SFINCS User Manual

Version 3

Revised July 26, 2016

Contents

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. As with other neoclassical codes, the input information used by `sfincs` is the equilibrium magnetic geometry together with the density, radial density gradient, temperature, and radial temperature gradient of each species. The code then solves a drift-kinetic equation for each species, yielding the (gyro-angle averaged) distribution function. Moments of the distribution function are computed such as the parallel flow, bootstrap current, radial particle flux, radial heat flux, and variation of the density over a flux surface. These moments are all saved in the output file, and if you wish, you can also save the distribution function itself. Optionally, a quasi-neutrality equation can be solved at the same time as the drift-kinetic equations, yielding the self-consistent variation of the electrostatic potential on a flux surface.

The kinetic equations solved in `sfincs` have four independent variables: poloidal angle θ, toroidal angle ζ, normalized speed $x = v/v_{thermal}$, and pitch angle $\xi = v_{||}/v$. The third velocity coordinate (gyro-angle) does not appear since gyro-averaged equations are solved. The flux surface label (radius) coordinate is only a parameter, rather than a full independent variable, since a radially local approximation is made.

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 the libraries `mumps` or `superlu_dist` to factorize the preconditioner matrix, discussed in section 1.4. 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`.

This manual describes "version 3" of `sfincs`. To preserve previous versions of the code that have been used for publications, two older versions of the code called singleSpecies and multiSpecies are also present in the repository. For all versions, both MATLAB and fortran editions exist which are independent of each other. The MATLAB editions exist primarily for debugging the fortran editions. The same algorithms are implemented in the two different languages, and for identical input parameters, the matrices and output quantities from the different editions should agree to several significant digits (roughly within the solver tolerance.) Any significant differences between the matrices and output of the MATLAB and fortran editions can be used to identify a bug. The MATLAB versions are serial whereas the fortran versions are parallelized, so the fortran versions are significantly faster and can access much more memory. For realistic experimental geometry and collisionality, the resolution (and hence memory) requirements will mean you will need to use the fortran edition.

1.1 Features

- Both self-species and inter-species collisions are treated using the most accurate linear operator available, the full linearized Fokker-Planck collision operator, with no approximation of the field-particle term or expansion in mass ratio. This collision operator conserves mass, momentum, and energy.

- Realistic experimental geometry can be simulated using an interface to `vmec`. Analytic model equilibria can also be used.

- Full coupling in the speed (or equivalently, kinetic energy) coordinate is retained, i.e. no monoenergetic approximation is made. However, if desired, `sfincs` can also be run in monoenergetic mode to compare with older codes.

- The code is formulated to permit solution of a wide variety of kinetic equations, whether or not phase space volume and/or energy are conserved, so individual terms can be turned on or off to examine their effect.

- A variety of models for terms involving the radial electric field are available to allow comparison between models.

- You can choose to include or not include the poloidal and toroidal magnetic drifts in the kinetic equation.

- The code takes advantage of modern algorithms (GMRES) and parallelized libraries (`PETSc`, `superlu_dist`, and `mumps`).

- Efficient representation of velocity space is achieved using a pseudospectral method based upon non-classical orthogonal polynomials. [2]

- The electrostatic potential can either be taken to be constant or non-constant on a flux surface.

- Optional nonlinear terms in the kinetic equation (involving both the non-Maxwellian distribution function and poloidal/toroidal electric field) can be included using Newton's method.

1.2 Limitations

- The `sfincs` code is radially local, in the sense that it approximates the radial derivative of the distribution function $\partial f/\partial\psi$ by the derivative of a Maxwellian flux function $\partial f_M(\psi,x)/\partial\psi$. This approximation is important for reducing the otherwise 5D space of independent variables

$(\psi, \theta, \zeta, x, \xi)$ to a 4D space (θ, ζ, x, ξ). As a result, `sfincs` cannot compute certain finite-orbit-width effects that occur when the radial extent of the particle orbits between bounces or transits is not small compared to the scale of radial variation in the equilibrium. Such finite orbit width effects are significant near the magnetic axis, and in strong transport barriers, such as the pedestal of a tokamak H-mode.

- Turbulence is neglected. There are good theoretical reasons to expect that the neoclassical effects computed by `sfincs` should decouple from turbulence, as detailed in [3]. However, this argument relies on an expansion in $\rho_* \ll 1$, and so may break down in some circumstances when ρ_* is not sufficiently small.

- It is assumed that nested toroidal magnetic surfaces exist. Thus, the code cannot accurately model regions of stochastic field, magnetic islands, or open field lines.

1.3 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. The primary switch for controlling the magnetic geometry in `sfincs` is the `geometryScheme` parameter in the `geometryParameters` input namelist. For more details about geometry options in `sfincs`, see section 3.2.

1.4 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.

1.5 `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. Using these scripts, it is also possible to scan other variables in the input file, and in particular, to scan the resolution parameters to ensure the physical output quantities are numerically converged. For a full list of the types of scans available, see section 3.9

1.6 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.

1.7 Questions, Bugs, and Feedback

We enthusiastically welcome any contributions to the code or documentation. For write permission to the repository, or 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_distis 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 1.4). 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.

To eliminate the need to set the environment variable and path as described above at each login session, you may find it convenient to set both in your startup script, such as `.bashrc`. In this startup script you may also want to load any modules needed by `sfincsScan` such as `python`, `numpy`, `scipy`, and `matplotlib` (if your computing system uses modules).

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 -j`. (The `-j` flag means compiliation will be done in parallel, i.e. faster.)

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.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.py` script will be run on any existing `sfincsOutput.h5` files in the `sfincs/fortran/version3/examples/` directories.

The `make test` functionality relies on several environment variables set in the `sfincs/fortran/version3/makefiles/makefile.SFINCS_SYSTEM` file, as well as on the `job.SFINCS_SYSTEM` files in each example subdirectory. If you experience problems with `make test`, there is a good chance that one of these files needs modification.

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 3×3 transport matrix. Presently implemented only for 1 species.

`RHSMode` = 3: Solve for the 2×2 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 vector of the linear system will be saved in PETSc's binary format. The preconditioner matrix will also be saved if `useIterativeLinearSolver` == `.true.`. These matrices and vectors are not very interesting for routine use of the code, only for code development and debugging. Regardless of how this parameter is set, the physically interesting input and output quantities will be saved in a separate `HDF5` file.

`binaryOutputFilename`
Type: string
Default: "sfincsBinary"
When it matters: Only when `saveMatricesAndVectorsInBinary` == `.true.`.
Meaning: Start of the filenames which will be used for binary output of the system matrices, right-hand-side vectors, and solution vectors. These matrices and vectors are not very interesting for routine use of the code, only for code development and debugging. Regardless of how this parameter is set, the physically interesting input and output quantities will be saved in a separate `HDF5` file.

`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:

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

(All the variables in this formula are discussed later in this namelist.)

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. (Regardless of the value of this parameter, 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: 4
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.

inputRadialCoordinateForGradients==4: Same as inputRadialCoordinateForGradients==2, except Er is used instead of dPhiHatdrHat. Thus, density gradients are specified by dnHatdrHats, temperature gradients are specified by dTHatdrHats, a single E_r is specified by Er, and the range of an E_r scan is specified by ErMin-ErMax.

No matter which option you pick, the gradients with respect to all 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 $\langle B^3 \rangle / \langle B^2 \rangle$), 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. Note that only one of the four parameters `dnHatdpsiHats`, `dnHatdpsiNs`, `dnHatdrHats`, or `dnHatdrNs` will be used, depending on the value of `inputRadialCoordinateForGradients` in the `geometryParameters` namelist. Similarly, only one of the four parameters `dTHatdpsiHats`, `dTHatdpsiNs`, `dTHatdrHats`, or `dTHatdrNs` will be used.

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 $\bar{n}$.

---

dTHatdrNs
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 $\bar{T}$.

---

withAdiabatic
Type: Boolean
Default: `.false.`
When it matters: Whenever `RHSMode` == 1 and `includePhi1` == `.true.`
Meaning: If this parameter is `.true.`, an adiabatic species is added into the quasineutrality equation (but has no other effect).

---

adiabaticZ
Type: real
Default: -1.0
When it matters: Whenever `RHSMode` == 1 and `includePhi1` == `.true.` and `withAdiabatic`

== .true.
Meaning: Charge of adiabatic species, in units of the proton charge e.

---

adiabaticMHat
Type: real
Default: 5.44617e-4
When it matters: Whenever RHSMode == 1 and includePhi1 == .true. and withAdiabatic == .true.
Meaning: Mass of adiabatic species, in units of the reference mass $\bar{m}$.

---

adiabaticNHat
Type: real
Default: 1.0
When it matters: Whenever RHSMode == 1 and includePhi1 == .true. and withAdiabatic == .true.
Meaning: Density of adiabatic species, in units of the reference density $\bar{n}$.

---

adiabaticTHat
Type: real
Default: 1.0
When it matters: Whenever RHSMode == 1 and includePhi1 == .true. and withAdiabatic == .true.
Meaning: Temperature of adiabatic species, in units of 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: Whenever RHSMode == 1.
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: $\texttt{alpha} = e\bar{\Phi}/\bar{T}$ (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.

