WDEC Reloaded – User Manual

Agnes Kim, Ph.D.
axk55@psu.edu

Faculty Member, Penn State Scranton

1. Message from the author

I hope the present documentation helps you get started using the code. While this documentation is minimal, I added a section near the end to encourage you to look at the source code and attempt to tell you how to use the source code to figure out what is what. It can also be useful if you are looking to make your own modifications to the code. I welcome email enquiries and will do my best to assist. I am first and foremost an Astronomer and not a developer so please pardon my coding.

I tried to structure the document to address the most pressing questions first. So I begin with a section on what you can do with this code. The next section looks boring (history), but it is important to read, as it explains some of the arcane vocabulary and structure you may encounter while using the code. Then I dig into the essentials: what the files are, how to compile the code, how to run it, what the output is. The next section gives an overview of the source code, followed by a section on the limitations. The last section is a troubleshooting section. It has figures. If you get weird output, see if your output matches one of the figures. Under each figure, I tell how to fix the issue shown.

Before you use the code to do science, please read the section on limitations. If the code crashes on you (which I am sure it will upon first use) read the limitations.

Finally, if you want more details on what the code does and how, see the software paper Bischoff-Kim et al. 2017.

Enjoy.

[Update March 7, 2019] Evidently, this had to happen. A second version of the code, with more free parameters is now available, for asteroseismic fitting purposes. If you want to reproduce a model from your favorite stellar evolution code, you want to use v15. v15 allows direct input of an oxygen profile, which you can take from the model of your choice. If you want to use the code for asteroseismic fitting, then use v16. You can vary *up to* 15 parameters. Both are useful versions of the code and so I had to post both. Check out my github repository for asteroseimic fitting tools if that's what interests you. You sould be aware that v16 only fits C/O profile white dwarfs (whether they are DAs or Dbs).

[Update May 26, 2023] I am so excited to introduce three improvements to the code. 1) v20 interfaces with a newer version of MESA for the equations of state and opacities. 2) Improvements in the parameterization of the oxygen abundance profile allows helium a little deeper in the core, and oxygen further out into the envelope. This allows to reproduce most oxygen abundance profiles from stellar evolution. By using the “mkgrid” utilities and setting flags in the “inputparameters” file, users can default easily to profiles close to that of the La Plata group (who helpfully post their models online http://evolgroup.fcaglp.unlp.edu.ar/TRACKS/tracks.html). 3) The ability to make helium core white dwarfs. Since v20 supersedes v16, v16 was retired from Github.

2. Use of the code

WDEC stands for White Dwarf Evolution Code. Unlike what the name suggests, this is not a stellar evolution code. The advantage of the WDEC over a stellar evolution code is that it runs MUCH faster, and is less of an art to use. If you are looking to construct a quick white dwarf and get its periods of pulsation, this is a good code to use. A computer savvy undergraduate student can learn how to use this code and process its output over the course of a semester.

The code does have its limitations, which I defer to the end of this document. That's not because it's not important, but because I think the reader's first interest may be in finding out how to compile and run the code (but please, DO read that last section).

For v15 of the code, you will have to provide your own interior chemical profiles. You can take one that came out of your favorite stellar evolution code, or any from the literature, or whatever crosses your mind (though again, check the limitations). You can't put gold in these white dwarfs. The interior chemical profiles are not evolved by the code (no time dependent diffusion of elements). What you put in is what you get out at the end. All other quantities are calculated by the code in order to make a model that is a solution to the proper stellar interior equations.

For v20 of the code, the oxygen profile is parameterized and the parameters entered in the file called gridparameters. More on that in later sections.

You also provide what temperature you want your white dwarf model to have. The code will calculate a model with that temperature. Then you have the option of having the code calculating pulsation properties of the model you produced.

This code can run on any modern laptop (and even older). It will take about 30 seconds to run for the first instance, less for subsequent models on the list, if running a batch job.

3. A brief history of the code

The original lines of code date back to the early 1970's. It was written in Fortran. The first paper written about the code (and only one specifically about the code until 2017) was Lamb & Van Horn 1975, Astrophysical Journal, 200, 306.

Don Winget inherited the code, when he was a very young graduate student at the University of Rochester. The present author had the opportunity to go back to 40 year old research notebooks that helped her make sense of what some of the quantities in the code are (just a few, the quest continues). He brought the code with him to the University of Texas at Austin, where generations of his students made additions and modifications. Some are documented in the source code.

Significant modifications code wise included a repackaging of the different pieces of the code by Travis Metcalfe in the 1990's for use on super computers. The original code had 3 parts to it: the evolution code which made a white dwarf model. Then there was a “prep” code, which added shells to the model to increase its resolution and computed some quantities needed in solving the equations of non-radial oscillations. Finally, a pulsation code would take output from the prep code and compute pulsation periods and other pulsational properties. You might see these three elements mentioned as you explore the code, so this part of history is important to know.

Two decades later, the present author rewrote the code in Fortran 90 and modularized it, in order to be able to incorporate into the code state of the art opacities and equations of state from MESA (Modules for Experiments in Stellar Astrophysics). Mike Montgomery wrote the wrapper routines that allows WDEC to interface with the MESA modules.

This is the version of the code you have here. The use of MESA code means that the code is more challenging to install and also runs less fast. But it is a good sacrifice to make for state of the art physics. The use of MESA modules also means that there is more flexibility in terms of what chemical elements can be included (though there are still limitations!).

I should note that you do have the option of running the code with older physics, in which case there is no need to install MESA and the code runs faster. For something quick and dirty, that might be a good option. Be aware that in that case, there are more restriction on the chemical profiles.

Because it is fast, the code has been used to calculate extensive grids of models to do period fitting. This is important to know, as it explains the slightly clunky way to run it, which involves first generating a file with input parameters then using that file as input. For a single model, that's a clunky method. But if you want to run a series of models, the input parameter file can be used to run batch jobs.

4. Packing list

The files fall into 3 categories: source files, input files, supporting files.

