SOFT Documentation

Release 1.0

Mathias Hoppe

Aug 09, 2019

Contents:

1 Introduction **3**

2 Compiling **5**
- 2.1 Dependencies . . . 5
- 2.2 Obtaining the code . . . 5
- 2.3 Compiling . . . 6
- 2.4 Usage . . . 6

3 How to run SOFT **7**
- 3.1 Examples . . . 7
- 3.2 Basic syntax . . . 8
- 3.3 Environments . . . 8

4 Magnetic equilibria **11**
- 4.1 Analytic circular . . . 11
- 4.2 Numeric . . . 12

5 Distribution functions **13**
- 5.1 File format . . . 13
- 5.2 Helper tools for CODE/NORSE . . . 14

6 Geometric kernels **15**
- 6.1 Output file . . . 16
- 6.2 Working with kernel function . . . 16

7 Polarization information **19**
- 7.1 What information does SOFT store? . . . 19
- 7.2 File format . . . 20

8 3D emission maps **21**
- 8.1 Solving for surface-of-visibility . . . 21
- 8.2 Visualizing . . . 22

9 Parameter reference **23**
- 9.1 Global options . . . 23
- 9.2 Particle settings . . . 25
- 9.3 Magnetic settings . . . 28
- 9.4 sycout settings . . . 29

10 The SDT format **35**
10.1 Basic structure 35
10.2 Example SDT file 35

11 Troubleshooting **37**

12 Indices and tables **39**

Index **41**

You have reached the Quickstart documentation of SOFT, the Synchrotron-detecting Orbit Following Toolkit. To get the source code of SOFT, check the SOFT website or SOFT GitHub repository. If you're looking for the gory mathematical details of how SOFT is implemented, you should look into the SOFT Manual.

CHAPTER 1

Introduction

SOFT is a synthetic synchrotron diagnostic that can be applied to study the synchrotron radiation emitted by runaway electrons in tokamaks. By solving the guiding-center equations of motion in a numeric magnetic equilibrium, the physics of the system are utilized and allows SOFT to be applied to experimental scenarios.

CHAPTER 2

Compiling

SOFT is written in C, and as such is straightforward to setup on a Linux system. While SOFT hasn't been tested on any other system, it should be possible compile and run on for example Windows and Mac with some additional effort.

2.1 Dependencies

SOFT depends on a number other technologies, some of which are required for compilation, while others can be compiled in optionally. Technologies that are absolutely mandatory in order to compile SOFT are

- CMake, for preparing necessary build files.
- A C compiler with OpenMP support (such as gcc).
- GNU Scientific Library, for various mathematical operations. If a version of GSL older than 2.0 is used, the GSL extension interp2d must also be installed.

A number of libraries are also optional for compilation, and can be compiled in for additional functionality. The optional libraries are

- HDF5 for reading/writing data in HDF5 format.
- MATLAB, for reading/writing data in MATLAB's *.mat format.
- An MPI library, such as MPICH or OpenMPI. Compiling in support for MPI allows running SOFT across multiple computers, such as on a supercomputer cluster.

2.2 Obtaining the code

You may clone the latest build from the SOFT GitHub repository via the command line:

```
$ git clone https://github.com/hoppe93/SOFT.git
```

or if you have your ssh keys configured with GitHub:

```
$ git clone git@github.com:hoppe93/SOFT.git
```

2.3 Compiling

Once the SOFT source code has been obtained and all required and desired dependencies have been installed, navigate to the directory cloned from GitHub:

```
$ cd SOFT
```

Next, to compile SOFT, create a `build` directory, navigate to it, run CMake followed by make, using the following set of commands:

```
$ mkdir build
$ cd build
$ cmake ../ -DUSE_HDF5=ON -DUSE_MATLAB=ON -DUSE_MPI=OFF
$ make
```

If the build was successful, the SOFT binary will be found under `build/src/soft`. The flags starting with `-D` specify configuration options, and in the command above we see that in this case SOFT would be configured with HDF5 and MATLAB support, but without MPI support. This is the default, and would have happened even if those flags were not specified. To enable/disable compilation for either of these libraries, simply specify `ON`/`OFF` as appropriate in the above.

2.4 Usage

All configuration of a SOFT run is done in a separate script file, commonly referred to as a `pi` file (for *Particle Information*). As such, running SOFT is as simple as

```
$ ./soft pi
```

assuming the `pi` file has been setup appropriately. There are a large number of options that can be specified in the `pi` file, and for this reason the details of using SOFT are left to the *How to run SOFT*.

CHAPTER 3

How to run SOFT

All configuration of a SOFT run is done in a separate configuration file, commonly referred to as a `pi` file. In this section the basic structure of a `pi` file will be explained in detail. For detailed information about which options can be set, pleaes consult the SOFT manual.

3.1 Examples

The best way to learn how to set up run scripts for SOFT is to see examples of such run scripts. A basic `pi` file can look like the following:

```
# Basic SOFT pi file
useequation=guiding-center-relativistic
usetool=sycamera

# Specify magnetic field
magnetic_field=circular   # Use analytic magnetic field
magnetic circular { B0=5; major_radius=0.68; minor_radius=0.22; safety_factor=1; }
domain_has_outer_wall=no  # Remove outer device walls to prevent from blocking␣
↪radiation

# Set phase-space
particles {
    t=0,-1
    rdyn=0.84,1000
    p=3e7,3e7,1
    pitch=0.15,0.15,1
}

# Specify properties for the sycamera tool
tool sycamera {
    aperture=0.006                # Side length (in m) of (square) detector
    cone=delta                    # Use the cone model (not full angular␣
↪distribution)
```

(continues on next page)

(continued from previous page)

```
    direction=0,1,0                 # Normal vector of detector surface (not␣
↪necessarily normalized)
    position=0,-1.069,0             # Position vector of detector, relative tokamak␣
↪point of symmtetry
    product=image                   # Output a synchrotron image when done
    radiation=synchrotron_spectrum # Take spectrum of radiation into account
    spectrum=5e-7,1e-6              # Detector spectral range
    toroidal_resolution=3500        # Number of steps in toroidal integral
    vision_angle=2.0                # Size of field-of-view
}

# Specify properties for the 'image' sycout
sycout image {
    pixels=1000
    name=image.dat
}
```

The settings available for SOFT are many more, and for a detailed list of which settings are available, please consult the SOFT manual. Further examples of `pi` files for different purposes are:

- distpi – Illustrates how SOFT can be run together with a runaway distribution function.
- hollowpi – An example of simulating a hollow electron beam.
- simplepi – The basic example shown above, setting just the most important options.
- orbitpi – Shows how to use the orbit following part of SOFT to simulate particle orbits.

