MOOG
An LTE Stellar Line Analysis Program

Chris Sneden
Department of Astronomy
University of Texas at Austin
April 22, 2002

I. Introduction

MOOG is a FORTRAN code that performs a variety of LTE line analysis and spectrum synthesis tasks. The typical use of MOOG is to assist in the determination of the chemical composition of a star. Here is an introduction to MOOG, to accompany a new version of the code that has been created in early 2002.

The basic equations of LTE stellar line analysis exist in a variety of literature sources and will not be repeated here. The appendices of my PhD Dissertation (1973, Univ. of Texas at Austin) quote and discuss these equations. The particular formulation employed in MOOG follows the development of F. N. Edmonds, Jr. (1969, *JQSRT*, **9**, 1427).

The coding is in various subroutines that are called from a few *driver* routines; all these routines (files ending in ".f") are written in fairly standard FORTRAN77. Most information intended to be shared between subroutines is stored in a set of COMMON blocks (files ending in ".com"). I have tried to stay away from a lot of machine-dependent coding, BUT! The standard MOOG version available from me has been developed and implemented on SUN Sparcstations running under SunOS or Solaris unix and on PCs (or laptops) running under Redhat linux. There are a few areas of coding that might not be graciously received by other operating systems and their associated FORTRAN compilers (DOS, VMS, ...). To those few of you who might such use alternate operating systems, good luck! If you find coding solutions that will benefit the general MOOG user, let me know and I will incorporate them into standard MOOG.

One of the chief assets of MOOG is its ability to do on-line graphics. This means, of course, that the plotting commands are given within the FORTRAN code. Inevitably I had to make a particular choice of plotting package. MOOG currently uses and supports only *sm* (sometimes called *SuperMONGO*). This graphics package was chosen for its ease of use and ability to do color graphics. The new MOOG release (April 2002) with *sm* graphics is the only code version that will be actively supported by me. The *sm* package is commercially available (see *http://www.astro.princeton.edu/~rhl/sm/*) and thus is the one likely to survive for the foreseeable future.

An older MOOG version using *LickMONGO* is still available, but that code does not have any of the latest program changes, and it will not be updated in the future. MOOG users running *LickMONGO* are urged to switch to the *sm* version now. To those of you who do not have access to sm or *LickMONGO*, again good luck! Fortunately, the plotting calls are concentrated in just a few routines, and it should be possible for users of other graphics packages (e.g. *pgplot*) to substitute their own commands (remember, the desired plotting routines must be FORTRAN-callable).

A more general cautionary note really needs to be taken to heart by all MOOG users. This code, like all others of its type, makes opening assumptions about the kinds of stellar atmosphere situations that it will face. It has certain internally-stored atomic and molecular data that have been culled from various literature sources. And it includes, for example, representations of the typical major opacity sources that are easily coded, and such representations have proven to be adequate for most tasks. **HOWEVER**, it is the user's responsibility at all times to think about the computations! Tests need to be carried out to verify the applicability of the MOOG calculations to particular spectroscopic problems. Corrections to MOOG will be cheerfully carried out whenever they are brought to my attention. But those who blindly use this code, publishing whatever numbers MOOG spits out, will have only themselves to blame if abundances deduced with MOOG turn out to be fundamentally flawed.

For the new MOOG release, some minor bug fixes have been attended to (as usual!), and some structural alteration of the code has been accomplished in the background. The program continues to be developed mostly with the fairly restrictive Redhat linux FORTRAN compiler. If MOOG works with this compiler and operating system, it probably will not have problems with more liberal FORTRAN compilers.

Here is a list of major new MOOG features:

- A redesign of the spectrum synthesis plots has been implemented. For instance, spectrum smoothing information is now written at the top of the plot. Also, the formerly very confusing texts that give abundance and isotopic data are now written in more coherent fashions. Specifically, if driver "synth" is running and there is isotopic information, it appears in a hopefully clearer manner at the top of the plot. If "isotop" is running, then elemental abundance information is written at the top. The overall metallicity is written on the line at the bottom of the box that contains the model atmosphere comments.

- Continuous opacities have been updated and expanded. I began with the routines appearing in ATLAS9, but in implementing them for MOOG I rewrote them in the FORTRAN style appearing in the rest of the code. The former single continuous opacity subroutine "Opacit.f" has been completely redone, and this routine now acts mainly to call other subroutines that do the real opacity calculations: "OpacHelium.f", "OpacHydrogen.f", "Opaccouls.f", "Opacmetals.f", and "Opacscat.f

- The EW abundance force-fitting driver "abfind" now can work on molecular lines (thus re-implementing an older feature that was lost in recent versions of the code). Of course, abundances from molecular lines cannot be derived without considering molecular equilibrium. There must be some simplifying assumptions made for such abundance computations to make any common sense, and here they are: (a) For hydrides, the abundance of the "interesting" constituent atom (e.g. oxygen for OH molecules) is the one whose abundance will be altered. (b) For non-hydrides (e.g. CN) the user will be asked to specify which elemental abundance to alter. (c) Since molecular equilibrium calculations must be involved, the procedure has to be done iteratively. First, molecular equilibrium will be done assuming the input abundances of constituent atoms. After force-fitting the EW data, then at the user's option the *average* abundance from the given species will be input into the molecular equilibrium routine and the EW computations done anew. The process can be repeated until the mean output abundance agrees with the abundance input into the molecular equilibrium computations.

- Partition function equations have been corrected and updated. These equations are polynomial representations of partition functions computed based on new compilations of atomic energy level data (Lawler & Sneden 2002, in preparation). Older versions of MOOG used the ATLAS9 polynomial fits to the data of Irwin (1981, *ApJS*, **45**, 621). These formulae have been replaced by new ones when partition functions changes more than 0.05 dex in the 5000-6000K temperature regime have been indicated by the new calculations. Eventually a completely updated set of partition functions will be coded into MOOG, but for now some changes have been made to those neutral and first ion data that were most in need of correction.

- All synthetic spectrum smoothing options may now be set in the parameter files. In previous MOOG versions only the FWHM of a Gaussian could be specified this way (under option *pscale*). Now *pscale* has been killed off (an explanatory error message will be given and the program will abort if that parameter is invoked by the user). In its place, new parameter *plotpars* will accomplish all the former *pscale* functions and allow other smoothing options as well.

- A small additional plotting capability is the ability to display an observed spectrum as a histogram. The command in the parameter file is cleverly called *histogram*.

The rest of this write-up is divided into these sections: II, instructions on how to obtain and implement MOOG; III, instructions on how to run MOOG; IV, examples of the necessary "parameter" files; V & VI, how to construct model atmosphere files, and examples; VII, explanation of the line data files, and examples; VIII, brief descriptions of the functions of each MOOG subroutine; IX, a look at common-block variable names; and X, one example of a "Makefile". Many thanks to Niall Gaffney and Danny Hiltgen for writing some of the MOOG subroutines. I also thank MOOG users (recently especially Suchitra Balachandran, Ann Boesgaard, Inese Ivans, Avi Mandell, Jennifer Simmerer, David Yong; many others also) for testing recent MOOG versions and giving important suggestions to improve the code.