Source files

Those are the fortran routines that make up the meat of the code and a Makefile for compiling.

```
block_data.f90
**calcp_mainroutine.f90
chemprofiles_subroutines.da.f90
**chemprofiles_subroutines.f90
chemprofiles_subroutines.heliumcore.f90
**commonblocks.f90
eos_wd.f90
eprep_subroutines.f90
evol_mainroutine.f90
**evol_subroutines.f90
**getpar_grid.f90
istat_subroutines.f90
nuax_subroutines.f90
ocon_functions.f90
phase_functions.f90
pulse_mainroutine.f90
pulse_subroutines.f90
utils_subroutines.f90
wd_eos_mod.f90
wd_opalz_mod.f90
```

```
wd_test.f90
Makefile
```

**Have a v15 and a v20 version.

Input files

These are the files you will be editing to get the code to produce the white dwarf(s) you want.

```
controlparams
gridparameters
inputprof
```

Each have a v15 and a v20 version. You will want to copy whichever set you want to the above file names.

Supporting files

These are files the code reads from while making models.

Directory called masses and the starter model files it contains.
Equation of state and opacity tables:

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

5. Compiling the code

A template Makefile is included with the source files. It can be used as is if you are on a Linux system and have either the gfortran or the ifort compilers (uncomment the proper lines). If you are running another OS and/or a different Fortran compiler, you will need to edit the flags as required, or write your own compiling script.

There is one compiling script for making different codes. Details below.

The quick and dirty old version (does not require MESA)

You can try compiling this one first for practice. In that case, you want to comment out the two MESA_DIR lines in Makefile. To compile, type

```
$ make makeda_orig
```

This version of the code uses the opacities and equations of state tables listed above. Everything it needs to run is right in the code directory. It runs faster than the state of the art version.

State of the art version

Before you can compile and run the state of the art version of the code, you must first download and install MESA. This is a non-trivial process and MESA is no small code, but there is excellent online and community support as MESA, unlike WDEC, was written by a team of professional developers (and astronomers).

To get MESA, go to http://mesa.sourceforge.net/index.html. VERY IMPORTANT: you need to download a specific version of MESA, version r22.11.1. Libraries move around and variables get renamed from version to version, so WDEC will not be able to interface with any other version of MESA. Of course, you can rewrite some of the source code to make WDEC interface with whatever version of MESA you want to run. But for the average user, the best option is to get the MESA version mentioned above and install that.

Once you have MESA installed, edit Makefile to point to the proper directory then compile the code you want to use. There are two different ones.

C/O core white dwarf (for v15, change accordingly for v20)

```
$ make makedx_v15
```

Helium core white dwarf

```
$ make makeda_he
```

6. Running the code

Aside from the equation of state tables and the starter models (located in the directory masses/), there are 3 input files that you will be editing: controparams, inputprof, and gridparameters. Here you will need to copy the correct version of controlparams, inputprof, and gridparameters. e.g. copy controlparams_v15 to controlparams etc...

Controlparams – Tell the code what you want it to do for you

Controlparams contains a series of questions to which you need to specify answers. The questions are in plain English, but some might be a little cryptic. See section 3 for some background. A brief explanation of each question follows.

The first two questions (about MESA equations of state and opacities) refer to using the old version of the code or the state of the art version. If you did not install MESA, you can still run the code if you answer N to these two questions. The code will use the equations of state tables and opacities that are right in the current directory.

“How do you want to treat convection?” The answer to that question is either 1, 2, or 3. Explanations given right in the controlparams files. For further details, see Bischoff-Kim et al. 2017.

“Do you want screen output from the evolution code?” Your answer is likely “N”, unless you want to see stuff scrolling on your screen during the execution. It might be useful for debugging purposes, but the screen output might be a little cryptic. It is the historical output.

“Do you want tape file output from the evolution code?” There is an advantage to answering “Y”, as this will produce a “radius.dat” file with the log radius in cm of the models. The rest of the output files are files that originally would get read back in by the prep code (see section 3). Such output no longer gets written out to tapes, but to files named e.g. tape28. The format of the files is not meant to be easy to read by humans, or to be easy to plot. Again, it might be useful for debugging purposes.

“Do you want file output from the prep code to make plots?” Your answer to that might be “Y”, if you want to build a single white dwarf model and learn about its interior structure. The file output is meant for easy reading by humans and/or use in plotting routines. It may take some work to figure out what is what, however, as most files do not have headers. In section 8, I give some guidance on how to do that. Section 7 gives a general description of the output files and is a useful overview.

The next two questions are self-explanatory. It may seem silly to ask about file output separately from calculating pulsations. If we calculate pulsations, we obviously want to write out pulsation output. The pulsation output gets written at different stages in the code, so this is not an innocent question. Essentially, there is a summary of periods that gets written out no matter what, and then more detailed output that one may activate. More on file output in section 7.

The last question refers to output that allows to plot weight functions that show where the modes are trapped (Montgomery et al. 2003). It is not that much trouble to incorporate as I have source code that does that, I just never got to it. If you are interested in incorporating this, or want to run kernels output separately, please contact me.

Inputprof – Provide a desired shape for the chemical profiles in the model

The file inputprof contains some explanations. In a nutshell, if running v15, you are asked to provide the oxygen abundance profile. It is important to specify how many points you are providing to specify the oxygen profile, as this programmer was too lazy to automate that. If it drives you crazy, you can go into the source code and make that improvement (please consider sharing the improved code).

Then there is a parameter that specifies how closely you want the actual profile to follow what you specified. If you are giving only a few points and want the code to build a smooth curve off of that, go with something closer to 10. If you are feeding it a profile from a stellar evolution code that you want to use as is, then use 500. This will cause the code to essentially use the profile you gave it and not smooth anything.

The last two parameters define buffer zones around locations of transition zones in the chemical profiles. The first one is buffer_inner (in Mr units) and the second is buffer_outer (in -log(1-Mr/M*) units). Start with 9.0d-1 and 5.0d-1 (0.9 and 0.5). These parameters may have to be tweaked in order for the chemical profiles to be smooth. See more details in the troubleshooting section.

