Documentation for PYRATS: Python Routines for Analyzing Transport Simulations

Introduction: PYRATS is a data processing tool designed to make creating plots of output data from GYRO, TGYRO, or NEO easy. It makes use of the python programming language, and the associated packages numpy and matplotlib. With minimal knowledge of the python language and matplotlib, the typical user should be able to create plots quickly and efficiently from the python command line interpreter.

To begin, type the following command into the terminal:

```
$ python
```

This will bring up the python interpreter, which is the interface you will use to interact with PYRATS. As a first example, we will look at the layout of TGYROData. Now execute the following commands:

```
>>> from pyrats.tgyro.data import TGYROData

>>> help(TGYROData)
```

This will bring up the built-in documentation for the class TGYROData. It lists all of the methods and attributes of the class, and the function of each one. This information is also contained on page x of this manual.

TGYROData can be used to load TGYRO data with the command:

```
>>> sim1 = TGYROData('example_directory')
```

where 'example_directory' is a directory containing TGYRO output files. The data is loaded into objects corresponding to the output file which the data came from.

profiles_genData contents:

- Data:
  - **data** *dictionary of numpy arrays* - Contains all of the data read in by profiles_genData. It is organized in a dictionary with the keys being the different column headers in the input.profiles file that is read. The keys for the dictionary can be requested with the following commands:

    ```
    $ python
    >>> from pyrats.profiles_gen.data import profiles_genData
    >>> sim1 = profiles_genData('example_directory')
    >>> sim1.data.keys()
    ```

    where, again, 'example_directory' is a directory containing the file input.profiles.
  - **n_exp** *int* - The number of timesteps in the simulation. It is also the length of each array in data.
  - **hlen** *int* - The length of the header of file input.profiles in rows.
  - **fignum** *int* - The number of the current active matplotlib figure.
  - **plotcounter** *int* - The number of the current active axes on the current active matplotlib figure.
  - **ar** *nested list of floats* - The sine coefficients for the r-component of the fourier series representation of the flux surfaces. Read from input.profiles.geo.
  - **br** *nested list of floats* - The cosine coefficients for the r-component of the fourier series representation of the flux surfaces. Read from input.profiles.geo.
  - **az** *nested list of floats* - The sine coefficients for the z-component of the fourier series representation of the flux surfaces. Read from input.profiles.geo.
  - **bz** *nested list of floats* - The cosine coefficients for the z-component of the fourier series representation of the flux surfaces. Read from input.profiles.geo.
  - **directory_name** *str* - The name of the directory containing file input.profiles and possibly input.profiles.geo.
- Methods:
  - **__init__(*str* directory='.')** - This is the constructor for the class. When a new instance of the class is created, the constructor is called and executed. The only argument is the name of the directory from which to build the class, and it defaults to the current directory. It calls these methods in the following order:

1. set_directory(directory)
2. init_data()
3. store_data()

- **compplot(*float* inner, *float* outer, *int* n, *bool* verbose) -** This method creates overlaid flux surface plots using both the fourier series decomposition method and the shaped Grad-Shafranov Miller-type equilibrium for the flux surfaces.
  - Inner specifies the innermost flux surface to be plotted
  - outer specifies the outermost flux surface to be plotted
  - n specifies the number of flux surfaces to plot in between inner and outer
  - if verbose is true, the legend will display the location of each flux surface
- **compute_fouriereq(*float* r) -** This method calculates the flux surface at radius r according to the general Grad-Shafranov Fourier-series equilibrium.
- **compute_mtypeeq(*float* r) -** This method calculates the flux surface at radius r according to the shaped Grad-Shafranov Miller-type equilibrium.
- **fourierplot(*float* inner, *float* outer, *int* n, *bool* verbose) -** This method creates flux surface plots using only the fourier series decomposition method for the flux surfaces.
  - inner specifies the innermost flux surface to be plotted
  - outer specifies the outermost flux surface to be plotted
  - n specifies the number of flux surfaces to plot in between inner and outer
  - if verbose is true, the legend will display the location of each flux surface
- **get(*str* var) -** This method returns the numpy array corresponding to var.
- **init_data() -** This method initializes all of the data objects.
- **match(*float* val, *list* vec) -** This method finds the closest match to val in a *list* of values (vec) and returns the index of that value.
- **millerplot(*float* inner, *float* outer, *int* n, *bool* verbose) -** This method creates flux surface plots using only the shaped Grad-Shafranov Miller-type equilibrium for the flux surfaces.
  - inner specifies the innermost flux surface to be plotted
  - outer specifies the outermost flux surface to be plotted
  - n specifies the number of flux surfaces to plot in between inner and outer

    * if verbose is true, the legend will display the location of each flux surface.

- **plot(*str* var, *int* n1=2, *int* n2=2, *int* plotcounter=0, *int* fignum=0)** - This method creates plots of the requested data (var) using matplotlib.
    * n1 is the horizontal number of plots in one window
    * n2 is the vertical number of plots in one window
    * plotcounter is the position on which the new graph is to be placed
    * fignum is the number of the matplotlib figure on which to place the new graph
- **read_data()** - This method reads in data from input.profiles. It returns a dictionary containing the data that was read in.
- **read_fourier()** - This method reads in data from input.profiles.geo, and stores that data in the class objects ar, br, az, and bz.
- **set_directory(*str* directory)** - This method sets the class attribute directory_name to directory.
- **store_data()** - This method reads data and renames it appropriately. It is necessary because the names of the differetn parameters are not uniformly formatted. store_data cleans them up by inserting spaces where necessary, and by deleting #-signs when necessary.

