SFINCS User Manual

Version 3

Revised May 21, 2015

Contents

1 Overview 4
1.1 Geometry options . . . 4
1.2 GMRES/KSP and preconditioning . . . 4
1.3 `sfincs` vs. `sfincsScan` . . . 5
1.4 Input and Output . . . 5
1.5 Questions, Bugs, and Feedback . . . 6

2 Installation 7
2.1 Requirements . . . 7
2.2 Cloning the repository . . . 8
2.3 Makefiles and environment variables . . . 8
2.4 Setting up `sfincs` on a new system . . . 8
2.5 Compiling . . . 9
2.6 Make test . . . 9

3 Input Parameters 10
3.1 The `general` namelist . . . 10
3.2 The `geometryParameters` namelist . . . 11
3.3 The `speciesParameters` namelist . . . 17
3.4 The `physicsParameters` namelist . . . 19
3.5 The `resolutionParameters` namelist . . . 24
3.6 The `otherNumericalParameters` namelist . . . 26
3.7 The `preconditionerOptions` namelist . . . 29
3.8 The `export_f` namelist . . . 31
3.9 Directives for `sfincsScan` . . . 34
3.10 `PETSc` commands . . . 35

4 Numerical resolution parameters 36
4.1 Relatively unimportant resolution parameters . . . 36
4.2 General suggestions . . . 36
4.3 Convergence testing . . . 37
4.4 Typical resolution requirements . . . 37

5 Specifying and running a computation 38
5.1 Normalizations . . . 38
5.2 Radial coordinates . . . 38
5.3 Trajectory models . . . 40

5.4 Parallelization 41
5.5 Issues with running on 1 processor 41
5.6 Monoenergetic transport coefficients 41

A Details for specific computing systems **42**
A.1 NERSC edison 42
A.2 IPP hydra 42
A.3 Chalmers University - Glenn cluster 42

CHAPTER 1

Overview

The `sfincs` code is a freely available, open-source tool for solving neoclassical-type kinetic problems in nonaxisymmetric or axisymmetric plasmas with nested toroidal flux surfaces. The code solves a drift-kinetic equation for an arbitrary number of species. Optionally, a quasi-neutrality equation can also be solved to obtain the self-consistent variation of the electrostatic potential on a flux surface.

This manual describes "version 3" of `sfincs`. There are two older versions of the code called singleSpecies and multiSpecies.

This document discusses the practical use and operation of the code. For more details about the specific equations implemented, see the version 3 technical documentation available in the `sfincs/docs` directory. Ref [1] gives many details and some early physics results.

Often, the limiting factor for `sfincs` is the ability of `mumps` or `superlu_dist` to factorize the preconditioner matrix. You may therefore find it useful to see the control parameters and error codes in the `mumps` user manual: `http://mumps.enseeiht.fr/doc/userguide_5.0.0.pdf`.

1.1 Geometry options

In `sfincs`, a variety of options are available for the magnetic field geometry. The geometry can be read directly from a `vmec wout` file, or from the `.bc` format Boozer-coordinate data files used at the Max Planck Institute for Plasma Physics (IPP). A general analytic model for the magnetic field is also available, given by equation (3.1), as are several analytic models for LHD and W7-X in which 3 or 4 Fourier components are retained. For more details about geometry options in `sfincs`, see section 3.2.

1.2 GMRES/KSP and preconditioning

At its heart, `sfincs` solves one or more large sparse linear systems $Ax = b$. Here b is a known right-hand side vector, A is a large (often millions $\times$ millions) known sparse matrix, and x is the desired and unknown solution vector. The direct way to solve such systems is to LU-factorize the matrix A into lower- and upper-triangular factors. Once the L and U factors are found, the solution of the linear system for any right-hand side vector can be rapidly obtained. However, even if the original matrix is sparse, the L and U factors are generally not sparse, and so a very large amount of memory can be required for a direct LU-factorization.

An alternative way to solve such large linear systems is with a so-called "Krylov-space" iterative method, which can dramatically reduce the memory required compared to a direct solution. For the non-symmetric matrices that arise in `sfincs`, the preferred Krylov-space algorithm is called GMRES (Generalized Minimal RESidual.) In `sfincs`, the `PETSc` library is used to solve the large systems of equations. `PETSc` calls its family of linear solvers KSP, so in the output of `sfincs` you will see a "KSP residual" reported as GMRES iterates towards the solution.

An important element of Krylov methods is preconditioning. The art of preconditioning is to find a linear operator which has similar eigenvalues to the "true" matrix you would like to invert (or more precisely, to LU-factorize), but which can be inverted faster. If a good preconditioner can be found, the number of GMRES iterations is greatly reduced. Many schemes for preconditioning exist, but the version adopted in `sfincs` is to explicity form and LU-factorize a preconditioning matrix which is similar to the true matrix (but somewhat simpler). There is a basic trade-off: the more similar the preconditioner matrix is to the true matrix, the fewer iterations will be required, but the more time will be required to LU-factorize the preconditioning operator. The usual preconditioner matrix in `sfincs` is obtained by dropping all coupling between grid points in the speed coordinate and dropping coupling between species. The preconditioner matrix need not be a physically accurate or meaningful operator; as long as GMRES converges, the solution obtained will be independent of the preconditioner to whatever tolerance is specified.

1.3 `sfincs` vs. `sfincsScan`

The core fortran part of `sfincs` solves the kinetic equation for each species at a single flux surface, a single value of E_r, and a single set of other parameters. However, often the goal is to determine the ambipolar E_r at one or more surfaces, or to scan some other parameter. For this task, the `sfincsScan` family of `python` scripts is available. For a full list of the types of scans available, see section 3.9

1.4 Input and Output

The input parameters for a `sfincs` computation are specified in a file named `input.namelist`. This file contains both information for the fortran part of `sfincs` (in standard fortran namelist format), as well as special lines beginning with `!ss` which are read by `sfincsScan`. The variables which can be specified in `input.namelist` are detailed in chapter 3. For scans over minor radius, an additional file named `profiles` is used to specify the profiles of density and temperature for each species, as well as the range of radial electric field to consider.

The output from a single `sfincs` computation is saved in `HDF5` format in the file `sfincsOutput.h5`. To browse this file you can enter `h5dump sfincsOutput.h5|less` from the command line. Every array saved in this file is annotated with strings that describe the array dimensions, for example `Ntheta`× `Nzeta`. One of the array dimensions may be `iteration`, which can either indicate the iteration of the Newton solver for a nonlinear calculation, or which right-hand side vector was used when computing a transport matrix. Many of the variables in the output file are also annotated with text that describes their meaning and normalization.

1.5 Questions, Bugs, and Feedback

To report any bugs, provide feedback, or ask questions, contact Matt Landreman at `matt.landreman@gmail.com`

CHAPTER 2

Installation

2.1 Requirements

To compile `sfincs` you need the `PETSc` library (real version, as opposed to complex version) and the `HDF5` library. Even if you will be running `sfincs` in parallel, you only need the serial version of `HDF5`, not the parallel version. `PETSc` is used for iterative solution of large linear and nonlinear systems of equations, and `HDF5` is used for saving output. We have developed and tested `sfincs` with `PETSc` versions 3.2 through 3.5. The commands in `PETSc` often change from version to version, so future versions of `PETSc` may require modifications to the `sfincs` source code.

Although `sfincs` can be run on a single processor, usually you want to run it in parallel. In this case, you need `MPI`, and you need at least one of the two libraries `mumps` or `superlu_dist`. (Note that `superlu_dist`is a parallel library which is different from the serial library `superlu`). Both `mumps` and `superlu_dist` are parallelized libraries for direct solution of large sparse linear systems, which `sfincs` uses to factorize the preconditioning matrix (discussed in secton 1.2). `PETSc` has a built-in serial sparse direct linear solver, but it sometimes gives an error that there is a "zero pivot" when `mumps` and `superlu_dist` have no problem solving the system; therefore you may want to use `mumps` or `superlu_dist` even for serial runs. In our experience, `mumps` requires less memory and time than `superlu_dist` for solving a given linear system.

If you want to load `VMEC wout` files in `netCDF` format, then you need the `netCDF` library. This library is not required for loading ASCII-format `VMEC wout` files. If you want to compile `sfincs` without `netCDF`, then edit `sfincs/fortran/version3/makefile` so that no value is assigned to `USE_NETCDF`.

The plotting routines `sfincsPlot` and `sfincsScanPlot` require `python` 2.X, `numpy`, `scipy`, and `matplotlib`. These `python` libraries are not required by the core fortran part of `sfincs`.

Although older `MATLAB` versions of `sfincs` are included in the `sfincs` repository, `MATLAB` is not required for running the fortran version of `sfincs`.

2.2 Cloning the repository

The source code for `sfincs` is hosted in a `git` repository at `https://github.com/landreman/sfincs`. You obtain the `sfincs` source code by cloning the repository. This requires several steps.