---

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

$$\texttt{nu\_n} = \bar{\nu}\frac{\bar{R}}{\bar{v}}, \tag{3.4}$$

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

$$\begin{aligned}\bar{\nu} &= \frac{4\sqrt{2\pi}\bar{n}e^4\ln\Lambda}{3(4\pi\epsilon_0)^2\sqrt{\bar{m}}\bar{T}^{3/2}} \quad \text{(SI units)} \\ &= \frac{4\sqrt{2\pi}\bar{n}e^4\ln\Lambda}{3\sqrt{\bar{m}}\bar{T}^{3/2}} \quad \text{(Gaussian units)}\end{aligned} \tag{3.5}$$

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 5.8 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 5.8 for more details.

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

$$\texttt{EParallelHat} = \langle \mathbf{E}\cdot\mathbf{B}\rangle \frac{\bar{R}}{\bar{\Phi}\bar{B}} \tag{3.6}$$

(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 5 variables dPhiHatdpsiHat, dPhiHatdpsiN, dPhiHatdrHat, dPhiHatdrN, or Er 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 5 variables dPhiHatdpsiHat, dPhiHatdpsiN, dPhiHatdrHat, dPhiHatdrN, or Er 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 5 variables dPhiHatdpsiHat, dPhiHatdpsiN, dPhiHatdrHat, dPhiHatdrN, or Er 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 5 variables dPhiHatdpsiHat, dPhiHatdpsiN, dPhiHatdrHat, dPhiHatdrN, or Er will be used, depending on inputRadialCoordinateForGradients.

`Er`
Type: real
Default: 0.0
When it matters: Whenever `RHSMode` == 1 and `inputRadialCoordinateForGradients` == 4
Meaning: The derivative of the normalized electrostatic potential $\hat{\Phi}$ with respect to the radial coordinate $\hat{r}$, multiplied by -1, i.e. `Er`$= -$`dPhiHatdrHat`. Notice that exactly 1 of the 5 variables `dPhiHatdpsiHat`, `dPhiHatdpsiN`, `dPhiHatdrHat`, `dPhiHatdrN`, or `Er` will be used, depending on `inputRadialCoordinateForGradients`.

`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 $\langle p_1 \rangle = 0$. The particle and heat sources have the form $S = (a_2 x^2 + a_0) e^{-x^2}$. The a_2 and a_0 coefficients are determined so that one source term provides particles but not energy, whereas the other source term provides energy but not particles.

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

`constraintScheme` = 3: Same as `constraintScheme` = 1, except the particle and heat sources have the form $S = (a_4 x^4 + a_0)e^{-x^2}$.

`constraintScheme` = 4: Same as `constraintScheme` = 1, except the particle and heat sources have the form $S = (a_4 x^4 + a_2 x^2)e^{-x^2}$.

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]:

$$-(\mathbf{v}_{ma} \cdot \nabla r) \frac{Z_s e}{2T_s x_s} \frac{d\Phi_0}{dr} \frac{\partial f_{a1}}{\partial x_s} \tag{3.7}$$

`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]:

$$\frac{(1-\xi^2)\xi}{2B^3} \frac{d\Phi_0}{dr} (\mathbf{B} \times \nabla r \cdot \nabla B) \frac{\partial f_{s1}}{\partial \xi} \tag{3.8}$$

`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 if `includePhi1inKineticEquation` = `.true.`, and the quasineutrality equation is solved at each point on the flux surface (make sure that your input densities fulfill quasineutrality, see `quasineutralityOption`). 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.
If true, the system becomes nonlinear in the unknowns f_{s1} and Φ_1. Newton's method will be used to solve the nonlinear system, meaning that the usual linear solve in `sfincs` must be iterated several times.

`includePhi1inKineticEquation`
Type: Boolean
Default: `.true.`
When it matters: Whenever `RHSMode` == 1 and `includePhi1` == `.true.`
Meaning: If true, the terms containing Φ_1 will be included in the kinetic equation. If false, and `includePhi1` = `.true.`, then Φ_1 is only included in the quasineutrality equation.

`quasineutralityOption`
Type: integer
Default: 1
When it matters: Whenever `RHSMode` == 1 and `includePhi1` == `.true.`
Meaning: This variable controls which quasineutrality equation is used, and must be set to either 1 or 2. Make sure that your input densities fulfill quasineutrality (also accounting for the adiabatic species if `withAdiabatic`=`.true.`).

`quasineutralityOption` = 1: Full quasineutrality equation. All species (kinetic and adiabatic if applicable) are included in the quasineutrality equation.

`quasineutralityOption` = 2: `EUTERPE` quasineutrality. `sfincs` must be run with an adiabatic species, `withAdiabatic`=`.true.`, and only the adiabatic species and the first kinetic species are included in the quasineutrality equation (their input densities must fulfill quasineutrality). This is the quasineutrality considered in Ref [4].

`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.

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

`magneticDriftScheme` = 1: Use the magnetic drift $v_m = (v_{||}/\Omega_c)\nabla \times (v_{||}\mathbf{b})$.

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

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 6 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 `forceOddNthetaAndNzeta` 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.

`ExBDerivativeSchemeTheta`
Type: integer
Default: 0
When it matters: Whenever the radial electric field is nonzero
Meaning: Options for controlling upwinding of the $\mathbf{E} \times \mathbf{B}$ drift terms. Note the options are different than for `thetaDerivativeScheme` and `zetaDerivativeScheme`, and are analogous to the options for `magneticDriftDerivativeScheme`. This option exists because when the radial electric field is large enough to be comparable to the resonance, such that the coefficient in front of the $\partial f/\partial\theta$ or $\partial f/\partial\zeta$ terms vanishes near the thermal speed of one of the species, the distribution function can develop unphysical grid-scale structure. This option allows upwinding of the $\partial f/\partial\theta$ spatial derivative, which eliminates the grid-scale oscillations. The parameter `ExBDerivativeSchemeZeta` serves the same function for the $\partial f/\partial\zeta$ derivative.

`ExBDerivativeSchemeTheta` = 0: Use same differentiation matrices for the $\mathbf{E} \times \mathbf{B}$ drift terms as for the parallel streaming term.

`ExBDerivativeSchemeTheta` = 1: Treat the $\mathbf{E} \times \mathbf{B}$ drift terms using a 4 point upwinded stencil (1 point on 1 side, 2 points on the other.)

`ExBDerivativeSchemeTheta` = 2: Treat the $\mathbf{E} \times \mathbf{B}$ drift terms using a 5 point upwinded stencil (1 point on 1 side, 3 points on the other.)

`ExBDerivativeSchemeTheta` = 3: Treat the $\mathbf{E} \times \mathbf{B}$ drift terms using a 6 point upwinded stencil (2 points on 1 side, 3 points on the other.)

The different settings require slightly different amounts of memory for factorization due to the different numbers of nonzeros. Option 1 requires the least memory, then 0, then 2, and 3 requires the most. The difference in memory required appears to be small, around 10-20% between settings 1 and 3. On the other hand, the denser the stencil, the more accurate the result. Hence, in order from least to most accurate, the options are 1, 0, 2, 3. Option 1 is substantially less accurate than the other options. You should use the default `ExBDerivativeSchemeTheta` = 0 except for very large radial electric fields, when the electric field is as large as the resonance or larger. In this case, `ExBDerivativeSchemeTheta` = 3 is recommended.

`ExBDerivativeSchemeZeta`
Type: integer
Default: 0
When it matters: Whenever the radial electric field is nonzero
Meaning: This parameter behaves exactly like `ExBDerivativeSchemeTheta`, except it applies to the ζ coordinate instead of θ.

---

magneticDriftDerivativeScheme
Type: integer
Default: 3
When it matters: Whenever `magneticDriftScheme` is nonzero
Meaning: Options for controlling upwinding of the magnetic drift terms. Note the options are different than for `thetaDerivativeScheme` and `zetaDerivativeScheme`, and are analogous to the options for `ExBDerivativeSchemeTheta`. This option exists because when magnetic drift terms involving $\partial f/\partial\theta$ and $\partial f/\partial\zeta$ are implemented using the same centered difference stencil as the parallel streaming term, the distribution function can develop unphysical grid-scale structure. This option allows upwinding of these spatial derivatives, which eliminates the grid-scale oscillations.

`magneticDriftDerivativeScheme` = 0: Use same differentiation matrices for the magnetic drift terms as for the parallel streaming term.

`magneticDriftDerivativeScheme` = 1: Treat the magnetic drift terms using a 4 point upwinded stencil (1 point on 1 side, 2 points on the other.)

`magneticDriftDerivativeScheme` = 2: Treat the magnetic drift terms using a 5 point upwinded stencil (1 point on 1 side, 3 points on the other.)

`magneticDriftDerivativeScheme` = 3: Treat the magnetic drift terms using a 6 point upwinded stencil (2 points on 1 side, 3 points on the other.)

The different settings require slightly different amounts of memory for factorization due to the different numbers of nonzeros. Option 1 requires the least memory, then 0, then 2, and 3 requires the most. The difference in memory required appears to be small, around 10-20% between settings 1 and 3. On the other hand, the denser the stencil, the more accurate the result. Hence, in order from least to most accurate, the options are 1, 0, 2, 3. Option 1 is substantially less accurate than the other options. The default setting of 3 is robust and recommended in most cases.

---

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.

`xGridScheme` = 7: Chebyshev grid on [0, `xMax`], forcing $f = 0$ at `xMax`. Original treatment of Rosenbluth potentials.

`xGridScheme` = 8: Chebyshev grid on [0, `xMax`], with no boundary condition imposed at `xMax`. Original treatment of Rosenbluth potentials.