3.2 Basic syntax

Options in a `pi` file are specified by first giving the name of the option, followed by an equal sign, followed by the value to assign to the option. White-space around the equal sign is ignored. Typically, everything between the equal sign and the end-of-line marker is considered part of the assigned value, except for any white-space coming either directly after the equal sign, or directly before the end-of-line-marker. It is however possible to put several settings on the same line by separating them with semi-colons (`;`).

Comments can be given by preceding the comment text with a hashtag symbol (`#`). Any text following the hashtag on the same line will be ignored. Note that comments *cannot* be ended with a semi-colon.

Some options should be assigned vectors of data, such as the `direction` and `rdyn` options (among others) in the above example. Each component of the vector must be separated by a comma, and any white-space surrounding commas is ignored. Note that all floating-point numbers can be specified using either decimal form (i.e. `1000` or `0.68`) or C scientific notation (i.e. `5e-7`).

3.3 Environments

Some options in SOFT are considered *global* and are specified directly in the file, such as for example `useequation` and `usetool` in the example above. Many options are however specific to certain modules of SOFT, and they are instead specified inside the appropriate option *environment*.

There are four different environments in SOFT, all of which are syntactically similar. With the exception of the `particles` environment (which really just sets what could be considered global options), they are also conceptually similar.

The `magnetic`, `tool` and `sycout` environments specify options for a particular SOFT module, and the name of the module must be specified in the environment *header*. The settings are then wrapped within curly brackets (`{` and `}`) and given to the specified module. Note that even if an environment for a module is present in the configuration file, it does not mean that the module will automatically used. Other options must be set to enable modules.

The basic syntax for an environment `environment` configuring the module named `module` is:

```
environment module {
    ...
}
```

The `particles` environment does not require any module name to specified.

3.3.1 magnetic

The `magnetic` environment specifies settings for the magnetic equilibrium to use, as well as the surrounding walls. Currently, there are two different so called *magnetic handler* modules that can be used. The first and simplest is the `circular` magnetic handler which implements simple analytic circular magnetic field with a constant safety factor. The second magnetic handler, named `numeric`, allows the specification of a magnetic field numerically from for example an HDF5 or MATLAB *.mat file.

3.3.2 particles

The `particles` environment sets a number of options relating to the phase-space of the run. Since it is necessarily tied to the *particles* module of SOFT, the module name part of the environment specification given above should be omitted.

In addition to specifying the bounds of and number of points in phase-space, the `particles` environment can also be used to specify a different mass or charge of the simulated particle species.

Note: The `orbit` tool for tracing particle orbits only allows simulating a single point of phase-space at a time, and can otherwise give rise to some very anonymous errors.

3.3.3 tool

The `tool` environment sets the options for particular tool. A tool, in SOFT, is a module which receives information about a computed orbit and processes it. Currently, there are two tools in SOFT, and these are the `orbit` and `sycamera` tools. The `orbit` tool simply traces a particle or guiding-center orbit, keeps track of a few addiational parameters, and outputs it all to a CSV file.

The `sycamera` tool is the synchrotron camera (or rather detector) tool which gives SOFT its name. A large part of the SOFT code is dedicated to this module, and the options set by this tool include for example the type of synchrotron radiation model to use, the number of toroidal steps to take, various detector properties among many other things.

3.3.4 sycout

Due to the great versatility of the `sycamera` tool, the types of output that could be obtained it are numerous. Since each of the output types requires its own set of settings, a separate environment for specifying settings to the output handler of the `sycamera` tool was created.

The `sycout` environment thus specifies settings of a `sycamera` output handler module. To date there are five different *sycout* modules, namely

Module name	Description
green	Generates a *Green's function* which relates the distribution of runaways to the resulting spectrum or image. (Can) allow fast computation of image/spectrum.
image	Generates a synthetic synchrotron image.
space3d	Stores 3D information about all particles contributing to a synchrotron image allows visualizing the corresponding surface-of-visibility.
spectrometer	Generates a spectrum curve.
topview	Stores information about where particles where located in the *xy*-plane when when they emitted towards the detector. Allows visualizing the toroidal distribution of particles that are visible to the detector.

CHAPTER 4

Magnetic equilibria

There are currently two magnetic handler modules available for SOFT. The `circular` handler implements a simple circular magnetic field with a constant safety factor, and is somewhat faster than the alternative. The `numeric` allows the magnetic field to be loaded from numeric data, which is interpolated. This handler is often the desired one as it allows complicated magnetic geometries to be simulated.

4.1 Analytic circular

The `circular` magnetic handler implements the magnetic field:

$$\boldsymbol{B}(r,\theta) = \frac{B_0}{1-(r/R_m)\cos\theta}\left(\frac{r}{q(r)R_m}\hat{\boldsymbol{\theta}} - \hat{\boldsymbol{\phi}}\right)$$

where r is the minor radius, θ the poloidal angle, B_0 the magnetic field strength on the magnetic axis ($r = 0$), R_m is the major radius, q is the safety factor, $\hat{\boldsymbol{\theta}}$ is a unit vector in the poloidal direction and $\hat{\boldsymbol{\phi}}$ is a unit vector in the toroidal direction. While this formula allows arbitrary q-profiles, SOFT currently only implements this magnetic field with a linear q-profile.

The magnetic field shown above has three free parameters, namely the field strength B_0, the tokamak major radius R_m and safety factor $q(r) = q_0$. These parameters must be specified by the user, and are set by specifying the options `B0`, `major_radius` and `safety_factor` respectively in the `magnetic circular` environment. For SOFT to be able to determine when a particle escapes confinement and hits the wall, the minor radius of the device must also be specified. A circular cross section is assumed. All options for the *circular* magnetic handler are set according to

```
magnetic circular {
    B0 = 5
    major_radius = 0.68
    minor_radius = 0.22
    safety_factor = 1
}
```

4.2 Numeric

One of the great strengths of SOFT is that magnetic equilibrias can be specified as numeric data, allowing complicated magnetic configurations, and in particular, experimentally measured data, to be plugged into SOFT. Specifying a numeric equilibrium in the `pi` file is as simple as

```
magnetic_handler=numeric
magnetic numeric {
    name=/path/to/magnetic/equilibrium.mat
}
```

Currently, the equilibrium data can be stored in either a HDF5, (MATLAB) MAT or SDT file. Both HDF5 and MATLAB files can be created easily with user-friendly tools such as Python or MATLAB, while SDT (for *Semi-Descriptive Text*) is a SOFT-specific text-based format which is likely the best choice if the magnetic equilibrium is generated using a small C/C++ program which is difficult to interface with HDF5 or MATLAB.

