NIUS Project Report

Author : Jayant Thatte[1]

Abstract

The report presented below gives a brief summary of the work done on the project under NIUS (National Initiative for Undergraduate Sciences), an initiative by TIFR (Tata Institute of Fundamental Research), Mumbai.

The aim of the project is to use computational techniques and knowledge of the rapidly developing field of Asteroseismology to deduce the internal parameters of a star such as density and pressure profiles, composition, existance and extent of convective core or envelope by analysing its oscillation frequency data. To achieve this, a multi-dimensional grid of stellar models was generated with each dimension in the grid giving gradation of one physical stellar property. Parameters used for making the grid included stellar mass, composition (X, Y, Z parameters), convective overshoot and mixing length. After the grid was generated, oscillation frequencies (for different modes) were calculated for each point in the grid.

The next step is to take real stellar frequency data[2] and try to fit this star into the grid using a fitting algorithm. Once the star has been placed on a region in the grid, the internal properties of the star can be predicted (with some tolerence) using interpolation.

The first few months were spent in reading and understanding of the first six chapters from *Theory of Stellar Evolution* by *Dina Prialnik* and chapter 1 and parts of chapter 2 and 3 of *Astroseismology* by *Aetrs, Christensen-Dalsgaard,* Kurtz. The work done also includes getting acquainted with YREC code, reading and understanding *YREC (Yale Rotating Evolutionary Code)* by *P.Demarque et. al.*, running the code for different stars of masses and learning basics of shell scripting in Linux.

The latter part of the project involved generating the stellar grid using YREC. Several parameters in the code need to be customised for generating each point in the grid and hence, the process cannot be done manually. Hence, this part of the project also requied heavy use of Linux shell scripting to automate the process and for efficient data management.

Sections 1 to 5 explain the basic concepts of structure and mechanics of the stellar interior. Sections 6 and 7 briefly cover a qualitative understanding of the causes and types of steller oscillations without going into rigorous mathematical details. A list of topics studied for the project is given in section 8. Section 9 gives a small paragraph about the working of YREC and an overview of the support package developed as a part of this project. Note that the picture of YREC protrayed in this section is an over-simplified one. In reality, one need to tweak numerous parameters in the code to generate reasonably accurate models in a practical computation time. Sections 10 and 11 give a detailed documentation of the YREC support package.

Under each section, only the key equations/concepts are given. Derivations and rigorous arguments are excluded from this document. This document does not give the complete details of all the work done, but only important points of major parts of work.

[1]Department of Electrical Engineering, Indian Institute of Technology Madras

[2]For this project, data from French mission CoRot (COnvection ROtation et Transits planétaires) was used.

Contents

Part I

Stellar Interior

1 Basic Equations

1.1 Equations of Stellar Evolution[3]

- **Energy Equation**: First law of thermodynamics applied to stellar interiors gives

$$q - \frac{\delta F}{\delta m} = \dot{u} + P\frac{d}{dt}\left(\frac{1}{\rho}\right) \approx 0 \tag{1}$$

  Because the dynamic time scale and thermal time scale of the star is much smaller than nuclear time scale, right hand side can be assumed to vanish.

- **Equation of Motion**: Balancing forces due to gravity and hydrostatic pressure

$$-\frac{Gm}{r^2} - 4\pi r^2\frac{\delta P}{\delta m} = \ddot{r} \approx 0 \tag{2}$$

  Because the dynamic time scale is very small, the star can be assumed to be under hydrostatic equilibrium during most of its lifetime. Hence $\ddot{r}$ can be set to zero.

- **Equations of Composition**: Let X_i be the mass fraction of the i^{th} element composed of n species, for $i = 1$ to n

$$\dot{X}_i = f(\rho, T, Xi) \tag{3}$$

1.2 Virial Theorem

- Virial theorem is derived from the equation of hydrostatic equilibrium and hence is **valid only when star is under hydrostatic equilibrium**.

$$P_sV_s - \int_0^{M_s}\frac{P}{\rho}dm = \frac{\Omega_s}{3} \tag{4}$$

  Ω_s is the energy needed to assemble a sphere of mass M_s with volume V_s and surface pressure P_s. For stars, typically $P_s = 0$.

- For ideal gas and non-relativistic degenerate electron gas $U_{gas} = -0.5\Omega_{gas}$ and hence[4] $E = U + \Omega = -U_{gas}$
- For radiation and relativistic electron degenerate gas $U_{rad} = -\Omega_{rad}$
- For a star $\Omega = \Omega_{gas} + \Omega_{rad}$ and $E_{net} = -U_{gas}$

1.3 Time Sclaes

Time scale of a quantity $\phi(t)$ is defined as $\tau(t) = \phi/\dot{\phi}$ and can be thought of as the time needed for the quantity to completely deplete given its instantaneous time derivative.

- **Dynamic Time Scale** (τ_{dyn}) is defined as

$$\tau_{dyn} = \frac{R}{\dot{R}} \approx \frac{R}{v_{esc}} = \left(\frac{R^3}{GM}\right)^{1/2} = \frac{1}{(G\overline{\rho})^{1/2}} \tag{5}$$

  where $\overline{\rho}$ is the average density of the star. For sun $\tau_{dyn} \approx 10^3 sec$. Stellar oscillations occur over dynamic time scale.

[3] Effects of rotation and magnetic field are neglected. We also assume that the star is isolated and with uniform initial composition.
[4] Kinetic energy of the stellar material is assumed to be zero as the star is in hydrostatic equilibrium

- **Thermal (Kelvin-Helmholtz) Time Scale** (τ_{kh}) is defined as

$$\tau_{kh} = \frac{U}{L} = \frac{GM}{R^2 L} \tag{6}$$

and is equal to the life time of the star had it used only its gravitational potential energy to shine. For sun, $\tau_{kh} \approx 10^{15} sec$ which is few tens of million years.

- **Nuclear Time Scale** (τ_{nuc}) is defined as the lifetime of the star had it used only its nuclear energy to burn. It is given by

$$\tau_{nuc} = \frac{\varepsilon M c^2}{L} \tag{7}$$

ε is the binding energy of nucleon per unit rest mass. τ_{nuc} is much larger than the actual lifetime of the star and larger than the age of the universe itself indicating that stars do not use all of their available nuclear energy for burning.

- $\tau_{dyn} \lll \tau_{kh} \ll T < \tau_{nuc}$
where T is the stellar lifetime. Thus star can be assumed to be in dynamic and thermal equilibrium for most of its lifetime, but never in nuclear equilibrium.

- Since $\tau_{dyn} \lll \tau_{kh}$, most of the processes occurring in the *interior regions* of the star can be assumed to be adiabatic. This is known as the **Adiabatic Approximation**. This approximation not valid in the parts of the star close to the surface.

2 Gas and Radiation Pressure in Stellar Interiors

Consider an isotropic distribution of particles impinging on an imaginary surface and bouncing off elastically. Let momentum distribution of the particles be given by $n(p)$ where n is the number of particles per unit volume with momentum p. Momentum transferred to the surface per unit area due to particles coming at an angle θ to the surface normal is given by

$$dp_\theta = [v \cos\theta dt][n(p,\theta) dp d\theta][2p \cos\theta] \tag{8}$$

Since particle distribution is isotropic, the number of particles coming in at angle θ is equal to the corresponding solid angle given by $\Omega_\theta = 0.5 \sin\theta d\theta$. Hence

$$dp_\theta/dt = [v\cos\theta][n(p)dp][\sin\theta d\theta/2][2p\cos\theta] = [pvn(p)dp][\cos^2\theta \sin\theta d\theta] \tag{9}$$

Hence pressure on the surface is given by

$$P = \left(\int_0^\infty pvn(p)dp\right)\left(\int_0^{\pi/2} \cos^2\theta \sin\theta d\theta\right) = \frac{1}{3}\int_0^\infty pvn(p)dp \tag{10}$$

This is called the **Pressure Integral** and relates velocity distribution of the particles to the pressure that they exert.

2.1 Gas Pressure

- **Ion Pressure**: The ions can be treated as an ideal gas mixture and hence using Maxwellian velocity distribution and pressure integral, $P_{ion} = R\rho T/\mu_{ion}$ where μ is the mean atomic mass of the stellar material and is a function of X, Y and Z.

- **Electron Pressure (non-degenerate)**: $P_e = R\rho T/\mu_e$ where μ_e^{-1} is the number of electrons per nucleon in the stellar material.

- **Degeneracy Pressure**: $P_{deg} \alpha \rho^{5/3} m^{-1}$
where ρ is the density of the material and m is the mass of the particle causing degeneracy pressure. Thus, it is clear that P_{deg} has a significance only in case of electrons. $P_{e,deg}$ plays an important role in stars with very high density.

2.2 Radiation Pressure

Using Planck Distribution and Pressure integral, we get

$$P_{rad} = \frac{8\pi^5 k^4}{15c^3h^3} \times \frac{T^4}{3} \equiv aT^4/3 \tag{11}$$

where a is called the radiation constant.

3 Eddington Luminosity and Convection and Stellar winds

Consider a star under hydrostatic and radiative equilibrium. Hence

$$\frac{dP}{dm} = -\frac{Gm}{4\pi r^4} \tag{12}$$

$$\frac{dT}{dm} = -\frac{3}{4ac}\frac{\kappa}{T^3}\frac{F}{(4\pi r^2)^2} \tag{13}$$

where $F(m)$ is the radiative flux and κ is the opacity. Also, $P_{rad} = \frac{1}{3}aT^4$. Hence

$$\frac{dP_{rad}}{dP} = \frac{dP_{rad}}{dT}\frac{dT}{dm}\frac{dm}{dP} = \frac{\kappa F}{4\pi cGm} \tag{14}$$

Note that $|dP_{rad}| < |dP|$ and both have the same sign. Hence, $dP_{rad}/dP < 1$. This gives

$$\kappa F < 4\pi Gcm \tag{15}$$

- Violation of this condition, either due to very high opacity or due to very large energy flux, means that radiation is not efficient enough to transport all the produced energy. At least one of hydrostatic equilibrium or radiative equilibrium must be violated in this case.
- In the interior of the star, hydrostatic equilibrium cannot be violated and hence convection sets in (radiative equilibrium violated) if the above inequality is not satisfied.
- The surface of the star *must* be radiative. Hence $\kappa F(M) < 4\pi GcM$ that is

  $$L < \frac{4\pi GcM}{\kappa} \equiv L_{edd} \tag{16}$$

  The right hand side of the above inequality is called **Eddington Luminosity** and is the upper cap of luminosity for all stable stars.
- If $L > L_{edd}$, then hydrostatic equilibrium must be violated at the surface and the star starts giving out very strong stellar winds.

4 Instabilities in Stars

4.1 Thermal Instability

4.1.1 Secular Thermal Equilibrium

This is true for stars with low electron degeneracy pressure. For a star under equilibrium, $L = L_{nuc}$. If due to some small perturbation, $L \lesssim L_{nuc}$. This causes E to rise and hence U_{gas} falls (because $E = -U_{gas}$). This means that Ω has to rise. Thus, the gas expands and cools. Since both temperature and density fall, L_{nuc} falls and equilibrium is restored. The star is said to be under secular equilibrium.

4.1.2 Thermonuclear Runaway (TNR)

TNRs are observed in stars whose pressure is dominated by electron degeneracy pressure. If, for such a star, due to a small perturbation, $L \lesssim L_{nuc}$. Here again, U_{gas} decreases, but temperature does not fall[5]. Hence, the gas expands, but does not cool. The rates of nuclear reactions are much more sensitive to temperature and not as much to density. Hence L_{nuc} does not fall and hence remains greater than L. This causes temperature to rise causing a further shoot in rate of nuclear reactions. This leads explosion of energy called TNR. During TNR gas expands and much of heat is given out in the 'flash'. Hence, degeneracy is lifted and secular equilibrium is reached.

4.2 Dynamic Instability

4.2.1 Condition for Dynamic Stability

Consider a star of equilibrium parameters P_0, ρ_0, r_0. Let the star undergo a small uniform, radial contraction/-expansion. Thus

$$r = r_0(1 - \epsilon) \tag{17}$$

where $|\epsilon| \ll 1$. By substituting this in the basic stellar equations, we get

$$\rho = \rho_0(1 + 3\epsilon) \tag{18}$$

$$P_{gas} = \rho^{\gamma_a} = P(1 + 3\gamma_a\epsilon) \tag{19}$$

$$P_h = P(1 + 4\epsilon) \tag{20}$$

where P_{gas} is the actual gas pressure after perturbation and P_h is the pressure required to withstand gravity. A star is stable if $(P_{gas} - P_h)\epsilon > 0$. This gives condition for stability.

- Star is in dynamic equilibrium if $\gamma_a > 4/3$ everywhere.
- Star is in neutral equilibrium if $\gamma_a = 4/3$ everywhere.
- Star has global dynamic instability if $\int_0^M (\gamma_a - 4/3)\frac{P}{\rho}.dm < 0$. Thus denser regions of the star (core) contribute more.

4.2.2 Causes of Dynamic Instability

- As a star becomes more and more degenerate, $\gamma_a \to 4/3$ and hence we completely degenerate stars are unstable. For stars with core mass below the Chandrasekhar Mass the degeneracy pressure balances gravity, but for higher masses, the star collapses on contraction leading to a supernova.
- Star with dominant radiation pressure are unstable ($\gamma_a \to 4/3$). This is also indicated by Virial Theorem. Virial Theorem suggests that for radiation, $E = U_{rad} + \Omega_{rad} = 0$. Thus the star becomes an unbounded system.
- Consider a gas where number of particles (N) is not fixed. If volume (V) reduces by a small amount, particles undergo recombination and hence N reduces. Since $P\alpha(N/V)$, pressure does not rise as much as in constant N case. This may lead to instability. Indeed if γ_a is plotted[6] as a function of fraction of particles ionised (Figure 1), we clearly see that completely ionised and completely non-ionised gases are stable, but mixtures are not.
- Pair formation and Photodisintegration also cause instability through a similar effect at very high temperatures.

[5]For degenerate stars U_{gas} is related to the zero point energies of the various shell and hence is insensitive to temperature. Hence fall in U_{gas} does not force drop in the temperature of the gas.