II. Obtaining and Implementing MOOG

For UT Austin Astronomy Department users, if you have direct access to the home disk of my public machine "bellini", you may execute MOOG with *sm* graphics simply by typing on your machine the following command:

```
/data1/bellini/utilities/moog/moog2002/MOOG
```

or if you wish to execute the previous version of MOOG, type:

```
/data1/bellini/utilities/moog/supermoog/MOOG
```

or to execute the previous version of MOOG with *LickMONGO* graphics, type:

```
/data1/bellini/utilities/moog/lickmoog/MOOG
```

and see Section III about what comes next.

You may also obtain the MOOG code directly via ftp to my machine. Make a directory for MOOG on your machine, go to that directory, and execute the following commands:

```
ftp verdi.as.utexas.edu
```

(your login name will be "anonymous"; give your email address as password)

```
cd pub/moogcodes
```

(changing to the public MOOG code directory on my machine)

```
get MOOG2002.tar.Z        (or: get LICKMOOG.tar.Z, get SUPERMOOG.tar.Z)
```

(getting the MOOG version either for *sm LickMONGO*)

```
bye                       (or: quit)      (or: exit)
```

(getting out of ftp and back to your machine)

```
uncompress MOOG2002.tar
```

(uncompressing the file)

```
tar -xvf MOOG2002.tar
```

(separating the individual files; you should end up with many *.f files, one *.c file, several *.com files, a few Makefiles, WRITEMOOG.ps, and a README file. The WRITEMOOG.ps can be sent directly to your laser writer)

Now the code must be compiled. If you have a SunOS workstation, type:

```
make -f Makefile.sunos
```

or if you have a Solaris workstation, type:

```
make -f Makefile.solaris
```

or if you have a Redhat linux PC or laptop, , type:

```
make -f Makefile.redhat
```

"Makefile.sunos" and "Makefile.solaris" differ mostly in that they use different library routine links. Only one subroutine, "begin" has different versions for the three machines, in order to correctly sense the screen lengths of windows on the different machines. *It may be necessary to modify this routine if either the program cannot sense the length of your text windows* (I suggest just hard-wiring the screen length variable in this routine) *or in the plotting windows configurations* (just play around with these parameters until things look OK). If you forget and just simply type "make", you will get back this warning message:

```
---------------------------------------------------------------

The appropriate command is either "make -f Makefile.sunos"
or "make -f Makefile.solaris" or "make -f Makefile.redhat".
You also may have to change paths to various libraries to
get the sm routines to work.

---------------------------------------------------------------
```

There also is an example of the kind of linking commands needed to run the FITSIO package; see "Makefile.solfitsio".

You must understand that the the "Makefile"s included with the MOOG package have suggested link sequences, but the libraries on your machine may (almost surely will) be different. Your simplest move here is consultation with your system guru for the correct library linking sequence.

III. Executing MOOG

To run MOOG, simply type "MOOG" on your machine in the appropriate directory. The program will then ask you questions about various input and output files. The first question always will ask for the name of the "parameter file" in which you have specified which *driver* to use. Depending on that *driver* specification, MOOG will ask for other files. It is always necessary to input a "model atmosphere file", and almost always a "line data file". Other files will be called for as needed, including several output files that can be new names or the names of old files that are to be overwritten.

The plotted output of the MOOG2002, SUPERMOOG, and LICKMOOOG versions of the code will have somewhat different "artistic" looks. Only one option in the parameter file must be different between the three versions. The output plotting medium is designated by a character string in *sm*-based MOOG2002 and SUPERMOOG, and by a number in Lick-MONGO-based LICKMOOG. Thus the user designates a character string for "device" the MOOG2002 and SUPERMOOG, and numerical value for "terminal" in LICKMOOG.

The parameter file is an important component of a MOOG run. This file tells MOOG which *driver* to use, how to process the data, and how to output the results. A parameter file can be given any name the user desires; traditionally I have given the name as "something.par". There are a lot of options that the user can invoke in commands that appear in the parameter files! However, don't be intimidated by this: many of the commands (e.g. *histogram*) are entirely optional, and the program can happily run without specifying them.

The first line of the parameter file is the name of the *driver* program to be used. The *driver* is kind of a "master program" which calls the various subroutines that make up MOOG. The current *drivers* are:

synth	→ spectrum synthesis, varying atomic abundances
isotop	→ spectrum synthesis, varying isotopic abundances
plotit	→ re-plotting of spectra that were created in a prior run
abfind	→ force-fitting abundances to match single-line equivalent widths
blends	→ force-fitting abundances to match blended-line equivalent widths
cog	→ curve-of-growth creation for individual lines
cogsyn	→ curve-of-growth creation for blended features
ewfind	→ calculation of equivalent widths of individual lines
calmod	→ converting a BEGN tauross model to an equivalent tau5000 scale
doflux	→ plot the overall flux curve of the model atmosphere
mydriver	→ dummy *driver*; user can substitute a specialty *driver* with this subroutine name

After the *driver* is named, the user may set values for any of the following parameters. A given parameter line consists of a "keyword" (maximum of 10 characters) and either an integer (maximum five digits), or a character string. For character strings to be correctly understood, it is always best to surround them with single quotes; see examples later. The current keywords and acceptable values are listed below. Default values of the parameters are marked with an asterisk (*).

atmosphere: controls the way the model atmosphere is shown on the standard output file:

0	do not output the atmosphere
* 1	output standard information about the atmosphere
2	output more details (continuous opacity factors, etc.)

molecules: controls molecular equilibrium calculations:

* 0	do not do molecular equilibrium
1	do molecular equilibrium but do not output results
2	do molecular equilibrium and output results

trudamp: should more detailed damping calculations for transitions be done if data exist in MOOG?

0	no, stick with the standard damping formulae
* 1	sure, why not? It's a black art anyway!

lines: controls the output of the line data:

0	output nothing about the input lines
* 1	output standard information about the input line list
2	output line opacities at line centers
3	output information about mean depth of line formation
4	output the partition functions

terminal: "terminal type". This is the only parameter which must be declared differently in *sm* and *LickMONGO* versions of MOOG. To find the appropriate parameter value for your use, consult the manual or on-line help for these MONGO versions.
The most popular terminal types for UT Astronomy Department use are:

* 0	MOOG will query the user for the desired terminal type
7	Sunview (*LickMONGO*)
11	Sun OpenWindows, or any X11 (*LickMONGO*)
13	Graph-on 200 series, or any VT/Retrographics (*LickMONGO*)
x11	Sun OpenWindows, or any X11 (*sm*)
xterm	xterm tektronix window (*sm*)
sunview	SunView window (*sm*)
graphon	graphon GO-250 (*sm*)

flux/int:

* 0	perform integrated flux calculations
1	perform central intensity calculations

damping: van der Waals line damping parameter options (N.B. There are "standard" internal approximations in MOOG for radiative and Stark broadening; the next MOOG iteration ought to allow those to be input by the user):

* 0	use the Unsold approximation, BUT: if a factor is read from the line list for an individual line, then if the factor is greater than 10^{-10}, multiply the Unsold value by the factor, otherwise replace the Unsold value by the factor
1	use the Unsold approximation multiplied by 6.3
2	use the Unsold approximation multiplied by a factor recommended by the Blackwell group

units: specifies what wavelength units MOOG will use mainly for output purposes.
All input wavelengths must be consistently in Ångstroms or μm. Values of this parameter can be:

* 0	Angstroms
1	microns

abundances: abundances to override those set during the model atmosphere input.
This option is generally useful only for the "synth" *driver.*
There are *two* required numbers on the line with this keyword; their meanings are:

i	# of elements that diverge from the preset abundances
j	# of different syntheses to be run (maximum=5)

FURTHER EXPLANATION: The lines that follow the "abundances" keyword line contain the information for adjusting the abundances. There must be "i" lines. Each line must contain first the atomic number of the element, then "j" logarithmic abundance differences from the preset values. If one of the elements is called 99, the program understands that the user wishes to vary ALL elemental abundances by the named differences in the syntheses.

synlimits: the wavelength parameters for syntheses. Only the keyword appears on this line. Then on the next line, *four* parameters must appear
in the following order:

a	the beginning synthesis wavelength/frequency
b	the ending synthesis wavelength/frequency
c	the wavelength/frequency step size in the synthesis
d	the wavelength/frequency $\pm\Delta$ from a spectrum point to consider opacity contributions from neighboring transitions

The wavelength units are those consistent with the "units" keyword.

fluxlimits: the wavelength parameters for syntheses. Only the keyword appears on this line. Then on the next line, *four* parameters must appear
in the following order:

- a the beginning flux curve wavelength
- b the ending flux curve wavelength
- c the wavelength step size for the flux curve

The wavelength units are those consistent with the "units" keyword.

coglimits: set the log(W/λ) parameters for curves-of-growth. Only the keyword appears on this line. Then on the next line, either the first *three* parameters or all *five* of parameters must appear in the following order:

- a the log(W/λ) lower limit for the c-o-g
- b the log(W/λ) upper limit for the c-o-g
- c the log(W/λ) step size for the c-o-g
- d an explicit line profile wavelength step size, if desired
- e the element whose abundance is desired to be varied, in a blended line c-o-g

If "e" is entered, then "d" must be put in (choose a zero for this if you wish to let MOOG decide the line profile wavelength step size.

blenlimits: set the parameters for blended line abundance matches. Only the keyword appears on this line. Then on the next line, the following *three* parameters must appear in the following order:

- a $\Delta\lambda$ blueward of the first feature and redward of the last features of the blend to extend the synthesis calculations
- b the wavelength step size of the syntheses
- c the element whose abundance is desired to be varied in order to match the blend EW

obspectrum: indicates the type of the input observed spectrum:

- * 0 no observed spectrum will be input
- 1 a true FITS file
- -1 a true FITS file
- 3 a true FITS file to be read with FITSIO (under development)
- 5 an ASCII file, with lambda as x-coord, flux as y-coord

iraf: allows the user to write an IRAF-readable output unsmoothed spectrum:

- * 0 no IRAF output spectrum with be generated
- 1 generate a spectrum able to be read by IRAF's "rtext'

plot: type of plot output (if any):

- * 0 do not make a plot
- 1 plot only synthetic spectra (synthesis *driver* only)
- 2 plot synthetic and observed spectra (synthesis *driver* only)
- n minimum # of species lines to generate a plot (abundance fit *driver* only)

isotopes: lists isotopes and isotopic ratios to be used. This option is generally useful only for the synthetic spectrum *driver*. There are *two* required numbers on the line with this keyword; their meanings are:

k # of number of isotopic ratios to be set
l # of different syntheses to be run

FURTHER EXPLANATION: "l" will take a maximum of 1 if driver "synth" is used (in other words, set one isotope ratio for all syntheses), and a maximum of 5 otherwise. The lines that follow the "isotopes" keyword line contain the information for adjusting the isotopic ratios. There must be "k" lines. Each line must contain first the floating point number identifying the isotope (so that features of that isotope appearing in the line data file may be identified). This must be given in a particular manner, or MOOG will be very confused. To the left of the decimal point there are the two digits of the atomic number of the atom or the four digits or the atomic numbers of the diatomic molecule. The first digit to the right of the decimal point is the ionization state (0 = neutral, 1 = first ion, etc.). Following that are the three digits for the atomic mass of the elemental isotope or the four digits of the isotopic masses of the molecular constituents. For a molecule, the convention further is that the lightest element of the pair is specified first. Thus for example, "608.01316" is how one must designate a CO molecule composed of C^{13} *and* O^{16}*. In this case one could then designate the ordinary CO molecule as "608.00000", but then MOOG would take internal-stored atomic masses; better to write "608.01216". And to take an atomic example, for* Li^{6} *write "3.006", and for* Li^{7} *write "3.007". After this atomic or molecular identifier should come the parent-to-isotope ratios (understand?); there must be "l" of these on each line. For an atomic example, an ionized line of the* Eu^{153} *is written 63.1153. NOTE A FUNDAMENTAL LIMIT: this style of identifier cannot handle a molecular isotope in which one of the atomic constituents has a mass greater than two digits! Such needs will be taken care of in a future MOOG release. For now, I suggest that the user hard-wire in some recognition of such an identifier if the need ever occurs.*

opacit: a fudge factor to play with the nominal continuous opacity values at a given wavelength:

* 0 no fudge factor is to be applied
a multiply the nominal continuous opacity by a factor:

$$10000 \times a/T$$

where a is a number >0 and T is the temperature of a given atmosphere layer.

freeform: line list reading options:

* 0 Read the line list in the old (7e10.3) formatted manner; blanks are read as zeros
1 Use unformatted reads for the line list; all values must be explicitly given, even zeros

strong: is there a separate line file for "strong" lines whose opacity contributions are to be considered a every wavelength step in a synthesis?

* 0 no, only a standard line data file is to be input
1 yes, ask the user for a separate file of strong lines

plotpars: allows the user to set all of the plotting parameters if they are reasonably well known in advance. It can have two values:

* 0 use default plotting parameters, and manually adjust by looking at the plot
1 set the default plotting parameters in the next *three* lines

OK, so if you choose option "1" of this keyword, you now must immediately put three more lines in the parameter file. On the first line, *four* parameters are required:

a left edge (wavelength/frequency units) of the plot box
b right edge (wavelength/frequency units) of the plot box
c lower edge (relative flux units) of the plot box
d upper edge (relative flux units) of the plot box

On the second line, *four* parameters are required:

e a velocity shift to be applied to the observed spectrum
f a wavelength shift to be applied to the observed spectrum
g a vertical additive shift to be applied to the observed spectrum
h a vertical multiplicative shift to be applied to the observed spectrum

On the third line, *six* parameters are required:

i a one-character smoothing type for the synthetic spectra
j the full-width-at-half-maximum of a Gaussian smoothing function
k *vsini* of a rotational broadening function
l limb darkening coefficient of a rotational broadening function
m the full-width-at-half-maximum of a macrotrubulent broadening function
n the full-width-at-half-maximum of a Lorentzian smoothing function

Permissible smoothing types are g (Gaussian), l (Lorentzian), v (rotational), m (macroturbulent), several combinations of these, and p (variable Gaussian). Just enter zeros for any of the parameters that are not needed for chosen smoothing types.

histogram: allows display of an observed spectrum as a histogram:

* 0 do not do a histogram; simply connect the points instead
1 do a histogram for the observed spectrum