Since SOFT assumes the magnetic field to be toroidally symmetric, the magnetic field components in a poloidal plane must be specified. SOFT uses a cylindrical coordinate system for specifying the magnetic field, so that $\boldsymbol{B} = B_r\hat{\boldsymbol{r}} + B_z\hat{\boldsymbol{z}} + B_\phi\hat{\boldsymbol{\phi}}$, where $B_r\hat{\boldsymbol{r}}$ denotes the component radially out from the point of symmetry of the tokamak, $B_z\hat{\boldsymbol{z}}$ denotes the component in the vertical direction, and $B_\phi\hat{\boldsymbol{\phi}}$ denotes the component in the toroidal direction, perpendicular to the poloidal plane in which the magnetic field is given.

4.2.1 Variables

Both HDF5, MATLAB and SDT files have a *variable* concept where data within the file is named. Because of this, SOFT looks for certain variables in the datasets, loads them and gives them meaning in the code. The following variables must be present in all SOFT magnetic equilibrium files:

Variable	Type	Description
`Br`	m-by-n matrix	Radial component of magnetic field (radius-by-z).
`Bphi`	m-by-n matrix	Toroidal component of magnetic field (radius-by-z).
`Bz`	m-by-n matrix	Vertical component of magnetic field (radius-by-z).
`desc`	String	A longer description of the equilibrium. Must be present, but may be empty.
`maxis`	1-by-2 vector	Specifies the location of the magnetic axis in the (R, z)-plane.
`name`	String	Name of the equilibrium. Must be present, but may be empty.
`r`	1-by-m vector	List radial points in which the components of the magnetic are given.
`separatrix`	2-by-many vector	List of contour points marking the separatrix in the (R, z)-plane.
`wall`	2-by-many vector	List of contour points marking the bounds of the device in the poloidal plane.
`z`	1-by-n vector	List of vertical points in which the components of the magnetic field are given.

Note: Only one of the `separatrix` and `wall` variables is required to be present in the equilibrium file. Both may be present, and in that case the domain contour to use can be specified as an additional option to the `numeric` magnetic handler. By default the `wall` contour will be used if available.

CHAPTER 5

Distribution functions

In SOFT, distribution functions depend on three variables, namely the major radius ρ at which the guiding-center orbit was initiated, the momentum p of the particle, as well as the cosine of the pitch angle $\xi = \cos\theta_{\mathrm{p}}$ in the outer midplane.

5.1 File format

Distribution functions are given to SOFT as Matlab MAT-files. SOFT expects the following variables to be present in the file:

Name	Description
`description`	String describing the distribution function.
`f`	Actual distribution function. An n_r-by-$n_p n_\xi$ matrix (see below).
`name`	String naming the distribution function.
`p`	Vector containing points of momentum. Size 1-by-n_p.
`punits`	String describing the units of `p`. Either `ev`, `normalized` or `si`.
`r`	Vector containing radial points. Size 1-by-n_r.
`xi`	Vector containing (cosine of) pitch angle points. Size 1-by-n_ξ.

The most important variable in a SOFT distribution function file is `f`, which is the actual distribution function. The variable is stored as a matrix with each row representing a momentum-space distribution function, i.e. with the radial coordinate changing along the row index.

Each row of `f` corresponds to a momentum-space distribution function, shaped as one long $n_p \times n_\xi$ vector. The elements are ordered into n_ξ groups of n_p elements, so that the first n_p elements of the vector corresponds to holding ξ fixed and varying p.

The `name` and `description` variables are fairly arbitrary and are only included to provide the user with basic information about the distribution function.

The `p`, `r` and `xi` variables are vectors consisting of n_p, n_r and n_ξ elements respectively. Together, the vectors specify the grid in momentum, radius and (cosine of) pitch angle on which the distribution function is defined.

To allow users to specify momentum coordinates in the units most convenient for them, and more importantly to prevent mix-ups of used units, the variable `punits` must be provided specifying the units used for the momentum variable. Allowed values are `ev` (for momentum in eV$/mc$), `normalized` (for $p \equiv \gamma\beta$, where γ is the electron's Lorentz factor and β is the electron's speed normalized to the speed of light) and `si` (for SI units, i.e. kg $\cdot$ m/s).

Note: Even though CODE is commonly used to generate distribution functions for SOFT, plain CODE distribution functions are not directly compatible with SOFT. The distribution function given as output by CODE consists of a set of Legendre polynomial coefficients used in evaluating the distribution function $f(p, \xi)$. SOFT on the other hand requires the function values to be already evaluated.

5.2 Helper tools for CODE/NORSE

A nice graphical helper tool has been developed for analyzing CODE/NORSE distributions and generating distributions readable by SOFT. The tool is called codeviz and is available on GitHub.

CHAPTER 6

Geometric kernels

The way SOFT is constructed makes it possible to rewrite the "SOFT equation" on the form

$$I_{ij} = \int \mathrm{d}\rho \mathrm{d}p_{\parallel} \mathrm{d}p_{\perp}\, f(\rho, p_{\parallel}, p_{\perp}) \hat{I}_{ij}(\rho, p_{\parallel}, p_{\perp}, \boldsymbol{x}_0)$$

where I_{ij} is the brightness of pixel (i, j), and $\hat{I}_{ij}$ denotes the *geometric kernel function* for a particular detector/tokamak combination, that connects the distribution of runaways of a particular velocity and initial position, with the image seen by a particular camera in a specific tokamak. The great benefit of this formulation is that only a set of multiplications are required to produce the synchrotron radiation image seen by a camera. A similar formulation for the synchrotron spectrum exists.

The format of the Green's function is specified in the `pi`-file using the `dimensions` option. The value of this option is a set of characters denoting each of the variables that should appear in the Green's function. For example, `dimensions = r12ij` would generate a Green's function containing information about radius, velocity coordinate 1, velocity coordinate 2 as well as both pixels of the image. The possible characters and their meaning are:

Function	Description
`1`	Velocity coordinate 1. Depends on which coordinates are used in the `pi`-file.
`2`	Velocity coordinate 2. Depends on which coordinates are used in the `pi`-file.
`i`	The "y"-axis of the image.
`j`	The "x"-axis of the image.
`r`	Radial coordinate.
`w`	Spectrum wavelength.

To generate a geometric kernel function with SOFT, create a new sycout environment in your `pi` file with the format

```
sycout green {
    format=mat
    output=greenW.mat
    function=r12ij
    pixels=60
}
```

All options for the `green` sycout are documented in the *Parameter reference*.

6.1 Output file

The file generated by SOFT containing the geometric kernel function will contain the variables listed in the table below. The actual geometric kernel functions is found as a vector named `func` which can be reshaped to be handled more easily.

