CheasePy is code written in Python to run the CHEASE (Cubic Hermite Element Axisymmetric Static Equilibrium) code, which solves the Grad-Shafranov equation for toroidal MHD equilibria using pressure and current profiles and fixed plasma boundaries that is defined by a set of experimental data points (R,Z). The CheasePy code allows an iterative running of the CHEASE code either to check the preservation of MHD equilibria or converging to an experimentally defined total toroidal plasma current by modifying any input quantity.

1 Toroidal MHD Equilibrium

1.1 Grad-Shafranov Equation

The MHD equilibrium equations are given by:

\[ J \times B = \nabla p \]  
(1)

\[ \nabla \times B = J \]  
(2)

\[ \nabla \cdot B = 0 \]  
(3)

The magnetic field can be represented as:

\[ B = T \nabla + \nabla \phi \times \nabla \Psi \]  
(4)

where \( \phi \) is the ignorable toroidal angle and \( \Psi \) is the poloidal magnetic flux function.

For static MHD equilibria, the Grad-Shafranov equation is given by:

\[ \nabla \cdot \frac{1}{R^2} \nabla \Psi = \frac{j_\phi}{R} = -p'(\Psi) - \frac{1}{R^2} TT'(\Psi) \]  
(5)

where \( j_\phi \) denotes the toroidal plasma current density, \( R \) the major radius of the torus. The nature of the equilibria is determined by the two free functions \( p'(\Psi) \) and \( TT'(\Psi) \), where the pressure \( p \) and the poloidal current flux function \( T \) are functions of \( \Psi \) only.

The total current everywhere inside the plasma (\( \Psi<0 \)) is positive and is given by:

\[ I = \int j_\phi dS = \int j_\phi (J/R) d\Psi d\chi \]  
(6)

where \( J \) is the Jacobian.
1.2 Current Profiles

To achieve MHD equilibrium CHEASE can treat different current profiles such as:

- The poloidal current flux:
  \[ TT'(\Psi) \]  

- The surface averaged current density:
  \[ I^*(\Psi) = \oint j_{\phi}(J/R)d\chi = -R_0^2 C_1 C_0 p'(\Psi) - R_0^2 \frac{C_2 TT'(\Psi)}{\mu_0} \]  

- The averaged parallel current:
  \[ I_\parallel = \oint \frac{\mathbf{J} \cdot \mathbf{B}}{\mathbf{B} \cdot \nabla \phi} d\chi = R_0 \langle \mathbf{J} \cdot \mathbf{B} \rangle \]  

- The averaged parallel current density:
  \[ J_\parallel = \frac{\langle \mathbf{J} \cdot \mathbf{B} \rangle}{B_0} \]  

where,
\[
\{C_0, C_1, C_2, C_3\} = \oint \left\{ \frac{1}{R}, 1, \frac{|\nabla \Psi|^2}{R^2}, \frac{|\nabla \Psi|^2}{R^2} \right\} Jd\chi
\]
\[
\langle \mathbf{J} \cdot \mathbf{B} \rangle = -T(\Psi)p'(\Psi) - T'(\Psi)\frac{B^2}{\mu_0}
\]
\[
\langle B^2 \rangle = \oint \frac{B^2 Jd\chi}{Jd\chi}
\]
\[
\langle T/R^2 \rangle = \oint \frac{T/R^2 Jd\chi}{Jd\chi}
\]

The toroidal current density can be expressed as:
\[
\dot{j}_\phi = \frac{1}{R} \left( \frac{C_0}{C_2} I^*(\Psi) + \left( \frac{C_1}{C_2} - R^2 \right) p'(\Psi) \right)
\]
\[
\dot{j}_\phi = \frac{1}{yR} \left( I_\parallel(\Psi) + \left( \frac{C_1}{C_2} - yR^2 \right) p'(\Psi) \right)
\]
where
\[
y = 1 + \frac{1}{T^2(\Psi)} \frac{C_3}{C_2}
\]

1.3 Normalized Quantities

The normalization used here has the following format:
\[
\Psi = B_0 R_0^2
\]
\[
I = \frac{B_0 R_0}{\mu_0} \quad \text{and} \quad J = \frac{B_0}{\mu_0 R_0}
\]
\[
p = \frac{B_0^2}{\mu_0} \quad \text{and} \quad p' = \frac{B_0}{\mu_0 R_0^2}
\]
\[
T = B_0 R_0 \quad \text{and} \quad T' = \frac{1}{R_0} \quad \text{and} \quad TT' = B_0
\]
2 CHEASE [Fortran] Code

When CHEASE code runs based on experimental data points it takes as an input three files. The EXPEQ file (which is a kind of EFIT file, also known as EQDSK file) which has the experimental equilibrium quantities such as the current profile, in addition to the magnetic configuration. The second file is the EXPTNZ file which contains profiles for the ion and electron temperatures ($T_i$ $T_e$) and densities ($n_i$ $n_e$), in addition to the effective atomic number ($Z_{eff}$) as a function of the normalized poloidal axis ($\psi_N$). The CHEASE code also takes the chease_namelist file which has the initialization parameters that depend on the plasma fusion machine shot, the type of the input physical quantities, and the run-mode of the CHEASE code. For more information about CHEASE code and the different options for the input parameters in the chease_namelist file you may go to their website which has all the required resources and references.

After a successful run, the CHEASE code produces several output files, but CheasePy code uses only four of them. The EXPEQ.OUT(IN) output files have specific experimental quantities from the EFIT file depending on the selected values of the input parameters in the chease_namelist file, such as pressure ($P$), poloidal coordinate grid points ($\rho(\psi)$), parallel current density profile ($J_\parallel$), etc. EXPTNZ.OUT(IN) output file should have exactly the same profiles in the EXPTNZ input file but with a higher resolution. CHEASE code also packs several output quantities related to the coordinate systems and physical quantities into an HDF5 file called ogypopi.h5, and it should be noticed here that all the quantities in the HDF5 file have SI units in contrary to the normalized quantities in the EXPEQ.OUT(IN) files. CHEASE code also gives a text file as an output that contains all the outcomes similar to the HDF5 file named based on the iteration number to be ogypopi.dat. The CHEASE code produces also several other output files but they aren’t needed by CHEASEPY.\footnote{CheasePy renames all the required input and output files to include the iteration number at the end of each filename, e.g. CheasePy renames EXPEQ.OUT to EXPEQ.iter000.OUT after the zeroth iteration and does the same after each iteration for all IN and OUT files. However, in case of the files ogypopi.dat and ogypopi.h5, CheasePy renames them to chease.iterxxx.dat and chease.iterxxx.h5, respectively, after each iteration.}

It should be noticed here that for calculating the MHD equilibrium in the zero iteration the CHEASE code uses an EXPEQ file that may have the EFIT or EQDSK format which contains an extensive amount of information about the machine geometry, magnetic configuration, and plasma status. However it can also use a reduced format of the EQDSK file which contains only the required data for the next iteration to run properly. In contrary, the format of the EXPTNZ doesn’t change from one iteration to another, it has the same profiles but with higher resolution as mentioned above. Some parameters in the chease_namelist file need to be changed from one iteration to another, but most of them keep their values. For example, the NEQDSK parameter takes 1 when using the EFIT file in EQDSK format, or it takes 0 to create the EXPEQ from input/given parameters. We will shed more lights on the description and function of the parameters needed to be modified to properly run the CheasePy code in other sections, but it worth mentioning here that the provided chease_namelist file with the CheasePy code has only the parameters that you may need to change based on the case you run such as pressure type (nppfun), current type (nsttp), coordinates resolution (NPSI,NCHI,NS,NT,NISO). Other parameters which are not provided in the basic chease_namelist file are set to their default values by CheasePy code in the namelistcreate() function, or by the CHEASE code itself. You may also add these parameters to the chease_namelist file in case you will change them in a regular basis.
3 CheasePy [Python] Code

To run the CheasePy code you need to have the CHEASE code compiled first in your local machine as instructed in the CHEASE developer website and have the executable file named `chease_hdf5` in the same directory of the `cheasepy.py`, `runchease.py`, and `efitools.py` files. Also, you need to have Anaconda for Python 2.7/3.7 version installed in your local machine, where the `cheasepy.py` script works with both versions of Python (2.7 or 3.7).

3.1 Directory Structure

In the main directory there should be at least two files, `runchease.py` and `chease_hdf5` with the path to `cheasepy.py` and `efitools.py` files being accessible. Also, the path to the files of the desired shot can be set to the ‘shotpath’ variable in the `runchease.py` script.

