Manual for v. 8.3.1

Sylvia Ekström

April 17, 2019

Contents

1 Installation and usage

GENEC_toolBox is a set of python routines designed to exploit the files generated by the Geneva evolution code and the SYCLIST tool for clusters and isochrones. It processes the files and provides the possibility to plot any variable as a function of any other one.
It can be downloaded on GitHub: https://github.com/GESEG/GENEC_toolBox

It requires

- python (tested on 2.7 and 3.5)
- matplotlib >= 1.2,
- numpy >= 1.7,
- and scipy >= 0.12.

1.1 Installation

You just have to save the `.py` file and the data directory in your favorite location.

To use it, open a terminal.

- log into a `ipython` console: `ipython --pylab=auto` (`--pylab=osx` for Mac OSX users);
- declare the directory where the routine sits: `sys.path.append('path/to/directory')`;
- import the set of routines:
  1. either with the command `from GENEC_toolBox import *`,
  2. or `import GENEC_toolBox as GtB` (or whatever short name you wish);
- use the commands either directly (if loaded with method 1) or with `GtB.`*`command`*.

Note that depending on your system, you might need to `import sys` first.

1.2 First run

The first time you run it, you will be asked to define the location of the data directory. Leave blank if you installed the package. Enter the path to the data directory in case you use the standalone. After that, you'll be asked to define the default figure directory. Leave blank to use the current location.

Your settings will be saved in a hidden config file `~/.GENEC_toolBox.ini`.

1.3 Usage in scripts

You can easily use the functionalities of GENEC_toolBox in python or ipython scripts. If you are the only one using these scripts, the simplest way is to import the program at the beginning of your script just as you do when you want to use it directly:

```
import sys
sys.path.append('/path/to/directory')
import GENEC_toolBox as GtB
```

If you think your routines can be useful for others, I recommend you use a more generic piece of code, by using the configuration file to get the path to the program:

```
import sys
import ConfigParser
Config = ConfigParser.ConfigParser()
Config.read(os.path.expanduser('~/.GENEC_toolBox.ini'))
toolBox_dir=Config.get('Paths','ProgPath')
sys.path.append(toolBox_dir)
import GENEC_toolBox as GtB
```

1.4 Quick outlook

The philosophy of GENEC_toolBox is to load a file, define the x-axis, and then plot the desired y variable.

Evolution files (`.wg`, `.dat`, grids format) are loaded with

- `loadE()`
- `loadEFromList()`
- `loadEFromDir()`.

Structure files (`*.v*`) are loaded with

- `loadS()`
- `loadSFromList()`
- `loadSFromDir()`.

Cluster / isochrones files generated by SYCLIST are loaded with

- `loadC()`
- `loadCFromList()`
- `loadCFromDir()`.

The variables available for plotting can be printed with `VarEvol()`, `VarStruc()`, or `VarCluster()`. New variables can be determined by using the command `Get_Var()` and `Set_Var()`, see example in the comments of the latter. After `Set_Var()`, the variable is available for plotting.

The x axis variable is set by `defX()`. By default, it is t [Myr] (`'t6'`) in EVOL mode, M_r/M (`'Mfrac'`) in STRUC mode, and $T_{\rm eff}$ (`'Teff'`) in CLUSTER mode. The plot command is `Plot('y')` for normal plots, and `Plot_colour('y')` for plots with lines or points colour-coded by the value of another variable. In CLUSTER mode, histograms can be plotted with the command `Histo()`.

Several pre-programmed plots exist, as:

- `HRD_plot()`: HR diagram in L vs $T_{\rm eff}$. With the optional parameter `corr=True`: L vs $T_{\rm eff,corr}$ ($T_{\rm eff}$ corrected for the wind thickness). With optional parameter `spectro=True`: sHRD.
- `CMD()`: colour-magnitude diagram, with a large choice of colours
- `rhoT()`: $T_{\rm c}$ vs $\rho_{\rm c}$ in EVOL mode, and T_r vs ρ_r diagram in STRUC mode
- `YTeff()`: $T_{\rm eff}$ vs $X(^4{\rm He})_{\rm c}$
- `gTeff()`: $T_{\rm eff}$ vs $g_{\rm surf}$
- `Abund()`: plot of the abundances. In EVOL mode, with input parameter `'s'` for surface, `'c'` for centre. In STRUC mode, abundance profiles with input parameter `'p'`.
- `NCNO()`: N/C vs N/O diagram.
- `eps()`: energy generation zones for H, He, C, and neutrinos (STRUC mode only).
- `Coeff()`: plot of the diffusion coefficients.
- `Nablas()`: plot of the three Nablas ($\nabla_{\rm ad}$, $\nabla_{\rm rad}$, ∇_μ) in STRUC mode.
- `j_profiles()`: plots the profile of the specific angular momentum j_r as well as $j_{\rm Schwarzschild}$, $j_{\rm Kerr}$, and $j_{\rm Kerr,max}$.
- `Kippen()`: Kippenhahn diagram for a given model. With the optional parameter `burn=True`, addition of the burning zones from an existing `*.burn` file.
- `plotRatio()`: plot of the ratio between two variables or one variable and the given index of a second one
- `Summary_plots()`: set of plots on 4 windows based on Raphael Hirschi's `dhr14.py` (EVOL) or `structure10.py` (STRUC), summarising the evolution or structure of a model.

Examples of more complex plots are given in Sec. 8.

For any questions or suggestions, contact sylvia.ekstrom@unige.ch

2 Reading of files and model selection

2.1 Individual loading

- **Loading an EVOL file**
  Command: `loadE('#file',num_⋆)`
  The optional parameters are:
  – `num_deb=i`, to skip `i` lines at the beginning of the file (default: `0`);
  – `num_end=j`, to stop before reading `j` lines at the end (default: `-1`);
  – `format='fmt'`, where `fmt` is one of the following:
  · `o2013`: standard or reduced `.wg` files;
  · `tgrids`: grids files;
  · `tools`: files generated by the interactive online tools;
  · `bin`: binary version of the code;
  · `preMS`: pre-MS version of the code;
  · `old_hirschi`: files from the 2004 version of the code;
  · `starevol`: starevol evolution files;
  If omitted, the format will be automatically detected (default).
  – `forced=True`, to skip the verification of the number entered as num_⋆ (default: `False`);
  – `wa=True`, to read an associated `.wa` file (only possible for `.wg` files, default: `False`);
  – `quiet=True`, to switch off the printings (default: `False`).
  Usage: `loadE('~/calc/Z002/P015z02S0/P015z02S0.wg',1)`
  `loadE('~/grids/tables/M015z02S0.dat',2)`

- **Loading a STRUC file**
  Command: `loadS('#file',num_⋆)`
  The optional parameters are:
  – `toread=i`, to read a given structure located in the file (default: all);
  – `format='fmt'`, where `'fmt'` is one of the following:
  · `o2013`: standard `.v` file;
  · `o2010`: old `.v` file;
  · `old_hirschi`: files from the 2004 version of the code;
  · `full`: full `StrucData` file;
  If omitted, the format will be automatically detected (default).
  – `quiet=True`, to switch off the printings (default: `False`).
  Usage: `loadS('P015z02S0.v0001001',1)`
  `loadS('P015z02S0.v0001051.gz',2)`
  `loadS('P015z02S0_StrucData_0001001.dat',3)`
  NB: The structure files don't need to be unzipped before loading.

- **Loading a CLUSTER file**
  Command: `loadC('#file',num_⋆)`
  The optional parameters are:
  – `num_deb=i`, to skip some lines at the beginning (default: `0`);
  – `format='fmt'`, where `'fmt'` is one of the following:
  · `cluster`;
  · `isochr`;
  If omitted, the format will be automatically detected (default);
  – `random=n`, to load only `n` randomly selected lines from the cluster;
  – `quiet=True`, to switch off the printings (default: `False`)
  Usage: `loadC('Cluster_z0.014_t07.900.dat',1)`
  `loadC('Isochr_Z0.006_Vini0.40_t09.240.dat',2)`

2.2 Grouped loading

- **Loading files from a list**
  Command: `loadXFromList('file_name')`
  where `X` stands for one of the following: `E`, `S`, or `C` and `file_name` is a text file containing the list of files to be loaded (be sure to write the full path).
  The optional parameters are:
  – `ini_index=i`, to set the num_⋆ of the first model (default: `1`).
  The numbering of the `n` models in the list is automatically set from `i` to `i+n` except in STRUC mode where more than one structure might exist in the file;
  – `format='format'`, to force the format definition (default: auto-detection);

– forced=True, to skip verification of the number entered as num_⋆ (default: False);
– quiet=True, to switch off the printings (default: False).

Usage: loadEFromList('MyLoadedFiles.txt')
loadCFromList('MyLoadedClusters.txt',12)

- **Loading models directly from a directory**

Command: loadXFromDir('dir_name')
where X stands for one of the following: E, S, or C

This command loads automatically all suitable files existing in the given directory:
– all .wg, .dat, or .grids files (EVOL);
– all *StrucData*.dat, or .v files (STRUC);
– all Clu*.dat, or Iso*.dat files (CLUSTER).

The optional parameters are:
– select='string', to restrict the loading to files having string in their name (default empty);
– ini_index, to set the num_⋆ of the first model (default: 1);
– wa, to read both wg and wa files;
– format='fmt', to set the format of the files. Left empty (default), the format is automatically detected;
– forced=True, to skip verification of the number entered as num_⋆ (default: False);
– quiet=True, to switch off the printings (default: False).

Usage: loadEFromDir('Grids2010/tables/Z014/')
loadEFromDir('Grids2010/wgred/Z014',select='S0',ini_index=12)
loadCFromDir('SYCLIST_Clusters/sylviaegn','v0.50',3)

2.3 Various commands

- **Adding a column for the reading of modified files**

Command: add_column(['varName',col_num],'label','category')
Back to normal with the command standard_columns().

- **Reloading a file**

Command: reloadX(num_⋆)
where X stands for one of the following: E, S, or C

NB: the options when loading the file are remembered, so only the num_⋆ is needed; to reload the file with different options, use the standard loadX() command with forced=True

2.4 Models selection

- **Switching from one mode to another**

Command: switch('mode')
where mode is one of the following: evol, struc, or cluster.

- **Selection:**

Command: select_model(num_⋆)
This command defines entirely the selected models (erases the previous selection)
Usage: select_model(2)
select_model([1,2,3,4,5])
select_model(range(1,6)) (same result as above)

Command: select_all()
This command selects all the loaded models of the current mode.