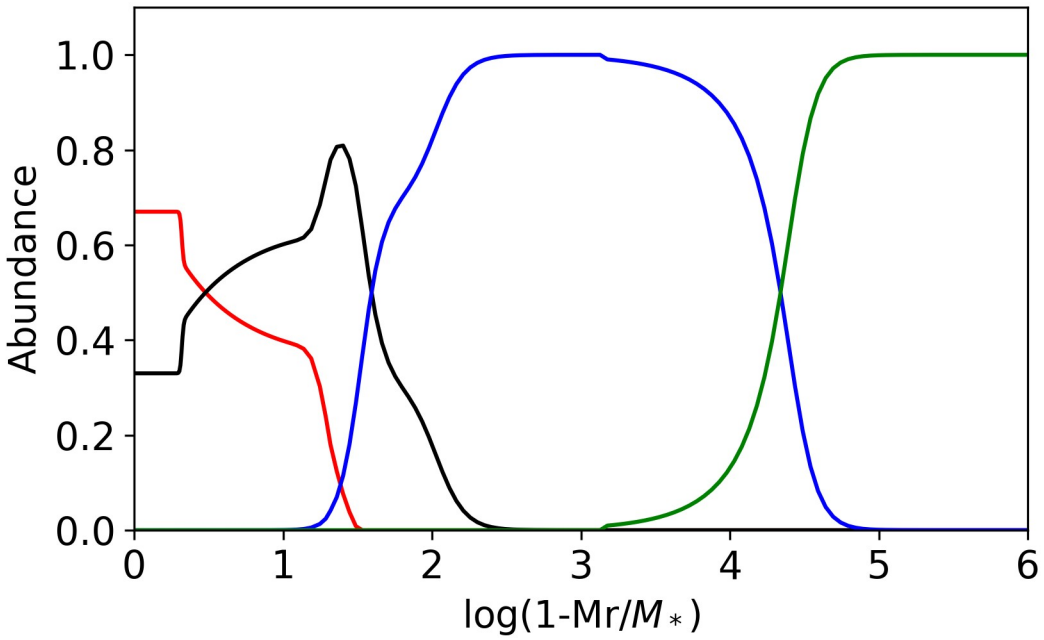
If you are running v20, then you do not need to provide an oxygen profile (and neither should you). Use inputprof_v20 (but rename or copy to inputprof so the code can find it).

Gridparameters – Provide additional input parameters, for one or more models

The file gridparameters must contain at minimum one line, specifying properties of the desired model. You can add more lines if you want to run a batch of models. For v20, the single line may look like this (here it's wrapped)

```
11000.0   600.0    161.1    212.2    412.2     69.0     13.3     10.3
96.0      67.0     56.0     38.0     52.0      2.0      41.0     2.0
```

This will make a white dwarf that has a temperature of 11,000 K, and a mass of 0.600 Msun. The rest of the parameters specify chemical abundance profiles. A picture is worth a thousand words so the figure below illustrate what the chemical abundance parameters specify.



For a description of each parameter, look in the source file `getpar_grid.f90`, starting on line 99. The best way to familiarize yourself with what each parameter does is to run models varying one parameter at a time and see how that parameter changes the chemical profiles. Also look at the conference proceedings included in the documentation folder along with this manual.

An easy way to produce a `gridparameters` file is to use the `mkgrid` utility, included in this depository. For more on this, see the Appendix.

As a side note, in case you are wondering why all parameters are specified as positive values greater than 1 (multiplying when necessary), it goes back to days when these parameters were fed into the code through shell scripting (and so no decimals allowed). This arcane convention remains because of laziness, but also because looping through whole numbers is conceptually easier when creating grids.

Once you have specified the input, simply run the executable by typing

```
$ ./makedx_v20
```

Execution should take under a minute. If it is longer, it is probably hanging. Kill the process and relaunch after editing `controlparams` to get screen output. Also try changing the input parameters in `gridparameters`.

For v15, there are fewer parameters in `gridparameters`. You only need the first 9 parameters. The oxygen abundance profile is not parameterized, but instead specified in the file `inputprof`.

Helium core white dwarfs

To make a helium core white dwarfs, compile makeda_he as instructed in section 5. For the rest, it works similarly to C/O core white dwarfs. There are differences, however, in the values you set for the parameters in the `gridparameters` file:

- Parameter 3 now sets the thickness of the hydrogen layer (where the hydrogen layer first takes off from zero)
- Parameter 4 now sets the location in the model where we go from a hydrogen/helium mix to pure hydrogen
- Parameter 5 is not used (feel free to set it to whatever float strikes your fantasy)

The rest of the parameters have the same meaning as before. Do not forget to set h1, h2, and h3 to zero (parameters 10, 11, and 12). The last 4 parameters become irrelevant and can just be left alone.

7. Description of key output files

In this section, we limit ourselves to discussing some key output files. Depending on the flags you set in `controlparams`, you will get anywhere between 1 and 30+ files.

calcperiods

If you do something silly like saying “N” to all output in `controlparams`, you still get the file `calcperiods`, but it will not list any periods. It might look like this:

```
 40000.  600.  500.  700.  800.   60. 10.0 10.0  1.00
  0.0000000000000000
  100000
```

It spat the parameters listed in `gridparameters` back out, then there is 0. and then 100000. I explain the latter below, with an example where we did ask to calculate pulsation periods. The code writes out the calculated periods to `calcperiods`. `Then calcperiods` might look like this:

```
 40000.  600.  500.  700.  800.   60. 10.0 10.0
          1   129.63071929222846
          1   167.30720084865592
          1   197.38566565316265
```

```
            1   226.92951357895177
            1   253.05251098532727
            1   284.69333321586623
            1   316.93330428778086
…
            1   1430.1725198348511
            1   1461.3989465485579
            1   1493.6983813392360
            1   1493.6983813392360
            2   102.93378501409387
            2   120.27516329156133
            2   139.25283753152820
            2   152.22916991972829
…
            2   1450.5330101765014
            2   1467.6631650174181
            2   1486.0908532143724
            2   1486.0908532143724
   0.00000000000000000
      100000
```

Now we have a set of periods (ell=1 and ell=2) before the 0. and the 100000. The zero used to be a place where the code would output a fitness parameter comparing the list of calculated periods to a list of periods that we were trying to match. The matching subroutine and the computation of the parameter have been stripped out of the code and is today a standalone code (available on request). For purposes of interfacing with existing tools, I left a double set to zero as a place holder. The 100000 acts as a separation between successive lists of periods, when running more than one model at a time in `gridparameters`.

