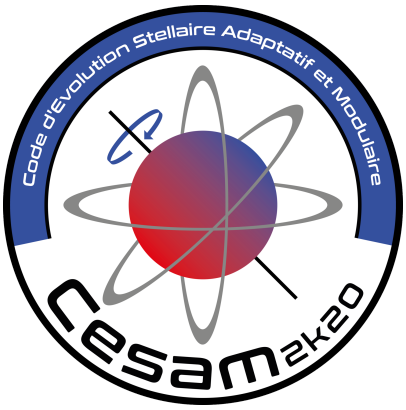

Cesam2k20 Workshop

7th- 8th Dec. 2023, Meudon

L. Manchon, M. Deal, Y. Lebreton J. P. C. Marques

December 12, 2023

Welcome to the 1st Cesam2k20 Workshop!

Unless you are already familiar with Cesam2k20, or its direct predecessor CESTAM, we advise you to start by the beginning: in Chapter 1, you will learn how to compute a standard solar model with Cesam2k20. Otherwise, you can jump to Chapter 2 and choose what you want to learn.

Contents

Chapter 1

The beginning: Standard solar model

1.1 Code installation

Clone the Cesam2k20 repository (ideally in your `home/username`):

```
git clone https://git.ias.u-psud.fr/joao.marques/cesam2k20_releases.git
mv cesam2k20_releases cesam2k20
cd cesam2k20
```

and then check you have all dependencies installed:

```
./check_deps.sh
```

Program `check_deps.sh` will check if all the dependencies are available. The program uses `ldconfig` to find which compiler is installed, libraries like `lapack`, `blas`, or interpreters like `bash`. If `ldconfig` is not installed, `check_deps.sh` will not find anything, while you may have the dependencies installed. In this case, tell your system where to find above mentioned programs by appending the appropriate paths to your `LD_LIBRARY_PATH`. After that, do

```
./configure.sh gfortran_h5
```

Now, you must say to your system where are the executables. Add following lines to your `.bashrc`, located in your `home`:

```
export CESDIR=${HOME}/cesam2k20

if [ -z ${LD_LIBRARY_PATH} ]
then
    export LD_LIBRARY_PATH="${CESDIR}/lib"
else
    export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:${CESDIR}/lib"
```

```
fi

export PATH="$PATH:${CESDIR}/bin/"
```

Finally, run

```
make install
```

And you are done !

Open a new terminal. Create a directory where you will compute your models (preferably not in the `cesam2k20` directory). Go in your models directory and type:

```
run_cesam2k20.py test
```

A new window must open. Try to understand available options ! Then click `OK` to launch the computation of a solar model. When the computation is finished (around 3 min depending on your machine), you end up with several files `test*`. Among them:

- `test.HR` : contains evolution of model's global properties with age;
- `test-nad.osc` : contains the internal structure of the model at last time step (a solar model in our case);
- `test.don` : contains input parameters (same information as the Graphic User Interface).

You can display evolution in HR diagram with:

```
plot_hr.py test
```

You can display internal structure with:

```
plot_osc.py test
```

1.2 Model parameters

The results of the computation of stellar models depend on many input parameters. There are two types of parameters: *(i)* the physical parameters, and *(ii)* the parameters used to model the physical processes in stellar interiors, such as convection. Some examples:

- Physical parameters:
  - Initial mass ;

  - Age ;
  - Initial chemical composition: X (hydrogen mass fraction), Y (helium mass fraction) and Z (metal mass fraction).

- Other parameters :
  - The mixing length theory parameter, α ;
  - Convective overshoot parameter ;
  - Parameters related to equation of state, opacity, nuclear reaction rates.

In the case of the Sun, we know the mass and the age very accurately. We also know reasonably well the current surface metallicity Z. If we neglect microscopic diffusion, we can assume that initial metallicity is equal to present surface metallicity. It remains two free parameters:

- The initial helium mass fraction $Y_{\rm init}$. Indeed, solar photosphere is too cold for helium lines to form;
- The α parameter of convection.

Now, compute few "solar" models by changing α and $Y_{\rm init}$.

1. The primordial Y (resulting from primordial nucleosynthesis) is 0.248. The range of possible Y is expected to be between 0.248 (primordial helium) and $\simeq 0.3$. Compute three models with 3 different values of Y in this range. What is the effect of a variation of Y on stellar models ?
2. The value of α must be around 1. Compute three models with three different values, between 1 and 2. What is the effect of a α-variation on stellar models ?

Help : You can draw several tracks on the same HR diagram:

```
plot_hr.py model1 model2 model3
```

where `model1`, `model2`, and `model3` are three different models.

1.3 Calibration of a solar model

In general, a model with given α and Y has neither solar luminosity nor solar radius at solar age. The calibration of a solar model consists in finding values of α and $Y_{\rm init}$ (we will call them α_0 and Y_0) such that

$$\begin{aligned} L_{\rm model}(\alpha_0, Y_0) &= L_\odot \\ R_{\rm model}(\alpha_0, Y_0) &= R_\odot, \end{aligned}$$

at solar age.

In practice, default values of α and Y are quite close to α_0 and Y_0. Let us write $\alpha_0 = \alpha + \delta\alpha$ and $Y_0 = Y + \delta Y$. The idea is the following:

1. Compute a model with solar age with the default values of α and Y. Check that:

$$\begin{aligned} L_{\rm model}(\alpha, Y) &\neq L_\odot \\ R_{\rm model}(\alpha, Y) &\neq R_\odot. \end{aligned}$$

2. Look for the corrections $\delta\alpha$ and δY such that:

$$\begin{aligned} L_{\rm model}(\alpha + \delta\alpha, Y + \delta Y) &= L_\odot \\ R_{\rm model}(\alpha + \delta\alpha, Y + \delta Y) &= R_\odot. \end{aligned}$$

   This is a system of two non-linear equations, with two unknowns. We take advantage of the fact that $\delta\alpha$ and δY are small to perform a lst order series expansion. You can then write a system of two linear equations.

3. You will have to compute four partial derivatives of $L_{\rm model}$ and $R_{\rm model}$:

$$\begin{matrix} \dfrac{\partial L_{\rm model}}{\partial \alpha} & \dfrac{\partial L_{\rm model}}{\partial Y} \\ \dfrac{\partial R_{\rm model}}{\partial \alpha} & \dfrac{\partial R_{\rm model}}{\partial Y} \end{matrix}.$$

   You can compute these derivatives *numerically* by computing two "shifted" models with small variation of α and Y.