- **Addition of a model:**

Command: add_model(num_⋆)
This commands keeps the previous selection of models and adds the num_⋆ one(s)

- **Removal of a model:**

Command: del_model(num_⋆)

- **Recall of loaded models:**

Command: Loaded('mode')
where mode is one of the following: evol, struc, cluster. Called without argument, it refers to the current mode.

3 Variables

3.1 Basics

- **List of available variables**
  Command: `VarEvol()` (EVOL)
  Command: `VarStruc()` (STRUC)
  Command: `VarCluster()` (CLUSTER)
  (see also section 7)

- **Getting the values of a variable for a model**
  Command: `Get_Var('variable',num_⋆)`

- **Creation of a new variable**
  Command: `Set_Var(newVar,'Var_Name',num_⋆)`
  The optional parameters are:
  – `label='#LaTeX_string'`;
  – `category='category'`
  with the following existing categories (but it's possible to create one):
  · EVOL : `model,surface,centre,abundances,rotation,winds,energetics`
  · STRUC : `structure,thermo,EOS,energy,abundances,rotation,magnetism`
  · CLUSTER : `initial conditions,global properties,rotation,abundances,colours`
  Usage:
  ```
  VVc = Get_Var('Vsurf',1)/Get_Var('Vcrit1',1)
  Set_Var(VVc,'VVc',1,label='$V/V_\mathrm{crit}$',category='rotation')
  ```

- **Deletion of a new variable**
  Command: `Del_Var('varName')`
  with the optional argument num_⋆. Called without it, it deletes the variable for all loaded models.

3.2 Specific work on variables

- **Getting the derivative of `variable 1` by `variable 2`**
  Command: `Deriv('var1','var2')`
  Called with a number, or a list of numbers, restricts the derivative calculation to the models number given.
  Usage: `Deriv('M','t',[1,2])`

- **Splitting of a vector into its positive and negative components**
  Command: `Vector_split('variable',num_⋆)`
  Creates vectors `variable_pos` and `variable_neg`.

- **Computation of the colours in EVOL mode**
  Command: `colours_calc(num_⋆)`
  with the optional argument num_⋆. Called without it, it performs the computation for all the clusters loaded.
  It computes the magnitudes in `U,B,V,R,I,J,H,K`, and Gaia's `G,Gpb,Grp`,
  and the colours `U-B,B-V,V-R,V-I,J-K,H-K,V-K,G-V,Gbp-V,Grp-V,Gbp-Grp`

- **Correction of magnitude and colour for a distance modulus and reddening excess (only in CLUSTER mode)**
  Command: `colour_corr(excess,dist_modulus,mag=mag,col=colour)`
  It creates new variables called `mag'_corr'` and `col'_corr'` with `mag` et `col` the name of the magnitude and colour respectively. By default, `mag=M_V` and `colour=B-V`
  The optional argument `num_star=num_⋆` restricts the correction to the cluster(s)/isochrone(s) specified.

- **Addition of noise on a variable (only in CLUSTER mode)**
  Command: `add_noise('varName',errorValue)`
  It creates new variables called `'varName_noised`.
  The optional argument num_⋆ restricts the correction to the cluster(s) specified.

4 Graphics

The philosophy of plotting with GENEC_toolBox is to first define which variable will sit on the *x*-axis, and then to do the plot with a commande giving the *y*-axis variable.

4.1 Basics

- **Definition of the *x* axis**
  Command: `defX('variable')`
  As default, the axis is *t* [Myr] in EVOL mode, M_r in STRUC mode, and $T_{\rm eff}$ in CLUSTER mode.

- **Basic command**
  Command: `Plot('variable_y')`

- **Plot variable *y* only if a condition is fulfilled**
  Command: `Plot('y',plotif='condition')`
  where `condition` can be a simple string or a list of strings.
  Usage: `Plot('He4s',plotif=['H1c>0.'])`
  `Plot('gsurf',plotif=['Vsurf>200.','Vsurf<=300'])`

- **Graph with variable(s) in log**
  Command: `logVar('axis')`
  with `axis=x,y,z,xy,xz,yz`, then `Plot('variable')`. Back to real values with `no_logVar('axis')`.
  NB: to plot the real value but on a logarithmic scale, use `logScale('axis')` (described in section 5.1)

- **3rd variable in colour code over the curve**
  Command: `Plot_colour('var_y','var_z')`
  The optional arguments are:
  – `binz=n`: to limit the number of colours in the map
  – `s='var_s'`: to resize the points according to the value of `'var_s'` (automatic switch to `Points(True)`);
  – `logs=True`: to take the log of this `'var_s'`;
  – `plotif='condition'`: to limit the plot to a given condition (cf. page 9);
  – `ticks=[tick list]`: to define the location of the colour bar ticks.
  The colour map can be changed with the command `set_colourMap('cmap')` (see Sec. 5.2, page 15)

4.2 Multiple plots

- **Superimposing plots**
  Command: `plot2var('mode')`
  By default, the mode is `'same'` that draws all the variables on the same *y* axis.
  The mode `'double'` allows to draw the 2nd variable with an independent axis on the right.
  NB: back to normal with `plot1var()`

- **Keeping the plot to draw something else on it**
  Command: `keep_plot(True)`
  Back to normal with `keep_plot(False)`

4.3 Predefined graphs

- **Ratio between two variables**
  Command: `plotRatio('var_1','var_2')`
  The optional argument `index=#line` allows to divide variable 1 by the value of variable 2 at that line. Useful for example to plot the relative evolution of a variable with respect to its initial value: `plotRatio('var','var',index=0)`.

Several predefined plots can be drawn with the following commands, without the need to define the *x*-axis beforehands. Depending on the mode (`evol`, `struc`, or `cluster`), different options can be available.

4.3.1 Graphs in EVOL mode

- **HR diagram**
  Command: `HRD_plot()`
  The following optional arguments are available:
  - `spectro=True`: for the sHRD ($T_{\rm eff}^4/g$ vs $T_{\rm eff}$);
  - `corr=True`: to use of $T_{\rm eff,corr}$ instead of $T_{\rm eff}$;
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`;
  - `binz=n`: to limit the number of colours when using `zcol` (default: `256`);
  - `plotif='condition'`: to limit the plot to a given condition (cf. page 9);
  - `ticks=[tick list]`: to define the location of the colour bar ticks.

- **Colour-magnitude diagram**
  Command: `CMD()`
  Entered without argument, plots the M_V versus B-V diagram. To plot other colours, enter a 3-character string like `'BUB'` (for M_B versus U-B) or `'VVI'` (for M_V versus V-I).
  The following optional arguments are available:
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`;
  - `binz=n`: to limit the number of colours when using `zcol` (default: `256`);
  - `plotif='condition'`: to limit the plot to a given condition (cf. page 9);
  - `ticks=[tick list]`: to define the location of the colour bar ticks.

- **Kippenhahn diagram**
  Command: `Kippen(num_⋆)`
  The following optional arguments are available:
  - `hatch='/','\','-','|','+','x','O','o','.','*'`: to hatch the convective zones (repeat the symbol to get a denser hatching: `'++','///'`);
  - `noshade=True` to remove the grey shading.

- **log T_c − log ρ_c diagram**
  Command: `rhoT()`
  The following optional arguments are available:
  - `deg=True/False`: to draw the degeneracy line (default: `True`);
  - `PISN=True`: to shade the $\Gamma_1 < 4/3$ zone;
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`;
  - `binz==n`: to limit the number of colours when using `zcol` (default: `256`);
  - `plotif='condition'`: to limit the plot to a given condition (cf. page 9);
  - `ticks=[tick list]`: to define the location of the colour bar ticks.

- **log g − log $T_{\rm eff}$ diagram**
  Command: `gTeff()`
  The following optional arguments are available:
  - `corr=True`: to use $T_{\rm eff,corr}$ instead of $T_{\rm eff}$;
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`;
  - `binz=n`: to limit the number of colours when using `zcol` (default: `256`);
  - `plotif='condition'`: to limit the plot to a given condition (cf. page 9);
  - `ticks=[tick list]`: to define the location of the colour bar ticks.

- **Y − log $T_{\rm eff}$ diagram**
  Command: `YTeff()`
  The following optional arguments are available:
  - `corr=True`: to use $T_{\rm eff,corr}$ instead of $T_{\rm eff}$;
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`;
  - `binz=n`: to limit the number of colours when using `zcol` (default: `256`);
  - `plotif='condition'`: to limit the plot to a given condition (cf. page 9);
  - `ticks=[tick list]`: to define the location of the colour bar ticks.

- **Abundances evolution**
  Command: `Abund('c')` (centre)
  Command: `Abund('s')` (surface)

- **N/C vs N/O diagram**
  Command: `NCNO()`
  The following optional parameter is available:
  – `plotif='condition'`: to limit the plot to a given condition on a variable

- **Set of plots summarising a model**
  Command: `Summary_plots(num_⋆)`
  This command automatically generates the following plots:
  – window 1: HRD, Abund(c), Kippenhahn, L and $T_{\rm eff}$ vs time
  – window 2 : $V_{\rm surf}$, $\Omega/\Omega_{\rm crit}$, $\dot{M}$, $\Omega_{\rm c}/\Omega_{\rm s}$
  – window 3 : Abund(s), N/O, N/H, N/C
  – window 4 : $T_{\rm c} - \rho_{\rm c}$, $T_{\rm c}$ and $\rho_{\rm c}$ vs time
  With the optional argument `'legend'`, a legend is added to the plots.

4.3.2 Graphs in STRUC mode

- **log T − log ρ diagram**
  Command: `rhoT()`
  The following optional parameter is available:
  – `plotif='condition'`: to limit the plot to a given condition on a variable

- **Energy generation zones**
  Command: `eps(mode)`
  The argument `mode` defines the x-axis variable:
  – `1`: $M_r/M_{\rm tot}$ (default);
  – `2`: $r\,[R_\odot]$;
  – `3`: free x variable defined by `defX()`.
  The optional parameter `conv=False` avoids the shading of convective zones (default: `True` if only one structure selected).

- **Gradients ($\nabla_{\rm ad}$ - $\nabla_{\rm rad}$ - ∇_μ)**
  Command: `Nablas(num_⋆)`

- **Abundances profile**
  Command: `Abund('p')`

- **Diffusion coefficients ($D_{\rm conv}$ - $D_{\rm shear}$ - $D_{\rm h}$ - $D_{\rm eff}$ - $K_{\rm ther}$)**
  Command: `Coeff()`

- **Addition of the convective zones on the plot**
  Command: `convZones(num_⋆)`
  The optional parameter `colour='colour'` changes the default colour (grey).