[6]γ_a is plotted using equation (3.60) from *Theory of Stellar structure and Evolution* by *Dina Prialnik* with $\chi \sim 10eV$.

4.3 Convective Instability

Consider a fluid element of mass Δm. The element was initially at equilibrium at (P_1, ρ_1) and then moved radially outward by a differential length to a location ($P_2(< P_1)$, $\rho_2(< \rho_1)$). The initial state of the fluid element has to be same as its surroundings. As the element moves outward, it must expand so that its internal pressure now matches with the new external pressure P_2. Since $\tau_{dyn} \ll \tau_{kh}$, the expansion can be taken as adiabatic. Let the new density of the fluid element be ρ_a. The described configuration is stable if and only if $\rho_a > \rho$. This gives the following condition for *stability*

$$\left(\frac{dP}{d\rho}\right)_{star} < \left(\frac{dP}{d\rho}\right)_{adiabatic} \tag{21}$$

that is, a region of star is convectively stable when $\gamma < \gamma_a$.

5 Adiabatic Exponents

$$\Gamma_1 \equiv \left(\frac{\partial \ln p}{\partial \ln \rho}\right)_{ad}; \frac{\Gamma_2 - 1}{\Gamma_2} \equiv \left(\frac{\partial \ln T}{\partial \ln p}\right)_{ad}; \Gamma_3 - 1 \equiv \left(\frac{\partial \ln T}{\partial \ln \rho}\right)_{ad} \tag{22}$$

For ideal gas ($\beta = 1$) $\Gamma_1 = \Gamma_2 = \Gamma_3 = 5/3$ whereas for pure radiation, $\Gamma_1 = \Gamma_2 = \Gamma_3 = 4/3$. But for $0 \neq \beta \neq 1$, the three adiabatic exponents are unequal.

Part II

Stellar Oscillations

6 Simple Waves in Stars

Notation: For a quantity p, p_0 denotes its equilibrium value and p' is a small *Eulerian* perturbation while δp denotes the corresponding *Lagrangian* perturbation.

$$p(\mathbf{r}, t) = p_0(\mathbf{r}) + p'(\mathbf{r}, t) \tag{23}$$

$$\delta p(\mathbf{r}) = p'(\mathbf{r_0}) + \delta\mathbf{r} \cdot \nabla p_0 \tag{24}$$

6.1 Acoustic Waves

Consider a very simple case of spatially homogeneous medium. Takig divergence of equation of motion, we get

$$\rho_0 \frac{\partial^2}{\partial t^2}(\nabla \cdot \delta\mathbf{r}) = -\nabla^2 p' \tag{25}$$

Substituting p' by ρ' using Γ_1, we get waves travelling with $\sqrt{\Gamma_{1,0} p_0/\rho_0}$. These are nothing but sound waves travelling through the medium.

6.2 Internal Gravity Waves

Consider stellar medium with gravity pointing in the radial direction. In such a system, two kinds of waves propagate: (i) acoustic waves discussed above and (ii) waves of much smaller frequency.
Consider a fluid element which moves radially upward by a small distance and hence expands and if its density is more than surroundings, it falls back, thus giving rise to oscillatory motion. As it expands, it pushes the surrounding fluid horizontally. This gives the oscillation *inertia* which increases with increase in the horizontal wavelength of the oscillation and hence the oscillation frequency decreases. Setting up and solving equations for the oscillation, a value of frequency is obtained. A real frequency suggests oscillatory motion whereas an imaginary frequency suggests convective instability so that the displaced fluid element continues to move radially outward.

6.3 Surface Gravity Waves

Consider the stellar surface. Since the length under consideration is small, the stellar surface can be assumed to be flat, horizontal. Assume uniform gravity in downward direction. Since fluid is assumed to be incompressible, we get that density perturbation is zero and hence divergence of any velocity of the fluid is zero. Surface gravity waves are waves similar to those on the surface of water. The frequency of these waves depends only upon gravitational acceleration and the wavelength. In general, surface gravity waves exist at any surface where density has a discontinuity. These oscillation are called gravity driven modes or *g modes.*

7 Oscillation in Stars

7.1 Modes

- **g modes** are gravity driven modes discussed above. They are non-radial modes and exist in the interior regions of the star. As the number of radial nodes increase, frequency of g-modes decreases.
- **p modes** are pressure driven modes and can be radial or otherwise. These modes exist in the outer regions of the star. As the number of radial nodes increase, frequency of g-modes increases.
- In addition to the g and p modes, there may also exists an **f mode** which is intermediate between p and g modes.

7.2 Quantum Numbers

Each mode is characterised by a set of three quantum numbers: *n, l* and *m.*

- n is called the overtone of the mode and is equal to the number of radial nodes.
- l is called the degree of the mode and is equal to the number of surface nodes. $0 \leqslant l \leqslant n-1$
- m is called the azimuthal order of the mode. Out of the l surface nodes $/m/$ nodes are longitudinal nodes. $|m| \leqslant l$. All modes with $m \neq 0$ are travelling modes. By convention, modes travelling prograde have $m > 0$.

7.3 Small and Large separation

- Large separation is equal to inverse of the time needed for the wave to travel from the stellar surface to its interior and back. Large separation depends upon the radius and hence can be used to find the mass of a main sequence star. Stars with larger mass have smaller value of large separation.
- Small separation is affected by the core composition (larger mass fraction of hydrogen in the core implies larger value of small separation) and hence gives information about the stellar interior.

7.4 Driving Mechanisms

- **Heat Driven** is similar to mechanism of a heat engine. During compression phase of the oscillation, a layer (usually radial) of star gains energy. This energy is then used to drive the expansion phase.
- **Opacity Driven (κMechanism)** is dominant in the different ionisation layers within the star. The steps involved are as follows.
  - Neutral atoms (usually H and He) block radiation resulting in increase in pressure due to which the star swells.
  - As the star swells, the elements ionise and become less opaque and hence the radiation can pass through.
  - The star contracts and the H and He ions recombine.
- ϵ **Driven** modes are the oscillation of the star due to variations in the energy (ϵ) produced in the star's core.

- **Stochastically Driven**: In convective zones of stars, close to the stellar surface, the moving fluid elements reach sonic speeds and act as sources of acoustic noise. If the noise contains fundamental frequencies of the star, energy gets drawn into these modes through resonance. This mechanism is found in stars with a convective envelope and mass up to 1.5 solar masses.

8 List of Other Topics Studied

The following topics were also studied during the winter but could not be elaborated here due to documentation constraints.

- **Stellar evolution** was read and understood from both *Prialnik* and *Astroseismology*
- Equation of **radiative heat transfer**
- **Opacity**: different causes and properties
- Nuclear reactions in star including **P-P** Chain, **CNO** Cycle and **Tripple Alpha** Reaction and their properties.
- **Polytropic Model**
- **Chandrasekhar Limit**
- **Instability of Thin Shells**
- General **hydrodynamics** of stars (from *Astroseismology*)
- **Shell scripting** in Linux (from online tutorials) and codes written in same

Part III

Implementation

9 YREC and Support Package Overview

The YREC code uses mass as the independent variable. It divides the star into different layers (called meshing). The number and thickness of layers are decided by constrains on the relative difference in the values various quantities in any two consecutive shells. Then the model is evolved in time. The time step needs to be customised by the user so that both accuracy and efficiency are maintained. After this redistribution of the shell points is done. The code calculates at each point in time, several useful quantities like radius, luminosity, mass fraction of hydrogen in core, degree of degeneracy etc. The code also gives quantities like pressure, density etc. at each shell point in the star. YREC needs a considerable amount of tuning for each run to go through in an optimum manner and hence it is not possible to do this task manually. YREC support package was developed as a part of this project for the same reason. The details of the package are given below.

The models were generated using YREC in a directory hierarchy, each level of directories giving gradation over a particular parameter, so as to facilitate proper sorting of the models. For each grid point, the star is evolved from birth up to Helium flash or till it settles down in the Red Lump (depending on the initial mass of the star). Each grid point is associated with a Physics file (.phy) and a Control File (.ctl). The physics file contains values of tolerences for various parameters, to be used during the YREC execution. These values need to be adjusted for optimum time stepping and meshing. If the tolerences are too tight, the models are accurate, but the computation time may become impractical. On the other hand, if the tolerences are very weak, the code may not converge and crash. The control file contains ending criterion for the run and the condition and frequency for writing out models.

For each grid point, the star is evolved from birth and models are written out as per requirement. After the run has completed, it is checked for crashes or absense of conversion. There may be cases where the models written out are not realistic even though the code did not crash. This usually when the code does not converge well and just about manages to pass the success check. For this purpose, a sanity check is performed on runs which pass the success check.

Usually from techniques other than asteroseismology, the rough location of the observed star is known on a g vs T (surface gravitational acceleration vs surface effective temperature). Hence for the purpose of fitting the star into the grid, only the models lying within this *g-T box* are used. Hence, after writing out the models, only those models, in any, which lie inside this specified box are automatically pulled out.

After this, oscillation frequencies were computed for the relevant models using frequncy computation code written by Dr. H. M. Antia and Echelle Diagram is plotted to identify the various modes of oscillation including the mixed modes.

A flow chart giving the overview of the entire process[7] is given in the last section of this report

10 Package Execution Guidelines

10.1 Generating Models

- Execute 'runyrec' with appropriate arguments. Use 'runyrec -q' for more details.
- 'runyrec' will create a hierarchical directory structure inside 'MODELS_LOC' specified by the user in the user configurable part of 'runyrec'.
- The time taken for the run to complete is written out in 'stagetime'

10.2 Success Check

In rest of the document, unless otherwise specified, all actions are to be done in the concerned innermost directory of the directory hierarchy.

- If any of the several runs crashes, for whatever reason, 'runyrec' will exit and a <stage_name>.flg file will be created. If a stage, which was unsuccessful during a prior run becomes a success, then the corresponding .flg file is removed.
- Details regarding the success of each of the stages is written out in a crash report named 'main.crash'.
- To perform a crash check on a particular stage run 'crash_check <stage_name>'. It will produce / delete .flg file for the correcponding run and generate '<stage_name>.crash' as crash report and also updates 'main.crash' suitably.
- At any point in time, the user can check all stages of a particular run by running 'checkall'.

10.3 Tolerences in .phy File

- 'hpttolchk.sh' and 'htolerchk.sh' can be used to check which tolerence value is being hit or violation of which criterion has caused the run to fail.
- Two files in the source code 'crrect.f' and 'hpoint.f' were edited to print flags to a file 'fort.52' in the working directory during execution of each stage.
- 'hpttolchk.sh' counts how many times each HPTTOL flag was raised in 'fort.52'.
- 'htolerchk.sh' counts how many times each HTOLER flag was raised in 'fort.52'.
- This gives empirical information pertaining to which tolerances and criteria are relevant for speeding up execution or making finer grids at a certain stage.
- The required tolerences can be adjusted in the .phy file for the concerned run(s).

[7]The entire process from understanding the required theory, implementing the grid, computing frequencies, fitting observed stars to finally getting the internal parameters of the star is a very long process. This project handles mainly the theoretical and computational aspects. Further work needs to be done on data fitting and parameter extraction.

10.4 Sanity Check

- After verifying that the run has gone through successfully, the next step is to perform a sanity check on the models.
- Execute ‘checkrun’. It will generate a GNUPlot file named ‘check_plot’. Load this into GNUPlot. This will generate plots which will help in performing the sanity check.

10.5 Interest Check

- After a run has cleared sanity check also, the next step is to check if the run passes through the required g-T box and to know how many models were written out inside the box.
- Run ‘makebox’. This script runs on ‘gtplot’ file generated by ‘runyrec’ and generates a GNUPlot file named ‘gtbox’. This plot shows whether or not the track passes through the required g-T box, in a multiplot style. The x and y range of the box can be editted within the ‘makebox’ script.
- If the track seems to pass through the box satisfactorily, run ‘box’. This will output the number of models lying within the g-T box (as specified in the user configurable part of ‘box’). It will also generate a file called ‘box_models’ which will contain the ages of all the models lying within the box.

10.6 Computing Frequencies

- Pick a model lying within the box and run ‘hmarun_editting’. This script is a version of ‘hmarun’ with few parameters changed to suit red giants.
- Further editting and fine tuning will probably be needed.
- This will generate a .freq file.
- Echelle diagram for the .freq file may be plotted using ‘ech’.
- Give ‘ech’ without any arguments to see the usage.

10.7 Other

The entire package including ‘runyrec’ and all the support scripts are present in ‘~/data/research/YREC/BIN’ directory of ‘Vega (IP - 158.144.42.72)’ Each script is well documented and the user is urged to go through the introduction section for each script for a detailed understanding of the scripts.

A spreadsheet named crash_table.xls containing details of the models made so far is stored in ‘/core/research/-giants’.

For any clarifications on the procedure or any of the scripts in specific, please contact the author of the concerned script.

11 Optimising .phy and .ctl Files

11.1 .ctl File

The main issue in optimizing the ctl files was to carefully set the ENDAGE criteria especially while ascending the red giant branch. The other issue which occured in case of the higher masser (>=2.1m_sun) was that the convective envelope disappeared shortly before the brgb. This was corrected by changing the SENV0A to 5D-4.

The following are the changes that were made to the ctl files in the models made by us:

m1800 The endage criteria are turned off and the stages by the nmodels criteria for z1678 through z1907. (The extrapolated fit of the endages was not very good for this mass)

m2000 uses the separate masterfile #2000. z1642->z1907: zams_tams NMODLS=150(not 75) (These models were made before we decided to optimize roughly for a range of masses instead of optimizing tightly each mass)

m2100 uses the masterfile #2100 for z1474 to z1573 and #2000 otherwise. also z1411: tams_brgb ENDAGE=0.796

11.2 .phy File

Of the various optimization handles available in the phy files, the HTOLER, HPTTOL and ATIME values were mainly used to optimize the execution. Here the source code files crrect.f and hpoint.f were edited to print a flag to the output file fort.52 in the models directory whenever one of the convergence criteria were hit. The scripts hpttolchk.sh and htolerchk.sh when called in that directory scan the file fort.52 and print the number of instances when each flag was raised. These statistics may be used additionally to decide which criteria may need to be tightened or relaxed in a particular stage.