4. The solution of the system (remember, the corrections are the unknowns) allows to have more accurate α and Y values. However, you will certainly need to reiterate this process to obtain a model sufficiently close to the Sun.

1.3.1 Useful tips

It is easier to write a Python script that automates the computations. You script must start with

```
1 import os
2 os.sys.path.append(os.environ["CESDIR"] + "/python3")
3 from pycesam import *
```

The results and several methods that allow you to read the output are available with the class `CModel`. To create an instance of this class (e.g. for a model called `1msun`):

```
mdl = CModel("1msun")
```

The instance `mdl` contains all methods and available data of model `1msun`.
The input parameters are stored in the class `mdl.params`. For instance,

- α : `mdl.params.alpha`
- Y : `mdl.params.y0`.

It is possible to copy all parameters from a model to another one. For instance, we want all the parameters of a model `mdl1` and `mdl2` to be the same, except α of `mdl1`. Then, we start by copying all parameters of `mdl2` in `mdl1`:

```
mdl1.params.copy(mdl2.params)
```

Then, we change the α value of `mdl1`:

```
mdl1.params.alpha = 1.2
```

A Python magic method allows you to launch the computation of a model from your script:

```
mdl1()
```

Few attributes available after the computation:

- `mdl.age`: `numpy array` that contains the age at each time step (read in .HR file);
- `mdl.log_teff`: $\log T_{\rm eff}$;
- `mdl.log_l`: $\log(L_\star/L_\odot)$;
- `mdl.log_r`: $\log(R_\star/R_\odot)$;
- `mdl.var`: list that contains internal structure of last model: `mdl.var[i]`, with `i` the index corresponding to the physical quantity:
  - 0: r (cm),
  - 1: $m/M_\star$,
  - 2: T (K),
  - 3: $P_{\rm tot}$ (dyn/cm^2),
  - 4: ρ (g/cm^3),
  - 5: ∇,
  - 6: L (erg/s),

  - 7: κ (cm^2/g),,
  - 8: ε (erg/g/s),
  - 9: Γ_1,
  - 10: $\nabla_{\rm ad}$,
  - 21: $\nabla_{\rm rad}$,
  - 31: u,
  - 45: $X_{\rm H1}$,
  - 46: $X_{\rm He3}$,
  - 47: $X_{\rm He4}$,
  - 48: $X_{\rm C12}$,
  - 49: $X_{\rm C13}$,
  - 50: $X_{\rm N14}$,
  - 51: $X_{\rm N15}$,
  - 52: $X_{\rm O16}$,
  - 53: $X_{\rm O17}$,
  - 54: $X_{\rm Si29}$.

- `mdl.nom_vars` : list that contains the name of all physical quantities stored in `mdl.var`;
- `mdl.ab_c` and `mdl.ab_s`: dictionary that contains chemical abundances at centre and surface, for each element. The values in the dictionary are lists of abundances at each time step. E.g. : `mdl.ab_c['H1']`, contains core abundance of hydrogen along evolution.
- `mdl.nom_elem`: list that contains names of elements followed in `mdl.ab_c` and `mdl.ab_s`.

1.4 Calibrated solar model

We will now compare calibrated solar models with observations. Calibrated models are:

- The one you just computed (without diffusion);
- Four models in `START/models/` :
  - Models with diffusion :
    * With chemical composition of Grevesse & Noels (1993; GN93) ;
    * With chemical composition of Asplund et al. (2009; AGS09).
  - Models without diffusion :

* With chemical composition of GN93;
* With chemical composition of AGS09.

All these models have, of course, same radius and luminosity: they can't be distinguished with their surface properties. Their interiors is however different.

Helioseismology allows to constrain solar interior, for instance through its sound speed profile. This profile can be found in the directory `START/obs_sound_speed` in file `bp99soundspeeds.dat`, with a script `read_sound_speed.py` that shows you how to read it.

1. Compute the sound speed c_s as a function of radius for all calibrated models. We recall that $c_\mathrm{s}^2 = \Gamma_1 P/\rho$.
2. Compare the sound speed of the models with the one obtained from helioseismic data.
3. Plot $\left(c_s^{\mathrm{model}} - c_s^{\mathrm{obs}}\right)/c_s^{\mathrm{model}}$ as a function of radius for all the models. Which one fits observations the best ?

Chapter 2

À la carte discovery of Cesam2k20

2.1 Compute high (or low) mass stellar models

You can use Cesam2k20 to compute massive (or light) stellar models. This section focuses on models of massive stars, but the method can be used for low mass stars. There is no upper limit on the initial stellar mass Cesam2k20 can handle, but always keep in mind that, as a modeller, you must ensure that the physical ingredients you choose are adapted for the model you want to compute. It is therefore possible that for extreme $\rho - T$ conditions, the suitable physical inputs may still lack in Cesam2k20 (if you want to collaborate on such topic, please tell us !). That being said, this exercise will teach you how to proceed if you want to compute a massive stellar model.

1. Try to compute a $6M_\odot$ stellar model, and stop after the 1$^{\mathrm{st}}$ time step (in the `.don` file, set `NB_MAX_MODELES = 1`, or in the GUI, tab 'Evolution', field 'Maximum number of models'), keeping default values for all other parameters. While for some of us, $6M_\odot$ is not very massive, it is massive enough for the purpose of this exercise.

2. It cannot start ? Of course: you need to understand how Cesam2k20 computes the initial model.

In an initial PMS model, all energy comes from released potential energy. Since it is fully convective, it is chemically homogeneous and isentropic. Therefore, time derivative of the specific entropy s is constant in the star. In this case, energy equation writes

$$\frac{\partial L}{\partial m} = \varepsilon_{\rm G} = -T\frac{\partial s}{\partial t} = c(t)T, \tag{2.1}$$

where $c(t)$ is a contraction constant, or Iben's constant. Initial time step is simply obtained by saying that the energy radiated during a time step Δt equals the variation of gravitational energy:

$$\frac{L(t)+L(t+\Delta t)}{2}\Delta t \simeq \mathcal{G}M_\star^2\left(\frac{1}{R_\star(t+\Delta t)}-\frac{1}{R_\star(t)}\right) \tag{2.2}$$

The two models at time t and $t+\Delta t$ are computed assuming a value of $c(t)$ and $c(t+\Delta t) = 1.1c(t)$. The structure of these initial models is computed starting from an initial, pre-computed, PMS structure, that should be close enough to the model you are willing to compute.