Results.dat

To get more information on the pulsation periods, say "Y" to output from the pulsation code and open results.dat. It has radial overtone values, kinetic energies of the modes, and more.

Evolved

Evolved is a key file. You obtain it if you say "Y" to "Do you want tape file output from the evolution code". The file evolved contains key quantities that describe the model. Unfortunately, it is not in a human or plotting routine readable version, as it lists the values in blocks on a page 4 column wide. See section 8 for more on deciphering that file.

Plotting ready files

An easy way to extract most information from the evolved file is to say "Y" to "Do you want file output from the prep code to make plots?". Then you get a series of files with model values organized in columns, which a plotting routine can easily grab. Most such files have ".dat" endings. Some file titles are self-explanatory, most are not. The document "description_of_key_variables.pdf" comes in handy here. Also see section 8 for help on determining what is what.

8. Some uses of peaking at the source code and some help on that

As has been eluded to in the previous section, one big use of peaking at the source code is to figure out file output in detail. I will also discuss how to find out what at least some of the variables in the code are (always a big issue and an ongoing process), and how to recover what the paramaters set in `gridparameters` are and to see what some of the defaults are. I begin with a practical example of figuring out what is in a given output file.

Determining what is in the output files for the non-Fortran savvy

As an example, let's say you are trying to figure out what gets written out to the all important “evolved” file. First, find mentions of this file name in the source code, using grep. E.g. in the folder where the source code is, type

```
$ grep 'evolved' *.f90
```

Two outputs come up. One has an exclamation point starting it. That means it is a comment, so unlikely to be what we are looking for. The relevant output is

```
evol_subroutines.f90:    fname = "evolved"
```

We open evol_subroutines.f90 and search for 'evolved' to find that given line of code. It appears that 'evolved' gets saved as a string variable called fname. So search for that. We find the line

```
open(50,file=fname,status='unknown')
```

This is useful. It tells us that Fortran opens the file named 'evolved' as unit number 50 (some loose Fortran 90 here). The line write under it says

```
open(50,file=fname,status='unknown')
write(50,161) modnr,ssg,p2,t2,ucent,rm,tel,bl,bnt,bax,10.**amxc
```

Search for other instances of 'write(50', or if that fails 'write (50' (add a space between “write” and the partenthesis). We don't find any other ones in evol_subroutines.f90. That means the rest is elsewhere. Using grep again, we find

```
eprep_subroutines.f90: write(50,1030) nshell, nl,nel,amhyhe,amheca, &
eprep_subroutines.f90: write(50,1020) (aa(i,n), n=1,nshell)
evol_subroutines.f90: write(50,161) modnr,ssg,p2,t2,ucent,rm, ...
pulse_mainroutine.f90: write(50,3005) mstar,model,age,llsun,rrsun,...
pulse_mainroutine.f90: write(50,3007) amhyhe,amheca,alph(1), ...
pulse_subroutines.f90: write(50,1003) gnu0,per0,tdyn
```

The second output seems to be the promising one, as it looks like it is writing out a lot of numbers (a 2D array with at least one dimension being the number of shells). But we are still left in the dark. What quantities does the array aa(:,:) contain? Grep!

```
$ grep 'aa(' *.f90
```

And here, we score. Lines and lines of output, but the top ones are very useful (we land on one of the better commented parts of the code).

```
commonblocks.f90:! aa(1,:)  = radius
commonblocks.f90:! aa(2,:)  = mr
commonblocks.f90:! aa(3,:)  = lr
commonblocks.f90:! aa(4,:)  = temperature
commonblocks.f90:! aa(5,:)  = density
commonblocks.f90:! aa(6,:)  = pressure
commonblocks.f90:! aa(7,:)  = neutrino emission rate
commonblocks.f90:! aa(8,:)  = cv
commonblocks.f90:! aa(9,:)  = chr
commonblocks.f90:! aa(10,:) = cht
commonblocks.f90:! aa(11,:) = epst
commonblocks.f90:! aa(12,:) = epsr
commonblocks.f90:! aa(13,:) = kapr
commonblocks.f90:! aa(14,:) = kapt
commonblocks.f90:! aa(15,:) = del
commonblocks.f90:! aa(16,:) = delad
commonblocks.f90:! aa(17,:) = xhe
commonblocks.f90:! aa(18,:) = kap (opacity)
commonblocks.f90:! aa(19,:) = ledoux term
commonblocks.f90:! aa(20,:) = oxygen abundance
```

This is an illustration of using code reverse engineering to figure out what the output is, and a great segway into the modules of commonblocks.f90

The two files that contain most of the variables

`commonblock.f90` is where the more globally used variables are defined, and that's a lot of them. The vast majority of them remain unidentified, but as I worked on the modern version of the code, I added comments, so you may find some useful information about the variables in that file.

Another file where one finds global variables is `block_data.f90`. That file is more oriented toward defining constant values (or as Fortran calls variables that are not to be modified in the code, “parameters”). For historical reasons, it also contains more global variables, not defined in `commonblocks.f90`.

getpar.f90 – recover what gridparameter lists and see what some of the defaults are

The last source file I want to emphasize is `getapar.f90 or getpar_grid.f90`. Perhaps this is where we should have started, as that is the main program (if you were looking for it).