The following are the changes that were made to the phy files in the models made by us (default values without changes given in parentheses):

m1600

- z1411-z1642,z1714-z1907: brgb_rgb1 HPTTOL(7)=0.0001(not 0.001)
- z1541: brgb_rgb1 ATIME(7)=0.1(not 0.06)

m1700

- z1411: brgb_rgb1 HPTTOL(6)=0.003(not 0.0001)
- z1411,z1539,z1573: rgb3_trgb: ATIME(7)=0.1(not 0.06) z1442-z1907: brgb_rgb1: HPTTOL(7)=0.0001(not 0.001)
- z1828: rgb2_rgb3,rgb3_trgb: 0.1(not 0.04,0.06)

m1800

- z1411-z1642: brgb_rgb1: HPTTOL(7)=0.0001(not 0.001)
- z1714: brgb_rgb1: HPTTOL(7)=0.0001(not 0.001) ATIME(7)=0.005(not 0.01) rgb1_rgb2: ATIME(7)=0.01(not 0.03) rgb2_rgb3: ATIME(7)=0.1(not 0.04) rgb3_trgb: ATIME(7)=0.1(not 0.06)

m1900

- z1411-z1907: brgb_rgb1: HPTTOL(7)=0.001(not 0.0001)
- z1642: rgb1_rgb2: ATIME(7)=0.05(not 0.01)

m2000 This directory was initially optimized too tight and therefore has too many deviations from the default values to be significantly useful. However the models are all alright.

12 Images

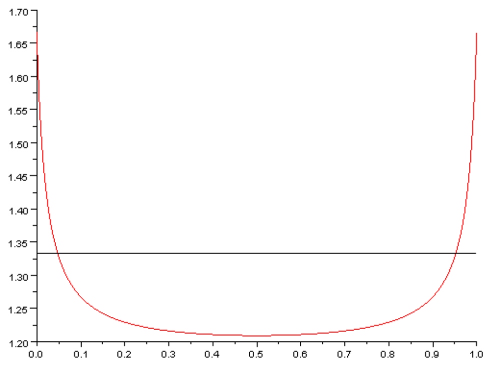
Figure 1 The first figure shows degree of ionisation on x axis and γ_a on y axis. The second plot is HR diagram for a two solar mass star with $\log(T/T_\odot)$ on x axis and $\log(L/L_\odot)$ on y axis.

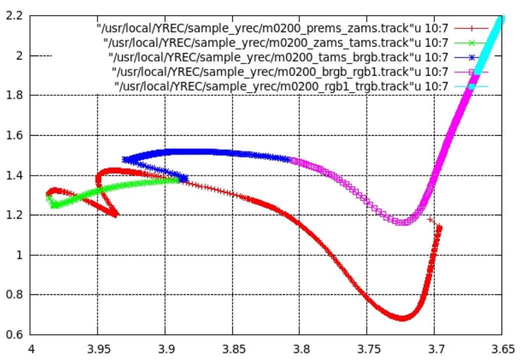
Figure 2 The YREC code was evolved for stars with mass $2M_\odot$ and $M_\odot$. The HR diagram for $2M_\odot$ star is shown as an example. The red part if the Pre-Main Sequence stage, the green line is the Main Sequence. The magenta and cyan tracks represent the lower and upper parts of red giant branch respectively. The track file generated was observed and an attempt was made to draw analogies between the observations and theory learnt.

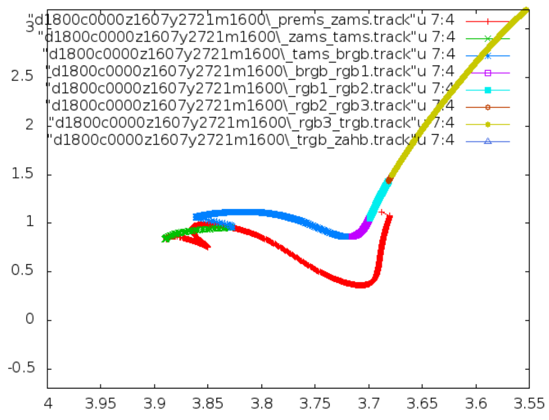
Figure 3 HR Diagram of a 1.6 Solar mass star with mixing length 1.8, no core overshoot, Z=.01607, Y=.2721

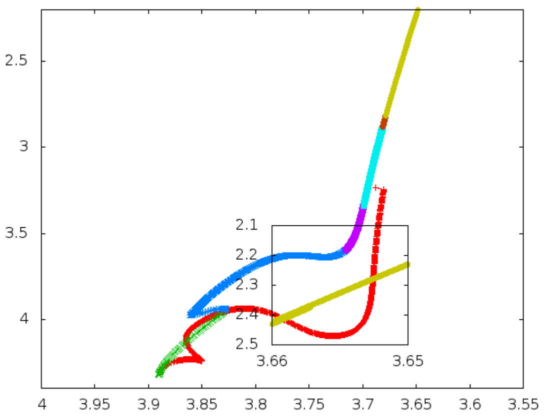
Figure 4 The figure shows a plot of logarithm of gravitational acceleration g on x axis vs logarithm of effective temperature T_{eff} on y axis and showing the region of our interest in the inset

