

PYRATS: Python Routines for Analyzing Transport Simulations

A suite of python classes for accessing and plotting data from GYRO/NEO/TGYRO

Introduction: PYRATS (Python Routines for Analyzing Transport Simulations) 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 some 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. First is a presentation of some of the more basic features of PYRATS. Code documentation is given on the subsequent pages.

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. To create a basic plot of the particle densities and temperatures as a function of radius, type the following commands:

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(1)
>>> ax1 = fig.add_subplot(221)
>>> ax2 = fig.add_subplot(222)
>>> ax3 = fig.add_subplot(223)
>>> ax4 = fig.add_subplot(224)
>>> ax1.plot(sim1.data['r/a'][-1], sim1.data['te'][-1])
>>> ax2.plot(sim1.data['r/a'][-1], sim1.data['ti'][-1])
>>> ax3.plot(sim1.data['r/a'][-1], sim1.data['ne'][-1])
>>> ax4.plot(sim1.data['r/a'][-1], sim1.data['ni'][-1])
>>> ax1.set_xlabel('r/a')
>>> ax1.set_ylabel('Electron Temperature (keV)')
>>> ax2.set_xlabel('r/a')
>>> ax2.set_ylabel('Ion Temperature (keV)')
>>> ax3.set_xlabel('r/a')
>>> ax3.set_ylabel('Electron Particle Density (1/cm^3)')
```

```
>>> ax4.set_xlabel('r/a')
>>> ax4.set_ylabel('Ion Particle Density (1/cm^3)')
>>> plt.show()
```

For the TGYRO example simulation treg01, the above code produces the following plots:

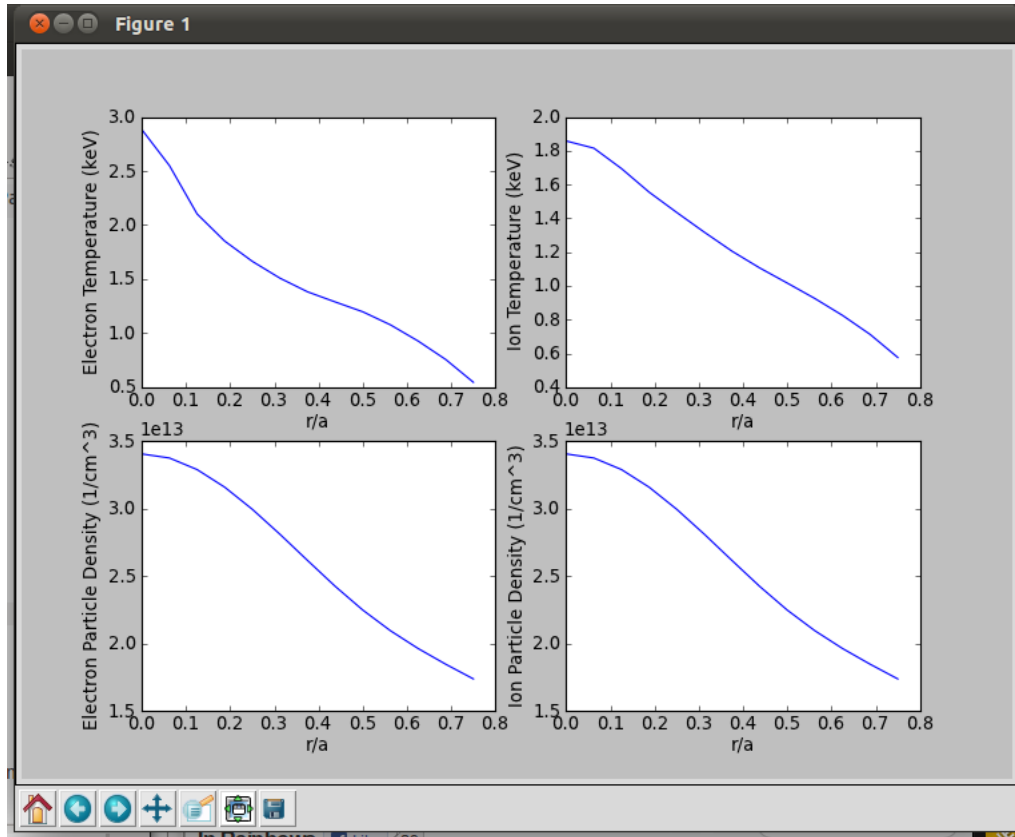


Figure 1: Summary plots for an example TGYRO simulation.