Variable	Type	Description
`func`	1-by-$n_\rho n_1 n_2 n_\lambda n_i n_j$ vector	Geometric kernel function
`param1`	1-by-n_1 vector	Velocity parameter #1
`param1name`	String	Name of velocity parameter #1. E.g. `ppar`.
`param2`	1-by-n_2 vector	Velocity parameter #2
`param2name`	String	Name of velocity parameter #2. E.g. `pperp`.
`pixels`	Integer	Number of pixels
`r`	1-by-n_ρ vector	List of radial points
`stokesparams`	Integer	1 if elements are Stokes parameters. 0 if only intensities are stored.
`type`	String	Type of geometric kernel. Either of the functions listed in the table above.
`wavelengths`	1-by-n_λ vector	List of wavelength points.

6.2 Working with kernel function

To more easily work with the geometric kernel function it should be reshaped into an appropriately dimensioned array. In Matlab, this can be done through

```
load softOutput % Kernel function assumed to be located in 'softOutput.mat'

% Get number of elements in each dimension
nw = length(wavelengths);
n1 = length(param1);
n2 = length(param2);
nr = length(r);

% Reshape kernel function
Ihat = reshape(func, [pixels, pixels, nw, n2, n1, nr]);

% Access image at radius #1, param1 #2, param2 #3 and wavelength #4
I = squeeze(Ihat(:, :, 4, 3, 2, 1));
```

and similarly in Python

```
import numpy as np
import scipy.io

# Load mat-file
matfile = scipy.io.loadmat('softOutput.mat')

# Set variables
func = matfile['func'][0]
pixels = matfile['pixels'][0][0]
param1 = matfile['param1'][0]
param1name = matfile['param1name'][0]
```

(continues on next page)

(continued from previous page)

```
# ...and the same for all other variables...

# Get number of elements in each dimension
nr = r.size
n1 = param1.size
n2 = param2.size
nw = wavelengths.size

# Reshape kernel function
Ihat = np.reshape(func, (nr, n1, n2, nw, pixels, pixels))
```

Note: The order in which the number of elements are given to `reshape` is very significant!

The above examples are for a function of type `r12ij`.

CHAPTER 7

Polarization information

When `radiation=synchrotron_spectrum` SOFT will also store information about the polarization of the detected radiation. Using the `polimage` and `polspectrometer` sycouts, it is possible to generate output files containing the polarization information in image or spectrum format. In this section usage and interpretation of the data will be briefly be discussed.

7.1 What information does SOFT store?

SOFT stores the four Stokes parameters, S, Q, U and V, averaged over the relevant parameters (depending on which model is being used). The emitted synchrotron power per unit frequency, per unit solid angle, can be written in terms of the two quantities $A_\parallel$ and $A_\perp$ as

$$\frac{\mathrm{d}^2 P}{\mathrm{d}\omega \mathrm{d}\Omega} \propto \left| -\boldsymbol{\epsilon}_\parallel A_\parallel + \boldsymbol{\epsilon}_\perp A_\perp \right|^2 .$$

where $\boldsymbol{\epsilon}_\parallel$ is a vector corresponding to polarization in the gyration plane, and $\boldsymbol{\epsilon}_\perp$ to polarization in the plane orthogonal to that. It can be shown that the Stokes parameters can be expressed using $A_\parallel$ and $A_\perp$ through

$$\begin{aligned} I &\propto A_\parallel^2 + A_\perp^2, \\ Q &\propto \left(A_\perp^2 - A_\parallel^2 \right) \cos 2\beta, \\ U &\propto \left(A_\perp^2 - A_\parallel^2 \right) \sin 2\beta, \\ V &\propto 2 A_\parallel A_\perp \cos 2\beta. \end{aligned}$$

The angle β is the angle between the plane of parallel polarization and the plane in which the horizontal polarization is *measured*. The first Stokes parameter, I, is just the intensity of the radiation as obtained also from the SOFT `image` sycout.

The fourth Stokes parameter V is often quoted as identically zero in the literature, a result stemming from that the object $A_\parallel A_\perp$ is odd in the angle ψ between the guiding-center's emission cone and a line-of-sight. When averaged over all emission angles, the contribution to V therefore cancels identically. In the angular and spectral distribution implemented in SOFT however, we do not neglect the finite emission width, and therefore obtain a finite contribution

to the V parameter, since it is possible for only part of the emission cone to overlap the detector (corresponding to "cut-offs" in the integration over emission angle).

For a derivation of the full $\frac{\mathrm{d}^2 P}{\mathrm{d}\omega \mathrm{d}\Omega}$ *, see for example Jackson's "Electrodynamics", Landau-Lifshitz "The Classical Theory of Fields" or Mathias Hoppe's Master's thesis* (link).

7.2 File format

The `polimage` sycout of SOFT outputs a variable-based file (such as SDT, HDF5 or Matlab) containing the following variables:

Variable	Description
`detectorPosition`	Vector specifying the position of the detector
`detectorDirection`	Central viewing direction of the detector
`detectorVisang`	Vision angle of the detector
`StokesI`	Stokes I parameter
`StokesQ`	Stokes Q parameter
`StokesU`	Stokes U parameter
`StokesV`	Stokes V parameter
`wall`	Wall data used for the simulation

CHAPTER 8

3D emission maps

Due to the highly anisotropic nature of bremsstrahlung and synchrotron radiation combined with the fact that radiation is only detected if it's emitted directly at the detector, a given detector can only measure radiation from particles in a certain regions of space. It can be shown that these regions of space all satisfy (approximately) the condition

$$\hat{\boldsymbol{b}}(\boldsymbol{x}) \cdot \frac{\boldsymbol{x} - \boldsymbol{X}}{|\boldsymbol{x} - \boldsymbol{X}|} = \cos\theta_{\mathrm{p}}, \tag{8.1}$$

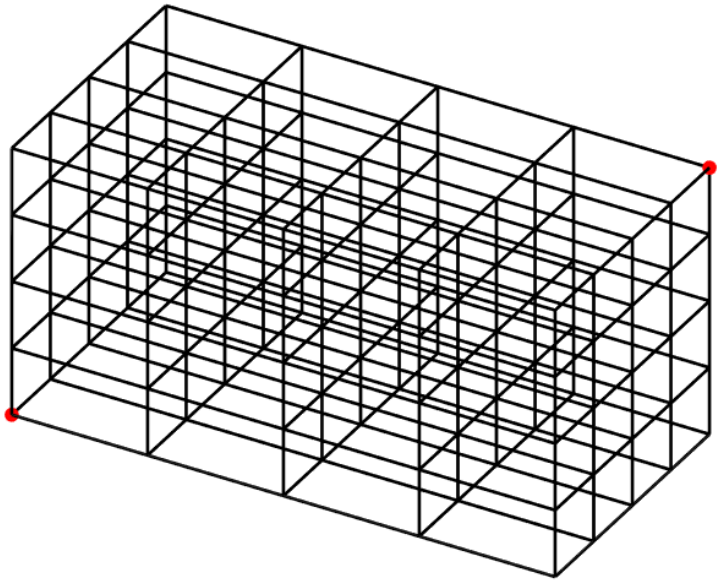
where $\hat{\boldsymbol{b}}$ is the magnetic field unit vector, $\boldsymbol{x}$ is the particle's position, $\boldsymbol{X}$ is the detector's position and θ_{p} denotes the particle's pitch angle (note that the pitch angle also varies as the particle moves in the inhomogeneous magnetic field, and therefore picks up a dependence on $\boldsymbol{x}$). The solution to this equation, i.e. the points $\boldsymbol{x}$ satisfying it, trace out a surface in real space which we refer to as the *surface-of-visibility*. When the detector is located in the midplane, this surface typically takes the shape of a twisted cylinder.

