The 0^{th} order introduction to makedx and running grids of models

Agnès Kim, Ph.D.

Georgia College & State University

Fall 2010

Table of Contents

1 Who is this guide for and what does it hope to achieve?

This guide is written for undergraduate students, but really is meant for anyone who wishes to get started running grids of white dwarf models with makedx and associated tools. makedx refers to any version of makewd (makeda, makedb, makedk, …). This is a fairly bare bone guide. It assumes that the user:

- has some knowledge of stellar structure and non-radial oscillations in stars
- knows basic unix commands
- knows how to open and edit text files with odd extensions or none at all and how to search and navigate them
- is literate in computer programming
- knows how to use some kind of graphing software to make plots from data files (only necessary for exercises but a must-have skill for research)

If you do not have all that background, do not despair. Ask questions as you go along to fill in any gaps or develop any lacking skills if you get stuck before coming back to this guide.

It is expected that you will run into problems, that things won't work out as they're supposed to, and that some of the information you find here is inaccurate or outdated (I will be elated if you fix the mistakes, though keeping this up to date is a losing battle). Think of this guide as a starting point. If you run into problems, ask an expert.

Most of the guide is meant to be tried as you follow along. To provide further hands on activities, I suggest some exercises throughout the guide. The best way to learn how to use a tool is to actually use it.

2 What makedx does

Makedx makes a model of a white dwarf, calculates its modes of vibrations (periods), compares these calculated periods with a list of observed periods, and calculates the root mean square difference between the two lists (the goodness of fit).

What does the above mean? "making a model of a white dwarf" means that given some input parameters provided by the user (the surface temperature, the mass, and structure parameters to be discussed in section 3.3.3), makedx numerically solves differential equations and comes up with values for key quantities at each point in the model. Some key quantities include the temperature, density, pressure, and many more. All these quantities are written out to a single, large text file called "evolved".

"Calculating the modes of vibrations" means once makedx knows all the key quantities in the model white dwarf, it goes on to solve numerical equations of (non-radial) stellar pulsations and comes up with a list of frequencies (written out as periods) that the model would vibrate at if vibrations were set off.

The “observed periods” are provided by the user in the text file wd40.dat. Usually, we want to study some real life white dwarf and the list comes from the literature or colleagues who have observed a pulsating white dwarf and processed the data to arrive at a list of observed periods.

“Calculating the root mean square difference” means that makedx pairs up each observed period with the calculated period that is closest to it and adds up the residuals (essentially calculating a standard deviation). This results in a single number, called “goodness of fit”. The smaller the number, the closer the two lists match and so the better the model. In makedx, that number is called ff and is returned by the function ff_dx.f (which actually does all the work). As a rule of thumb, a goodness of fit smaller than 2 indicates a good fit.

3 Getting started

3.1 Packing list

The pristine makedx folder contains the following files:

```
AUXIN5
EEOSC
EEOSH
EEOSHE
IEOSC
IEOSO
SQOPAC
makegrid
```

And folders:

```
src
masses
```

The capital letter files are table of numbers makedx uses in its calculations. The EOS files are equation of state tables, which list temperatures, density and pressure relations. The SQOPAC files lists opacities, which dictate how energy flows through the model. These tables are the result of separate calculations. It helps speed up the computation of the models, as the code looks up numbers and interpolates instead of solving the equations that led to these tables each time.

makegrid is a shell script used when one wants to compute an entire grid of models as opposed to just one. We discuss makegrid in section 6.

The folder “masses” contains input files makedx uses as a starting point. Any numerical computation requires an initial guess. The files in “masses” are such initial guesses and give guess values makedx needs as input in order to get started. There is one file for each different white dwarf masses required. The files in the folder “masses” are named e.g. M=0.450, meaning this is an input file for a 0.450 solar mass white dwarf. Most white dwarfs in nature have a mass of around 0.6 solar mass, but they can be as

low as 0.450 solar mass or as high as 1.4 solar mass. In our line of work, we rarely have to go over 0.900 solar mass. Things get numerically and physically touchy at higher masses.

The folder “src” stands for “source” and contains the source files for makedx. This is where we begin.

3.2 Compiling makedx

At minimum, the src folder contains the following files:

```
ff_dx.f
getpar.f
Makefile
wd40.dat
```

ff_dx.f is the fortran function that creates the models, make them vibrate, compute periods, compares the calculated periods with the observed periods, and calculates the goodness of fit. It is called by the main program getpar.f.

While it is the main program, getpar.f doesn’t do much. It reads off parameters provided on the command line when the program is run, passes these off to ff_dx.f, and assigns the goodness of fit calculated by ff_dx.f to the variable called ff.