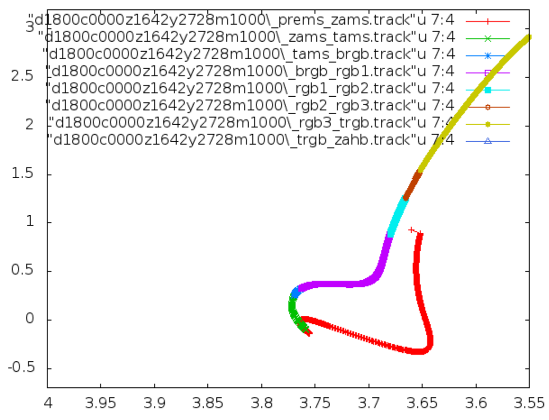
Figure 5 HR Diagram of the Sun



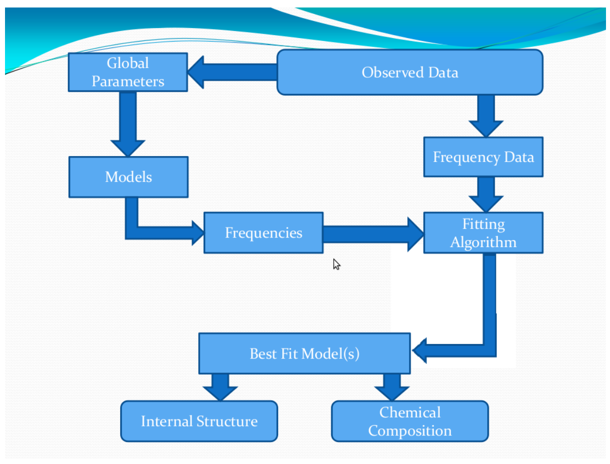








for the same star



13 Appendix

13.1 Appendix A: Central Script which Calls YREC

```
#!/bin/bash

#############################################################################


# This is the main script responsible for running YREC
# Please go through usage for how-to-use instructions.
# Uses ‘crash_check’

# Authors*: Jayant Thatte, Tamaghna Hazra, December 2011.
# *Basic framework for this script is borrowed from ‘runcesam’
```

```
################################################################################


scrname=`basename $0`
version="2.6.0"

##################### User Configurable Part Begins
   ##########################

function config
{
    # The script creates a directory called models inside $MODELS_LOC
       directory.
    # If models directory already exists, the existing directory is used.
    # All models made by this script are stored in $MODELS_LOC/models

    MODELS_LOC=/core/research/giants

    # The script uses master files .batch, .ctl and .phy
    # The location of these files should be given $MASTER_LOC
    # $MASTER_LOC must have a .batch file and directories ctl and phy
    # All .ctl files should be in this ctl folder. All .phy files should be
       in this phy folder.

    MASTER_LOC=/core/research/giants/master_files

    if [[ protodonfl -eq 1 ]]; then
        echo "Using $protodonfile instead of default master folder."
        echo "Remove -p token to use default master folder."
        MASTER_LOC=$protodonfile
    fi

    pathhom=/data/research/YREC/DATA/PREMS_GS98
    febyhbin=/data/research/bin
    MDEFAULT=m2000
}
###################### User Configurable Part Ends
   ###########################

function main
{
    # This is the collection of the main executable lines of the script
    # The script is modular in structure and from this function the basic
    # steps of operation are carried out by calling the relevant functions.

    if [ $# -eq 0 ]; then  # Script invoked with no command-line args
        usage          # Exit and explain usage, if no argument(s) given.
    fi

    ############### Initialise all the flags
    flag_initialise

    ############### Parse the arguments passed to the script
    parse_command_args "$@"
    shift $?

    ############### Check the consistency of the chosen options
    optconsist
    ############### Making a hierarchical directory structure
    config
    make_dir
    prepare_directory

    choose_homfile
    prepctl
}

function usage
{
    # This function describes the usage of the script in brief.
    # Called from: main, parse_command_args, get_option_argument
    # Calls to: none

        if [ $# -gt 0 ]; then
           echo -e "${scrname}: $*\n" 1>&2
           exit 1
        fi

        less<< EOHD

    The $scrname script is intended for generating stellar models of a given
    mass from the ZAMS to the evolved stages specified in terms of age in
       Myrs,
    central hydrogen abundance, or logarithm of effective temperature.
    The CESAM evolutionary code is used to generate the models. The final
       output
    files are: "{prefix}aA.AA.osc" for models specified by age = A.AA x 1000
        (Myrs),
    or          "{prefix}xX.XX.osc" for models specified by X_c = X.XX,
    or          "{prefix}tT.TT.osc" for models specified by log Teff = log(T.
       TT),
    where {prefix} is by default: "mM.MM" if mass = M.MM in solar units"

    Usage: $scrname options

    Common options are (equivalent long options in brackets):
    Mass:
      -m (--mass) <arg> : set mass of model
    Evolution:
      -A (--age)            : express age of desired models in Myrs (not
         implemented)
      -X (--X_C)            : express age of desired models in central H (not
          implemented)
      -T (--teff)           : express age of desired models in log Teff (not
         implemented)
      -i (--initial) <arg> : initial age/X_c/log Teff (not implemented)
      -f (--final)   <arg> : final age/X_c/log Teff (not implemented)
      -Z (--ZAMS)           : create a ZAMS model (not implemented)
    Chemical composition:
      -x (--X0)   <arg> : initial abundance of H
      -y (--Y0)   <arg> : initial abundance of He
      -z (--ZSX0) <arg> : initial value of (Z/X)
      -F (--Fe/H) <arg> : initial value of [Fe/H]
    Convection:
      -a (--alpha) <arg> : mixing length in terms of H_p
      -c (--core)  <arg> : core overshoot in terms of H_p
    Diffusion:
      -D (--diffusion) <arg> : enable or disable diffusion
    Filenames etc.:
      -p (--protodon) <file> : use master files stored in /core/research/
         giants/basu_master_files
      -d (--datfile)  <file> : use <file> as .dat file to continue evolution
      -u (--use)             : use the existing .phy and .ctl files instead
         of copying from master files
      -q (--query)           : give a preview of parameters of models to be
         created
      -ng (--no_graphs)      : do not write out any plotting related files (
         gtplot, hrplot etc.)
      -s (--start_at)        : start the run from a custom stage
      -e (--end_at)          : end the run at a custom stage

EOHD
    exit 1
}

function flag_initialise
{
    # Initialise the flags for various options
    # Called from: main
    # Calls to: none

    massfl="0"

    evolfl="0"
    age_fl="0"
    xc__fl="0"
    tefffl="0"
    ageifl="0"
    ageffl="0"
    agesfl="0"

    chemfl="0"
    febyhfl="0"
    xzerofl="0"
    yzerofl="0"
    zzerofl="0"

    alphafl="0"
    ovshtsfl="0"

    protodonfl="0"
    usefl="0"
    newfl="0"
    queryfl="0"
    execfl="0"
}

function parse_command_args ()
{
    # This function parses the command-line arguments
    # This is imported from: http://www.gnu.org/server/source/diffmon
    # Thanks: Noah Friedman <friedman@prep.ai.mit.edu>
    # Called from: main
    # Calls to: get_option_argument, flag_dbl, usage

    local orig_number_options=$#

    # If you add new options be sure to change the wildcards below to make
    # sure they are unambiguous (i.e. only match one possible long option)
    # Be sure to show at least one instance of the full long option name to
    # document what the long option is canonically called.
    # Long options which take arguments will need a '*' appended to the
# canonical name to match the value appended after the '=' character.
while [ $# -gt 0 ]; do
    case z$1 in
        z-A | z--age | z--AGE* )        # To be processed.
            flag_dbl "$age_fl" "$1"
                shift
            age_fl="1"
                evolfl="1"
            echo Models will be charcterized by age.
        ;;
        z-a | z--alpha* | z--ALPHA* )
                flag_dbl "$alphafl" "$1"
                get_option_argument ALPHA "$1" "$2"
                shift $?
                alphafl="1"
                echo "alpha value = $ALPHA"
        ;;
        z-c | z--core-overshoot* | z--core* | z--OVSHTS* | z--ovshts* )
                flag_dbl "$ovshtsfl" "$1"
            get_option_argument OVSHTS "$1" "$2"
            shift $?
            ovshtsfl="1"
            echo "Core Overshoot = $OVSHTS"
        ;;
        z-E | z--exec* | z--EXEC* )                 # To be processed.
            flag_dbl "$execfl" "$1"
            newexec=`which $2`
            if [[ "$?" -eq "0" ]]; then
                echo "Executable program $newexec found successfully."
            else
                echo "Executable program $2 not found in your PATH.
                   Exiting"
                exit 1
            fi
            get_option_argument EXEC "$1" "$newexec"
            shift $?
            execfl="1"
        ;;
        z-e | z--END_AT* )
            flag_dbl "$endfl" "$1"
            get_option_argument stop_evol_at "$1" "$2"
            shift $?
            endfl="1"
        ;;
        z-ng | z--NO_GRAPHS* )
            flag_dbl "$ngfl" "$1"
            shift
            ngfl="1"
        ;;
        z-F | z--Fe/H* | z--febyh* )
            flag_dbl "$febyhfl" "$1"
            get_option_argument febyh "$1" "$2"
            shift $?
            febyhfl="1"
            chemfl="1"
        ;;
        z-f | z--fe/h* | z--Febyh* )     # To be processed.
            flag_dbl "$febyhfl" "$1"
            get_option_argument febyh "$1" "$2"
            shift $?
            febyhfl="1"
```

```
                chemfl="1"
            ;;
            z-i | z--initial-age* | z--initial* )     # To be processed.
                flag_dbl "$ageifl" "$1"
                get_option_argument agei "$1" "$2"
                shift $?
                ageifl="1"
            ;;
            z-m | z--mass* | z--MASS* | z--MTOT* | z--mtot* )
                flag_dbl "$massfl" "$1"
                get_option_argument MTOT "$1" "$2"
                shift $?
                massfl="1"
                echo "Total mass = $MTOT"
            ;;
#           z-N | z--new* | z--NEW* )      # To be processed.
#               flag_dbl "$newfl" "$1"
#               shift
#               newfl="1"
#           ;;
            z-p | z--protodonfile* | z--proto* )     # To be processed.
                flag_dbl "$protodonfl" "$1"
#               get_option_argument protodonfile "$1" "$2"
#               shift $?
                shift
                protodonfl="1"

                protodonfile=~/giant/basu_master_files
            ;;
            z-q | z--query | z--q* )
                queryfl="1"
                shift
            ;;
#           z-s | z--age-step* | z--step* )    # To be processed.
#               flag_dbl "$agesfl" "$1"
#               get_option_argument agestep "$1" "$2"
#               shift $?
#               agesfl="1"
#           ;;
            z-u | z--use* )
                flag_dbl "$usefl" "$1"
                shift
                usefl="1"
            ;;
            z-v* | z--version* )
                echo "$scrname version $version"
                exit 0
            ;;
            z-x | z--x0* | z--x_0* | z--X_0* | z--X0* )
                flag_dbl "$xzerofl" "$1"
                get_option_argument X0 "$1" "$2"
                shift $?
                xzerofl="1"
                echo "x value = $X0"
            ;;
            z-y | z--y0* | z--y_0* | z--Y_0* | z--Y0* )
                flag_dbl "$yzerofl" "$1"
                get_option_argument Y0 "$1" "$2"
                shift $?
                yzerofl="1"
                chemfl="1"
            ;;
            z-z | z--zsx0* | z--zsx_0* | z--ZSX_0* | z--ZSX0* | z--Z/X* )
                flag_dbl "$zzerofl" "$1"
                get_option_argument ZSX0 "$1" "$2"
                shift $?
                zzerofl="1"
                echo "z value = $ZSX0"
            ;;
            z-s | z--START_AT* )
                flag_dbl "$startfl" "$1"
                get_option_argument start_evol_at "$1" "$2"
                shift $?
                startfl="1"

            ;;
            z-- )
                queryfl="1"
                shift
                break
            ;;
            z-* )
                usage "\`${1}' is not a valid option."
                exit 1
            ;;
            * )
                echo -e "At least one invalid option chosen.\nExiting."
                exit 1
            ;;
        esac
    done
    if [[ "$startfl" -ne "1" ]]; then
        start_evol_at="prems"
    fi
    echo "start at = $start_evol_at"
    if [[ "$endfl" -ne "1" ]]; then
    stop_evol_at="tahb"
    fi
    echo "end at = $stop_evol_at"

    # Return number of shifted arguments so calling function can shift
    # appropriate amount.
    return $[ orig_number_options - $# ]
}

function get_option_argument ()
{
    # Usage: get_option_argument VARIABLE OPTION ARG {OPTIONAL}
    #    where VARIABLE is shell variable that will be set to the value ARG.
    #    Long option syntax is `--foo=bar' or `--foo bar'.  3rd argument ARG
    #    won't get used if first long option syntax was used.  If 4 arg
    #    OPTIONAL is non-empty, option isn't required to have an argument;
        if
    #    the argument is missing, VARIABLE is set to the empty value.
    # Returns number of positions caller should shift

    # This is imported from: http://www.gnu.org/server/source/diffmon
    # Thanks: Noah Friedman <friedman@prep.ai.mit.edu>
    # Called from: parse_command_args
    # Calls to: usage

     local variable="$1"
     local option="$2"
     local arg="$3"
     local arg_optional="$4"

    # All long options must be at least 3 characters long (--o*), whereas
    # short options are only two chars (-o) and arguments are always
    # separate.