The shots folder contains subfolders for each available experimental output measurements. Each shot subfolder is named upon the user choice, but it is preferred to be named after the machine/shot name, e.g. `machine.name_shot.number`, and contains at least a profile file and an equilibrium geometry file with their names formatted to have ‘the parent folder name’ then ‘underscore’ then the type of data file, e.g. `machine.name_shot.number_FILE.TYPE`:

| FILE.TYPE | Description                                                                 |
|-----------|-----------------------------------------------------------------------------|
| EQDSK     | Geometry Equilibrium and Current and Pressure Profiles (full)               |
| EXPEQ     | Geometry Equilibrium and Current and Pressure Profiles (reduced)            |
| EXPTNZ    | Electron and Ions Profiles (reduced)                                        |
| ITERDB    | Electron and Ions Profiles (reduced)                                        |
| PROFILES  | A Complete Equilibrium Profiles for Ions, Electrons, Impurities, etc. (full)|
| CHEASE    | A Complete Geometry and Profiles output from a previous Chease run (full)   |

You may add more shots to the shot folder by following the same method of naming the shot folder and the equilibrium geometry and profile files. Having a wrong name for the shot folder and/or any of its contents might cause CheasePy code doesn’t work properly.

CheasePy has the capability to take the electron profiles and ion profiles from different sources, it can also take the pressure profile from the EQDSK (EFIT) file or calculate it from any of the input profile sources. Because the profiles on each source are projected on its own grid, CheasePy allows users to unify all the provided profiles to a single grid by setting a source for the rhomesh grid.

---

5 It is recommended to set/export the $PATH to the location of the `cheasepy.py` and `efitools.py` files, and to keep the `runchease.py` file with the `chease_hdf5` executable in the same folder.

6 `cheasepy.py` is a Python package that contains all the required functions to perform all operations in the CheasePy code such as profile and geometry files creation, profile and geometry files reading, etc.

7 `runchease.py` is a Python script that allows the user of CheasePy to setup the sources and types of all input profiles and geometry, and it also allows the user to setup the mode of operation of CheasePy code.

8 `efitools.py` is a Python package that contains functions to read the eft, profile, and iterdb files, and calculate the outer most closed magnetic surface that does not have the x-point.

9 Calculating the pressure from EXPTNZ or ITERDB ignores the contribution of the fast particles ($n_b$).

10 Considering both $\rho_0 N$ and $\rho_0 \phi$ coordinates, CheasePy prefers a rhomesh grid source that has both coordinates to ease the interpolation process in different sources.
3.2 Creating chease_namelist File

The CheasePy code starts with copying the required profiles and equilibrium geometry files of the selected shot to the current directory, and then it uses the user-defined namelist parameters in runchease.py script file to create the chease_namelist file (which is required for CHEASE code to run as we indicated above) using the create_namelist(setParam={}) function in the cheasepy.py package.

There is a long list of CHEASE namelist parameters that are used to set the operation mode of CHEASE code, but working on experimental measurements helps in reducing this list to smaller number of parameters, with all other parameters to set to their default values defined by either CHEASE or CheasePy. A list of the required CHEASE parameters can be found in the runchease.py script, and they can be modified based on the provided experimental facts. The namelist parameters are stored in a Python dictionary structure called namelistVals with the namelist parameters are the dictionary keys named as defined in CHEASE code. The value of each namelist parameter can be single-valued or list-of-values depending on the number of times the input namelist parameters will change from one iteration of running CHEASE to another. For example, if CheasePy runs CHEASE for 11 times but each parameter to the namelistVals is single-valued, then the same namelist input file created in the zeroth iteration will be used in the other 10 iterations. However, if the parameters in the namelistVals are defined as a list of two values, the first value will be used for the zeroth iteration of CHEASE then the second value will be used for the other 10 iterations. Generally, CheasePy will create a new chease_namelist file based on the number of values in the namelistVals and use the latest one for the rest of iterations.

The description of the CHEASE namelist parameters available in the runchease.py script can be found in the following table:

| Parameter | Default | Description |
|-----------|---------|-------------|
| NS        | 256     | Number of tr-intervals for equilibrium calculation. |
| NT        | 256     | Number of 0-intervals for equilibrium calculation. |
| NISO      | 256     | Number of s-intervals to define \(I^*(s)\) or \(I_{||}(s)\). |
| NPSI      | 1024    | Number of radial stability-s intervals. |
| NCHI      | 1024    | Number of poloidal nodes for ballooning. |
| NRBOX     | 60      | Number of R points used to save equilibrium in EQDSK. |
| NZBOX     | 60      | Number of Z points used to save equilibrium in EQDSK. |
| RELAX     | 0       | Under-relaxation parameter used if magnetic axis converges slowly |
| NSTTTP    | 1       | Type of input current profiles: 1=\(ff'\), 2=\(I^*\), 3=\(I_{||}\), and 4=\(J_{||}\). |
| NPROPT    | 3       | Type of output current profiles: 1=\(ff'\), 2=\(I^*\), 3=\(I_{||}\), and 4=\(J_{||}\). |
| NPPFUN    | 8       | Type of input pressure profiles: 4=\(P'\) and 8=\(P\). |
| NEQDSK    | 0       | Source of equilibrium geometry: 0=EXPEQ and 1=EQDSK |
| TENSBND   | 0       | Interpolation weight around the x-point. |
| COCOS_IN  | 2       | To determine the poloidal coordinate systems from input EXPEQ. |
| TENSPROF  | 0       | Interpolation weight of the profiles. |
| COCOS_OUT | 12      | To determine the poloidal coordinate systems from output EXPEQ. |
| NRHOMESH  |         | Type of input grid: 0=\(p_{ρN}\) and 1=\(p_{θN}\). |

The TENSBND parameter is set to a negative value, e.g. -1, when running CHEASE with NEQDSK=1, i.e. using the EQDSK file as a source for the equilibrium geometry such as the (R,Z) coordinates for the magnetic boundary surface to eliminate the x-point by interpolation. This is not applicable when with NEQDSK=0 because a EXPEQ file will be constructed in a different way that is explained below.

---
8The namelistVals is passed to setParam argument in create_namelist function.
9For a full list of the CHEASE input parameters and their detailed description you may read this reference: [The CHEASE code for toroidal MHD equilibria](#) by H. Lutjens, A. Bondeson, O. Sauster
10You may read more details about how to choose the values of cocos_in and cocos_out in the [Tokamak coordinate conventions: COCOS](#) by O.Sauter and S.Yu.Medvedev. 

---
3.3 Reading Input/Output Files

There are different input/output files that can be used with CheasePy such as EQDSK, EXPEQ, EXPTNZ, PROFILES, ITERDB, and CHEASE. Each of these files organizes the data/measurements in a different way, and each input/output file has its own grid type and scale. In the following subsection we will show how a user can use CheasePy to read these input/output files, and get CheasePy to interpolate these read data/measurements on another grid specified/provided by the user.

3.3.1 Reading ‘Profiles’ File

The equilibrium profiles of different quantities measured in the fusion reactors, such as electron, ion, impurities, and fast ion densities, toroidal and poloidal speeds, etc. are provided by the sentimentalist, and are usually projected on a $\psi_N$ normalized grid and have the same equilibrium for the physical and geometrical quantities in the EFIT file.

CheasePy provides the `read_profiles(profilesfpath, setParam, **kwargs)` function to read the profiles file. This function takes the following arguments:

| Argument     | Default    | Description                                                                 |
|--------------|------------|------------------------------------------------------------------------------|
| exptnzfpath   | user-input | The path to the profile file.                                                |
| setParam     | ()         | nrhopsi=0 ($\rho_{\psi_N}$) and nrhopsi=1 ($\rho_{\phi_N}$) grid.           |
| **kwargs     | None       | Zeff=True for $Z_{eff}$ profile or Zeff=False for fixed-value $Z_{eff}$.      |

The first argument, `profilesfpath`, takes the path to the `profiles` file which **must** be provided by the user. If the second argument, `setParam`, is an empty dictionary, the `read_profiles` function returns all quantities projected on the original grid ($\psi_N$) provided in the profiles file with the original $Z_{eff}$ profile calculated using this equation:

$$Z_{eff} = \frac{\sum_n n_s Z_n^2}{n_e}$$

(19)

However, if Zeff key is set in `setParam` to False, all grid points of the $Z_{eff}$ profile is set to the average value of the effective atomic number profile calculated using the equation above.