On the parameter options given on this page, note that the string can either be a simple file name, or it can be the file name with a path attached at the beginning, so that files not in the current directory can be accessed. It is important to understand that if you specify one of the output files in the manner below (rather than letting MOOG prompt you for output file names as it runs) then no previous file name check will be done! That is, if you specify a file that already exists, the new MOOG run will write over the old file. So if you want to guard against that, don't specify the output files in your parameter file; let MOOG query you at run time. If you don't care about over-writing, then using these commands may save you time and aggravation. For the input files, if you specify a non-existent file name, then MOOG will ask you for the right name as it runs.

standard_out	'string'	the filename for the standard (verbose) output file
summary_out	'string'	the filename for the EW summary *OR* the raw synthesis output
smoothed_out	'string'	the filename for the smoothed synthetic spectrum output
iraf_out	'string'	the filename for the smoothed synthetic spectrum output to be used in IRAF
summary_in	'string'	the filename for the raw synthetic spectrum (made previously) read in for plotting
model_in	'string'	the filename for the input model atmosphere
lines_in	'string'	the filename for the input line list
stronglines_in	'string'	the filename for the input strong line list
observed_in	'string'	the filename for the input observed spectrum

IV. MOOG Inputs: Parameter File Examples

The first example is one that will produce a synthesis of a 20 Å stretch of spectrum, including the [O I] feature at 6300 Å. In the three syntheses requested, the O abundance will be varied from that specified in the model atmosphere input by –0.5, 0.0, and +0.5 dex. Both the synthetic spectra and an observed spectrum will be displayed, with the plot set to the parameters specified by *plotpars*.

```
synth
standard_out   'out1'
summary_out    'out2'
smoothed_out   'out3'
model_in       '../models/K634.mdf'
lines_in       'lin6305'
observed_in    'K634.6280rd'
terminal       'x11'
atmosphere    1
molecules     2
lines         1
flux/int      0
plot          2
abundances    1    3
         8    -0.5     0.0     0.5
synlimits
  6290.0  6310.0    0.02    1.00
obspectrum    1
plotpars      1
  6290.0  6310.0    0.40    1.03
  0.0      0.000   0.000    1.00
   g        0.25    0.00    0.00    0.00    0.00
```

The second example is one that will do abundance force-fits to equivalent widths of individual (clean) lines of one or more species. Plots of the abundances versus a) excitation potential, b) reduced equivalent width, and c) wavelength will be done (only) if four or more lines are present in the line list for a given species.

```
abfind
terminal       'x11'
standard_out   'out1'
summary_out    'out2'
model_in       'modhm'
lines_in       'linsol.mine'
atmosphere    1
molecules     2
lines         1
freeform      0
flux/int      0
damping       1
plot          4
```

V. MOOG Inputs: Model Atmosphere File Explanations

The subroutine "Inmodel.f" does the work of reading in model atmosphere data. Please consult that code to clear up questions that are not dealt with here. Examples of model atmosphere input files are given in the next section; compare those examples to what is said in this section. There are four different types of model atmosphere input that currently may be handled by MOOG. In any model atmosphere file, the first three lines of information are always the same:

- *line 1:* a left-justified keyword that clues in MOOG to which atmosphere type is about to follow. Permissible model type keywords are 1) KURUCZ, for models generated with the *ATLAS* code; 2) BEGN, for models generated with the *MARCS* code; 3) KURTYPE, for *ATLAS*-generated models that come without continuous opacities even though they are on a "rhox" depth scale (this is a specialized model type); 4) KURPADOVA, for models with quantities arranged in a way for input from the Padova Observatory *ATLAS* program; 5) NEWMARCS, for newest *MARCS* models; and 6) GENERIC, for models that have a "tau" depth scale but no corresponding continuous opacities.

- *line 2:* a comment line; it has no computational purpose, but will appear on various plots and outputs, so take the time to make this line informative!

- *line 3:* the number of depth points. This is an integer $\leq$76 that can be put in any spaces after the first 10 spaces of this line.

The next set of lines (whose count depends on the number of atmosphere layers, hereafter called "ntau") are devoted to the model atmosphere physical quantities. These are slightly different in each model type case, and thus the data will be as follows (choose one):

- if model type is KURUCZ, each of the next *ntau* lines contains these quantities in the following order: ρx, T, P_g, N_e, κ_{Ross}. Here, all quantities have their usual meanings (ρx$\equiv$rhox), and κ is a Rosseland mean opacity on a mass scale. The proper way to get these data is simply to take the direct summary output from an *ATLAS* run. MOOG is supposed to be clever enough to distinguish between the P_g and its logarithm, and between N_e, P_e or either of their logarithms, without any user intervention.

- if model type is BEGN, each of the next *ntau* lines contains these quantities in the following order: τ_{Ross}, T, P_g, N_e, μ, κ_{Ross}. Here, in addition to the remarks for KURUCZ models, note that μ is the mean molecular weight. The proper way to get these data is simply to take the direct summary output from a *MARCS* run.

- if model type is KURTYPE, it is first necessary to have a line that declares the wavelength at which the reference opacity and optical depths will be computed (typically this is set to 5000 Å). Then each of the next *ntau* lines contains these quantities in the following order: ρx, T, P_g, N_e. See remarks for the KURUCZ models, but note that in this case MOOG will compute internally κ_{ref} and τ_{ref} at λ_{ref}.

- if model type is KUR-PADOVA, it is first necessary to have a line that declares the wavelength at which the reference opacity and optical depths will be computed (typically this is set to 5000 Å). Then each of the next *ntau* lines contains the *ATLAS* model quantities τ_{ref}, T, κ_{ref}, N_e, P_g, and ρ.

- if model type is NEWMARCS, each of the next *ntau* lines contains in order τ_{Ross}, T, N_e, P_g, ρ, $v_{\rm turb}$, and κ_{Ross}.

- if model type is GENERIC, the input model is done as in KURTYPE, but instead of the ρx depth variable it is necessary to have τ_{ref}. This τ_{ref} is assumed to have been computed at λ_{ref} (in other words, it is *not* a Rosseland mean opacity). MOOG will compute internally κ_{ref} at λ_{ref}.

The next input data are the microturbulence value(s). This must be a formatted read! The format is 6e13 (6 numbers maximum per line, each taking 13 spaces, with decimal points required). There are two options for this input:

- if only a single number appears in the first 13 spaces, then this is assumed to be the microturbulent velocity for all atmosphere layers. The units of this number may either be km s^{-1} or cm s^{-1}; MOOG should be able to figure it out.

- if more than one number appears, it is assumed that there will be a depth-dependent microturbulent velocity! In this case, it is necessary to designate *ntau* values of the microturbulence, six per line, for as many lines as it takes.