The recommended value for this parameter is the default, 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.

`PETSCPreallocationStrategy`
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.

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

`PETSCPreallocationStrategy` = 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.

`Nxi_for_x_option`
Type: integer

Default: 1
When it matters: Always
Meaning: This setting controls how the number of Legendre polynomials depends on the speed x for representing the distribution function. You can see the actual number of Legendre polynomials used for each x grid point by examining the `Nxi_for_x` variable in the output file.

`Nxi_for_x_option` = 0: Use the same number (`Nxi`) of Legendre polynomials for all x. This approach is simplest, but it is somewhat inefficient since small values of x represent higher collisionality, so there is weaker pitch-angle dependence of the distribution function.

`Nxi_for_x_option` = 1: The number of Legendre polynomials will be a linear function of x up to $x = 2$, at which point `Nxi` polynomials will be used. For $x > 2$, the number of Legendre polynomials used will remain at `Nxi`.

`Nxi_for_x_option` = 2: The number of Legendre polynomials will be a quadratic function of x up to $x = 2$, at which point `Nxi` polynomials will be used. For $x > 2$, the number of Legendre polynomials used will remain at `Nxi`.

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.

`preconditioner_magnetic_drifts_max_L`
Type: integer
Default: 2
When it matters: Whenever `useIterativeLinearSolver` = `.true.` and `magneticDriftScheme` $>$ 0
Meaning: Maximum Legendre mode number for which the poloidal and toroidal magnetic drift terms are included in the preconditioner. As this parameter is increased, more memory is required

for factorization, but fewer KSP iterations are required. Setting this parameter to $\geq$ Nxi means that the magnetic drifts are always included in the preconditioner. Setting this parameter to < 0 means the magnetic drifts are never included in the preconditioner. The default value should be good.

---

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: Resolution 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. In this type of scan, `sfincsScan` creates a subdirectory for each value of minor radius, and a `scanType` = 2 scan is run in each of these subdirectories.

`scanType` = 21: Read in a list of requested runs from a file `runspec.dat`. See `sfincs/fortran/utils/sfincsScan_21` for an example file. If the file has a different name than `runspec.dat`, for instance `thefilename.dat`, this name can be specified by adding the line `!ss runSpecFile = thefilename.dat`

3.9.1 Parameters related only to scanType = 1 (resolution convergence scans).

The resolution parameters discussed in section 3.5 each have 3 associated `sfincsScan` parameters which are used for convergence scans (`scanType` = 1): `...MinFactor`, `...MaxFactor`, and `...NumRuns`. The first two of these set the range by which the associated resolution parameter is scaled in a convergence scan. The `...NumRuns` parameter sets the number of values tried in a convergence scan. The code attempts to space the values evenly in a logarithmic sense, as in Matlab's 'logspace' function. For example, the following settings

```
Nxi = 20
!ss NxiMinFactor = 0.5
!ss NxiMaxFactor = 2.0
!ss NxiNumRuns = 3
```

would mean the values `Nxi`= 10, 20, and 40 would be tried in a convergence scan. If you don't want to scan a variable in a convergence scan, set the associated `...NumRuns` parameter to 0, or do not include this parameter in the input file. For each resolution parameter (`Ntheta`, `Nzeta`, `Nxi`, etc.), the value itself is read by Fortran and so should not be preceded by `!ss`. However the `...MinFactor`, `...MaxFactor`, and `...NumRuns` quantities are read by `sfincsScan` and so must be preceded by `!ss`

NthetaMaxFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum factor by which `Ntheta` will be multiplied in a convergence scan.

NthetaMinFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Minimum factor by which `Ntheta` will be multiplied in a convergence scan.

NthetaNumRuns
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum number of values of `Ntheta` which will be used in a convergence scan. Only odd integers can be used for `Ntheta`, so the actual number of `Ntheta` values used in the scan may be less than `NthetaNumRuns`.

NzetaMaxFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum factor by which `Nzeta` will be multiplied in a convergence scan.

NzetaMinFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Minimum factor by which `Nzeta` will be multiplied in a convergence scan.

NzetaNumRuns
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum number of values of `Nzeta` which will be used in a convergence scan. Only odd integers can be used for `Nzeta`, so the actual number of `Nzeta` values used in the scan may be less than `NzetaNumRuns`.

NxiMaxFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum factor by which `Nxi` will be multiplied in a convergence scan.

NxiMinFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Minimum factor by which `Nxi` will be multiplied in a convergence scan.

NxiNumRuns
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum number of values of `Nxi` which will be used in a convergence scan. Only integers can be used for `Nxi`, so the actual number of `Nxi` values used in the scan may be less than `NxiNumRuns`.

NxMaxFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum factor by which `Nx` will be multiplied in a convergence scan.

NxMinFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Minimum factor by which `Nx` will be multiplied in a convergence scan.

NxNumRuns
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 1.
Meaning: Maximum number of values of `Nx` which will be used in a convergence scan. Only integers can be used for `Nx`, so the actual number of `Nx` values used in the scan may be less than `NxNumRuns`.

---

solverToleranceMaxFactor
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 1.

Meaning: Maximum factor by which solverTolerance will be multiplied in a convergence scan.

---

solverToleranceMinFactor
Type: real
When it matters: Only when sfincsScan is run with scanType = 1.
Meaning: Minimum factor by which solverTolerance will be multiplied in a convergence scan.

---

solverToleranceNumRuns
Type: integer
When it matters: Only when sfincsScan is run with scanType = 1.
Meaning: Number of values of solverTolerance which will be used in a convergence scan.

NLMaxFactor
Type: real
When it matters: Only when collisionOperator = 0 and sfincsScan is run with scanType = 1.
Meaning: Maximum factor by which NL will be multiplied in a convergence scan.

NLMinFactor
Type: real
When it matters: Only when collisionOperator = 0 and sfincsScan is run with scanType = 1.
Meaning: Minimum factor by which NL will be multiplied in a convergence scan.

NLNumRuns
Type: integer
When it matters: Only when collisionOperator = 0 and sfincsScan is run with scanType = 1.
Meaning: Maximum number of values of NL which will be used in a convergence scan. Only integers can be used for NL, so the actual number of NL values used in the scan may be less than NLNumRuns.

NxPotentialsPerVthMaxFactor
Type: real
When it matters: Only when collisionOperator = 0, xGridScheme < 5, and sfincsScan is run with scanType = 1. Since the recommended value of xGridScheme is 5, this parameter is basically obsolete.
Meaning: Maximum factor by which NxPotentialsPerVth will be multiplied in a convergence scan.

NxPotentialsPerVthMinFactor
Type: real
When it matters: Only when collisionOperator = 0, xGridScheme < 5, and sfincsScan is

run with scanType = 1. Since the recommended value of xGridScheme is 5, this parameter is basically obsolete.
Meaning: Minimum factor by which NxPotentialsPerVth will be multiplied in a convergence scan.

NxPotentialsPerVthNumRuns
Type: integer
When it matters: Only when collisionOperator = 0, xGridScheme < 5, and sfincsScan is run with scanType = 1. Since the recommended value of xGridScheme is 5, this parameter is basically obsolete.
Meaning: Number of values of NxPotentialsPerVth which will be used in a convergence scan.

---

xMaxMaxFactor
Type: real
When it matters: Only when collisionOperator = 0, xGridScheme < 5, and sfincsScan is run with scanType = 1. Since the recommended value of xGridScheme is 5, this parameter is basically obsolete.
Meaning: Maximum factor by which xMax will be multiplied in a convergence scan.

---

xMaxMinFactor
Type: real
When it matters: Only when collisionOperator = 0, xGridScheme < 5, and sfincsScan is run with scanType = 1. Since the recommended value of xGridScheme is 5, this parameter is basically obsolete.
Meaning: Minimum factor by which xMax will be multiplied in a convergence scan.

---

xMaxNumRuns
Type: integer
When it matters: Only when collisionOperator = 0, xGridScheme < 5, and sfincsScan is run with scanType = 1. Since the recommended value of xGridScheme is 5, this parameter is basically obsolete.
Meaning: Number of values of xMax which will be used in a convergence scan.

3.9.2 Parameters related only to scanType = 2 (scans of radial electric field).

In this scan of the radial electric field, the values of electric field used will always be uniformly (linearly) spaced. Notice that exactly 1 of the 5 variables dPhiHatdpsiHatMax, dPhiHatdpsiNMax, dPhiHatdrHatMax, dPhiHatdrNMax, or ErMax will be used, depending on inputRadialCoordinateForGradients. Similarly, exactly 1 of the 5 variables dPhiHatdpsiHatMin, dPhiHatdpsiNMin, dPhiHatdrHatMin, dPhiHatdrNMin, or ErMin will be used.

NErs
Type: integer
When it matters: Only when sfincsScan is run with scanType = 2.
Meaning: Number of values of radial electric field to consider in a scan.

---

dPhiHatdpsiHatMax
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 0 and sfincsScan is run with scanType = 2.
Meaning: Maximum value of dPhiHatdpsiHat to use in the scan.

---

dPhiHatdpsiHatMin
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 0 and sfincsScan is run with scanType = 2.
Meaning: Minimum value of dPhiHatdpsiHat to use in the scan.

---

dPhiHatdpsiNMax
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 1 and sfincsScan is run with scanType = 2.
Meaning: Maximum value of dPhiHatdpsiN to use in the scan.