8.1 Solving for surface-of-visibility

Using SOFT it is possible to solve (8.1), accounting for the finite detector size. This is done by adding the sycout `space3d` (see space3d for a parameter reference) to your SOFT runscript. As with every sycout, you must also add a line `product=space3d` to the `tool sycamera` block. One example definition of the sycout is a follows

```
sycout space3d {
    output=outfile.mat
    type=pixels
    pixels=200
    point0=-0.5,-0.25,-0.25
    point1=0.5,0.25,0.25
}
```

The `output` parameter specifies the name of the output file, and the `type` parameter specifies the algorithm to use for storing 3D information. Setting `type=pixels` means SOFT will divide the space into N^3 cells, where N is the value assigned to the `pixels` parameter, between the two edge points `point0` and `point1` (see the figure below; the red dots indicate the locations of the edge points). During the SOFT run, each cell records the radiation being emitted from the box and accumulates it.



The other value available for `type` is `real`, which stores the exact coordinates of each point that contributes to the final image. This means that the output will be more detailed, but it will also grow with each particle.

8.2 Visualizing

Visualization of space3d files is complicatd by the fact that each point represents emitted light, which adds together along lines of sights. A simple C program has been written by Mathias for generating sequences of PNG images from S3D output files. The program is available on GitHub: s3dvid.

CHAPTER 9

Parameter reference

There are a number of settings that can be specified in a `pi` file, and each of the SOFT modules introduces its own set of options. In this section a complete list of all the options that can be set in a `pi` file are given.

Contents

- *Global options*
- *Particle settings*
- *Magnetic settings*
  - *circular*
  - *numeric*
- *sycout settings*
  - *green*
  - *image*
  - *space3d*
  - *spectrometer*
  - *topview*

9.1 Global options

debug

Default value: 0
Example line: `debug=1`

Allowed values: 0 or 1

If set to 1, debug output will be generated and written to `stdout` during the run. Default value is 0.

`domain_has_outer_wall`

Default value: yes
Example line: `domain_has_outer_wall=no`
Allowed values: `yes` or `no`

If set to `no`, ignores all points of the wall/separatrix outside $R = R_m$, where R_m denotes the radial coordinate of the magnetic axis. This will allows the placement of a detector outside the device. The mid-pole will still be present to block out radiation.

`interptimestep`

Default value:
Example line:
Allowed values:

TODO

`magnetic_field`

Default value: None
Example line: `magnetic_field=numeric`
Allowed values: `circular` and `numeric`

Specifies the name of the magnetic field handler module to use. Either `circular` or `numeric`.

`maxtimestep`

Default value: None
Example line: `maxtimestep=1e-11`
Allowed values: Any positive real value

Sets the maximum allowed size of a timestep in the equation solver (whichever it may be). If the adaptive timestep becomes larger than this, it is automatically adjusted to this value. By default there is no limit on how long the timestep can be.

`nodrifts`

Default value: no
Example line: `nodrifts=yes`
Allowed values: `yes` or `no`

If set to `yes`, ignores the drift terms in the first-order guiding-center equations of motion (effectively solving the zeroth-order guiding-center equations of motion). This option only influences behaviour of the code when the guiding-center equations of motion are solved. By default the value of this option is `no` so that the drift terms are kept.

`progress`

Default value: 0
Example line: `progress=10`

Allowed values: Any non-negative integer

Specifies how many times during the run SOFT should print information about the current progress. Information will be printed in uniform steps as particles (defined as points in phase-space) are completed.

threads

Default value: Number of threads suggested by OpenMP
Example line: `threads=3`
Allowed values: Any positive integer (no upper limit)

Overrides the number of threads started by each (MPI) process. By default, SOFT will start the number of threads indicated by the `OMP_NUM_THREADS` environment variable in each process.

tolerance

Default value: 1e-12
Example line: `tolerance=4e-13`
Allowed values: Any positive real number

Specifices the tolerance in the RKF45 solver. The default tolerance is set by the tool used in the run. The `orbit` tool defaults to a tolerance of 10^{-7}, while the `sycamera` defaults to a tolerance of 10^{-12}.

useequation

Default value: None
Example line: `useequation=guiding-center-relativistic`
Allowed values: `guiding-center`, `guiding-center-relativistic`, `particle`, `particle-relativistic`.

Determines which set of equations of motion to solve. Note that the `sycamera` tool requires that the (relativistic) guiding-center equations of motion be solved. Possible values for this option are `particle`, `particle-relativistic`, `guiding-center` and `guiding-center-relativistic`.

usetool

Default value: None
Example line: `usetool=sycamera`
Allowed values: `orbit`, `sycamera`

Sets the name of the tool to use. Can either be `orbit` (which traces orbits), or `sycamera` (which computes various synchrotron-radiation quantities for runaway electrons).

9.2 Particle settings

charge

Default value: One electron charge (i.e. `-1`)
Example line: `charge=4`
Allowed values: `orbit`, `sycamera`

The charge of the particle to simulate, in units of the elementary charge ($e \approx 1.602 \times 10^{-19}$ C). The default value is -1, i.e. the electron charge.

cospitch

Default value: None

Example line: `cospitch=1,0.95,100`

Allowed values: A number $\in [0, 1]$; A number $\in [0, 1]$; any positive integer

Specifies the range of cosines of the particle's pitch anle with which to initiate orbits. The first argument specifies the first value in the range to give to particles, while the second argument argument specifies the last value in the range. The third argument specifies the total number of values to simulate. Example: `cospitch = 0.999,0.97,10`, while initiate ten particles with cosine of the pitch angle values between 0.97 and 0.999.

gc_position

Default value: Yes

Example line: `gc_position=no`

Allowed values: `yes` or `no`

If set to `yes`, assumes that the position given specifies the guiding-center position when solving the guiding-center equations of motion. If set to `no`, the program instead assumes that the particle position is specified and compensates accordingly when solving the guiding-center equations of motion. Has no effect when solving the full particle orbit.