Makefile compiles the program. The contents of Makefile look like the following:

```
FFLAGS = -132 -r8 -save -zero
F77 = ifort

makedk:    getpar.f ff_dk.f
           $(F77) $(FFLAGS) -o ../$@ $^
```

There is likely also a bunch of lines that start with #. These lines are ignored when Makefile is run so we shall ignore them as well.

The first line defines the flags we want to use in compiling makedx.

-132 indicates that the compiler should expect individuals lines of code to be as long as 132 characters (the default is 80 but that chops off some of the lines in ff_dx.f, which confuses the compiler to no end).

-r8 sets the default size of real numbers to 8 bites.

-save saves all the variables

-zero initializes all variables to zero.

The second line defines which compiler to use. Here, ifort.

The third and fourth line use these two definitions and puts everything together into a compiling command.

It instructs the compiler to make the executable “makedk” by linking together the source files getpar.f and ff_dk.f, using the compiler ifort with the options defined on the first line, and place the executable one directory up.

To compile, type the command (at the prompt $):

```
$ make makedk
```

If all goes well, you get the output:

```
ifort -132 -r8 -save -zero -o ../makedb getpar.f ff_dk.f

ff_dk.f(4377): warning #6371: A jump into a block from outside the
block may have occurred.   [45]

          go to 45

----------------^
```

And after 10-20 seconds, a new prompt.

The first line of output is the command the Makefile put together for you when you ran “make makedk”. It has the name of the compiler (ifort, a binary), the flags set in Makefile, the –o for “create an executable”, “`../makedb`” gives the name of the executable (makedb) and places it up one directory (that’s what the ../ does). Finally, the source files to link and compile are listed.

The lines that follow look bad, but we are just elated that we are getting away with just a warning. It would take work to get rid of that warning and so far, it has not hindered the code from running.

Instead of typing “make makedk” at the prompt, you could also enter the entire command line put together by the Makefile. It’s just that it’s quicker to type “make makedk” and it also makes it easier to modify options and source files in the Makefile.

If compiling was successful, the executable “makedk” now appears in the makedx directory.

3.3 Running makedx

3.3.1 Setting up the input

If it’s not in the makedk directory yet, copy wd40.dat from the src directory. The contents of the wd40.dat text file look like this:

```
Periods
Pmin    Pmax    lmax
50.     1500.   2
number of observed periods to match
8
observed periods to match
195.0
```

```
204.6
256.9
281.0
333.5
350.6
525.4
539.8
```

The number labeled “Pmin”, here 50, is the lowest period that should be computed. “Pmax” is the highest period that should be computed. 50 and 1500 about cover it, as the lowest possible period of vibration is around 100 seconds and we don’t observe periods over 1500 seconds. It is good to keep the range as wide as possible, as one never knows what set of observed periods we want to fit. The only advantage in reducing the range is that it speeds up the computations by making the output files smaller (fewer periods to list).

The number “lmax” is the maximum l value we are interested in for periods. Leave it as 2.

The next line is self-explanatory and important. The “8” tells the code that the list that follows has 8 periods in it.

The file ends with a list of periods to match.

3.3.2 Running command

To run makedx type e.g.

$ ./makedk 29500 520 216 602 99 32

Makedx expects 6 parameters following the run command ./makedk. The parameters are the surface temperature, the mass of the white dwarf wanted, followed by 4 parameters that have to do with the internal chemical composition of the star, detailed in the following subsection.

