numbers per row. The numbers are assumed to be double
 *              precision.
 *
 *      get_par_data: This function reads the parameter file data. Because
 *              this file will be changed often, this function is smarter
 *              than the other functions in terms of reading the data. In
 *              particular, this function skips commented lines (lines that
 *              begin with the '#' character).
 *
 */

/* FUNCTION: get_data
```

```
 *
 * PASSED VARIABLES
 *
 * fname        The name of the data file to be opened.
 * data         An array to hold the data read
 * max          The maximum number of data points to read
 */

void get_data(char *fname, double *data, int max)
{
  FILE *fp;
  int i;
  extern void exit_error(char *);

  if ( (fp=fopen(fname,"r")) == NULL)
    exit_error(fname);
  for (i=0;fscanf(fp,"%lf", data+i) != EOF && i<max;i++)
     ;
  fclose(fp);
}

/* FUNCTION: get_2_data
 *
 * PASSED VARIABLES
 *
 * fname        The name of the data file to be opened.
 * data         A 2-D array into which the data is read.
 */

int get_2_data(char *fname, double data[][2])
{
  int i;
  FILE *fp;
  extern void exit_error(char *);

  if ( (fp=fopen(fname,"r")) == NULL)
    exit_error(fname);
  for (i=0;fscanf(fp,"%lf %lf", &(data[i][0]), &(data[i][1])) != EOF;i++)
    ;
  fclose(fp);
  return i;
}

/* FUNCTION: get_3_data
 *
 * PASSED VARIABLES
 *
 * fname        The name of the data file to be opened.
```

```
 * data          A 3-D array into which the data is read.
 */

int get_3_data(char *fname, double data[][3])
{
   int i;
   FILE *fp;
   extern void exit_error(char *);

   if ( (fp=fopen(fname,"r")) == NULL)
      exit_error(fname);
   for (i=0;fscanf(fp,"%lf %lf %lf", &(data[i][0]),
                                     &(data[i][1]),
                                     &(data[i][2])) != EOF;i++)
      ;
   fclose(fp);
   return i;
}

/* === double change(double,double) ==== */
/* This function is used in File: O3cons.c to determine the rate change
 * of species.
 */
double change(double old,double new)
 {
    if (new > 0)
       return fabs((old-new)/new);
    else
       return 0.0;
 }

/* === double avgM(int,double *) === */
/* This function is used in File: O3cons.c to determine the average
 * M between layers
 */
double avgM(int pos, double *M)
{
   if (pos == 0)
      return (M[0]);
   else if (pos == SPACE_POINTS)
      return (M[SPACE_POINTS-1]);
   else
      return ((M[pos]+M[pos+1])/2);
}
/* === double getM(int,double *) === */
/* This function is used in File: O3cons.c to determine the
 * M in a specific layer
 */
```

```
double getM(int pos, double *M)
{
  if (pos == -1)
    return M[0];
  else if (pos == SPACE_POINTS)
    return M[SPACE_POINTS-1];
  else
    return M[pos];
}

double getKz(int pos, double *Kz)
{
    return ((Kz[pos]+Kz[pos+1])/2.0);
}
/* === double trans(int,double [][],int,int,double,double,double *) === */
/* This function is used in File: O3cons.c to determine the
 * transport term in a specific layer
 */
double trans(int which, double con[SPACE_POINTS+2][TIME_POINTS+1],
             int i, int j, double rhop, double rhom, double *M)
{
  double retval;

  if (which == 0)
  {
    retval = rhom*con[i-1][j]/getM(i-1,M) +
             rhop*con[i+1][j]/getM(i+1,M);
  }
  else
    retval = rhom/getM(i,M) + rhop/getM(i,M);
  return retval;
}