mass

Default value: One electron mass ($0.000548579909\,\mathrm{u}$)

Example line: `mass=2`

Allowed values: Any positive real number

The particle mass in unified atomic mass units (u). The default value is 0.000548579909, corresponding to the electron mass.

p

Default value: None

Example line: `p=1e6,1.2e7,10`

Allowed values: Any real number; any real number; any positive integer

Specifies the range of momenta with which to initiate orbits. The first argument specifies the first momentum value to give to particles while the second argument specifies the last momentum value. The third argument specifies the total number of momentum values to simulate. Example: `p = 3e7,4e7,5`.

pitch

Default value: None

Example line: `pitch=0.05,0.15,14`

Allowed values: A number $\in [0, \pi]$; a number $\in [0, \pi]$; any positive integer

Specifies the range of pitch angles with which to initiate orbits. The first argument specifies the first pitch angle to give to particles while the second argument specifies the last pitch angle. The third argument specifies the total number of pitch angles to simulate. Example: `pitch = 0.03,0.25,15`.

ppar

Default value: None

Example line: `ppar=1e6,1.2e7,14`

Allowed values: Any real number; any real number; any positive integer

Specifies the range of parallel momenta with which to initiate orbits. The first argument specifies the first parallel momentum to give to particles while the second argument specifies the last momentum value. The third argument specifies the total number of momentum values to simulate. Example: `ppar = 3e7,4e7,5`.

pperp

Default value: None

Example line: `pperp=1e6,1.2e7,14`

Allowed values: Any real number; any real number; any positive integer

Specifies the range of perpendicular momenta with which to initiate orbits. The first argument specifies the first perpendicular momentum to give to particles while the second argument specifies the last momentum value. The third argument specifies the total number of momentum values to simulate. Example: `pperp = 3e6,7e6,15`.

r

Default value: None

Example line: `r=0.68,0.84,14`

Allowed values: Any real number inside device; any real number inside device; any positive integer

Specifies the range of radii with which to initiate orbits. The first argument specifies the first radius to give to particles while the second argument specifies the last radius. The third argument specifies the total number of radii to simulate. Example: `r = 0.68,0.84,80`.

rdyn

Default value: None

Example line: `rdyn=0.84,14`

Allowed values: Any real number inside device; any positive integer

Specifies the outermost radius at which to initiate orbits, as well as the number of radii to drop particles on. The innermost radius is automatically set as the magnetic axis, and particles will only be dropped at a radius in the interval if their "effective magnetic axis" radial location is less than the currently simulated. The "effective magnetic axis" arises due to orbit drifts, and if it's presence is not properly accounted for, weird bright or dark spots will show up in synchrotron image (when orbit drifts are taken into account). Example: `rdyn = 0.84,80`.

t

Default value: `0,-1`

Example line: `t=0,1e-6`

Allowed values: Any real number; any real number

The first argument of this parameter specifies the reference time. For most purposes this parameter is most conveniently set to 0. The second argument specifies the end time, at which point an orbit should be considered finished and no longer followed. If the second argument is less than the reference time (the first argument), the orbit will be followed for one full *poloidal* orbit, or until the simulation clock is greater than minus the end time.

9.3 Magnetic settings

Two different magnetic handler modules are provided with SOFT. These are the `circular` module, implementing a simple analytical magnetic field with a circular cross-section and constant safety factor, as well as the `numeric` module, which loads 2D numeric magnetic fields.

Performance-wise, the `numeric` module is somewhat slower than the `circular` model, due to that the former interpolates the 2D magnetic field with a cubic spline. The difference is however only about a factor of two.

9.3.1 circular

B0

Default value: `1`
Example line: `B0=5.2`
Allowed values: Any real number

Specifies the magnetic field strength on the magnetic axis, i.e. on the circle $R = R_{\mathrm{m}}, Z = 0$. In units of Tesla.

major_radius

Default value: `1`
Example line: `major_radius=2`
Allowed values: Any positive real number

Specifies the major radius of the tokamak. In units of meter.

minor_radius

Default value: `1`
Example line: `minor_radius=1`
Allowed values: Any real number

Specifies the minor radius of the device. In units of meter. This parameter only influences the location of the walls of the tokamak, and does not affect the magnetic field.

safety_factor

Default value: `1`
Example line: `B0=1`
Allowed values: Any real number

The safety factor, or q-factor of the tokamak magnetic field. In this analytical model of the magnetic field, the safety factor is a constant.

9.3.2 numeric

axis

Default value: *Set in equilibrium file*
Example line: `axis=0.68,-0.002`
Allowed values: Any positive real number; any real number

Specifies the location of the magnetic axis in a poloidal plane. The first coordinate specifies the major radial location (R) of the axis, and the second coordinate specifies the vertical location (Z) of the axis. SOFT requires the magnetic equilibrium data file to give this value, but under some circumstances it may be desirable to override the value set in the equilibrium file, in which case this parameter can be used.

`file`

Default value: None
Example line: `file=/path/to/magnetic/equilibrium.mat`
Allowed values: Any real number

Specifies the name of the file containing the magnetic equilibrium data to use. The format that this file must have is described under *Magnetic equilibria*. The format of the file is determined by analyzing the file name extension. All file formats supported by the SOFT file interface can be used.

`format`

Default value: `auto`
Example line: `format=mat`
Allowed values: `auto`, `hdf5` or `mat`

Overrides the format specifier for the magnetic equilibrium data file. `auto` is the default, which causes SOFT to determine the file format based on the filename extension. `hdf5` causes SOFT to interpret the data file as an HDF5 file. `mat` causes SOFT to interpret the data file as a Matlab MAT file.

`wall`

Default value: `any`
Example line: `wall=separatrix`
Allowed values: `any`, `separatrix`, `wall`

Specifies which type of wall should be used. Equilibrium data files can contain two types of "walls", namely the actual tokamak wall cross-section or the separatrix/last closed flux surface. SOFT only requires one of these two types to be present in the data file, and with `any` set, the tokamak wall will be first be considered, but if it's not present in the file, the separatrix will be used instead. The `wall` and `separatrix` options forces the use of either of the two types. *The wall is the structure beyond which particles will be considered as lost and no longer followed.*

9.4 sycout settings

A **sycout** (short for *SYnchrotron Camera OUTput*) is an output module that is coupled to the `sycamera` tool of SOFT. Currently the following sycouts are available:

- **green** – Generates a Green's function
- **image** – Generates a camera image
- **polimage** – Generates a camera image with polarization information
- **polspectrometer** – Generates a spectrum with polarization information
- **space3d** – Stores 3D information about the contributions to an image
- **spectrometer** – Generates a spectrum
- **topview** – Stores X and Y coordinates of contributions to an image. Creates a top-down "map" of contributions.

