TOP Documentation

Release 1.0-rc1

D. Resse, J. Ballot, B. Putigny

Dec 07, 2016

CONTENTS

CHAPTER ONE

OVERVIEW

TOP consists of a python module, a compiler and several *templates* (see *Stellar Models in TOP*) to read stellar models. This allows users to write their own set of equations in a flexible way.

The basic workflow with TOP is the following:

1. write an equation file (see *Equation Files*)
2. compile this file with `top-build`
3. compute oscillations modes and frequencies with `top` python module (see *Python API*):
    - read input parameters
    - read a stellar model
    - run the Arnoldi-Chebyshev method

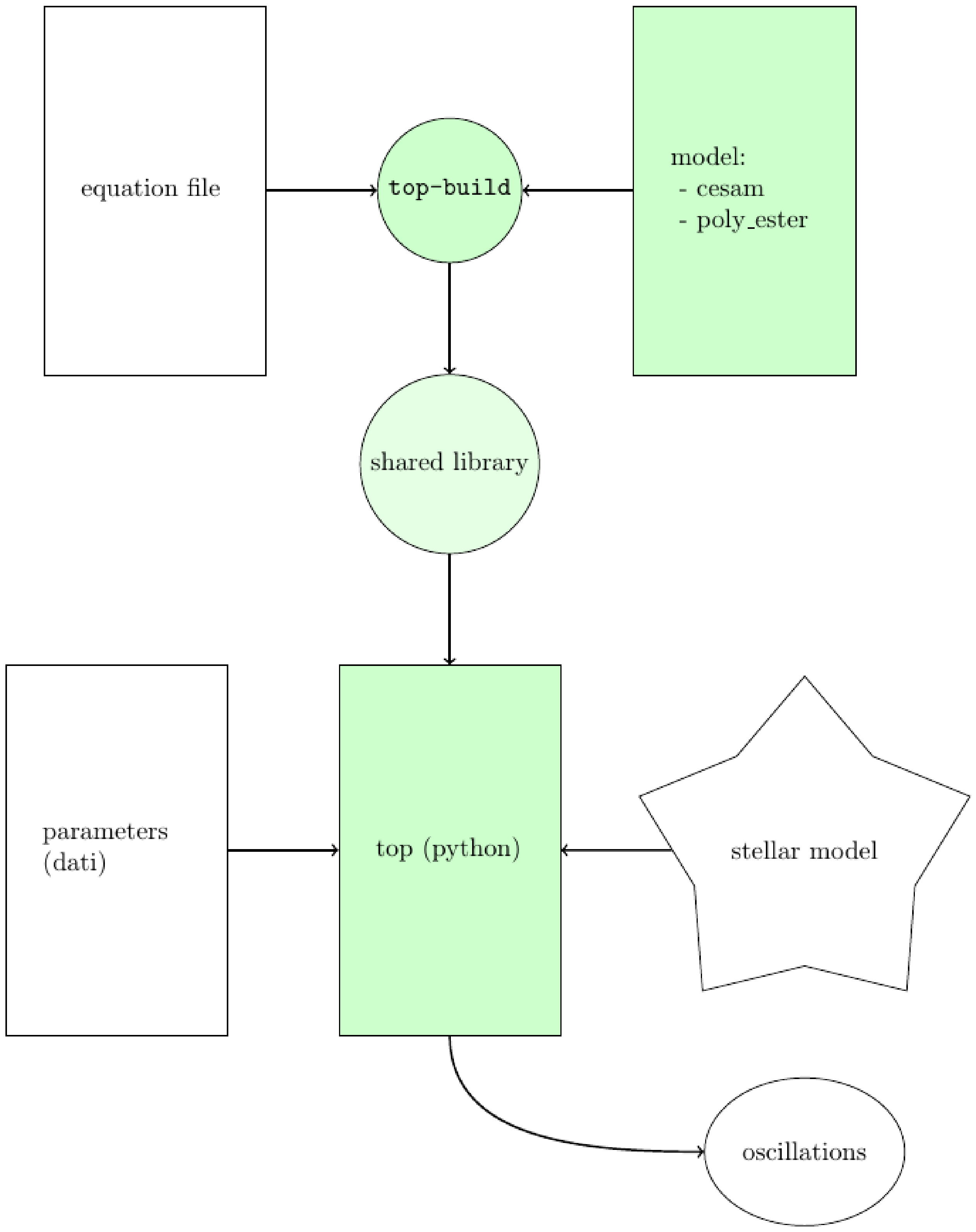

Fig. 1.1: TOP's Software Architecture

CHAPTER TWO

DOCUMENTATION

2.1 Install

2.1.1 Quick Install Guide

TOP uses the standard `autotools` (autoconf, automake) install procedure, you should be able to install it using:

```
# get the code:
git clone https://github.com/top-devel/top.git

# enter the source code directory:
cd top

# download the optional dependencies:
git submodule init
git submodule update

# prepare the configure script:
./bootstrap

# configure and install TOP:
./configure && make install
```

You can get more details and options reading the following sections: *getting the code*, *configure* and *compile and install*.

2.1.2 Getting the code

From the git repository

The best way to get the latest version of the code is to download it from the repository:

```
# with ssh:
git clone git@gitlab.com:top-dev/top.git

# with https:
git clone https://gitlab.com/top-dev/top.git
```

This will download the latest version of the code in a directory named `top`. Enter this directory:

```
cd top
```

You can optionally download the parser to write oscillation equations with the new equation format:

```
git submodule init
git submodule update
```

Run the `bootstrap` script that will create the configure script:

```
./bootstrap
```

Note: In order to run the `bootstrap` script you will need to have `autoconf` (>=2.59), `automake` (>=1.9) and `libtool` installed.

You can then proceed to the *configure* steps.

From a source archive

Download a source archive from the *download page*.

If you don't need the latest version of TOP, you can use a source archive. Extract your source archive and enter to source directory:

```
tar xvjz top-x.y.tar.bz2
cd top-x.y
```

And proceed to the *configure* steps.

2.1.3 Configure