- **Set of plots summarising a structure**
  Command: `Summary_plots(num_⋆)`
  This command automatically generates the following plots:
  – window 1 : Abund(p)
  – window 2 : T, r, P, ρ
  – window 3 : L and ϵ, T vs ρ, $\epsilon_{\rm reac}$ and $\epsilon_{\rm grav}$, nablas and kappa
  – window 4 : D, N^2, $c_{\rm sound}$ and $V_{\rm MLT}$, magn. field variables
  – window 5 : Ω and $\Omega/\Omega_{\rm crit}$, $V_{\rm eq}$, j, U and V
  With the optional argument `'legend'`, a legend is added to the plots.

4.3.3 Graphs in CLUSTER mode

- **HR diagram**
  Command: `HRD_plot()`
  The following optional parameters are available:
  – `spectro=True`: for the sHRD ($T_{\rm eff}^4/g$ vs $T_{\rm eff}$)
  – `corr=True`: to use of $T_{\rm eff,corr}$ instead of $T_{\rm eff}$
  – `dark=True`: to use of L and $T_{\rm eff}$ corrected for the gravity(+limb) darkening
  – `zcol='z_var'`: to colour-code the line with the value of `z_var`
  – `binz=n`: to limit the number of colours when using `zcol` (default: `256`)
  – `plotif='condition'`: to limit the plot to a given condition on a variable

- **Colour-magnitude diagram**
  Command: `CMD()`
  Entered without argument, plots the M_V versus B-V diagram. To plot other colours, enter a 3-character string like `'BUB'` (for M_B versus U-B) or `'VVI'` (for M_V versus V-I).
  The following optional arguments are available:
  - `noised='xy'`: to plot the noised variables (created with `add_noise()`)
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`
  - `binz=n`: to limit the number of colours when using `zcol` (default: `256`)
  - `plotif='condition'`: to limit the plot to a given condition on a variable

- **log g − log $T_{\rm eff}$ diagram**
  Command: `gTeff()`
  The following optional arguments are available:
  - `dark=True`: to use $T_{\rm eff,lgd}$ instead of $T_{\rm eff}$
  - `corr=True`: to use $T_{\rm eff,corr}$ instead of $T_{\rm eff}$
  - `surf=True`: to use $g_{\rm surf}$ instead of $g_{\rm pol}$
  - `mean=True`: to use $g_{\rm mean}$ instead of $g_{\rm pol}$
  - `noised='xy'`: to plot the noised variables (created with `add_noise()`)
  - `zcol='z_var'`: to colour-code the line with the value of `z_var`
  - `binz=n`: to limit the number of colours when using `zcol` (default: `256`)
  - `plotif='condition'`: to limit the plot to a given condition on a variable

- **Abundances evolution**
  Command: `Abund('s')`

- **N/C vs N/O diagram**
  Command: `NCNO()`
  The following optional parameter is available:
  - `plotif='condition'`: to limit the plot to a given condition on a variable

- **Histogram of variable `var`**
  Command: `Histo(var,bin_number)`
  The optional parameter `cum=True` makes a cumulative histogram.

4.4 Saving the figures

The python window allows to save the current figure in png format, with a clic on  .

For a vectorised format, the command `MyFig('#file_name')` can be used. The default format is pdf, but it might be customised with the optional argument `format='#format'`.

5 Adaptations to specific needs

The following set of commands allows you to modified many settings for the rendering of your plots.

NB: At any time, you can go back to the default ones (as at launch) with the command `default_settings()`.

5.1 Limits and axes

- **Change of the limits**
  Command: `Limits(xmin=,xmax=,ymin=,ymax=)`
  Back to normal with `noLimits('x','y' or 'xy')`. `noLimits()` without argument corresponds to `'xy'`.

  Command: `CBLimits(min=,max=)`
  This command sets the limits for `var_z` in `Plot_colour()`. Back to normal with `noCBLimits()`.

- **Displaying the limits of the current axes**
  Command: `get_limits()`

- **Keeping the actual limits**
  Command: `keep_limits()`
  Back to normal with `keep_limits(False)`.

- **Inversion of an axis**
  Command: `axis_inv('x','y' or 'xy')`
  NB: back to normal with `no_axis_inv('x','y' or 'xy')`.
  `no_axis_inv()` without argument corresponds to `'xy'`.

- **Axis in logarithmic scale**
  Command: `logScale('x','y' or 'xy')`
  The following optional parameters are accepted:
  – `grid=False/True` to display a grid on the plot for the axis selected (default: `True`);
  – `ls` to set the line style (default: `'-'`);
  – `lw` to set the line width (default: `0.2`);
  – `lc` to set the line colour (default: `'0.80'`).
  NB: back to normal with `no_logScale()`

- **Modification of an axis label**
  Command: `change_label('axis','label')`

- **Change the number of minor ticks**
  Command: `set_tickNumber(N)`
  NB: back to default behaviour (automatic) with `set_tickNumber(0)`

- **Displaying a grid on the plot**
  Command: `display_grid()`
  The following optional parameter are accepted:
  – `ax` to display the grid only to one particular axis (default: `'both'`);
  – `ls` to set the line style (default: `'-'`);
  – `lw` to set the line width (default: `0.2`);
  – `lc` to set the line colour (default: `'0.80'`).
  NB: back to normal empty plot area with `display_grid(False)`

5.2 Points/curves

- **Switch from curves to points**
  Command: `Points(True)`
  Back to curves with `Points(False)`

- **Choice of line style**
  Command: `set_lineStyle('style')`
  where `style` can be:
  – `cycle_colour` so that the line style changes only after a whole colour sequence (default);
  – `cycle_all` so that each curve is drawn with a different style;
  – any of the following: `-`, `--`, `:`, `-.` for solid, dashed, dotted, and dash-dotted lines, respectively

- **Choice of point style**
  Command: `set_pointStyle('style')`
  where `'style'` can be:
  – `cycle_colour` so that the points style changes only after a whole colour sequence (default);
  – `cycle_all` so that each model points are drawn with a different style;
  – any of the following: `'o'` = ○, `'s'` = □, `'p'` = ⬠, `'d'` = ◊, `'*'` = ☆, `'v'` = ▽, `'^'` = △, `'>'` = ▷, `'<'` = ◁;
  – a tuple `(numsides,style,angle)`. `style` can be `0` (polygon), `1` (star-like), or `2` (squeletal). `angle` can be omitted.

- **Choice of point size**
  Command: `set_pointSize(value)`
  This command accepts either the direct value (`set_pointSize(12)`), or a factor `f=x` to multiply the actual size by `x`:
  `set_pointSize(f=5.)`
  Called with the argument `'default'` (`set_pointSize('default')`), it recovers the default point size (24).

- **Choice of min and max point sizes in `Plot_colour()` and `plotExternal()`**
  Command: `set_PSminmax(min,max)`
  To recover the default size (`5,200`), call `set_PSminmax('default')`.

- **Drawing empty symbols**
  Command: `emptyPoints(True)`
  Back to filled symbols (default) with `emptyPoints(False)`

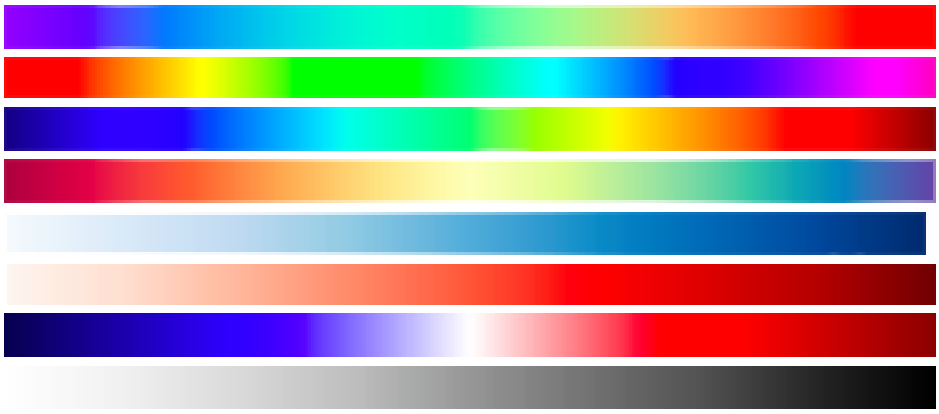
5.3 Colours

- **Choice of a given colour for plotting**
  Command: `set_colourFlag('value')`
  where `value` can be any of the following:
  – a string stating:
  · the shortcut for the main colour names (**b**, **g**, **r**, **c**, **m**, **y**, **k**, **w**), or the full colour name as in html palette;
  · `cycle`: to recover a colour sequence behaviour;
  – a float to set a shade of grey;
  – a triplet of floats to code direct RGB values (in a `0` to `1` scale).

- **Start the colour sequence `n` steps further**
  Command: `Plot('y',cshift=n)`

- **Choice of a colour sequence when plotting more than one model**
  Command: `set_colourSequence('value')`
  Sets the colour sequence applied to the tracks when more than one is plotted.
  `value` can be one of the following:
  – `'c'`: for `'contrast'`, the default sequence (black,red,green,blue,cyan,magenta,orange,olive,pink,brown,grey);
  – `'i'`: for `'iris'` (black,blue,cyan,green,yellow,orange,red,magenta,purple,grey);
  – `'p'`: to create a personalised sequence. Following the instructions, you'll have to enter a name for the sequence and then the new list in the format `[[colours],[names]]`;
  – any existing sequence name.

- **Choice of a colour map for the 3rd variable**
  Command: `set_colourMap('name')`
  where `name` can be any known cmap from matplotlib. We particularly recommend the following:
  - `rainbow`
  - `gist_rainbow`
  - `jet`
  - `spectral`
  - `Blues`
  - `Reds`
  - `seismic`
  - `Greys`



5.4 Windows

- **Drawing of 2 or 4 plots on the same window**
  Command: `multiPlot(2 or 4)`
  NB: back to normal with `multiPlot(1)`

5.5 Navigation within the graphs

The python window allows to easily zoom on a part of the graph after a clic on  . It is also possible to navigate within the graph after clicking on  . The original view is recovered with the  button.