In the first part of the code, you will recognize where the logical flags are read from the input file `controlparams`. Following that section, there is the part where input values are read for the evolution code. Comments may prove useful. The input file `gridparameters` contains what we

have traditionally (even if some of the tradition is recent) modified routinely. One can make more changes to the models in the source code right in `getpar_grid.f90`.

These start at `datain(12)`. In particular, note that you can choose what period range you want the code to calculate. The rest is a little more cryptic and will require more code reverse engineering to figure out, which is beyond the scope of this manual.

9. Limitations of the code

There are probably more limitations to the code, but a big one is the constraints on the chemical profiles. In its current state, the code can only accommodate C12, O16, He4, and H1. Furthermore, where each can exist is limited and depends on which version of the code you are running.

Here we must talk about a key feature of the code, which it has had since the beginning. It is its skeleton and not something that can be removed or fixed (though we've mended any broken bones, albeit ungracefully at times). Partly because of the wide range of orders of magnitudes involved in some of the quantities, the models are broken down into two parts: the core and the envelope. The two parts are numerically defined by a set boundary (which you will find in the source code under the name "stop mass" and as a variable "`stpms`"). It is important to note that this boundary is purely numerical, and has no physical significance whatsoever. It has to be carefully defined, however, as some physical processes are allowed in the core and not in the envelope (such as neutrino production). The core is most of the model and the envelope the outer portion that's left.

Be default, `stpms` is set to 0.99 (99% is core, outer 1% is envelope). This is more robust numerically (especially if running `makeda_orig`). If you need core physics further out in the model, there is the option of changing `stpms` to 0.9994. If you do that, however, you will need to change two things in the code (grep for each):

comment out:

```
if(istpms.eq.2) stpms = -0.0043648012 !(99.0%)
```

and uncomment:

```
if(istpms.eq.2) stpms = -0.000260655  !(99.94%)
```

(a comment line in Fortran is announced by an exclamation point)

Do the same with this snippet of code

```
stpmsid = 9900   ! up to 99% stop mass
! stpmsid = 9995   ! up to 99.94% stop mass
```

This boundary is especially crucial to know about when using the non-MESA version of the code, the one that uses old opacities and equation of state tables. You will notice that the equation of state files bear names like `EEOSC` (Carbon EOS for the Envelope) and `IEOSC` (Carbon EOS for the Interior, or core). Also, that there is an `IEOSO`, but no `EEOSO`. This is key. Oxygen cannot exist past the core-envelope boundary. We don't have the old EOS tables for that. That is one limitation of the old version of the code. This particular limitation has been lifted for the new code, as the MESA equations of state tables and opacities exist for the entire model.

If you want to dig into the source code to learn more about what element can be included where, search for the array variable "`xavec`".

The code can compute models from ~0.3 solar masses (you can push it lower, but will start to experience difficulties with the code converging) to ~ 1.0 solar masses (again, the code starts to struggle at the higher masses).

We also note that because of the crude treatment of the atmospheres, this code is not very accurate when it comes to ages of cooler white dwarfs (under 5,000 K or so).

10. Troubleshooting

- *The code exits with an error message* ("Bad eos get" or something about getting to the end of `inpuprof`). This means that the code is having trouble converging. The best approach is to give up on that particular set of parameters. Modify the parameters in `gridparameters` and try again. Go back to the last set of parameters that worked and work your way incrementally from there.

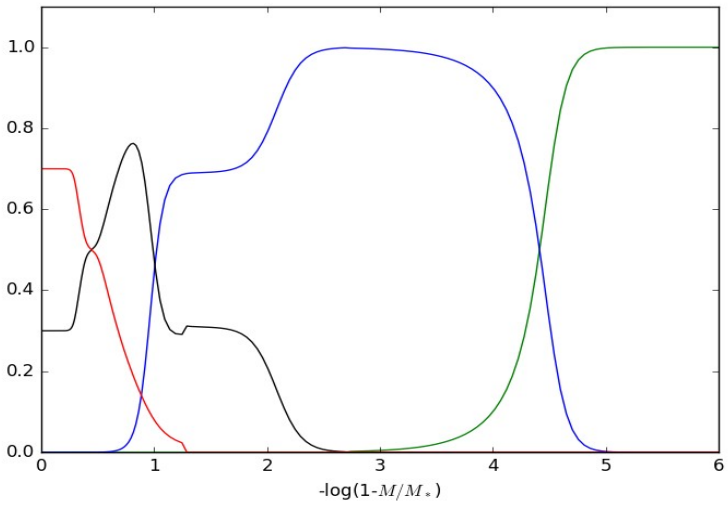
- *My input profile gets ignored!* It doesn't get ignored, but may be getting smoothed beyond recognition. Try a larger value of the smoothing parameter in `inputprof`.

- *My value for convective efficienty that I specify in `gridparameters` appears to get ignored.* That value only gets used if in `controlparams`, you set the treatment of convection flag to 1. Otherwise, it does get ignored and set through other methods, hard coded.

- *My chemical profiles look funky.* Below are some scenarios, with figures and fixes.

Issue #1 (v15 only): The oxygen abundance (red) suddenly drops down to zero, causing carbon (black) to suddenly jump up.

The problem: In our definition of the oxygen profile in inputprof, we allowed it to go too far out.



Define points as x y coordinates, `x being in Mr/Mstar`

```
0.          0.7
0.5         0.7
0.6         0.5
0.7         0.5
0.9         0.1
0.92        0.02
1.00        0.0
```

Fix: modify the last few lines of the oxygen profile definition so it doesn't go as far out.

```
Define points as x y coordinates, x being in Mr/Mstar
0.          0.7
0.5         0.7
0.6         0.5
0.7         0.5
0.8         0.1
0.82        0.02
0.90        0.0
```

Issue #2: The carbon abundance (black) suddenly drops from 1, causing a sharp feature in the profile. The helium abundance picks up suddenly.