A source for $\psi_N$ ($\rho_{\psi_N}$) and $\phi_N$ ($\rho_{\phi_N}$) should be provided in the **kwargs input argument, hence setting nrhopsi in `setParam` to 0 will return all the profiles interpolated into the new $\psi_N$ ($\rho_{\psi_N}$) grid provided in the source file, however all the profiles will be interpolated into the new $\phi_N$ counterpart if nrhopsi is set to 1, and `read_profiles` function will return only the new $\rho_{\psi_N}$, $\rho_{\phi_N}$, $\psi_N$, and $\phi_N$ grids to the calling function/script.

To provide a source for the grid points, we add the type of that source and a path to the source file to the **kwargs input in the `read_profiles` function call. Because the user may request to interpolate the profiles onto $\psi_N$ ($\rho_{\psi_N}$) and $\phi_N$ ($\rho_{\phi_N}$) grids the source must provide both grids point, otherwise CheasePy will **raise** an error in case the interpolation onto the $\phi_N$ ($\rho_{\phi_N}$) grid is requested but the input source doesn’t provide both grid points to implement that interpolation.[11] Therefore, the type of the sources allowed for interpolating onto a grid of different type, e.g. $\phi_N$ ($\rho_{\phi_N}$), are limited to CHEASE, EQDSK, and IMPORTED[12] The last source allows you to import $\psi_N$ ($\rho_{\psi_N}$) and $\phi_N$ ($\rho_{\phi_N}$) directly from any external source, and they will override any other grid provided in CheasePy[13]

[11]If the user wants to interpolate the profiles onto a new grid of the same type, the source may have only the new grid points, e.g. $\psi_N$ ($\rho_{\psi_N}$), and provided it to the **kwargs argument in `read_profiles` function.

[12]Limiting the sources for the grid to these sources is because they have the q-profile which make it possible to convert from $\psi_N$ ($\rho_{\psi_N}$) to $\phi_N$ ($\rho_{\phi_N}$) using this formula: $\phi = \int \psi dq$

[13]The imported values is explained more in the description of running CheasePy and setting its mode of operation in the `runchease.py` file.
The `read_profiles` function returns all the profiles encapsulated in a dictionary in SI units. The name (=key) and description of the returned profiles are listed in the following table:

| Measurement | Description            |
|-------------|------------------------|
| Te          | Electron Temperature   |
| Ti          | Ion Temperature        |
| Tb          | Fast Ions Temperature  |
| ne          | Electron Density       |
| ni          | Ion Density            |
| nb          | Fast Ions Density      |
| nz          | Impurities Density     |
| Pb          | Fast Ions Pressure     |
| Zeff        | Effective Atomic Number|
| Vtor        | Toroidal Velocity      |
| Vpol        | Poloidal Velocity      |
| PSIN        | $\psi_{N}$            |
| PHIN        | $\phi_{N}$ (with grid source) |
| rhopsi      | $\rho_{\psi_{N}}$     |
| rhotor      | $\rho_{\phi_{N}}$ (with grid source) |
| ppprime     | Total Pressure Gradient|
| pressure    | Total Pressure of All Species |

### 3.3.2 Reading ‘EXPTNZ’ File

The `EXPTNZ` file is an INPUT/OUTPUT file for `CHEASE` code to provide the density and temperature profiles of the ions and electrons in SI units. The first line of the `EXPTNZ` file shows the number of grid points, grid type, i.e. rhopsi ($\rho_{\psi_{N}}$) or rhotor ($\rho_{\phi_{N}}$), and the order of the given profiles, such as electron temperature ($T_e$), electron density ($n_e$), effective atomic number ($Z_{eff}$), ion temperature ($T_i$), and ion density ($n_i$), in the file. These profiles are organized in a single column with each of them has a length equals the given number of grid points. The header of a `EXPTNZ` file looks like:

512 rhopsi, Te, ne, Zeff, Ti, ni profiles

`CheasePy` provides the `read_exptnz(exptnzfpath, setParam, **kwargs)` function to read the `EXPTNZ` file. This function takes the following arguments:

| Argument   | Default       | Description                                                                 |
|------------|---------------|------------------------------------------------------------------------------|
| exptnzpath | user-input    | The path to the `EXPTNZ` file.                                               |
| setParam   | {}            | Select between nrhopsi=0 ($\rho_{\psi_{N}}$) or nrhopsi=1 ($\rho_{\phi_{N}}$) grid. |
| **kwargs   | None          | Choose a source for the $\rho_{\psi_{N}}$ and $\rho_{\phi_{N}}$ grid to interpolate on. |

The first argument, `exptnzfpath`, takes the path to the `EXPTNZ` file which must be provided by the user. If the second argument, `setParam`, is an empty dictionary, the `read_exptnz` function returns all quantities projected on the original grid ($\rho_{\psi_{N}}$) provided in the `EXPTNZ` file, otherwise, a source for $\rho_{\psi_{N}}$ and $\rho_{\phi_{N}}$ should be provided in the `**kwargs` input argument, hence setting `nrhopsi` in `setParam` to 0 will return all the profiles interpolated into the new $\rho_{\psi_{N}}$ grid provided in the source file, however all the profiles will be interpolated into the new $\rho_{\phi_{N}}$ grid provided in the source file, and the calling function/script and ignore the original grids provided in the `EXPTNZ` file. The source of new grids are passed to the `**kwargs` input argument of the `read_exptnz` function as described in section 3.3.1.

In addition to the input profiles, `read_exptnz` function calculates and returns the density and temperature of the impurities and the total pressure. The total pressure calculated in the `read_exptnz` doesn’t include the contribution of the fast ions (if any), which might make the pressure calculated from the `EXPTNZ` file is different from the actual pressure measured in the experiment\(^\text{14}\).

\(^\text{14}\)The actual total pressure measured in the experiment is found in the `profiles` and `EQDSK` files.
The `read.exptnz` function returns all the profiles encapsulated in a dictionary in SI units. The name (=key) and description of the returned profiles are listed in the following table:

| Measurement | Description         |
|-------------|---------------------|
| Te          | Electron Temperature|
| Ti          | Ion Temperature     |
| ne          | Electron Density    |
| ni          | Ion Density         |
| nz          | Impurities Density  |
| Zeff        | Effective Atomic Number |
| rhopsi      | $\rho_{\psi N}$   |
| rhotor      | $\rho_{\phi N}$ (with grid source) |
| pprime      | Total Pressure Gradient |
| pressure    | Total Pressure of All Species |

3.3.3 Reading ‘ITERDB’ File

The `ITERDB` file provides the density and temperature profiles for electrons, ions, and impurities, in addition to the toroidal velocity defined on a `rhotor` grid (i.e. $\rho_{\phi N}$) and SI units.

`CheasePy` provides the `read.iterdb(iterdbfpath, setParam, **kwargs)` function to read the `ITERDB` file. This function takes the following arguments:

| Argument     | Default     | Description                                                                 |
|--------------|-------------|------------------------------------------------------------------------------|
| iterdbfpath  | user-input  | The path to the `ITERDB` file.                                               |
| setParam     | {}          | Select between `nrhopsi=0` ($\rho_{\psi N}$) and `nrhopsi=1` ($\rho_{\phi N}$) grid. |
| **kwargs     | None        | Choose a source for the $\rho_{\psi N}$ and $\rho_{\phi N}$ grid to interpolate on. |

The first argument, `iterdbfpath`, takes the path to the `ITERDB` file which must be provided by the user. If the second argument, `setParam`, is an empty dictionary, the `read.iterdb` function returns all quantities projected on the original grid ($\rho_{\phi N}$) provided in the `EXPTNZ` file, otherwise, a source for $\rho_{\psi N}$ and $\rho_{\phi N}$ should be provided in the **kwargs input argument, hence setting `nrhopsi` in `setParam` to 0 will return all the profiles interpolated into the new $\rho_{\psi N}$ grid provided in the source file, however all the profiles will be interpolated into the new $\rho_{\phi N}$ counterpart if `nrhopsi` is set to 1, and `read.iterdb` function will return only the new $\rho_{\psi N}$ and $\rho_{\phi N}$ grids to the calling function/script and ignore the original grids provided in the `ITERDB` file. The source of new grids are passed to the **kwargs input argument of the `read.iterdb` function as described in section 3.3.1.

In addition to the input profiles, `read.iterdb` function calculates and returns the total pressure. The total pressure calculated in the `read.iterdb` doesn’t include the contribution of the fast ions (if any).