3.3.3 Input parameters

Surface temperature

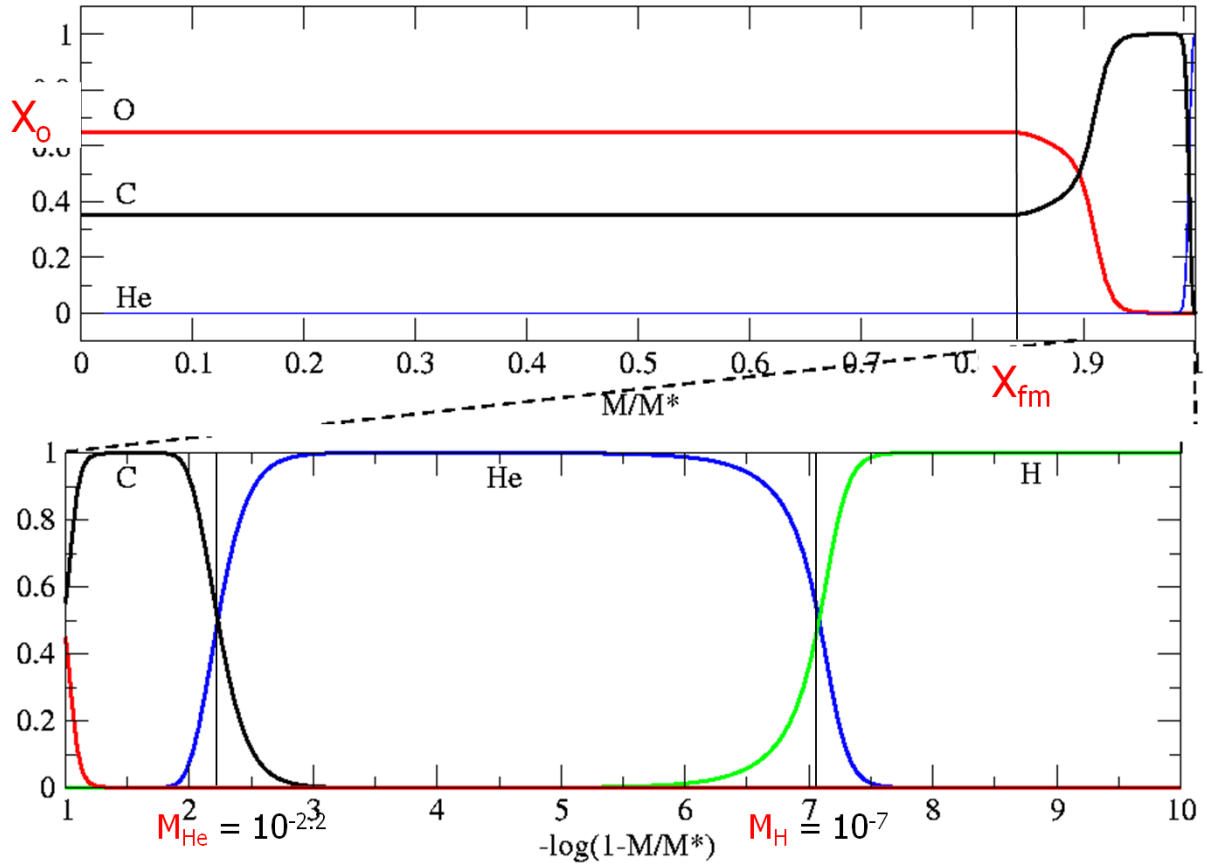
Called “effective temperature” by astronomers. It is expressed in degrees Kelvin. For pulsating white dwarfs, we are typically interested in two temperature ranges: 10,000 K – 13,000 K and 20,000 – 30,000 K.

Stellar mass

The mass of the white dwarf, in solar masses (e.g. M = 1.00 for a white dwarf that has the same mass as the Sun, M = 0.50 for one with half the mass of the Sun). Because of the limitations of doing arithmetic with unix shells (can only be done with integers), stellar mass has been scaled up by a factor of 1000 so that “520” really means 0.52.

Chemical composition parameters.

Loosely speaking, a white dwarf is a ball of carbon/oxygen surrounded by a thin, pure layer of Helium and sometimes surrounded by a thin layer of hydrogen. We describe that chemical structure with 4 parameters: M_{He}, M_H, X_o, and X_{fm} (sometimes also called q_{fm}). Typical interior composition profiles are illustrated below.



The top panel represents the core of the white dwarf and the bottom panel the outer 10% of the white dwarf (by mass). The x coordinate used in the bottom panel, distinct from the straight mass coordinate used for the top panel is design to expand the outer layers of the star for clarity. On each, deeper in the model is on the left, while to the right, we are moving toward the surface.

Each curve represents the relative abundance of an element (oxygen, carbon, helium and hydrogen). Inside the star, we start out with 65% oxygen and 35% carbon. The central oxygen abundance, denoted by X_o, is one of our structure parameters. We have a constant mix of the two out to mass coordinate X_{fm} = 0.84 M_* (another structure parameter). This part of the star is called the “homogeneous core”.

On top of the homogeneous carbon and oxygen core, there is a layer of pure carbon (the abundance of carbon rises to 1 or 100%). On top of the pure layer of carbon, is a pure layer of helium. The point where the helium abundance becomes greater than the carbon abundance is our fourth structure parameter,

M_{He}. With our outer layer special mass coordinate, M_{He} is here equal to $10^{-2.2}$. This means that $1 - 10^{-2.2}$ = 0.99369 = 99.369% of the star by mass is inside that shell and 1-0.99369 = 0.63% is outside that shell.