If the computation of the $6M_\odot$ model failed, it is simply that you kept default Iben's constant and default initial PMS model. These are well suited for the Sun, not for more massive models. Three initial models are available with Cesam2k20 in ASCII format. These models are located in `SUN_STAR_DATA`, with name `8d-5.pms`, `5d-4.pms`, `2d-2.pms`. The name corresponds to the associated Iben's constant (you can however set a different Iben's constant, not too far).

3. In the GUI, go in tab *"Run options"* and set `5d-4.pms` in *"Mod init"*, and `5e-4` for *"Iben's constant"* (these values can also be modified in file `.run`. Run the model again.

4. Run the same model again, with a mass of $7M_\odot$. Cannot start ? Try a Iben's constant of `1e-4`.

5. Run the same model again, with a mass of $8M_\odot$. Cannot start ? Try a Iben's constant of `1e-3` and `2d-2.pms` as initial model.

You understand that you have to find the correct combination of Iben's constant and initial PMS file. It can be very long, and you may reach an initial mass where no combination works. However, there is a more simple way

5. Re-run a $6M_\odot$ model with identical settings as in question 3., with name `6msun`. In the current directory, notice that a file with extension `6msun_B.pms` has been created. This file, in binary format, works like the ASCII files `8d-5.pms`, `5d-4.pms`, `2d-2.pms`, and can be used to initialise Cesam2k20 track.

6. From this model, we will compute a $15M_\odot$ model with name `15msun`. Create a copy of `6msun_B.pms` with name `15msun_B.pms`. If you try to compute directly the model starting from `15msun_B.pms` (which was computed for a $6M_\odot$ model), the run will not be able to start. To circumvent this problem, we need to slowly increase the initial mass. In the GUI, for model `15msun`, set a mass of $9M_\odot$, check also that you are still computing only the 1st time step. To start from this binary model, in the GUI, go in tab *"Run options"* select "Binary" and set the name of the `15_B.pms` file in *"Mod init"*.

7. Run the model. If it is a success, increase the mass to $12M_\odot$, run it again, and finally increase it to $15M_\odot$. If it fails, slightly increase the Iben's constant, and try again.

2.2 Convection

Options for controlling convection in the `.don` file are the following:

```
&NL_CONV
NOM_CONV = 'conv_jmj',
NOM_ALPHA = 'alpha_f',
ALPHA = 1.6,
NOM_OVSHTS = 'ovshts_f',
OVSHTS = 0.0,
OVSHTI = 0.0,
JPZ = F,
CPTURB = 0.0,
LEDOUX = F
/
```

- `NOM_CONV` is the name of the convection formalism used. Available options are:
  - `conv_jmj` (resp. `conv_jmj_thick`): MLT formalism assuming linear temperature distribution inside a bubble of rising material (resp. assuming a temperature distribution that verifies a diffusion equation; Henyey et al., 1965).
  - `conv_cm` (resp. `conv_cm_reza`): Canuto & Mazitelli (1991) model without accounting for $\delta = \partial\rho/\partial T_p$ in the mixing length (resp. accounting for it).
  - `conv_cgm_reza`: Canuto et al. (1996) model.
  - `conv_a0`: Model of Eggleton (1972) with mixing length going to 0 near the boundary of RZ/CZ.
- `NOM_ALPHA`: controls how the mixing length parameter varies through time (see Sect.):
  - `alpha_f`: fixed along evolution.

  - `alpha_sonoi19`: α follows prescription of Sonoi et al. (2019).
  - `entropy_ludwig99`: α is controlled by the entropy following Ludwig et al. (1999)'s functional fit.
  - `entropy_magic13`: α is controlled by the entropy following Magic et al. (2013)'s functional fit.
  - `entropy_tanner16`: α is controlled by the entropy following Tanner et al. (2016)'s functional fit.

- `ALPHA`: α value.
- `NOM_OVSHTS`: gives a prescription for `OVSHTS`:
  - `ovshts_f`: takes value given in `OVSHTS`.
  - `ovshts_claret18`: set value according to mass dependent scaling of Claret et al. (2018).
- `OVSHTS`: Overshoot value above the CZ.
- `OVSHTI`: Overshoot value under the CZ.
- `JPZ`: use of convective penetration formalism of Zahn (1991).
- `CPTURB`: Coefficient of turbulent pressure.
- `LEDOUX`: use of Ledoux criterion.

2.2.1 Standard 1D convection

1. Play with above options !

2.2.2 Entropy-calibrated convection

How it works

The choice of the value of α is a well known problem in stellar modelling. Newly developed methods (Spada et al., 2018, 2019, 2021, Manchon et al. submitted) propose to set this value by using prescriptions for the entropy of the adiabat derived from 3D models of surface convection. Since the value of α controls the adiabat the model is on, α can be tuned along evolution, so that at any time the entropy of the adiabat of the 1D model matches the value prescribed by fits depending on $T_{\rm eff}$, $\log g$ and [Fe/H]. Entropy-calibration (EC) significantly increase computation time.

Prescriptions are extracted by fitting functions on set of entropies obtained from grids of 3D models of stellar surface convection. The coefficients involved in these functions have been calibrated on several grids. In order to use values listed in original paper (Ludwig et al. 1999,

Magic et al., 2013, etc), use `extra%ec_grid = 'original'`. To use values calibrated on the CIFIST grid (preferred), use `extra%ec_grid = 'cifist'`.

Grids of 3D model atmospheres are computed with a given EoS, and a given chemical composition. To be used in 1D stellar evolution code, the prescription must be corrected in two ways:

- entropy is defined up to a constant $\mathrm{d}s$ and this constant may differ from the EoS used in the 1D and the 3D codes. To compute $\mathrm{d}s$, you must have a reference model, for which we have a good idea of the value of the entropy in the adiabat. Since we have 3D models of stellar atmosphere available, we can use a solar models as the reference model. Take one of the solar model computed in Chapter 1, choose a prescription (e.g. Magic et al., 2013) and run :

```
from pycesam import *

mdl = CModel( "your_solar_model" )
ds = mdl.get_ds( 'magic', grid='cifist' )
```

- entropy also depends on the chemical composition. The chemical composition evolves through time, and is different from the one used in 3D atmosphere models. The effect of a different chemical composition can be corrected at 1$^{\mathrm{st}}$ order with the ration of the mean molecular weight between 3D and 1D model: $f_\mu = \mu_{3\mathrm{D}}/\mu_{1\mathrm{D}}$.