    if [ ${#option} -ge 3 -a "z${option#*=}" != "z${option}" ]; then
        arg="${option#*=}"  # Strip off anything before and including ‘=’
           char
        #eval ${variable}=\’"${arg}"\’
        eval ${variable}="${arg}"
        return 1
    else
        if [ -z "${arg}" -a -z "${arg_optional}" ]; then
            usage "option \‘${option}’ requires argument."
        fi
        #eval ${variable}=\’"${arg}"\’
        eval ${variable}="${arg}"
        return 2
    fi
}

function flag_dbl
{
    # This function traps options chosen twice by error.
    # Called from: parse_command_args
    # Calls to: none

        if [[ "$1" -ne "0" ]]; then
            echo -e "\nOption $2 has been specified twice. This is ambiguous
               .\nExiting."
            exit 1
        fi
}

function optconsist
{
    # This function checks the consistency of different flags and checks the
    # validity of the argument values.
    # Called from: main
    # Calls to : checkval, checkflag

    # Set a variable for true/false checking. We shall use it for some
       options
    tfvar=‘echo "t T f F"‘

    # Option -m (mass)
       ------------------------------------------------------
    # Mass should be between 1 and 100 Msun
    if [[ "$massfl" -eq "1" ]]; then
        checkval mass $MTOT 0 100
    fi

    # Option -A (age)
       ------------------------------------------------------
    # Age flag is incompatible with X_c or T_eff
    # At least final age should be specified
    if [[ "$age_fl" -eq "1" ]]; then
        checkflag A "X $xc__fl 0" "T $tefffl 0" "f $ageffl 1"
    fi
```

```
# Option -i (start of loop)
   ------------------------------------------------
if [[ "$ageifl" -eq "1" ]]; then
    # Needs either of -A, -X or -T flags and -f and -s flags
    checkflag i "A $age_fl 2" "X $xc__fl 2" "T $tefffl 3 -A or -X or -T"
        "f $ageffl 1" "s $agesfl 1"

    # Initial age/X_c must be lesser/greater than final age/X_c for -A/-
       X options.
    # For -T, this is not imposed
    chfl=0
    if [[ "$age_fl" -eq "1" ]]; then
        checkval "initial age" $agei 0 15000
        chfl=`echo $agei $agef | awk '{ print ($1 < $2) ? "0" : "1" }'`
    elif [[ "$xc__fl" -eq "1" ]]; then
        checkval "initial X_c" $agei 0 1
        chfl=`echo $agei $agef | awk '{ print ($1 > $2) ? "0" : "1" }'`
    fi

    if [[ "$chfl" -eq "1" ]]; then
        echo -e "\nThe supplied values of age or X_c are inconsistent
           with the sense of evolution.\nExiting\n"
        exit 1
    fi
fi

# Option -f (end of loop)
   --------------------------------------------------
if [[ "$ageffl" -eq "1" ]]; then
# Needs either of -A, -X or -T flags
    checkflag f "A $age_fl 2" "X $xc__fl 2" "T $tefffl 3 -A or -X or -T"
    if [[ "$age_fl" -eq "1" ]]; then
        # Final age cannot exceed 15 Gyrs
        checkval "final age" $agef 0 15000
    elif [[ "$xc__fl" -eq "1" ]]; then
        # Final X_c must lie between 0 and 1
        checkval "final X_c" $agef 0 1
    elif [[ "$tefffl" -eq "1" ]]; then
        # Final log Teff must lie between 0 and 5, but allow for bi-
           directional sign
        checkval "final log Teff" $agef -5.0 5.0
    fi
fi

# Option -s (step of loop)
   -------------------------------------------------
if [[ "$agesfl" -eq "1" ]]; then
# Needs either of -A, -X or -T flags and -i and -f flags
    checkflag s "A $age_fl 2" "X $xc__fl 2" "T $tefffl 3 -A or -X or -T"
        "i $ageifl 1" "f $ageffl 1"

    checkval "age step" $agestep 0 100000

    # The step in age must not exceed the total interval of age
    adiff=0
    if [[ "$age_fl" -eq "1" ]]; then
        adiff=`echo $agef $agei $agestep | awk '{ ad=$1-$2-$3; print (ad
            >= -1e-8 ) ? "0" : "1" }'`
    elif [[ "$xc__fl" -eq "1" ]]; then
        adiff=`echo $agei $agef $agestep | awk '{ ad=$1-$2-$3; print (ad
            >= -1e-8 ) ? "0" : "1" }'`
    fi

    if [[ "$adiff" = "1" ]]; then
        echo "\nThe step size of age or X_c is larger than the range of
            evolution.\nExiting.\n"
        exit 1
    fi
fi

# Option -F ([Fe/H] value)
    ---------------------------------------------------
# -F is incompatible with -x, -y or -z
if [[ "$febyhfl" -eq "1" ]]; then
    #checkflag F "x $xzerofl 0" "y $yzerofl 0" "z $zzerofl 0"
    checkflag F "x $xzerofl 0" "z $zzerofl 0"
fi

# Option -x (X0 value)
    -------------------------------------------------------
# -x is incompatible with -F, -z and requires -y
if [[ "$xzerofl" -eq "1" ]]; then
    checkflag x "F $febyhfl 0" "y $yzerofl 1" "z $zzerofl 0"
    checkval "X0" $X0 0 1
    X0=`awk "BEGIN { printf \"%14.6e\", $X0}"`
fi

# Option -y (Y0 value)
    -------------------------------------------------------
# -y is incompatible with -F and requires either -x or -z
if [[ "$yzerofl" -eq "1" ]]; then
    #checkflag y "F $febyhfl 0" "x $xzerofl 2" "z $zzerofl 3 -x or -z"
    checkflag y "F $febyhfl 2" "x $xzerofl 2" "z $zzerofl 3 -x or -z or
        -F"
    checkval "Y0" $Y0 0 1
    Y0=`awk "BEGIN { printf \"%14.6e\", $Y0}"`
fi

# Option -z (ZSX0 value)
    -----------------------------------------------------
# -z is incompatible with -F, -x and requires -y
if [[ "$zzerofl" -eq "1" ]]; then
    checkflag z "F $febyhfl 0" "y $yzerofl 1" "x $xzerofl 0"
    checkval "ZSX0" $ZSX0 0 1
    ZSX0=`awk "BEGIN { printf \"%14.6e\", $ZSX0}"`
fi
# Option -c (Core overshoot)
    -------------------------------------------------
# The core overshoot parameter must be between 0 and 1
if [[ "$ovshtsfl" -eq "1" ]]; then
    checkval "OVSHTS" $OVSHTS 0 1
fi
# Option -N (ignore existing PROTOMODEL.DON)
    -------------------------------
# The option -N is inconsistent with -p or -u
if [[ "$newfl" -eq "1" ]]; then
    checkflag N "p $protodonfl 0" "u $usefl 0"
fi

# Option -p (protodon file)
    ------------------------------------------------
    if [[ "$protodonfl" -eq "1" ]]; then
        # The option -p is inconsistent with -N
        checkflag p "N $newfl 0"

#       # Check for existence of protodonfile
#       if test ! -f $protodonfile ; then
#           echo -e "\nProtoype $protodonfile not found.\nExiting"
#           exit 1
#       fi
#       # Check if protodonfile has the correct format
#       chdon=`head -1 $protodonfile| grep -i -w NL_CESAM > /dev/null ; echo
    $?`
#       if [[ "$chdon" -ne "0" ]]; then
#           echo "\nThe prototype $protodonfile does not conform to the
   standard format.\nExiting."
#           exit 1
#       fi
    fi
    # Option -u (update)
       --------------------------------------------------------
    # The option -u is inconsistent with -N
    if [[ "$usefl" -eq "1" ]]; then
        checkflag u "N $newfl 0"
    fi
}

function checkflag
{
    # This function checks for the dependency and consistency of different
       flags
    # Called from: optconsist
    # Calls to: none

    # The logic in this function is a bit convoluted, but it works! Will try
        to
    # simplify later. Commenting is postponed till then.

    errorfl=0
    efl=1
    testarg=$1
    shift
    for arg in "$@"
    do
        set -- $arg
        if [[ "$3" -gt "1" ]];then
            if [[ "$2" -eq "1" ]]; then
                efl=0
            fi
                if [[ "$3" -gt "2" ]];then
                errorfl=$efl
                if [[ "$errorfl" -eq "1" ]]; then
                    break
                fi
            fi

        elif [[ "$2" -ne "$3" ]]; then
            errorfl=1
            break
        fi
    done
    if [[ "$errorfl" -eq "1" ]]; then
        echo
        if [[ "$3" -eq "0" ]];then
            echo "Option -$testarg is incompatible with option -$1"
        elif [[ "$3" -eq "1" ]];then
            echo "Option -$testarg requires option -$1 to be specified."
        elif [[ "$3" -gt "1" ]]; then
            shift
            shift
            shift
            echo "Option -$testarg requires either option $@ to be specified
               ."
        fi
        echo -e "Exiting.\n"
        exit 1
    fi
}

function checkval
{
    # This function checks the validity of the values of the numerical
    # arguments passed with the flags.
    # Called from: optconsist
    # Calls to: none
    # No. of arguments: 4

    # First check for any leading or trailing garbage with the passed
       argument
    testval="$2"
    lt=${#testval}
    let "lt-=1"
    declare -a junkar
    local junkar=( ‘echo "= _ \~ \| / \\\\\" \\\\’ \\\\\\\‘ \< \> ? \$ \( \)
        \{ \} \[ \]"‘ )
    local fcr=${testval:0:1}
    local lcr=${testval:$lt:1}
    for i in ${junkar[@]}
    do
        if [[ "$fcr" = "$i" || "$lcr" = "$i" ]]; then
            echo -e "The supplied argument of $1 cannot be understood.\
               nExiting.\n"
            exit 1
        fi
    done

    chfl=0
    chfl=‘echo $2 $3 $4 |awk ’{ print ($1 >= $2 && $1 <= $3) ? "0" : "1" }’‘
    if [[ "$chfl" -eq "1" ]]; then
        echo "\nThe $1 value must lie between $3 and $4.\nExiting.\n"
        exit 1
    fi
}

function make_dir
{
    cd
    cd $MODELS_LOC
    if [[ ‘echo $?‘ -eq 0 ]]; then
        echo "Directory $MODELS_LOC found. Models will be created here."
    else
        echo -e "$MODELS_LOC is not a valid directory.\nExiting."
    exit 1
fi
dir_exist models

# Extracting the correct master directory
if [[ $massfl -gt 0 ]]; then
    MTOT_str=`echo $MTOT | awk 'BEGIN {FS=" "} {printf "%6.4f",$1/10.0}'
         | awk 'BEGIN {FS="."} {printf "%s", $2}'`
else
    MTOT_str=`cat $MASTER_LOC/$MDEFAULT/ctl/prems_zams.ctl | grep -w "
        RSCLM" | awk 'BEGIN {FS="="} {printf "%6.4f",$2/10.0}' | awk '
        BEGIN {FS="."} {printf "%s", $2}'`
    MTOT=`cat $MASTER_LOC/$MDEFAULT/ctl/prems_zams.ctl | grep -w "RSCLM"
         | awk 'BEGIN {FS="="} {printf "%6.4f",$2}'`
fi

var=`echo $MTOT | awk 'BEGIN {FS=" "} {printf "%3.1f",$1/10.0}' | awk '
   BEGIN {FS="."} {printf "%s",$2}'`
if [[ $var -eq 0 ]]; then
    var=1
fi

difmin=100000
for i in `ls $MASTER_LOC |grep m[0-9][0-9]`
do
    j=`basename $i | cut -c 2-5`
    dif=`awk -v m=$MTOT_str -v j=$j 'BEGIN { x=m-j; print (x>=0) ? x : -
        x}'`
    chosen=`awk -v dif=$dif -v min=$difmin -v j=$j -v ch=$chosen 'BEGIN
        { print ( dif < min ) ? j : ch}'`
    difmin=`echo $difmin $dif | awk '{print ($1<$2)?$1:$2}'`
done

# Making Mixing length directory
if [[ $alphafl -gt 0 ]]; then
    ALPHA_str=`echo $ALPHA | awk 'BEGIN {FS=" "} {printf "%6.4f",$1
        /10.0}' | awk 'BEGIN {FS="."} {printf "%s", $2}'`
    dir_exist d$ALPHA_str
else
    ALPHA_str=`cat $MASTER_LOC/m$chosen/ctl/prems_zams.ctl | grep -w "
        CMIXLA(1)" | awk 'BEGIN {FS="="} {printf "%6.4f",$2/10.0}' | awk
        'BEGIN {FS="."} {printf "%s", $2}'`
    dir_exist d$ALPHA_str
    ALPHA=`cat $MASTER_LOC/m$chosen/ctl/prems_zams.ctl | grep -w "CMIXLA
        (1)" | awk 'BEGIN {FS="="} {printf "%6.4f",$2}'`
fi

# Making Core Overshoot directory
if [[ $ovshtsfl -gt 0 ]]; then
    OVSHTS_str=`echo $OVSHTS | awk 'BEGIN {FS=" "} {printf "%6.4f",$1}'
        | awk 'BEGIN {FS="."} {printf "%s", $2}'`
    dir_exist c$OVSHTS_str
else
    OVSHTS_str=`cat $MASTER_LOC/m$chosen/phy/prems_zams.phy | grep -w "
        ALPHAC" | awk 'BEGIN {FS="="} {printf "%6.4f",$2}' | awk 'BEGIN {
        FS="."} {printf "%s", $2}'`
    dir_exist c$OVSHTS_str
    OVSHTS=`cat $MASTER_LOC/m$chosen/phy/prems_zams.phy | grep -w "
        ALPHAC" | awk 'BEGIN {FS="="} {printf "%6.4f",$2}'`
fi
```

```
    #Getting x and z from given febyh value
    if [[ $febyhfl -gt 0 ]]; then
        ZSX0=`${febyhbin}/febyh -f ${febyh} | awk 'BEGIN {FS=" "} {if (NR
            ==3) print $3}'`
        X0=`${febyhbin}/febyh -f ${febyh} | awk 'BEGIN {FS=" "} {if (NR==3)
            print $1}'`
        echo "X0 = "$X0",Z0 = "$ZSX0"as computed from febyh value"
        xzerofl=1
        zzerofl=1
    fi

    # Making Z directory
    if [[ $zzerofl -gt 0 ]]; then
        ZSX0_str=`echo $ZSX0 | awk 'BEGIN {FS=" "} {printf "%6.4f",$1*10.0}'
            | awk 'BEGIN {FS="."} {printf "%s", $2}'`
        dir_exist z$ZSX0_str
    elif [[ $xzerofl -gt 0 ]]; then
        ZSX0_str=`echo 1 $X0 $Y0 | awk 'BEGIN {FS=" "} {printf "%6.4f",($1-
            $2-$3)*10.0}' | awk 'BEGIN {FS="."} {printf "%s", $2}'`
        dir_exist z$ZSX0_str
        ZSX0=`echo 1 $X0 $Y0 | awk 'BEGIN {FS=" "} {printf "%6.4f",($1-$2-$3
            )}'`
    else
        ZSX0_str=`cat $MASTER_LOC/m$chosen/ctl/prems_zams.ctl | grep -w "
            ZENV0A" | awk 'BEGIN {FS="="} {printf "%6.4f",$2*10.0}' | awk '
            BEGIN {FS="."} {printf "%s", $2}'`
        dir_exist z$ZSX0_str
        ZSX0=`cat $MASTER_LOC/m$chosen/ctl/prems_zams.ctl | grep -w "ZENV0A"
            | awk 'BEGIN {FS="="} {printf "%6.4f",$2}'`
    fi

    # Fetching X value
    if [[ $xzerofl -eq 0 ]]; then