The `read.iterdb` function returns all the profiles encapsulated in a dictionary in SI units. The name (=key) and description of the returned profiles are listed in the following table:

| Measurement | Description         |
|-------------|---------------------|
| Te          | Electron Temperature|
| Ti          | Ion Temperature     |
| ne          | Electron Density    |
| ni          | Ion Density         |
| nz          | Impurities Density  |
| Zeff        | Effective Atomic Number |
| Vtor        | Toroidal Velocity   |
| rhopsi      | $\rho_{\psi N}$ (with grid source) |
| rhotor      | $\rho_{\phi N}$    |
| pprime      | Total Pressure Gradient |
| pressure    | Total Pressure of All Species |
3.3.4 Reading ‘EXPEQ’ File

The EXPEQ file is an INPUT/OUTPUT file for CHEASE that provides the equilibrium pressure and current profiles on a specific grid. The EXPEQ file also provides the equilibrium geometry, such as the (r,z) components of the last closed magnetic surface. All profiles and geometries provided in EXPEQ file are ‘normalized’ using the equations in section(1.3).

Because of the variety of profiles and grid types that can be used in the EXPEQ file, each profile and grid type is assigned a numeric value to refer to it. These numbers are written in the two lines after the list of (r,z) coordinates in the EXPEQ file. The first number in the first line specifies the number of grid points of the \( \rho \psi_N \) or \( \rho \phi_N \) coordinates. The second number in the first line is ‘NPPFUN’ and it takes 8 for pressure or 4 for pressure gradient depending on which one is included in the EXPEQ file. The first number in the second line is ‘NSTTP’ and it takes 1 for the current flux density (\( f'f' \)), 2 for surface current (\( I^* \)), 3 for parallel current (\( I || \)), or 4 for parallel current density (\( J || \)). The second number in the second line is ‘NRHOMESH’ and it takes 0 for rhopsi (\( \rho \psi_N \)) or 1 for rhotor (\( \rho \phi_N \)).

CheasePy provides the `read_expeq(expeqfpath, setParam, **kwargs)` function to read the EXPEQ file. This function takes the following arguments:

| Argument  | Default | Description |
|-----------|---------|-------------|
| expeqfpath | user-input | The path to the EXPEQ file. |
| setParam   | {}      | Select between nrhopsi=0 (\( \rho \psi_N \)) and nrhopsi=1 (\( \rho \phi_N \)) grid. |
| **kwargs   | None    | Choose a source for the \( \rho \psi_N \) and \( \rho \phi_N \) grid to interpolate on. |

The first argument, expeqfpath, takes the path to the EXPEQ file which must be provided by the user. If the second argument, setParam, is an empty dictionary, the read_expeq function returns all quantities projected on the original grid (\( \rho \psi_N \) or \( \rho \phi_N \)) provided in the EXPEQ file, otherwise, a source for \( \rho \psi_N \) and \( \rho \phi_N \) should be provided in the **kwargs input argument, hence setting nrhopsi in setParam to 0 will return all the profiles interpolated into the new \( \rho \psi_N \) grid provided in the source file, however all the profiles will be interpolated into the new \( \rho \phi_N \) counterpart if nrhopsi is set to 1, and read_expeq function will return only the new \( \rho \phi_N \) and \( \rho \phi_N \) grids to the calling function/script and ignore the original grids provided in the EXPEQ file. The source of new grids are passed to the **kwargs input argument of the read_expeq function as described in section(3.3.1).

The read_expeq function returns all the ‘normalized’ pressure profiles and equilibrium geometry encapsulated in a dictionary. The name (=key) and description of the returned profiles are listed in the following table:

| Measurement | Description |
|-------------|-------------|
| epsilon     | Inverse aspect ratio |
| zgeom       | Geometric mean of z-mesh |
| pedge       | Pressure at the Edge |
| nRZmesh     | Size of (r,z) mesh |
| nrhomesh    | Size of \( \rho \psi_N \) or \( \rho \phi_N \) grid |
| nrhotype    | 0 = \( \rho \psi_N \) or 1 = \( \rho \phi_N \) |
| nsttp       | 1 = \( f'f' \), 2 = \( I^* \), 3 = \( I || \), and 4 = \( J || \) |
| nppfun      | 8 = \( P \) or 4 = \( P' \) |
| rhopsi      | \( \rho \psi_N \) (if nrhotype = 0) |
| rhotor      | \( \rho \phi_N \) (if nrhotype = 1) |
| pprime      | Total Pressure Gradient (if nppfun = 4) |
| pressure    | Total Pressure of All Species (if nppfun = 8) |
| fprime      | current flux density (if nsttp = 1) |
| lstr        | surface current (if nsttp = 2) |
| lprl        | parallel current (if nsttp = 3) |
| jprl        | parallel current density (if nsttp = 4) |

---

This magnetic surface should not include any singular point, i.e. x-point, not to cause a problem in finding a state of MHD equilibrium in CHEASE code, or an incomplete simulation with an exit error.
It worth mention that if the value of ‘NPROPT’ parameter in the chease.namelist file is set to a different value of the ‘NSTTP’, CHEASE uses the current profile that corresponds to the ‘NPROPT’ number in the output EXPEQ file instead of the one corresponds to the ‘NSTTP’. In addition, the sign of the ‘NPROPT’ number is flipped to negative if ‘NPPFUN’ is set to 8 and kept positive otherwise. A different value for ‘NPROPT’ from its ‘NSTTP’ counterpart is needed to run CHEASE in an iterative mode with a different input current profile. Hence, we make CHEASE to output the desired current profile in the output EXPEQ file by setting ‘NPROPT’ to the number corresponding that current profile.

3.3.5 Reading ‘EQDSK’ File

The EQDSK file is an INPUT/OUTPUT file for CHEASE that provides the equilibrium pressure, such as the pressure \( P(\psi) \) and pressure gradient \( dP/d\psi \), and equilibrium current, such as the current flux \( ff' = fdf/d\psi \). It also provides the equilibrium geometry, such as the \((r,z)\) components of the last closed magnetic surface, the safety factor \( q \) profile, the poloidal mesh \( \psi(R,Z) \), and enough information to construct the \( \psi_N, \phi_N, \rho_{\psi N} \) and \( \rho_{\phi N} \) grids. All equilibrium profiles and geometries provided in the EQDSK file are given in the SI units.

CheasePy provides the `read_eqdsk(eqdskfpath, setParam, Normalized, **kwargs)` function to read the EQDSK file. This function takes the following arguments:

| Argument      | Default  | Description                                      |
|---------------|----------|--------------------------------------------------|
| eqdskfpath    | user-input | The path to the EQDSK file.                     |
| setParam      | {}       | Select between nrhopsi=0 \( \rho_{\psi N} \) and nrhopsi=1 \( \rho_{\phi N} \) grid. |
| Normalized    | False    | Return quantities in SI units (False) or Normalized (True). |
| **kwargs      | None     | Choose a source for the \( \rho_{\psi N} \) and \( \rho_{\phi N} \) grid to interpolate on. |

The first argument, `eqdskfpath`, takes the path to the EQDSK file which must be provided by the user. If the second argument, `setParam`, is an empty dictionary, the `read_eqdsk` function returns all quantities projected on the original \( \phi_N \) grid provided in the EQDSK file, otherwise, a source for \( \psi_N \) and \( \phi_N \) should be provided in the **kwargs input argument, hence setting `nrhopsi` in `setParam` to 0 will return all the profiles interpolated into the new \( \psi_N \) grid provided in the source file, however all the profiles will be interpolated into the new \( \phi_N \) counterpart if `nrhopsi` is set to 1, and `read_eqdsk` function will return only the new \( \psi_N, \phi_N, \rho_{\psi N}, \) and \( \rho_{\phi N} \) grids to the calling function/script and ignore the original grids provided in the EQDSK file. The source of new grids are passed to the **kwargs input argument of the `read_eqdsk` function as described in section 3.3.1.

The `read_eqdsk` function returns all the profiles and equilibrium geometry encapsulated in a dictionary. The name (=key) and description of the returned profiles are listed in the following table.\(^{16}\)

| Measurement | Description                   |
|-------------|-------------------------------|
| q           | Safety Factor Profile         |
| BCTR        | Toroidal B at Magnetic Axis   |
| RCTR        | R-Coordinate at Magnetic Axis |
| CURNT       | Total Experimental Toroidal Current |
| PSIN        | \( \psi_N \)                  |
| PHIN        | \( \phi_N \) (with grid source) |
| rhopsi      | \( \rho_{\psi N} \)          |
| rhostor     | \( \rho_{\phi N} \) (with grid source) |
| rbound      | r-coordinate of magnetic boundary surface |
| zbound      | z-coordinate of magnetic boundary surface |
| pprime      | Total Pressure Gradient \( (P'(\psi)) \) |
| pressure    | Total Pressure of All Species \( (P(\psi)) \) |
| ffprime     | current flux function \( (ff'(\psi)) \) |

\(^{16}\)Due to the long list of parameters returned from the `read_eqdsk` function after reading the EQDSK file, we ONLY listed here those are used frequently by CheasePy code. The `read_vfit_file` function has a full list of all parameters that are read from the EQDSK file. This [link](#) has a full description of the EQDSK parameters.
3.3.6 Reading ‘CHEASE’ File

The CHEASE file is an OUTPUT file for CHEASE that provides the density and temperature for electrons, ions, and impurities, equilibrium pressure and currents, equilibrium geometry, safety factor (q) profile, and poloidal and toroidal grids. All equilibrium profiles and geometries provided in the CHEASE file are given in the SI units.