The corrected prescribed entropy of the adiabat reads:

$$s^{\mathrm{corr}}_{\mathrm{prescr}} = s_{\mathrm{prescr}} f_\mu + \mathrm{d}s \tag{2.3}$$

Setup you `.don` file

1. In the `NL_CONV` section, choose an entropy prescription, e.g.

```
NOM_CONV = 'conv_jmj',
NOM_ALPHA = 'entropy_magic13',
ALPHA = 2.0,
```

Since $\alpha$is adjusted iteratively, you need to provide a first guess for the initial model. Let us try $\alpha = 2.0$.

While f_μ is automatically computed, the value of $\mathrm{d}s$ must be set in the `.don` file. To do so, add:

```
&NL_RLG
/
&NL_EXTRA
extra%alpha_autoguess = F,
extra%ds = set you ds value here,
extra%ec_grid = 'cifist'
/
```

Run the model

2. Run the model. Now, take a look at the file `HRnew`. If it stays empty for more than 2 minutes, kill the process, and restart by increasing (or decreasing) α by 0.2. If the first time step is computed successfully, but then next time steps take forever to compute, it may be because your initial guess for α was still too far from the correct value. Look for the `alpha` column in `HRnew`, remember value in 1st row, kill your model, and use this value as a new initial guess.

3. After the computation is on track, you can compare evolutionary track with your reference model with the command

```
plot_hr.py ref_model ec_model -lm o
```

For grid computation, an automatic guess can be estimated with option

```
extra%alpha_autoguess = T
```

but it requires more preparation work.

2.3 Atomic diffusion and adhoc turbulent diffusion coefficient

The options for atomic diffusion and turbulent mixing can be found in the `model.don` file in the `NL_DIFF` namelist:

```
&NL_DIFF
DIFFUSION = T,
NOM_DIFFM = 'diffm_mp',
NOM_DIFFT = 'difft_cte',
NOM_DIFFT_FING = 'brown2013',
NOM_DIFFT_SMC = 'no_smc',
```

```
D_TURB = 0.0,
RE_NU = 0.0,
NOM_FRAD = 'no_frad'
/
```

or in the tab `diffusion` of the graphical interface.

2.3.1 Atomic diffusion

`DIFFUSION` can be True or False. If False, no atomic diffusion is taken into account and the transport in convective zones is considered instantaneous. If True, atomic diffusion is taken into account and the convective transport is diffusive with a turbulent diffusion coefficient defined by EXTRA%d_conv. `NOM_DIFFM` is the name of the atomic diffusion routine. There are three choices (they can be found in `diffm.f90`):

- `diffm_0` includes no atomic diffusion but the diffusion equation is solved for other diffusive processes.
- `diffm_mp` includes the Michaud & Proffitt 1993 formalism.
- `diffm_br` includes the Burgers 1969 formalism. Note that radiative accelerations cannot yet be used with this option.

`NOM_FRAD` defines the radiative acceleration routines. There are three choices (they can be found in `f_rad.f90`):

- `alecian2004` includes Alecian & LeBlanc 2004 Single Valuate Parameters (SVP) method.
- `alecian2020` includes the updated Alecian & LeBlanc 2020 SVP method.
- `no_frad` includes no radiative accelerations.

For radiative accelerations to be included the specific nuclear reactions nets in which `grad` appears in the name are required, namely `NOM_NUC = 'ppcno9grad'` and `NOM_NUC = 'ppcno13gradNi'`. The computation of the Rosseland mean opacity can be done with the OP monochromatic opacities to follow the variation of mixture induced by atomic diffusion with the option `NOM_OPA = 'opa_mono_op'`. **With this option, the computation time is drastically increased and a minimum of 1Go of RAM should be available for the tables to be loaded.** The control parameter `EXTRA%goop` can be used to prevent the computation of radiative accelerations and the use of monochromatic opacity before a specific age given in Myr. Note that for the moment, radiative accelerations are automatically disabled after the TAMS for numerical stability and because the process is almost negligible at this evolutionary stage (see Moedas et al. 2022). Extra outputs can be provided for the diffusion velocities and radiative accelerations with the option `NUM%vdiff_out = .TRUE.` and `NUM%grad_out = .TRUE.`, respectively.

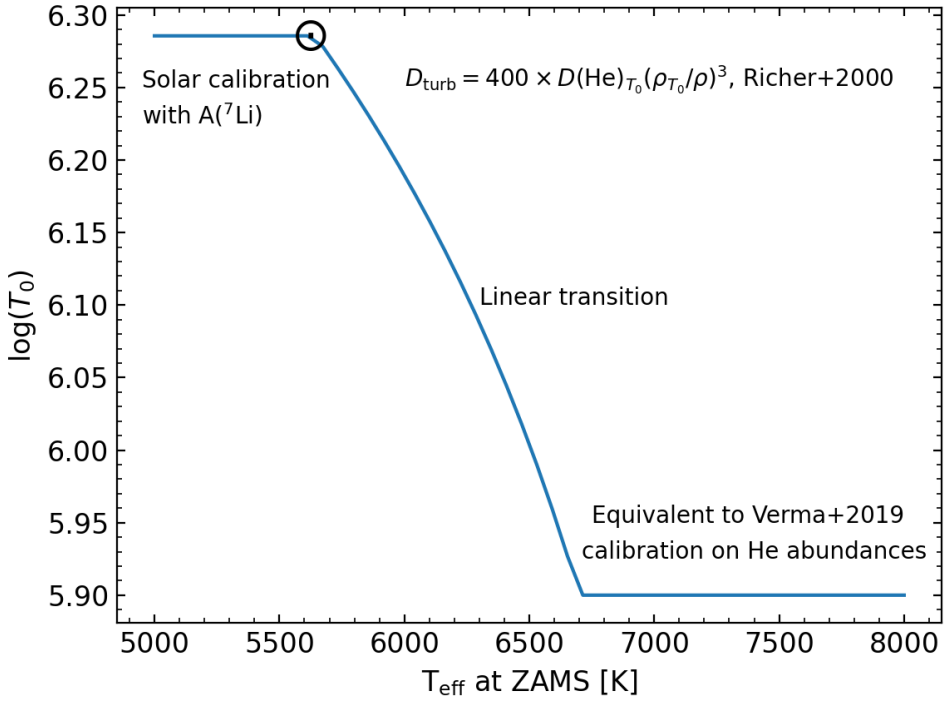

Figure 2.1: Variation of $D_{\rm turb}$ with the `difft_grille` option.