        if [[ $zzerofl -gt 0 ]]; then
            X0=`echo 1 $ZSX0 $Y0 | awk 'BEGIN {FS=" "} {printf "%6.4f",$1-$2
                -$3}'`
        else
            X0=`cat $MASTER_LOC/m$chosen/ctl/prems_zams.ctl | grep -w "
                XENV0A" | awk 'BEGIN {FS="="} {printf "%6.4f",$2}'`
        fi
    fi

    # Making Y directory
    Y0=`echo $X0 $ZSX0 | awk 'BEGIN {FS=" "} {printf "%6.4f",1.0-$1-$2}'`
    Y0_str=`echo $Y0 | awk 'BEGIN {FS="."} {printf "%s",$2}'`
    dir_exist y$Y0_str

    # Making Mass directory
    dir_exist m$MTOT_str
}

function dir_exist
{
    new_dir_fl=0
    ls -l | grep -w $1 > /dev/null
    if [[ `echo $?` -eq 0 ]]; then
        cd $1
        if [[ `echo $?` -eq 0 ]]; then
            echo -e "Directory \"$1\" already exists. Using the existing
                directory."
        else
```

```
            echo -e "Directory \"$1\" does not exiss. Creating directory \"
               $1\" for further use."
            mkdir $1
            cd $1
        fi
    else
        echo -e "Directory \"$1\" does not exist. Creating directory \"$1\"
           for further use."
        new_dir_fl=1
        mkdir $1
        cd $1
    fi
}

function copy_files
{
    if [[ $usefl -eq 0 ]]; then
        echo "Copying the required master files into the working directory."
        if [[ $MTOT_str = 2100 ]]; then
            cp -r $MASTER_LOC/#2100/ctl/${1}.ctl ctl/${1}.ctl
            cp -r $MASTER_LOC/#2100/phy/${1}.phy phy/${1}.phy
            echo "Files copied from ‘echo $MASTER_LOC/#2100/‘"
        else
            if [[ $MTOT_str = 2000 ]]; then
                cp -r $MASTER_LOC/#2000/ctl/${1}.ctl ctl/${1}.ctl
                cp -r $MASTER_LOC/#2000/phy/${1}.phy phy/${1}.phy
                echo "Files copied from ‘echo $MASTER_LOC/#2000/‘"
            else
                if [[ $MTOT_str = 2200 ]]; then
                    cp -r $MASTER_LOC/#2200/ctl/${1}.ctl ctl/${1}.ctl
                    cp -r $MASTER_LOC/#2200/phy/${1}.phy phy/${1}.phy
                    echo "Files copied from ‘echo $MASTER_LOC/#2200/‘"
                else
                    cp -r $MASTER_LOC/m$chosen/ctl/${1}.ctl ctl/${1}.ctl
                    cp -r $MASTER_LOC/m$chosen/phy/${1}.phy phy/${1}.phy
                    echo "Files copied from ‘echo $MASTER_LOC/m$chosen/‘"
                fi
            fi
        fi

        #cp -r $MASTER_LOC/mchck/ctl/${1}.ctl ctl/${1}.ctl
        #cp -r $MASTER_LOC/mchck/phy/${1}.phy phy/${1}.phy
        #echo "Files copied from ‘echo $MASTER_LOC/mchck/‘"
    fi
}

function prepare_directory
{
    if [[ $start_evol_at = "prems" ]]; then
        echo -e "Preparing working directory."
        if [[ $new_dir_fl -eq 0 ]]; then
            echo "Removing existing data files."
            mkdir $MODELS_LOC/models/temp
            cp -r ctl $MODELS_LOC/models/temp/
            cp -r phy $MODELS_LOC/models/temp/

            rm -r *
            cp -r $MODELS_LOC/models/temp/* .
            rm -r $MODELS_LOC/models/temp
            echo "Clean-up complete."
        else
```

```
            mkdir ctl
            mkdir phy
        fi
        cp $MASTER_LOC/m$chosen/yrec7.batch .
    fi
}

function choose_homfile
{
    # This function chooses the initial baseline model to use
    # Called from: main
    # Calls to: none

    # First check the closest baseline file if at all any exist
    aschomexist=`ls ${pathhom}/bl.m*a &> /dev/null ; echo $?`
    if [[ "$aschomexist" -eq "0" ]]; then
        # Run through the available baseline models to determine the closest
            and also
        # store the difference in mass with the closest match
        availar=( `ls ${pathhom}/bl.m*a` )
        difmina="0.05"
        for i in ${availar[@]}
        do
            mcomp=`basename $i | cut -c5-7 | sed 's/p/./'`
            mdiff=`awk "BEGIN { x=$MTOT-$mcomp;  print (x>=0) ? x : -x }"`
            closest=`awk "BEGIN { print $mdiff < $difmina ? 1 : 0 }"`
            if [[ "$closest" -eq "1" ]]; then
                homfile=$i
                choosen="1"
                break
            fi
        done
        if [[ $choosen -ne 1 ]]; then
            echo "No suitable baseline model file found. Exiting."
            exit 1
        else
            echo -e "Baseline model has mass $mcomp\nRescale mass is $MTOT"
            echo "Rescale mass differs from baseline mass by $mdiff"
        fi
    else
        echo "No baseline model files found. Exiting."
            exit 1
    fi

    # This is a double check. If the logic above is correct, we need not
       check
    # for the existence of the file. But still ...

    if [ ! -f $homfile ]; then
        echo "Baseline model file $homfile not found. Exiting."
        exit 1
    fi

    if [[ "$queryfl" -ne "1" ]]; then
        echo "Using baseline model $homfile"
    fi
}

function prepctl
{
    #This function prepares the default ctl files copied into the working
```

```
   directory
#and renames it to the format mMMMM_stage_stage.ctl
#It also calls yrec7.x and creates the models
#After this it EXITS

#MassSTring extracts mMMMM if the current working directory contains
   such a string
MST="m$MTOT_str"
#ARR=( prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_rgb2 rgb2_rgb3
   rgb3_rgb4 rgb4_rgb5 rgb5_rgb6 rgb6_trgb trgb_zahb zahb_tahb )
#ARR=( prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_zahb zahb_tahb )
ARR=( prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_rgb2 rgb2_rgb3
   rgb3_trgb trgb_zahb zahb_tahb )
#ARR=( prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_rgb2 rgb2_rgb3
   rgb3_rgb4 rgb4_trgb trgb_zahb zahb_tahb )

donedit yrec7.batch FNML1 \'ctl/$MST.ctl\'
donedit yrec7.batch FNML2 \'phy/$MST.phy\'

flag=0
for k in ${ARR[@]}
do
    if [[ "$INITIALIZED" -eq "0" ]]; then
        if [[ "${start_evol_at:0:4}" == "${k:0:4}" ]]; then
        INITIALIZED="1"
        else
        continue
        fi
    fi
    copy_files $k
    str="d${ALPHA_str}c${OVSHTS_str}z${ZSX0_str}y${Y0_str}m${MTOT_str}"
    #change description
    if [[ "$k" == "prems_zams" ]]; then
        donedit ctl/$k.ctl RSCLM $MTOT
        donedit ctl/$k.ctl RSCLX $X0
        donedit ctl/$k.ctl XENV0A $X0
        donedit ctl/$k.ctl RSCLZ $ZSX0
        donedit ctl/$k.ctl ZENV0A $ZSX0
        donedit ctl/$k.ctl FFIRST \'$homfile\'
        if [[ "$ngfl" -ne "1" ]]; then
            echo -n "set yrange [-0.7:3.2]; set xrange [4:3.55]; p " >
               hrplot
            echo -n "set multiplot; set yrange [4.4:2.2]; set xrange
               [4:3.55]; p " > gtplot
        fi
        if [[ $MTOT_str -ge 1600 && $MTOT_str -le 1800 ]]; then
            donedit ctl/$k.ctl "SENV0A(5)" 1.0D-9
            donedit ctl/$k.ctl "SENV0A(4)" 1.0D-9
        fi
    else
        donedit ctl/$k.ctl FFIRST \'${MST}.${k:0:4}\'
    fi
    if [[ "$k" == "tams_brgb" ]]; then
        mass=`echo $MTOT_str.0 1000.0 | awk 'BEGIN {FS=" "} {printf
           "%5.3f",$1/$2}'`
        if [[ $MTOT_str -ge 1600 && $MTOT_str -le 2100 ]]; then
            brgb_age=`echo $mass | awk '{printf "%6.4f",97.9802*exp
               (-2.57244*$1)+0.365894}'`
        elif [[ $MTOT_str -ge 1000 && $MTOT_str -lt 1600 ]]; then
            brgb_age=`echo $mass | awk '{printf "%6.4f",667.871*exp
               (-4.1963*$1)+1.22559}'`
            fi
#           donedit ctl/$k.ctl ENDAGE ${brgb_age}D+09
        fi
        if [[ "$k" == "brgb_rgb1" ]]; then
            mass=`echo $MTOT_str.0 1000.0 | awk 'BEGIN {FS=" "} {printf
                "%5.3f",$1/$2}'`
            if [[ $MTOT_str -ge 1600 && $MTOT_str -le 2100 ]]; then
                rgb1_age=`echo $mass | awk '{printf "%6.4f",50.2044*exp
                    (-2.08064*$1)+0.206441}'`
            elif [[ $MTOT_str -ge 1000 && $MTOT_str -lt 1600 ]]; then
                rgb1_age=`echo $mass | awk '{printf "%6.4f",600.595*exp
                    (-4.04844*$1)+1.22738}'`
            fi
#           donedit ctl/$k.ctl ENDAGE ${rgb1_age}D+09
        fi
        if [[ "$k" == "rgb1_rgb2" ]]; then
            mass=`echo $MTOT_str.0 1000.0 | awk 'BEGIN {FS=" "} {printf
                "%5.3f",$1/$2}'`
            if [[ $MTOT_str -ge 1600 && $MTOT_str -le 2100 ]]; then
                rgb2_age=`echo $mass | awk '{printf "%6.4f",76.3197*exp
                    (-2.33896*$1)+0.293359}'`
            elif [[ $MTOT_str -ge 1000 && $MTOT_str -lt 1600 ]]; then
                rgb2_age=`echo $mass | awk '{printf "%6.4f",608.527*exp
                    (-4.05862*$1)+1.32109}'`
            fi
#           donedit ctl/$k.ctl ENDAGE ${rgb2_age}D+09
        fi
        if [[ "$k" == "rgb2_rgb3" ]]; then
            echo $MTOT_str
            mass=`echo $MTOT_str.0 1000.0 | awk 'BEGIN {FS=" "} {printf
                "%5.3f",$1/$2}'`
            if [[ $MTOT_str -ge 1600 && $MTOT_str -le 2100 ]]; then
                rgb3_age=`echo $mass | awk '{printf "%6.4f",67.8887*exp
                    (-2.25807*$1)+0.279895}'`
            elif [[ $MTOT_str -ge 1000 && $MTOT_str -lt 1600 ]]; then
                rgb3_age=`echo $mass | awk '{printf "%6.4f",584.916*exp
                    (-4.00963*$1)+1.29851}'`
            fi
#           donedit ctl/$k.ctl ENDAGE ${rgb3_age}D+09
        fi
        if [[ "$k" == "rgb3_trgb" ]]; then
            donedit ctl/$k.ctl ENDAGE 12D+09
        fi
        if [[ "$k" == "trgb_zahb" ]]; then
            donedit ctl/$k.ctl ENDAGE 12.0D+09
        fi
        if [[ "$k" == "zahb_tahb" ]]; then
            donedit ctl/$k.ctl ENDAGE 12.0D+09
        fi
        donedit ctl/$k.ctl CMIXLA $ALPHA
        donedit phy/$k.phy ALPHAC $OVSHTS
        donedit ctl/$k.ctl FTRACK \'$str\_$k.track\'
        donedit ctl/$k.ctl FSHORT \'$str.short\'
        donedit ctl/$k.ctl FPMOD \'$str\'
        donedit ctl/$k.ctl FPENV \'$str\'
        donedit ctl/$k.ctl FPATM \'$str\'
        #rewrite the string mMMMM
        #FinalNameExtension extracts the final stage name eg tams
        FNE="`expr ${k} : '.*\(_[a-z][a-z][a-z][a-z,0-9]\)' |cut -c2-5`"
        #name of last model from which evolution of next stage should start
            eg. m0200.zams
        end_of_run_model="${MST}.${FNE}"
        #if [[ -f $starting_model_file ]]; then
        sed -i "/FLAST=/c\ FLAST='$end_of_run_model'" ctl/$k.ctl
        cd ctl
        ln -sf $k.ctl ${MST}.ctl
        cd ../phy
        ln -sf $k.phy ${MST}.phy
        cd ..

        if [[ flag -eq 0 ]]; then
            echo Running YREC now...
        fi
        flag=`expr $flag + 1`
        echo $k >>stagetime
        if [[ "$ngfl" -ne "1" ]]; then
            if [[ "$k" == "prems_zams" ]]; then
                echo -n "\"$str\_$k.track\"u 7:4 w lp" >>hrplot
                echo -n "\"$str\_$k.track\"u 7:6 notitle w lp" >>gtplot
            else
                echo -n ",\"$str\_$k.track\"u 7:4 w lp" >>hrplot
                echo -n ",\"$str\_$k.track\"u 7:6 notitle w lp" >>gtplot
            fi
        fi
        (time yrec7.x) 2>>stagetime
        rename
        crash_check $k
        if [[ `echo $?` -eq 0 ]]; then
            echo -e "Run of ${k} failed. Check crash report for details.\
               nExiting."
            exit 1
        fi
        if [[ "$?" -eq "0" && $end_of_run_model -nt $starting_model_file ]];
            then
            echo
            echo "Run of ${k} done"
            echo
        else
            echo
            echo "Run of ${k} failed. Exiting"
            echo "crash=$flag"
                    exit 1
        fi
        head -1 $k.flg >/dev/null 2>/dev/null
        if [[ "$stop_evol_at" == "$FNE" ]]; then
            ALLDONE="1"
        fi
        if [[ "$ALLDONE" -eq "1" ]]; then
            echo
            echo "All runs completed sucessfully"
            echo "crash=0"
            exit
        fi
    done
#   sed -i '$s/.$//' hrplot
#   sed -i '$s/.$/; /' gtplot
#   echo "set origin 0.4,0.1; set size 0.4,0.4; set xrange [3.65:3.66]; set
   yrange [2.1:2.5]; set xtics 0.01 set ytics 0.1; replot" >> gtplot
}

function donedit
{
    # This function modifies the ctlfile (first arg) by replacing the
       existing
    # value of a parameter (second arg) with a supplied value (third arg).
    sed -i "/^ $2/s%=.*%=$3%" $1
}

function rename
{
    echo "Renaming files. This may take a few seconds..."
    for j in ‘ls *_*.short‘
        do FILE_str="${j:0:25}"
        AGE_str=‘cat $j | grep "AGE(GYRS)" | awk ’BEGIN {FS=" "} {printf
           "%7.5f",$8/100.0}’ | awk ’BEGIN {FS="."} {printf "%s",$2}’‘
        mv ‘basename $j .short‘.pmod ${FILE_str}a${AGE_str}.pmod
        mv ‘basename $j .short‘.patm ${FILE_str}a${AGE_str}.patm
        mv ‘basename $j .short‘.penv ${FILE_str}a${AGE_str}.penv
        mv $j ${FILE_str}a${AGE_str}.short
    done
    echo "Renaming complete."
}

main "$@"
exit 0
```