This functionality is also available from the terminal command line (as opposed to the python interpreter) with the simple command:

```
$ tgyro_plot -p ps
```

When executed from the treg01 directory, the above command produces almost the same plot:

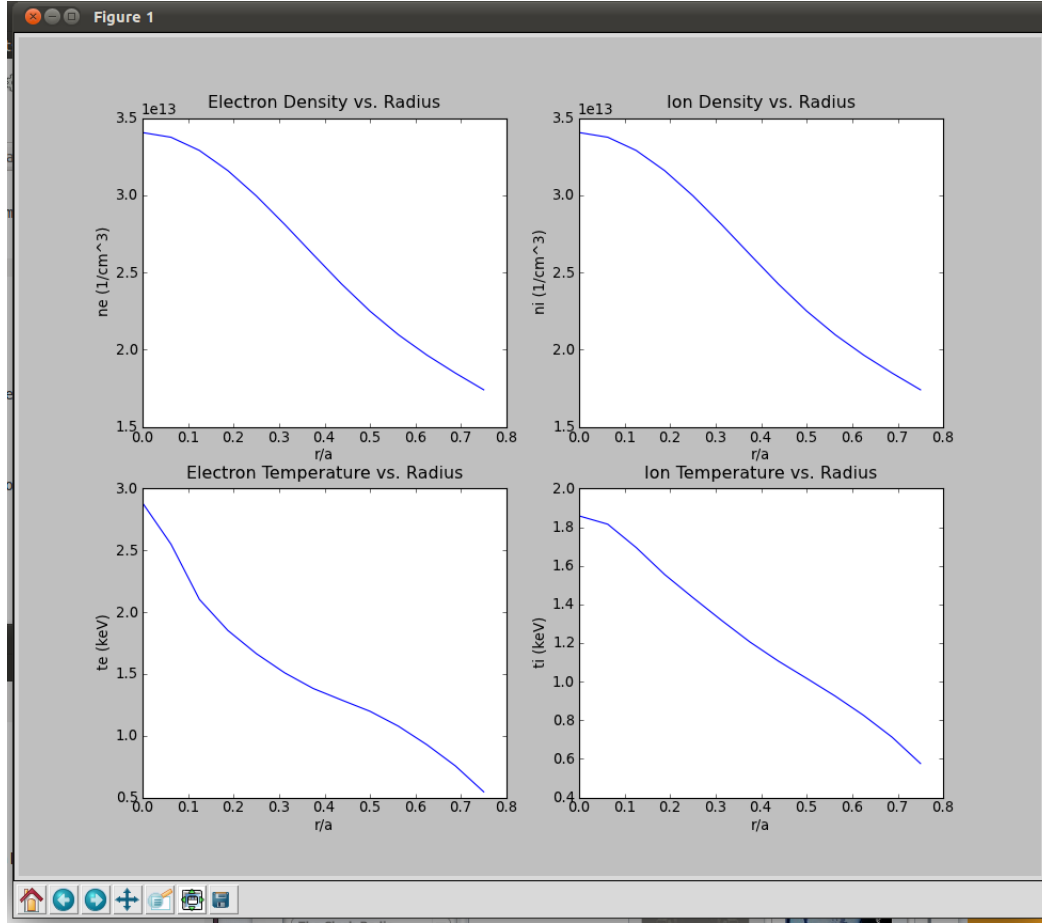


Figure 2: Summary plots for an example TGYRO simulation produced with command line tools.