1. Create an account on `github.com`, and sign in to `github`.
2. Go to your account settings page, by clicking the wrench icon on the top right.
3. Click on "SSH keys" on the left, and add an SSH key for the computer you wish to use. To do this, you may wish to read see the "generating SSH keys" guide which is linked to from that page.
4. From a terminal command line in the computer you wish to use, enter
   ```
   git clone git@github.com:landreman/sfincs.git
   ```
   to download the repository.

Any time after you have cloned the repository in this way, you can download future updates to the code by entering `git pull` from any subdirectory within your local copy.

2.3 Makefiles and environment variables

To use `sfincs` you must set the environment variable `SFINCS_SYSTEM`. (For example, using the `bash` shell on the `edison` computer, you would type

```
export SFINCS_SYSTEM=edison
```

at the command line or in your `.bashrc` startup script.) This variable is used in two ways. First, `make` uses this variable to look for the appropriate makefile in the `sfincs/fortran/version3/makefiles` directory. Second, the `SFINCS_SYSTEM` environment variable is used by `sfincsScan` to determine the command for submitting jobs to the system's queue.

You will probably want to add the directory `sfincs/fortran/version3/utils/` to your path. This directory contains the scripts for plotting output and running parameter scans.

2.4 Setting up `sfincs` on a new system

If you are setting up `sfincs` on a new system, one for which there is no file `sfincs/fortran/version3/makefiles/makefile.XXX`, there are several things you need to do.

First, copy one of the existing makefiles, and edit it as appropriate.

Second, you will need to edit `utils/sfincsScan`. Look for the `if` block near the top with sections for `sfincsSystem = edison,hydra`, and `laptop`. Add an analogous block for your system to set the command used to submit jobs, and a `nameJobFile` function.

Third, if you want `make test` to work (see section 2.6), you will need to create files `job.SFINCS_SYSTEM` for each example in the `sfincs/fortran/version3/examples/` directory that you want to include in the tests. You may be able to use the same `job.SFINCS_SYSTEM` file for each example, but for the largest examples, you may want to use different numbers of processes or different queues for different examples.

2.5 Compiling

If your system uses "modules", make sure you have loaded any required modules. (Requirements are discussed in section 2.1). There may be instructions for the specific modules required on your system in the comments in the appropriate makefile `sfincs/fortran/version3/makefiles/makefile.SFINCS_SYSTEM` for your system.

Next, to compile, go to the directory `sfincs/fortran/version3/` and run `make`.

2.6 Make test

To test that your `sfincs` executable is working, you can run `make test` from the `sfincs/fortran/version3/` directory. Doing so will run `sfincs` for some or all of the examples in the `sfincs/fortran/version3/examples/` directories. (The runs will be performed in series if no queueing system is available, otherwise the runs will all be submitted to the queueing system.) After each example completes, several of the output quantities (such as parallel flows and radial fluxes) will be checked, using the `tests_small.py` or `tests_large.py` script in the example's directory.

If you run `make retest` from the `sfincs/fortran/version3/` directory, no new runs of `sfincs` will be performed, but the `tests_small.py` or `tests_large.py` script will be run on any existing `sfincsOutput.h5` files in the `sfincs/fortran/version3/examples/` directories.

CHAPTER 3

Input Parameters

In this chapter we first describe all the parameters which can be included in the `input.namelist` file. Then we list some of the command-line flags associated with `PETSc` which can be useful. Note that all parameters in `input.namelist`, both for `sfincs` and `sfincsScan`, are case-insensitive.

3.1 The `general` namelist

The default values are usually best for the parameters in this namelist.

`RHSMode`
Type: integer
Default: 1
When it matters: Always
Meaning: Option related to the number of right-hand sides (i.e. inhomogeneous drive terms) for which the kinetic equation is solved.

`RHSMode=1`: Solve for a single right-hand side.

`RHSMode=2`: Solve for 3 right-hand sides to get the 3x3 transport matrix. Presently implemented only for 1 species.

`RHSMode=3`: Solve for the 2x2 monoenergetic transport coefficients. When this option is chosen, `Nx` is set to 1 and only 1 species is used.

`outputFileName`
Type: string
Default: “sfincsOutput.h5”
When it matters: Always

Meaning: Name which will be used for the HDF5 output file. If this parameter is changed from the default value, `sfincsScan` will not work.

`saveMatlabOutput`
Type: Boolean
Default: `.false.`
When it matters: Always
Meaning: If this switch is set to true, Matlab m-files are created which store the system matrix, right-hand side, and solution vector. If an iterative solver is used, the preconditioner matrix is also saved. PETSc usually generates an error message if you ask to save Matlab output when the size of the linear system is more then 1400×1400, so usually this setting should be false except for very small test problems.

`MatlabOutputFilename`
Type: string
Default: "sfincsMatrices"
When it matters: Only when `saveMatlabOutput == .true.`
Meaning: Start of the filenames which will be used for Matlab output.

`saveMatricesAndVectorsInBinary`
Type: Boolean
Default: `.false.`
When it matters: Always
Meaning: If this switch is set to true, the matrix, right-hand-side, and solution of the linear system will be saved in PETSc's binary format. The preconditioner matrix will also be saved if `tryIterativeSolver == .true.`

`binaryOutputFilename`
Type: string
Default: "sfincsBinary"
When it matters: Only when `saveMatricesAndVectorsInBinary == .true.`
Meaning: Start of the filenames which will be used for binary output.

`solveSystem`
Type: Boolean
Default: `.true.`
When it matters: Always
Meaning: If this parameter is false, the system of equations will not actually be solved. Sometimes it can be useful to set this parameter to `.false.` when debugging.

3.2 The `geometryParameters` namelist

The parameters in this namelist define the magnetic geometry, and so you will almost certainly want to modify some of these parameters.

---

geometryScheme
Type: integer
Default: 1
When it matters: Always
Meaning: How the magnetic geometry is specified.

`geometryScheme`==1: Use the following 3-helicity model:

$$\begin{aligned} B(\theta,\zeta)/\bar{B} &= (\texttt{B0OverBBar})[1+(\texttt{epsilon\_t})\cos(\theta) \\ &\quad +(\texttt{epsilon\_h})\cos((\texttt{helicity\_l})\theta-(\texttt{helicity\_n})\zeta) \\ &\quad +(\texttt{epsilon\_antisymm}) \\ &\quad \times\sin((\texttt{helicity\_antisymm\_l})\theta-(\texttt{helicity\_antisymm\_n})\zeta)] \end{aligned} \tag{3.1}$$

`geometryScheme`==2: Use a 3-helicity model of the LHD standard configuration at `rN`=0.5.

`geometryScheme`==3: Use a 4-helicity model of the LHD inward-shifted configuration at `rN`=0.5.

`geometryScheme`==4: Use a 3-helicity model of the W7-X standard configuration at `rN`=0.5.

`geometryScheme`==5: Read the `vmec wout` file specified in `equilibriumFile` below. The file can be either ASCII format or `netCDF` format. (`sfincs` will auto-detect the format.).

`geometryScheme`==11: Read the IPP `.bc` format Boozer-coordinate file specified in `equilibriumFile` below. The file is assumed to be stellarator-symmetric.

`geometryScheme`==12: Read the IPP `.bc` format Boozer-coordinate file specified in `equilibriumFile` below. The file is assumed to be stellarator-asymmetric.

---

inputRadialCoordinate
Type: integer
Default: 3
When it matters: When `geometryScheme` == 1, 5, 11, or 12
Meaning: Which radial coordinate to use to specify the flux surface for a single calculation, or to specify the range of flux surfaces for a radial scan. (When `geometryScheme` == 2, 3, or 4, the flux surface used will be `rN`=0.5.) See section 5.2 for more information about radial coordinates.

`inputRadialCoordinate`==0: Use the flux surface specified by `psiHat_wish` for a single run, and use the range specified by `psiHat_min` and `psiHat_max` for radial scans.

`inputRadialCoordinate`==1: Use the flux surface specified by `psiN_wish` for a single run, and use the range specified by `psiN_min` and `psiN_max` for radial scans.

`inputRadialCoordinate`==2: Use the flux surface specified by `rHat_wish` for a single run, and use the range specified by `rHat_min` and `rHat_max` for radial scans.

`inputRadialCoordinate`==3: Use the flux surface specified by `rN_wish` for a single run, and use the range specified by `rN_min` and `rN_max` for radial scans.

No matter which option you pick, the value of all 4 radial coordinates used will be saved in the output `HDF5` file.

`inputRadialCoordinateForGradients`
Type: integer
Default: 2
When it matters: Whenever `RHSMode`==1.
Meaning: Which radial coordinate is used to use to specify the input gradients of density, temperature, and electrostatic potential, i.e. which radial coordinate is used in the denominator of these derivatives. See section 5.2 for more information about radial coordinates.