6 Extra informations on (or from) the graphs

- **Activating the cursor**
  Command: `cursor()`
  The coordinates of clicks are written on the terminal. To exit the cursor mode, just hit `Enter` on the terminal window.

6.1 Marks

- **Addition of marks at (a) given age(s)**
  Command: `dot_age(age)`
  where `age` can be a single value or a list, and expressed in log or in natural time
  (if all values are <12, the routine considers it is log).
  The following optional parameters are accepted:
  – `num_star=num_`$\star$ to restrict to the given stars;
  – `marker='marker'`: can be a single or a list of markers (default: `'o'`);
  – `colour='colour'`: can be a single or a list of colours (default: `'k'`);
  – `age_print=True/False`: to print the age on the plot near the dot (default: `False`);
  – `precision=n`: number of decimals to be displayed (default: `5`);
  – `legend_star=num_`$\star$: restricts the printing of the age(s) to star(s) num_$\star$.

- **Addition of marks at regular time steps**
  Command: `timesteps(True)`
  This command must precede the plotting command.
  The time step can be determined by the command `set_deltat(value)`. By default, this command acts on all stars selected for plotting. It is possible to limit it to some specific stars by entering a number, or a list of numbers after the value.
  Example: `set_deltat(5.e5,[2,3])`
  The type of marker can be chosen by `timestep_marker('marker')`.

- **Marking the beginning and end of a given burning phase**
  Command: `mark_phase('fuel')`
  with `fuel` being one of the following: `H`, `He`, `C`, `Ne`, `O`, or `Si`.
  The following optional parameters are accepted:
  – `marker=[' ',' ']`: changes the marker style (default `['o','x']`);
  – `colour`: changes the marker colour (default: `'k'`).

- **Drawing iso-radius lines in an HRD**
  Command: `isoRadius()`
  The following optional parameters are accepted:
  – `colour='colour'`: sets the colour of the line (default: `'0.80'`);
  – `line='style'`: sets the line style. It can be any of the matplotlib styles (see Sec. 5.2, default: `'-'`);
  – `fontsize=size`: sets the fontsize for the radius labels.

6.2 Lines, dots, and shades

- **Drawing a vertical or horizontal line at a given value**
  Command: `xline(value)` (vertical)
  Command: `yline(value)` (horizontal)
  The following optional parameters are accepted:
  – `colour='colour'`: sets the colour of the line (default: `'0.80'`);
  – `line='style'`: sets the line style. It can be any of the matplotlib styles (see Sec. 5.2, default '-');
  – `lw=float`: sets the line width.

- **Drawing a line from(x_1,y_1) to (x_2,y_2)**
  Command: `line(`x_1`,`x_2`,`y_1`,`y_2`)`
  The following optional parameters are accepted:
  – `colour='colour'`: sets the colour of the line (default: `'0.80'`);
  – `line='style'`: sets the line style. It can be any of the matplotlib styles (see Sec. 5.2, default: `'-'`).

- **Drawing a line with a given slope**
  Command: `slope(value)`
  The following optional parameters are accepted:
  - `centre=[x,y]` allows to centre the line at point (x,y), default: `(0,0)`;
  - `colour='colour'`: sets the colour of the line (default: `'0.80'`);
  - `line='style'`: sets the line style. It can be any of the matplotlib styles (see Sec. 5.2, default: `'-'`).

- **Drawing a dot at the coordinate (x,y)**
  Command: `dotxy(x,y)`
  The following optional parameters are accepted:
  - `'style'` allows to enter directly the style of the point: `dotxy(x,y,'ro')`;
    Otherwise, the style is defined by the values entered with `set_pointStyle()` and `set_colourFlag()`.
  - `err=[xerr,yerr]` allows to draw error bars;
    For asymmetric errorbars, write `err=[[xerr_left,xerr_right],[yerr_down,yerr_up]]`.
  - `label='string'` writes `string` besides the point;
  - `ha='pos'` allows to put this string left or right of the point;
  - `fontsize=int` sets the size of the font.

- **Shading a zone defined by vectors x and y**
  Command: `shade(x,y)`
  The following optional parameters are accepted:
  - `colour='colour'`: sets the colour of the shaded zone (default: `'0.80'`);
  - `alpha=value`: sets the transparency (20% by default);
  - `hatch='/','\','-','|','+','x','O','o','.','*'` (repeat symbol to get a denser hatching: `'++','///'`).

- **Shading a vertical or horizontal region around $x1$ and $x2$, or $y1$ and $y2$**
  Command: `shade_x(x1,x2)` ((vertical))
  Command: `shade_y(y1,y2)` ((horizontal))
  The following optional parameters are accepted:
  - `colour='colour'`: sets the colour of the shaded zone (default: `'0.80'`);
  - `alpha=value`: sets the transparency (default: `0.2`);
  - `hatch='/','\','-','|','+','x','O','o','.','*'` (repeat symbol to get a denser hatching: `'++','///'`).

6.3 External data

- **Addition of external data on a graph**
  Command: `plotExternal('#File_name',x_column,y_column)`
  The following optional arguments are accepted:
  - `skip=n`: to skip `n` lines at the beginning the file `#File_name`.
    Note that header lines starting by `#` are automatically skipped;
  - `last=n`: to stop at line `n`;
  - `style='style'`: to define the points/lines style (example: `'ro','b+','g.','m-'`).
    By default, the general style (defined by `set_colourFlag()` and either `set_pointStyle()` or `set_lineStyle()`) will be taken;
  - `colz=n`: to define the column of the variable that will colour-code the points (works only with `Points(True)`);
  - `cols=n`: to define the column of the variable that will give the size of the points (works only with `Points(True)`);
  - `log='x','y','z','s'` or any combination `'xy','xz',...`: to take the log of the variable(s);
  - `zlabel=str`: to define the label of the z-axis;
  - `clim='new'/'old'`: to set new limits to the colour bar or to retrieve the existing ones;
  - `new`: to create a new figure rather than adding the data to the current figure.

6.4 Getting informations

- **Measurement of the distance between two points**
  Command: `dist()`
  The clicks are validated by hitting `Enter` on the terminal window.
  The coordinates are written on the terminal, as well as Δx, Δy and the distance.

- **Getting the closest line at a point on the curve**
  Command: `closest_line()`
  Prints the line number on the terminal.
  The optional parameter `print_line=True` makes the full line to be printed on the terminal.

- **Finding a polynomial fit of degree N to the curve**
  Command: `fit_poly(N)`
  The following optional arguments are accepted:
  – `y=var`: to change the target curve (default: last curve drawn);
  – `colour=colour`: to change the colour of the fit curve (default: `'0.80'`).

6.5 Text and legends

- **Setting the fontsize**
  Command: `set_fontSize()`
  This command accepts either the size directly (`set_fontSize(12)`),
  or a factor to apply to the actual fontsize (`set_fontSize(f=0.5)`).
  Called with `'default'` as argument (`set_fontSize('default')`), it recovers the default fontsize (`24`).

- **Addition of the legend**
  Command: `put_legend()`
  The following optional arguments are accepted:
  – `pos=int`: sets the position of the legend box on the plot. The value can be entered directly as first argument.
  The integer coding the positions are:
  · 1: top right (default);
  · 2: top left;
  · 3: bottom left;
  · 4: bottom right;
  · 5: middle right;
  · 6: middle left;
  · 7: middle right again (don't ask me why);
  · 8: bottom centre;
  · 9: top centre;
  · 10: middle centre.
  – `label=['line 1','line 2']`: to define the label (default: variable *y*);
  – `fontsize=int`: sets the fontsize (default: axis fontsize / 1.5).

- **Addition of text in the graph**
  Command: `add_label(x,y,'string')`
  The following optional parameters are accepted:
  – `colour='colour'`: sets the colour of the line (default: `'k'`);
  – `fontsize=value` modifies the fontsize (default: `24`);
  – `ha='pos'`: sets the horizontal position relative to (*x*,*y*). `'pos'` can be any of the following:
  `'left'`, `'center'`, or `'right'` (default: `'left'`).
  – `va='pos'`: sets the vertical position relative to (*x*,*y*). `'pos'` can be any of the following:
  `'baseline'`, `'bottom'`,`'center'`, or `'top'` (default: `'bottom'`).

- **Addition of a title at the top of the window**
  Command: `top_label('string')`
  The optional argument `fontsize=value` modifies the fontsize only for this command.

7 List of the variables

7.1 In EVOL mode

- MODEL:

line :	model num	ageadv :	log(time before collapse [yr])
M :	M [$M_\odot$]	t_rel :	fraction of burning phases (0-1: H-b, 1-2: He-b, 2-3: adv. phases)
t :	t [yr]		
t6 :	t [Myr]	tauKH :	τ_{KH} [yr]
t9 :	t [Gyr]	rhom :	$\rho_{\rm m}$ [g cm^3]
t_tauH :	$t/\tau_{\rm H}$		

Besides, the following global variables are known:

FileName :	`#loaded_file`	tau :	lifetimes [yr] ([$\tau_{\rm H}$,$\tau_{\rm He}$,$\tau_{\rm C}$,$\tau_{\rm Ne}$,$\tau_{\rm O}$])
Mini :	$M_{\rm ini}$ [$M_\odot$]	format :	`#format`

- CENTRE:

Mcc :	$M_{\rm cc}$ [$M_\odot$]	rhoc :	log($\rho_{\rm c}$ [g cm^{-3}])
Mccrel :	$M_{\rm cc}/M_{\rm tot}$	Tc :	log($T_{\rm c}$ [K])

- SURFACE:

Teff :	log($T_{\rm eff}$ [K])	R :	R [$R_\odot$]
Teffcorr :	log($T_{\rm eff}$ [K]) corrected for the wind thickness (WR)	Rpol :	$R_{\rm pol}$ [$R_\odot$]
		gsurf :	log($g_{\rm surf}$ [cm s^{-2}])
L :	log($L/L_\odot$)	gpol :	log($g_{\rm pol}$ [cm s^{-2}])
Mbol :	$M_{\rm bol}$	fwg:	log($g/(T_{\rm eff}/10'000\,{\rm K})^4$)
sL:	log($\mathscr{L}/\mathscr{L}_\odot$)	ZCext :	$M_{\rm ZC,ext}$
GammaEdd :	$\Gamma_{\rm Edd}$		

- ROTATION:

Omega_surf :	$\Omega_{\rm surf}$ [s^{-1}]	oblat :	$R_{\rm pol}/R_{\rm eq}$
Omega_cen :	$\Omega_{\rm cen}$ [s^{-1}]	GammaOmega :	$\Omega/min(\Omega_{\rm crit,1},\Omega_{\rm crit,2})$
OOc :	$\Omega/\Omega_{\rm crit}$	rot_corr :	F_Ω
VVc:	$V/V_{\rm crit}$	Ltot :	$\mathcal{L}_{\rm tot}$ [10^{53} g cm^2 s^{-1}]
Vsurf :	$V_{\rm surf}$ [km s^{-1}]	Ltotint :	$\mathcal{L}_{\rm tot,int}$ [10^{53} g cm^2 s^{-1}]
Vcrit1 :	$V_{\rm crit,1}$ [km s^{-1}]	jspe5 :	$j_{5M_\odot}$ [10^{16} cm^2 s^{-1}]
Vcrit2 :	$V_{\rm crit,2}$ [km s^{-1}]	jspe3 :	$j_{3M_\odot}$ [10^{16} cm^2 s^{-1}]
period:	P [d]	mominert :	I [10^{57} g cm^2]

- WINDS:

Mdot :	log($\dot{M}$ [$M_\odot$ yr^{-1}])	Vesc :	$V_{\rm esc}$ [km s^{-1}]
Mdot_mech :	(log($\dot{M}$)$_{\rm mech}$ [$M_\odot$ yr^{-1}])	dlelex :	$\Delta\,\mathcal{L}_{\rm rad+aniso+mech}$ [10^{53} g cm^2 s^{-1}]
dMmech :	d$M_{\rm mech}$ [$M_\odot$]	Bmin :	$B_{\rm min}$ [G] (minimal magn. field for a wind-surface coupling)
Pwinds :	$P_{\rm winds}$ [erg s^{-1}]		
Vinf :	V_∞ [km s^{-1}]		

- ENERGETICS:

Epot :	$E_{\rm pot}$ [E_{51}]	Erot :	$E_{\rm rot}$ [E_{51}]
Egaz :	$E_{\rm th,gaz}$ [E_{51}]	phase : evolutionary phase	
Erad :	$E_{\rm rad}$ [E_{51}]		

- ABUNDANCES:

H1s,H1c :	^{1}H (surf., centr.) [mass frac.]	B8c :	^{8}B (centr.) [mass frac.]
He3s,He3c :	^{3}He (surf., centr.) [mass frac.]	C12s,C12c :	^{12}C (surf., centr.) [mass frac.]
He4s,He4c :	^{4}He (surf., centr.) [mass frac.]	C13s,C13c :	^{13}C (surf., centr.) [mass frac.]
Be7c :	^{7}Be (centr.) [mass frac.]	N14s,N14c :	^{14}N (surf., centr.) [mass frac.]

O16s,O16c :	^{16}O (surf., centr.) [mass frac.]
O17s,O17c :	^{17}O (surf., centr.) [mass frac.]
O18s,O18c :	^{18}O (surf., centr.) [mass frac.]
Ne20s,Ne20c :	^{20}Ne (surf., centr.) [mass frac.]
Ne22s,Ne22c :	^{22}Ne (surf., centr.) [mass frac.]
Al26s,Al26c :	^{26}Al (surf., centr.) [mass frac.]
Zsurf :	$Z_{\rm surf}$ [mass frac.]
C12C13 :	$\log(^{12}\mathrm{C}/^{13}\mathrm{C}$ [numb.])
C12C13rel :	$\log(^{12}\mathrm{C}/^{13}\mathrm{C}) - \log(^{12}\mathrm{C}/^{13}\mathrm{C})_{\rm ini}$
NH :	log(N/H [numb.] + 12)
NHrel :	$\log(\mathrm{N/H}) - \log(\mathrm{N/H})_{\rm ini}$
NC :	log(N/C [numb.])
NCrel :	$\log(\mathrm{N/C}) - \log(\mathrm{N/C})_{\rm ini}$
NO :	log(N/O [numb.])
NOrel :	$\log(\mathrm{N/O}) - \log(\mathrm{N/O})_{\rm ini}$

7.2 In STRUC mode

• STRUCTURE:

shell :	shell number
Mfrac :	$M_r/M_{\rm tot}$
Mr :	M_r [$M_\odot$]
r_cm :	r [cm]
r :	r [$R_\odot$]
rprev :	$r_{\rm prev}$ [$R_\odot$]
g :	g_r [cm s^{-2}]
N2 :	N^2 [s^{-1}]
Nmu2 :	N^2_μ [s^{-1}]
NT2 :	N^2_T [s^{-1}]

Besides, the following global variables are known:

M_tot :	$M_{\rm tot}$ [$M_\odot$]
FileName :	`#loaded_file`
Model :	model number
nshell :	total shells number
age :	age [yr]
timestep :	δt [s]

• ENERGY:

L :	$L_r/L_{\rm tot}$
epsH :	ϵ(H) [erg g^{-1}s^{-1}]
epsHe :	ϵ(He) [erg g^{-1}s^{-1}]
epsC :	ϵ(C) [erg g^{-1}s^{-1}]
eps3a :	$\epsilon(3\alpha)$ [erg g^{-1}s^{-1}]
epsCagO :	$\epsilon(^{12}C(\alpha,\gamma)^{16}O)$ [erg g^{-1}s^{-1}]
epsOagNe :	$\epsilon(^{16}O(\alpha,\gamma)^{20}Ne)$ [erg g^{-1}s^{-1}]
epsnu :	$-\epsilon_\nu$ [erg g^{-1}s^{-1}]
eps_reac :	$\epsilon_{\rm nucl} + \epsilon_\nu$ [erg g^{-1}s^{-1}]
epsgrav :	$\epsilon_{\rm grav}$ [erg g^{-1}s^{-1}]
dEdP :	$\mathrm{d}\ln E/\mathrm{d}\ln P$
dEdT :	$\mathrm{d}\ln E/\mathrm{d}\ln T$

• EOS:

rho :	ρ [g cm^3]
mu :	μ (mean molecular weight)
muprev :	μ of previous iteration
mufit :	μ (smoothed)
mue :	μ_e
Nabmu :	∇_μ
drhodP :	$\mathrm{d}\ln\rho/\mathrm{d}\ln P$
delta :	$\delta = -\mathrm{d}\ln\rho/\mathrm{d}\ln T$
cs :	$c_{\rm sound}$ [cm s^{-1}]
psi :	ψ (indicator of degeneracy)

• THERMO:

P :	P [g cm^{-1} s^{-2}]
beta :	$\beta = P_{\rm gas}/P_{\rm tot}$
Hp :	H_P [cm]
T :	T [K]
Kther :	$K_{\rm ther}$ [cm^2 s^{-1}]
Nabla :	∇
Nabad :	$\nabla_{\rm ad}$
Nabrad :	$\nabla_{\rm rad}$
Nabla_int :	$\nabla_{\rm int}$
kappa :	κ [cm^2 g^{-1}]
dkdP :	$\mathrm{d}\ln\kappa/\mathrm{d}\ln P$
dkdT :	$\mathrm{d}\ln\kappa/\mathrm{d}\ln T$
Cp :	C_P [ergs g^{-1} K^{-1}]
V_MLT :	$V_{\rm MLT}$ [cm s^{-1}]

• ROTATION:

Omega :	Ω [s^{-1}]
Omegaprev :	Ω of previous time step
Omegacons :	Ω (when only local conservation of angular momentum is applied)
Omfit :	Ω (smoothed)
dlodlr :	$\mathrm{d}\ln\Omega/\mathrm{d}\ln r$
obla :	$r_{\rm pol}/r_{\rm eq}$
Vr :	V_r [cm s^{-1}]
Veq :	$V_{\rm eq}$ [km s^{-1}]
Lang :	$\mathcal{L}_r$ [g cm^2 s^{-1}]
jr :	j_r [cm^{2}s^{-1}]
jS :	$j_{\rm Schwarzschild}$ [cm^{2}s^{-1}]
jK :	$j_{\rm Kerr}$ [cm^{2}s^{-1}]
jKmax :	$j^{\rm max}_{\rm Kerr}$ [cm^{2}s^{-1}]
Dh :	$D_{\rm h}$ [cm^2 s^{-1}]
Dshear :	$D_{\rm shear}$ [cm^2 s^{-1}]
Deff :	$D_{\rm eff}$ [cm^2 s^{-1}]
Dcirc :	$D_{\rm circ}$ [cm^2 s^{-1}]

Dconv :	$D_{\rm conv}$ [cm^2 s^{-1}]	Richardson :	Ri = $N^2/({\rm d}V/{\rm d}z)^2$
Ur :	U_r [cm s^{-1}]		

• MAGNETISM:

Br :	B_r [G]	etask :	η/K
Bphi :	B_ϕ [G]	N2mag :	$N^2_{\rm mag}$ [s^{-1}]
qmin :	$q_{\rm min}$	DmagO :	$D_{{\rm mag},\Omega}$ [cm^2 s^{-1}]
alfven :	$\omega_{\rm Alfven}$ [s^{-1}]	DmagX :	$D_{{\rm mag},X}$ [cm^2 s^{-1}]

• ABUNDANCES:

H1 :	^{1}H [mass frac.]	Si28,Si28_alu :	^{28}Si [mass frac.]
He3,He4 :	^{3,4}He [mass frac.]	S32 :	^{32}S [mass frac.]
C12,C13,C14 :	^{12,13,14}C [mass frac.]	Ar36 :	^{36}Ar [mass frac.]
N14,N15 :	^{14,15}N [mass frac.]	Ca40 :	^{40}Ca [mass frac.]
O16,O17,O18 :	^{16,17,18}O [mass frac.]	Ti44 :	^{44}Ti [mass frac.]
F18,F19 :	^{18,19}F [mass frac.]	Cr48 :	^{48}Cr [mass frac.]
Ne20,Ne21,Ne22 :	^{20,21,22}Ne [mass frac.]	Fe52 :	^{52}Fe [mass frac.]
Na23 :	^{23}Na [mass frac.]	Ni56 :	^{56}Ni [mass frac.]
Mg24,Mg25,Mg26 :	^{24,25,26}Mg [mass frac.]	protons :	protons [mass frac.]
Al26,Al27 :	^{26,27}Al [mass frac.]	neutrons :	neutrons [mass frac.]