The rest of this document is documentation for the classes that make up PYRATS. Each class also has an associated help file that describes the contents of the different data containers in more detail. The help for each code (GYRO, NEO, and TGYRO) can be accessed from the corresponding PYRATS subfolder in the gacode structure, like so:

```
~$ cd gacode/shared/python/pyrats
~gacode/shared/python/pyrats$ ls
build  examples.py  fileupload.pyx  gyro  __init__.pyc  profiles_gen  tgyro
doc    fileupload.c  fileupload.so   __init__.py  neo          setup.py
~gacode/shared/python/pyrats$ cd tgyro
~gacode/shared/python/pyrats/tgyro$ ls
data.py  data.pyc  help.py  __init__.py  __init__.pyc  interpreter.py
~gacode/shared/python/pyrats/tgyro$ python help.py ~/treg01 data
```

If there is a folder named treg01 in the home directory, this command will print out all of the variables stored in the "data" container.

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(*str 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 different parameters are not uniformly formatted. store_data cleans them up by inserting spaces where necessary, and by deleting #-signs when necessary.

NEO related classes:

NEOObject contents:

- Data:
 - **data** *dictionary* - Holds the data associated with the object.
 - **units** *str* - Describes the units of the data.
 - **descriptor** *str* - Provides a description of the data.
- Methods:
 - **__init__**(*dict* **data**, *str* **units**, *str* **descriptor**) - This method stores the appropriate arguments in the appropriate containers.

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_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 str* 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:
 - **loc_n_ion** *int* - Number of ions in simulation.
 - **tgYRO_mode** *int* - Mode number of TGYRO operation.
 - **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.
 - **data** *dictionary of lists of numpy arrays* - Large dictionary containing all loaded data by variable name and then by iteration number.
- Methods:
 - **__init__**(*str sim_directory*) - Constructor which is called when a new TRYGOData object is created. It calls the following methods:
 1. **set_directory**(*str sim_directory*)
 2. **init_data**()
 3. **read_data**()
 - **get_input**(*str input_name*) - Returns the specified variable from input.tgyro.gen.
Ex: **get_input**("TGYRO_MODE")
 - **get_local_res**(*int field=1, int iteration=-1*) - Returns a numpy array of the local field residual for the given field and iteration. The default is the last iteration, and field 1.
 - **get_stability_at_radius**(*int radius=0, str frequency='r', str direction='ion'*) - Returns a list of frequency vs. ky at specified radius from stability analysis. The parameter radius is the requested radial index; it is an integer between 0 and n_r-1. The parameter frequency can be either the real ('r') or the imaginary ('i') spectrum. The parameter direction specifies the direction of the spectrum. It can be either the ion direction ('ion') or the electron direction ('elec').
 - **init_data**() - Initializes object data.
 - **make_gradient_vs_field_plot**(*int r=1, int evolve_field=1, str grad='a/LTi', str profile='ti', bool arrows=False*) - Return a matplotlib.pyplot figure of gradient_vs_field_space. The parameter r is the radial index; it is an integer between 0 and n_r-1. The parameters are passed to **make_gradient_vs_field_space**, and are described there.
 - **make_gradient_vs_field_space**(*int r=1, int evolve_field=1, str grad='a/LTi', str profile='ti'*) - Returns a matrix of Space[iteration][gradient,profile,residual] at givel radial point. eeg: Space[[4.0, 1.7, 2.5], [3.6, 1.5, 0.1]] if grad 4.0 → 3.6 while Ti → 1.5 and local residual went from 2.5 → 0.1. The parameter evolve_field specifies the field type to pass to **get_local_res**. The parameter grad is the gradient factor to pass to **get_gradient.factor**. The parameter profile is the profile factor to pass to **get_profile.factor**.
 - **read_chi_e**() - Internal method. This method reads in chi_e.out and stores it in self.chi_e.
 - **read_chi_i**() - Internal method. This method reads in chi_i.out and stores it in self.chi_i.
 - **read_control**() - Internal method. This method reads in control.out to set resolutions.