9.4.1 green

The `green` sycout allows you to generate Green's functions for images, spectra or any kind of function you can imagine. Green's functions are sometimes also known as weight functions and are essentially mappings from a distribution function to a quantity such as an image, spectrum or combination thereof.

Instructions on how to use this sycout are available under :ref:'geomkern'.

`format`

Default value: Auto-determined from output filename extension
Example line: `format=mat`
Allowed values: `h5`, `hdf5`, `mat`, `out`, `sdt`

Overrides the default setting for what file format to store the output in. If not set, the output file format is determined based on the filename extension of the output file. `h5` and `hdf5` forces HDF5 output. `mat` forces Matlab MAT output. `out` and `sdt` forces SOFT self-descriptive text (SDT) format output (text-based).

`function`

Default value: None
Example line: `function=r12ij`
Allowed values: Any (non-repeating) combination of the characters `1`, `2`, `i`, `j`, `r`, `w`

Sets the shape and contents of the Green's function. A more detailed description of how this option works can be found under *Geometric kernels*.

`output`

Default value: None
Example line: `output=outputfile.mat`
Allowed values: Any non-line-breaking string

Sets the name of the output file. The format of the output file is determined based on the extension part of this setting unless the `format` option has also been specified. *By extension is meant everything that comes after the last dot (.).*

`pixels`

Default value: None
Example line: `pixels=520`
Allowed values: Any positive integer

Sets the number of pixels of the image, i.e. the number of elements in each of the `i` and `j` dimensions. Only required if either `i` or `j` appears in the `function` option.

`stokesparams`

Default value: `no`
Example line: `stokesparams=yes`
Allowed values: `yes` or `no`

If set to `yes`, each of the four Stokes parameters I, U, Q and V will be stored in the Green's function (thereby giving it an extra dimension with four elements). If set to `no`, only the intensity parameter is stored, which is the value commonly measured by spectrometers and cameras.

suboffseti

suboffsetj

Default value: 0
Example line: `suboffseti=20`
Allowed values: Any non-negative integer

Green's functions for images tend to become quite large, and in many cases much of the Green's function is zero and provides no interesting information. In these cases, a subset of the image can be stored so that the correct wide-angle image distortion is still present. These offset parameters specify the offsets in the i and j directions respectively from which the image that is to be stored should start.

subpixels

Default value: *Same as ''pixels''*
Example line: `subpixels=30`
Allowed values: Any positive integer

Specifies the number of pixels in each of the i and j directions of the subset image. Since the subset image must lie within the full image, `suboffseti``+``subpixels` and `suboffsetj``+``subpixels` must both be less than or equal to `pixels`.

9.4.2 image

The `image` sycout generates a camera image.

brightness

Default value: `intensity`
Example line: `brightness=histogram`
Allowed values: `bw`, `histogram`, `intensity`

Specifies how pixels should be colored. With `bw` (for black-and-white), pixels are simply marked if they receive a contribution. Thus, if any radiation hits the pixel during the run, the pixel will contain the value 1 at the end of the run and 0 otherwise.

The `histogram` option specifies that each hit in a pixel should increase the value of the pixel by 1. The radiation intensity reaching the pixel is not considered.

The `intensity` option takes the emitted radiation intensity into account, including spectral effects (if enabled through other options).

includeseparatrix

Default value: `yes`
Example line: `includeseparatrix=no`
Allowed values: `no` and `yes`

Specifies whether or not to include separatrix data from the input magnetic equilibrium data file in the output. By default, it is set to `yes`. If no separatrix data is available, the `separatrix` variable is omitted from the output file.

includewall

Default value: `yes`
Example line: `includewall=no`
Allowed values: `no` and `yes`

Specifies whether or not to include wall data from the input magnetic equilibrium data file in the output. By default, it is set to `yes`. If no wall data is available, the `wall` variable is omitted from the output file.

`name`

Default value: None
Example line: `name=output-file.mat`
Allowed values: Any string allowed by the underlying file system

Specifies the name of the file to which the output will be written. The output is written through the SOFT file interface which means it will be either in a HDF5 file, a Matlab MAT file or a SOFT SDT (Self-Descriptive Text) format. The file format is determined based on the filename extension. For HDF5, use either *.h5* or *.hdf5*, for Matlab MAT use *.mat*, and for SDT any other extension (though *.sdt* is recommended).

`pixels`

Default value: None
Example line: `pixels=300`
Allowed values: Any positive integer

Sets the number of pixels in the image. Images are always square and have the same number of pixels in the x (i) direction as in the y (j) direction.

9.4.3 space3d

The `space3d` can be used to store 3D data about the points of space that contribute to an image. A description about how to use it can be found in space3d.

`output`

Default value: None
Example line: `output=name-of-outputfile.mat`
Allowed values: Any string allowed by the underlying file system

Name of the file to which output should be written. The `space3d` module uses the SOFT file interface, meaning output can be written in either HDF5, Matlab MAT or SOFT SDT (Self-Descriptive Text) format. The format of the output file is determined based on the filename extension. For HDF5 use *.h5* or *.hdf5*, for Matlab MAT use *.mat*, and for SDT use any other extension (though *.sdt* is recommended).

`pixels`

Default value: None
Example line: `pixels=300`
Allowed values: Any positive integer

When `type=pixels`, sets the number of pixels in each direction of the bounding box. A value of for example 100 means that there will be a total of $100 \times 100 \times 100 = 1\,000\,000$ "pixels" in the box.

`point0`

Default value: None
Example line: `point0=.40,-.75,.20`
Allowed values: Any real number; any real number; any real number

Specifies one of the two defining edge points of the bounding box.

`point1`

Default value: None
Example line: `point1=.63,-.15,-.20`
Allowed values: Any real number; any real number; any real number

Specifies one of the two defining edge points of the bounding box.

`type`

Default value: None
Example line: `type=pixels`
Allowed values: `pixels`, `real`

Specifies the type of 3D object to store. `pixels` divides the bounding box into a number of smaller boxes and collects the contribution in each of those (the number of boxes is determined by the `pixels` option). This 3D type is fixed in size and is represented as a simple 3D array.

The `real` type stores the real location of each particle that contributes to the image. This 3D grows in size with the number of particles that hit the detector, and is stored as a sparse matrix. It's usually very difficult to determine the final size of this 3D type, but it gives much more detailed data and can sometimes be the more space-efficient option.

9.4.4 spectrometer

The `spectrometer` sycout stores spectra.