2.3.2 Turbulent mixing

`NOM_DIFFT` is the name of the adhoc turbulent diffusion coefficient routine. There are eight choices (they can be found in `difft.f90`):

- `difft_c0` includes no additional turbulent diffusion coefficient.
- `diffm_cte` includes a constant diffusion coefficient defined by `D_TURB` in $cm^2.s^{-1}$.
- `diffm_expo` include a double exponential decrease of the turbulent diffusion coefficient $D_{\rm turb} = \omega_1 \exp(\ln(2) * \frac{(r-r_{\rm cz})}{R^* \sigma_1}) + \omega_2 \exp(\ln(2) * \frac{(r-r_{\rm cz})}{R^* \sigma_2})$.
  ω_1 and ω_2 are defined by `D_TURB` and `EXTRA%D_expo_2`, respectively. σ_1 and σ_2 are defined by `EXTRA%L_expo` and `EXTRA%L_expo_2`, respectively. $r_{\rm cz}$ is the radius of the base of the convective envelope.
- `diffm_montreal` includes a profile for the turbulent diffusion coefficient following Richer et al. 2000: $D_{\rm turb} = \omega D(\mathrm{He})_0 \left(\frac{\rho_0}{\rho}\right)^{\rm n}$ where ω and n are constants defined by `D_TURB` and `EXTRA%expo_mix`, respectively. ρ and $D(\mathrm{He})$ are the local density and diffusion coefficient of helium. The indices 0 indicate that the value is taken at a reference depth. The reference

depth can be defined either in mass of the envelope (according to the total mass of the star) or in temperature by the parameter `EXTRA%M_mix` and `EXTRA%T_mix`, respectively.

- `diffm_montreal_cz` includes profiles for the turbulent diffusion coefficient following Richer et al. 2000 imposing the reference depth to the base of the convective envelope. ω and n are constants defined by `D_TURB` and `EXTRA%expo_mix`, respectively.

- `diffm_grille` includes the expression of Richer et al. 2000 but with two different values depending on the effective temperature at the ZAMS for the model. Between the two points in the grid, a linear variation in depth of the mixing is applied. An example of the variation of the depth is given in Fig. 2.1. ω and n are constants defined by `D_TURB` and `EXTRA%expo_mix`, respectively. The reference depth can be defined only in temperature by the parameter `EXTRA%t0_Sun` for the first point (generally the Sun is used as reference) and by `EXTRA%t0_2` for the second reference point. The selection points in effective temperature are defined by `EXTRA%teff_zams_Sun` and `EXTRA%teff_zams_2`.

- `diffm_gab` extends the convective mixing for $t < 10^6$ K.

- `diffm_sun` includes a specific turbulent diffusion coefficient profile for the Sun following M. Gabriel's work.

An extra constant turbulent diffusion coefficient can be added to `diffm_morgan`, `diffm_morgan_2` and `diffm_grille` with `EXTRA%cst_mix`.

The transport induced by the rotation is always added to the diffusion coefficient.

2.3.3 Thermohaline convection

`NOM_DIFFT_FING` is the name of the thermohaline convection routine. There are two choises (they can be found in `difft_fing.f90`):

- `no_fing` no thermohaline convection.

- `brown2013` include thermohaline mixing with the formalism of Brown et al. 2013.

An extra output can be provided with the option `NUM%th_out = .TRUE.`.

2.3.4 Semi-convection

`NOM_DIFFT_SMC` is the name of the semi-convection routine. There are two choices (they can be found in `difft_smc.f90`):

- `no_smc` no semi-convection.

- `langer1983` include thermohaline mixing with the formalism of Langer et al. 1983.

2.3.5 Transport induced by radiative viscosity

RE_NU is the radiative viscosity diffusion coefficient defined in Morel & Thevenin 2002. This coefficient is added if RE_NU>0. It has been shown by Alecian & Michaud 2005 that the physical meaning of this process was questionable. **We strongly advise not to include values different from** 0 **for this parameter.**

2.3.6 Pratical exercice

1. Compute a first model without diffusive transport.
2. Compute one or more models using any combination of these prescriptions.
3. Plot the variations of the surface abundances for different elements for comparison.

2.4 Rotation

The transport of angular momentum is one of the biggest issue in stellar physics. Currently models perfectly fail at reproducing observed stellar rotation rates. Cesam2k20 implements many options to model such transport but remains unrealistic. In this section, we will therefore learn how to not model properly angular momentum distribution.

2.4.1 A reminder

Since convective zone are fully mixed, a precise modelling of the transport of angular momentum (AM) in these regions will not results in strong improvement of global parameters determination, e.g. of the age. Convective envelope only looses AM because of magnetic braking. Therefore, we assume solid-body rotation in CZ and we focus on radiative zones (RZ). There, the classical transport AM model relies on three main hypotheses. First, baroclinic equilibrium is rotating stars is fulfilled because a large scale circulation, the meridional circulation, takes place, which advects AM. Second, differential rotation creates shear that induces turbulence because of the Richardson instability. This shear induced turbulence diffuses angular velocity. Third, since this diffusion is much stronger horizontally than vertically, we assume the angular velocity is constant in shells (shellular approximation) which allows to keep a 1D description.

There are two ways to initialize rotation in Cesam2k20. You can either set a given rotation rate, or use the disk-locking model. In this case, it is assume that when the star forms, the magnetic field forces convective zones to co-rotate as a solid body with the remaining accretion disk. Then Cesam2k20 only needs to know the life time of the disk $\tau_{\rm disk}$ and its period of rotation $P_{\rm disk}$.