Prerequisites:

The configure script allows you to configure the build environment of TOP. In order to install TOP, you will need:

- a Fortran compiler supporting procedure interface (`gfortran (>=4.9)`)
- a recent version of Python (`python (>=2.7)`)
- the program `f2py`, usually shipped with `numpy`
- the following python modules: `numpy` and `h5py`

Configure will try to detect the libraries installed in your system, if it fails to find both a BLAS and a LAPACK library it will return an error. You can try to re-run configure with some of the following option to help it find you libraries:

Configure options:

- `FC`: allows you to choose your Fortran compiler (e.g. `FC=gfortran`)
- `LDFLAGS`: sets linker flags. This can be used to specify libraries search directory (e.g. `LDFLAGS=-L$HOME/local/lib`)
- `LIBS`: what libraries should be linked with TOP. (e.g. `LIBS=-ltatlas`)
- `CPPFLAGS`: preprocessor flags, this can be used to tell the compiler where to find header files (e.g. `CPPFLAGS=-I$HOME/local/include`)
- `PYTHON`: the python interpreter to use (e.g. `PYTHON=python3`)
- `--prefix=`: this option allows you to set TOP's install directory (by default the prefix is set to `$HOME/local`)

Example:

If you want to use Intel compiler (`ifort`) and the ATLAS library (installed in `$HOME/local/lib`), you want to configure with the following command line:

```
./configure FC=ifort LDFLAGS=-L$HOME/local/lib LIBS=-ltatlas
```

2.1.4 Compile & Install

After running successfully the configure script, you can compile and install TOP by running:

```
make install
```

TOP is composed of a compiler wrapper `top-build` installed in `$prefix/bin`, a few libraries installed in `$prefix/lib` and a python module installed in `$prefix/lib/python-version/site-packages/top`.

As few examples are also availiable in `$prefix/share/top/models`

Note: You can source the shell script `activate-top.sh` created in the directory where you compiled TOP to set up the environment variables PATH, LD_LIBRARY_PATH and PYTHONPATH with the path where TOP was installed.

2.1.5 Check you Install

See *usage*.

2.1.6 Using `libester`

In order to use ESTER stellar models, TOP needs to find where ESTER was installed on your system. In order to tell TOP's configure script where to find `libester`, you need to provide it with the options: `LDFLAGS=-L$PATH_TO_ESTER/lib` and `CPPFLAGS=-I$PATH_TO_ESTER/include`.