`inputRadialCoordinateForGradients`==0: Density gradients are specified by `dnHatdpsiHats`, temperature gradients are specified by `dTHatdpsiHats`, a single E_r is specified by `dPhiHatdpsiHat`, and the range of an E_r scan is specified by `dPhiHatdpsiHatMin`-`dPhiHatdpsiHatMax`.

`inputRadialCoordinateForGradients`==1: Density gradients are specified by `dnHatdpsiNs`, temperature gradients are specified by `dTHatdpsiNs`, a single E_r is specified by `dPhiHatdpsiN`, and the range of an E_r scan is specified by `dPhiHatdpsiNMin`-`dPhiHatdpsiNMax`.

`inputRadialCoordinateForGradients`==2: Density gradients are specified by `dnHatdrHats`, temperature gradients are specified by `dTHatdrHats`, a single E_r is specified by `dPhiHatdrHat`, and the range of an E_r scan is specified by `dPhiHatdrHatMin`-`dPhiHatdrHatMax`.

`inputRadialCoordinateForGradients`==3: Density gradients are specified by `dnHatdrNs`, temperature gradients are specified by `dTHatdrNs`, a single E_r is specified by `dPhiHatdrN`, and the range of an E_r scan is specified by `dPhiHatdrNMin`-`dPhiHatdrNMax`.

No matter which option you pick, the gradients with respect to all 4 radial coordinates will be saved in the output `HDF5` file.

`psiHat_wish`
Type: real
Default: -1
When it matters: Only when `inputRadialCoordinate` == 0 and `geometryScheme` == 1, 5, 11, or 12.
Meaning: Requested flux surface for the computation. See section 5.2 for more information about radial coordinates.

`psiN_wish`
Type: real
Default: 0.25

When it matters: Only when `inputRadialCoordinate` == 1 and `geometryScheme` == 1, 5, 11, or 12.
Meaning: Requested flux surface for the computation. See section 5.2 for more information about radial coordinates.

`rHat_wish`
Type: real
Default: -1
When it matters: Only when `inputRadialCoordinate` == 2 and `geometryScheme` == 1, 5, 11, or 12.
Meaning: Requested flux surface for the computation. See section 5.2 for more information about radial coordinates.

`rN_wish`
Type: real
Default: 0.5
When it matters: Only when `inputRadialCoordinate` == 3 and `geometryScheme` == 1, 5, 11, or 12.
Meaning: Requested flux surface for the computation. See section 5.2 for more information about radial coordinates.

`B0OverBBar`
Type: real
Default: 1.0
When it matters: Only when `geometryScheme` == 1. Otherwise, `B0OverBBar` will be set according to the requested `geometryScheme`.
Meaning: Magnitude of the (0,0) Boozer harmonic of the magnetic field strength (equivalent to $\left\langle B^3\right\rangle / \left\langle B^2\right\rangle$), normalized by $\bar{B}$.

`GHat`
Type: real
Default: 3.7481
When it matters: Only when `geometryScheme` == 1. Otherwise, `GHat` will be set according to the requested `geometryScheme`.
Meaning: G is $(c/2)\times$ the poloidal current outside the flux surface. Equivalently, G is the coefficient of $\nabla\zeta_B$ in the covariant representation of $\mathbf{B}$ in terms of Boozer coordinates (θ_B, ζ_B):

$$\mathbf{B}(\psi,\theta_B,\zeta_B) = \beta(\psi,\theta_B,\zeta_B)\nabla\psi + I(\psi)\nabla\theta_B + G(\psi)\nabla\zeta_B. \tag{3.2}$$

`GHat` is G normalized by $\bar{B}\bar{R}$.

`IHat`
Type: real
Default: 0.0
When it matters: Only when `geometryScheme` == 1. Otherwise, `IHat` will be set according to the requested `geometryScheme`.

Meaning: I is $(c/2)\times$ the toroidal current inside the flux surface. Equivalently, I is the coefficient of $\nabla\theta_B$ in the covariant representation of $\mathbf{B}$ in terms of Boozer coordinates (θ_B, ζ_B) in (3.2). `IHat` is I normalized by $\bar{B}\bar{R}$.

`iota`
Type: real
Default: 0.4542
When it matters: Only when `geometryScheme` == 1. Otherwise, `iota` will be set according to the requested `geometryScheme`.
Meaning: Rotational transform (rationalized), equivalent to $1/q$ where q is the safety factor.

`epsilon_t`
Type: real
Default: -0.07053
When it matters: Only when `geometryScheme` == 1.
Meaning: Toroidal variation in B, as defined by (3.1).

`epsilon_h`
Type: real
Default: 0.05067
When it matters: Only when `geometryScheme` == 1.
Meaning: Helical variation in B, as defined by (3.1).

`epsilon_antisymm`
Type: real
Default: 0.0
When it matters: Only when `geometryScheme` == 1.
Meaning: Stellarator-antisymmetric variation in B, as defined by (3.1).

`helicity_l`
Type: integer
Default: 2
When it matters: Only when `geometryScheme` == 1.
Meaning: Poloidal mode number of the helical variation in B, as defined by (3.1).

`helicity_n`
Type: integer
Default: 10
When it matters: Only when `geometryScheme` == 1.
Meaning: Toroidal mode number of the helical variation in B, as defined by (3.1).

`helicity_antisymm_l`
Type: integer
Default: 1
When it matters: Only when `geometryScheme` == 1.

Meaning: Poloidal mode number of the stellarator-antisymmetric variation in B, as defined by (3.1).

---

helicity_antisymm_n
Type: integer
Default: 0
When it matters: Only when `geometryScheme` == 1.
Meaning: Toroidal mode number of the stellarator-antisymmetric variation in B, as defined by (3.1). Note that you can create an up-down asymmetric tokamak by setting `helicity_antisymm_n`=0, `epsilon_h`=0, and `epsilon_antisymm`>0.

---

psiAHat
Type: real
Default: 0.15596
When it matters: Only when `geometryScheme` == 1. Otherwise, `psiAHat` will be set according to the requested `geometryScheme`.
Meaning: `psiAHat` = $\psi_a/(\bar{B}\bar{R}^2)$ where $2\pi\psi_a$ is the toroidal flux at the last closed flux surface.

---

aHat
Type: real
Default: 0.5585
When it matters: Only when `geometryScheme` == 1. Otherwise, `aHat` will be set according to the requested `geometryScheme`.
Meaning: The effective minor radius at the last closed flux surface, in units of $\bar{R}$. The code only uses `aBar` for converting between the various radial coordinates in input and output quantities.

---

equilibriumFile
Type: string
Default: “”
When it matters: Only when `geometryScheme` == 5, 11, or 12.
Meaning: Filename from which to load the magnetic equilibrium, either in `vmec wout` ASCII or `netCDF` format, or IPP `.bc` format.

VMECRadialOption
Type: integer
Default: 1
When it matters: Only when `geometryScheme` == 5.
Meaning: Controls whether the nearest available flux surface in the `vmec wout` file is used, or whether radial interpolation is applied to the `vmec` data to obtain the magnetic field components on the exact surface requested.

`VMECRadialOption`=0: Use the exact `XXX_wish` flux surface requested, by interpolating from the `vmec` radial grid.

`VMECRadialOption`=1: Use a surface that may be slightly different from `XXX_wish` to get the nearest available flux surface from `vmec`’s HALF grid. The components of $\mathbf{B}$ in `vmec` are stored on the half grid, so interpolation is then unnecessary.

`VMECRadialOption`=2: Use a surface that may be slightly different from `XXX_wish` to get the nearest available flux surface from `vmec`'s FULL grid. I'm not sure why you would want this, but the feature is implemented for completeness.

`min_Bmn_to_load`
Type: real
Default: 0.0
When it matters: Only when `geometryScheme` == 5, 11, or 12.
Meaning: Filters the magnetic field read from an input file. Only Fourier modes (m, n) for which $B_{m,n}$ is at least `min_Bmn_to_load` will be included.

3.3 The `speciesParameters` namelist

This namelist defines which species are included in the calculation, along with the density and temperature and gradients thereof. You will definitely want to set the parameters in this namelist.

`Zs`
Type: 1D array of reals
Default: 1.0
When it matters: Always
Meaning: Charges of each species, in units of the proton charge e

---

`mHats`
Type: 1D array of reals
Default: 1.0
When it matters: Always
Meaning: Masses of each species, in units of the reference mass $\bar{m}$

`nHats`
Type: 1D array of reals
Default: 1.0
When it matters: Whenever `RHSMode` == 1
Meaning: Densities of each species, in units of the reference density $\bar{n}$

`THats`
Type: 1D array of reals
Default: 1.0
When it matters: Whenever `RHSMode` == 1
Meaning: Temperatures of each species, in units of the reference temperature $\bar{T}$

`dnHatdpsiHats`
Type: 1D array of reals
Default: 0.0

When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 0
Meaning: Radial density gradients of each species, with respect to the radial coordinate $\hat{\psi}$, normalized by the reference density $\bar{n}$.