13.2 Appendix B: Other Scripts

```
#!/bin/bash

# Script to check the number of models lying in a user-specified g-T box
# Stand Alone.
# Used in none.
# Jayant Thatte, December 2011

# INFO: The script scans all the .pmod files in the directory from which it is called.
# ARGUMENTS: None.
# OUTPUT: Outputs number of models in the g-T to the standard output.
# OUTPUT: Writes the ages of these models to a file called ‘box_models’.

function initialise
{

############ User Configurable Part Begins ############

        dr=$PWD
        lgsun=4.43775
        lt_min=3.65
        lt_max=3.66
        lg_min=2.1
        lg_max=2.5

############# User Configurable Part Ends #############

        count=0
}

function extract # Extracts the required values from the .pmod files and recursively check
{
        cd $dr
        rm box_models 2> /dev/null
        for i in ‘ls *.pmod‘
                do m_msun=‘cat $i | grep MASS | awk ’{print $2}’‘
                lt=`cat $i | grep TEFF | awk '{print $4}'`
                lr_rsun=`cat $i | grep R/RSUN | awk '{print $4}'`
                lg=`echo $lgsun $m_msun $lr_rsun | awk '{var=$1+log($2)/log(10)-2*$3; prin

                lt_flag=`echo $lt $lt_min $lt_max | awk '{print (($2 <= $1) && ($3 >= $1))
                lg_flag=`echo $lg $lg_min $lg_max | awk '{print (($2 <= $1) && ($3 >= $1))

                if [[ $lg_flag -eq 1 && $lt_flag -eq 1 ]]; then
                        ok=1
                        count=`expr $count + 1`
                        basename $i .pmod >> box_models
                fi
        done
        echo $count
}

function main
{
        initialise "$@"
        extract "$@"
}

main "$@"
exit 0
```

```
#!/bin/bash

# A script which checks all the stages of a particular run
# Uses 'crash_check'
# Used in none.
# Tamaghna Hazra, December 2011

# ARGUMENTS: None.
# OUTPUT: Outputs to the standard output, the stages in which Yrec encountered a problem

for i in prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_rgb2 rgb2_rgb3 rgb3_trgb trgb_zahb
do
        crash_check $i | grep FATAL
done
```

```
#!/bin/bash
# A script which plots several graphs which help in checking sanity of a model
# Stand Alone
# Used in none.
# Jayant Thatte, December 2011

# INFO: Takes required data from the .track files.
# ARGUMENTS: None.
# OUTPUT: Generates a Gnuplot file 'check_plot'. Load this in Gnuplot to view these graphs

filename=`echo $PWD | awk 'BEGIN {FS="/"} {print $6$7$8$9$10}'`
cat crash 2> /dev/null
echo "set multiplot; set size 0.5,0.5; set origin 0,0; set title \"no. of mesh points vs a
```

```
#!/bin/bash

# This script checks whether a given stage of Yrec went through successfully.
# Stand Alone.
# Used in ‘checkall’, ‘runyrec’ and ‘testall’.
# Jayant Thatte, December 2011

# INFO: Checks whether the user specified run has ended according to at least one of the c
# INFO: This script is also incorporated in ‘runyrec’ to stop Yrec from running subsequent

# OUTPUT: Generates a crash report file named <stage_name>.crash for the particular stage
# OUTPUT: Generates a file called main.crash containing the crash reports of all stages fo
# OUTPUT: Also generates an empty <stage_name>.flg file if a certain stage has crashed. Th

function initialise
{
        model_n=1
        age=3
        xc=26
        hit_n=0
        hit_a=0
        hit_x=0
        filename=‘echo $PWD | awk 'BEGIN {FS="/"} {print $6"c0000"$8$9$10}'‘
        ARR=( prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_rgb2 rgb2_rgb3 rgb3_trgb )
        safety_factor=2
        if [[ $1 = "prems_zams" ]]; then
                prev_run=x
        elif [[ $1 = "zams_tams" ]]; then
                prev_run=prems_zams
        elif [[ $1 = "tams_brgb" ]]; then
                prev_run=zams_tams
        elif [[ $1 = "brgb_rgb1" ]]; then
                prev_run=tams_brgb
        elif [[ $1 = "rgb1_rgb2" ]]; then
                prev_run=brgb_rgb1
        elif [[ $1 = "rgb2_rgb3" ]]; then
                prev_run=rgb1_rgb2
        elif [[ $1 = "rgb3_trgb" ]]; then
                prev_run=rgb2_rgb3
        else
                echo $1 is not a valid run-stage "in" Yrec. Exiting.
        fi
}

function get_last # Extracts information about the last model of the specified stage.
{
        tail -1 $filename\_$1.track >/dev/null 2>/dev/null
        if [[ ‘echo $?‘ -ne 0 ]]; then
                echo -e File $filename\_$1.track does not exist.
                echo -e Crash report not updated. exiting.
                exit
        fi
        last_model_curr=‘tail -1 $filename\_$1.track | awk 'BEGIN {FS=" "} {print $1}'‘
        if [[ $1 != "prems_zams" ]]; then
                last_model_prev=‘tail -1 $filename\_$prev_run.track | awk 'BEGIN {FS=" "}
                str_to_num $last_model_prev > j
                last_model=‘cat j‘
        else
                last_model_prev=0
        fi
        last_age=‘tail -1 $filename\_$1.track | awk 'BEGIN {FS=" "} {print $3}'‘
        last_xc=`tail -1 $filename\_$1.track | awk 'BEGIN {FS=" "} {print $26}'`

        str_to_num $last_model_curr > j
        last_model_curr=`cat j`

        last_model=`echo $last_model_curr $last_model_prev | awk '{var=$1-$2} END {print v

        str_to_num $last_age > j       # in Gyrs
        last_age=`cat j`
        last_age=`echo $last_age | awk '{var=$1*exp(9*log(10))} END {print var}'`

        str_to_num $last_xc >j
        last_xc=`cat j`
}

function get_condition # Extracts the user-specified stopping criteria from the .ctl file
{
        cat ctl/$1.ctl >/dev/null 2>/dev/null
        if [[ `echo $?` -ne 0 ]]; then
                echo -e File ctl/$1.ctl does not exist.
                echo -e Crash report not updated. exiting.
                exit
        fi
        nmodels=`cat ctl/$1.ctl | grep NMODLS | awk 'BEGIN {FS="="} {print $2}' | awk 'BEG
        endage=`cat ctl/$1.ctl | grep ENDAGE | awk 'BEGIN {FS="="} {print $2}'`
        endxc=`cat ctl/$1.ctl | grep ENDXC | awk 'BEGIN {FS="="} {print $2}'`

        str_to_num $nmodels >j
        nmodels=`cat j`

        str_to_num $endage >j
        endage=`cat j`

        str_to_num $endxc >j
        endxc=`cat j`
}

function get_step # Calculates age stepping and the stepping in core hydrogen content
{
        age_step=`tail -2 $filename\_$1.track | awk 'BEGIN {ageprev=0} {age_step=$3-agepre
        xc_step=`tail -2 $filename\_$1.track | awk 'BEGIN {xcprev=0} {xc_step=xcprev-$26 ;

        str_to_num $age_step >j    # in Gyrs
        age_step=`cat j`
        age_step=`echo $age_step | awk '{var=$1*exp(9*log(10))} END {print var}'`

        str_to_num $xc_step >j
        xc_step=`cat j`
        rm j

        age_tol=`awk -v sf=$safety_factor -v as=$age_step 'BEGIN {print sf*as}'`
        xc_tol=`awk -v sf=$safety_factor -v xs=$xc_step 'BEGIN {print sf*xs}'`
}

function check # Checks if the stage stopped due to at least one of the criterion.
{
        if [[ $nmodels -eq $last_model ]]; then
                hit_n=1
        fi

        age_cor=`echo $endage $last_age | awk '{var=$1-$2} END {print var}'`
        xc_cor=`echo $last_xc $endxc | awk '{var=$1-$2} END {print var}'`

        hit_a=`echo $age_cor $age_tol | awk '{print (($1 <= $2) && ($1 >= 0)) ? 1 : 0}'`
        hit_x=`echo $xc_cor $xc_tol | awk '{print (($1 <= $2) && ($1 >= 0)) ? 1 : 0}'`
}

function report_gen # Generates a report having details about the stage.
{
        rm $1.crash 2> /dev/null
        if [[ $hit_n -eq 0 && $hit_a -eq 0 && $hit_x -eq 0 ]]; then
                touch $1.flg
                echo -e "\n"FATAL\! Yrec encountered severe problem "in" stage $1.>> $1.cr
                echo -e This run has terminated "in" an unexpected manner.>> $1.crash
                echo -e Consider discarding the run and starting again with better paramet
                echo -e NMODLS=$nmodels"\t\t"Last model no. $last_model"\t\t"Cum. model no
                echo -e ENDAGE=$endage "\t"Age of "last" model = $last_age"\t"Age step = $
                echo -e ENDXC=$endxc"\t\t"X_c of "last" model = $last_xc"\t\t"X_c step = $
        fi
        if [[ $hit_n -eq 1 ]]; then
                rm $1.flg 2> /dev/null
                echo -e "\n"The stage $1 completed successfully.>> $1.crash
                echo -e Stopping condition\: NMODLS"\n">> $1.crash
                echo -e NMODLS=$nmodels"\t\t"Last model no. $last_model"\t\t"Cum. model no
                echo -e ENDAGE=$endage "\t"Age of "last" model = $last_age"\t"Age step = $
                echo -e ENDXC=$endxc"\t\t"X_c of "last" model = $last_xc"\t\t"X_c step = $
        fi
        if [[ $hit_a -eq 1 ]]; then
                rm $1.flg 2> /dev/null
                echo -e "\n"The stage $1 completed successfully.>> $1.crash
                echo -e Stopping condition\: ENDAGE>> $1.crash
                echo -e NMODLS=$nmodels"\t\t"Last model no. $last_model"\t\t"Cum. model no
                echo -e ENDAGE=$endage "\t"Age of "last" model = $last_age"\t"Age step = $
                echo -e ENDXC=$endxc"\t\t"X_c of "last" model = $last_xc"\t\t"X_c step = $
        fi
        if [[ $hit_x -eq 1 ]]; then
                rm $1.flg 2> /dev/null
                echo -e "\n"The stage $1 completed successfully.>> $1.crash
                echo -e Stopping condition\: ENDXC"\n">> $1.crash
                echo -e NMODLS=$nmodels"\t\t"Last model no. $last_model"\t\t"Cum. model no
                echo -e ENDAGE=$endage "\t"Age of "last" model = $last_age"\t"Age step = $
                echo -e ENDXC=$endxc"\t\t"X_c of "last" model = $last_xc"\t\t"X_c step = $
        fi
        rm main.crash 2> /dev/null
        for i in ${ARR[@]}
                do cat $i.crash >> main.crash 2>/dev/null
        done
        ls *.flg 2>/dev/null >/dev/null
        if [[ `echo $?` -eq 0 ]]; then
                touch crash
        else
                rm crash >/dev/null 2>/dev/null
        fi
}

function str_to_num # Converts the numbers in <.>E<..> format to understandable form.
{
        flag=X
        if [[ `echo $1 | grep e | wc -l` -eq 1 ]]; then
                flag=e
        elif [[ `echo $1 | grep E | wc -l` -eq 1 ]]; then
                flag=E
        elif [[ `echo $1 | grep d | wc -l` -eq 1 ]]; then
                flag=d
        elif [[ `echo $1 | grep D | wc -l` -eq 1 ]]; then
                flag=D
        fi
        if [[ flag != "X" ]]; then
                echo $1 | awk -F $flag '{var=$1*(exp($2*log(10)))} END {print var}'
        else
                echo $1
        fi
}

function main
{
        initialise "$@"
        get_last "$@"
        get_condition "$@"
        get_step "$@"
        check "$@"
        report_gen "$@"
        cat $1.crash
}

main "$@"
```

```
#!/bin/bash

# Jayant Thatte, December 2011
# Read usage
if [[ `echo $#` -eq 0 ]]; then
        less << EOU
Usage:

ARGUMENTS: Name of the .freq file should be given, without the extension, as the first argu
ARGUMENTS: If a different value of large separation, than what is calculated in the .struc

OUTPUT: The script generates a gnuplot file named <p> and launches Gnuplot. Load <p> in Gn

EOU
else
        time hmarun_editting --adi $1.pmod
        if [[ `echo $#` -eq 2 ]]; then
                large_sep=$2
        else
                large_sep=`cat "$1.struc" | grep -w "Delta Nu" | awk 'BEGIN {FS=" "} {prin
        fi
        cat "$1.freq" | awk 'BEGIN {FS=" "} {column=($3%'$large_sep')/'$large_sep'; print
        echo $1.ech generated successfully.
        for j in `seq 0 3`
                do cat $1.ech | awk -v k=$j 'BEGIN {FS=" "} {if ($1 == k) print $1,"\t"$2,"
        done
        echo p \"$1.ech0\" u 6:3,\"$1.ech1\" u 6:3,\"$1.ech2\" u 6:3,\"$1.ech3\" u 6:3 > p
        gnuplot

fi
```

```
#!/bin/bash

# Edited by Jayant Thatte, December 2011.
# Original script: hmarun.sh

# Suitable changes have beem made to ‘basic_quants’ and ‘config_jcd’ functions.
# Changes inlcude changes in FSTART, FEND, iscan and a few extra ‘echo’ statements.

function usage ()
{
        echo "Usage: ‘basename $0‘ [--struc] [--adi] [--hma] [--jig] <modelfilename>"
}
function main ()
{
if [[ "$#" -lt "1" ]]; then
        usage
        exit 1
fi

initialise_flags

parse_command_args "$@"
shift $?

basic_quants $FILENAME

if [[ "$hmafl" -eq "1" || "$adiplsfl" -eq "1" || "$strucfl" -eq "1" ]]; then
        echo -n "Writing out structure file..."
        makestruc
        echo "done"
fi

if [[ "$jigfl" -eq "1" ]]; then
        echo -n "Using JIG6 to compute frequencies..."
        makejig
        echo "done"
fi

if [[ "$hmafl" -eq "1" ]]; then
        echo -n "Using H. M. Antia's code to compute frequencies..."
        makehma
        echo "done"
fi

if [[ "$adiplsfl" -eq "1" ]]; then
        echo -n "Using ADIPLS to compute frequencies..."
        makeadipls
        echo "done"
fi
exit

}

function initialise_flags ()
{
        strucfl="0"
        adiplsfl="0"
        hmafl="0"
        jigfl="0"
        strucfl="0"
}
```

```
function parse_command_args ()
{
# This function parses the command-line arguments
# This is imported from: http://www.gnu.org/server/source/diffmon
# Thanks: Noah Friedman <friedman@prep.ai.mit.edu>
# Called from: main
# Calls to: get_option_argument, flag_dbl, usage

    local orig_number_options=$#

    # If you add new options be sure to change the wildcards below to make
    # sure they are unambiguous (i.e. only match one possible long option)
    # Be sure to show at least one instance of the full long option name to
    # document what the long option is canonically called.
    # Long options which take arguments will need a '*' appended to the
    # canonical name to match the value appended after the '=' character.
    while [ $# -gt 1 ]; do
       case z$1 in
          z--stru*| z--STRU* )
             shift
             strucfl="1"
             ;;
          z--adi* | z--ADI* )
             shift
             adiplsfl="1"
             ;;
          z--hma* | z--HMA* )
             shift
             hmafl="1"
             ;;
          z--jig* | z--JIG* )
             shift
             jigfl="1"
             ;;
          z-- )
             shift
             break
            ;;
          z-* )
             echo "`basename $0`:\`${1}' is not a valid option."
             usage
             exit 1
            ;;
          * )
             break
            ;;
       esac
    done

    FILENAME="$1"

}

function basic_quants ()
{
        GRAVCON="6.67259e-8"
        PI="3.1415926535898"
        SOLSCALE="2.429032"

        MASSSTUB="${1:20:5}"
        FILESTUB="${1:0:31}"
        IDTYPE="${1:25:1}"

        PMODFILE="${FILESTUB}.pmod"
        PENVFILE="${FILESTUB}.penv"
        PATMFILE="${FILESTUB}.patm"

        NMESH="`grep SHELLS ${PMODFILE} |awk '{print $8}'`"