 - **read_data**() - Internal method. This method reads in object data and calls the following other methods:
 1. **read_control**()

2. `read_chi_e()`
 3. `read_chi_i(loc_n_ion)`
 4. `read_gyrobohm()`
 5. `read_profile()`
 6. `read_geometry()`
 7. `read_flux(loc_n_ion)`
 8. `read_mflux(loc_n_ion)`
 9. `read_gradient()`
 10. `read_residual()`
- **`read_file(str file_name)`** - Reads TGYRO output file. Output is `data['column_header'][iteration]`
 - **`read_flux(int num_ions=1)`** - Reads `flux_e.out`, `flux_i(2-5).out`, and `flux_target.out`. `Num_ions` determines how many ion files to read.
 - **`read_geometry()`** - Reads and stores `geometry.out` in `self.geometry`.
 - **`read_gradient()`** - Reads and stores `gradient.out` in `self.gradient`.
 - **`read_gyrobohm()`** - Reads and stores `gyrobohm.out` in `self.gyro_bohm_unit`.
 - **`read_mflux()`** - Reads and stores `mflux_e.out`, `mflux_i(2-5).out`, and `mflux_target.out`.
 - **`read_num_ions()`** - Reads `LOC_N_ION` and stores it as `self.loc_n_ion`.
 - **`read_profile()`** - Reads and stores `profile.out` in `self.profile`.
 - **`read_profile2()`** - Reads and stores `profile2.out` in `self.profile2`.
 - **`read_profile3()`** - Reads and stores `profile3.out` in `self.profile3`.
 - **`read_profile4()`** - Reads and stores `profile4.out` in `self.profile4`.
 - **`read_profile5()`** - Reads and stores `profile5.out` in `self.profile5`.
 - **`read_residual()`** - Reads `residual.out`.
 - **`read_stab_file(str file_name)`** - Reads files generated with stability analysis mode. The parameter `file_name` is the name of the requested stability file, such as "`wi_elec.out`". It returns a list of `[r, ks, freq]`.
 - **`read_stabilities()`** - Reads output files from `TGYRO_METHOD=2`, and stores them into variables, as follows:
 - * `wr_ion.out` → `self.wr_ion`
 - * `wi_ion.out` → `self.wi_ion`
 - * `wr_elec.out` → `self.wr_elec`
 - * `wi_elec.out` → `self.wi_elec`
 - **`read_tgyro_mode()`** - Reads the `TGYRO_MODE` and stores it as `self.tgyro_mode`.
 - **`set_directory(str sim_directory)`** - Sets the simulation directory. Stores `sim_directory` in `self.directory_name`.

GYROData Contents:

- Data

- **directory_name** *str* - The name of the simulation directory.
- **profile** *dictionary of numpy arrays* - Dictionary where control information from out.gyro.profile is stored. The keys can be obtained by typing: `sim1.profile.keys()`
- **geometry** *dictionary of numpy arrays* - Dictionary where geometry data from out.gyro.geometry is stored. The keys can be obtained by typing: `sim1.geometry.keys()`
- **t** *dictionary of numpy arrays* - Dictionary where time data from t.out is stored. The keys are: 't/deltat', '(cbar_s/a)t', and 'n_time' which is an int, not a numpy array.