For instance if ESTER was installed in `$HOME/local`, TOP should be able to find it if you configure with:

```
./configure LDFLAGS=-L$HOME/local/lib CPPFLAGS=-I$HOME/local/include
```

2.2 Usage

TOP consists of a language that defines equations to be solved (see *equation files*), a compiler wrapper designed to compile these equation files, and a python module to actually run computations.

2.2.1 Compiling an Equation File

Compiling an equation file is performed by the compiler wrapper `top-build`. Equation files are attached to a given star model, you have to provide `top-build` with the model corresponding to the equation file with the `--model=` option.

Example `top-build --model=poly_ester eq_poly_ester`

Options

The options you can pass to `top-build` are:

--parser	use the parser for the new language (see *New Equation Format*).
--order=FILE	use the file named `FILE` instead of the default one to manage the order of variable and equation.
--model=NAME	this option is mandatory, it tells TOP which stellar model to use.
--cplx	forces TOP to compile with the complex version.
--debug	enable debug mode for the file compiled.

2.2.2 Running TOP

In order to use your newly compiled equation file, you need to use the top python module:

```python
import top                          # imports the top module
import numpy as np                  # imports numpy

p = top.load('eq_poly_ester')       # loads your compiled equation file
p.read_dati('dati')                 # reads the parameter file `dati`

model = 'model/'                    # path to the model
shift = p.dati.shift

m = p.init_model(model)             # initializes the model stored in the directory
r = p.run_arncheb(shift)            # runs Arnoldi-Chebyshev method

# get the solutions
for i in range(0, r.nsol):          # for all solutions
    for var in p.get_vars(0):       # for each variable (in the fist domain)
        # plot the solution
        r.plot(0, i, v)             # quick plot of the solution

        # plot an expression of the solution:

        # get the solution
        val, vec, l = r.get_sol(0, i, v)

        # get the grid
        radius, theta = r.get_grid()
        cost = np.cos(theta)

        # get a field from the model
        h = m['hh']

        # project the solution onto the grid
        gv = top.leg.eval2d(vec, cost, l[0], 2, r.dati['m'])

        # actually plot the expression
        r.plot_val(gv*np.sqrt(h)**r.dati['pindex'])
```

For a detailed description of all functionalities available through the python module, see *python API*.

2.3 Stellar Models in TOP

This page describes the star models supported by TOP as well as the fields defined in the model.

2.3.1 Polytropic Model

Name `poly_ester`

Fields:

`hh`

`hht`

`hhz`

`hhzz`

`hhzt`

`lnhht`

`lambda`

`alpha`

`aplat`

`omega_K`

`omga`

`r_t`

`r_z`

`r_map`

`re_t`

`re_z`

`re_map`

`r_zz`

`r_zt`

`r_tt`

`re_zz`

`re_zt`

`re_tt`

`zeta`

`cost`

`sint`

`cott`

2.3.2 ESTER Model

Name `ester`

Fields:

`zeta`

`theta`

2.3.3 CESAM Model

Name `cesam`

Fields:

`rhom` density ρ

`rhom_z` $\frac{\partial \rho}{\partial \zeta}$

`rhom_t` $\frac{\partial \rho}{\partial \theta}$

`pm` pressure p

`pm_z` $\frac{\partial p}{\partial \zeta}$

`pm_t` $\frac{\partial p}{\partial \theta}$

`Gamma1` first adiabatic exponent

`NN` Brunt-Vaisala frequency (not perturbated)

`NNr` $\frac{\partial NN}{\partial \zeta}$

`NNt` $\frac{\partial NN}{\partial \theta}$

`pe` pressure potential

`pe_z` $\frac{\partial pe}{\partial \zeta}$

`pe_t` $\frac{\partial pe}{\partial \theta}$

2.4 Equation Files

There are 2 different formats to write oscillation equations in TOP: the *legacy* one and the *newer* one:

2.4.1 Legacy Equation Format

In this format, equations consists of series of commands that build the numerical system.

Overview

In order to write oscillation equations with this format, one need to write them projected onto the spherical harmonics basis.