        MASS="`grep MASS ${PMODFILE}| awk '{printf \"%5.3f\n\", $2}'`"
        RADIUS="`grep \"R/RSUN\" ${PMODFILE}| awk '{printf \"%5.3f\n\", 10**$4}'`"
        SCALE="`awk -v rad=\"$RADIUS\" -v mas=\"$MASS\" 'BEGIN {print (rad>0) ? sqrt(mas/r

    if [[ "$IDTYPE" == "x" ]]; then
        XCFLAG="1"
        XC="`echo ${FILESTUB:6:4}| awk '{printf "%11.5e", $1/10000}'`"
    elif [[ "$IDTYPE" == "a" || "$IDTYPE" == "_" ]]; then
        XCFLAG="0"
        AGE="`echo ${FILESTUB:6:4}| awk '{printf "%11.5e", $1/1000}'`"
    else
        echo "File is not identified by age or Xc, exiting."
        exit 1
    fi
    if [[ "$XCFLAG" == "1" ]]; then
        echo "Mass $MASS M/Msun; Xc $XC ; Radius $RADIUS R/Rsun"
    else
        echo "Mass $MASS M/Msun; Age $AGE Gyrs; Radius $RADIUS R/Rsun"
    fi
        echo "Sqrt(rhobar/rhobar_sun) $SCALE"

        SCFLAG="`echo $SCALE | awk '{print ($1==1.00000) ? "0" : "1"}'`"

 if [[ "$SCFLAG" -eq "0" ]]; then
        echo "Scaling factor not found. Exiting."
#       exit 1
 fi

FSTART="`echo $SCALE | awk '{printf "%6.3f", 0.1*$1}'`"
FEND="`echo $SCALE | awk '{printf "%6.3f", 2*$1}'`"
FDEL="`echo $SCALE | awk '{printf "%f", 50*$1}'`"
FSTART_mu="`echo $SCALE | awk '{printf "%9.3f", 1000*0.1*$1}'`"
#FEND_mu="`echo $SCALE | awk '{printf "%9.3f", 1000*5*$1}'`"
# Get only low order modes -- reduce FEND_mu in JIG6
FEND_mu="`echo $SCALE | awk '{printf "%9.3f", 1000*3*$1}'`"
#FDEL_mu="`echo $SCALE | awk '{printf "%f", 5.0*$1}'`"
# Get the g-modes for a giant -- reduce scanning interval
FDEL_mu="`echo $SCALE | awk '{printf "%f", 2.0*$1}'`"

###PSTART="`echo $FEND   | awk -v pi="$PI" '{printf "%6.3f", 1/(2*pi*$1)}'`"
###PEND="`echo $FSTART   | awk -v pi="$PI" '{printf "%6.3f", 1/(2*pi*$1)}'`"
###PDEL="`echo $FDEL     | awk -v pi="$PI" '{printf "%6.3f", 1/(2*pi*$1)}'`"
}


 function makestruc ()
{
STRUCSAMPLE="/data/research/YREC/SAMPLES/struc.nml"
MODELEXTR="/data/research/YREC/BIN/modextr_yrec.x"

STRUCFILE="${FILESTUB}.struc"
 if [[ "$hmafl" -eq "1" ]]; then
AVARFILE="${FILESTUB}.avar"
```

```
else
AVARFILE="\\/dev\\/null"
fi
if [[ "$adiplsfl" -eq "1" ]]; then
ADIPLSFILE="${FILESTUB}.mjcd"
else
ADIPLSFILE="\\/dev\\/null"
fi

echo "
s/pmodfile/${PMODFILE}/
s/penvfile/${PENVFILE}/
s/patmfile/${PATMFILE}/
s/strucfile/${STRUCFILE}/
s/adiplsfile/${ADIPLSFILE}/
s/avarfile/${AVARFILE}/
" > SED$$

sed -f SED$$ $STRUCSAMPLE > struc.nml

$MODELEXTR #> /dev/null
rm -f SED$$ struc.nml
}


function makejig()
{

JIGCODE="/data/research/JIG/BIN/jig6a.x"
SORTCODE="/data/research/JIG/BIN/sortmodes.x"
JIGSAMPLE="/data/research/JIG/SAMPLES/jig6a.nml"
JOUTFILE="${FILESTUB}.jout"
FJIGEXT="fjig"
FREQDETFILE="${FILESTUB}.fdet"
FREQFILE="${FILESTUB}.${FJIGEXT}"

echo "
s/pmodfile/${FILESTUB}.pmod/
s/penvfile/${FILESTUB}.penv/
s/patmfile/${FILESTUB}.patm/
s/outputfile/${JOUTFILE}/
/FSTART/c\ FSTART=$FSTART_mu
/FEND/c\ FEND=$FEND_mu
/FDEL/c\ FDEL=$FDEL_mu
" > SED$$

#### Scan in periods -- not working in JIG6??
###/PSTART/c\ PSTART=$PSTART
###/PEND/c\ PEND=$PEND
###/PDEL/c\ PDEL=$PDEL

sed -f SED$$ $JIGSAMPLE > jig6a.nml

rm -f SED$$

$JIGCODE

rm -f SCRATCH.OUT

#echo -n "Sorting ..."
if [[ -f $JOUTFILE ]]; then
```

```
echo "1
$JOUTFILE
2
$JOUTFILE
5" > SRT$$
fi

$SORTCODE < SRT$$ > /dev/null
echo "#    l   np   ng   nu(microHz)" > ${FREQFILE}
awk '{printf "%5i%4i%4i%12.3f\n",$8,$4, $6, $10}' ${JOUTFILE}_s >> ${FREQFILE}

rm -f SRT$$ sum

#mv ${JOUTFILE}_s ${FREQDETFILE}
rm ${JOUTFILE}_s ${JOUTFILE} ${JOUTFILE}_t
#mv ${JOUTFILE}_t ${FREQFILE}

#echo "done"
}

function makehma ()
{
HMACODE="/data/research/astero/progs/bin/freq2.2"
BOUNDCOND="ando"
##BOUNDCOND="free"  --> delta p = 0
if [[ "$adiplsfl" -ne "1" && "$jigfl" -ne "1" ]]; then
        FHMAEXT="freq"
else
        FHMAEXT="fhma"
fi

FREQDETFILE="${FILESTUB}.fout"
FREQFILE="${FILESTUB}.${FHMAEXT}"

if [[ -f "$AVARFILE" ]]; then
        NN="`head -1 $AVARFILE | awk '{printf \"%i\", $5/2 -1}'`"
else
        echo "$AVARFILE not found. A(i) variables not calculated by $MODELEXTR. Exiting"
        exit 1
fi

echo "'$AVARFILE'
$NN 1 2
1 0 8 4
1 0 1 1 10
p
$BOUNDCOND
/
/
'$FREQDETFILE'
'$FREQFILE'
/
0 3 1 1 100 $FSTART $FEND
10 5 /" > freq.inp

$HMACODE < freq.inp > /dev/null

rm -f freq.inp models TEMP fort.* egvt* invcof

}
```

```
function config_jcd
{
# This function sets the file names, frequency limits etc. for
# JCD's frequency code, ADIPLS.

# Path for ADIPLS root directory
        freqpath="/data/research/ADIPLS"

# Name of ADIPLS executable for frequency calculations
# --- need not be changed usually
        MAKEFREQ="${freqpath}/bin/adipls.n.d"
# Name of sample input file for ADIPLS frequency calculation
        FREQINPF="/data/research/ADIPLS/samples/adipls.n.in"

# Name of ADIPLS executable for redistribution of mesh
# --- need not be changed usually
        REDISTRIB="${freqpath}/bin/redistrb.d"
# Name of sample input file for ADIPLS redistribution of mesh points
        REDISTRF="/data/research/ADIPLS/samples/redistrb.in"
# Choose whether redistribution of mesh will be done. By default it is NOT
# done (ver. 2.1.1). Set to "1" to enable.
        redistrfl="0"

# Name of ADIPLS executable for frequency output from summary file
# If this is set-obs.d then we need to set the option value also.
# If rotational kernels are being computed (irotkr=1 below) then the option
# value is automatically changed later.
        #SCANPROG="${freqpath}/bin/scan-agsm.d"
        SCANPROG="${freqpath}/bin/set-obs.d"
        setobscase="15"

# Name of the file where all stdout from frequency code is redirected.
        freqoutfile="/dev/null"

# Now we define the important input parameters for ADIPLS
# To define the search frequencies in dimensionless units, istsig <=1
        istsig="1"
# For low order p-modes nsig=1.
# For high order p-modes (recommended nsig=2), with sufficiently high
# iscan, nsig=1 should be ok too.
# For high order g-modes, we must have nsig=3.
        nsig="3"
# To step in frequency from freql
        itrsig="1"
# Limits of frequency in terms of square of dimensionless frequency: sigma**2
        SFAC="`echo $SCALE $SOLSCALE $PI $GRAVCON|awk '{print (2*$3)/sqrt($4)/($1*$2)}'`"
        freql="`echo $FSTART $SFAC |awk '{print ($1*$2*1e-3)**2}'`"
        freqh="`echo $FEND $SFAC |awk '{print ($1*$2*1e-3)**2}'`"
echo
echo
echo "SCALE=$SCALE"
echo "SFAC=$SFAC"
echo "freql=$freql"
echo "freqh=$freqh"
echo "FSTART=$FSTART"
echo "FEND=$FEND"
echo
#       freqh="1000.0"
# Set the number of steps of scanning the frequency interval
        iscan="50000"
# Set the surface boundary conditions (Look at ADIPLS documentation for details)
```

```
        istsbc="1"
#       istsbc="0" --> delta p = 0
        icaswn="10010"
#       icaswn="10"  with istsbc=0
# To calculate the rotational kernels and the Ledoux constant, set irotkr=1
# For irotkr=1, the beta values are also written out in the freqfile
        irotkr="1"
        if [[ "$irotkr" -eq "1" ]]; then
            setobscase="`echo $setobscase| awk '{print $1+10}'`"
        fi

# Files to be removed after completion of the ADIPLS run
# Note the "echo ' '" to protect the pseudo filenames!
        rmadipls="`echo '$prtfile $ssmfile $rotkerfile $tmpmodelfile $freqmodelfile $FREQIN

# Filename extensions
# ---- need not be changed usually
        modelext="mjcd"
if [[ "$hmafl" -ne "1" && "$jigfl" -ne "1" ]]; then
        freqext="freq"
else
        freqext="fadi"
fi
        gsmext="gsm"
        ssmext="ssm"
        prtext="prt"
        rkrext="rkr"

################################################################################
############## User-configurable part for ADIPLS ends #####################
################################################################################

# Do not change the following lines
        export PATH="$PATH:${freqpath}/bin"
        config_jcd_called="1"

# Set the variable aprgdir, required by ADIPLS
        export aprgdir=${freqpath}
}

function makeadipls ()
{
# This function calculates the frequencies of the CESAM-generated model using
# JCD's ADIPLS package
# Calls to: config_jcd, jcdedit

        #echo -n "Calculating frequencies ...."
# Configure the pathnames, frequency limits etc. (once only)
        if [[ "$config_jcd_called" -ne "1" ]]; then
            config_jcd
        fi

# Name the files
        freqmodelfile="${FILESTUB}.${modelext}"
        tmpmodelfile="adiplsmodel"
        prtfile="${FILESTUB}.${prtext}"
        gsmfile="${FILESTUB}.${gsmext}"
        ssmfile="${FILESTUB}.${ssmext}"
        freqfile="${FILESTUB}.${freqext}"
        rotkerfile="${FILESTUB}.${rkrext}"
        FREQINPLOC="adipls.n.in"
        REDISTRLOC="redistrb.in"

# Get the no. of mesh points from the oscfile
        nmesh="$NMESH"

# Redistribution of mesh ————————————————————————————————————————
        if [[ "$redistrfl" -eq "1" ]]; then
# Temporarily move the model to another file
        mv $freqmodelfile $tmpmodelfile
# Generate the input file for ADIPLS redistribution program
        cp $REDISTRF $REDISTRLOC
# Edit the filenames and the no. of points in the model
        jcdedit $REDISTRLOC file.model.old $tmpmodelfile
        jcdedit $REDISTRLOC file.model.new $freqmodelfile
        jcdedit $REDISTRLOC nn $nmesh
# Allow interpolation from two sides of convective zone and put a double point
# at the boundary
        jcdedit $REDISTRLOC icvzbn 12
        jcdedit $REDISTRLOC icase 11
# Redistribute the mesh
        $REDISTRIB $REDISTRLOC >> ${freqoutfile}
        fi

# Calculation of frequencies ——————————————————————————————————————
# Generate the input file for ADIPLS frequency computation
        cp $FREQINPF $FREQINPLOC
# Edit the filenames
        jcdedit $FREQINPLOC file.model $freqmodelfile
        jcdedit $FREQINPLOC file.prt $prtfile
        jcdedit $FREQINPLOC file.gsm $gsmfile
        jcdedit $FREQINPLOC file.ssm $ssmfile
        if [[ "$irotkr" -eq "1" ]]; then
            jcdedit $FREQINPLOC file.rkr $rotkerfile
        fi
        jcdedit $FREQINPLOC nprmod $nmesh

# Set the frequency limits and the scanning interval
        jcdedit $FREQINPLOC istsig $istsig
        jcdedit $FREQINPLOC sig1 $freql
        jcdedit $FREQINPLOC sig2 $freqh
        jcdedit $FREQINPLOC itrsig $itrsig
        jcdedit $FREQINPLOC nsig $nsig
        jcdedit $FREQINPLOC iscan $iscan
# Set the boundary conditions
        jcdedit $FREQINPLOC istsbc $istsbc
        jcdedit $FREQINPLOC icaswn $icaswn

#Compute the rotational kernels, but do not print them
        jcdedit $FREQINPLOC irotkr $irotkr

# Compute the frequencies
        $MAKEFREQ $FREQINPLOC >> ${freqoutfile}
        if [[ "$?" -ne "0" ]]; then
            echo
            echo
            echo "ADIPLS had runtime problems. Please check. Stopping."
            exit 1
        fi

# Writing out the frequencies from summary file ————————————————————
# Output the frequencies from the grand summary file into the frequency file
```

```
        #$SCANPROG ${gsmfile} > ${freqfile}
        $SCANPROG $setobscase ${gsmfile} ${freqfile} >> ${freqoutfile}
#       echo "done!"

# Clean up files
        eval rm -f $rmadipls
}


function jcdedit {
# This function edits the input file required for JCD's frequency code.
# Called from: freq_jcd
# Calls to: none

        modiffile="$1"
        string="$2"
        newstring="$3"

        headlines=`grep -i -n -w ^-1 $modiffile | awk 'BEGIN { FS = ":" }; {print $1+1}'`
        ntot=`wc $modiffile | awk '{print $1}'`

        nhead=`grep -i -n -w $string $modiffile | awk 'BEGIN { FS = ":" }; {print $1}'`
        ntail=`awk -v ntot=$ntot -v nhead=$nhead 'BEGIN {print ntot-nhead-1}'`

        if [[ "$nhead" -le "$headlines" ]]; then
            nhead=`awk -v nh=$nhead 'BEGIN {print nh-1}'`
            ntail=`awk -v nt=$ntail 'BEGIN {print nt+1}'`
            head -$nhead $1 > tmp$$
            unitno=`grep -i -w $string $modiffile | awk '{print $1}'`
            echo "$unitno '$newstring' @" >> tmp$$
        else
            head -$nhead $modiffile > tmp$$
            echo "$newstring, @" >> tmp$$
        fi

        tail -$ntail $modiffile >> tmp$$
        mv tmp$$ $modiffile
}

##############################################################################
################# Actual executable statements start here ##################
##############################################################################

main $@
exit 0




#!/bin/bash

# Tamaghna Hazra, July 2011.
# Checks which all 'HPTTOL' are hit how many times by counting the flags in 'fort.52' file

for i in `seq 1 12`; do echo -n "HPTTOL($i)"; cat fort.52 | grep "HPTTOL($i)" | wc -l; don




#!bin/bash
```

```
# Tamaghna Hazra, July 2011
# Checks which 'HTOLER' was hit how many times by counting the flags in 'fort.52' file.

for i in `seq 1 5`; do echo -n "HTOLER($i,1)"; cat fort.52 | grep "HTOLER($i,1)" | wc -l ;
```

```
#!/bin/bash
# Plots g-T graph for a specific evolutionary track along with the error box
    within which the star is expected to lie. Observing this plot, the user can
    tell whether or the the evolution track which is tested is useful or not.
# Depends on 'runyrec' to generate the 'gtplot' file.
# Used in: None.

# Jayant Thatte, December 2011.
# INPUT: Runs on the 'gtplot' file in the PWD.
# OUTPUT: Outputs a file named gnuplot 'gtbox'.
# ARGS: None.

cat gtplot > gtbox 2>/dev/null
echo "; set origin 0.1,0.55; set size 0.4,0.4; set xrange [3.65:3.66]; set
    yrange [2.1:2.5]; set xtics 0.01; set ytics 0.1; replot" >> gtbox
```

```
#!/bin/bash

# Checks which all runs have been successful by recursively going to each directory in the
# Jayant Thatte, December 2011.
# Uses: 'crash_check', 'makebox'
# Used in: None.

ARR=( prems_zams zams_tams tams_brgb brgb_rgb1 rgb1_rgb2 rgb2_rgb3 rgb3_trgb )
cd ~/giant/models/d1800/c0000/
for i in `ls | grep z[0-9][0-9][0-9]`
do      cd ~/giant/models/d1800/c0000/$i
        for j in `ls | grep y[0-9][0-9][0-9]`
        do      cd ~/giant/models/d1800/c0000/$i/$j
                for k in `ls | grep m[0-9][0-9][0-9]`
                do      cd ~/giant/models/d1800/c0000/$i/$j/$k
                        for m in ${ARR[@]}
                        do      crash_check $m
                        done
                        makebox
                done
        done
done
```

14 Acknowledgements

- I would like to thank Homi Bhabha Centre for Science Education (HBCSE), TIFR, Mumbai for giving me this stimulating opportunity under the program: National Initiative for Undergraduate Sciences (NIUS).
- My sincere thanks to Dr. Anwesh Mazumdar, TIFR, Mumbai for mentoring the project throughout it's duration.
- It was an enjoyable experience to work with Tamaghna Hazra, IIT Kanpur who was my colleague during this project.