/* function : get_zenith(int timestep)
 * This is used in File:O3j.c to determine the zenith angle
 */
double get_zenith(int timestep)
{
  double zenith;

  zenith = acos(sin(inclination)*sin(latitude) +
               cos(inclination)*cos(latitude) *
               cos(PI*(double)(timestep-TIME_POINTS/2) /
                    ((double)TIME_POINTS/2.0)));
  return zenith;
}
/* ==== Function: TerRxn ======
 * Been used in file O3_rxn.h and O3_kter.h
```

```
 *
 * purpose: This function calculates the termolecular reaction rate
 * given the temperature (T), number density (M), and the four reaction
 * perameters (see DeMore et al.).
 *
 * arguments:
 *
 * T: Temperature
 * numden: Number Density
 * K300: Low pressure K limit at 300 kelvin
 * N: Low pressure temperature factor
 * KINF300: High pressure K limit at 300 kelvin
 * M: High pressure temperature factor
 */

double TerRxn(double TT, double numden, double K300, double N, double KINF300, double
MM)
 /* double TerRxn(TT, numden, K300, N, KINF300, MM)
  * double TT, numden, K300, N, KINF300, MM;
  */


{
   double K0, KINF;

   K0   = K300    * pow(TT/300.0,-N);
   KINF = KINF300 * pow(TT/300.0,-MM);
   return ( K0 * numden / ( 1.0 + K0 * numden / KINF ) ) *
         pow(0.6,1.0/(1.0+pow(log10(K0*numden/KINF),2.0)));
}

void tridag(double a[],double b[],double c[],double r[],double u[],int n)
{
   int j;
   double bet, *gam;
   extern void nrerror();

   gam = (double *) malloc((n)*sizeof(double));
   if (b[1] == 0.0)
      nrerror("Error 1 in Tridag");
   u[1]=r[1]/(bet=b[1]);
   for (j=2;j<=n;j++)
   {
      gam[j] = c[j-1]/bet;
      bet = b[j] - a[j]*gam[j];
      if (bet == 0.0)
          nrerror("Error 2 in Tridag");
      u[j] = (r[j]-a[j]*u[j-1])/bet;
   }
```

```
  for (j=(n-1);j>=1;j--)
    u[j] -= gam[j+1]*u[j+1];
  free(gam);
}

void nrerror(char error_text[])
{
  fprintf(stderr,"%s\n", error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
 }

/* the end of File: O3utils.c  */
```

13. /* === File: O3util.h */

```
/* This header file contains all the prototype declarations of funtions
 * and variables used in the O3 model.
 *
 * Part.1 are all functions prototype
 * Part.2 are variables declarations.
 */
#include "O3model.h"

/* ----- Part.1 ----- functions */
 /* from file: O3acs.c      ==========      18 */
double getacsO2(double);   /*  <---- */
double getacsO3(double);   /*  <---- */
double getacsNO2(double,double);
double getacsN2O(double,double);
double getacsHNO3(double,double);
double getacsH2O2(double,double);
double getacsNO3(double,double);
double getacsH2CO(double,double);
double getacsCF2Cl2(double,double);
double getacsCFCl3(double,double);
double getacsClONO2(double,double);
double getacsHCl(double,double);
double getacsHOCl(double,double);
double getacsN2O5(double,double);
double getacsCH3OOH(double,double);
double getacsCCl4(double,double);
double getacsHNO4(double,double);
double getacsH2O(double,double);
double getacsClO(double,double);
double getacsCl2(double,double);
double getacsOClO(double,double);
double getacsClOO(double,double);
double getacsCl2O2(double,double);
```

```
double getacsClNO(double,double);
double getacsClONO(double,double);
double getacsClNO2(double,double);

/* from file: O3actinic.c ======          8 */
double AMdir(double);
void get_layer_density(double *, double [][], double *);
double get_O3_optical_depth(int, double *, double);
double get_O2_optical_depth(int, double *, double);
void  fill_mat(double *, double *,double *,double *,double *,
        double *,double *, double *, double, double);
void initialize_angles(double *);
int get_solar_data(double *, double *, double *);
double avg(double *,int, int);