`name`

Default value: None
Example line: `name=spectrum.mat`
Allowed values: Any string allowed by the file system

Name of the output file.

9.4.5 topview

The `topview` sycout generates a top map of the tokamak, showing where in the xy-plane radiation comes from. Note that the image is line-integrated along the z direction, and bright areas in the top view therefore do not necessarily correspond to bright areas in the image.

`brightness`

Default value: `intensity`
Example line: `brightness=histogram`
Allowed values: `bw`, `histogram`, `intensity`

Specifies how pixels should be colored. With `bw` (for black-and-white), pixels are simply marked if they receive a contribution. Thus, if any radiation hits the pixel during the run, the pixel will contain the value 1 at the end of the run and 0 otherwise.

The `histogram` option specifies that each hit in a pixel should increase the value of the pixel by 1. The radiation intensity reaching the pixel is not considered.

The `intensity` option takes the emitted radiation intensity into account, including spectral effects (if enabled through other options).

name

Default value: None

Example line: `name=output-file.mat`

Allowed values: Any string allowed by the underlying file system

Specifies the name of the file to which the output will be written. The output is written through the SOFT file interface which means it will be either in a HDF5 file, a Matlab MAT file or a SOFT SDT (Self-Descriptive Text) format. The file format is determined based on the filename extension. For HDF5, use either *.h5* or *.hdf5*, for Matlab MAT use *.mat*, and for SDT any other extension (though *.sdt* is recommended).

pixels

Default value: None

Example line: `pixels=300`

Allowed values: Any positive integer

Sets the number of pixels in the topview. Topviews are always square and have the same number of pixels in the x (i) direction as in the y (j) direction.

CHAPTER 10

The SDT format

The SDT format (for *SOFT Descriptive Text* or *Self-Descriptive Text* or *Semi-Descriptive Text*) was developed in order to import magnetic field data from systems with no HDF5 or MATLAB support. It is a very simple text-based format without fancy features and with little safety. It is recommended that users stick to HDF5 or MATLAB files whenever possible.

10.1 Basic structure

Just as MATLAB files (and in a sense, HD5 files), SDT files contains a set of variables. Each variable consists of a *header* and a *body*. The header is always one line and specifies the name and dimensions of the variable. The body (which comes on the very next line) is at least one line and contains the data of the variable, in ASCII format. Variables are separated by empty lines.

The header always consists of two integers and a string, all separated by spaces. The first integer specifies the number of rows in the data. The second integer specifies the number of columns in the data (or number of characters, in the case of strings). The name is an ASCII string of arbitrary length (but without any whitespace characters).

Note that strings and numeric variables are, technically, encoded differently. Data should always be readable by a human in a text-editor, meaning that numeric values are converted to their ASCII equivalent, while strings are stored directly without converting every single character to an ASCII digit. There is no indication in the header about which type of data a variable contains and so it is up to the user to read variables using the correct decoder.

Matrices are stored by converting each element to its ASCII equivalent. Elements in the same row are separated by single spaces, while rows are separated by (Unix) newlines (i.e. just one `\n` or `0xA` character).

10.2 Example SDT file

An example SDT file containing three variables is shown below:

```
1 2 maxis
1.688510 0.048496

3 3 someMatrix
1.1 2.2 3.3
4.4 5.5 6.6
7.7 8.8 9.9

1 29 someString
This is an SDT example string
```

CHAPTER 11

Troubleshooting

Images contain very sharp, very bright lines. Are particles with large pitch angles being simulated? If so, there's a numerical issue that could potentially arise. In the calculation of a synchrotron image, the Jacobian for the orbit $J = (\partial R/\partial\rho)(\partial Z/\partial\tau) - (\partial R/\partial\tau)(\partial Z/\partial\rho)$ must be computed. The derivatives with respect to τ are straightforward, but to find the derivatives with respect to ρ (the radius at which the particle is initiated) we must compute one additional orbit, at $\rho + \epsilon$, where ϵ is an arbitrarily small number. Internally, this number is fixed to 10^{-6}, which should be sufficient for most cases. If however the orbits corresponding to ρ and $\rho + \epsilon$ lie on opposite sides of the trapped-passing boundary, this will lead to a large error in the computations of the derivatives which will amplify one particular orbit. Currently, the best approach for fixing this should be to make the value of macro `JACOBIAN_ORBIT_STEP` in `src/include/global.h` smaller.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Index

A

axis
    command line option, 28

B

B0
    command line option, 28
brightness
    command line option, 31, 33

C

charge
    command line option, 25
command line option
    axis, 28
    B0, 28
    brightness, 31, 33
    charge, 25
    cospitch, 25
    debug, 23
    domain_has_outer_wall, 24
    file, 29
    format, 29, 30
    function, 30
    gc_position, 26
    includeseparatrix, 31
    includewall, 31
    interptimestep, 24
    magnetic_field, 24
    major_radius, 28
    mass, 26
    maxtimestep, 24
    minor_radius, 28
    name, 32–34
    nodrifts, 24
    output, 30, 32
    p, 26
    pitch, 26
    pixels, 30, 32, 34
    point0, 32
    point1, 33
    ppar, 26
    pperp, 27
    progress, 24
    r, 27
    rdyn, 27
    safety_factor, 28
    stokesparams, 30
    suboffseti, 30
    suboffsetj, 31
    subpixels, 31
    t, 27
    threads, 25
    tolerance, 25
    type, 33
    useequation, 25
    usetool, 25
    wall, 29
cospitch
    command line option, 25

D

debug
    command line option, 23
domain_has_outer_wall
    command line option, 24

F

file
    command line option, 29
format
    command line option, 29, 30
function
    command line option, 30

G

gc_position
    command line option, 26

I

includeseparatrix
    command line option, 31
includewall
    command line option, 31
interptimestep
    command line option, 24

M

magnetic_field
    command line option, 24
major_radius
    command line option, 28
mass
    command line option, 26
maxtimestep
    command line option, 24
minor_radius
    command line option, 28

N

name
    command line option, 32–34
nodrifts
    command line option, 24

O

output
    command line option, 30, 32

P

p
    command line option, 26
pitch
    command line option, 26
pixels
    command line option, 30, 32, 34
point0
    command line option, 32
point1
    command line option, 33
ppar
    command line option, 26
pperp
    command line option, 27
progress
    command line option, 24

R

r
    command line option, 27
rdyn
    command line option, 27

S

safety_factor
    command line option, 28
stokesparams
    command line option, 30
suboffseti
    command line option, 30
suboffsetj
    command line option, 31
subpixels
    command line option, 31

T

t
    command line option, 27
threads
    command line option, 25
tolerance
    command line option, 25
type
    command line option, 33

U

useequation
    command line option, 25
usetool
    command line option, 25

W

wall
    command line option, 29