`dTHatdpsiHats`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 0
Meaning: Radial temperature gradients of each species, with respect to the radial coordinate $\hat{\psi}$, normalized by the reference temperature $\bar{T}$.

`dnHatdpsiNs`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 1
Meaning: Radial density gradients of each species, with respect to the radial coordinate ψ_N, normalized by the reference density $\bar{n}$.

`dTHatdpsiNs`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 1
Meaning: Radial temperature gradients of each species, with respect to the radial coordinate ψ_N, normalized by the reference temperature $\bar{T}$.

`dnHatdrHats`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 2
Meaning: Radial density gradients of each species, with respect to the radial coordinate $\hat{r}$, normalized by the reference density $\bar{n}$.

`dTHatdrHats`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 2
Meaning: Radial temperature gradients of each species, with respect to the radial coordinate $\hat{r}$, normalized by the reference temperature $\bar{T}$.

---

dnHatdrNs
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 3
Meaning: Radial density gradients of each species, with respect to the radial coordinate r_N, normalized by the reference density $\bar{n}$.

---

dTHatrNs
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 3
Meaning: Radial temperature gradients of each species, with respect to the radial coordinate r_N, normalized by the reference temperature $\bar{T}$.

3.4 The `physicsParameters` namelist

The parameters in this namelist determine which terms are included or excluded in the kinetic equation. You will want to be aware of most of these parameters.

Delta
Type: real
Default: 4.5694e-3
When it matters: Always
Meaning: Roughly speaking, `Delta` is ρ_* at the reference parameters. The precise definition is

$$\begin{aligned} \texttt{Delta} &= \frac{c\bar{m}\bar{v}}{e\bar{B}\bar{R}} \quad \text{(Gaussian units)} \\ &= \frac{\bar{m}\bar{v}}{e\bar{B}\bar{R}} \quad \text{(SI units)}, \end{aligned} \tag{3.3}$$

where c is the speed of light, e is the proton mass, and quantities with a bar are the normalization reference parameters discussed in section 5.1. The default value `Delta` = 4.5694e-3 corresponds to $\bar{B}$ = 1 Tesla, $\bar{R}$ = 1 meter, $\bar{m}$ = proton mass, and $\bar{T}$ = 1 keV.

---

alpha
Type: real
Default: 1.0
When it matters: Whenever `RHSMode` == 1 and E_r is nonzero.
Meaning: `alpha` $= e\bar{\Phi}/\bar{T}$ (both Gaussian and SI units) where e is the proton mass, and $\bar{\Phi}$ and $\bar{T}$ are the normalization reference parameters discussed in section 5.1. The default value `alpha` = 1.0 corresponds to $\bar{T}$ = 1 keV and $\bar{\Phi}$ = 1 kV. The default value `alpha` = 1.0 also corresponds to $\bar{T}$ = 1 eV and $\bar{\Phi}$ = 1 V.

EParallelHat
Type: real
Default: 0.0
When it matters: Whenever `RHSMode` == 1
Meaning: Inductive parallel electric field:

$$\texttt{EParallelHat} = \langle \mathbf{E}\cdot\mathbf{B}\rangle \frac{\bar{R}}{\bar{\Phi}\bar{B}} \tag{3.4}$$

(in both Gaussian and SI units) where $\langle \ldots \rangle$ denotes a flux surface average, $\mathbf{E}$ and $\mathbf{B}$ are the electric and magnetic field vectors, and quantities with a bar are the normalization reference parameters discussed in section 5.1.

---

dPhiHatdpsiHat
Type: real
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 0
Meaning: The derivative of the electrostatic potential with respect to the radial coordinate $\hat{\psi}$, i.e. the radial electric field up to a constant. Notice that exactly 1 of the 4 variables `dPhiHatdpsiHat`, `dPhiHatdpsiN`, `dPhiHatdrHat`, and `dPhiHatdrN` will be used, depending on `inputRadialCoordinateForGradients`.

---

dPhiHatdpsiN
Type: real
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 1
Meaning: The derivative of the electrostatic potential with respect to the radial coordinate ψ_N, i.e. the radial electric field up to a constant. Notice that exactly 1 of the 4 variables `dPhiHatdpsiHat`, `dPhiHatdpsiN`, `dPhiHatdrHat`, and `dPhiHatdrN` will be used, depending on `inputRadialCoordinateForGradients`.

---

dPhiHatdrHat
Type: real
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 2
Meaning: The derivative of the electrostatic potential with respect to the radial coordinate $\hat{r}$, i.e. the radial electric field up to a constant. Notice that exactly 1 of the 4 variables `dPhiHatdpsiHat`, `dPhiHatdpsiN`, `dPhiHatdrHat`, and `dPhiHatdrN` will be used, depending on `inputRadialCoordinateForGradients`.

---

dPhiHatdrN
Type: real
Default: 0.0

When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 3
Meaning: The derivative of the electrostatic potential with respect to the radial coordinate r_N, i.e. the radial electric field up to a constant. Notice that exactly 1 of the 4 variables `dPhiHatdpsiHat`, `dPhiHatdpsiN`, `dPhiHatdrHat`, and `dPhiHatdrN` will be used, depending on `inputRadialCoordinateForGradients`.

`nu_n`
Type: real
Default: 8.330e-3
When it matters: Whenever `RHSMode` == 1
Meaning: Dimensionless collisionality at the reference parameters:

$$\texttt{nu\_n} = \bar{\nu}\frac{\bar{R}}{\bar{v}}, \tag{3.5}$$

where $\bar{R}$ and $\bar{v}$ are the normalization reference parameters discussed in section 5.1, and $\bar{\nu}$ is the dimensional collision frequency at the reference parameters. This frequency is defined as

$$\begin{aligned} \bar{\nu} &= \frac{4\sqrt{2\pi}\bar{n}e^4 \ln\Lambda}{3(4\pi\epsilon_0)^2\sqrt{\bar{m}}\bar{T}^{3/2}} \quad \text{(SI units)} \\ &= \frac{4\sqrt{2\pi}\bar{n}e^4 \ln\Lambda}{3\sqrt{\bar{m}}\bar{T}^{3/2}} \quad \text{(Gaussian units)} \end{aligned} \tag{3.6}$$

where e is the proton charge, $\bar{n}$, $\bar{m}$, and $\bar{T}$ are the normalization reference parameters discussed in section 5.1, and $\ln\Lambda$ is the Coulomb logarithm. The default value `nu_n` = 8.330e-3 corresponds to $\bar{R}$ = 1 meter, $\bar{m}$ = proton mass, $\bar{n}$ = 10^{20} m^{-3}, $\bar{T}$ = 1 keV, and $\ln\Lambda = 17$.

`nuPrime`
Type: real
Default: 1.0
When it matters: Only when `RHSMode` == 3.
Meaning: Dimensionless collisionality used in place of `nHats`, `THats`, `mHats`, `Zs`, and `nu_n` for computing monoenergetic transport coefficients. See section 5.6 for more details.

`EStar`
Type: real
Default: 0.0
When it matters: Only when `RHSMode` == 3.
Meaning: Normalized radial electric field used in place of `dPhiHatdXXX` for computing monoenergetic transport coefficients. See section 5.6 for more details.

`collisionOperator`
Type: integer
Default: 0
When it matters: Always

Meaning: Which collision operator to use:

`collisionOperator` = 0: Full linearized Fokker-Planck operator.

`collisionOperator` = 1: Pitch-angle scattering operator (with no momentum-conserving field term).

`constraintScheme`
Type: integer
Default: -1
When it matters: Always
Meaning: Controls a small number of extra rows and columns of the system matrix which (1) eliminate the null space of the matrix, and (2) ensure that a steady-state solution to the kinetic equation exists even when phase-space volume and/or energy are not conserved. These issues are detailed in section III of Ref [1].

`constraintScheme` = -1: Automatic. If `collisionOperator`==0 then `constraintScheme` will be set to 1, otherwise `constraintScheme` will be set to 2.

`constraintScheme` = 0: No constraints.

`constraintScheme` = 1: 2 constraints per species: $\langle n_1 \rangle = 0$ and $\langle p_1 \rangle = 0$.

`constraintScheme` = 2: `Nx` constraints per species: $\langle f(L = 0) \rangle = 0$ at each x.

You should set `constraintScheme` to -1 unless you know what you are doing.

`includeXDotTerm`
Type: Boolean
Default: `.true.`
When it matters: Whenever `RHSMode` < 3 and the radial electric field is nonzero.
Meaning: Whether or not to include the term in the kinetic equation corresponding to a change in speed proportional to the radial electric field. This term is given by $\dot{x}$ in equation (17) of [1].