When we run makedx, we would enter this M_{He} as 220 (take the log, drop the negative sign, and multiply by 100). The last parameter, M_H, marks the location of the base of the pure hydrogen layer and is defined in a similar manner. In that specific example, we would enter it as "700".

Exercises – some code reverse engineering

While tedious, code reverse engineering is a useful skill to have, as it allows you to determine exactly what kind of input an under documented, academic code is expecting and what the unlabeled output is.

Look at the source code in getpar.f and ff_dx.f and determine how the input parameters given on the command line are scaled and passed to the subroutine evolve.

Look at the source code in ff_dx.f and look at how the input data is read from wd40.dat (it's the first thing in the function after all the variable declarations).

3.4 Output from makedx

Makedx puts out information to the screen and to two files (evolved and param.log)

To the screen, makedx puts out a list of calculated periods (one series for l=1 and one for l=2) and the value of the goodness of fit.

The file param.log is human eyes friendly and short. It may look like the following:

```
T = 28950.  M = 0520  log(Mhe) = -2.16  log(Mh) = - 6.02
 1 evolving
   14.0000000000000      -7.888500000000000E-003
 2 pulsating
   195.000000000000        192.794722626343               2
   204.600000000000        204.622306007529               1
   256.900000000000        257.251025036508               2
   281.000000000000        279.549531208984               1
   333.500000000000        331.252066175139               2
   350.600000000000        350.742913044854               2
   525.400000000000        523.042512943955               2
   539.800000000000        541.702735407401               2
```

It informs us of what the first 4 parameters we gave makedx to run where (if I weren't so lazy, it would tell all 6), lets us know how the run went and what the results are.

The mysterious line below "evolving" tells us two numbers that affect the run numerically. The first one is a time step (changing it can help the code run more smoothly and sometimes get it out of a numerical bind). The second one is an artifact of how the code is designed and if chosen wrongly, can yield to completely meaningless results (if the code runs at all). It is best to leave it as it is. Get worried if it is much smaller than -.0132 (=log(.97)).

The list below “pulsating” is informative. It shows the match between the observed periods we listed in wd40.dat (1st column) and the periods of the model. Param.log is a nice file to look at to check things.

The file “evolved” is not human eyes friendly, but immensely important. It is the actual model ff_dx.f computed that when pulsated, gives the list of periods found. It has, in a raw format, all the information about the model: temperature, pressure, density, luminosity, neutrino rates, thermal timescales, …. It is useful if one wants to check how sound the model is and also to present results, with further processing described in section 4.

Exercises

Look at the source code in ff_dx.f and find how the goodness of fit parameter is calculated (hint: it’s called ff and is calculated in the main body of ff_dx.f). Write down the formula on a sheet of paper so it’s easy for humans to read. Identify all quantities.

Match the periods that are printed to the screen to the periods listed in wd40.dat best you can. Write your best match down. Using the formula you wrote down in the previous exercise, calculate a fitness parameter. Look at param.log and see how makedx did the matching and what the goodness of fit is. Do you agree?

4 Looking at the models using the prep code

The prep code may be found in the folder wdcode/prep/. It is called “prep” because originally, it was used to prepare the models for calculating periods (by refining them). It is useful because it also produces output files that may be used to make plots.

4.1 Compiling the prep code

The source code for the prep code may be found in wdcode/prep/source. To compile, type ($ is the prompt, don’t type that):

$ make prpwdxdp

This produces the executable prpwdxdp and puts up one directory.

4.2 Running the prep code

Go back to the wdcode/prep/ directory. Copy the “evolve” output file produced by ff_dx.f into that directory. First we need to edit the “evolve” file to give it the format prpwdxdp is expecting. This could easily be done by modifying ff_dx.f so that it outputs a ready to use “evolve” output file, but again, lazyness go the better of me.

We need to change the second line of evolved:

```
  29  8.2670E+07 23.205 7.719 6.516  8.965 4.424729  -1.1084 -10.0000  0.000
  422  =   149  +   273
 1.208166016059208E+07 1.304555016051031E+07 1.408648203574435E+07 1.521062189271040E+07
 1.642463531242949E+07 1.773572880165290E+07 1.915169500476571E+07 2.068096208684868E+07
 2.233264776020736E+07 2.411661850107072E+07 2.604360235828949E+07 2.812513915948372E+07
 3.037376724438211E+07 3.280307961215113E+07 3.542782215302935E+07 3.826400343710256E+07
 4.132901737052814E+07 4.464178157302295E+07 4.822289463084931E+07 5.209518939759207E+07
 5.628299819391551E+07 6.081321537932174E+07 6.571533631351838E+07 7.102178776783712E+07
 7.676831922267552E+07 8.299447026848325E+07 8.974413397220254E+07 9.706624237769610E+07
 1.050187994819890E+08 1.136620040841851E+08 1.230665141110836E+08 1.333131680094751E+08

Etc…
```