/* from file: O3cons.c    ========      1  */
void get_cons(int *,double );
/* from file: O3init.c    =======       1  */
void get_initial_cons(int *);
/* from file: O3j.c       ========      22 */
double Actinic(int,double,int);

double
        getJO2(double,int,int),
        getJO3_J2(double,int,int),
        getJO3_J3(double,int,int),
        getJNO2(double,int,int),
        getJN2O(double,int,int),
        getJHNO3(double,int,int),
        getJH2O2(double,int,int),
        getJNO3_J8(double,int,int),
        getJH2CO_J9(double,int,int),
        getJH2CO_J10(double,int,int),
        getJCF2Cl2(double,int,int),
        getJCFCl3(double,int,int),
        getJClONO2(double,int,int),
        getJHCl(double,int,int),
        getJHOCl(double,int,int),
        getJN2O5(double,int,int),
        getJCH3OOH(double,int,int),
        getJCCl4(double,int,int),
        getJHNO4(double,int,int),
        getJClO(double,int,int),
        getJNO3_J22(double,int,int),
        getJH2O(double,int,int);
void get_J(double *, double *, double *);

/* from file: O3k.c     ========     1 */
```

```
void get_K(double *,double *,double *);

/* from file: O3output.c  =======     1 */
void output_data(int,int *,double [][2],double *);
/* from file: O3qy.c  ========      21  */
double qyO2(double,double);
double qyO3_J2(double,double);
double qyO3_J3(double,double);
double qyNO2(double,double);
double qyN2O(double,double);
double qyHNO3(double,double);
double qyH2O2(double,double);
double qyNO3_J8(double,double);
double qyNO3_J22(double,double);
double qyH2CO_J9(double,double);
double qyH2CO_J10(double,double);
double qyCF2Cl2(double,double);
double qyCFCl3(double,double);
double qyClONO2(double,double);
double qyHCl(double,double);
double qyHOCl(double,double);
double qyN2O5(double,double);
double qyCH3OOH(double,double);
double qyCCl4(double,double);
double qyHNO4(double,double);
double qyH2O(double,double);
double qyClO(double,double);
double qyCl2(double,double);
double qyOClO(double,double);
double qyClOO(double,double);
double qyCl2O2(double,double);
double qyClNO(double,double);
double qyClONO(double,double);
double qyClNO2(double,double);

/* from file: O3readflux.c  ========= 1 */
void readflux(char *);
/* from file: O3stdat.c  ========    1  */
void get_std_atmos(double *,double *,double *,double [][2]);
/* from file: O3util.c */
void exit_error(char *);
int  getline(FILE *fp,char s[]);
void get_data(char *,double*,int);
int get_2_data(char *,double [][2]);
int get_3_data(char *,double [][3]);
void get_par_data(char *,int *, int *,int *,double *);
double change(double,double);
double avgM(int,double *);
```

```
double getM(int,double *);
double getKz(int, double *);
double trans(int,double [][],int,int,double,double,double *);
double get_zenith(int);
double TerRxn(double,double,double,double,double,double);
void solve_mat(double **,int,int,int,double *);
void bandec(double **,
            unsigned long,
            int,
            int,
            double **,
            unsigned long [],
            double *);
void banbks(double **,
            unsigned long,
            int,
            int,
            double **,
            unsigned long [],
            double []);
void tridag(double *,double *,double *,double *,double *,int);

double **dmatrix(long,long,long,long);
double **submatrix(double **,long,long,long,long,long,long);
void free_submatrix(double **,long,long,long,long);
void nrerror(char []);