Equation example

$$\lambda b_m^l = \sum_{l'=|m|}^{\infty} - \iint_{4\pi} \{Y_l^m\}^* Y_{l'}^m d\Omega \partial_\zeta u_m^{l'} - \iint_{4\pi} \frac{2\zeta H + \zeta^2 N H_\zeta}{r^2 r_\zeta} \{Y_l^m\}^* Y_{l'}^m d\Omega u_m^{l'} + ...$$

In order to write such an equation in TOP, one need to split it into terms and add them incrementally in the numerical system. The purpose of TOP, in particular `top-build` is to ease writing such equations. Several commands can help inserting such terms in the numerical system, see section *Commands* for a list of commands available in TOP's language.

Few other features can help understanding how to write equations in TOP: for a description of the type of terms TOP can handle, see section *Type of Terms*. And see section *String pre-processing* for a description of string pre-processing and special variables.

Type of Terms

For each term to be added in the numerical system, one need to provide TOP with its type. The type of a term specify whether it depends on nothing (scalar), the radial coordinate r, l or l'.

The type of terms currently supported by TOP are:

s scalar term

r term only depending on r (or ζ) (only available for 1D equations)

tt term depending on l and l' (only available in 2D equations)

rt term depending on r (or ζ) and l (only in 2D equations)

rtt term depending on r (or ζ), l and l' (only available in 2D equations)

String pre-processing

$a stands for the current coupling matrix. This variable can only be used in a term definition

$prev the previous coupling matrix. It correspond to variable `$a` of the last term definition.

$leq the array containing l values of the current equation.

$lvar the array containing l values of the current variable.

$leq(x) is the x^{th} l value of the current equation.

$lvar(x) is the x^{th} l value of the current variable.

$eq index of the current equation.

$var index of the current variable.

$nr radial resolution of the problem.

$i indices of the radial coordinate (this will result in the generation of a FORTRAN loop over all radial points).

$jl indices of the horizontal coordinate l (this will result in the generation of a FORTRAN loop over all values of l).

`$j2` indices of the horizontal coordinate l' (this will result in the generation of a FORTRAN loop over all values of l').

Commands

input

This command allow the user to define a parameter for the set of equation to be written.

Syntax `input parameter format`

Arguments

- `parameter`: name of the parameter.
- `format`: FORTRAN format specifier. This is used to read the parameter input file.

Example `input mass 0pf5.2`

stamp

Used to define a string to appear in the output files.

Syntax `stamp string`

Arguments

- `string`: string to appear in output files.

Example `stamp eq_ESTER_all_lagrange`

definition

Used to define named constants.

Syntax `definition type name value`

Arguments

- `type`: type of the constant (`integer`, `double_precision`, `complex`)
- `name`: name of the constant
- `value`: value of the constant

Example `definition double_precision gamma_p 1d0 + 1d0/pindex` will define `gamma_p` with a value of $1 + \frac{1}{pindex}$ (where $pindex$ has to be another variable (`definition` or `input`)

eqlist

Defines the name of equation to be defined in the equation file.

Syntax `eqlist eq1 eq2 eq3 ...  # and so on`

Arguments

- `eq1`: name of the equation

Example `eqlist eqEr eqdP eqPhi eqPhiP` defines 4 equations named eqEr, eqdP, eqPhi and epPhiP

varlist

Defines variables of the equation set.

Syntax `varlist var1 var2 var3 ...  # and so on`

Arguments

- `var1`: name of the variable

Example `varlist Er dP Phi PhiP` defines 4 variables named Er, dP, Phi and PhiP

leq

In TOP equation are projected into the spherical harmonic basis. This command is use to define the starting l for this projection.

Syntax `leq eqName value`

Arguments

- `eqName`: name of the equation
- `value`: starting value of l

Example `leq eqEr abs(m)+iparity`

lvar

In TOP variables are projected into the spherical harmonic basis. This command is use to define the starting l for this projection.

Syntax `lvar varName value`

Arguments

- `varName`: name of the variable
- `value`: starting value of l

Example `lvar Er abs(m)+iparity`

equation

This command is used to start defining an equation. This means that further command in the equation file will apply to the *current* equation.

Syntax `equation eqName`

Arguments

- `eqName`: name of the equation

Example `equation eqEr`

sub

This is use to insert a term in the *current* equation: this term will be computed by calling a FORTRAN subroutine.