CheasePy provides the read_chease(cheasefpath, setParam, Normalized, **kwargs) function to read the CHEASE file. This function takes the following arguments:

| Argument     | Default | Description                                                                 |
|--------------|---------|-----------------------------------------------------------------------------|
| cheasefpath  | user-input | The path to the CHEASE file.                                                |
| setParam     | {}      | Select between nrhopsi=0 (ρψN) and nrhopsi=1 (ρφN) grid.                   |
| Normalized   | False   | Return quantities in SI units (False) or Normalized (True).                 |
| **kwargs     | None    | Choose a source for the ρψN and ρφN grid to interpolate on.                |

The first argument, cheasefpath, takes the path to the CHEASE file which must be provided by the user. If the second argument, setParam, is an empty dictionary, the read_chease function returns all quantities projected on the original φN grid provided in the CHEASE file, otherwise, a source for ψN and φN should be provided in the **kwargs input argument, hence setting nrhopsi in setParam to 0 will return all the profiles interpolated into the new ψN grid provided in the source file, however all the profiles will be interpolated into the new φN counterpart if nrhopsi is set to 1, and read_chease function will return only the new ψN, φN, ρψN, and ρφN grids to the calling function/script and ignore the original grids provided in the CHEASE file. The source of new grids are passed to the **kwargs input argument of the read_chease function as described in section (3.3.1).

The read_chease function returns all the profiles and equilibrium geometry encapsulated in a dictionary. The name (=key) and description of the returned profiles are listed in the following table:

| Measurement       | Description                              |
|-------------------|------------------------------------------|
| q                 | Safety Factor Profile                    |
| B0EXP             | Toroidal B at Magnetic Axis              |
| R0EXP             | R-Coordinate at Magnetic Axis            |
| ITEXP             | Total Experimental Toroidal Current      |
| PSIN              | ψN                                        |
| PHIN              | φN (with grid source)                    |
| rhopsi            | ρψN                                      |
| rtor              | ρφN (with grid source)                   |
| rbound            | r-coordinate of magnetic boundary surface |
| zbound            | z-coordinate of magnetic boundary surface |
| shear             | Magnetic Shear (s(ψ))                    |
| signeo            | Neoclassical Conductivity (σneo(ψ))      |
| Te (Ti)           | Electron (Ion) Temperature (T_e(ψ) T_i(ψ)) |
| ne (ni,nz)        | Electron (Ion,impurities) Density (n_e(ψ) (n_i(ψ),n_z(ψ))) |
| pprime            | Total Pressure Gradient (P'(ψ))          |
| pressure          | Total Pressure of All Species (P(ψ))     |
| fprime            | current flux function (f'(ψ))            |
| Ibs               | Bootstrap Current (Ibs(ψ))               |
| Jbs               | Bootstrap Current Density (Jbs(ψ))       |
| Jprl              | Parallel Current (I_{prl}(ψ))           |
| Jprl              | Parallel Current Density (J_{prl}(ψ))    |
| Istr              | Surface Current (I^{s}(ψ))               |
| Itor              | Toroidal Current (I_{tor}(ψ))            |
| Jtor              | Toroidal Current Density (J_{tor}(ψ))    |

17Due to the long list of parameters returned from the read_chease function after reading the CHEASE file, we ONLY listed here those are frequently used by CheasePy code. The read_chease function has a full list of all parameters that are read from the CHEASE file. This paper has a full description of the CHEASE parameters.
There are also a complete list of the magnetic coefficients \((g_{ij})\), Jacobian transformation matrix \((J)\), and other parameters that can be of interested to the users.
3.4 Creating Input Files

Before any iteration of CheasePy the write_exptnz and write_expeq functions are called to create the EXPTNZ and EXPEQ files, respectively, which are used as input files to CHEASE code. However, if the namelist parameter NEQDSK is set to zero, only the write_exptnz function is called and the EQDSK file is used as input to CHEASE. The user can specify the sources of each profile to be used to create the EXPTNZ and EXPEQ files in the setParam input argument to the write_exptnz and write_expeq functions, as we explained in sections 3.4.2 and 3.4.1. After every iteration, the output files are renamed by adding a suffix _iterxxx to reflect the iteration number, for instance chease_iter000.h5, EXPEQ_iter000.OUT, and EXPTNZ_iter000.OUT for the zeroth iteration.

3.4.1 Creating EXPEQ File

The EXPEQ file has the (r,z) coordinates of the LCMS, in addition to the pressure and current profiles. CheasePy provides the write_expeq(setParam,outfile,**kwargs) function to create the EXPEQ file and/or return all the contents of that file as a dictionary structure with field names that follow the description in section 3.3.4. This function takes the following arguments:

| Argument | Default | Description |
|----------|---------|-------------|
| setParam | {}      | Specify the sources for different profiles. |
| outfile  | True    | Create an output EXPEQ file. |
| **kwargs | None    | Specify the path to the source files. |

The first argument, setParam, has a dictionary structure to specify the types and sources of the grid (nrhomesh), the pressure profile (nppfun), and current profile (nsttp), each of these inputs takes a list that has the structure of [type,source]. There are three sources accepted in setParam for nrhomesh: {0,’chease’}, {1,’eqdks’}, or {7,’imported’}, six sources for nppfun: {0,’chease’}, {1,’eqdks’}, {2,’expeq’}, {3,’exptnz’}, {4,’profiles’}, {5,’iterdb’}, or {7,’imported’}, and four sources for nsttp: {0,’chease’}, {1,’eqdks’}, {2,’expeq’}, or {7,’imported’}. The path to the source files, i.e. ‘chease’, ‘eqdsk’, ‘expeq’, ‘exptnz’, ‘profiles’, and ‘iterdb’, must be provided in the **kwargs argument. However, if the {7,’imported’} sources is chosen, a dictionary of the profiles ([name, data}) must be provided in the **kwargs argument, e.g. {'pressure',list of pressure data}. All the returned profiles are projected (interpolated) into the grid provided in the nrhomesh source by setting the grid source in the setParam in the read function to the common grid source.

On the other hand, the type of the grid (nrhomesh) is either {0,’rhopsi’} or {1,’rhotot’}, the type of the pressure profile (nppfun) is either {4,’pprime’} or {8,’pressure’}, and the type of the current profile (nsttp) is either {1,’fprime’}, {2,’istr’}, {3,’jprime’}, or {4,’jprime’}. The CheasePy user has to ensure that the source file has the requested profile. For example, the EQDSK file has a single current profile, i.e. fprime and both pressure profiles, i.e. pressure and pprime. The current profile in the EXPEQ file depends on the value of NPROPT parameter in the input chease_namelist file. However, the CHEASE source has all types of profiles. Finally, the user should remember that all quantities in the EXPEQ source are normalized, which is not the case in other sources.

If the setParam is empty dictionary, the types are set to their default values, i.e. nrhomesh = 0, nppfun = 8, and nsttp = 1. Hence, any provided source in the **kwargs will be used to create the new EXPEQ file, otherwise, CheasePy will search for the EQDSK and use it as a source for the profiles. In a similar way, the source for the (r,z) coordinates of the LCMS is taken to be the same source for the nrhomesh grid.

The second input argument (outfile) takes True if the user wants to create the EXPEQ file and receive a return dictionary with the content of the EXPEQ file, or False if the user wants only that returned dictionary without creating the EXPEQ file.