/* ===== Part.2  ===== variables    */
/* --- concentrations ---  */
   double O[SPACE_POINTS+2][TIME_POINTS+1],
          O1D[SPACE_POINTS+2][TIME_POINTS+1],
          O3[SPACE_POINTS+2][TIME_POINTS+1],
          H[SPACE_POINTS+2][TIME_POINTS+1],
          OH[SPACE_POINTS+2][TIME_POINTS+1],
          HO2[SPACE_POINTS+2][TIME_POINTS+1],
          H2O2[SPACE_POINTS+2][TIME_POINTS+1],
/*          N[SPACE_POINTS+2][TIME_POINTS+1], */
          NO[SPACE_POINTS+2][TIME_POINTS+1],
          NO2[SPACE_POINTS+2][TIME_POINTS+1],
          NO3[SPACE_POINTS+2][TIME_POINTS+1],
          N2O5[SPACE_POINTS+2][TIME_POINTS+1],
          HNO3[SPACE_POINTS+2][TIME_POINTS+1],
          HNO4[SPACE_POINTS+2][TIME_POINTS+1],
          Cl[SPACE_POINTS+2][TIME_POINTS+1],
          ClO[SPACE_POINTS+2][TIME_POINTS+1],
          HOCl[SPACE_POINTS+2][TIME_POINTS+1],
          ClONO2[SPACE_POINTS+2][TIME_POINTS+1],
          CH3[SPACE_POINTS+2][TIME_POINTS+1],
```

```
        CH3O[SPACE_POINTS+2][TIME_POINTS+1],
        CH3O2[SPACE_POINTS+2][TIME_POINTS+1],
        CH3OOH[SPACE_POINTS+2][TIME_POINTS+1],
        H2CO[SPACE_POINTS+2][TIME_POINTS+1],
        HCl[SPACE_POINTS+2][TIME_POINTS+1],
        Cl2[SPACE_POINTS+2][TIME_POINTS+1],
        ClOO[SPACE_POINTS+2][TIME_POINTS+1],
        OClO[SPACE_POINTS+2][TIME_POINTS+1],
        Cl2O2[SPACE_POINTS+2][TIME_POINTS+1],
        ClNO[SPACE_POINTS+2][TIME_POINTS+1],
        ClNO2[SPACE_POINTS+2][TIME_POINTS+1],
        ClONO[SPACE_POINTS+2][TIME_POINTS+1];
    double
        T[SPACE_POINTS+2],
        P[SPACE_POINTS+2],
        M[SPACE_POINTS+2],
        Z[SPACE_POINTS+2][2],
          Kz[SPACE_POINTS+2],
          O3_column[SPACE_POINTS+13],
          O2_column[SPACE_POINTS+13],
          CO[SPACE_POINTS+2],
        H2O[SPACE_POINTS+2],
        N2[SPACE_POINTS+2],
        O2[SPACE_POINTS+2],
        H2[SPACE_POINTS+2],
        N2O[SPACE_POINTS+2],
        CH4[SPACE_POINTS+2],
        CF2Cl2[SPACE_POINTS+2],
        CFCl3[SPACE_POINTS+2],
        CCl4[SPACE_POINTS+2],
        CH3Cl[SPACE_POINTS+2],
        NOx[SPACE_POINTS+2];
double   inclination,
         latitude,
         crit;
int      solve[34],
         output[34],
         converge[34],
         outputtype;
int num_wavelength_bins;
double ActFlux[WAVELENGTH_POINTS][ANGLE_POINTS][SPACE_POINTS+13],
      ActAngle[ANGLE_POINTS],
      ActBin[WAVELENGTH_POINTS];
/*
   List of absorption cross sections for ozone at various wavelengths.
   The 0 colum contains the wavelengths and the 1 column contains the
   absorption cross sections.
```