7.3 In CLUSTER mode

• INITIAL CONDITIONS:

Zini:	$Z_{\rm ini}$	Angle:	i [°]
Mini:	$M_{\rm ini}$ [$M_\odot$]	Bin:	binary
Oini:	$\Omega/\Omega_{\rm crit,ini}$	M1M2:	M_1/M_2

Besides, the following global variables are known:

FileName:	#loaded_file

• GLOBAL PROPERTIES:

M:	M [$M_\odot$]	sL:	$\log(\mathscr{L}/\mathscr{L}_\odot)$
R:	R [$R_\odot$]	Mbol:	$M_{\rm bol}$
Rpol:	$R_{\rm pol}$ [$R_\odot$]	gsurf:	$\log(g_{\rm surf}$ [cm s^{-2}])
L:	$\log(L/L_\odot)$	gpol:	$\log(g_{\rm pol}$ [cm s^{-2}])
L_gd:	$\log(L/L_\odot)_{\rm grav.dark}$	gmean:	$\log(g_{\rm mean}$ [cm s^{-2}])
L_lgd:	$\log(L/L_\odot)_{\rm limb+grav.dark}$	fwg:	$\log(g/(T_{\rm eff}/10'000\,{\rm K})^4)$
Teff:	$\log(T_{\rm eff}$ [K])	rhom:	$\rho_{\rm m}$ [g cm^3]
Teffcorr:	$\log(T_{\rm eff}$ [K])	Gammaedd:	$\Gamma_{\rm Edd}$
Teff_gd:	$\log(T_{\rm eff}$ [K])$_{\rm grav.dark}$	Mdot:	$\log(\dot{M}$ [$M_\odot$ yr^{-1}])
Teff_lgd:	$\log(T_{\rm eff}$ [K])$_{\rm limb+grav.dark}$	dMmech:	d$M_{\rm mech}$ [$M_\odot$]

• ROTATION:

Vcrit1:	$V_{\rm crit,1}$ [km s^{-1}]	OOc:	$\Omega/\Omega_{\rm crit}$
Vcrit2:	$V_{\rm crit,2}$ [km s^{-1}]	Omega_surf:	$\Omega_{\rm surf}$ [s^{-1}]
Vsurf:	$V_{\rm surf}$ [km s^{-1}]	period:	P [d]
Vsini:	$V\,\sin\,i$ [km s^{-1}]	oblat:	$R_{\rm pol}/R_{\rm eq}$

• ABUNDANCES:

H1s:	^{1}H [surf. mass frac.]	O18s:	^{18}O [surf. mass frac.]
He4s:	^{4}He [surf. mass frac.]	Ne20s:	^{20}Ne [surf. mass frac.]
C12s:	^{12}C [surf. mass frac.]	Ne22s:	^{22}Ne [surf. mass frac.]
C13s:	^{13}C [surf. mass frac.]	Al26s:	^{26}Al [surf. mass frac.]
N14s:	^{14}N [surf. mass frac.]	C12C13:	$\log(^{12}{\rm C}/^{13}{\rm C}$ [numb.])
O16s:	^{16}O [surf. mass frac.]	C12C13rel:	$\log(^{12}{\rm C}/^{13}{\rm C})$-$\log(^{12}{\rm C}/^{13}{\rm C})_{\rm ini}$
O17s:	^{17}O [surf. mass frac.]	NH:	log(N/H [numb.]+12)

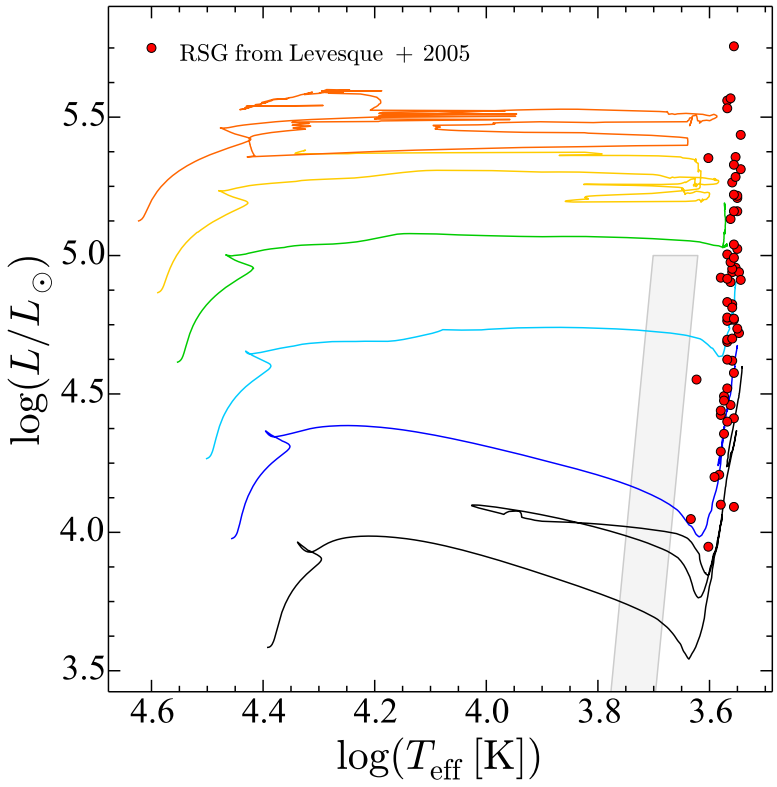
NC:	log(N/C [numb.])	NO:	log(N/O [numb.])
NCrel:	$\log(N/C)-\log(N/C)_{ini}$	NOrel:	$\log(N/O)-\log(N/O)_{ini}$

- COLOURS:

M_V:	M_V	U-B:	U-B
M_V_noise:	M_V noised	V-K:	V-K
M_B:	M_B	V-I:	V-I
B-V:	B-V	V-R:	V-R
B-V_noise:	B-V noised	H-K:	H-K
B2_V1:	$B_2 - V_1$	J-K:	J-K

8 Examples

8.1 HR diagram with observational data points



Loading of the package:

```
from origin_tools.Origin_Tools import *
```

Loading of the evolution files:

```
loadE('/Path/to/file/P009z14S0.dat',1)
loadE('/Path/to/file/P012z14S0.dat',2)
loadE('/Path/to/file/P015z14S0.dat',3)
loadE('/Path/to/file/P020z14S0.dat',4)
loadE('/Path/to/file/P025z14S0.dat',5)
loadE('/Path/to/file/M032Z14V0.dat',6)
```

Choice of the colour sequence `'iris'`:

```
set_colourSequence('i')
```

Change of the upper limit in *y*:

```
Limits(ymax=5.9)
```

Drawing of the HR diagram (predefined command):

```
HRD_plot()
```

Passing to `'points'` mode:

```
Points(True)
```

Drawing of external data, read in a file:

```
plotExternal('RSG_Levesque2005.dat',0,2,log='x',style='ro')
```

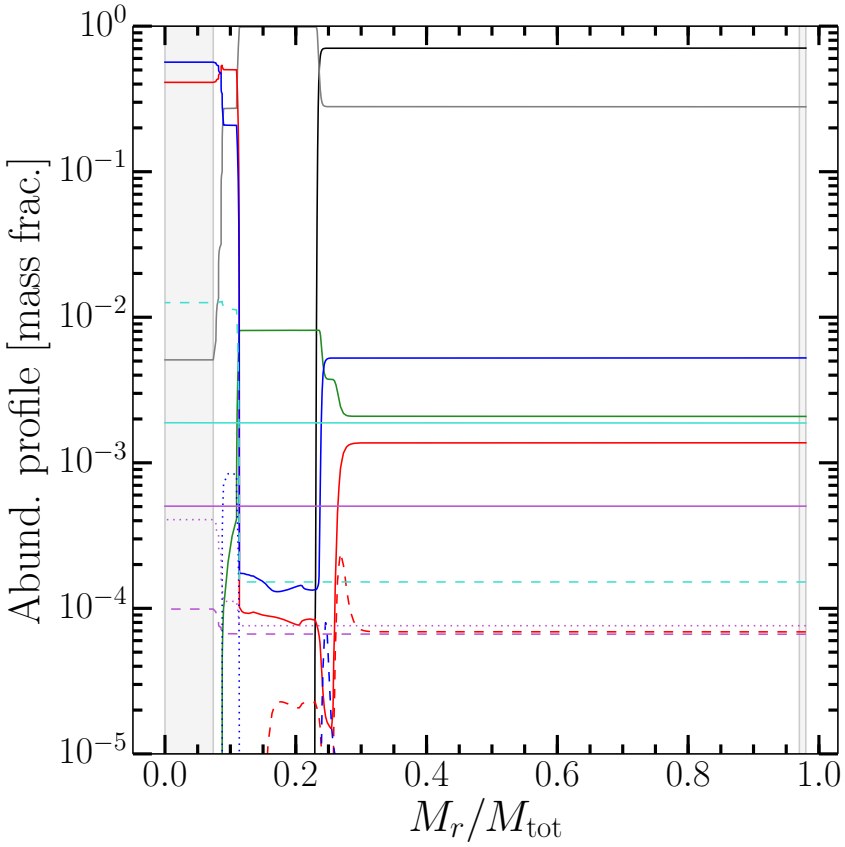
Drawing the legend of the external data:

```
dotxy(4.6,5.75,style='ro',label='$\mathrm{RSG\ from\ Levesque+2005}$',fontsize=16)
```

Saving the figure:

```
MyFig('HRDrainbow_RSG')
```

8.2 Abundances profiles with the convective zones shaded



Loading of a structure file:

```
loadS('/Path/to/file/P007z14S0.v0022721',1)
```