2.4.2 Available options

Options for controlling rotation in the .don file are the following:

```
&NL_ROT
W_ROT = 1e-06,
UNIT = 'kms/s',
NOM_DIFFW = 'diffw_mpz',
NOM_DIFFMG = 'diffmg_0',
NOM_GSF = 'no_gsf',
NOM_THW = 'diff_tz97',
ROT_ZC = 'solide',
NOM_PERTW = 'pertw_matt15',
P_PERTW = 1.2260825804832511e+31,
LIM_JPZ = T,
MIXED_MODES = F,
INIT_ROT = F,
NOM_DES_ROT = 'no_des',
TAU_DISK = 5.0,
P_DISK = 10.0,
W_SAT = 10.0
/
```

- `W_ROT` and `UNIT`: Initial angular velocity and its unit. Only used if $\tau_{\text{disk}} \leq 0$ and $P_{\text{disk}} \leq 0$.
- `NOM_DIFFW`: name of the prescription for the coefficients of the shear-induced turbulence. Can be
  - `'diffw_0'`: No diffusion;
  - `'diffw_p03'`: Prescription of Palacios (2003);
  - `'diffw_mpz'`: Prescription of Mathis et al. (2004);
  - `'diffw_mathis2018'`: Prescription of Mathis et al. (2018);
  - `'diffw_cte'` : Constant D_{eff}, D_{h}, D_{v}, set as `extra`parameters.
- `NOM_DIFFMG`: name of the prescription used for AM transport by Tayler-Spruit (TS) instability:
  - `'diffmg_0'`: No TS instability;
  - `'diffmg_ts'`: Prescription of Spruit et al. (2002).
- `NOM_GSF`: name of the prescription used for AM transport by GSF instability instability:
  - `'no_gsf'`: No GSF instability.
  - `'hirschi10'`: Prescription of Hirschi et al. (2010).

  - 'barker19_eq': Prescription of Barker et al. (2019) at the equator.

- NOM_THW: name of the prescription used for the angular momentum transport:
  - 'rot_0': No AM transport;
  - 'cons_glob_mnt_cin': constant angular velocity;
  - 'cons_loc_mnt_cin': constant angular momentum;
  - 'diff_tz97': Prescription of Talon & Zahn (1997) (recommended for stability);
  - 'diff_mz04': Prescription of Mathis & Zahn (2004).

- ROT_ZC: Rotation in CZ:
  - 'solide': constant angular velocity;
  - 'local_j': constant angular momentum.

- NOM_PERTW
  - 'pertw_0': No mass loss;
  - 'pertw_kaw': Prescription of Kawaler et al. (1988);
  - 'pertw_matt15': Prescription of Matt et al. (2015);
  - 'pertw_rm': Prescription of Reiners & Mohanty (2012)
  - 'pertw_sch': Prescription of Schumanisch.
  - 'pertw_loc': Loss of AM according to a local law: $\dot{J}_{\rm W} = TR_\star^2\Omega_\star^2$, where T is a constant.
  - 'pertw_ptm': loss of AM from mass-loss.

- P_PERTW: Value of the constant involved in the various prescriptions.

- LIM_JPZ: If true, assumes $\mathrm{d}\Omega/\mathrm{d}m = 0$, else, $U_2 = 0$ (U_2 is the velocity of the meridional circulation) at the transition RZ/CZ (Warning: is almost not used any more).

- MIXED_MODES: Account for AM transported by mixed modes.

- INIT_ROT: True if angular velocity profile is read from a file (warning: do not use).

- NOM_DES: Legacy option, do not use.

- TAU_DISK: Lifetime of the disk (in Myrs).

- P_DISK: Rotation period of the disk (in days).

- W_SAT: Angular velocity at which the dynamo saturate.

2.4.3 Practical exercise

1. Play with these options !
2. For models with a given mass, compute various models with different combinations of `TAU_DISK` and `P_DISK`. Plot the surface angular velocity through time.
3. Do the same, adding AM loss.

2.5 Running grids

To build your grid you will need the files `input.in`, `run_grid.in` and `template.don` available in the `/cesam2k20/python3` folder.

2.5.1 Creation of the .don files

To build the `.don` file for your grid, modify the template.don file with all the inputs you want for the inputs physics, the outputs and the precision.

Then modify the `input.in` file so that all the parameters you want to vary appear.

```
# parameter |  ini  |  end  |  step
# ----------+-------+-------+-----------
mtot          0.7     1.5      0.1      #mass of the stars in solar mass
y0            0.24    0.30     0.01     #initial helium mass fraction
#z0            0.003   0.04     0.003    #initial metal mass fraction
alpha         0.5     0.8      0.05     #convection parameter
ovshts        0.0     0.3      0.05     #coreovershoot parameters in Hp
[Fe/H]0        -0.1    0.1      0.05     #initial iron abondance in dex
# ----------+-------+-------+--------
#           |  ini  |  end  |  Y_p
# ----------+-------+-------+--------
dYdZ          0.4     3.0     0.248    #limit the dydZ value, comment to ignore
# ----------+-------+-------+--------
overshoot     n                        #limit the overshoot values
sobol         1000                     #use the Sobol algorithm, comment to
↪  ignore
```

Note that `z0` and `[Fe/H]0` cannot be used at the same time. You can also add parameters that will not vary (e.g. alpha and ovshts to take a `.don` value). The first column is the name of the parameter in the `.don` file, the second column is the minimum value, the third column the maximum value and the last column is the step. Anything that is not entered in the `input.in` file will be taken from the `template.don` file and will not be modified for the entire grid. Once you have modified your `template.don` and `input.in` files, enter the command :

Figure 2.2: Overview of the directory for grid computation.

```
make grid.py input.in <grid_name> [init number (continuation of existing grids)]
```

Your `.don` files will be created as you can see in Fig.2.2. An index file is also created to give the correspondance inputs/model number. The `.don` files can be erased with the command:

```
make grid.py --clean <grid_name>
```

2.5.2 Computation of the grid

Once you have created your `.don` files, you need to modify the `run_grid.in` file to indicate whether the models start on PMS or ZAMS.

```
# parameter |   value
#-----------+----------------
job           zams              # how to build initial model (pms or zams)
type_file     ASCII             # type of file
mod_init      m010.zams         # name of file
c_iben        5e-4              # Iben number for PMS model
folders       n                 # are the models stored in folders?
#-----------+----------------
freqs         y                 # compute frequencies?
modes         p-modes
lmax          3
lmin          0
dels          1
remesh        a
npoints       4800
sig1          500
sig2          3000
nsig          Frequency
nstep         100
```

Some files for the PMS or ZAMS initial models can be found in `cesam2k20/SUN_STAR_DATA`. Once you have modified your `run_grid.in` file, you can calculate your grid using the command:

```
run_grid.py run_grid.in <grid_name> [--ncore=<ncores>]
```

<ncores> being the number of cores you wish to use to calculate the grid, optional.

2.6 Interpolating in grids

Cesam2k20 is coupled with two codes allowing the interpolation in grids, namely AIMS and SPInS. Converting Cesam2k20 outputs into AIMS and SPInS inputs is explained below.