`includeElectricFieldTermInXiDot`
Type: Boolean
Default: `.true.`
When it matters: Whenever `RHSMode` < 3 and the radial electric field is nonzero.
Meaning: Whether or not to include the term in the kinetic equation corresponding to a change in pitch angle ξ proportional to the radial electric field. This term is given by the last line of equation (17) of [1].

`useDKESExBDrift`
Type: Boolean
Default: `.false.`

When it matters: Whenever `RHSMode` < 3 and the radial electric field is nonzero.
Meaning: If true, the $\mathbf{E} \times \mathbf{B}$ drift term multiplying $\partial f/\partial\theta$ and $\partial f/\partial\zeta$ is taken to be $\mathbf{E} \times \mathbf{B} \cdot \nabla(\theta \text{ or } \zeta)/\left\langle B^2 \right\rangle$ instead of $\mathbf{E} \times \mathbf{B} \cdot \nabla(\theta \text{ or } \zeta)/B^2$.

`include_fDivVE_term`
Type: Boolean
Default: `.false.`
When it matters: Never
Meaning: Obsolete

`includePhi1`
Type: Boolean
Default: `.false.`
When it matters: Whenever `RHSMode` == 1.
Meaning: If false, no terms involving $\Phi_1 = \Phi - \langle\Phi\rangle$ are included in the kinetic equation, and the quasineutrality equation is not solved. If true, then terms involving Φ_1 are included in the kinetic equation, and the quasineutrality equation is solved at each point on the flux surface. In this latter case, many more quantities are computed and saved in the output file, such as radial fluxes associated with the radial $\mathbf{E} \times \mathbf{B}$ drift.

`nonlinear`
Type: Boolean
Default: `.false.`
When it matters: Whenever `RHSMode` == 1.
Meaning: If true, certain terms that are nonlinear in the unknowns will be included in the kinetic equation. Newton's method will be used to solve the nonlinear system, meaning that the usual linear solve in `sfincs` must be iterated several times. Running with `nonlinear=.true.`requires `includePhi1=.true.`

`includeTemperatureEquilibrationTerm`
Type: Boolean
Default: `.false.`
When it matters: Whenever `RHSMode` == 1.
Meaning: When true, the term $C_{ab}[f_{Ma}, f_{Mb}]$ is included in the kinetic equation, i.e. collisions between the leading-order Maxwellians of different species. This term is nonzero when the temperature is not the same for all species. The resulting contribution to the non-Maxwellian distribution function is isotropic and so does not directly give any parallel or radial transport.

`magneticDriftScheme`
Type: integer
Default: 0
When it matters: Whenever `RHSMode` == 1.
Meaning: This variable controls the poloidal and magnetic drifts, and does not affect the radial magnetic drift. THIS FUNCTIONALITY IS WORK IN PROGRESS, AND RESULTS ARE BUGGY WHEN THIS VARIABLE IS NONZERO.

`magneticDriftScheme` = 0: No poloidal or toroidal magnetic drift.

`magneticDriftScheme` = 1: Use the grad-B and curvature drift, plus the parallel velocity correction $v_\perp^2/(2\Omega_c)\mathbf{b}\mathbf{b}\cdot\nabla\times\mathbf{b}$.

`magneticDriftScheme` = 2: Use the magnetic drift $v_m = (v_{||}/\Omega_c)\nabla(v_{||}\mathbf{b})$.

3.5 The `resolutionParameters` namelist

In this namelist, there are 4 parameters you definitely need to be aware of and adjust: `Ntheta`, `Nzeta`, `Nxi`, and `Nx`. See chapter 4 for details. You may or may not need to adjust `solverTolerance`. The other parameters in this namelist almost never need to be adjusted.

`Ntheta`
Type: integer
Default: 15
When it matters: Always
Meaning: Number of grid points in the poloidal angle. This parameter should be odd; see `forceOddNthetaAndNzet` in this namelist. Memory and time requirements DO depend strongly on this parameter. For stellarator calculations, this parameter can usually be in the range 15-25. For tokamak calculations at low collisionality, the value of this parameter may need to be higher.

`Nzeta`
Type: integer
Default: 15
When it matters: Always
Meaning: Number of grid points in the toroidal angle (per identical segment of the stellarator.) This parameter should be odd; see `forceOddNthetaAndNzeta` in this namelist. Memory and time requirements DO depend strongly on this parameter. Set this parameter to 1 for a tokamak calculation. For stellarator calculations, the value of this parameter required for convergence depends strongly on the collisionality. At high collisionality, this parameter can be several 10s, depending on the complexity of $B(\theta,\zeta)$. At low collisionality, this parameter may need to be many 10s or even > 100 for convergence.

`Nxi`
Type: integer
Default: 16
When it matters: Always
Meaning: Number of Legendre polynomials used to represent the pitch-angle dependence of the distribution function. Memory and time requirements DO depend strongly on this parameter. The value of this parameter required for convergence depends strongly on the collisionality. At high collisionality, this parameter can be as low as 5. At low collisionality, this parameter may need to be many 10s or even > 100 for convergence.

`Nx`
Type: integer
Default: 5
When it matters: Always
Meaning: Number of grid points in energy used to represent the distribution function. Memory and time requirements DO depend strongly on this parameter. This parameter almost always needs to be at least 5. Usually a value in the range 5-8 is plenty for convergence, though in exceptional circumstances you may need to go up to 10-15.

`solverTolerance`
Type: real
Default: 1e-6
When it matters: Whenever `useIterativeLinearSolver` == `.true.`
Meaning: Tolerance used to define convergence of the Krylov solver. This parameter does not affect memory requirements but it does affect the time required for solution somewhat. Occasionally you may want to ease this tolerance to 1e-5 so fewer iterations of the Krylov solver are needed.

`NL`
Type: integer
Default: 4
When it matters: Whenever `collisionOperator` == 0.
Meaning: Number of Legendre polynomials used to represent the Rosenbluth potentials. This number can basically always be 4, since results barely change when `NL` is increased above this value. Memory and time requirements do NOT depend strongly on this parameter.

`NxPotentialsPerVth`
Type: real
Default: 40.0
When it matters: Only when `collisionOperator` == 0 and `xGridScheme` $<$ 5. Since `xGridScheme` = 5 is recommended, this parameter is basically obsolete.
Meaning: Number of grid points in energy used to represent the Rosenbluth potentials for the original implementation of the Fokker-Planck operator described in [2]. Memory and time requirements do NOT depend strongly on this parameter.

`xMax`
Type: real
Default: 5.0
When it matters: Only when `collisionOperator` == 0 and `xGridScheme` $<$ 5. Since `xGridScheme` = 5 is recommended, this parameter is basically obsolete.
Meaning: Maximum normalized speed for the Rosenbluth potential grid for the original implementation of the Fokker-Planck operator described in [2]. Memory and time requirements do NOT depend strongly on this parameter.

`forceOddNthetaAndNzeta`
Type: Boolean

Default: `.true.`
When it matters: Always
Meaning: If true, 1 is added to `Ntheta` any time a run is attempted with even `Ntheta`, and 1 is added to `Nzeta` any time a run is attempted with even `Nzeta`. When false, the even and odd grid points are effectively decoupled so results are unstable. This parameter should be true unless you know what you are doing.

3.6 The `otherNumericalParameters` namelist

The parameters in this namelist are advanced, and the default values are best for routine use of the code.

`thetaDerivativeScheme`
Type: integer
Default: 2
When it matters: Always
Meaning: Discretization scheme for the poloidal angle coordinate theta.

`thetaDerivativeScheme` = 0: Fourier spectral collocation. The differentiation matrix in theta is dense.

`thetaDerivativeScheme` = 1: Finite differences with a 3 point stencil. (The differentiation matrix in theta is tridiagonal, aside from the corners.)

`thetaDerivativeScheme` = 2: Finite differences with a 5 point stencil. (The differentiation matrix in theta is pendadiagonal, aside from the corners.).

The best value for this parameter is usually 2.

`zetaDerivativeScheme`
Type: integer
Default: 2
When it matters: Always
Meaning: Discretization scheme for the toroidal angle coordinate zeta.

`zetaDerivativeScheme` = 0: Fourier spectral collocation. The differentiation matrix in zeta is dense.

`zetaDerivativeScheme` = 1: Finite differences with a 3 point stencil. (The differentiation matrix in zeta is tridiagonal, aside from the corners.)

`zetaDerivativeScheme` = 2: Finite differences with a 5 point stencil. (The differentiation matrix in zeta is pendadiagonal, aside from the corners.).

The best value for this parameter is usually 2.

---

xGridScheme
Type: integer
Default: 5
When it matters: Whenever `RHSMode` is 1 or 2.
Meaning: Discretization scheme for the speed coordinate x.

`xGridScheme` = 1: New orthogonal polynomials with no point at $x = 0$. Original treatment of Rosenbluth potentials.