Syntax `sub type power routine variable`

Arguments

- `type`: the type of term see *type of terms in TOP*.
- `power`: the power of the eigenvalue preceded by a `w`.
- `routine`: name of the FORTRAN subroutine to be called to compute the coupling coefficient.
- `variable`: name of the variable involved in the coupling. Further characters can be used indicate radial derives. For instance, `Er'` mean $\frac{\partial Er}{\partial r}$. Higher derivative order can be achieved either by chaining the `'` character or with the `^` character followed by the derivative order: `Er^2` is equivalent to `Er''`.

Example `sub rtt w1 Illm(sint/roz,$a,$leq,$lvar) u`: this basically add the term $\omega \int\int(\frac{sin(\theta)}{roz}) * u$ in the current equation, where ω is the eigenvalue.

subbc

This is use to insert a boundary condition term in the *current* equation: this term will be computed by calling a FORTRAN subroutine.

Syntax `subbc type location power routine variable(index)`

Arguments

- `type`: the type of term see *type of terms in TOP*.
- `location`: the location where the boundary condition should be inserted in the **numerical** system. This is basically tells the line in the matrix to be replaced with the boundary condition.
- `power`: the power of the eigenvalue preceded by a `w`.
- `routine`: name of the FORTRAN subroutine to be called to compute the coupling coefficient.
- `variable`: name of the variable involved in the coupling. Further characters can be used indicate radial derives. For instance, `Er'` mean $\frac{\partial Er}{\partial r}$. Higher derivative order can be achieved either by chaining the `'` character or with the `^` character followed by the derivative order: `Er^2` is equivalent to `Er''`.
- `index'` radial coordinate of the boundary condition.

Example `subbc tt nr w0 Illmbc(hhz(1,:),$a,$leq,$lvar) v(1)`, here we can see that `location` and `index` are different: the boundary condition is imposed at the center (`v(1)` stands for v at $r = 0$), but in the **numerical** system, the condition is imposed on the last line of the matrix.

term

Used to insert a term in the equation.

Syntax `term type power expression variable`

Arguments

- `type`: the type of term see *type of terms in TOP*.

- `power`: the power of the eigenvalue preceded by a `w`.
- `expression`: the mathematical expression of the term to be inserted.
- `variable`: name of the variable involved in the coupling. Further characters can be used indicate radial derives. For instance, `Er'` mean $\frac{\partial Er}{\partial r}$. Higher derivative order can be achieved either by chaining the `'` character or with the `^` character followed by the derivative order: `Er^2` is equivalent to `Er''`.

Example `term s w0 -2d0 Pi''`: this would insert the term $-2\frac{\partial^2 Pi}{\partial r^2}$ in the current equation.

termbc

Used to insert a term in a boundary condition of the system.

Syntax `termbc type location power expression variable(index)`

Arguments

- `type`: the type of term see *type of terms in TOP*.
- `power`: the power of the eigenvalue preceded by a `w`.
- `location`: the location where the boundary condition should be inserted in the **numerical** system. This is basically tells the line in the matrix to be replaced with the boundary condition.
- `expression`: the mathematical expression of the term to be inserted.
- `variable`: name of the variable involved in the coupling. Further characters can be used indicate radial derives. For instance, `Er'` mean $\frac{\partial Er}{\partial r}$. Higher derivative order can be achieved either by chaining the `'` character or with the `^` character followed by the derivative order: `Er^2` is equivalent to `Er''`.
- `index'` radial coordinate of the boundary condition.

Example `termbc t $nr w0 1d0 Phi($nr)'`: this would insert the term $\Phi(r = surf)$ in the boundary condition. (The last line of the matrix would be replaced with this boundary condition).

instruction

Used to add ad-hoc FORTRAN instruction in the module responsible for computing coupling integrals.

Syntax `instruction fortran`

Arguments

- `fortran`: the FORTRAN instruction to be inserted.

Example `instruction call modify_l0($prev,$nr,abs(m)+iparity)`: will insert the code `call modify_l0(dm(1)%artt(:,:,:),grd(1)%nr,abs(m)+iparity)`. See *String pre-processing*.

2.4.2 New Equation Format

Introduction

This equation format aims at simplifying the way to write oscillation equations. With this format equation are written in there mathematical expression after projection on the spherical harmonics.