The next input data are the abundances. MOOG internally stores the "solar" abundance set of Anders and Grevesse (1989, *Geochim. Cosmichim. Acta*, **53**, 197). The abundances are given as log ϵ values, where log ϵ(X) $\equiv$ log(N_X/N_H) + 12.0. The solar photospheric values from their paper are adopted except for a) very uncertain solar data, or b) Li, Be, and B; for these cases the meteoritic values are adopted. Also, MOOG adopts a "low" Fe abundance of log ϵ = 7.52, which is roughly consistent with meteoritic results. Here is how stellar abundances are set up in the model atmosphere file:

- the first line contains *two* numbers. These numbers may exist anywhere on the line after the first 10 spaces. The first is the number of elements (hereafter called *natoms*) whose abundances are to be explicitly designated on the following lines. The second is the overall metal deficiency, in the standard [M/H] notation: [M/H] $\equiv \log_{10}(N_M/N_H)_{\rm star} - \log_{10}(N_M/N_H)_{\odot}$. Here, "M" is any element whose abundances is not to be explicitly input; MOOG will add [M/H] to all internal Anders & Grevesse abundances for elements not entered below.

- the next set of lines (as many as needed) contain *natoms* pairs of (atomic number, log ϵ) values. The log ϵ's for these elements override MOOG's internal abundances, and override any overall metallicity [M/H]. This is a free-form read; one can put as many of these number pairs on a line as desired, up to *natoms*.

The final input data are the names of the molecules to be included in any molecular equilibrium calculations. If the user in the parameter file has set "molecules" equal to zero, then forget it! No molecular equilibrium is to be done, no more data from the atmosphere file will be read in. But assuming that the "molecules" keyword has been set to 1 or 2, then here is how the user designates the molecules to be considered:

- the first line contains the number of molecules to take part in molecular equilibrium computations; hereafter that number is called *nmol*. The number may appear on the line anywhere after the first 10 spaces.
- on succeeding lines (as many as needed) the molecule names are to be listed; as many as one likes may be put on one line. The molecule names are concatenations of the two-digit atomic numbers of the constituent elements, with the rule of increasing atomic number from left to right in the name. For examples: H_2 is designated 101 (the leading zero is dropped); CO is 608; CO_2 is 60808. In the equilibrium computations, the neutral atoms will be put in automatically, but if the user wishes to have ionized atoms considered, they must be designated explicitly. An example is the case of MgH, in which much of the Mg exists as Mg II. In this case the user might wish to have four species in the Mg equilibrium: Mg I, Mg II, MgH and MgO. For this, it will be necessary to designate 12.1, 112, and 812 in the model atmosphere file at this point (the neutral atom, 12.0, will be put in automatically).

VI. MOOG Inputs: Model Atmosphere File Examples

The first example is for a *MARCS* model.

```
BEGN
A MARCS model for 4800/2.70/+0.18
NTAU         34
   1.0000e-04   3.4356e+03   2.0960e+00  -2.2620e+00   1.3050e+00    .7300e-03
   2.0000e-04   3.4995e+03   2.2020e+00  -2.1160e+00   1.3050e+00    .9651e-03
       .             .            .             .            .             .
       .             .            .             .            .             .
   7.9433e+00   7.8062e+03   4.4170e+00   2.4290e+00   1.2910e+00    .3921e+00
   1.0000e+01   8.0262e+03   4.4230e+00   2.5640e+00   1.2870e+00    .2040e+00
      2.5e+05
NATOMS        1    +0.18
       7.0      8.20
NMOL         12
     101.0     106.0     107.0     108.0     606.0     607.0     608.0     707.0
     708.0     808.0   10108.0   60808.0
```

The vertical dots shown above represent 30 missing (unprinted) atmosphere layers. This model is for a metal-rich ([M/H] = +0.18) K-giant star. MOOG realizes automatically P_g (the third number on the line for each of the model atmosphere layer) is actually log P_g, and that N_e (the fourth number) is actually log P_e. The abundance of nitrogen is set to log ϵ(N) = 8.20. The microturbulence is 2.5 km s^{-1} at all atmosphere layers. Molecular equilibrium computations will include 12 common molecules of the HCNO element group. Note that the number of atmosphere layers, the number of elements with special abundances, and the number of molecules begin after the first 10 spaces of their input lines, so that one can type NTAU, NATOMS, and NMOL as guides to what those numbers represent.

The second example is for a KURUCZ model. Mostly it is self-explanatory. The last number (0.00E+00) on each model atmosphere layer line is never read by MOOG. The other numbers on these lines are, in order, ρx, T, P_g, N_e,κ_{Ross}, and the radiative acceleration (not used by MOOG). Note the more extensive molecular equilibrium network that has been requested.

```
KURUCZ
          Teff=  5750.0     Log g=    4.90
NTAU          40
 .78541897E-02  3322.1 0.624E+03 0.218E+11 0.407E-02 0.203E+00 0.000E+00
 .10917768E-01  4105.3 0.867E+03 0.127E+12 0.524E-02 0.140E+00 0.000E+00
      .             .            .            .            .            .
      .             .            .            .            .            .
 .39326713E+01 13713.0 0.312E+06 0.483E+17 0.692E+03 0.143E+04 0.000E+00
 .39561312E+01 14935.5 0.314E+06 0.576E+17 0.756E+03 0.156E+04 0.000E+00
     .11E+06
NATOMS        2      0.0
      3.00      3.30     20.00      6.26
NMOL         19
     606.0     106.0     607.0     608.0     107.0     108.0     112.0     707.0
     708.0     808.0      12.1   60808.0   10108.0     101.0       6.1       7.1
       8.1     822.0      22.1
```

VII. MOOG Inputs: The Line Data File

In this file one designates all of the spectral features to take part in MOOG computations. The quantities desired are very straightforward, with only a few variations that depend on the particular MOOG *driver* to be employed. The file always begins with a comment line which has no computational function. But like the atmosphere comment line, this line will be displayed in various plots and output files, so it pays to make it fairly informative. Then on succeeding lines of this file, spectral feature data are entered in either a formatted (7e10) or an unformatted manner, depending on the value of *freeform* in the parameter file. MOOG can hold the data for up to 500 spectral lines in memory at any one time. Thus for deriving abundances by force-fitting equivalent widths, there should be $\leq$500 lines per species. For doing synthetic spectra, there should be ≤500 lines included in the line opacity calculations at any one wavelength step. (MOOG will complain if you exceed that limit, so in such rare cases either decrease the wavelength interval to include lines in such computations, or modify MOOG internally to expand beyond the 500 line limit). Here are the line list inputs; the order of their appearance cannot be changed.

- wavelength, always in Å: For the *driver* "blends", the wavelength of the first feature of a blend has a positive sign, and the wavelengths of the rest of the features of the same blend have negative signs. In all other cases the sign of the wavelength is always positive.
- atomic or molecular identification, as described before: a two digit or concatenated two-digit designation to the left of the decimal point, and a single digit to the right of the decimal point to represent the ionization stage (0 = neutral, 1 = singly ionized, etc.).
- line excitation potential, always in electron volts (eV).
- gf, or log(gf); MOOG figures out which form it is dealing with by looking for a negative value for any one of the entries for gf.
- the van der Waals damping parameter C6, or a factor to multiply an internally generated C6 value; this quantity is optional, and may be left blank in formatted reads. See more information in the discussion on the "damping" option of the parameter file.
- the dissociation energy D_0 (in units of eV) for a molecular feature; this quantity is irrelevant for atomic features, and may be left blank in formatted reads. The reason for requiring this quantity is that the dissociation energy for some molecules is still somewhat uncertain, so the user is forced to enter a favorite value.
- the equivalent width, either EW in units of mÅ, or RW [≡log(EW/λ)]. This quantity is only needed for those *drivers* that force-fit abundances to input equivalent widths, and so may be left blank in formatted reads..