18 CheasePy renames EQDSK file to EXPEQ to be recognized by CHEASE code.
3.4.2 Creating EXPTNZ File

The EXPTNZ file contains the density and temperature profiles for electrons and ions, and the effective atomic number. CheasePy provides the `write_exptnz(setParam, outfile, **kwargs)` function to create the EXPTNZ file and/or return all the contents of that file as a dictionary structure with field names that follow the description in section 3.3.2. This function takes the following arguments:

| Argument | Default | Description                      |
|----------|---------|----------------------------------|
| setParam | {}      | Specify the sources for different profiles. |
| outfile  | True    | Create an output EXPEQ file.     |
| **kwargs | None    | Specify the path to the source files. |

The first argument, `setParam`, has a dictionary structure to specify the types and sources of the grid (`nrhomesh`), the electron density and temperature profiles (`eprofiles`), and ion density and temperature profiles (`iprofiles`). There are three sources accepted in `setParam` for `nrhomesh`: {0,‘chease’}, {1,‘eqdks’}, or {7,‘imported’}, four sources for `eprofiles` and `iprofiles`: {0,‘chease’}, {3,‘exptnz’}, {4,‘profiles’}, {5,‘iterdb’}, and/or {7,‘imported’}. The path to the source files, i.e. ‘chease’, ‘eqdsk’, ‘exptnz’, ‘profiles’, and ‘iterdb’, must be provided in the `**kwargs` argument. However, if the {7,‘imported’} sources is chosen, a dictionary of the profiles ({name,data}) must be provided in the `**kwargs` argument, e.g. `{‘pressure’,[list of pressure data]}`. All the returned profiles are projected (interpolated) into the grid provided in the `nrhomesh` source by setting the grid source in the `setParam` in the read function to the common grid source. On the other hand, the type of the grid (`nrhomesh`) is either {0,‘rhopsi’} or {1,‘rhopsi’}.

If the `setParam` is empty dictionary, the type of the grid is set to its default value, i.e. `nrhomesh` = 0. Hence, any provided source will be used to create the EXPTNZ file, otherwise, the EXPTNZ will be the source of the profiles and grid. Finally, the user should remember that all quantities in the EXPTNZ source are in SI units.

The second input argument (outfile) takes `True` if the user wants to create the EXPTNZ file and receive a return dictionary with the content of the EXPTNZ file, or `False` if the user wants only that returned dictionary without creating the EXPTNZ file.
3.5 Running CheasePy

A user can run CheasePy code in a directory that has CHEASE code executable named chease_hdf5 and runchease.py by typing this command in a terminal window:

```
python runchease.py
```

This command calls the cheasepy function to prepare the required input files and run CHEASE code. The cheasepy function takes four input arguments, i.e. srcVals, namelistVals, cheaseVals, and importedVals, that set the sources of density, temperature, and pressure profiles, geometry parameters, the list of CHEASE namelist parameters, and the CheasePy mode of operation. Details for these input arguments are described below.

3.5.1 Profiles and Geometry Sources

In the runchease.py script a dictionary variable named srcVals is initialized, and then the sources are specified for each type. The temperature and density profiles for the ions, electrons, and impurities can be taken from different sources, such as profiles, exptnz, iterdb, or chease files, with the source for electrons and ions+impurities profiles can be different. The density and temperature of electrons, ions, and impurities (if any) taken from any combinations of the previous sources can also be used to calculate the total pressure. Otherwise, the total pressure, the current, and equilibrium geometry are taken directly from eqdsk, expeq, or chease files.

With these different sources, each source is identified by its name or a number as shown in the table below:

| Number | Source       | 0 | 1 | 2 | 3 | 4 | 5 | 7 |
|--------|--------------|---|---|---|---|---|---|---|
|        | chase        | 0 |   |   |   |   |   |   |
|        | eqdsk        | 1 |   |   |   |   |   |   |
|        | expeq        | 2 |   |   |   |   |   |   |
|        | exptnz       | 3 |   |   |   |   |   |   |
|        | profiles     | 4 |   |   |   |   |   |   |
|        | iterdb       | 5 |   |   |   |   |   |   |
|        | imported     | 7 |   |   |   |   |   |   |

Example 3.1. If the sources for electron profile, ion profile, pressure, and current are exptnz, profiles, eqdsk, and expeq files, respectively. Also, all the profiles will be interpolated on the $\rho_\phi N$ defined in eqdsk file. Hence, srcVals variable will be defined as follow:

```python
srcVals = {}
srcVals['iprofiles_src'] = 'profiles'
srcVals['eprofiles_src'] = 'exptnz'
srcVals['pressure_src'] = 'eqdsk'
srcVals['current_src'] = 'expeq'
srcVals['rhomesh_src'] = 'eqdsk'
```

Example 3.2. If the sources for electron and ion profiles are taken from the profiles file, however, the pressure and current profiles are taken from the eqdsk file. The $\rho_\phi N$ grid is taken from the common source of each related profile, i.e. the $\rho_\phi N$ grid used in the generated exptnz file is the same as that in the profiles file, and the $\rho_\phi N$ grid used in the generated expeq is the same as that in the eqdsk file. Hence, srcVals variable will be defined as follow:

```python
srcVals = {}
srcVals['iprofiles_src'] = 'profiles'
srcVals['eprofiles_src'] = 'profiles'
srcVals['pressure_src'] = 'eqdsk'
srcVals['current_src'] = 'eqdsk'
srcVals['rhomesh_src'] = None
```

In addition to the profiles and grid sources defined in the srcVals variable, there is another variable also defined in the srcVals which is not a source of any profile but a method of how to treat the last closed magnetic surface (LCMS). This variable is called boundary_type, and it takes 0 or ‘asis’
in case the LCMS will be written as it is read from the source file without any massaging or processing of its coordinates. However, if boundary.type receives 1 or 'interp' the coordinates taken from the source file will be interpolated to have the LCMS defined at a specific length of \( \rho_N \). This interpolation is needed when the LCMS has an x-point (i.e. singular point) which makes CHEASE either crashes or gives irrelevant/incorrect results in case the TENSBNDF parameter wasn’t set to the right value, i.e. -1.\(^{19}\)

The interpolation function magsurf_solvflines is defined in the efittools.py python package, and it returns the (r,z) coordinates of the LCMS after solving the magnetic field lines equations using a fifth-order Runge-Kutta method. The magsurf_solvflines function receives the path to the eqdsk file in the eqdskfpath argument, a cutoff \( \psi_N \) value for the LCMS in the psi argument. When the NEQDSK in chease namelist parameter is set to 0, the function magsurf_solvflines is called with psi 0.999.\(^{20}\)

In the table of profile and grid sources above we referred to a source for the profiles called imported. This source type is useful when CheasePy user wants to import a profile for any quantity from an external source, or the user wants to do some processing on any profile’s data before include it as an input to CHEASE code. All imported profiles which are passed to cheasepy function through the importedVals argument must be accompanied with the \( \rho_N \) and \( \rho_{\phi_N} \) grids. Passing the imported profiles along with their corresponding grid is required because these grids are used for interpolating those profiles onto the common grid on each input file to CHEASE code, i.e. exptnz and expeq. This means if the user has profiles from several external sources, all of these profiles have to be projected onto the same grids before passing them to the cheasepy function.

Example 3.3. A file, externalSource.dat, has four columns for \( \rho_N \), \( \psi_N \), \( n_e \), and \( T_e \) profiles, and a user need to use these profiles as input to CHEASE. The steps the user has to follow to use these profiles from that external source to CHEASE are as follow:

```python
import numpy as npy
eprofiles = npy.loadtxt(externalSource.dat)

importedVals = {}
importedVals['rhotor'] = npy.array(eprofiles[:,0])
importedVals['rho psi'] = npy.array(eprofiles[:,1])
importedVals['Te'] = npy.array(eprofiles[:,2])*1.0e3
importedVals['ne'] = npy.array(eprofiles[:,3])*1.0e19

srcVals = {}
srcVals['iprofiles_src'] = 'exptnz'
srcVals['eprofiles_src'] = 'imported'
srcVals['pressure_src'] = 'eqdsk'
srcVals['current_src'] = 'expeq'
srcVals['rhomesh_src'] = 'eqdsk'
```

In example 3.3, the electron density and temperature profiles, and the \( \rho_N \) and \( \rho_{\phi_N} \) grids are assigned to different keys in the importedVals dictionary. Then, the 'eprofiles_src' item in the srcVals dictionary is set to 'imported'. Hence, all the electron profiles will be taken from the 'imported' profiles. However, if the user wants to take only the electron temperature from the imported profiles.

\(^{19}\)Although TENSBNDF is a very good tool provided by CHEASE to interpolate around the x-point, the code user doesn’t have any control on how close the new LCMS will be close to the original one. However, using the 'interp' attribute with boundary.type allows the user to set a cutoff value for \( \rho_N \) that defines the LCMS.