Add a bunch of zeroes at the end of the second line (I believe prpwdxdp is expecting 6 numbers there, but to be safe, put more).

```
  29  8.2670E+07 23.205 7.719 6.516  8.965 4.424729  -1.1084 -10.0000  0.000
  422  =   149  +   273 0 0 0 0 0 0 0 0 0 0 0 0
 1.208166016059208E+07 1.304555016051031E+07 1.408648203574435E+07 1.521062189271040E+07
 1.642463531242949E+07 1.773572880165290E+07 1.915169500476571E+07 2.068096208684868E+07
 2.233264776020736E+07 2.411661850107072E+07 2.604360235828949E+07 2.812513915948372E+07
 3.037376724438211E+07 3.280307961215113E+07 3.542782215302935E+07 3.826400343710256E+07
 4.132901737052814E+07 4.464178157302295E+07 4.822289463084931E+07 5.209518939759207E+07
 5.628299819391551E+07 6.081321537932174E+07 6.571533631351838E+07 7.102178776783712E+07
 7.676831922267552E+07 8.299447026848325E+07 8.974413397220254E+07 9.706624237769610E+07
 1.050187994819890E+08 1.136620040841851E+08 1.230665141110836E+08 1.333131680094751E+08

Etc…
```

Now run the prep code by typing
$ ./prpwdxdp < evolved

This feeds evolved as an input file into the prep code. Some unintelligible output appears on the screen:

```
torfreq0 pertor=  2.234742503518114E-002   44.7478847529735
   64.3141186740880        115.185833473065        4.39882612792223
   35682403.8116800
```

Many output files are also created (you can view the list by typing “ls” at the prompt):

```
chirt.dat     dp_xtal.dat   lrad.dat      prop.dat            tape18.dat    temprp.dat
corsico.dat   epsrt.dat     modelp.dat    prop2.dat           tape19.dat    thorne.dat
cpvtga1.dat   gam_xtal.dat  output.dat    reflection.dat      tape28.dat    xhe.dat
deld.dat      kaprt.dat     pform.dat     struc.dat           tape29.dat
```

Most are columns of number ready to be fed into the plotting program of your choice. The quantities each contains are detailed in the appendix.

Exercises

Look at the source code in prpwdxd.f and check that the quantities written out to the files are as given in the appendix (I guarantee there are mistakes). Determine how bvfreqmag is calculated.

Run the prep code on your output file "evolve". Plot the following:

- The oxygen mass fraction, carbon mass fraction, helium mass fraction, and hydrogen mass fraction. Use partt as the x coordinate.
- The log of the temperature. Use mr as the horizontal axis. Check that the model does have a surface temperature of 11500 K. What is the central temperature?
- The density. Use fr as the horizontal axis.
- The mass (mr) vs fr. Check that the model has a mass of 0.600 M_{Sun}.

5 Further pulsation analysis with the pulse code

If, in addition to the l identification, the k identification of modes or other pulsation information is required, you will have to run the model through the pulsation code.

5.1 Compiling the pulsation code

The source code for the prep code may be found in wdcode/pulse. To compile, type ($ is the prompt, don't type that):

$ make cjhanro

This produces the executable cjhanro.

5.2 Running the pulsation code

To run, cjhanro needs several input files:

```
period.dat
modelp.dat
tape28.dat
tape29.dat
```

period.dat should already be present in the directory. It looks something like this:

```
600502 2
 100. 1200. 1000 2
 0
```

If it is not present, create one and write in the above numbers in it.

The numbers on the first line are some code used for bookkeeping. Don't worry about them. Most of the time, you will want to modify the second line.

The first two numbers on the second line are the Pmin and Pmax we had in our wd40.dat file. The next number (1000) you will want to keep as is. It has to do with the number of period guesses that are made

in order to find the right ones. The higher the number, the more precise the calculations but also the slower. 1000 is a good number. The last number (2) is the lmax we had in wd40.dat.

Don't worry about the 0 on the 3rd line. Leave it.

modelp.dat, tape28.dat, and tape29.dat are output files from the prep code. So once you run the "evolve" file through the prep code, copy these three output files over to the pulse directory. Run cjhanro by typing

$ ./cjhanro

The only output is the output files. There is no output to the screen. After a few seconds, you get a new prompt, signifying cjhanro is done.