```
   List of absorption cross sections for other gases at various wavelengths.
   The 0 colum contains the wavelengths, the 1 column contains the
   absorption cross sections, and the 2 column contains the linear
   change in ACS per Kelvin temperature change.
 */
 /* ---acs ---  absorption cross sections --- */
double acsO2[MAX][2],                    /* O2  J1  */
       acsO3[MAX][2],                /* O3  J2 J3 */
       acsNO2[MAX][3],                   /* NO2  J4  */
       acsN2O[MAX][2],                   /* N2O  J5  */
       acsHNO3[MAX][2],                  /* HNO3  J6  */
       acsH2O2[MAX][2],                  /* H2O2 J7  */
       acsNO3[MAX][2],                   /* NO3  J8 J22 */
       acsH2CO[MAX][2],                  /* H2CO  J9 J10  */
       acsCF2Cl2[MAX][2],            /* CF2Cl2 J11 */
       acsCFCl3[MAX][2],             /* CFCl3 J12 */
       acsClONO2[MAX][2],            /* ClONO2 J13 */
       acsHCl[MAX][2],                   /* HCl  J14  */
       acsHOCl[MAX][2],                  /* HOCl  J15  */
       acsN2O5[MAX][2],                  /* N2O5  J16  */
       acsCH3OOH[MAX][2],            /* CH3OOH  J17  */
       acsCCl4[MAX][2],                  /* CCl4  J18  */
       acsHNO4[MAX][2],                  /* HNO4  J19  */
       acsH2O[MAX][2],                   /* H2O  J24  */
       acsClO[MAX][2],                   /* ClO  J20  */
       acsCl2[MAX][2],               /* Cl2  J21  */
       acsClOO[MAX][2],                  /* ClOO  J23  */
       acsOClO[MAX][2],                  /* OClO  J25  */
       acsCl2O2[MAX][2],             /* Cl2O2  J27  */
       acsClNO[MAX][2],                  /* ClNO  J28  */
       acsClNO2[MAX][2],             /* ClNO2  J29  */
       acsClONO[MAX][2];             /* ClONO  J30  */
/* The number of elements in the ACS arrays. */
int acsO2_NUM,
    acsO3_NUM,
    acsNO2_NUM,
    acsN2O_NUM,
    acsHNO3_NUM,
    acsH2O2_NUM,
    acsNO3_NUM,
    acsH2CO_NUM,
    acsCF2Cl2_NUM,
    acsCFCl3_NUM,
    acsClONO2_NUM,
    acsHCl_NUM,
    acsHOCl_NUM,
    acsN2O5_NUM,
    acsCH3OOH_NUM,
```

```
    acsCCl4_NUM,
    acsHNO4_NUM,
    acsH2O_NUM,
    acsClO_NUM,
    acsCl2_NUM,
    acsClOO_NUM,
    acsOClO_NUM,
    acsCl2O2_NUM,
    acsClNO_NUM,
    acsClNO2_NUM,
    acsClONO_NUM;