`xGridScheme` = 2: New orthogonal polynomials with a point at $x = 0$. Original treatment of Rosenbluth potentials.

`xGridScheme` = 3: Uniform finite differences on [0, `xMax`], forcing $f = 0$ at `xMax`. 2-point stencil for interpolating to other grids.

`xGridScheme` = 4: Uniform finite differences on [0, `xMax`], forcing $f = 0$ at `xMax`. 4-point stencil for interpolating to other grids.

`xGridScheme` = 5: New orthogonal polynomials with no point at $x = 0$. New treatment of Rosenbluth potentials.

`xGridScheme` = 6: New orthogonal polynomials with a point at $x = 0$. New treatment of Rosenbluth potentials.

The recommended value for this parameter is 5. When `xGridScheme` = 5 or 6, then the following quantities do not matter: `NxPotentialsPerVth`, `xMax`, and `xPotentialsGridScheme`.

---

xGrid_k
Type: integer
Default: 0
When it matters: Whenever `RHSMode` is 1 or 2 and `xGridScheme` = 1, 2, 5, or 6.
Meaning: For `xGridScheme` = 1, 2, 5, or 6, the distribution function will be represented in terms of polynomials $P_n(x)$ that are orthogonal under the weight $\int_0^\infty dx\ x^k \exp(-x^2) P_n(x) P_m(x) \propto \delta_{n,m}$ where k is an exponent set by the parameter `xGrid_k` here. A good value to use is 0, 1, or 2.

---

xPotentialsGridScheme
Type: integer
Default: 2
When it matters: Whenever `RHSMode` is 1 or 2 and `xGridScheme` is <5. Since the recommended setting for `xGridScheme` is 5, this parameter is rarely relevant.
Meaning: When an explicit grid is used for the Rosenbluth potentials, which grid and interpolation scheme to use.

`xPotentialsGridScheme` = 1: Uniform grid. 5-point stencil for derivatives. 2-point stencil for interpolating to other grids.

`xPotentialsGridScheme` = 2: Uniform grid. 5-point stencil for derivatives. 4-point stencil for interpolating to other grids.

`xPotentialsGridScheme` = 3: Use same grid as for distribution function, so no interpolation needed for the self-collision operator. You must set `xGridScheme` = 3 or 4 to use this setting. Use 2-point stencil for interpolating to other species' grids.

`xPotentialsGridScheme` = 4: Same as option 3, except use a 4-point stencil for interpolating to other species' grids.

The recommended setting is `xPotentialsGridScheme` = 2.

`useIterativeLinearSolver`
Type: Boolean
Default: `.true.`
When it matters: Always
Meaning: If false, a sparse direct solver will be used. The direct solver is faster for small (i.e. low-resolution) problems and always yields a solution (as long as there is sufficient memory). For large (high resolution) problems, the iterative solver will usually be faster and will use much less memory, but it may not always converge.

`whichParallelSolverToFactorPreconditioner`
Type: integer
Default: 1
When it matters: Always
Meaning: Which software package is used to LU-factorize the preconditioner matrix.

`whichParallelSolverToFactorPreconditioner` = 1: Use `mumps` if it is available, otherwise use `superlu_dist`.

`whichParallelSolverToFactorPreconditioner` = 2: Force use of `superlu_dist` even if `mumps` is available.

`PESCPreallocationStrategy`
Type: integer
Default: 1
When it matters: Always
Meaning: This setting changes the estimated number of nonzeros (nnz) used for allocating memory for the system matrix and preconditioner.

`PESCPreallocationStrategy` = 0: Old method with high estimated nnz. This method involves relatively simpler code but uses WAY more memory than necessary.

`PESCPreallocationStrategy` = 1: New method with lower, more precise estimated nnz. This method should use much less memory.

Use `PETSCPreallocationStrategy` = 1 unless you know what you are doing.

3.7 The `preconditionerOptions` namelist

This namelist controls how elements are removed from the "real" matrix in order to obtain the preconditioner matrix. The default values are usually best, but if you find that there are more than 100 iterations of GMRES/KSP, it may be worth adjusting these settings. As long as KSP converges, these parameters should have no impact (to several digits) on the physical outputs such as parallel flows and radial fluxes. Therefore, do not worry about (for example) "dropping coupling between species" in the first parameter below, since full inter-species coupling will be retained in the real equations that are being solved.

---

preconditioner_species
Type: integer
Default: 1
When it matters: Whenever `useIterativeLinearSolver` = `.true.` and there are 2 or more species.
Meaning:
`preconditioner_species` = 0: Keep all coupling between species.

`preconditioner_species` = 1: Drop all coupling between species.

The default value of 1 is recommended, except perhaps at high collisionality where 0 may be preferable.

---

preconditioner_x
Type: integer
Default: 1
When it matters: Whenever `useIterativeLinearSolver` = `.true.` and `RHSMode` = 1 or 2.
Meaning:
`preconditioner_x` = 0: Keep full x coupling.

`preconditioner_x` = 1: Drop everything off-diagonal in x.

`preconditioner_x` = 2: Keep only upper-triangular part in x.

`preconditioner_x` = 3: Keep only the tridiagonal terms in x.

`preconditioner_x` = 4: Keep only the diagonal and superdiagonal in x.

The default value of 1 is strongly recommended, except perhaps at high collisionality where 0 may be preferable.

`preconditioner_x_min_L`
Type: integer
Default: 0
When it matters: Whenever `useIterativeLinearSolver` = `.true.` and `RHSMode` = 1 or 2 and `preconditioner_x` > 0.
Meaning: The x structure of the matrix will only be simplified for Legendre index L is $\geq$ this value. Set `preconditioner_x_min_L` = 0 to simplify the matrix for every L. Recommended values are 0, 1, or 2.

`preconditioner_theta`
Type: integer
Default: 0
When it matters: Whenever `useIterativeLinearSolver` = `.true.`
Meaning:
`preconditioner_theta` = 0: Keep full θ coupling.

`preconditioner_theta` = 1: Use a 3-point finite difference stencil for $d/d\theta$.

`preconditioner_theta` = 2: Drop all θ coupling.

`preconditioner_theta` = 3: Replace $d/d\theta$ with the identity matrix.

The default value of 0 is strongly recommended.

`preconditioner_theta_min_L`
Type: integer
Default: 0
When it matters: Whenever `useIterativeLinearSolver` = `.true.` and `preconditioner_theta` > 0.
Meaning: The θ structure of the matrix will only be simplified for Legendre index L is $\geq$ this value. Set `preconditioner_theta_min_L` = 0 to simplify the matrix for every L.

`preconditioner_zeta`
Type: integer
Default: 0
When it matters: Whenever `useIterativeLinearSolver` = `.true.`
Meaning:
`preconditioner_zeta` = 0: Keep full ζ coupling.

`preconditioner_zeta` = 1: Use a 3-point finite difference stencil for $d/d\zeta$.

`preconditioner_zeta` = 2: Drop all ζ coupling.

`preconditioner_zeta` = 3: Replace $d/d\zeta$ with the identity matrix.

The default value of 0 is strongly recommended.

`preconditioner_zeta_min_L`
Type: integer
Default: 0
When it matters: Whenever `useIterativeLinearSolver` = `.true.` and `preconditioner_zeta` > 0.
Meaning: The ζ structure of the matrix will only be simplified for Legendre index L is $\geq$ this value. Set `preconditioner_zeta_min_L` = 0 to simplify the matrix for every L.

`preconditioner_xi`
Type: integer
Default: 1
When it matters: Whenever `useIterativeLinearSolver` = `.true.`
Meaning:
`preconditioner_xi` = 0: Keep full ξ coupling.

`preconditioner_xi` = 1: Drop terms that are ± 2 rows from the diagonal in ξ, so the preconditioner matrix becomes tridiagonal in ξ. (Normally the preconditioner matrix is pentadiagonal in ξ.)

Either a setting of 0 or 1 can be good for this parameter.

`reusePreconditioner`
Type: Boolean
Default: `.true.`
When it matters: Only when `nonlinear` = `.true.`
Meaning: If true, the nonlinear term will not be included in the preconditioner matrix, meaning the preconditioner matrix is the same at every iteration, and so the preconditioner matrix only needs to be LU-factorized once. If false, the preconditioner matrix for the Jacobian will be different at each iteration of the Newton solve, so the preconditioner needs to be LU-factorized at each iteration. The nonlinear term also introduces a lot of nonzeros into the preconditioner matrix, so setting `reusePreconditioner =.true.` not only dramatically reduces the time required for a nonlinear calculation, but also the memory required.

3.8 The `export_f` namelist

This namelist controls whether and how the distribution function is saved in `sfincsOutput.h5`. For each of the 4 coordinates (θ, ζ, x, ξ), the distribution function can be given with the same discretization used for solving the kinetic equation, or you can interpolate to a different grid/discretization. For all available settings, the distribution function will be reported on a tensor product grid in the 4 coordinates.