---

dPhiHatdpsiNMin
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 1 and sfincsScan is run with scanType = 2.
Meaning: Minimum value of dPhiHatdpsiN to use in the scan.

---

dPhiHatdrHatMax
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 2 and sfincsScan is run with scanType = 2.
Meaning: Maximum value of dPhiHatdrHat to use in the scan.

---

dPhiHatdrHatMin
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 2 and sfincsScan is run with scanType = 2.
Meaning: Minimum value of dPhiHatdrHat to use in the scan.

---

dPhiHatdrNMax
Type: real
When it matters: Only when inputRadialCoordinateForGradients = 3 and sfincsScan is run with scanType = 2.
Meaning: Maximum value of dPhiHatdrN to use in the scan.

---

dPhiHatdrNMin
Type: real

When it matters: Only when `inputRadialCoordinateForGradients` = 3 and `sfincsScan` is run with `scanType` = 2.
Meaning: Minimum value of `dPhiHatdrN` to use in the scan.

`ErMax`
Type: real
When it matters: Only when `inputRadialCoordinateForGradients` = 4 and `sfincsScan` is run with `scanType` = 2.
Meaning: Maximum value of `Er` to use in the scan.

`ErMin`
Type: real
When it matters: Only when `inputRadialCoordinateForGradients` = 4 and `sfincsScan` is run with `scanType` = 2.
Meaning: Minimum value of `Er` to use in the scan.

3.9.3 Parameters related only to `scanType` = 3 (scans of an arbitrary input parameter).

`scanVariable`
Type: string. Must be of the fortran namelist parameters that takes an integer or real value. Case-insensitive.
When it matters: Only when `sfincsScan` is run with `scanType` = 3.
Meaning: Name of the variable to scan in a `scanType` = 3 scan.

`scanVariableMax`
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 3.
Meaning: Maximum value of `scanVariable` to use in a `scanType` = 3 scan.

`scanVariableMin`
Type: real
When it matters: Only when `sfincsScan` is run with `scanType` = 3.
Meaning: Minimum value of `scanVariable` to use in a `scanType` = 3 scan.

`scanVariableN`
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 3.
Meaning: Number of values of `scanVariable` to use in a `scanType` = 3 scan.

`scanVariableScale`
Type: string. Must be ‘linear’, ‘lin’, ’logarithmic’, or ’log’
When it matters: Only when `sfincsScan` is run with `scanType` = 3.
Meaning: Whether to space the values of `scanVariable` in a linear or logarithmic manner. The settings ‘linear’ and ‘lin’ have identical behavior. The settings ‘logarithmic’ and ‘log’ have identical behavior.

3.9.4 Parameters related only to `scanType` = 4 or 5 (radial scans).

Notice that exactly 1 of the 4 variables `psiHat_max`, `psiN_max`, `rHat_max`, and `rN_max` will be used, depending on `inputRadialCoordinate`. Similarly, exactly 1 of the 4 variables `psiHat_min`, `psiN_min`, `rHat_min`, and `rN_min` will be used.

`profilesScheme`
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 4 or 5.
Meaning: How to specify the profiles of density, temperature, and (when `scanType` = 5) the range of radial electric field to consider.

`profilesScheme` = 1: Read a '`profiles`' file which contains the input profiles on a grid in one of the 4 available radial coordinates.

`profilesScheme` = 2: Read a '`profiles`' file which contains the input profiles expressed as polynomials in one of the 4 available radial coordinates.

`Nradius`
Type: integer
When it matters: Only when `sfincsScan` is run with `scanType` = 4 or 5.
Meaning: Maximum number of values of minor radius to consider in the scan. Depending on `geometryScheme` and `VMECRadialOption`, it may be that only surfaces available in the magnetic equilibrium file will be used, in which case fewer than `Nradius` radii may be used.

`psiHat_max`
Type: real
When it matters: Only when `inputRadialCoordinate` = 0 and `sfincsScan` is run with `scanType` = 4 or 5.
Meaning: Maximum value of `psiHat` to use in the scan.

`psiHat_min`
Type: real
When it matters: Only when `inputRadialCoordinate` = 0 and `sfincsScan` is run with `scanType` = 4 or 5.
Meaning: Minimum value of `psiHat` to use in the scan.

`psiN_max`
Type: real
When it matters: Only when `inputRadialCoordinate` = 1 and `sfincsScan` is run with `scanType` = 4 or 5.
Meaning: Maximum value of `psiN` to use in the scan.

---

psiN_min
Type: real
When it matters: Only when inputRadialCoordinate = 1 and sfincsScan is run with scanType = 4 or 5.
Meaning: Minimum value of psiN to use in the scan.

---

rHat_max
Type: real
When it matters: Only when inputRadialCoordinate = 2 and sfincsScan is run with scanType = 4 or 5.
Meaning: Maximum value of rHat to use in the scan.

---

rHat_min
Type: real
When it matters: Only when inputRadialCoordinate = 2 and sfincsScan is run with scanType = 4 or 5.
Meaning: Minimum value of rHat to use in the scan.

---

rN_max
Type: real
When it matters: Only when inputRadialCoordinate = 3 and sfincsScan is run with scanType = 4 or 5.
Meaning: Maximum value of rN to use in the scan.

---

rN_min
Type: real
When it matters: Only when inputRadialCoordinate = 3 and sfincsScan is run with scanType = 4 or 5.
Meaning: Minimum value of rN to use in the scan.

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. The memory required by GMRES is typically quite small compared to the memory required for the LU factorization.

-pc_factor_mat_solver_package <packagename>
Meaning: Which sparse direct solver package is used to LU-factorize the preconditioner matrix. This command-line flag overrides the related namelist parameter `whichParallelSolverToFactorPreconditioner`. See section 5.6 for further information about the available packages.

3.11 mumps commands

The `mumps` solver package has many control parameters which are documented in the manual, available here. In `sfincs`, as in any `PETSc` application, these control parameters can be set using the command-line flags `-mat_mumps_cntl_X YYYY` (for floating point parameters) and `-mat_mumps_icntl_X YYYY` (for integer control parameters). Here, `X` is the numeric index of the control parameter, and `YYYY` is the desired setting. Here we list some of the more useful options.

-mat_mumps_icntl_4 <integer>
Meaning: How much diagnostic information will be printed by `mumps`. Default is 3, causing extensive diagnostic information to be printed to standard output about the memory required for factorizing the preconditioner. Set this parameter to 0 to suppress this output from mumps.

-mat_mumps_icntl_14 <integer>
Meaning: Percentage margin allowed for increase of certain arrays during the LU factorization. The default value set by `sfincs` is 50 (higher than the original default value in `mumps`.) If `sfincs` exits with the `mumps` error `INFO(1)=-9`, then further increasing this parameter may help.

-mat_mumps_icntl_22 1
Meaning: Turns on the out-of-core solve capability, which reduces the memory required at the cost of speed. See section 5.7 for further details.

-mat_mumps_icntl_28 2
Meaning: Uses one of the parallelized libraries ParMETIS or PT-SCOTCH for analyzing the matrix, instead of the default serial algorithm.

CHAPTER 4

Kinetic equations

The `sfincs` code is capable of solving many different variants of the drift-kinetic equation. In this section we summarize the most common drift-kinetic equations which can be solved with the code, giving the associated input parameters (which are all in the `physicsParameters` namelist.) For other choices of these input parameters, other combinations of terms in the drift-kinetic equation can be used than those given here. For more information, see section 5.3, section 5.4, section 5.5, and the version 3 technical documentation. The following terms are always included in `sfincs`: parallel streaming, the magnetic mirror force, the collision operator, and the inhomogeneous drive term from radial gradients. Other terms generally can be turned on and off as desired using parameters in the `physicsParameters` namelist.

Throughout this section, gradients and other partial derivatives are taken at constant $x_a = v/\sqrt{2T_a/m_a}$ and at constant $\xi = v_{||}/v$, unless denoted otherwise with subscripts. We use s to denote the species, r to denote any radial coordinate, (expressions are independent of the specific radial coordinate used,) C_a to denote the collision operator, and S_a to denote the source-sink term. A more detailed discussion of the equations implemented in the code can be found in the technical documentation for `sfincs` version 3, available in the `/doc` directory of the repository.