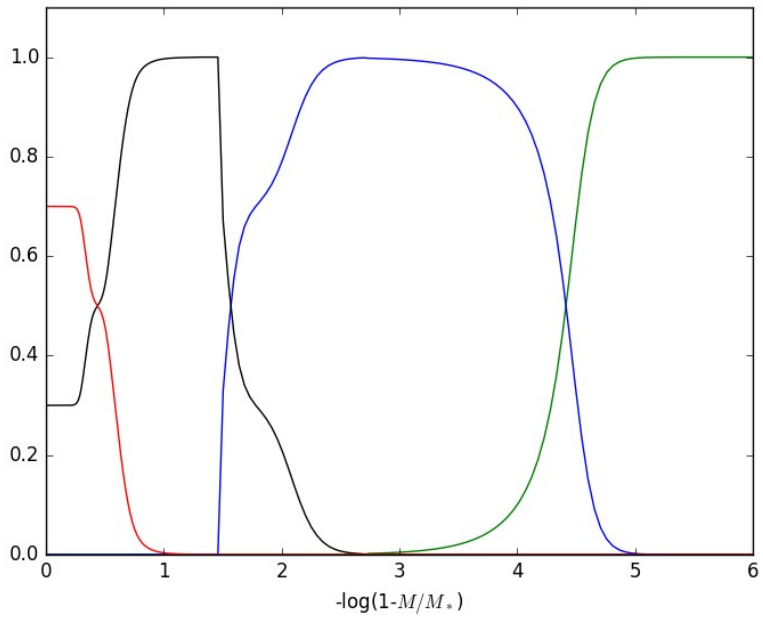
Problem: buffer_inner, defined at the end of inputprof, is not large enough

(last two lines of inputprof)

```
8.0d-4
1.0d0
```

Fix: Increase buffer_inner

```
8.0d-1
1.0d0
```



Issue #3: The carbon abundance (black) suddenly drops down to zero after turning over, causing a discontinuity in the helium profile as well (blue).

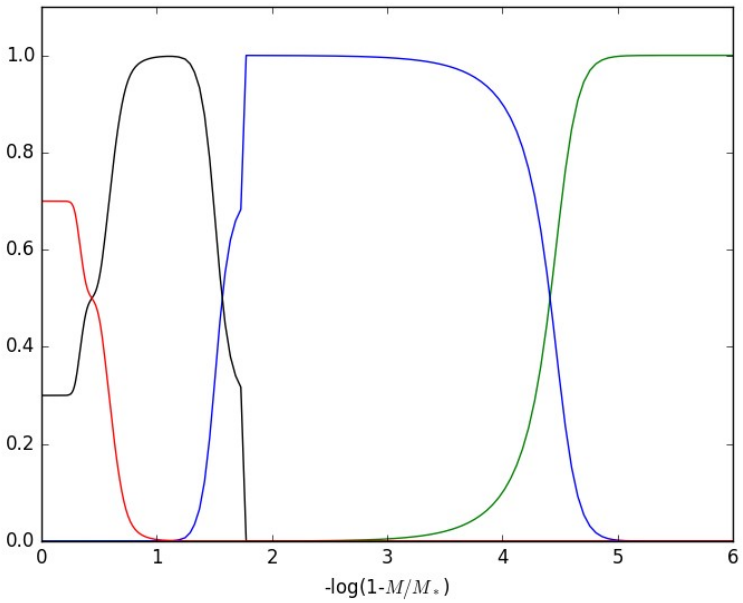
Problem: buffer_outer is too big

(last two lines of inputprof)

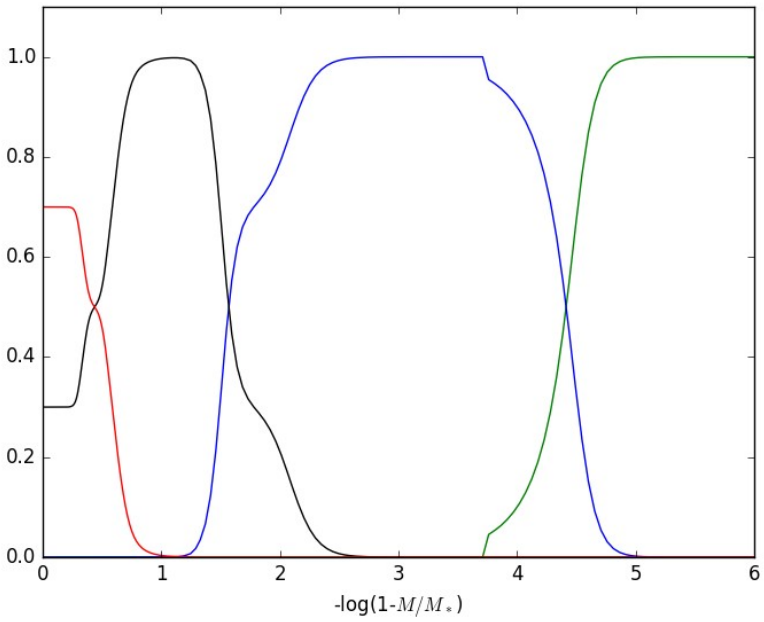
```
8.0d-1
2.0d0
```

Fix: Decrease buffer_outer

```
8.0d-1
1.0d0
```



Issue #4: The helium abundance (blue) has a weird discontinuity at the helium/hydrogen interface. Hydrogen (green) also picks up in a discontinuous manner.