As a first example of a formatted line data file, here is part of a line list that would be used to generate synthetic spectra of an [O I] line.: MOOG realizes that it is receiving gf's, not log(gf)'s since all values of this quantity are positive. It will be looking for the dissociation energy (= 7.65 eV here) each time that it encounters a CN feature. The last 10 columns are comments only, and are ignored.

```
A line list near the [O I] feature
  6299.610       24.0       3.84   1.00E-3
  6299.660       40.0      1.520  1.585E-1
  6299.691      607.0       0.23   4.34E-3                   7.65          12Q2314,0
  6300.265      607.0       1.28   5.78E-3                   7.65          12R11410,5
  6300.310        8.0       0.00  1.78E-10
```

```
  6300.336      28.0     4.266  1.000E-3
  6300.482     607.0      1.31   6.82E-3                    7.65          12R21810,5
  6300.482     607.0      1.24   2.01E-3                    7.65          12Q10210,5
  6300.698      21.1     1.507  1.200E-2
```

And here is how the same list might look with an unformatted read.

```
A line list near the [O I] feature
6299.610 24.0  3.84   1.00E-3  0.  0.  0.
6299.660 40.0  1.520  1.585E-1 0.  0.  0.
6299.691 607.0 0.23   4.34E-3  0. 0.  7.65         12Q2314,0
6300.265 607.0 1.28   5.78E-3  0. 0.  7.65         12R11410,5
6300.310 8.0   0.00  1.78E-10 0.  0.  0.
6300.336 28.0  4.266  1.000E-3 0.  0.  0.
6300.482 607.0 1.31   6.82E-3  0. 0.  7.65         12R21810,5
6300.482 607.0 1.24   2.01E-3  0. 0.  7.65         12Q10210,5
6300.698 21.1  1.507  1.200E-2  0.  0.  0.
```

As a second example of a line data file, here is a short list of lines for which equivalent widths will be used to derive abundances for elements C, Sc, Ti, Fe (from two different species) and Nd. A formatted read is assumed.

```
AR Per 1
  5380.322       6.0      7.68    -1.760                                  70.5
  5640.988      21.1      1.50    -1.010                                 111.8
  7949.149      22.0      1.50    -1.404                                  56.9
  4779.984      22.1      2.05    -1.370                                 181.2
  6301.500      26.0      3.65    -0.720                                 100.0
  7583.796      26.0      3.02    -1.990                                  76.8
  4893.817      26.1      2.83    -4.450                                  54.6
  5249.590      60.1      0.98     0.210                                  79.4
```

As a third example of a line data file, here is a short list of lines to be used in a spectrum synthesis of the CH G-band. Both isotopic forms of molecular lines are considered in this list. The isotopic ratios must be given in the "synth" or "isotop" *driver*, and the molecular names must agree exactly with those specified in the line list (see the conventions for writing these names in discussion of the "isotopes" parameter of the parameter file.

```
A CH synthesis list
  4195.423 106.00112     1.150  4.39E-02                    3.47
  4195.519      28.0      4.09 4.375e-01
  4195.618      26.0      3.02 2.213e-02
  4195.771 106.00113     1.145  4.58E-02                    3.47
  4195.795 106.00113     1.145  4.39E-02                    3.47
  4195.911 106.00112     3.294  9.97E-02                    3.47
  4196.208      26.0      3.40 3.556e-01
```

VIII. MOOG Routines

Here are some very brief descriptions of the subroutines, intended to provide guides for those wanting to understand the inner logic of the program, or those who might want to write their own *driver* codes. The listing here is simply *alphabetically arranged* for ease of finding a particular routine. The *calling sequence* in MOOG can be most easily traced as follows. First look at the code for Moog.f, and find the subroutine that is called for a particular *driver* that you are interested in. Go to that subroutine (which controls the action of MOOG for a particular *driver*), and follow all the subroutine calls that it makes. There should be enough comments in those subroutines, combined with the brief descriptions below, to show what a particular *driver* accomplishes.

Abfind.f: This *driver* routine derives abundances for individual atomic lines. To access it, the driver keyword must be "abfind". As part of the atomic or molecular line list, equivalent widths must be given for each line.

Abunplot.f: This routine does the actual labor of producing the plots of derived abundances versus excitation potential, reduced equivalent width, and wavelength.

Batom.f: Contains atomic data, ordered by atomic number. The sets of data are: atomic symbols, solar abundances, atomic masses, first, second and third ionization potentials, and partition functions (from Kurucz 1970, *ATLAS* write-up) for the first four ionization states of elements.

Begin.redhat.f (or Begin.solaris.f, Begin.sunos.f): Starts up MOOG, initializes some parameters, tries to figure out text window lengths, and declares the properties of the plotting screen. It is mostly for internal use, but the user might want to fool around with this routine if: (a) **Screenlength.c** cannot be induced to figure out the length of a text window on a particular machine; and (b) to make the plotting screens happier (in length, width, background color ...).

Blends.f: This *driver* routine does abundance derivations from blended spectral features; only one element will have its abundance derived per run. To access it, the driver keyword must be "blends". It is useful to determine, say, N abundances from a bunch of CN blended features.

Bmolec.f: Contains molecular equilibrium constant data for most common diatomic and triatomic molecules. The source for this data is Kurucz (1970, *ATLAS* write-up). The ionized atom equilibrium data was done via a least squares fit to the relevant Saha equation data.

Calmod.f: This specialized *driver* re-calculates BEGN τ_{ross} models on a τ_{5000} scale. To access it, the *driver* keyword must be "calmod".

Cdcalc.f: Calculates continuum and line+continuum contribution functions. Contribution functions (see Edmonds' article) are the integrands of the line depth integrals, and are useful in their own right to show where in the stellar atmosphere the major contributions to the line depths occur.

Chabund.f: This routine, which can be called after spectrum synthesis on-line plots, changes the input abundances according to the users request and re-runs the synthesis with the new abundances, OVERWRITING the original output files, but using the same model, line list, etc.

Cog.f: This *driver* routine produces sets of curves-of-growth for each line of the line list. To access it, the *driver* keyword must be "cog".

Cogplot.f: This routine does the actual labor of producing the curve-of-growth plots.

Cogsyn.f: This *driver* routine produces a curve-of-growth of a blended feature, altering only a specified species' abundance. This routine is useful for understanding the influence of contaminants on a blended line. To access it, the *driver* keyword must be "cogsyn".

Curve.f: Does the actual work of producing a curve-of-growth for a single line.

Defcolor.f: This routine sorts out the numbering scheme for colors on plots (and sets all colors to black on paper plots).

Discov.f: Finds the number of times a particular atom appears in a given molecule. It is used in molecular equilibrium computation.

Eqlib.f: Solves the equations of molecular dissociative equilibrium. The equations can include neutral and ionized molecules and atoms.

Ewfind.f: This *driver* routine predicts equivalent widths for individual atomic lines. To access it, the *driver* keyword must be "ewfind".

Fakeline.f: This produces an internal curve-of-growth for an arbitrary ordinary atomic line. Its purpose is to make a curve-of-growth shape that is used by the Abfind.f routine to deduce abundances from equivalent widths.