2.6.1 AIMS

AIMS can be downlaoded here: https://gitlab.com/sasp/aims where the installation steps are described. Note that the informations provided below only work for version 2.0 and after. AIMS needs a list file in a particular format in order to build the binary file for the grid. The first line should be the path where the frequency files are stored followed by the the extension of the frequency files. The following lines should contain at least, and in this order: the name of the frequency file without the extension, the mass in g, the radius in cm, the luminosity in erg/s, Zini , Xini , age in million years, the $T_{\rm eff}$, and an adimensional age. You can then add all the parameters you wish to optimise with AIMS. To build this file you can enter the command in the folder where the models are stored:

```
convert2aims.py input.in <path_to_the_frequency_files> <extension> (without the
↪  dot, e.g. agsm)
```

The `convert2aims.py` file is located in `cesam2k20/python3` and can be modified to add the extra parameters.

2.6.2 SPInS

SPInS can be downloaded here: https://dreese.pages.obspm.fr/spins/project.html where the installation steps are described. Similarly to AIMS, SPIns needs a list file in a particular format in order to build the binary file for the grid. The format is an adimensional age, the age in million years, the mass in solar units, the luminosity in solar units, the $T_{\rm eff}$, the radius in solar units, and [M/H].

To build this file you can enter the command in the folder where the models are stored:

```
convert2spins.py
```

Again the script can be modified in `cesam2k20/python3` to add extra parameters.

2.7 Optimal Stellar Models (OSM)

The goal of the Optimal Stellar Model (OSM) program is to find the model that corresponds the best to a set of observational constraints, by tuning free parameters. It was developed by Reza Samadi (https://pypi.org/project/osm/) and recently refactored by L. Manchon to handle multiple systems. OSM is based on a Levenberg-Marquardt algorithm and can deal with non seismic and seismic constraints. OSM only works with the Cesam2k20 code and is distributed with it under the name of `osm3`. This section explains how to use it.

2.7.1 How it works

OSM tries to minimize the distance χ^2 between some observed and theoretical stellar properties:

$$\chi^2 = \sum_i \left(\frac{X_i^{\rm obs} - X_i^{\rm model}}{\sigma_i} \right)^2 , \tag{2.4}$$

where $X_i^{\rm model}$ (resp. $X_i^{\rm obs}$) is the theoretical (resp. observed) value of stellar parameter X_i and σ_i the uncertainty on the observed value.

OSM needs two files to work:

- A Cesam2k20 `.don` file that contains all the parameters of the models, especially the parameters that will not be tuned by OSM.
- A XML file that tells OSM what it needs: observable constraints, free parameters, and parameters that control the run of the Cesam2k20 models, the computation of oscillations, and the options of the Levenberg-Marquardt algorithm.

Both files must have the same name (without the extension).

An example of XML file is given in `A_LA_CARTE/OSM/example.xml`. The file contains four types of fields.

- `<parameters>`: contains information on a free parameter OSM must adjust. For example:

```
<parameter name="agemax">
    <value> 300.  </value>
    <step> 20. </step>
    <rate> 5. </rate>
    <bounds> 10. , 10000.</bounds>
</parameter>
```

  where the `name` is the name of free parameter, `<value>` is the value of the first guess, `<step>` is the amount by which agemax is shifted to compute the numerical derivative needed by the algorithm, `<rate>` controls the correction applied to the parameter between

consecutive iterations (see Eq. (2.5)), and <bounds> is the minimum and maximum value allowed for the parameter. Between two iterations, the value of the parameter is computed with:

$$p^{i+1} = \max\left[\min\left[p^i + \text{sign}(\Delta p) \times \min\left[|\Delta p|, \left|p^i \frac{\text{rate}}{100}\right|\right], p_{\max}\right], p_{\min}\right], \tag{2.5}$$

with p^i the value of a parameter at iteration i, Δp the correction found by the Levenberg-Marquardt algorithm, rate is the value defined in <rate> and $(p_{\min}, p_{\max})$ corresponds to values defined in <bounds>.

- <target>: contains information about a non-seismic constraint. For example:

```
<target name="logg">
    <value>4.5181</value>
    <sigma> 0.01</sigma>
</target>
```

where the name is the name of observable, <value> is the value OSM will try to reach with the optimal model and <sigma> the uncertainty on the measure.

- <seismic_constraints>: contains information on seismic constraints. For example:

```
<seismic_constraints>
    <file>tablefreq.txt</file>
    <types>d01</types>
    <matching>frequency</matching>
</seismic_constraints>
```

where <file> is the name of the file that contains individual mode frequencies, <type> specifies the type of seismic constraint that OSM must check (individual frequencies, large separations, ratios, etc.; user can set several values), and <matching> is the method followed to associate theoretical to observed frequencies (match closest frequencies, same radial order, etc.).

- <settings>: contains options that controls the run of Cesam2k20 models, the computation of oscillation frequencies, and the Levenberg-Marquardt algorithm itself. For instance:

```
<settings>

<models>
    <start>pms</start>
```

```
    <cf>5e-6</cf>
    <dy_dz> 2. </dy_dz>
    <yp>0.248</yp>
    <zp>0.</zp>
</models>

<modes>
    <l>0,1</l>
    <nmin>10</nmin>
    <nmax>20</nmax>
    <dn>1</dn>
    <oscprog>adipls</oscprog>
</modes>

<levmar>
    <maxiter> 25 </maxiter>
    <chi2min> 1e-3 </chi2min>
    <ftol> 1e-3 </ftol>
</levmar>

</settings>
```

- `<models>`: `<start>` is the starting point of each model computation, `cf` is the Iben's constant (needed only if you start from the PMS). `<yp>`, `<zp>` and `<dy_dz>` are needed to choose the chemical composition when diffusion is included and initial helium or metal abundances are not among the targets. In this case, initial Y and Z are linked by:

$$Y = Y_{\rm p} + \frac{\Delta Y}{\Delta Z}(Z - Z_{\rm p}), \tag{2.6}$$

  where $Y_{\rm p}$ and $Z_{\rm p}$ are the primordial helium and metal abundance, and $\Delta Y/\Delta Y$ is the helium to metal enrichment rate. Classical values are $Y_{\rm p} = 0.248$, $Z_{\rm p} = 0.0$ and $\Delta Y/\Delta Y \simeq 2$ (for the Sun, obtained from solar calibration).