5.3 Output from cjhanro

The main file we are interested in is results.dat. And within results.dat, we are mainly going to be interested in the first 3 columns. There is also lots of other numbers cjhanro produces that may be used for more advanced pulsation analysis.

6 Running a grid of models with makegrid

makegrid is a simple script that uses nested loops to run the set of parameters of your choice. It steps from a minimum to a maximum with given steps, for all 6 parameters.

6.1 Ensuring makegrid is an executable

makegrid is a script, so it does not need to be compiled. However, it needs to be an executable. To check, type:

$ ls -l makegrid

The output of such a command should be something like:

-rwxr-xr-x 1 agnes users 1346 Nov 2 2007 makegrid

If you don't have x's, then it means it's not an executable. Another sign makegrid is not an executable is if you try to run and it tells you something like "command not found". To make the script executable (if necessary) type:

$ chmod +x makegrid

6.2 Setting up makegrid

Some rules of good programming practices were followed in writing makegrid, so most of the time, you will only have to modify quantities near the top of the file. Namely the following lines (yours may look slightly different – this is the C shell version):

```
@ teff0 = 26400
@ teffmax = 26800
@ step1 = 400
@ teff = $teff0

@ mass0 = 450
@ massmax = 460
@ step2 = 10
@ mass = $mass0

@ Mhe0 = 400
@ Mhemax = 440
@ step3 = 40
@ Mhe = $Mhe0

@ Mh0 = $Mhe0 + 200
@ Mhmax = 840
@ step4 = 40
@ Mh = $Mh0

@ Xc0 = 50
@ Xcmax = 110
@ step5 = 10
@ Xc = $Xc0

@ Xfm0 = 35
@ Xfmmax = 95
@ step6 = 10
@ Xfm = $Xfm0
```

There is a block for each of the 6 parameters. In each case, you provide a starting value, a maximum value, and a step size. The rest of the script uses nested loops to run all combinations of parameters within the given ranges. C shells can only do math with integers, so all parameters have been scaled to allow us to cover the parameter space we want using only integers.

The script checks to make sure that M_h and M_{He} are separated by at least 2 dex. makedx doesn't work properly if the pure helium layer is too thin.

Exercise

Consider a grid of models with the following specifications (bounds are inclusive):

Parameter	Starting value	Maximum value	Step size
Temperature	11200	11300	100
Mass	0.600	0.610	0.010
M_{He}	$10^{-2.4}$	10^{-2}	0.2
M_H	10^{-4}	$10^{-4.2}$	0.2
X_o	0.80	0.85	0.05
X_{fm}	0.60	0.64	0.02

How many models is that going to be (watch the constraint between M_{He} and M_H).

Edit makegrid so that it is set up to run this grid.

6.3 Output from makegrid

The main body of makegrid looks like the following:

```
echo make1db $teff $mass $Mhe $Mh $Xo $Xfm
echo $teff $mass $Mhe $Mh $Xo $Xfm >> calcperiods
make1db $teff $mass $Mhe $Mh $Xo $Xfm >> calcperiods
echo 100000 >> calcperiods
```

Let's say at that stage, $teff = 11200, $mass = 600, $Mhe = 200, $Mh = 400, $Xo = 80 and $Xfm = 60.

Line 1 prints out the screen "make1db 11200 600 200 400 80 60".

Line 2 prints out the 6 parameters to a file called "calcperiods".
Line 3 appends to that file the output of running the command "make1db 11200 600 200 400 80 60". That's a list of calculated periods and the goodness of fit, that normally get written to the screen.

Line 4 appends to "calcperiods" the integer 100000, at the end of the period list. That number was chosen somewhat arbitrarily but carefully to serve as a separator for the period matching program I introduce in the next section.

Exercise

Run the grid you set up in the previous section. Examine the output file calcperiods. How many models were computed?

7 Analyzing the grids of models

makegrid puts at our disposal one large file with period listing for thousands of models. While this includes the goodness of fit for a particular set of observed periods that were provided in the wd40.dat input file used at the time the grid was built, we do not need to limit ourselves to fitting only that list of periods. We can compare other lists of observed periods and try to see if there is a model on the grid that fits these observed periods well.

The code that fills that purpose is fitper.

7.1 fitper packing list

The needed source files are:

```
main.f90
fit.subroutine.f90
Makefile
```

And the input files

```
calcperiods
obsperiods
```

main.f90 is the main program. It's short and sweet and I think beautifully written (it's got comments on top that explain what each of the input variables are!)