Findtic.f: Calculates tic marks at nice round intervals; it is used by the plotting routines.

Finish.f: Simply wraps up MOOG; the user should not need to mess with it.

Fluxplot.f: This *driver* routine produces a plot of continuum flux versus wavelength.

Getasci.f: Asks for information from the user, by printing out a message on the screen that requires a user answer. That answer is then used to direct MOOG to further action.

Getcount.f: This extracts the number of characters in an ASCII array until the first blank is detected.

Infile.f: This [integer function] opens up disk files for reading or writing, as appropriate.

Inlines.f: Reads in the line data from a file. The data be entered in a formatted or unformatted manner, as the user chooses with the "freeform" parameter. later section.

Inmodel.f: Reads in the model atmosphere data. See the input section for further details.

Invert.f: Inverts a matrix. It is used in the molecular equilibrium computations.

Isotope.f: This *driver* routine runs syntheses for differing isotope ratios of an element (which is part of a molecule). To access it, the *driver* keyword must be "isotop".

Jexpint.f: Computes the n'th exponential integral of variable "x". It is used in the line and continuum flux calculations.

Linlimit.f: Marks the range of lines to be considered in a particular line calculations, depending on the type of calculation (e.g. synthetic spectrum, single line curve-of-growth, etc.)

Moog.f: The main program for MOOG. It simply causes the parameter file to be read, and sends control to various *driver* subroutines.

Mydriver.f: This is a dummy *driver* routine now. It is simply a placeholder, so that the user can substitute a new *driver* routine in its place, for experimental or specialized MOOG use.

Nansi.f: A collection of routines that allow the screen to be "painted" with characters at certain screen positions, rather than having the screen simply scroll line-by-line. I haven't a clue where these routines came from, but they work. Maybe Mike Fitzpatrick (now at NOAO) gave them to me.

Nearly.f: Calculates the line opacity at line center ($\equiv$ "kapnu0") for each line. When kapnu0 is multiplied by the Voigt function, one gets the line opacity at an arbitrary wavelength. The damping parameter "a" and the Doppler factor "dopp" are also computed.

Number.f: Decodes a character string (usually created with the "Getasci.f" command) into a double precision floating point number.

Obshead.f: Decodes the header records of the observed FITS spectrum file.

Oneline.f: Computes the complete profile of a single isolated line.

OpacHelium.f: Calculates the He^- b-f and f-f opacities.

OpacHydrogen.f: Calculates the H I b-f, H I f-f, H^- b-f, and H^- f-f opacities.

Opaccouls.f: These are Coulomb interaction functions used by other opacity routines.

Opacit.f: Calculates the continuous opacities at reference or individual wavelengths, and calculates the optical depths also. The formulas have been taken from Kurucz (1970, *ATLAS* write-up).

Opacmetals.f: Calculates the Mg I, Mg II, Al I, Si I, Si II, and Fe I b-f opacities.

Opacscat.f: Calculates the opacities from scattering by H I, He I, and e^-.

Out.f: This is an output routine. Most messy formatted output statements are shunted to this routine, to leave the other routines more clean and readable.

Params.f: Reads in the desired *driver* keyword, and then various other keywords to set up the MOOG run as the user desires it. The various options are explained in another section of this write-up.

Partfn.f: Computes partition functions using data and formulae from ATLAS9. Usually the data in "Batomic.f" are employed, but for at least Nd and Th more recent partition function data have been coded into this routine.

Partnew.f: Computes partition functions from newer data that have been turned into polynomial fits with log(T).

Plotit.f: This *driver* routine re-plots a spectrum that already has been created in a preceding MOOG synthesis run. The raw spectrum needs to exist in a disk file.

Pltabun.f: This routine controls the plotting of abundances derived from equivalent width matches, and user decisions associated with these plots.

Pltcog.f: This routine controls the plotting of curves-of-growth, and user decisions associated with these plots.

Pltflux.f: This routine controls the plotting of flux curves, and user decisions associated with these plots.

Pltspec.f: This routine controls the plotting of synthetic and observed spectra, and user decisions associated with these plots.

Prinfo.f: Prints out lines of output abundance information screen, watching to ensure that the screen does not overflow its vertical limit.

Putasci.f: This routine dumps messages to the windows in hopefully a controlled manner.

Readobs.f: Reads in an observed spectra to compare it to a synthetic spectrum. The file format currently can be either a FITS file or an ASCII (x,y pair) style file.

Resynth.f: Synthesizes a section of spectrum and orders a plot to be made comparing synthetic and observed spectra. This is like routine "Synth.f", but it is only called after "Synth.f", and reruns the synthesis without asking for new input or output files; the original output files will be overwritten, and no information from the original synthesis will be preserved.

Rinteg.f: This workhorse numerical integration routine is taken from Kurucz (1970, *ATLAS* write-up).

Screenlength.c: This routine finds out the number of lines available in the text window in which MOOG is running.

Slines.f: Performs the input tasks for the "strong" line data file.

Smooth.f: Prepares synthetic spectrum data for plotting by smoothing the data in various ways, according to the user's wishes. Available options for smoothing are Gaussian, Lorentzian, rotational velocity, or combined rotation and Gaussian smoothing.

Snearly.f: Performs the same functions as "Nearly.f", but only for the lines of the strong line list.

Specplot.f: This routine does the actual labor of producing the synthetic and observed spectrum plots.

Stats.f: Does abundance statistics for multiple lines of a single species.

Strudamp.f: Calculates explicit damping parameters for strong lines that have accurately known laboratory damping parameters.

Sunder.f: Breaks up the name of a molecule into its component atoms. This is useful for molecular equilibrium, etc.

Synspec.f: Does the actual work of calculating a single synthetic spectrum

Synth.f: This *driver* routine synthesizes a section of spectrum and compares it to an observation file. It orders the execution of "Synspec.f" and associated routines.

Taukap.f: Calculates the line absorption coefficient and the line opacity at a given wavelength for all lines in the spectrum.

Total.f: Integrates over line depths to get a total equivalent width of a feature.

Trudamp.f: Calculates explicit damping parameters for lines that have accurately known laboratory damping parameters

Ucalc.f: Decodes the partition function data. The source of these data is Kurucz (1970, *ATLAS* write-up).

Vargauss.f: Prepares synthetic spectra for plotting, by smoothing the data with a Gaussian whose FWHM has been specified by the user at various wavelengths along the spectrum. This is a specialized routine originally designed to accommodate 4m "HYDRA" data.

Vmacro.f: This routine, feeding into Smooth.f, computes a radial-tangential macroturbulence profile to fold into a raw computed spectrum.

Voigt.f: This function calculates the Voigt function using the approximations found in the "New Series – Astronomy and Astrophysics" volume of Landolt-Bornstein.

Wavecalc.f: uses the wavelength coefficients of the observed spectrum file to create a wavelength scale, using appropriate polynomial formulae.

IX. MOOG Common Blocks

Most of the information is shared between routines through FORTRAN common blocks. These are contained in their own files, which are:

Atmos.com: Model atmosphere iformation, file numbers, flags, etc.

Chars.com: Character strings (these cannot be combined with numeric variables in common blocks) with titles, messages, etc.