NEOData contents:

- Data:
  - **master** *str* - Holds the name of the master directory containing NEO output files.
  - **directory_name** *str* - Holds the name of the currently open directory.
  - **fignum** *int* - The number of the current active matplotlib figure.
  - **plotcounter** *int* - The number of the current active axes on the current active matplotlib figure.
  - **toplot** *list* - The list of variables to plot when a plot command is called.
  - **transport** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.transport. The keys are the transport variables, each of which correspond to a NEOObject.
  - **HH_theory** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.theory. The keys are the flows and fluxes predicted by the Hinton-Hazeltine model, each of which correspond to a NEOObject.
  - **CH_theory** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.theory. The keys are the ion heat fluxes predicted by the Chang-Hinton model, each of which correspond to a NEOObject.
  - **TG_theory** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.theory. The keys are the ion heat fluxes predicted by the Taguchi model, each of which correspond to a NEOObject.
  - **S_theory** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.theory. The keys are the bootstrap currents predicted by the Sauter model, each of which correspond to a NEOObject.
  - **HS_theory** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.theory. The keys are the fluxes predicted by the Hirshman-Sigmar model, each of which correspond to a NEOObject.
  - **control** *dictionary of dictionaries of NEOObjects* - Contains data read in from out.neo.control.
- Methods:
  - **__init__(***str* **sim_directory)** - Constructor that is executed when a new NEOData object is created. It takes a directory name as its

argument, and then executes a top down walk down that directory. It then calls read_data() whenever it is in a subdirectory with NEO output files. Finally, it executes store_data()

– **get_CH_theory(*str* var)** - Returns a NEOObject object that corresponds to the Chang-Hinton theory prediction of var. If requested variable does not exist or is zero everywhere, returns None.

– **get_HH_theory(*str* var)** - Returns a NEOObject object that corresponds to the Hinton-Hazeltine theory prediction of var. If requested variable does not exist or is zero everywhere, returns None.

– **get_HR_theory(*str* var)** - Returns a NEOObject object that corresponds to the Hinton-Rosenbluth theory prediction of var. If requested variable does not exist or is zero everywhere, returns None.

– **get_HS_theory(*str* var)** - Returns a NEOObject object that corresponds to the Hirshman-Sigmar theory prediction of var. If requested variable does not exist or is zero everywhere, returns None.

– **get_S_theory(*str* var)** - Returns a NEOObject object that corresponds to the Sauter theory prediction of var. If requested variable does not exist or is zero everywhere, returns None.

– **get_TG_theory(*str* var)** - Returns a NEOObject object that corresponds to the Taguchi theory prediction of var. If requested variable does not exist or is zero everywhere, returns None.

– **get_control(*str* var)** - Returns a NEOObject object that corresponds to the control parameter associated with var. If requested variable does not exist or is zero everywhere, returns None.

– **get_input(*str* input_name)** - Returns requested variable from input.neo.gen.

– **init_data()** - Initializes object data.

– **plot(*str* var, *int* n1=2, *int* n2=2, *int* plotcounter=0, *int* fignum=0, *bool* legend=True, *bool* verbose=False, *str* cols='bgkcmyrw', *list of strs* styles=['-', '–', '-.', ':'])** - Plots var as a matplotlib scatter plot with data from different directories coming in different colors, and different species coming in different line styles. Automatically searches for both theoretical and simulated values, and plots everything that is available.

– **print_vars()** - Prints all available simulated variables.

– **read_data()** - Read in object data. Calls read_grid(), read_equil(), read_theory(), read_transport(), read_transport_gv().

– **read_equil()** - Reads out.neo.equil. The data is eventually stored by store_data() in control.

– **read_file()** - Loads data from NEO output file into buffer for manipulation and storage.

- **read_grid()** - Reads out.neo.grid. The data is eventually stored by store_data() in control.
- **read_theory()** - Reads out.neo.theory. The data is eventually stored by store_data() in HH_theory, TG_theory, CH_theory, S_theory, HR_theory, and HS_theory.
- **read_transport()** - Reads out.neo.transport. The data is eventually stored by store_data() in transport.
- **read_transport_gv()** - Reads out.neo.transport_gv. The data is eventually stored by store_data() in transport.
- **set_directory(*str* path)** - Sets the current directory to path.
- **split(*list* array)** - Takes a 2-D list which has entries with multiple elements and converts it to a 2-D list with only one element per entry.
- **store_data()** - Stores data into data dictionaries by variable name and directory. Data can be accessed with two dictionary keys, like so:

  sim1.transport[parameter][directory].

TGYROData contents:

- Data:
  - **n_iterations** *int* - Number of TGYRO iterations in simulation.
  - **n_fields** *int* - Number of fields in simulation.
  - **n_radial** *int* - Number of radial gridpoints in simulation.
  - **directory_name** *str* - Name of loaded directory.
  - **chi_e** *dictionary of lists of numpy arrays* -