`export_full_f`
Type: Boolean
Default: `.false.`
When it matters: Always
Meaning: Whether or not to save the full distribution function (the sum of the leading-order Maxwellian and the departure from it) in the output file.

`export_delta_f`
Type: Boolean
Default: `.false.`
When it matters: Always
Meaning: Whether or not to save the departure from a Maxwellian distribution function in the output file.

`export_f_theta_option`
Type: integer
Default: 2
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`
Meaning: Controls which grid in θ is used for exporting the distribution function.

`export_f_theta_option` = 0: Report the distribution function on the original θ grid (with `Ntheta` points) used for solving the kinetic equation.

`export_f_theta_option` = 1: Interpolate to a different grid, specified by `export_f_theta`. Linear interpolation will be used. No sorting of the requested values is performed.

`export_f_theta_option` = 2: Do not interpolate. Use the values of the θ grid that are closest to the values requested in `export_f_theta`. Values of θ will be in increasing order. If multiple requested values are close to the same grid point, the number of points returned will be less than the number of points requested.

For all of these options, you can see `export_f_theta` in `sfincsOutput.h5` for the actual grid used in the end.

`export_f_zeta_option`
Type: integer
Default: 2
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`
Meaning: Controls which grid in ζ is used for exporting the distribution function.

`export_f_zeta_option` = 0: Report the distribution function on the original ζ grid (with `Nzeta` points) used for solving the kinetic equation.

`export_f_zeta_option` = 1: Interpolate to a different grid, specified by `export_f_zeta`. Linear interpolation will be used. No sorting of the requested values is performed.

`export_f_zeta_option` = 2: Do not interpolate. Use the values of the ζ grid that are closest to the values requested in `export_f_zeta`. Values of ζ will be in increasing order. If multiple requested values are close to the same grid point, the number of points returned will be less than the number of points requested.

For all of these options, you can see `export_f_zeta` in `sfincsOutput.h5` for the actual grid used in the end.

`export_f_theta`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`, and `export_f_theta_option` > 0.
Meaning: Values of θ on which you want to save the distribution function. modulo$(\ldots, 2\pi)$ will be applied. See `export_f_theta_option` for details

---

`export_f_zeta`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`, and `export_f_zeta_option` > 0.
Meaning: Values of ζ on which you want to save the distribution function. modulo$(\ldots, 2\pi/\texttt{NPeriods})$ will be applied. See `export_f_zeta_option` for details

---

`export_f_xi_option`
Type: integer
Default: 1
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`
Meaning: Controls which discretization in ξ is used for exporting the distribution function.

`export_f_xi_option` = 0: Report the distribution function as amplitudes of `Nxi` Legendre polynomials, as used internally by `sfincs` for solving the kinetic equation.

`export_f_xi_option` = 1: Report the distribution function on the values of ξ specified by `export_f_xi`. No sorting of the requested values is performed.

`export_f_xi`
Type: 1D array of reals
Default: 0.0
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`, and `export_f_xi_option` = 1.
Meaning: Values of ξ on which you want to save the distribution function. Values must lie in the

range $[-1, 1]$.

---

export_f_x_option
Type: integer
Default: 0
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`
Meaning: Controls which grid in $x = v/\sqrt{2T/m}$ is used for exporting the distribution function.

`export_f_x_option` = 0: Report the distribution function on the original x grid (with `Nx` points) used for solving the kinetic equation.

`export_f_x_option` = 1: Interpolate to a different grid, specified by `export_f_x`. Polynomial spectral interpolation will be used. No sorting of the requested values is performed.

`export_f_x_option` = 2: Do not interpolate. Use the values of the internal x grid that are closest to the values requested in `export_f_x`. Values of x will be in increasing order. If multiple requested values are close to the same grid point, the number of points returned will be less than the number of points requested.

For all of these options, you can see `export_f_x` in `sfincsOutput.h5` for the actual grid used in the end.

---

export_f_x
Type: 1D array of reals
Default: 1.0
When it matters: Whenever `export_full_f` or `export_delta_f` is `.true.`, and `export_f_x_option` > 0.
Meaning: Values of x on which you want to save the distribution function. Values must be ≥ 0.

3.9 Directives for `sfincsScan`

The parameters for `sfincsScan` begin with the code `!ss` and so are not read by the fortran part of `sfincs`. These parameters matter only when `sfincsScan` is called and are all ignored when `sfincs` is executed directly. These parameters can appear anywhere in the `input.namelist` file, in any namelist or outside of any namelist. Note that `sfincsScan` parameters do not have defaults, unlike fortran namelist parameters.

scanType
Type: integer
When it matters: Any time `sfincsScan` is called.
Meaning: Which type of scan will be run when `sfincsScan` is called.

`scanType` = 1: Convergence scan. (Scan the parameters in the resolutionParameters namelist.)

`scanType` = 2: Scan of E_r.

`scanType` = 3: Scan any one input parameter that takes a numeric value.

`scanType` = 4: Scan radius, taking the density and temperature profiles from the `profiles` file. In this type of scan, the same radial electric field is used at every radius. See `sfincs/fortran/utils/profiles.XXX` for examples.

`scanType` = 5: Scan radius, and at each radius, scan E_r. Density and temperature profiles are again taken from the `profiles` file; see `sfincs/fortran/utils/profiles.XXX` for examples.

`scanType` = 21: Read in a list of requested runs from a file `runspec.dat`. See `sfincs/fortran/utils/sfincsScan_21` for an example file.

3.10 `PETSc` commands

Command-line flags can be used to modify the behavior of any `PETSc` application, including `sfincs`. There are hundreds of `PETSc` options, and a list can be obtained by running with the command-line flag `-help`. Here we list some of the more useful options.

`-help`
Meaning: Dumps a list of available command-line options to stdout.

`-ksp_view`
Meaning: Dumps detailed information to stdout related to the linear solver.

`-ksp_gmres_restart` `<integer>`
Meaning: After how many iterations will GMRES restart. Default is 2000. The convergence of GMRES slows every time a restart occurs, but restarts also free up memory.

`-mat_mumps_icntl_4` `<integer>`
Meaning: How much diagnostic information will be printed by `mumps`. Default is 0. Set to 2 or 3 to print out useful diagnostic information about the memory required for factorizing the preconditioner.

CHAPTER 4

Numerical resolution parameters

Results from `sfincs` should only be believed if you are confident they are converged with respect to the numerical resolution parameters `Ntheta`, `Nzeta`, `Nxi`, and `Nx`. That is, you want to be sure the physics output of the code does not change significantly when any of these parameters are increased. The values of `Ntheta`, `Nzeta`, `Nxi`, and `Nx` required for convergence depend strongly on the magnetic geometry and collisionality, with modest dependence also on the radial electric field. It is strongly recommended that you test for convergence with respect to `Ntheta`, `Nzeta`, `Nxi`, and `Nx` whenever beginning `sfincs` calculations for a new scenario.

Note that "convergence" in this sense (convergence with respect to resolution parameters assuming the discretized system is solved exactly) is separate from the convergence of GMRES/KSP.

4.1 Relatively unimportant resolution parameters

There are several resolution parameters which are almost never the limiting factor for convergence, and so which almost never need to be adjusted. These parameters and good values for them are `NL=4`, `solverTolerance` = 10^{-6}, `xMax=5.0`, and `NxPotentialsPerVth=40.0`. The latter two of these parameters are in fact ignored for the recommended and default `xGridMode` setting, 5.

4.2 General suggestions

The time and memory requirements of the code increase significantly when `Ntheta`, `Nzeta`, `Nxi`, or `Nx` are increased. Therefore, you probably want to only scan one of these four parameters at a time (rather than increasing two or more of them simultaneously) when testing for convergence. (This recommended approach is the one taken in `sfincsScan` automated convergence scans, discussed in section 4.3)

When the mean-free-path is shorter than the parallel length scale of the equilibrium, (`nuPrime` $\geq$ 1), the parameters required for convergence do not depend much on collisionality. In the opposite limit in which the mean-free-path is longer than the parallel length scale of the equilibrium, (`nuPrime` ≤ 1) values of `Nzeta` and `Nxi` required for convergence increase dramatically as collisionality decreases. The required value of `Ntheta` increases as well, but often not quite as dramatically. The `Nx` required for convergence does not depend much on collisionality, though typically the required value increases slightly with collisionality at high collisionality

The `Nx` required for convergence may need to increase slightly with the number of species. Typically you can expect to use `Nx`=5-8.

The resolution parameters do not need to vary much with the radial electric field as long as the electric field is below about 1/3 of the resonant value. (In the notation of [1], when $E_* < 1/3$). For almost all experimentally relevant situations (except for HSX where T_i/T_e is extremely small), the electric field is far below the resonance, in which case you should not need to vary the resolution parameters with the electric field. However, if you do approach the E_r resonance, `Nx` will likely need to be increased.