- **diff** *numpy array* - Numpy array with dimensions of (n_kinetic, n_field, moments=2, n_time). It contains the Gyrobohm-normalized diffusivities averaged over radius and summer over mode number. The moments are:
 1. D_σ/χ_{GB} (particle diffusivity)
 2. χ_σ/χ_{GB} (energy diffusivity)
- **diff_i** *numpy array* - Numpy array with dimensions of (n_kinetic, n_field, moments=2, n_x, n_time). It contains the Gyrobohm-normalized diffusivities as a function of radius, summed over mode number. The moments are:
 1. $D_\sigma(r)/\chi_{GB}$ (particle diffusivity)
 2. $\chi_\sigma(r)/\chi_{GB}$ (energy diffusivity)
- **diff_n** *numpy array* - Numpy array with dimensions of (n_kinetic, n_field, moments=2, n_n, n_time). It contains the Gyrobohm-normalized diffusivities averaged over radius for each mode number. The moments are:
 1. $D_{\sigma,n}/\chi_{GB}$ (particle diffusivity)
 2. $\chi_{\sigma,n}/\chi_{GB}$ (energy diffusivity)
- **gbflux** *numpy array* - Numpy array with dimensions of (n_kinetic, n_field, moments=4, n_time). It contains the Gyrobohm-normalized fluxes averaged over radius and summed over mode number. The moments are:
 1. $\Gamma_\sigma/\Gamma_{GB}$ (particle flux)
 2. Q_σ/Q_{GB} (energy flux)
 3. Π_σ/Π_{GB} (momentum flux)
 4. $S_{W,\sigma}^{tur}/S_{GB}$ (exchange power density)
- **gbflux_i** *numpy array* - Numpy array with dimensions of (n_kinetic, n_field, moments=4, n_x, n_time). It contains the Gyrobohm-normalized fluxes as a function of mode number and averaged over radius. The moments are:
 1. $\Gamma_\sigma(r)/\Gamma_{GB}$ (particle flux)
 2. $Q_\sigma(r)/Q_{GB}$ (energy flux)
 3. $\Pi_\sigma(r)/\Pi_{GB}$ (momentum flux)
 4. $S_{W,\sigma}^{tur}(r)/S_{GB}$ (exchange power density)
- **gbflux_n** *numpy array* - Numpy array with dimensions of (n_kinetic, n_field, moments=4, n_n, n_time). It contains the Gyrobohm-normalized fluxes as a function of mode number and averaged over radius. The moments are:
 1. $\Gamma_\sigma/\Gamma_{GB}$ (particle flux)
 2. Q_σ/Q_{GB} (energy flux)

- 3. Π_σ/Π_{GB} (momentum flux)
- 4. $S_{W,\sigma}^{tur}/S_{GB}$ (exchange power density)
- **moment_u** *numpy array* - Complex numpy array with dimensions of (n_theta_plot, n_x, i_field=n_field, n_n, n_time). It contains the potential expansion coefficients. More information is available online at https://fusion.gat.com/theory/Gyrousermanual#Computed_Quantities. The fields are described below. There can be up to three different fields:
 1. $\frac{e\delta\phi_n}{T_e}$ (electrostatic potential)
 2. $\frac{\bar{c}_s}{c} \frac{e\delta A_{\parallel n}}{T_e}$ (electromagnetic potential)
 3. $\frac{\delta B_{\parallel}}{B_{unit}(r)}$ (compressional perturbation)
- **moment_n** *numpy array* - Complex numpy array with dimensions of (n_theta_plot, n_x, n_kinetic, n_n, n_time). It contains the density moment expansion coefficients: $\frac{\delta n_{\sigma,n}}{\bar{n}_e}$. More information is available online at https://fusion.gat.com/theory/Gyrousermanual#Computed_Quantities.
- **moment_e** *numpy array* - Complex numpy array with dimensions of (n_theta_plot, n_x, n_kinetic, n_n, n_time). It contains the energy moment expansion coefficients: $\frac{\delta E_{\sigma,n}}{\bar{n}_e T_e}$. More information is available online at https://fusion.gat.com/theory/Gyrousermanual#Computed_Quantities.
- **moment_v** *numpy array* - Complex numpy array with dimensions of (n_theta_plot, n_x, n_kinetic, n_n, n_time). It contains the parallel velocity moment expansion coefficients: $\frac{\delta V_{\sigma,n}}{\bar{n}_e \bar{c}_s}$. More information is available online at https://fusion.gat.com/theory/Gyrousermanual#Computed_Quantities.
