SFINCS User Manual

Version 3

Revised May 21, 2015

Contents

CHAPTER 1

Overview

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

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

This document discusses 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.

1.1 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.

1.2 `sfincs` vs. `sfincsScan`

The core fortran part of `sfincs` solves the kinetic equation for each species at a single flux surface, a single value of E_r, and a single set of other parameters. However, often the goal is to determine the ambipolar E_r at one or more surfaces, or to scan some other parameter. For this task, the `sfincsScan` family of `python` scripts is available. For a full list of the types of scans available, see section 3.3

1.3 Bugs and feedback

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

CHAPTER 2

Installation

2.1 Requirements

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

Although sfincs can be run on a single processor, usually you want to run it in parallel. In this case, you need MPI, and you need either 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. 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:landr` 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 looks for the appropriate makefile in the `sfincs/fortran/version3/makefiles` directory. Second, the `SFINCS_SYSTEM` environment variable affects the behavior of `sfincsScan` in several ways, such as determining the command used to submit jobs to the system's queue.

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

2.4 Setting up `sfincs` on a new system

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

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

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

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

2.5 Compiling

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

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

2.6 Make test

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

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

CHAPTER 3

Input Parameters

In this chapter we first describe all the parameters which can be included in the `input.namelist` file. Then we list some of the command-line flags associated with `PETSc` which can be useful.

3.1 The `general` namelist

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

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

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

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

`outputFileName`
Type: string
Default: "sfincsOutput.h5"
When it matters: Always
Meaning: Name which will be used for the HDF5 output file. If this parameter is changed from the default value, `sfincsScan` will not work.

`saveMatlabOutput`
Type: Boolean

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

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

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

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

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

3.2 The `geometryParameters` namelist

`inputRadialCoordinate`
Type: integer
Default: 3
When it matters: When `geometryScheme`==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. 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.

3.3 Directives for `sfincsScan`

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

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

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

`scanType=2`: Scan of E_r.

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

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

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

3.4 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.

CHAPTER 4

Numerical resolution parameters

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

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

4.1 Relatively unimportant resolution parameters

There are several resolution parameters which are almost never the limiting factor for convergence. You can almost always use `NL=4`, `solverTolerance` = 10^{-5} or 10^{-6}, `xMax=5.0`, and `NxPotentialsPerVth=40.0`. The latter two of these parameters are in fact ignored for the recommended and default `xGridMode` setting, 5.

4.2 General suggestions

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

When the mean-free-path is shorter than the parallel length scale of the equilibrium, (`nuPrime` ≥ 1), the parameters required for convergence do not depend much on collisionality. In the opposite limit in which the mean-free-path is longer than the parallel length scale of the equilibrium, (`nuPrime` ≤ 1) values of `Nzeta` and `Nxi` required for convergence increase dramatically as colli-

sionality decreases. The required value of `Ntheta` increases as well, but often not quite as dramatically. The `Nx` required for convergence does not depend much on collisionality, though typically the required value increases slightly with collisionality at high collisionality

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

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

4.3 Convergence testing

4.4 Typical resolution requirements

CHAPTER 5

Specifying and running a computation

5.1 Normalizations

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

mits any of these four radial coordinates to be used, and different radial coordinates can be used in different parts 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.

To specify a run, you can make several independent choices for radial coordinates. One parameter you select is `inputRadialCoordinate` in the `geometryParameters` namelist. This parameter controls which coordinate is used to specify the flux surface. The possible values of `inputRadialCoordinate` are:

0: Use `psiHat`, the toroidal flux (divided by 2π) normalized by $\bar{B}\bar{R}^2$.

1: Use `psiN`, the toroidal flux normalized by its value at the last closed flux surface.

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

3: Use `rN`, defined as $\sqrt{\texttt{psiN}}$.

5.3 Parallelization

Choosing the number of nodes and procs.

5.4 Issues with running on 1 processor

References

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