/* --- reaction contants --- */
/* --- bimolecular reactions --- */
   double KO_O3[SPACE_POINTS+1],
          KO1D_N2[SPACE_POINTS+1],
          KO1D_O2[SPACE_POINTS+1],
          KO1D_O3_1[SPACE_POINTS+1],
          KO1D_O3_2[SPACE_POINTS+1],
          KO1D_N2O_1[SPACE_POINTS+1],
          KO1D_N2O_2[SPACE_POINTS+1],
          KNO_O3[SPACE_POINTS+1],
          KNO2_O[SPACE_POINTS+1],
           KNO2_O3[SPACE_POINTS+1],
          KNO_NO3[SPACE_POINTS+1],
          KO1D_H2O[SPACE_POINTS+1],
          KO1D_CH4_1[SPACE_POINTS+1],
          KO1D_CH4_2[SPACE_POINTS+1],
          KO1D_H2[SPACE_POINTS+1],
          KH_O3[SPACE_POINTS+1],
          KOH_O3[SPACE_POINTS+1],
          KHO2_O3[SPACE_POINTS+1],
          KOH_O[SPACE_POINTS+1],
          KHO2_O[SPACE_POINTS+1],
          KH2O2_O[SPACE_POINTS+1],
          KOH_CH4[SPACE_POINTS+1],
          KHO2_NO[SPACE_POINTS+1],
          KOH_CO[SPACE_POINTS+1],
          KOH_H2[SPACE_POINTS+1],
          KOH_HNO3[SPACE_POINTS+1],
          KOH_H2O2[SPACE_POINTS+1],
          KOH_HO2[SPACE_POINTS+1],
          KOH_OH[SPACE_POINTS+1],
          KOH_H2CO[SPACE_POINTS+1],
          KHO2_HO2[SPACE_POINTS+1],
          KH_HO2_1[SPACE_POINTS+1],
          KH_HO2_2[SPACE_POINTS+1],
          KH_HO2_3[SPACE_POINTS+1],
```

KH2CO_O[SPACE_POINTS+1],
KCH3O2_NO[SPACE_POINTS+1],
KCH3O2_HO2[SPACE_POINTS+1],
KCH3OOH_OH[SPACE_POINTS+1],
KCH3O_O2[SPACE_POINTS+1],
KHCO_O2[SPACE_POINTS+1],
KO1D_CF2Cl2[SPACE_POINTS+1],
KO1D_CFCl3[SPACE_POINTS+1],
KO1D_CH3Cl[SPACE_POINTS+1],
KOH_CH3Cl[SPACE_POINTS+1],
KCl_O3[SPACE_POINTS+1],
KClO_O[SPACE_POINTS+1],
KClO_NO[SPACE_POINTS+1],
KO1D_HCl[SPACE_POINTS+1],
KCl_CH4[SPACE_POINTS+1],
KCl_HO2[SPACE_POINTS+1],
KCl_H2[SPACE_POINTS+1],
KHCl_H[SPACE_POINTS+1],
KCl_H2O2[SPACE_POINTS+1],
KOH_HCl[SPACE_POINTS+1],
KHO2_ClO[SPACE_POINTS+1],
KH_ClO[SPACE_POINTS+1],
KClONO2_O[SPACE_POINTS+1],
KClONO2_OH[SPACE_POINTS+1],
KH2CO_Cl[SPACE_POINTS+1],
KH2_O[SPACE_POINTS+1],
KO1D_CCl4[SPACE_POINTS+1],
KOH_HOCl[SPACE_POINTS+1],
KHNO4_OH[SPACE_POINTS+1],
KO_NO3[SPACE_POINTS+1],
KO_HNO4[SPACE_POINTS+1],
KH_NO2[SPACE_POINTS+1],
KO_CH3[SPACE_POINTS+1],
KO3_CH3[SPACE_POINTS+1],
KCH3O2_CH3O2_1[SPACE_POINTS+1],
KCH3O2_CH3O2_2[SPACE_POINTS+1],
KCH3O2_CH3O2_3[SPACE_POINTS+1],
KCl_CH3Cl[SPACE_POINTS+1],
KCl_HOCl[SPACE_POINTS+1],
KCl_NO3[SPACE_POINTS+1],
KClO_OH[SPACE_POINTS+1],
KClO_CH4[SPACE_POINTS+1],
KClO_H2[SPACE_POINTS+1],
KOH_CFCl3[SPACE_POINTS+1],
KOH_CF2Cl2[SPACE_POINTS+1],
KO_HCl[SPACE_POINTS+1],
KO_HOCl[SPACE_POINTS+1],
KO1D_Cl2[SPACE_POINTS+1],

```
        KCl_HO2_2[SPACE_POINTS+1],
        KCl_OClO[SPACE_POINTS+1],
        KCl_ClOO_1[SPACE_POINTS+1],
        KCl_ClOO_2[SPACE_POINTS+1],
        KCl_Cl2O2[SPACE_POINTS+1],
        KCl_ClONO2[SPACE_POINTS+1],
        KCl_ClNO[SPACE_POINTS+1],
        KClO_NO3[SPACE_POINTS+1],
        KClO_H2CO[SPACE_POINTS+1],
        KClO_CO[SPACE_POINTS+1],
        KClO_N2O[SPACE_POINTS+1],
        KClO_ClO[SPACE_POINTS+1],
        KClO_O3[SPACE_POINTS+1],
        KOH_Cl2[SPACE_POINTS+1],
        KOH_ClNO2[SPACE_POINTS+1],
        KOClO_O[SPACE_POINTS+1],
        KOClO_O3[SPACE_POINTS+1],
        KOClO_OH[SPACE_POINTS+1],
        KOClO_NO[SPACE_POINTS+1];