All the work is done by the subroutine fit.subroutine.f90. That one's got more clutter. There is a description of what it does and how it works (in rough terms) in an appendix of my dissertation, but the program has evolved since. I am particularly proud of having added the ability to pick which l to assign to each observed mode.

To compile, type

```
$ make fitper
```

This makes an executable called fitper. fitper requires the calcperiods file that makegrid produced and an input file called obsperiods. obsperiods looks something like this:

```
nobs  iflag       numpar      siglim      fitlim
15    0           3           3.          10.
812.8 1
768.9 1
721.9 1
684.2 1
645.9 1
623.5 2
612.0 1
581.5 2
563.3 1
537.7 2
528.9 2
459.2 1
433.9 0
399.2 0
372.3 0
```

The first line is a header. The second line lists a number of input parameters for the program.

- nobs is the number of observed periods in the list that follows.
- iflag is obsolete. Leave it as 0.
- numpar gives the program the number of parameters that were used in the models. In our case, that's 6. numpar is not crucial, it is only used in calculating a special fitness number at the end.
- siglim (3. seconds in this case) tells the program to only write out fits that have a goodness of fit better than that number. If you want to see all of the fits, you can set it to something ridiculous like 1.e6. Once you know you have good fits, then 3. is a good number. If you don't get any output when your run fitper, try upping siglim.
- A fitlim of 10. tells the program to label and tally any single period on the list that fits to better than 10. seconds. It is another measure of how good a fit is. A standard deviation alone doesn't give all the information. It can be small if 9/10 periods are spot on, but the last one may be completely off and we'd like to know that.

The rest of the flag lists the periods we want to fit. Each period is followed by a flag that may be 0, 1 or 2. The flag is set to 0 if we have no opinion on whether that period should be an l=1 or an l=2 mode. If you want to force that period to be identified as an l=1 mode, then set the flag to 1. For an l=2 mode, set it to 2.

With the two necessary input files, run fitper by typing:

$ ./fitper

7.2 fitper output

fitper produces 2 output files. One is fitnesspars.dat and the other is periods.dat.

7.2.1 fitnesspars.dat

The first few lines of fitnesspars.dat may look like this:

```
10600  520   200   400   100   60    2.924867      1.997112      8     1.00   1.00
10600  530   200   400   50    60    2.997224      2.021663      8     1.00   1.00
10600  530   200   400   60    60    2.908120      1.991386      7     1.00   1.00
10600  530   200   400   70    60    2.870731      1.978543      7     1.00   1.00
10600  530   200   400   80    60    2.868462      1.977761      8     1.00   1.00
10600  530   200   400   90    60    2.996939      2.021568      8     1.00   1.00
10600  530   200   400   100   60    2.848874      1.970997      8     1.00   1.00
10600  540   200   520   70    80    1000.000000   1000.000000   0     0.00   0.00
```

The first 6 columns list the parameters of each model. The 7th column is the raw standard deviation (the same goodness of fit ff_dx.f would calculate). The 8th column is a statistic that takes into account the number of observed periods and parameters we have. The fewer observed periods, the easier it is to find a better match. The more parameters, the easier it is to find a match. That different measure of goodness of fit attempts to compensate for these effects.

The 9th column is the number of periods that fit to better than fitlim. The last two columns are normalization factors that compensate for the fact that some lists of calculated in calcperiods are longer than other (hence again, making it easier to find a fit for each observed period). In this run, the 2 normalization factors were set to 1. In a small patch of parameter space, it is a fair thing to do.

The last model on the sample list listed above is an example of a model that did not fit. If the fit is too poor, fitper gives up, assigns 1000. to all goodness of fit parameters and moves on to the next model.

Exercise

Examine fit.subroutine.f90 to determine how the two goodness of fit parameters are calculated.

Run the calcperiods file from your makegrid run through fitper (make up a small list of observed periods using a list from calperiods as a guide). Play around. You can even pick a perfect subset of periods from one of the models. Does fitper find it, and does it assign it a 0 goodness of fit?

Sort the results by goodness of fit.

Subselect the models that have the same mass and structure parameters (pick a set) so they differ only by temperature (you should have 3 models).

For that subset, plot the goodness of fit vs the temperature (you will get 3 points).