4.1 Default: Full $\mathbf{E} \times \mathbf{B}$ trajectories; no poloidal or toroidal magnetic drifts; flux function potential

$$
\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r\right)\cdot\nabla f_{s1} \\
&+\left[-\frac{(1-\xi^2)v}{2B}\nabla_{||}B + \frac{(1-\xi^2)\xi}{2B^3}\frac{d\Phi_0}{dr}\mathbf{B}\times\nabla r\cdot\nabla B\right]\frac{\partial f_{s1}}{\partial \xi} \\
&-(\mathbf{v}_{ma}\cdot\nabla r)\frac{Z_s e}{2T_s x_s}\frac{d\Phi_0}{dr}\frac{\partial f_{a1}}{\partial x_s} \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} = C_a + S_a
\end{aligned} \tag{4.1}
$$

Note that this equation is equivalent to the following one, in which the independent variables are $(\mu, v_{||})$ instead of (ξ, x_s):

$$\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r\right)\cdot(\nabla f_{s1})_{\mu,v_{||}} \\
&+\left[-\mu\nabla_{||}B - \frac{v_{||}}{B^2}\frac{d\Phi_0}{dr}\mathbf{b}\times\nabla B\times\nabla r\right]\left(\frac{\partial f_{a1}}{\partial v_{||}}\right)_{\mu} \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} = C_a + S_a
\end{aligned} \tag{4.2}$$

These equivalent forms of the kinetic equation are selected using
`includeXDotTerm` = `.true.` (Default)
`includeElectricFieldTermInXiDot` = `.true.` (Default)
`useDKESExBDrift` = `.false.` (Default)
`magneticDriftScheme` = 0 (Default)
`includePhi1` = `.false.` (Default)
`includeRadialExBDrive` = `.false.` (Default)
`nonlinear` = `.false.` (Default)

4.2 DKES $\mathbf{E}\times\mathbf{B}$ trajectories; no poloidal or toroidal magnetic drifts; flux function potential

This form of the kinetic equation is useful for benchmarking with DKES and other codes that use the same equation.

$$\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{\langle B^2\rangle}\mathbf{B}\times\nabla r\right)\cdot\nabla f_{s1} \\
&-\frac{(1-\xi^2)v}{2B}(\nabla_{||}B)\frac{\partial f_{s1}}{\partial \xi} \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} = C_a + S_a
\end{aligned} \tag{4.3}$$

This form of the kinetic equation is selected using
`includeXDotTerm` = `.false.` (Not default)
`includeElectricFieldTermInXiDot` = `.false.` (Not default)
`useDKESExBDrift` = `.true.` (Not default)
`magneticDriftScheme` = 0 (Default)
`includePhi1` = `.false.` (Default)
`includeRadialExBDrive` = `.false.` (Default)
`nonlinear` = `.false.` (Default)

4.3 Full $\mathrm{E} \times \mathrm{B}$ trajectories; including poloidal and toroidal magnetic drifts; flux function potential

$$
\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r + \mathbf{v}_{ms}\right)\cdot\nabla\theta\frac{\partial f_{s1}}{\partial\theta} \\
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r + \mathbf{v}_{ms}\right)\cdot\nabla\zeta\frac{\partial f_{s1}}{\partial\zeta} \\
&+\left\{-\frac{(1-\xi^2)v}{2B}\nabla_{||}B + \frac{(1-\xi^2)\xi}{2B^3}\frac{d\Phi_0}{dr}\mathbf{B}\times\nabla r\cdot\nabla B\right. \\
&\quad\left.-\frac{T_s x_s^2(\nabla r\cdot\nabla\theta\times\nabla\zeta)}{Z_s e B^3}(1-\xi^2)\xi\left[\left(\frac{\partial B_r}{\partial\zeta} - \frac{\partial B_\zeta}{\partial r}\right)\frac{\partial B}{\partial\theta} + \left(\frac{\partial B_\theta}{\partial r} - \frac{\partial B_r}{\partial\theta}\right)\frac{\partial B}{\partial\zeta}\right]\right\}\frac{\partial f_{s1}}{\partial\xi} \\
&-(\mathbf{v}_{ma}\cdot\nabla r)\frac{Z_s e}{2T_s x_s}\frac{d\Phi_0}{dr}\frac{\partial f_{a1}}{\partial x_s} \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right]f_{aM} = C_a + S_a
\end{aligned}
\tag{4.4}
$$

Notice the magnetic drifts affect the coefficients of $\partial f_{s1}/\partial\theta$, $\partial f_{s1}/\partial\zeta$, and $\partial f_{s1}/\partial\xi$, but there is no change to the coefficient of $\partial f_{s1}/\partial x_s$. This form of the kinetic equation is selected using
`includeXDotTerm` = `.true.` (Default)
`includeElectricFieldTermInXiDot` = `.true.` (Default)
`useDKESExBDrift` = `.false.` (Default)
`magneticDriftScheme` = 1 (Not default)
`includePhi1` = `.false.` (Default)
`includeRadialExBDrive` = `.false.` (Default)
`nonlinear` = `.false.` (Default)

4.4 Full $\mathrm{E} \times \mathrm{B}$ trajectories; no poloidal or toroidal magnetic drifts; leading Φ_1 term

In this form of the drift-kinetic equation, we include the largest term involving Φ_1, associated with the adiabatic response. However, other terms considered in Ref [4] are not included.

$$
\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r\right)\cdot\nabla f_{s1} \\
&+\left[-\frac{(1-\xi^2)v}{2B}\nabla_{||}B + \frac{(1-\xi^2)\xi}{2B^3}\frac{d\Phi_0}{dr}\mathbf{B}\times\nabla r\cdot\nabla B\right]\frac{\partial f_{s1}}{\partial\xi} \\
&-(\mathbf{v}_{ma}\cdot\nabla r)\frac{Z_s e}{2T_s x_s}\frac{d\Phi_0}{dr}\frac{\partial f_{a1}}{\partial x_s} \\
&+\frac{Z_s e}{T_s}f_{sM}v_{||}\nabla_{||}\Phi_1 \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right]f_{aM} = C_a + S_a
\end{aligned}
\tag{4.5}
$$

Note that this equation is equivalent to the following one, in which the independent variables are $(\mu, v_{||})$ instead of (ξ, x_s):

$$\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r\right)\cdot(\nabla f_{s1})_{\mu,v_{||}} \\
&+\left[-\mu\nabla_{||}B - \frac{v_{||}}{B^2}\frac{d\Phi_0}{dr}\mathbf{b}\times\nabla B\times\nabla r\right]\left(\frac{\partial f_{a1}}{\partial v_{||}}\right)_{\mu} \\
&+\frac{Z_s e}{T_s} f_{sM} v_{||}\nabla_{||}\Phi_1 \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} = C_a + S_a
\end{aligned} \tag{4.6}$$

These equivalent forms of the kinetic equation are selected using
`includeXDotTerm` = `.true.` (Default)
`includeElectricFieldTermInXiDot` = `.true.` (Default)
`useDKESExBDrift` = `.false.` (Default)
`magneticDriftScheme` = 0 (Default)
`includePhi1` = `.true.` (Not default)
`includeRadialExBDrive` = `.false.` (Default)
`nonlinear` = `.false.` (Default)

4.5 Full $\mathbf{E}\times\mathbf{B}$ trajectories; no poloidal or toroidal magnetic drifts; García-Regaña Φ_1 terms

This form of the drift-kinetic equation is nearly identical to equation (11) in Ref [4]. The one difference (which should be small) is that at the end of (11), García-Regaña has a term $\propto \mathbf{v}_m\cdot\nabla\Phi_1$, which is not (yet) in `sfincs`.

$$\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r\right)\cdot\nabla f_{s1} \\
&+\left[-\frac{(1-\xi^2)v}{2B}\nabla_{||}B + \frac{(1-\xi^2)\xi}{2B^3}\frac{d\Phi_0}{dr}\mathbf{B}\times\nabla r\cdot\nabla B - \frac{Z_s e}{v m_s}(1-\xi^2)(\nabla_{||}\Phi_1)\right]\frac{\partial f_{s1}}{\partial\xi} \\
&+\left[-(\mathbf{v}_{ma}\cdot\nabla r)\frac{Z_s e}{2T_s x_s}\frac{d\Phi_0}{dr} - \frac{Z_s e\xi}{v_s m_s}\right]\frac{\partial f_{a1}}{\partial x_s} \\
&+\frac{Z_s e}{T_s} f_{sM} v_{||}\nabla_{||}\Phi_1 \\
&+(\mathbf{v}_E\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} = C_a + S_a
\end{aligned} \tag{4.7}$$

Note that this equation is equivalent to the following one, in which the independent variables are $(\mu, v_{||})$ instead of (ξ, x_s):

$$
\begin{aligned}
&\left(v_{||}\mathbf{b} + \frac{d\Phi_0}{dr}\frac{1}{B^2}\mathbf{B}\times\nabla r\right)\cdot(\nabla f_{s1})_{\mu,v_{||}} \\
&+\left[-\frac{Z_s e}{m_s}\nabla_{||}\Phi_1 - \mu\nabla_{||}B - \frac{v_{||}}{B^2}\frac{d\Phi_0}{dr}\mathbf{b}\times\nabla B\times\nabla r\right]\left(\frac{\partial f_{a1}}{\partial v_{||}}\right)_{\mu} \\
&+\frac{Z_s e}{T_s} f_{sM} v_{||}\nabla_{||}\Phi_1 \\
&+(\mathbf{v}_E\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} \\
&+(\mathbf{v}_{ma}\cdot\nabla r)\left[\frac{1}{n_a}\frac{dn_a}{dr} + \frac{Z_s e}{T_s}\frac{d\Phi_0}{dr} + \left(x_s^2 - \frac{3}{2}\right)\frac{1}{T_s}\frac{dT_s}{dr}\right] f_{aM} = C_a + S_a
\end{aligned}
\tag{4.8}
$$

These equivalent forms of the kinetic equation are selected using
`includeXDotTerm` = `.true.` (Default)
`includeElectricFieldTermInXiDot` = `.true.` (Default)
`useDKESExBDrift` = `.false.` (Default)
`magneticDriftScheme` = 0 (Default)
`includePhi1` = `.true.` (Not default)
`includeRadialExBDrive` = `.true.` (Not default)
`nonlinear` = `.true.` (Not default)

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^{20} m^{-3}, 10^{19} 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 `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`.

4: Same as option 2, except the radial electric field is specified using `Er`. Thus, derivatives with respect to `rHat` will be used: Density gradients are specified using `dnHatdrHats`, temperature gradients are specified using `dTHatdrHats`, a single E_r is specified using `Er`, and a range of E_r for a scan is specified using `ErMin`-`ErMax`.