Labels written with LaTeX:

```
iLatex(True)
```

Enhancement of the axes ticks:

```
set_tickSize(length=12,width=2)
```

Modification of the lower limit in *y*:

```
Limits(ymin=1.e-5)
```

y-axis in logarithmic scale:

```
logScale('y')
```

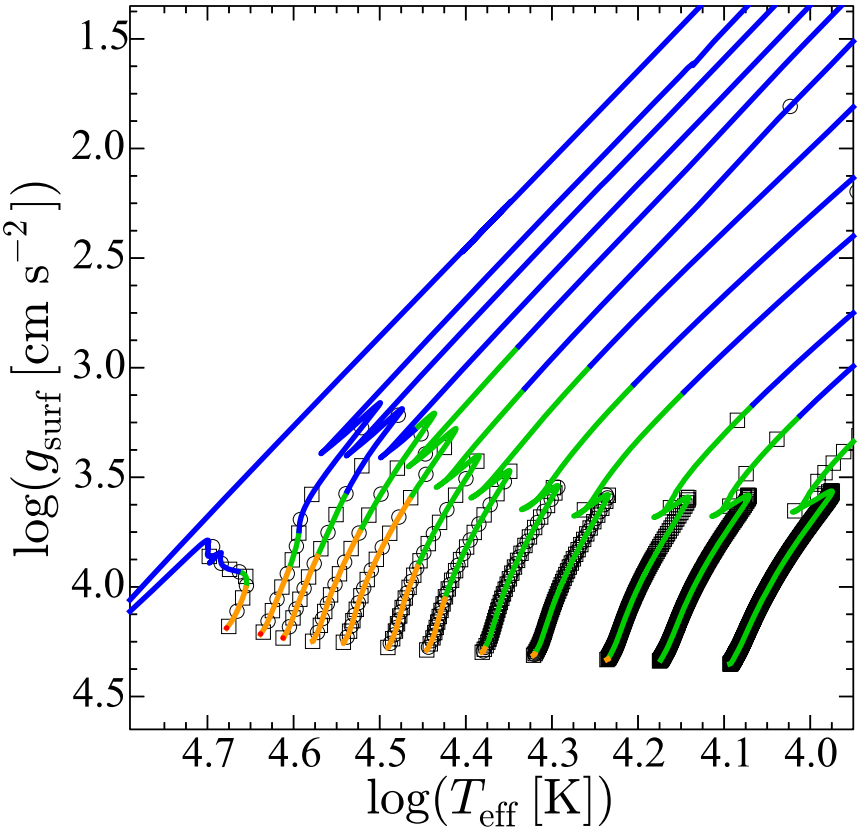
Drawing of the abundances profiles (predefined command):

```
Abund('p')
```

Addition of the convective zones as shaded areas:

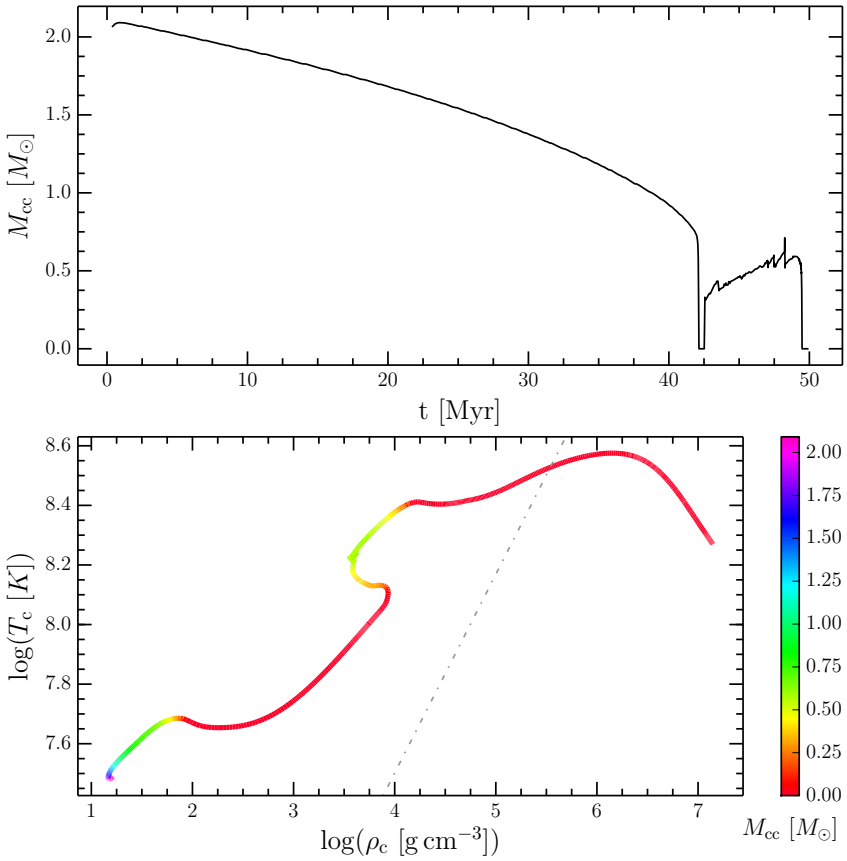
```
convZones(1)
```

8.3 $g_{\rm surf}$ vs $T_{\rm eff}$ with time steps and various colours for velocities zones



Loading of a set of evolution files listed in a text file:
```
loadEFromList('loadModels.txt')
```
Setting the line width to zero (no curves drawn):
```
set_lineStyle('-',width=0)
```
Switching on the markers for time steps:
```
timesteps(True)
```
Setting the timesteps marker to squares:
```
timestep_marker('s')
```
Definition of the x axis:
```
defX('Teffncorr')
```
Changing the natural limits (can be done in one command):
```
Limits(xmin=3.95,xmax=4.79)
Limits(ymin=1.35,ymax=4.65)
```
Inversion the y axis:
```
axis_inv('y')
```
Choice of a unique colour for all curves (black):
```
set_colourFlag('k')
```
Drawing $g_{\rm surf}$:
```
Plot('gsurf')
```
Keeping the window for further plotting:
```
plot2var()
```
Selection of a subset of models:
```
select_model([4,5,6,7,8,9,10,11,12])
```
Changing the time step for those models:
```
set_deltat(5.e5)
```
Setting the timestep markers to circles:
```
timestep_marker('o')
```
Drawing $g_{\rm surf}$:
```
Plot('gsurf')
```
Switching off the markers for time steps:
```
timesteps(False)
```
Setting the line width to 2:
```
set_lineStyle('-',width=2)
```
Selection of all models again:
```
select_model([1,2,3,4,5,6,7,8,9,10,11,12])
```
Changing the colour to blue:
```
set_colourFlag('b')
```
Drawing $g_{\rm surf}$ only if $V_{\rm eq} < 100\,{\rm km\,s^{-1}}$:
```
Plot('gsurf',plotif='Vsurf<100.')
```
Changing the colour to customed green:
```
set_colourFlag((0.,0.8,0.))
```
Drawing $g_{\rm surf}$ only if $100 < V_{\rm eq} \leq 200\,{\rm km\,s^{-1}}$:
```
Plot('gsurf',plotif=['Vsurf>100.',
                     'Vsurf<=200.'])
```
Changing the colour to orange:
```
set_colourFlag((1.,0.6,0.))
```
Drawing $g_{\rm surf}$ only if $200 < V_{\rm eq} \leq 300\,{\rm km\,s^{-1}}$:
```
Plot('gsurf',plotif=['Vsurf>200.',
                     'Vsurf<=300.'])
```
Changing the colour to red:
```
set_colourFlag('r')
```
Drawing $g_{\rm surf}$ only if $V_{\rm eq} > 300\,{\rm km\,s^{-1}}$:
```
Plot('gsurf',plotif='Vsurf>300.')
```

8.4 Double figure with one of them having a 3rd variable in colour code



Loading of an evolution file:

```
loadE('/Path/to/file/P007z14S0.dat',1)
```

Labels written with LaTeX:

```
iLatex(True)
```

Passing to two figures in a window mode:

```
multiPlot(2)
```

Reduction of the labels size:

```
set_fontSize(12)
```

Drawing of the first figure (default *x*-axis):

```
Plot('Mcc')
```

Choice of the *x*-axis for the 2nd figure:

```
defX('rhoc')
```

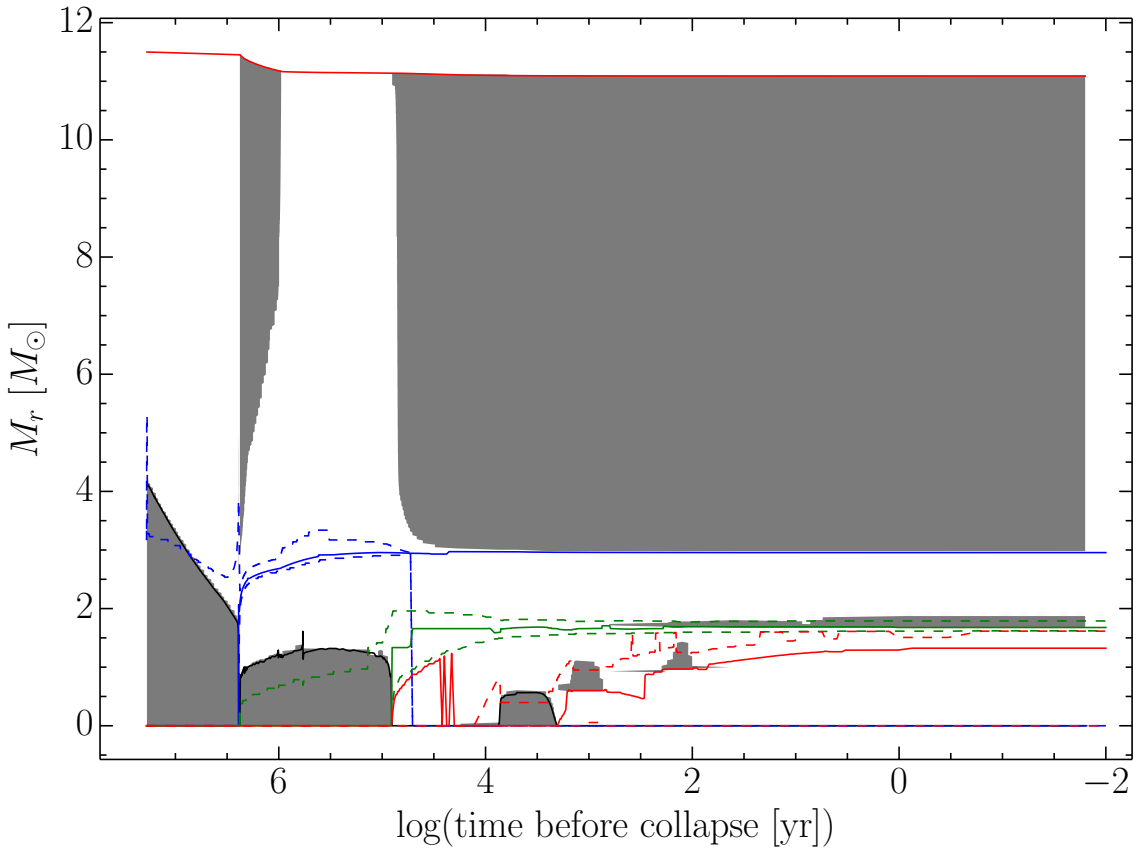
Drawing with a 3rd variable (`Mcc`) in colour code:

```
Plot_colour('Tc','Mcc')
```