Dummy.com: re-useable arrays for convenience in internal calculations.

Eqdata.com: Molecular equilibrium data.

Equivs.com: Not a common block, but a series of statements allowing memory locations to be used by more than one variable.

Factor.com: Information to be used in altering elemental or isotopic data for multiple syntheses.

Kappa.com: Continuous opacity data

Linex.com: Line information, parameters controlling synthesis or curve-of-growth repeated calculations, etc.

Mol.com: Mostly molecular equilibrium information.

Obspars.com: Parameters of observed (FITS) spectrum files.

Pstuff.com: Plotting arrays and parameters.

Quants.com: Atomic data.

Strong.com: Line information about declared very strong lines.

To help guide the way through the MOOG codes, on the next two pages we show two of the most important MOOG common blocks, and define some of the variables of these blocks.

Here is the common block file **Atmos.f**:

```
      reel*8        t(75), theta(75), tkev(75), tlog(75),
     .              kapref(75), tauref(75), xref(75),
     .              kaplam(75), taulam(75), pgas(75), ne(75), rho(75),
     .              nhtot(75), vturb(75), scont(75), numdens(8,2,75)
      real*8        elem(95), xabund(95), u(95,4,75)
      real*8        flux, fudge, wavref, abscale
      integer       ipr(10), iopr(10)
      integer       ntau, jtau5, iunits, itru, iraf, modelnum
      integer       nparam, nfmodel, nflines, nfslines, nfobs
      integer       nwrite, n2write, ndump, n2dump

      common/atmos/t, theta, tkev, tlog,
     .              kapref, tauref, xref,
     .              kaplam, taulam, pgas, ne, rho,
     .              nhtot, vturb, scont, numdens,
     .              elem, xabund, u,
     .              flux, fudge, wavref, abscale,
     .              ipr, iopr,
     .              ntau, jtau5, iunits, itru, iraf, modelnum,
     .              nparam, nfmodel, nflines, nfslines, nfobs,
     .              nwrite, n2write, ndump, n2dump
```

ntau: the number of model atmosphere layers ($\leq$75).

t, theta, tkev, tlog: different computational forms of temperature.

pgas, ne, rho, vturb: , gas pressure, electron number density, (gas) mass density, and microturbulent velocity of a layer.

kapref, kaplam, tauref, taulam: continuous opacities at the reference wavelength (or the Rosseland opacity) and at the wavelength of spectrum interest, and the corresponding optical depths.

nhtot, scont: the fictitious total number density of hydrogen, and the continuum source function.

numdens: the number densities of elements involved in continuous opacity calculations.

u: computed partition functions.

xabund: element abundances.

Here is the common block file **Linex.com**

```
 real*8       a(1000,75), dopp(1000,75), kapnu0(1000,75)
 real*8       gf(1000), wave1(1000), atom1(1000), e(1000,2),
.             chi(1000,3), amass(1000), charge(1000), c6(1000),
.             gf1(1000), width(1000), abundout(1000), widout(1000),
.             strength(1000), rdmass(1000)
 real*8       kapnu(75), taunu(75), cd(75), sline(75)
 real*8       d(5000), dellam(400), w(100),
.             rwtab(3000), gftab(3000)
 real*8       delta, start, stop, step,
.             oldstart, oldstop, oldstep, olddelta
 real*8       rwlow, rwhigh, rwstep, wavestep, cogatom,
.             delwave, wave, st1
 integer      lim1, lim2, mode, ndepths, ncurve, nlines,
.             lim1line, lim2line, ntabtot, group(1000),
.             iabatom, molflag

 common/linex/a, dopp, kapnu0,
.             gf, wave1, atom1, e,
.             chi, amass, charge, c6,
.             gf1, width, abundout, widout,
.             strength, rdmass,
.             kapnu, taunu, cd, sline,
.             d, dellam, w,
.             rwtab, gftab,
.             delta, start, stop, step,
.             oldstart, oldstop, oldstep, olddelta,
.             rwlow, rwhigh, rwstep, wavestep, cogatom,
.             delwave, wave, st1,
.             lim1, lim2, mode, ndepths, ncurve, nlines,
.             lim1line, lim2line, ntabtot, group,
.             iabatom, molflag
```

wave1, atom1, e, gf, c6, width: the feature wavelength, identification, excitation potential, oscillator strength, van der Waals parameter, and equivalent width (these quantities are read in).

a, dopp, kapnu0, chi, amass, charge: feature damping parameter, Doppler factor, line opacity at line center, ionization or dissociation energy, atomic or molecular mass, species charge (these quantities are calculated).

gf1, abunout, widout: oscillator strength, abundance, equivalent width (these are output quantities for some *drivers*).

kapnu, taunu, cd, sline: line opacity and optical depth, line contribution function, line source function.

X. A Sample Makefile to Compile and link MOOG

This set of "make" instructions works on UT Austin Astronomy Department SUN Ultra 10 and Sparcstations running under Solaris (SYSV unix). Undoubtedly you will need to alter the line that begins with "f77" even if you have a SPARCstation (your library path links will surely be different than those set up on my machine).

```
#     makefile for MOOG with all of the common block assignments;
#     this is for SUN solaris machines

#     here are the object files
OBJECTS = Abfind.o Abunplot.o Batom.o Begin.redhat.o Blends.o Bmolec.o \
          Calmod.o Cdcalc.o Chabund.o Cog.o Cogplot.o Cogsyn.o \
          Curve.o Defcolor.o Discov.o Doflux.o Ewfind.o Eqlib.o Fakeline.o \
          Findtic.o Finish.o Fluxplot.o Getasci.o Getcount.o Infile.o \
          Inlines.o Inmodel.o Invert.o Isotope.o Jexpint.o Linlimit.o \
          Moog.o Mydriver.o Nansi.o Nearly.o Number.o Obshead.o \
          Oneline.o Opaccouls.o OpacHelium.o OpacHydrogen.o Opacit.o \
          Opacmetals.o Opacscat.o Out.o Params.o Partfn.o Partnew.o \
          Plotit.o Pltabun.o Pltcog.o Pltflux.o Pltspec.o Prinfo.o \
          Putasci.o Readobs.o Resynth.o Rinteg.o Slines.o Smooth.o \
          Snearly.o Specplot.o Stats.o Strudamp.o Sunder.o Synspec.o \
          Synth.o Taukap.o Total.o Trudamp.o Ucalc.o Vargauss.o \
          Vmacro.o Voigt.o Wavecalc.o

#     here are the common files
COMMON =  Atmos.com Chars.com Dummy.com Factor.com Kappa.com Linex.com \
          Mol.com Obspars.com Pstuff.com Quants.com Strong.com Equivs.com

CC = cc
FC = f77 -w

# the following lines point to some needed libraries
X11LIB = /opt/local/X11R6/lib

#        here are the compilation and linking commands
all: MOOG ;

MOOG:  $(OBJECTS);
$(FC) $(OBJECTS) -o MOOG -R$(X11LIB) -L$(X11LIB) -lX11 \
        -lsocket -lnls -ldl -Bstatic -L/opt/local/lib -lplotsub \
        -ldevices -lutils -L/usr/ucblib -lucb

$(OBJECTS): $(COMMON)

clean:
-rm -f *.o MOOG libMOOG.a
```