\(^{20}\)The value of psi in the call to the magsurf_solvflines can be increased or decreased by CheasePy user. However, with a psi value that is very close to 1.0 the magnetic field lines solver might return (r,z) coordinates of an open field surface (outside the separatrix), which causes CHEASE code to crash. So the CheasePy user has to verify that a closed surface is returned from magsurf_solvflines function by using a suitable psi value.
external source and take the electron density from the profiles file, the user can import only the electron temperature from the external source, and keep the 'eprofiles_src' item in the srcVals assigned to 'profiles', as shown in example 3.4 below.

Example 3.4. A user wants to import only the temperature profile from an external source and take the electron density and ion profiles from the profiles file.

```python
import numpy as npy
eprofiles = npy.loadtxt(externalSource.dat)

importedVals = {}
importedVals['rhotor'] = npy.array(eprofiles[:,0])
importedVals['rho psi'] = npy.array(eprofiles[:,1])
importedVals['Te'] = npy.array(eprofiles[:,2])*1.0e3

srcVals = {}
srcVals['iprofiles_src'] = 'profiles'
srcVals['eprofiles_src'] = 'profiles'
srcVals['pressure_src'] = 'eqdsk'
srcVals['current_src'] = 'expeq'
srcVals['rhomesh_src'] = 'eqdsk'
```

The question that might be arisen now is: How will CheasePy use the 'imported' electron temperature if the user didn’t pass it to the srcVals dictionary? The answer to this question is simple: When importedVals dictionary is passed to cheasepy function as an input argument, it, in turn, passes it to write.exptnz and write.expeq functions. When any of these functions receives the content of importedVals dictionary, they inspect that content and enforce any imported profile to override the corresponding profile provided in the srcVals dictionary. Therefore, the 'imported' electron temperature profile provided in importedVals overrides the same profile provided in srcVals. This takes place for any profile used to construct EXPTNZ and EXPEQ files.

As we explained earlier, the pressure profile is compromised of (at least) two components from ions and electrons, where \( P_T = n_e T_e + n_i T_i \). In addition, there might be contributions from the impurities and fast ions with the total pressure given by: \( P_T = n_e T_e + n_i T_i + n_z T_z + n_b T_b \), where \( n_b \) and \( T_b \) are the density and temperature of fast ions. A common mistake is done in examples 3.3 and 3.4 when the pressure profile is taken directly from the EQDSK file without considering the changes in one of its components which is the electron density and/or temperature profiles. Therefore, the total pressure has to be recalculated using the contribution of the imported profiles of the electron density and/or temperature. The following example explains how to calculate the new total pressure using the new electron profile.

Example 3.5. A user has two iterdb files, basecase.iterdb which has the original profiles of the electrons, ions, and impurities, and modTeprof.iterdb which has a modified electron temperature (\( T_e \)) profile. The following code shows how to include the contribution of the modified electron temperature into the pressure profile.

```python
import cheasepy

inParam = {nrhomesh:0}

baseITERDB = cheasepy.read_iterdb(iterdbfpath='basecase.iterdb',
                                 setParam=inParam, eqdsk='EQDSK')
modITERDB = cheasepy.read_iterdb(iterdbfpath='modTeProf.iterdb',
                                setParam=inParam, eqdsk='EQDSK')
```
```python
importedVals = {}
importedVals['rhotor'] = modITERDB['rhotor']
importedVals['rhopsi'] = modITERDB['rhopsi']
importedVals['Te'] = modITERDB['Te']

eqdskdata = cheasepy.read_eqdsk(eqskfpath='EQDSK')
importedVals['pressure'] = eqdskdata['pressure'] - baseITERDB['pressure'] + modITERDB['pressure']

tsrc ={}
tsrc['iprofiles_src'] = 'iterdb'
tsrc['eprofiles_src'] = 'iterdb'
tsrc['pressure_src'] = 'expeq'
tsrc['current_src'] = 'expeq'
tsrc['rhomesh_src'] = 'eqdsk'

In example(3.5), although the 'pressure_src' is set to 'eqdsk', the imported value of the total pressure will override the sourced profile in the write_expeq functions. Similarly, the imported electron temperature profile will override the sourced profile in the write_exptnz functions.

All the profiles that can be externally imported into CheasePy an override sourced profiles are listed in the following table: 

| Profile  | Description       | Profile  | Description       |
|----------|-------------------|----------|-------------------|
| rhopsi   | Poloidal Grid     | rhotor   | Toroidal Grid     |
| ne       | Electron Density  | Te       | Electron Temperature |
| ni       | Ion Density       | Ti       | Ion Temperature   |
| nz       | Impurity Density  | Tz       | Impurity Temperature |
| Iprl     | Parallel Current  | Jprl     | Parallel Current Density |
| fprime   | Current Flux Density |    |                  |
| pressure | Total Pressure    | pprime   | Pressure Gradient |
| Zeff     | Effective Atomic Number | |                  |

All the examples and discussions above consider only the sources for input profiles for the zeroth iteration in CheasePy which reconstruct the MHD equilibrium depending on the input profiles and equilibrium geometry. If CheasePy code runs for several iterations it will keep using the same profiles given in the source files, but ignoring the imported profiles after the zeroth iteration. Hence, another source of the profiles and equilibrium geometry sometimes required to track the changes made in the zeroth iteration by the imported profiles. CheasePy allows the user to change the source of each profile from one iteration to another by providing these sources in a Python list. For every profile the user needs to change its source, this source has to be of type list and of length equals the number of source that the user needs to switch between them. CheasePy code will keep using the last source in the list for the rest of iterations in case the length of the source list is less than the number of iterations. Usually the generated EXPTNZ, EXPEQ, EQDSK, and CHEASE files after each iteration have updated profiles and equilibrium geometry.

If the user needs to do several iteration for the study case in example(3.5), the second source of 'eprofile_src' need to be taken from EXPTNZ or CHEASE files, however, the 'pressure_src' need to be taken from EXPEQ, EQDSK, or CHEASE files to consider the imported profiles that was enforced in the zeroth iteration. Hence, example(3.5) can be rewritten as follow:

---

21 This list can be extended to include other imported profiles by considering them in the write_expeq and write_exptnz functions.

---
Example 3.6. A user has two *iterdb* files, *basecase.iterdb* which has the original profiles of the electrons, ions, and impurities, and *modTeProf.iterdb* which has a modified electron temperature \( T_e \) profile. The following code shows how to include the contribution of the modified electron temperature into the pressure profile, and how to consider these variations in further iterations. Also, the user needs to change the source of the grid after the zeroth iteration to be taken from the *CHEASE* source file.

```python
import cheasepy

inParam = {nrhomesh:0}

baseITERDB = cheasepy.read_iterdb(iterdbfpath='basecase.iterdb',
setParam=inParam,eqdsk='EQDSK')
modITERDB = cheasepy.read_iterdb(iterdbfpath='modTeProf.iterdb',
setParam=inParam,eqdsk='EQDSK')

importedVals = {}
importedVals['rhotor'] = modITERDB['rhotor']
importedVals['rhopsi'] = modITERDB['rhopsi']
importedVals['Te'] = modITERDB['Te']

eqdskdata = cheasepy.read_eqdsk(eqdskfpath='EQDSK')

importedVals['pressure'] = eqdskdata['pressure']-baseITERDB['pressure']