- `<modes>`: contains options that control the computation of theoretical oscillation spectra of Cesam2k20 models. `<l>` corresponds to the mode degree used in the comparison with seismic constraints (must match exactly the degrees given in the frequency file), `<nmin>` and `<nmax>` are the minimum and maximum radial order used (must match exactly the minimum and maximum order given in the frequency file), `<dn>` is the tolerance used to compute mean large separation (it is computed by shifting radial order of `-dn`, `-dn-1`, etc.), and finally, `<oscprog>` corresponds to the code used

to compute oscillation frequencies. If used, you must also provide an input file in the format used by the code.

- `<levmar>`: contains options that control the algorithm. Maximum number of iterations are controlled through `<maxiter>`, the minimum χ^2 value at which we stop through `<chi2min>` and the minimum χ^2 variation allowed through `<ftol>` (otherwise OSM stops).

OSM is run with

```
osm.py name_of_model
```

where files needed by OSM are called `name_of_model.xml` and `name_of_model.don`. Results are stored in directory `name_of_model`. Optimal model is renamed `name_of_model_fin`.

2.7.2 Practical exercise

A first simple case

1. Run OSM with the test files `test1` provided in `A_LA_CARTE/OSM/`. Try to understand output on screen, find the optimal model in `test1` directory. Check that $\log g$ and $T_{\rm eff}$ of the optimal model correspond to targeted values.

2. Create you own setup file to reproduce a standard solar model as in Chapter 1. You will tune α and Y_0, to reproduce solar luminosity and radius at present solar age. Use default option for the Cesam2k20 models.

Seismic constraints *(The runs of OSM in this section take a long time to converge. Don't hesitate to start another exercise in parallel, or to grab a cup of coffee.)*

1. Add an additional target `"largesep"`, with solar large frequency separation ($\Delta\nu_\odot \simeq 134$ μHz). Add also an additional free parameter `"zsx0"`, the initial Z/X ratio (try $Z/X = 0.018$ as first guess). In the `<settings>` section, add

```
<modes>
    <l>0,1</l>
    <nmin>10</nmin>
    <nmax>20</nmax>
    <dn>1</dn>
    <oscprog>adipls</oscprog>
</modes>
```

Finally, include microscopic diffusion in your model (see in Sect. 2.5 how to modify the `.don` file). For OSM to work, you finally need to include an input ADIPLS file. You can

find one called `adipls.in` in `A_LA_CARTE/OSM/`. Copy it in your directory. ADIPLS will compute frequencies with a 4^{th} order shooting method, and a re-meshing on 8000 points, adapted to solar p-modes. If you want to know more about ADIPLS, documentation can be downloaded with the code here: https://users-phys.au.dk/~jcd/adipack.v0_3b/.

2. The target `"largesep"` allows you to compare average large separation in a given set of frequencies. You can also compare separation $\Delta\nu_{n\ell}$ for all modes of consecutive order in a given set of frequencies. File `solar_frequencies.txt` in `A_LA_CARTE/OSM/` contains a large set of measured solar frequencies. OSM only accepts ASCII tables with 4 columns corresponding to n, ℓ, ν and associated uncertainty. For example:

```
#  n l nu [muHz] sigma [muHz]
11    0    1686.5511   0.1019
13    0    1957.2972   0.0963
14    0    2093.3608   0.0920
```

From the file `solar_frequencies.txt`, create a table for OSM, name for instance `tablefreq.txt`. Use only $m = 0$ modes. Values of n and ℓ must also span the same range as given in the field `<modes>` in above question.

3. Use same settings as in 1., and replace the `"largesep"` target by

```
<seismic_constraints>
    <file>tablefreq.txt</file>
    <types>dnu0</types>
    <matching>frequency</matching>
</seismic_constraints>
```

Run OSM.

Binaries *(The runs of OSM in this section are even longer than before. Don't hesitate to grab two cups of coffee.)*

OSM has recently been adapted in order to work with multiple systems. It allows to impose free parameters (e.g. age or chemical composition) identical for all stars in the system. These parameters are declared in the XML file as usual. Free parameter adjusted individually for each star are declared as

```
<parameter name="alpha!dr_jekyll">
    <value>1.65 </value>
    <step> 0.01 </step>
    <rate> 3. </rate>
    <bounds> 1.5, 2.5</bounds>
```

```
</parameter>

<parameter name="alpha!mr_hyde">
    <value>1.65 </value>
    <step> 0.01 </step>
    <rate> 3. </rate>
    <bounds> 1.5, 2.5</bounds>
</parameter>
```

where `dr_jekyll` and `mr_hyde` are some names that you give to the stars in the system. If some parameter is known from observations, and different for each star (e.g. mass from interferometric measurements), it needs to be written in the XML file as a target, but specifying it is an input:

```
<target name="mtot!dr_jekyll">
<value>0.9373</value>
<sigma> input </sigma>
</target>

<target name="mtot!mr_hyde">
<value>1.1055</value>
<sigma> input </sigma>
</target>
```

A synthetic young (PMS) binary system has the following properties:

	1st star	2nd star
$T_{\rm eff}$ [K]	5697 ± 50	6090 ± 50
$\log g$	4.47 ± 0.01	4.45 ± 0.01
Mass [$M_\odot$]	1.0	1.15

The mass is supposed to be known exactly.

1. You will tune the age (fixed for both stars), and the $\alpha_{\rm MLT}$ parameters.
2. Create targets for both stars.
3. Assume very simple physics, the run can take a long time. Run OSM.

2.8 Interface with oscillation code

Cesam2k20 is coupled with two oscillation evolution codes: ADIPLS[1] and the ACOR (2D non-perturbative oscillation code, Ouazzani et al. 2012).

[1] https://users-phys.au.dk/~jcd/adipack.v0_3b/

2.8.1 Practical exercise

1. Open the GUI, leave default options unchanged. Then tick "Calculate frequencies", and open the "Oscillation option" tab.

2. Choose ADIPLS, try to understand options and configure it the best you can (you can use the notes provided by JCD with the ADIPLS package). Click ok. When the model and the frequencies are computed, the frequencies of the last model are stored in `agsm` file.

3. Do the same but choose ACOR. In order to not recompute the model, just untick "Compute model". Try to understand options and configure it the best you can. The frequencies of the last model are stored in a file `_1d.acor`.

4. Compare frequencies from both codes and try to understand origins of the differences.