4.3 Convergence testing

4.4 Typical resolution requirements

CHAPTER 5

Specifying and running a computation

5.1 Normalizations

Dimensional quantities in `sfincs` are normalized to "reference" values that are denoted by a bar:
$\bar{B}$ = reference magnetic field, typically 1 Tesla.
$\bar{R}$ = reference length, typically 1 meter.
$\bar{n}$ = reference density, typically 10^{19} m^{-3}, 10^{20} m^{-3}, or something similar.
$\bar{m}$ = reference mass, typically either the mass of hydrogen or deuterium.
$\bar{T}$ = reference temperature in energy units, typically 1 eV or 1 keV.
$\bar{v} = \sqrt{2\bar{T}/\bar{m}}$ = thermal speed at the reference temperature and mass
$\bar{\Phi}$ = reference electrostatic potential, typically 1 V or 1 kV.

You can choose any reference parameters you like, not just the values suggested here. However, if you use a `vmec` or `.bc` magnetic equilibrium by choosing `geometryScheme` = 5, 11, or 12, then you MUST use $\bar{B}$ = 1 Tesla and $\bar{R}$ = 1 meter. The code "knows" about the reference values only through the 3 combinations `Delta`, `alpha`, and `nu_n` in the `physicsParameters` namelist.

Normalized quantities are denoted by a "hat". Taking the magnetic field as an example, $\hat{B} = B/\bar{B}$, where $\hat{B}$ is called `BHat` in the fortran code and `HDF5` output file.

5.2 Radial coordinates

A variety of flux-surface label coordinates are used in other codes and in the literature. One common choice (used in `vmec`) is ψ_N, the toroidal flux normalized to its value at the last closed flux surface. Another common choice is an "effective normalized minor radius" r_N, defined by $r_N = \sqrt{\psi_N}$. For gradients of density, temperature, and electrostatic potential (i.e. the radial electric field), it is useful to use a dimensional local minor radius $r = r_N a$, where a is some measure of the plasma effective outer minor radius. Finally, one could also use ψ directly. For maximum flexibility, `sfincs` permits any of these four radial coordinates to be used, and different radial coordinates can be used in different aspects of a given computation. Output quantities which depend on the radial coordinate, such as radial fluxes, are often given with respect to all radial coordinates. In `sfincs`, the four radial coordinates are named as follows:

`psiHat` = $\hat{\psi}$ is the toroidal flux (divided by 2π), normalized by $\bar{B}\bar{R}^2$.

`psiN` = ψ_N is the toroidal flux normalized by its value at the last closed flux surface.

`rHat` = $\hat{r}$ is defined as $\texttt{aHat}\sqrt{\texttt{psiN}}$, where `aHat` is an effective minor radius of the last closed flux surface normalized by $\bar{R}$.

`rN` = r_N is defined as $\sqrt{\texttt{psiN}}$.

These four radial coordinates are identified by the numbers 0, 1, 2, and 3 respectively.

When setting up a run, you can make several independent choices for radial coordinates. One parameter you select is `inputRadialCoordinateForGradients` in the `geometryParameters` namelist. This parameter controls which coordinate is used to specify the gradients. The possible values of `inputRadialCoordinateForGradients` are:

0: Use derivatives with respect to `psiHat`: Density gradients are specified using `dnHatdpsiHats`, temperature gradients are specified using `dTHatdpsiHats`, a single E_r is specified using `dPhiHatdpsiHat`, and a range of E_r for a scan is specified using `dPhiHatdpsiHatMin`-`dPhiHatdpsiHatMax`.

1: Use derivatives with respect to `psiN`: Density gradients are specified using `dnHatdpsiNs`, temperature gradients are specified using `dTHatdpsiNs`, a single E_r is specified using `dPhiHatdpsiN`, and a range of E_r for a scan is specified using `dPhiHatdpsiNMin`-`dPhiHatdpsiNMax`.

2: Use derivatives with respect to `rHat`: Density gradients are specified using `dnHatdrHats`, temperature gradients are specified using `dTHatdrHats`, a single E_r is specified using `dPhiHatdrHat`, and a range of E_r for a scan is specified using `dPhiHatdrHatMin`-`dPhiHatdrHatMax`.

3: Use derivatives with respect to `rN`: Density gradients are specified using `dnHatdrNs`, temperature gradients are specified using `dTHatdrNs`, a single E_r is specified using `dPhiHatdrN`, and a range of E_r for a scan is specified using `dPhiHatdrNMin`-`dPhiHatdrNMax`.

Another choice involving radial coordinates is how to specify the flux surface for the computation. This choice is made using the parameter `inputRadialCoordinate` in the `geometryParameters` namelist, which is again an integer from 0 to 3, and this parameter need not be the same as `inputRadialCoordinateForGradients`. An extra complication with specifying the flux surface is that the magnetic equilibrium file will contain data on a finite number of surfaces, and you may wish to use one of these surfaces. For this reason, the parameters for specifying the flux surface have `_wish` appended to the name. In other words, the allowed values for `inputRadialCoordinate` are:

0: Specify the flux surface using `psiHat_wish`.

1: Specify the flux surface using `psiN_wish`.

2: Specify the flux surface using `rHat_wish`.

3: Specify the flux surface using `rN_wish`.

When using `geometryScheme` == 11 or 12, `sfincs` will always shift the "wish" value so it matches an available surface in the magnetic equilibrium file. For `geometryScheme` == 5, the `VMECRadialOption` parameter lets you can choose whether to shift to the nearest surface in the magnetic equilibrium file, or to interpolate the `vmec` data onto the exact value of radius you specify.

If you perform a radial scan, then there is a third choice you can make: which radial coordinate to use in the `profiles` file. This choice is made with an integer 0, 1, 2, or 3 in the first non-comment line of the `profiles` file. The radial coordinate used in the `profiles` file need not be the same as either `inputRadialCoordinate` or `inputRadialCoordinateForGradients`. Note however that the maximum and minimum radial electric field specified in the `profiles` file must be defined as the derivative of the electrostatic potential with respect to the radial coordinate `inputRadialCoordinateForGradients`.

For more details about the behavior of `inputRadialCoordinate`, `inputRadialCoordinateForGradie` and `VMECRadialOption`, see section 3.2.

5.3 Trajectory models

As discussed in [1], one of the capabilities of `sfincs` is to compare various models for the terms in the kinetic equation involving E_r. These variations of the kinetic equation are called "trajectory models" in [1]. The relevant terms in the kinetic equation can be turned off and on by certain Boolean parameters in the `physicsParameters` namelist. The models described in [1] are selected as follows:

Full trajectories:
```
includeXDotTerm = .true.
includeElectricFieldTermInXiDot = .true.
useDKESExBDrift = .false.
```

Partial trajectories:
```
includeXDotTerm = .false.
includeElectricFieldTermInXiDot = .false.
useDKESExBDrift = .false.
```

DKES trajectories:
```
includeXDotTerm = .false.
includeElectricFieldTermInXiDot = .false.
useDKESExBDrift = .true.
```

5.4 Parallelization

Choosing the number of nodes and procs.

5.5 Issues with running on 1 processor

5.6 Monoenergetic transport coefficients

By setting `RHSMode=3`, `sfincs` can be run in a mode where it solves the same kinetic equation (prior to discretization) as `dkes` and other monoenergetic codes. When `RHSMode=3`, the values of `Zs`, `THats`, `nHats`, `mHats`, `nu_n`, and `dPhiHatdXXX` are all ignored. Instead, the collisionality is set by `nuPrime`, and the radial electric field is set by `EStar`. The first of these quantities is the dimensionless collisionality

$$\texttt{nuPrime} = \frac{(G+\iota I)\nu}{v B_0} \tag{5.1}$$

where G and I are defined in (3.2), $\iota = 1/q$ is the rotational transform, v is the speed at which the monoenergetic calculation is being performed, and B_0 is the (0,0) Fourier harmonic of B with respect to the Boozer poloidal and toroidal angles. To do: How is ν defined here? The normalized radial electric field is

$$\texttt{EStar} = \frac{cG}{\iota v B_0} \frac{d\Phi}{d\psi} \tag{5.2}$$

(Gaussian units). When `RHSMode` == 1, `nuPrime` and `EStar` are ignored. To do: Should be change the behavior of RHSMode=2 so it uses nuPrime and EStar instead of nu_n?

APPENDIX A

Details for specific computing systems

A.1 NERSC edison

A.2 IPP hydra

A.3 Chalmers University - Glenn cluster

References

[1] M. Landreman, H. M. Smith, A. Mollén, and P. Helander. *Phys. Plasmas*, **21**, 042503 (2014).

[2] M. Landreman and D. Ernst. *J. Comp. Phys.*, **243**, 130 (2013).