srcVals = {}
srcVals['iprofiles_src'] = 'iterdb'
srcVals['eprofiles_src'] = ['iterdb','exptnz']
srcVals['pressure_src'] = ['eqdsk','expeq']
srcVals['current_src'] = 'expeq'
srcVals['rhomesh_src'] = ['eqdsk','chease']
```

### 3.5.2 CHEASE Namelist Parameters

A list of namelist parameters is provided in section[3.2]. These namelist parameters are required for *CHEASE* code to run properly and give relevant results to the case under study. *CheasePy* code has a default value for each parameter in the namelist, but the user can change any of them depending on several factors, such as the input profiles, input sources, and desired resolution. *CheasePy* provides a dictionary-type variable called `namelistVals` that you can use to assign a new value for any of the parameter in the namelist. Then, this `namelistVals` is passed as an input argument to the `cheasepy` function to create the `chease_namelist` file using the user-defined and default values for each parameter.[22]

If the *CheasePy* user runs multiple iterations of the *CHEASE* code and needs to change any of the input parameters from one iteration to another, an equal length list of these parameter values need to be assigned to each item in the `namelistVals` input argument, even if the values of these parameters don’t change from one iteration to another. In other words, if a user needs to change the type of couple of namelist parameters from int or float to list without changing all the type of other namelist parameters an error will be raised by *CheasePy* as it expects all parameters in the namelist of the same type, i.e single-valued or list.

---

[22]The *CheasePy* user can revise the list of parameters required by *CHEASE* code in the table in section[3.2].
Example 3.7. A user needs to run CHEASE code using pressure and current profiles from the EQDSK for the zeroth iteration and from EXPEQ for other iterations. Also, CHEASE code should expect $f f'$ in the zeroth iteration, and $I_||$ for other iterations.

\[
\begin{align*}
\text{namelistVals} & = \{ \\
\text{namelistVals['NS']} & = [128,128] \\
\text{namelistVals['NT']} & = [128,128] \\
\text{namelistVals['NISO']} & = [128,128] \\
\text{namelistVals['NPSI']} & = [512,512] \\
\text{namelistVals['NCHI']} & = [512,512] \\
\text{namelistVals['RELAX']} & = [0.0,0.0] \\
\text{namelistVals['NEQDSK']} & = [1,0] \\
\text{namelistVals['TENSBND']} & = [0,0] \\
\text{namelistVals['TENSPROF']} & = [0,0] \\
\text{namelistVals['NRHOMESH']} & = [0,0] \\
\text{namelistVals['cocos_in']} & = [1,1] \\
\text{namelistVals['cocos_out']} & = [11,11] \\
\text{namelistVals['NSTTP']} & = [1,3] \\
\text{namelistVals['NPROPT']} & = [3,3] \\
\text{namelistVals['NPPFUN']} & = [8,8] \\
\end{align*}
\]

It worth reminding the user that there are many other parameters those are set to their default values, and out-of-all the most two important parameters are NRBOX and NZBOX which specify the resolution of the profiles and equilibrium geometry in the output EQDSK file. So, if the user wants to use or have an EQDSK file of higher resolution their values need to be changed from the default, i.e. 60, and go higher in value, e.g. 513.

CheasePy code will call the create_namelist function and pass the list of parameters which will in turn create the chease_namelist file required by CHEASE code.

3.5.3 Modes for CheasePy Running

There are several options for a user who runs CheasePy that can be accessed by setting the value of runmode in cheaseVals dictionary variable:

Option 1: To setup the environment for CHEASE code to run by creating the required files, i.e. EXPEQ, EXPTNZ, and chease_namelist, and start send the right command to the system to run the chease_hdf5 executable file in the terminal to establish the MHD equilibrium for the current input data. At the end of each successful run, CheasePy code will generate several plots to compare the input and output profiles in the cheaseresults.pdf file by calling plot_chease function.

Option 2: In case of unsuccessful run that doesn’t produce cheaseresults.pdf file, the second option that runmode can take instruct CheasePy code to search for the output HDF5 files from CHEASE code and plot their contents against the input profiles.

Option 3: This option instructs CheasePy to search for all the available input and output files from any previous CHEASE run and remove all of these files. Hence, the current folder will contains only chease_hdf5 and runchease.py. This option is helpful when a user wants to start a new CHEASE run, because when CheasePy starts running it searches first for any files from a previous run and automatically continue from where that previous run stopped.

\[\text{The output EQDSK file overwrites the old version of the EQDSK file, in case the user wants to use the latest output EQDSK file as as source for profiles or equilibrium geometry for a new iteration.}\]
When a user selects option 1 for the runmode in cheaseVals which instructs CheasePy to run CHEASE code, the user can setup the environment using removeinputs and removeoutputs options before the run starts, and the user have three other options:

Option 1: If the user sets both removeinputs and removeoutputs options to yes, CheasePy will remove all the input and output files related any previous run before running CHEASE code. This requires a path to the shot files to be provided by the user for CheasePy to copy them into the current folder before starting a new run.

Option 2: If the user sets removeinputs to no and removeoutputs to yes, the CheasePy code will keep the input files and remove all the output files from any previous run, then CheasePy code will start an new run using the available input files in the current directory without any need to copy these files from their source directory.

Options 3: If the user sets both removeinputs and removeoutputs options to no, CheasePy will keep all the input and output files from any previous run and use the simulation data of the last iteration from that previous run as the initial condition of the current run and starts the simulation from there. This option is helpful when a previous run crashes or reaches the maximum wall time before the end of the simulation, so CheasePy can restarted from where it left.

The next option the user need to set in the cheaseVals dictionary is the path to the folder/directory of the shot measurements to copy these profile and equilibrium geometry files to the current directory. This step is required if the input and output files from a previous run were removed. The CheasePy user needs to remember that the filename of profile and equilibrium geometry files needs to start with folder name followed by an ‘underscore’ then the type of the data files. For example, for a shot folder named DIIID_174082, the profiles and equilibrium geometry files should be named: DIIID_174082_EXPTNZ, DIIID_174082_EXPEQ, DIIID_174082_CHEASE, DIIID_174082_EQDSK, or DIIID_174082_PROFILES.

The last option the CheasePy user has to set before calling the cheasepy function is the cheasemode entity in the cheaseVals dictionary which can be set to any of the following options:

Option 1: Setting cheasemode to ‘1’ tells CheasePy to reconstruct the MHD equilibrium using CHEASE code using the input profiles and equilibrium geometry. If the iterTotal entity in srcVals is set to a number ‘n’, CHEASE will rerun for ‘n’ iterations, and the source files may change depending on the length of profile’s sources list.

Option 2: Setting cheasemode to ‘2’ tells CheasePy to reconstruct the MHD equilibrium from the input profiles and equilibrium geometry using CHEASE code. Then, CheasePy will use the error in estimating the total toroidal current to correct the parallel current ($I_{||}$ or $J_{||}$), and use this corrected quantity as input to the CHEASE next iteration. CheasePy will estimate that correction in every iteration until the error is small enough to stop the simulation.

Option 3: Setting cheasemode to ‘3’ tells CheasePy to reconstruct the MHD equilibrium from the input profiles and equilibrium geometry using CHEASE code. Then, CheasePy will use the error in estimating the total toroidal current to correct the pressure component ($P$ or $P'$), and use this corrected quantity as input to the CHEASE next iteration. CheasePy will estimate that correction in every iteration until the error is small enough to stop the simulation.

The current_correction and pressure_correction functions in the cheasepy.py are called automatically every time a correction in parallel current (option 2) or pressure (option 3), respectively,
needs to be done. The CheasePy user is free to do any modifications to these functions to handle the correction in a more suitable way for the case under study. But it is recommended that the user treat these functions as a template and do not change the variable name of the returned values for the CheasePy to run properly and give relevant results.

3.6 Plotting the Outputs

When CheasePy code completes the current run it plots some fields such as the density and temperature profiles, the current flux profile, the magnetic field, the toroidal current, etc., with all figures packed together in a single PDF file. The user can plot the fields in the outputs of a previous run of CheasePy directly without rerun the code by by calling the `plot_chease(OSPATH,skipfigs)` function which takes the path to the HDF5 files as the first argument (OSPATH), and the number of of skipped figures as the second argument (skipfigs). The number of skipped figures is useful when the CHEASE does large number of iterations and the user need to plot only few of them, so the user can set skipfigs to a number of figures that will be skipped between the plotted ones. When passing a path to a folder as the first argument to the `plot_chease` function, it searches for all files named `chease_iterxxx.h5`, parse them one by one, extract all the fields and coordinates, and then plot the selected fields. However, if you passed the path to a specific HDF5 file, it plots only the fields in this single file.

4 Future Plans

CHEASEPY code is under continuous development to add new features to the code. However, this version of the code is stable and supposed to give no issues. In case the user has any concern or need to add more features to the code you may contact Ehab Hassan at ehab@utexas.edu. Otherwise, users can add these features themselves but they supposed to inform the original developers with these added features.

5 Acknowledgment

I definitely appreciate the huge contribution of Gabrielle Merlo with his valuable time through a continuous discussion, feedback, and guidance since in October 2018 to have this work done. Also, I would like to thank Olivier Sauter for replying to several questions via email, and responding to our concerns about understanding the workflow in CHEASE code.

I need also to thank Mike Kotschenreuther, David Hatch, and Yasutaro Nishimura for feedback and discussion about the best workflow for CheasePy.

Also, thanks should be given to Michael Halfmoon and Max Curie for running test cases and provide me with valuable feedback and comments that help me to put up this user guide together in this form.

6 References

- Lütjens, Hinrich, Anders Bondeson, and Olivier Sauter. "The CHEASE code for toroidal MHD equilibria." Computer physics communications 97.3 (1996): 219-260.
- Sauter, O., and S. Yu Medvedev. "Tokamak coordinate conventions: COCOS." Computer Physics Communications 184.2 (2013): 293-302.

---

24It is easy to add plots for other fields those are not plotted by copying the chunk of code used to plot another field and replace the name of the existing field with the new field name. You need also to rename the figure object and add it to the PDF file.