Addition of the gas degeneracy line on the plot:

```
degenerate_line()
```

8.5 Kippenhahn diagram with burning zones



Loading of the evolution file:

```
loadE('/Path/to/file/P011p5z14S0.dat',1)
```

Labels written with LaTeX:

```
iLatex(True)
```

Definition of the x variable:

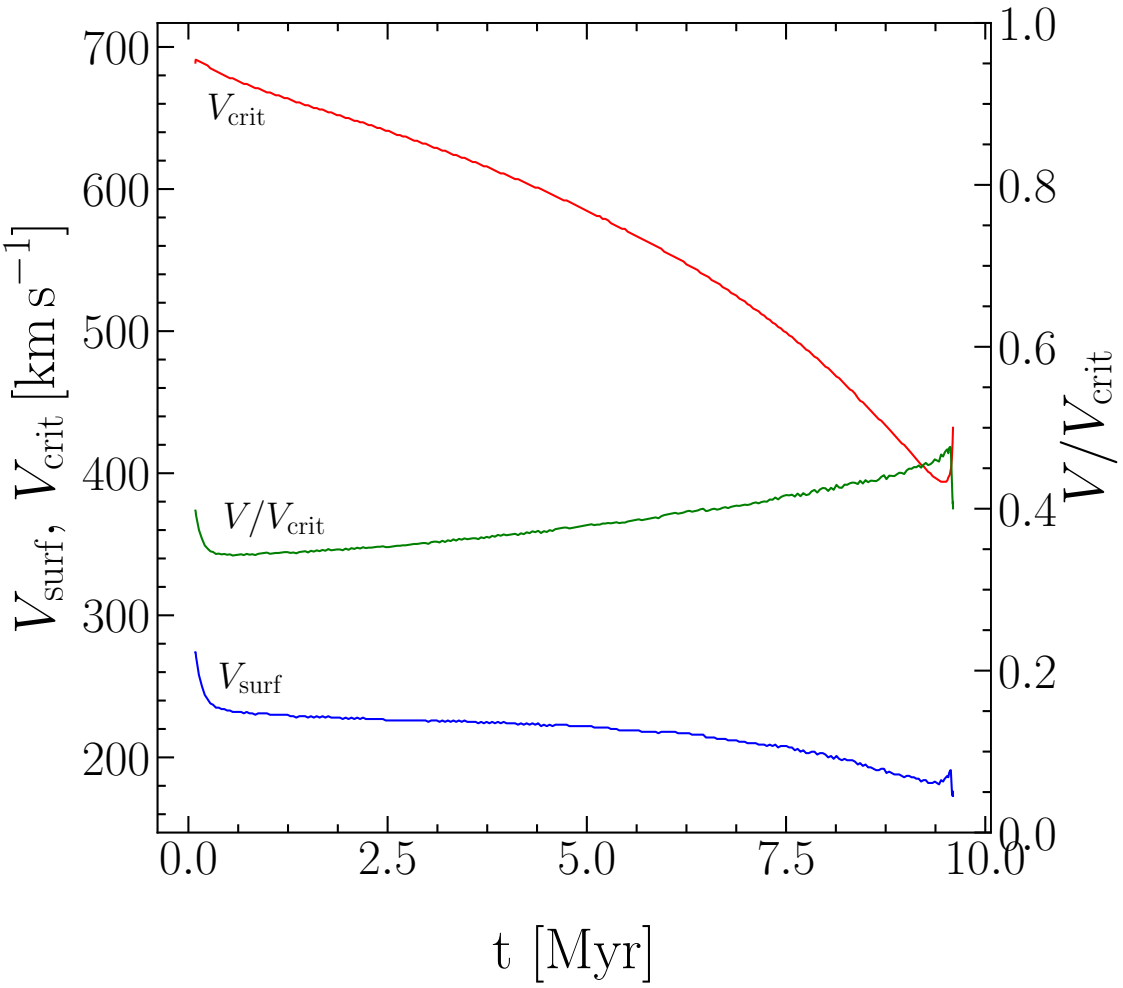
```
defX('ageadv')
```

Drawing of the Kippenhahn diagram (predefined command):

```
Kippen(1,burn=True)
```

NB : the burning zones (plotted with the optional argument `burn=True`) are read in a `#starname.burn` file that must exist somewhere. This `.burn` file is automatically generated when the star's directory has been cleaned with the `cleanfiles` routine with the option `-b`. It can be generated afterwards on the `.v` file of a model with the script `BurningZonesCalc.py`. This script can be requested to sylvia.ekstrom@unige.ch.

8.6 Two different *y* axes



Loading of the evolution file:

```
loadE('/Path/to/file/P018z14S4.dat')
```

Setting the colour to `blue`:

```
set_colourFlag('b')
```

Plotting the first curve, for the MS only:

```
Plot('Vsurf',plotif='H1c>0.')
```

Adding the label:

```
add_label(0.37,240.,'$V_\mathrm{surf}$',fontsize=18)
```

Keeping the figure open for further plotting:

```
keep_plot(True)
```

Setting the colour to `red`:

```
set_colourFlag('r')
```

Plotting the second curve:

```
Plot('Vcrit1',plotif='H1c>0.')
```

Adding the label:

```
add_label(0.23,640.,'$V_\mathrm{crit}$',fontsize=18)
```

Changing the *y* label:

```
change_label('y','$V_\mathrm{surf},\,V_\mathrm{crit}\,[\mathrm{km\,s}^{-1}]$')
```

Preparing for plotting on the second *y* axis:

```
plot2var('double')
```

Setting the limits on this axis:

```
Limits(ymin=0.,ymax=1.)
```

Setting the colour to `green`:

```
set_colourFlag('g')
```

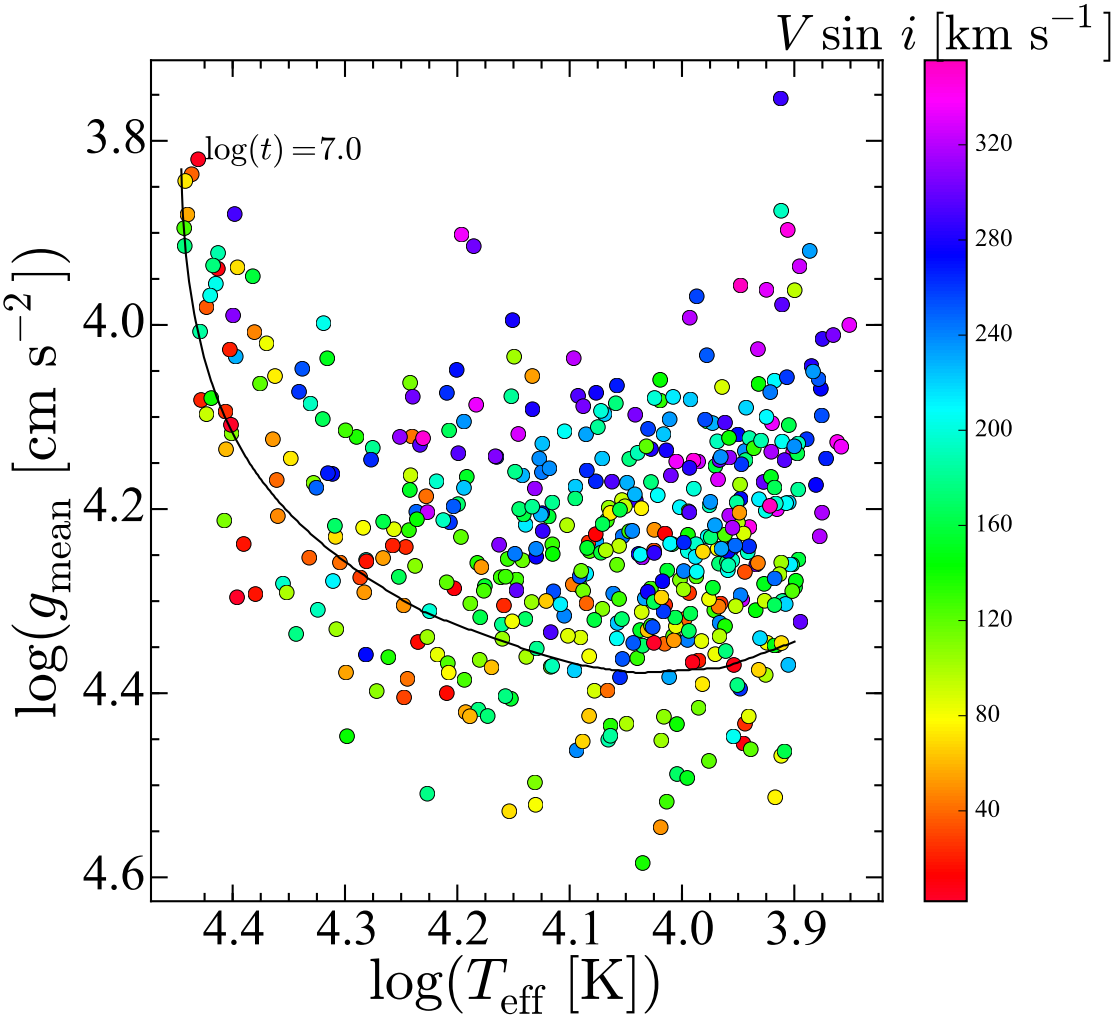
Plotting the last curve:

```
Plot('VVc',plotif='H1c>0.')
```

Adding the label:

```
add_label(0.43,0.36,'$V/V_\mathrm{crit}$',fontsize=18)
```

8.7 Noised g vs $T_{\rm eff}$ with $V \sin i$ in colours



Loading of the cluster and the isochrone:

```
loadC('/Path/to/file/Cluster_z0.014_t07.000.dat')
loadC('/Path/to/file/Isochr_Z0.014_Vini0.50_t07.000.dat',2,format='isochr')
```

Adding a noise on the mean gravity of the cluster:

```
add_noise('gmean',0.1)
```

Plotting the line of the isochrone:

```
select_model(2)
Points(False)
gTeff()
```

Overplotting the cluster points with the $V \sin i$ in colour:

```
keep_plot(True)
select_model(1)
Points(True)
gTeff(dark=True,mean=True,noised='y',zcol='Vsini')
```

adding a label:

```
add_label(4.425,3.82,'$\log(t)=7.0$',fontsize=16)
```