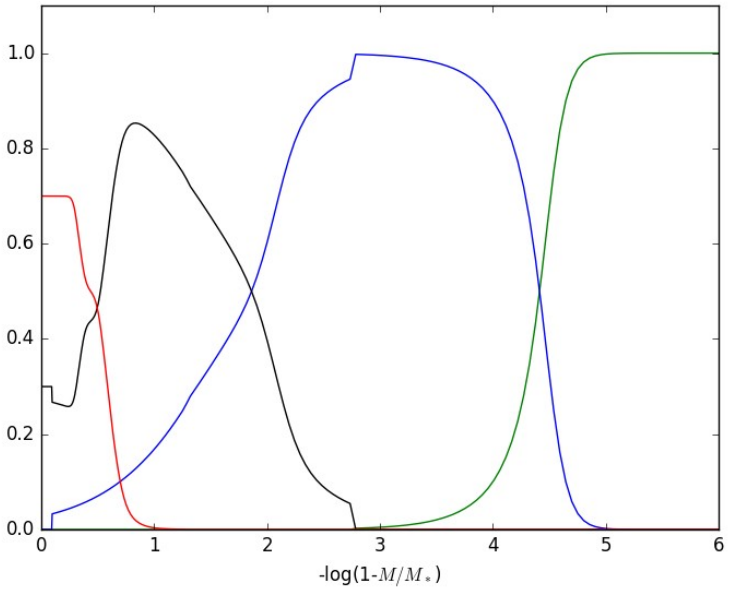
Problem: buffer_outer is too small
(last two lines of inputprof)

```
8.0d-1
1.0d-3
```

Fix: Increase buffer_outer

```
8.0d-1
1.0d0
```

Issue #5: Aaaaaaah!



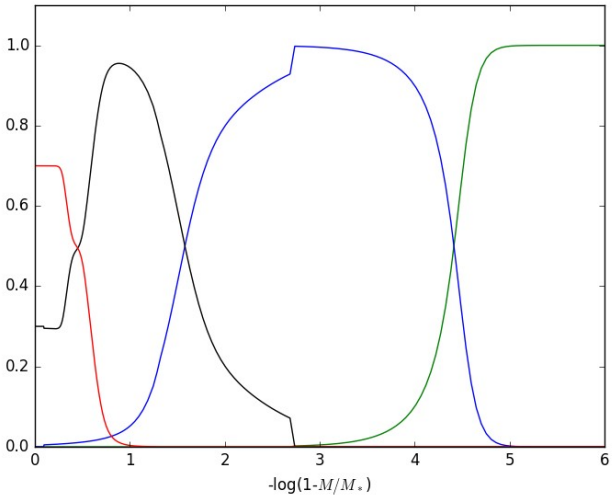
It's a miracle this converged.

Problem: The helium profile at the carbon/helium interface is too gentle in its slope. It does not go down to zero fast enough toward the center, causing all kinds of trouble.

Fix: increase the value of the diffusion coefficient in gridparameters.

```
11288.0   593.0   160.0   218.0   420.0    69.0     2.0    9.0   1.3
11288.0   593.0   160.0   218.0   420.0    69.0     6.0    9.0   1.3
```

Issue #6: The helium abundance (blue) grows slowly, then suddenly jumps to 1.

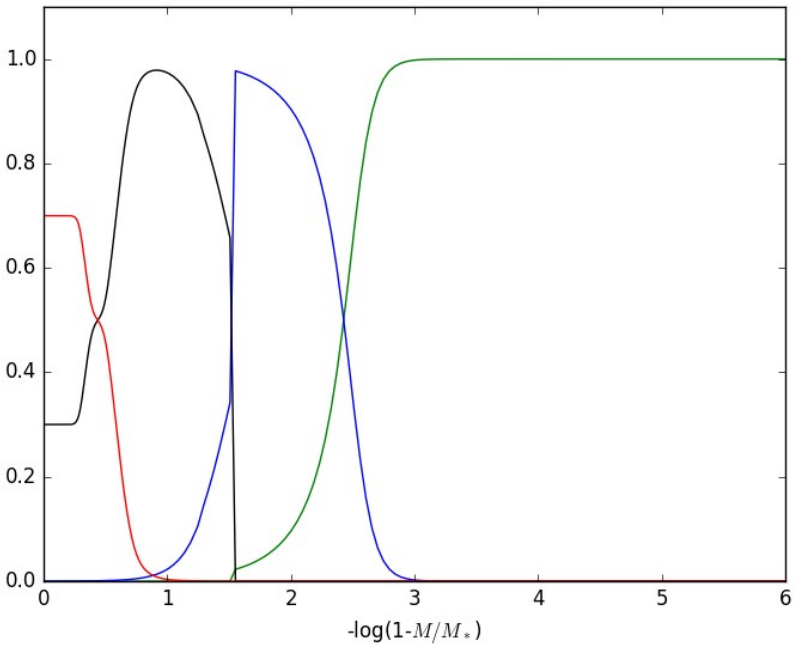


Problem: A diffusion coefficient in gridparameter is too low, causing the helium profile to not get to 1 soon enough.

Fix: Increase the diffusion coefficient in gridparameters

```
11288.0   593.0   160.0   218.0   420.0    69.0     6.0    2.0   1.3
11288.0   593.0   160.0   218.0   420.0    69.0     6.0    9.0   1.3
```

Issue #7: Issue #3 (sudden drop of carbon to zero) and Issue #4 (discontinuous tail in the hydrogen profile) both happen at once.