Here is an example of such equation if the 1D case:

```
lh*(lh+1) * Et =
    avg(r/Gamma1) * dP_P    +
    avg(r)        * Er'     +
    2             * Er
with (r=1)
    dr(Er, -1) = lh * dr(Et, -1)
    at r = 0
```

Note: TOP solves eigenvalue problems. Therefore equations written for TOP must be linear.

Language Description

The Parameters Section

At the beginning of an equation file, one is able to define several parameter that can be used within the definition of equation. These parameters can be defined with the `input` keyword, followed by its type (`double`, `int` or `string`) ant its name:

Example

```
input double mass
input double rota
```

Variable and Model Definition

In order to *understand* the meaning of the equation to appear, TOP's compiler need to know what are the variables of the problem, and what are the field of the model.

Variable Definition

In order to define variables of the problem, one has to define them using the `var` keyword followed by a comma separated list of names.

Note: 2 or 3 variables surrounded by parenthesis means that they are component of vector.

Example

```
var Phi, PhiP, (Er, Et), dP_P
```

Will define 5 variables, `Er` and `Et` being first and second component of a vector.

Model's Field definition

In the same way variables can be defined, fields and scalar defined from the stellar model can be defined with `field` or `scalar` keywords:

Example

```
field pm, g_m, r, rhom, dg_m, rhom_z
scalar Gamma1, Lambda
```

Equation Definition

After the definition/declaration section, we can start defining the equation after the `in` keyword.

Defining an equation

In order to add an equation in the system, one can use the `equation` keyword, followed by the name of the equation, followed by a **:** (colon) and the expression of the equation.

Example

```
equation eqdP_P:
lh*(lh+1) * Et =
    avg(r/Gamma1) * dP_P    +
    avg(r)        * Er'     +
    2             * Er
```

Note: Every identifier (*i.e.,* name) involved in an equation need to be defined (either as a *variable*, a *field or a scalar* or *a parameter*).

Boundary Condition

In order to define boundary condition, one simply need to define of after the equation with the following syntax:

Syntax `with (r=numerical_location) epxression at r = location` where:

numerical_location is the line of the matrix to be replaced with the boundary condition.

expression if the expression of the boundary condition.

location is the *physical* location of the boundary condition.

Example

```
equation eqdP_P:
lh*(lh+1) * Et =
    avg(r/Gamma1) * dP_P    +
    avg(r)        * Er'     +
    2             * Er
with (r=1)
    dr(Er, -1) = lh * dr(Et, -1)
    at r = 0
```

Internal variables, and Functions

A few functions and variables are already defined with TOP and can be used without prior declarations, here is a list of such symbols:

Internal Variables

fp

Syntax `fp`

Semantics `fp` is the eigenvalue of the problem. It should appear in equation definition

Example `fp^2 * r * Et -pm/rhom * dP_P -Phi -g_m = 0`

Internal Functions

dr

Syntax `dr(var,order)`

Semantics derivative of `var` of order `order`

Example `dr(Phi,2)` stands for $\frac{\partial^2\Phi}{\partial r^2}$

Note: Radial derivatives can also be expressed with the `'` (apostrophe) post-fixed operator: `dr(Phi,2)` and `Phi''` are two notations strictly equivalent.

avg

Syntax `avg(expr)`

Semantics average of expression `expr` on the point of the grid used for for derivation or interpolation (therefore it depends on the numerical scheme used).

Example `avg(r/Gamma1) * dP_P`

Comments

Comments can be added in equation file using a pound sign (#), the remaining of the line will be ignored.