- **moment_zero** *numpy array* - Numpy array with dimensions of (n_x, n_kinetic, moments=n_moment, n_time). It contains the flux-surface average of the $n = 0$ component of the density and energy moments. The moments are described below. There can be up to two different moments:
 1. $\langle \frac{\delta n_{\sigma,0}}{\bar{n}_e} \rangle$
 2. $\langle \frac{\delta E_{\sigma,0}}{\bar{n}_e T_e} \rangle$
- **flux_velocity** *numpy array* - Numpy array with dimensions of (n_energy, n_lambda, n_kinetic, i_field=n_field, moments=2, n_n, n_time). It contains the velocity-space flux densities:

$$\Gamma = \int d\varepsilon \int d\lambda \Gamma(\varepsilon, \lambda), \quad Q = \int d\varepsilon \int d\lambda Q(\varepsilon, \lambda)$$

The moments are:

1. $\Gamma_{\sigma,n}(\varepsilon, \lambda)$ (particle flux)
2. $Q_{\sigma,n}(\varepsilon, \lambda)$ (energy flux)

The possible fields are:

1. electrostatic component
2. electromagnetic component

- **k_perp_squared** *numpy array* - Numpy array with dimensions of (n_n, n_time). It contains the flux-surface and radial average of k_{\perp}^2 :

$$\frac{\langle \langle (k_{\perp} \bar{\rho}_{s,unit})^2 |\delta\phi_n|^2 \rangle \rangle_r}{\langle \langle |\delta\phi_n|^2 \rangle \rangle_r}$$

- **Methods:**

- **__init__**(*str* **sim_directory**) - This is the class constructor. It is called when a new object of that class is created, and it reads in data from sim_directory to create that object. It sets the variable directory_name to sim_directory, and appends the pyrats folder to the python search path. It also calls the following methods:

1. `init_data()`
 2. `read_data()`
 3. `equil_time()`
- **`equil_time()`** - This method counts the number of time steps present in the currently loaded arrays, and sets them all equal (to the length of the array with the fewest number of time steps).
 - **`get_input(str input_name)`** - This method returns the specified variable from `input.gyro.gen`.
Ex: `get_input("TIME_STEP")`
 - **`init_data()`** - This method initializes object data.
 - **`make_diff()`** - This method creates `self.diff` by computing it from `self.gbflux`.
 - **`make_diff_i()`** - This method creates `self.diff_i` by computing it from `self.gbflux_i`.
 - **`make_gbflux()`** - This method creates `self.gbflux` by averaging over the radial component of `self.gbflux_i`.
 - **`plot(list x, list y, tuple dim=(1, 1))`** - This method creates a matplotlib plot of the requested data.
 - **`read_data()`** - This method reads in object data. It executes:
 1. `read_profile()`
 2. `read_geometry()`
 3. `read_t()`
 4. `read_gbflux_i()`
 5. `read_gbflux_n()`
 - **`read_file(str fname, int dSize)`** - This method reads the GYRO data file named `out.gyro.fname`. It must be given the length in number of characters of the data elements in the file as a second argument.
 - **`read_flux_velocity()`** - This method reads in `flux_velocity` data, and stores it in `self.flux_velocity`.
 - **`read_freq()`** - This method reads in frequency data and stores it in `self.freq`.
 - **`read_gbflux_i()`** - This method reads in `gbflux_i` data and stores it in `self.gbflux_i`.
 - **`read_gbflux_n()`** - This method reads in `gbflux_n` data and stores it in `self.gbflux_n`.
 - **`read_geometry()`** - This method reads in `geometry_array` data and stores it in `self.geometry`.
 - **`read_k_perp_squared()`** - This method reads in `k_perp_squared` data and stores it in `self.k_perp_squared`.
 - **`read_moment_e()`** - This method reads in `moment_e` data and stores it in `self.moment_e`.
 - **`read_moment_n()`** - This method reads in `moment_n` data and stores it in `self.moment_n`.
 - **`read_moment_u()`** - This method reads in `moment_u` data and stores it in `self.moment_u`.
 - **`read_moment_v()`** - This method reads in `moment_v` data and stores it in `self.moment_v`.
 - **`read_moment_zero()`** - This method reads in `moment_zero` data and stores it in `self.moment_zero`.
 - **`read_profile()`** - This method reads `out.gyro.profile` to get control data and stores it in `self.profile`.
 - **`read_t()`** - This method reads `t.out` to get time data and stores it in `self.t`.
 - **`set_directory(str path)`** - This method sets the variable `self.directory_name` to `path`.