7.2.2 periods.dat

periods.dat lists period matches. The first few lines may look like this:

```
10600 520   200   400   100   60    2.924867    1.997112    0     1.00  1.00
   372.3   372.8     0.5   1   3
   563.3   556.9    -6.4   1   6
   645.9   645.5    -0.4   1   8
   684.2   684.3     0.1   1   9
   721.9   728.5     6.6   1  10
   528.9   520.6    -8.3   2   8
   581.5   583.6     2.1   2  10
   623.5   614.4    -9.1   2  11
100000
10600 530   200   400   50    60    2.997224    2.021663    0     1.00  1.00
   372.3   375.4     3.1   1   3
   563.3   554.9    -8.4   1   6
   645.9   638.3    -7.6   1   8
   684.2   685.1     0.9   1   9
   721.9   730.8     8.9   1  10
   433.9   440.3     6.4   2   5
   528.9   519.5    -9.4   2   8
   581.5   582.0     0.5   2  10
 100000
```

This includes the first two models of the grid and how they compare with the observed periods. For each model, the first line is a repeat of what appears in fitnesspars.dat. This is followed by a list of how the observed periods matched up with calculated periods.

The first column lists the observed periods. The second column lists the best fit period found among the calculated periods in the file calcperiods. The 3rd column is just the calculated period minus the observed period (column 2 minus column 1). The 4th column is the l identification of the mode. The last column is an index number (how far down the list in calcper the period was found). It's not a very useful quantity and certainly has no physical meaning.

The list of periods is followed by a separator (100000).

Exercise

Try fitting the observed periods in obsperiods using different l constraints on the mode(s) of your choice. check that fitper respected that constraint by examining the output in periods.dat.

8 The end

And that's it. I hope some of this made sense.

Appendix – prep code output

Variables and their (not always helpful) meaning	
n	Shell number
r	Radius coordinate (cm)
rstar	Radius of the star (cm)
mr	Mass coordinate. Denotes how much mass is inside the current shell (in grams – not kidding)
fr	r/rstar
mstar	Total mass of the star (g)
partt	Mass coordinate that emphasizes the core = log(1 – Mr/M*) (negative of commonly called q)
g	Acceleration due to gravity at r (cm/s^2)
lr	Luminosity at r (ergs/s)
t	Temperature at r (Kelvins)
rho	Density at r (g/cm^3)
p	Pressure at r ($dynes/cm^3$)
xi	ln(r/p)
eps	Neutrino energy loss rate (ergs/s/g)
cv	Heat capacity at constant volume (erg/K)
chr	A thermodynamics gradient
cht	A thermodynamics gradient
epsr	A thermodynamics gradient
epst	A thermodynamics gradient
kapr	A thermodynamics gradient
kapt	A thermodynamics gradient
del	A thermodynamics gradient that has to do with convection
delad	A thermodynamics gradient that has to do with convection
gam1	A thermodynamics gradient
gam3	A thermodynamics gradient
cv	A thermodynamics gradient
cp	A thermodynamics gradient
kap	Opacity (cm^2/g) Has to do with how easily photons make their way through
bled	"B Ledoux". Accounts for composition change contributions to the Brunt-Vaisala frequency
bvfreq	Brunt-Vaisala frequency – a key quantity that determines the periods
bvfrq	Brunt-Vaisala frequency without the Ledoux bumps in it (just the continuum)
bvfreqmag	Some scaled version of bvfreq
xo	Oxygen mass fraction
xc	Carbon mass fraction
xhe	Helium mass fraction
xh	Hydrogen mass fraction
acous	Acoustic frequency (has to do with how fast sound propagates)
tfreq	Not sure what that is
tth	Thermal timescale at r

File contents (partial list, just the more useful ones)

chirt.dat		
partt	kapr	kapt

corsico.dat (note: the person who designed this output file thoughtfully put in headers)						
partt	xo	xc	xhe	xh	bled	bvfreq

cpvtga1.dat				
partt	log(cv)	log(cp)	gam1	fr

deld.dat		
partt	del	delad

epsrt.dat				
mr/mstar	epsr	epst	log(eps)	fr

kaprt.dat								
partt	chr	cht	log(kap)	fr	xi	1/delad	log(tth)	gam3-1

lrad.dat			
partt	log(lr)	log(tth)	xi

prop.dat						
partt	acous	bvfreq	bvfrq	fr	xi	treq

temprp.dat						
partt	log(t)	log(rho)	log(p)	fr	xi	log(lr)

struc.dat								
n	partt	bvfreq	log(tth)	log(lr)	bvfrq	bvfreqmag	mr/mstar	xi

xhe.dat							
partt	xhe	bled	xo	xc	fr	xh	log(r/p)

tabe28.dat				
xi	r	g	rho	mr

tape29.dat
Important, but not useful for making plots.