/* --- termolecular Reactions--- */
   double KO_O2_M[SPACE_POINTS+1],
        KNO_O_M[SPACE_POINTS+1],
        KNO2_NO3_M[SPACE_POINTS+1],
        KH_O2_M[SPACE_POINTS+1],
        KOH_NO2_M[SPACE_POINTS+1],
        KOH_OH_M[SPACE_POINTS+1],
        KCH3_O2_M[SPACE_POINTS+1],
        KCl_O2_M[SPACE_POINTS+1],
        KClO_NO2_M[SPACE_POINTS+1],
        KHO2_NO2_M[SPACE_POINTS+1],
        KO_NO2_M[SPACE_POINTS+1],
        KHO2_HO2_M[SPACE_POINTS+1],
        KCl_NO_M[SPACE_POINTS+1],
        KCl_NO2_M1[SPACE_POINTS+1],
        KCl_NO2_M2[SPACE_POINTS+1],
        KCl_CO_M[SPACE_POINTS+1],
        KClO_ClO_M[SPACE_POINTS+1];

/* --- J --- photodissociation constants --- */
   double
        JO2[SPACE_POINTS+1][TIME_POINTS+1],
        JO3_J2[SPACE_POINTS+1][TIME_POINTS+1],
        JO3_J3[SPACE_POINTS+1][TIME_POINTS+1],
        JNO2[SPACE_POINTS+1][TIME_POINTS+1],
        JN2O[SPACE_POINTS+1][TIME_POINTS+1],
        JHNO3[SPACE_POINTS+1][TIME_POINTS+1],
        JH2O2[SPACE_POINTS+1][TIME_POINTS+1],
```

```
        JNO3_J8[SPACE_POINTS+1][TIME_POINTS+1],
        JH2CO_J9[SPACE_POINTS+1][TIME_POINTS+1],
        JH2CO_J10[SPACE_POINTS+1][TIME_POINTS+1],
        JCF2Cl2[SPACE_POINTS+1][TIME_POINTS+1],
        JCFCl3[SPACE_POINTS+1][TIME_POINTS+1],
        JClONO2[SPACE_POINTS+1][TIME_POINTS+1],
        JHCl[SPACE_POINTS+1][TIME_POINTS+1],
        JHOCl[SPACE_POINTS+1][TIME_POINTS+1],
        JN2O5[SPACE_POINTS+1][TIME_POINTS+1],
        JCH3OOH[SPACE_POINTS+1][TIME_POINTS+1],
        JCCl4[SPACE_POINTS+1][TIME_POINTS+1],
        JHNO4[SPACE_POINTS+1][TIME_POINTS+1],
        JNO3_J22[SPACE_POINTS+1][TIME_POINTS+1],
        JH2O[SPACE_POINTS+1][TIME_POINTS+1],
        JClO[SPACE_POINTS+1][TIME_POINTS+1],
        JCl2[SPACE_POINTS+1][TIME_POINTS+1],
        JClOO[SPACE_POINTS+1][TIME_POINTS+1],
        JOClO[SPACE_POINTS+1][TIME_POINTS+1],
        JCl2O[SPACE_POINTS+1][TIME_POINTS+1],
        JCl2O2[SPACE_POINTS+1][TIME_POINTS+1],
        JClNO[SPACE_POINTS+1][TIME_POINTS+1],
        JClNO2[SPACE_POINTS+1][TIME_POINTS+1],
        JClONO[SPACE_POINTS+1][TIME_POINTS+1];
/* the end of File: O3util.h */
```