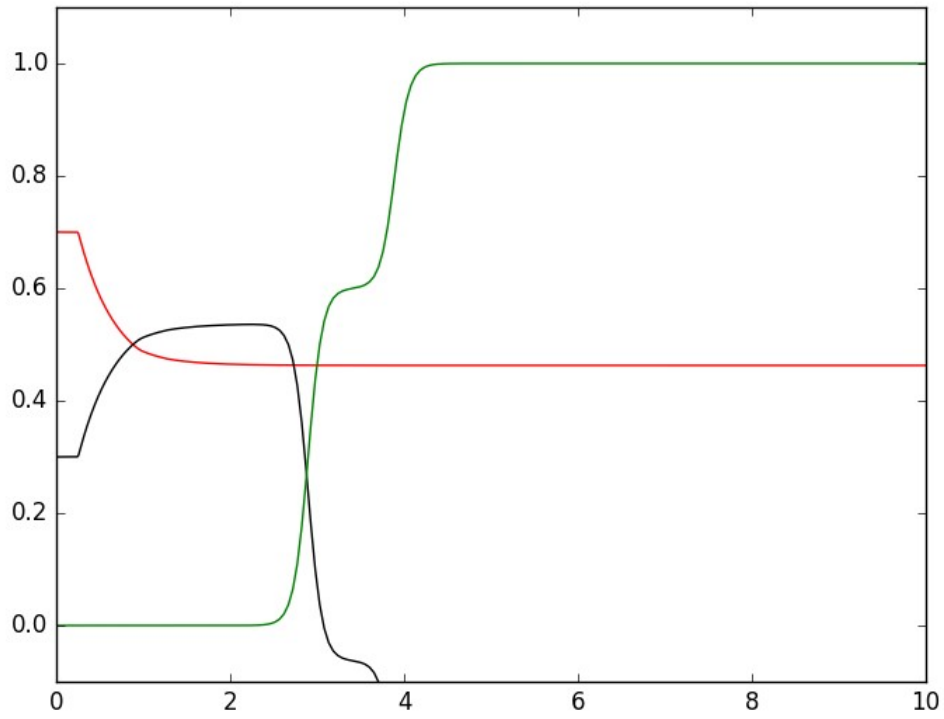
Problem: The helium profile is squeezed too tightly between carbon and hydrogen.

Fix: try increasing the values of the diffusion coefficients (see fix for issue #5 and #6) and adjust buffer_inner and buffer_outer (fix for issue #3 and #4). That may not work, however. You may need to simply give more room for the helium. Choose parameters in gridparameters that are such that they place the base of the hydrogen layer further from the base of the helium layer.

```
11288.0   593.0   160.0   218.0   220.0    69.0     6.0    12.0   1.3
11288.0   593.0   160.0   218.0   420.0    69.0     6.0    12.0   1.3
```

A good rule of thumb is to enforce the rule 5th parameter > 3rd parameter + 200 (here 420 > 160 + 200).

Issue #8 (20 only): My oxygen profile looks funky



Problem: some of the points specified to shape the oxygen profile are not in the core.

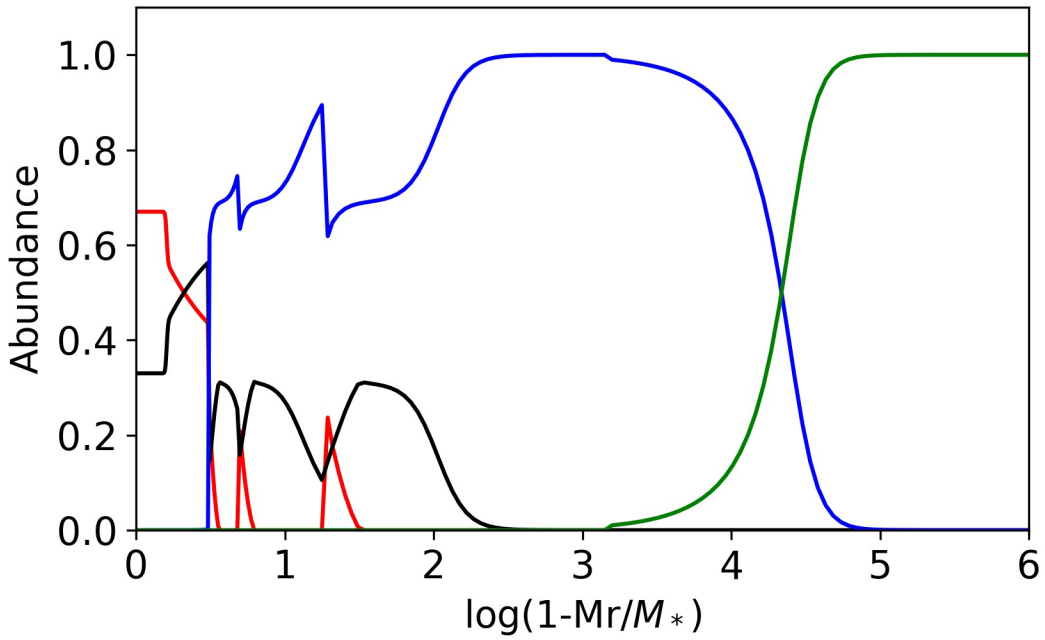
Fix: Make sure w1 + w2 + w3 + w4 add up to less than 100 (to be safe, make that 98).

```
25000.0     600.0     300.0     400.0    2000.0      60.0      12.0
12.0      60.0      70.0      70.0      80.0      45.0      45.0      36.0
```

45.0 + 45.0 + 36.0 = 126. Not good.

Tweak these values so that their sum is comfortably under 100.

Issue #9 (v20 only): I don't even know what this is



The helium layer goes too deep. Increase Menv (third parameter). 200 is safe, then try to go down in steps of 20 from there.

```
  50000.0     800.0     120.1     212.2     412.2      69.0      13.3
10.3      96.0      67.0      56.0      38.0      52.0       2.0      41.0
2.0
```

Appendix

The utility mkgrid produces batch files of parameters to run (the `gridparameters` file).

1. Find the code
You will find the code in the depository kim554/asteroseismic_fitting.

2. Compile the code
```
$ gfortran common_values.f90 subroutines.f90 generate_grid.f90 -o
mkgrid
```

3. Edit the file inputparameters
The file has headings and is self explanatory. The minima and maxima are inclusive (if the maximum is an integer number of steps away from minimum). To run a single value for any parameter, put the same value for minimum and maximum.

One “hidden”, cool feature is the ability to allow the code to set default values that will reproduce chemical profiles from stellar evolution (more specifically, those of Althaus et al. 2010). To enable that

feature, enter a “-1” for any of the envelope and core parameters. The code calculates optimal values for those parameters, based on the mass of the model. This only works for C/O core white dwarfs.

4. Run the code

```
$ ./mkgrid
```

The code will put out a `gridparameters` file for use with WDEC.