The most common choice is the default, 4. The quantity `Er` in both the input and output files is defined as $-d\hat{\Phi}/d\hat{r} = -(\bar{R}/\bar{\Phi})d\Phi/dr$.

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 given in terms of the electric field variable selected by `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.
```

There is not a significant difference in computational cost between these models.

5.4 Quasineutrality and variation of the electrostatic potential on the flux surface

One choice you should consider in setting up a computation is whether or not to include variation on the flux surface of the electrostatic potential, $\Phi_1(\theta, \zeta)$. Such variation does occur to some degree in a real plasma, but it is neglected in analytical theory and in many codes such as `dkes`. It can be proved that including Φ_1 has no effect on the particle or heat fluxes, parallel flows, or bootstrap current when $E_r = 0$, but generally there can be some difference when $E_r \neq 0$. (There is a subtlety in showing that the heat flux is the same with and without Φ_1, discussed in the notes 20150325-01 in `sfincs/doc/`.)

In `sfincs`, you can choose whether or not to include Φ_1 using the paramter `includePhi1` in the `physicsParameters` namelist. If and only if Φ_1 is included, a quasineutrality equation is solved at each point on the flux surface. Due to these extra unknowns (Φ_1) and extra equations (quasineutrality), the system matrix is slightly larger when `includePhi1` is `.true.`. Specifically, the number of rows and columns are each increased by `Ntheta`$\times$`Nzeta`$+1$. This increase is miniscule compared to the number of rows and columns associated with the kinetic equation, which depends not only on real space but also on velocity space and species. Thus, there is very little extra computational cost associated with `includePhi1`. You may wish to set `includePhi1` = `.true.` when using `sfincs` to model an experiment, and set `includePhi1` = `.false.` when comparing `sfincs` with analytic theory or with another code that does not include Φ_1.

5.5 Poloidal and toroidal magnetic drifts

You can choose to either include or not include the poloidal and toroidal magnetic drifts. These drifts are turned off by default. To turn them on, all you need to do is set `magneticDriftScheme` = 1 in the `physicsParameters` namelist. (Setting `magneticDriftScheme` = 2 uses a slightly different parallel magnetic drift, which gives indistinguishable results to setting 1 for all cases examined so far, and which is in fact exactly identical to setting 1 in the limit of vanishing plasma beta.) If the poloidal/toroidal magnetic drifts are turned on, you must use VMEC geometry (`geometryScheme` = 5), since the magnetic drifts depend on various derivatives of the components of the magnetic field which are not available in the simplified geometry models.

The magnetic drift terms introduce nonzeros in the system matrix, and therefore increase the memory and time required for factorization. The change is small; a typical increase in both memory and time is 20-40%. These magnetic drift terms typically have a minor effect on the physics outputs except when the radial electric field is near 0.

If the electrostatic potential is not constant on flux surfaces and poloidal/toroidal magnetic drifts are included, certain terms exist in the kinetic equation which have not yet been implemented in the code. Thus, at present it is not strictly correct to simultaneously set magneticDriftScheme > 0 and includePhi1 = .true..

5.6 Sparse direct solver packages

As discussed briefly in section 1.4, the most computationally demanding step in sfincs is the direct LU-factorization of a very large sparse nonsymmetric real matrix. The PETSc library which sfincs uses has interfaces to a large number of other packages for direct factorization of such matrices, making it possible to choose among the various solver packages with just a command-line flag (-pc_factor_mat_solver_package). Some lists of the direct solvers available in PETSc can be found here or in the "direct solvers"-"LU" section of this page. It is important for the LU-solver package to be one that is efficiently parallelized, in order to be able to solve problems at the high resolutions required for experimentally relevant collisionality and magnetic geometry. The recommended choice of LU solver is mumps (which is the default), and another good option is superlu_dist. (The PARDISO library available in the Intel Math Kernel Library is probably suitable as well, though we have not investigated it yet.) In side-to-side comparisons, we find mumps systematically uses substantially less memory and time than superlu_dist for factorization. In principle, other solver packages for asymmetric matrices that are interfaced to PETSc could be used as well, such as UMFPACK, PASTIX, etc.

There are two ways to choose between solver packages. One method is the sfincs parameter whichParallelSolverToFactorPreconditioner in the otherNumericalParameters namelist. This parameter only allows you to choose between mumps and superlu_dist, not other solver packages interfaced to PETSc. Another way to choose between solver packages is the command-line flag -pc_factor_mat_solver_package, followed by one of the options in quotation marks here. The command-line flag overrides the namelist parameter.

The physics outputs of the code should be independent of the solver package used to several significant digits. In principle, different solver packages solving the same linear system should find the identical solution. However there will be small differences in the solutions associated with roundoff error.

Note that superlu and superlu_dist are distinct libraries. The former is serial while the latter is parallelized. Therefore there is no reason to use superlu; superlu_dist is always preferable.

The mumps package has a large number of control parameters, which are documented in the mumps manual which can be downloaded here. You do not need to be aware of most of these control parameters. However, several parameters which may be useful are discussed in section 3.11. It is also worth being aware of the section of the mumps manual on error messages. This section is useful for interpreting the INFO(1) and INFO(2) error codes that are reported if sfincs exits with an error associated with mumps.

The superlu_dist package has many fewer options than mumps. You can find of list of the options by running sfincs with the -help command-line flag when using superlu_dist, and searching the output for lines containing superlu_dist. The superlu_dist options are also documented in the package's manual, available here. We have not found any advantage in adjusting any of the superlu_dist options.

The PETSc library includes a built-in sparse direct solver which works on only a single processor. You can select this solver using the command-line flag

`-pc_factor_mat_solver_package petsc`

This solver could potentially be useful if you are running on a system that does not have `mumps` or `superlu_dist` installed, and you are only considering problems that require sufficiently little memory (e.g. tokamaks) that parallelization is not required. However, this solver is less robust than `mumps` or `superlu_dist`, sometimes exiting with an error message that there is a zero pivot even though `mumps` and `superlu_dist` can solve the same system with no problem. Therefore, even if you plan to use only a single processor, we still recommend that you install `mumps` or `superlu_dist`.

5.7 Parallelization: Choosing the number of nodes & processors

Usually the limiting factor for `sfincs` is not time but memory. (The time required depends on the resolution used, but jobs for experimentally relevant W7-X parameters typically take under 10 minutes.) Therefore, when considering how many nodes to request for a `sfincs` job, the first issue to consider is ensuring you have requested sufficient total memory. A good way to determine the memory required (assuming you are using the default solver `mumps`) is to first run `sfincs` on 1 node using the parameters of interest and look for the following line in standard output:

```
** TOTAL space in MBYTES for IC factorization :  1072
```

The number at the end will generally be different; it depends not only on the resolution and number of species used, but also increases with the number of processors as discussed below. Make sure the number of nodes requested times the number of megabytes per node exceeds this number. This line in standard output is generated by `mumps`, so if you are using a different solver, this information is not printed, and you may need to determine the number of nodes by trial-and-error. Since some memory on each node is used by the operating system and by `sfincs` functions other than the solver, you may need to use a slightly higher number of nodes than this estimate suggests. For experimentally relevant W7-X and HSX parameters, we typically use 2-6 nodes with 64 GB each. Problems with lower resolution requirements, such as tokamaks, often can be run on a single node.

The 'IC' in the above line stands for 'in core', meaning the L and U factors are stored in memory rather than on disk. In `mumps`, one can also choose to do an 'out of core' (OOC) solve, in which case substantially less memory is typically required. The price you pay for this memory savings is time, since disk access is slow compared to memory access. Due to this slowdown, we have not used the OOC capability much, but you might find it useful in some circumstances. To see how much memory would be required for an OOC solve, look for the following line in standard output:

```
** TOTAL space in MBYTES for OOC factorization :  128
```

(The number at the end will generally be different.) To invoke out-of-core mode, you must take two steps. First, use the following command-line flag when calling `sfincs`:

`-mat_mumps_icntl_22 1`

Second, you must set the environment variable `MUMPS_OOC_TMPDIR` to some reasonable directory before calling `sfincs`. This environment variable could be set for example in the batch job file. Temporary files containing the L and U factors will be stored in the directory indicated.

Another issue to consider is how many processors to use. It is not always best to use the maximum number of processes available on the number of nodes you have chosen. The reason is that as the preconditioner matrix is divided among more and more processes, the LU factorization becomes less efficient, requiring more memory and more communication. If you examine the `mumps` IC and

OOC memory requirements indicated above, you will find they increase somewhat as the number of processes increase. One needs to find a balance between speed (favoring many processes) and memory requirements (favoring few processes). While it is almost always better overall to use 2 processes compared to 1 process, it is not always better to use 128 processes compared to 64 processes. The sweet spot is often in the range of 16-64 processes. To determine how to request fewer processes than the maximum available on a given number of nodes, see the documentation for your computing system.

If more memory is required than is available, the system will usually terminate your job with an out-of-memory (OOM) error. When this occurs, you need to either increase the number of nodes requested or decrease the number of processors requested.

Most of the time, `sfincs` is run via `sfincsScan` as some parameter is scanned, such as the radial electric field. In this case, the scan is "embarassingly parallel" in the sense that each job in the scan is completely independent of the other jobs. Even if each individual job requires only 1 or a small number of nodes, it is still useful to run `sfincs` on a computing system with many nodes so the scan can be carried out in parallel.

5.8 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. The collision frequency ν is here the value of ν_{ii} one would have if v were the thermal speed. That is, in SI units,

$$\nu = \frac{4}{3\sqrt{\pi}} \frac{n Z^4 e^4 \ln \Lambda}{4\pi \epsilon_0^2 m^2 v^3}. \tag{5.2}$$

The normalized radial electric field is

$$\texttt{EStar} = \frac{cG}{\iota v B_0} \frac{d\Phi}{d\psi} \tag{5.3}$$

(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?

The two parameters `nuPrime` and `EStar` are related to the corresponding DKES parameters `CMUL` and `EFIELD` by

$$\begin{aligned} \texttt{CMUL} &\equiv \frac{\nu_{\text{D}}}{v} = \frac{3\sqrt{\pi}}{4} \left(\text{erf}(1) - \text{Ch}(1)\right) \frac{B_0}{G + \iota I} \texttt{nuPrime}, & (5.4) \\ \texttt{EFIELD} &\equiv -\left[\frac{d\Phi}{dr}\right]_{\text{DKES}} \frac{1}{v B_0} = -\frac{\iota}{G} \left[\frac{d\Psi}{dr}\right]_{\text{DKES}} \texttt{EStar}, & (5.5) \end{aligned}$$

where Ch is the Chandrasekhar function and ν_{D} is the actual pitch-angle deflection frequency of the particle. To do: These expressions are in SI units. Should there be some factors of c to adhere to the Gaussian standard used here? Probably not.

5.9 Poloidal and toroidal angles

If you are interested in any of the output quantities that vary on a flux surface, such as the density or electrostatic potential, then it is important to know how the poloidal and toroidal angles (θ and ζ) in `sfincs` are defined. The definitions of the poloidal and toroidal angles in `sfincs` depend on the input parameter `geometryScheme`. When a `vmec` equilibrium is imported by setting `geometryScheme` = 5, then `sfincs` will use the same poloidal and toroidal angles as `vmec`. The toroidal angle in this case is the normal cylindrical coordinate. Note that field lines are not straight in these `vmec` coordinates. For any other setting of `geometryScheme`, `sfincs` will use Boozer coordinates.

CHAPTER 6

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. 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.

6.1 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. (These values are set as the defaults.) The latter two of these parameters are in fact ignored for the recommended and default `xGridScheme` setting, 5.

6.2 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 6.3)

When the mean-free-path is shorter than the parallel length scale of the equilibrium, 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, 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 nearly as much on collisionality. The reason is that at low collisionality, a boundary layer develops in the distribution function along the boundary between trapped and passing (untrapped) particles. The location of this boundary depends on θ, ζ, and ξ, and so high resolution is required in these coordinates to resolve the boundary layer. But the location of the trapped-passing boundary is independent of x, and hence the resolution requried in x is not particularly high.

Typically you can expect to use Nx= 5-8, with 5 being sufficient at low collisionality and 8 being required at high collisionality. The Nx required for convergence may need to increase slightly with the number of species.

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.

6.3 Convergence testing

To check how well the results of `sfincs` are converged with respect to the resolution parameters, run `sfincsScan` with scanType = 1. In this type of scan, a "base case" is first run at the values of Ntheta, Nzeta, Nxi, and Nx specified in the input file. Then, each of these parameters will be varied in turn, holding the other parameters fixed. The range of Ntheta in the scan is specified by the parameters NthetaMinFactor, NthetaMaxFactor, and NthetaNumRuns. Each of these three parameters is read by `sfincsScan` rather than by `sfincs` itself, and so it must be prefaced by `!ss` in the input namelist file. The ranges of Nzeta, Nxi, and Nx are set by parameters with analogous names, as detailed in section 3.9.1.

For example, suppose you initially run `sfincsScan` with Ntheta= 15, NthetaMinFactor = 0.7, NthetaMaxFactor = 1.5, and NthetaNumRuns = 5. This will generate `sfincs` runs at Ntheta= 11, 13, 15 (the base case), 19, and 23. Notice that `sfincsScan` automatically ensures that only odd values are used. (Only odd values of Ntheta and Nzeta are used internally in `sfincs`.) The maximum and minimum values for the scan, 11 and 23, are the nearest odd integers to Ntheta×NthetaMinFactor and Ntheta×NthetaMaxFactor respectively. Notice also that there are 'gaps' at Ntheta= 17 and 21 since `sfincsScan` attempts to space the values logarithmically rather than uniformly.

In addition to the 4 resolution parameters above, `sfincsScan` also allows the parameters solverTolerance, xMax, and NxPotentialsPerVth to be scanned. The last two of these parameters are not used for the default xGridScheme so it is usually unnecessary to scan them. It is also usually unnecessary to scan solverTolerance since the default value is usually robust.

A directory will be created for each run. Each directory will contain a copy of `input.namelist` in which Ntheta, Nzeta, Nx, or Nxi has been altered appropriately by `sfincsScan`. Each directory will also contain a job file for the run.

You do not need to scan all variables. For example, if you do not wish to scan Nx, you can either set NxNumRuns = 0, or you can not specify NxNumRuns in the `input.namelist` file.

Once the individual runs in the scan have begun to finish, you can plot the results by running

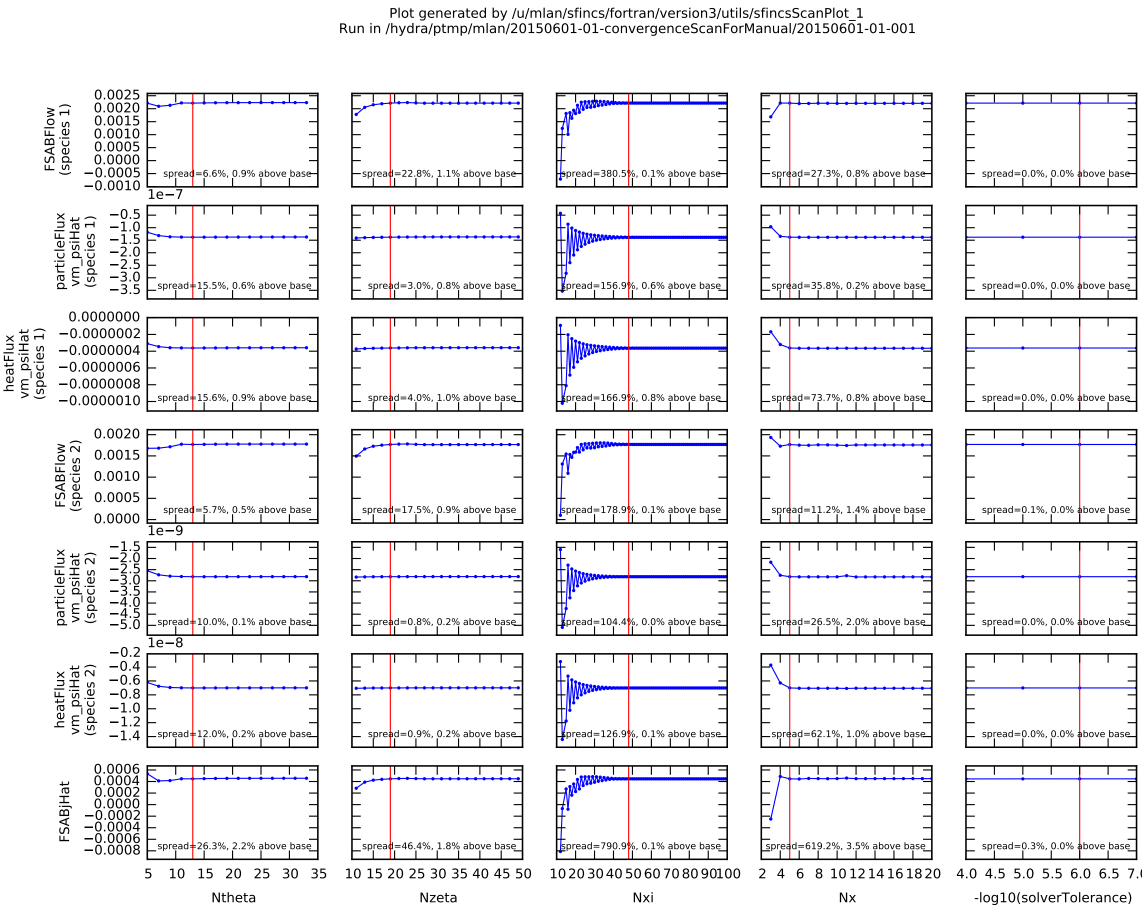

Figure 6.1: Plot generated by `sfincsScanPlot` showing a resolution convergence scan for the example `geometryScheme4_2species_noEr`.

sfincsScanPlot. You can plot the results before all the runs in the scan finish, although at least the base case run must be completed. Different quantities will be displayed depending on RHSMode and includePhi1.

Typical convergence behavior is illustrated in figure 6.1, showing the results of sfincsScan and sfincsScanPlot for the example geometryScheme4_2species_noEr. In this figure, a very large number of runs are included in the scan, more than you would likely include in routine use of the code. The red vertical lines emphasize the "base case" resolution parameters. In each figure the spread is printed, defined as the maximum value $-$ minimum value, divided by the value half-way between maximum and minimum. If some runs have resolution below the base resolution, the spread is also computed and printed excluding runs below the base case resolution. (This is the second percentage printed in each plot, and is usually more important than the first spread percentage.) For all parameters scanned, the physical output quantities (particle flux, heat flux, etc) do not visibly change on the scale of the plots when any of the resolution parameters are increased beyond the base case value. Indeed, the spread for each output quantity is $\leq 3.5\%$ when any resolution parameter is increased. Thus, the results in the base case are well converged, and so we can believe the results. Observe in the figure that the output quantities tend to oscillate when Nxi is incremented by 1 and insufficient Nxi is used. Results for even and odd Nxi converge to the same value, as one would hope.

For routine use of the code, it is not necessary to include as many runs in the convergence scan as shown in figure 6.1, and so a more typical routine scan would look like figure 6.2. This scan is for a pure hydrogen plasma in the W7-X standard configuration, using rN = 0.19, n_e = 10^{20} m^{-3}, T_i = 3.6 keV, and T_e = 5.5 keV. Notice the vertical scales in figure 6.2 have suppressed zeros. For all of the physical output quantities, the spread beyond the base case is $\leq 3.4\%$, i.e. results change by no more than this percentage when each resolution parameter is increased by 50%. Hence we can conclude that the results are suitably converged at the base case resolution parameters. The sfincs resolution parameters and sfincsScan parameters used for this scan were as follows:

```
Ntheta= 19
!ss NthetaMinFactor = 0.7
!ss NthetaMaxFactor = 1.5
!ss NthetaNumRuns = 6
Nzeta= 125
!ss NzetaMinFactor = 0.7
!ss NzetaMaxFactor = 1.5
!ss NzetaNumRuns = 8
Nxi= 140
!ss NxiMinFactor = 0.7
!ss NxiMaxFactor = 1.5
!ss NxiNumRuns = 8
Nx= 5
!ss NxMinFactor = 1
!ss NxMaxFactor = 1.6
!ss NxNumRuns = 4
```

These sfincsScan parameters are good for routine convergence tests, although the sfincs parameters proper (Ntheta, Nzeta, Nxi, and Nx) should be tailored to your application.

If you wish to add more runs to the scan at a later time, you can alter the relevant sfincsScan pa-

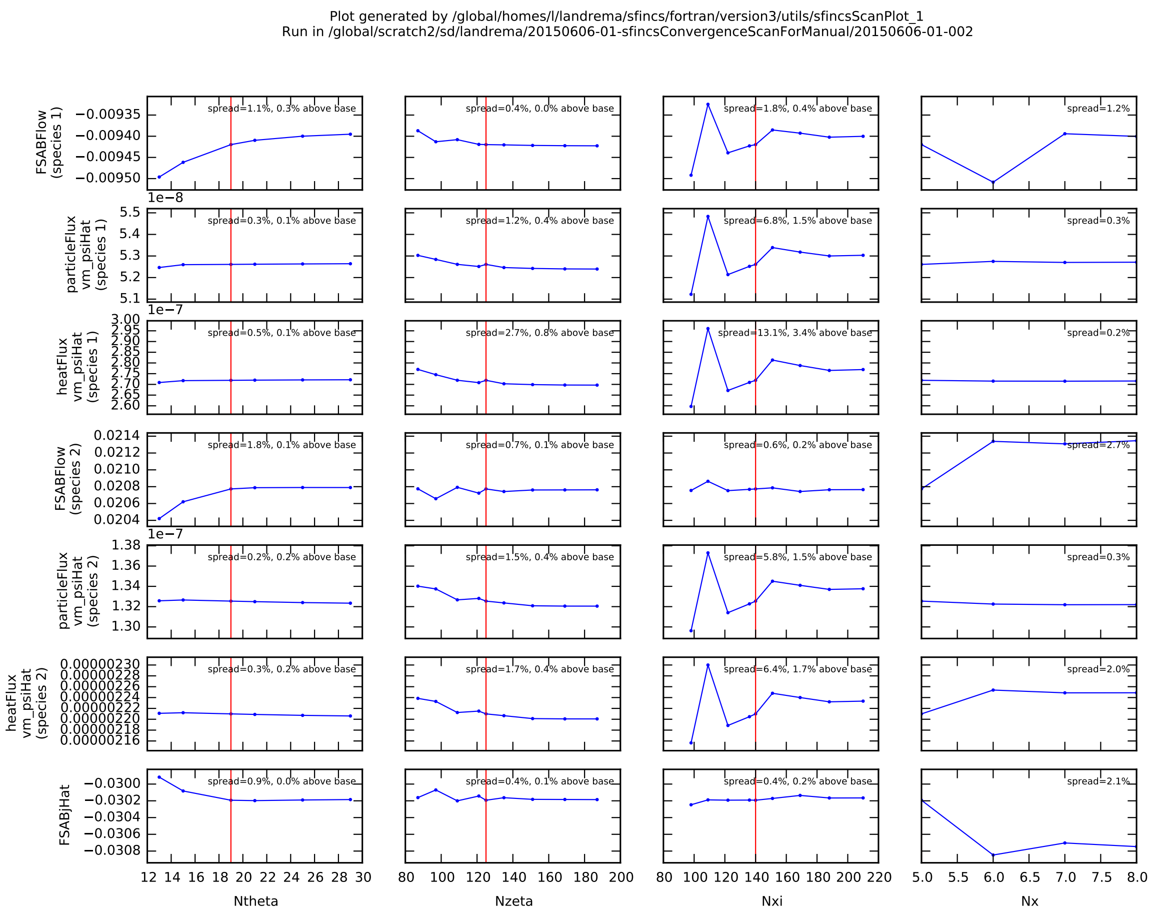

Figure 6.2: Plot generated by `sfincsScanPlot` showing a typical resolution convergence scan for W7-X, as you might generate during routine use of `sfincs`.

rameters in the original `input.namelist` file and run `sfincsScan` again from the original directory. The `sfincsScan` code will automatically detect which runs have already been submitted, so duplicate runs will not be generated. For example, consider the scan of `Ntheta` described at the start of this section, and suppose you wish to fill in the gaps at `Ntheta`= 17 and 21 in the original scan. To do this, you could set `NthetaNumRuns` to a large number like 100 and re-run `sfincsScan`. Since `sfincsScan` intelligently avoids duplication, the only new runs generated will be `Ntheta`= 17 and 21.

If you are ultimately intending to scan E_r, you probably only need to run a convergence scan at a single value of E_r. This is because the resolution requirements of `sfincs` are not sensitive to E_r, as discussed above. Also, if you are ultimately intending to run `sfincs` at a range of minor radii, it is reasonably to only run a convergence scan at a single radius close to the magnetic axis. This is because the typically peaked shape of the temperature profile means that collisionality is lowest near the axis. Since the resolution requirements of `sfincs` are most demanding at low collisionality, then if the code is converged at the radius of lowest collisionality, it should be converged at all radii.

6.4 Examples of resolution requirements

To estimate the appropriate resolution parameters for various circumstances, you can look at the examples in the `sfincs/fortran/version3/examples/` directory. Some other examples of appropriate resolution parameters are given in the following sections.

6.4.1 W7-X with anticipated experimental density and temperature

The following parameters have been extensively tested with W7-X geometry for densities near 10^{20} m^{-3} and temperatures up to 6 keV, and found to give convergence to $\sim 3\%$ (as shown in figure 6.2):

```
Ntheta= 19
Nzeta= 125
Nxi= 140
Nx= 5
```

Note that at lower temperature and/or higher density, the collisionality will be higher, so comparable convergence could be achieved at lower `Nzeta` and `Nxi`. Conversely, at higher temperatures and/or lower desntiy, the collisionality will be lower, so comparable convergence would likely require higher `Nzeta` and `Nxi`.

6.4.2 Figure 3 of the original SFINCS paper

For figure 3 in Ref [1], corresponding to a pure plasma in W7-X with $n = 6.6 \times 10^{19}$ m^3 and $T_i = T_e = 1$ keV, the calculations used

```
Ntheta= 19
Nzeta= 59
Nxi= 60
Nx= 5.
```

Note the lower values of `Nzeta` and `Nxi` used compared to section 6.4.1, which were sufficient since the temperatures were lower.

6.4.3 Figure 4 of the original SFINCS paper

For figure 4 in Ref [1], corresponding to a collisionality scan in LHD, the following resolution parameters were used:

nuPrime	Ntheta	Nzeta	Nxi	Nx
0.001	85	41	103	5
0.01	21	25	70	5
0.1	15	13	37	5
0.3	15	13	34	6
1	13	13	13	8
10	15	13	13	8
100	15	13	13	8

6.4.4 Figure 5 of the original SFINCS paper

For figure 5 in Ref [1], corresponding to a collisionality scan in W7-X, the following resolution parameters were used:

nuPrime	Ntheta	Nzeta	Nxi	Nx
0.001	29	83	180	5
0.01	11	64	100	5
0.1	11	37	37	5
0.3	11	29	30	5
1	13	31	24	6
10	13	35	12	7
100	11	37	13	8

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).

[3] I. G. Abel, G. G. Plunk, E. Wang, M. Barnes, S. C. Cowley, W. Dorland, and A. A. Schekochihin. *Rep. Prog. Phys.*, **76**, 116201 (2013).

[4] J. M. Garcia-Regana, R. Kleiber, C. D. Beidler, Y. Turkin, H. Maassberg, and P. Helander. *Plasma Phys. Controlled Fusion*, **55**, 074008 (2013).