Example

```
# define the first equation
equation eqdP_P:
lh*(lh+1) * Et =            # this is the LHS of the equation
    avg(r/Gamma1) * dP_P +  # this the RHS
    avg(r) * Er'
```

2.5 Python API

2.5.1 Equation

The *top.load* class in responsible for loading a previously compiled equation file, loading or setting parameters, loading stellar models and solving the underlying eigenvalue problem.

class `top.load`(*name*)

Main class for running TOP:

It is responsible for loading a previously compiled equation file, initializing the model and running the Arnoldi-Chebyshev algorithm

Parameters **`name`** (*`str`*) – name of the equation filed to load

Example poly = load('eq_poly_ester')

`get_grid`()

returns the grid used by the model (requires a call to init_model)

Return type tuple (r, theta)

Returns the grid coordinates *r* and *theta*

`get_results`()

returns an *result* object containing the results of a call to run_arncheb

Return type results object

Returns results of the previous call to run_arncheb

`get_sol`(*idom*, *isol*, *var*)

returns *isol* th solution in domain number *idom* for variable named *var*

Return type tuple (eigenvalue, eigenvector, Legendre degree)

Returns eigenvalue, eigenvector of the solution. The solution is given in spectral space (Legendre), so the of the degrees of Legendre polynomial are also returned

Parameters

- **`idom`** (*`int`*) – domain index of the solution
- **`isol`** (*`int`*) – solution number
- **`var`** (*`str`*) – variable name

`get_vars`(*idom*)

returns a list of variables in domain *idom*

Return type [str]

Returns list of variables in domain number idom

Parameters **`idom`** (*`int`*) – domain index

`get_version`()

returns the version of TOP that compiled the currently loaded equation file

`get_zeta`()

returns the zeta radial coordinate

Return type tuple (r, theta)

Returns the grid coordinates *r* and *theta*

`init_model`(*filename*)

initializes the star model with the file *filename*

Parameters **`filename`** (*`str`*) – file to read to initialize the star model

`read_dati`(*filename*)

reads dati parameter file named *filename*

Parameters **`filename`** (*`str`*) – dati file to read

`run_arncheb`(*shift*)
runs the Arnoldi-Chebyshev algorithm with a shift provided by the *shift* argument. Calls call to run_arncheb must be preceded by a call to *read_dati* to initialize problem parameters and a call to *init_model* to initialize the start model.

Parameters **`shift`** (*`float`*) – starts the Arnoldi-Chebyshev algorithm arnoud frequency given by shift

Return type results object

Returns the results of the Arnoldi-Chebyshev algorithm

`write_output`(*dir*)
write the results computed by run_arncheb to location *dir*, this for retro compatibility purpose

2.5.2 Results Objects

class `top.`**`results`**(*result_file=''*)
This class stores results of a previous computation

Parameters **`result_file`** (*`str`*) – (optional) file with previous results stored

Return type `results` object

Returns results of a previous computation (if result_file is provided) an empty results object otherwise

`append`(*res*)
appends results stored in res

`get_grid`(*idom=-1*)
returns the grid used by the model (requires a call to init_model)

if *idom* is provided returns the grid only for this particular domain

`get_sol`(*idom, isol, var*)
returns the *isol* th solution in domain number *idom* for variable named *var*

Return type tuple (eigenvalue, eigenvector, Legendre degree)

Returns eigenvalue, eigenvector of the solution. The solution is given in spectral space (Legendre), so the of the degrees of Legendre polynomial are also returned

Parameters

- **`idom`** (*`int`*) – domain index of the solution
- **`isol`** (*`int`*) – solution number
- **`var`** (*`str`*) – variable name

`plot`(*idom, isol, var, m=None*)
Plots isol'th solution of variable var in domain number idom

Parameters

- **`idom`** (*`int`*) – domain number
- **`isol`** (*`int`*) – solution to plot
- **`var`** (*`str`*) – variable to plot

`plot_val`(*mat*)
Plots the values stored in *mat*

Parameters **mat** (*matrix*) – function to be plotted

read_model()

Reads the model that was used to perform the computation

Return type `model` (see *model*)

Returns the model used to perform computation

save(*h5file*, *ids=[]*)

saves the results in HDF5 format in file named *h5file*

2.5.3 Star Models

In order to access fields of the model, you can use square bracket operator *[]* with the name of the field between brackets (e.g, *m['h']* would return the enthalpy of the model *m*).

But you need to know what are the fields defined for each model. See *models* for a list of supported models in TOP, and a list a fields defined by each model.

class top.**model**(*libmodel*, *filename*)

This class storing a star model

access to fields of the model can be achieved with square bracket operator. But you need to know the fields stored in your model.

Example m['w'] # access the field omega of the model *m*

2.6 Usage Examples

2.7 Download

Download sources releases at: https://top-devel.github.io/top/download.html

INDEX