Implementation of poloidal density variation in collision operator

Aylwin Iantchenko

December 8, 2017

The aim of this document is to explain the modifications in `SFINCS` that have been done, in order to include poloidal density variation in the collision operator (in both full Fokker-Planck and pure pitch angle scattering operator options).

As is explained in Ref. [1] poloidal density variation can be included by modifying the lowest order distribution function

$$f_{Ms}(\psi) \rightarrow f_{0s} = f_{Ms}(\psi) e^{-Z_s e\Phi_1(\theta,\zeta)/T_s},$$

where $f_M(\psi)$ is the flux-function Maxwellian, Φ_1 is first order variation of the electrostatic potential, and Z_s, T_s are the charge and temperature of species s respectively. To implement this change one may simply modify the species density accordingly

$$n_s \rightarrow n_s e^{-Ze\Phi_1(\theta,\zeta)/T}, \tag{1}$$

in the original equations for the various terms in the collision operator, which are described in Ref. [3]. The linearised collision operator takes the new form

$$\begin{aligned} C_{ab}^{L:f0} = C_{ab}\left\{f_{aM}, f_{bM}\right\} e^{-(Z_a/T_a+Z_b/T_b)e\Phi_1(\theta,\zeta)} + C_{ab}\left\{f_{a1}, f_{bM}\right\} e^{-Z_b/T_b e\Phi_1(\theta,\zeta)} + \\ + C_{ab}\left\{f_{aM}, f_{b1}\right\} e^{-Z_a/T_a e\Phi_1(\theta,\zeta)}, \end{aligned} \tag{2}$$

which describes collisions between species a and b. In normalised `SFINCS` units this becomes

$$\begin{aligned} \hat{C}_{ab}^{L:f0} = \hat{C}_{ab}\left\{f_{aM}, f_{bM}\right\} e^{-\left(\hat{Z}_a/\hat{T}_a+\hat{Z}_b/\hat{T}_b\right)\alpha\hat{\Phi}_1(\theta,\zeta)} + \hat{C}_{ab}\left\{f_{a1}, f_{bM}\right\} e^{-\hat{Z}_b/\hat{T}_b\alpha\hat{\Phi}_1(\theta,\zeta)} + \\ + \hat{C}_{ab}\left\{f_{aM}, f_{b1}\right\} e^{-\hat{Z}_a/\hat{T}_a\alpha\hat{\Phi}_1(\theta,\zeta)}. \end{aligned} \tag{3}$$

The first term on the RHS of Eq. (3) is the *temperature equilibration term*. Since this term does not include f_{1s} it has to be treated differently compared to the other two cases, as will be seen shortly. In the code, the calculation of the residual (see Ref. [2] for definition of the residual and Jacobian in this context) has to be modified to include the factors specified in Eq. (3).

The contribution to the Jacobian are $\delta\hat{C}_{ab}^{L:f0}/\delta\hat{f}_{1s}$ and $\delta\hat{C}_{ab}^{L:f0}/\delta\hat{\Phi}_1$. The first contribution is the same as in Eq. (3) (without the temperature equilibration term) except that the collision operator does not act on the distribution function. In the second case we get

$$\frac{\delta \hat{C}_{ab}^{L:f0}}{\delta \hat{\Phi}_1} = -\left(\hat{Z}_a/\hat{T}_a + \hat{Z}_b/\hat{T}_b\right) \alpha \hat{C}_{ab}\left\{f_{aM}, f_{bM}\right\} e^{-\left(\hat{Z}_a/\hat{T}_a + \hat{Z}_b/\hat{T}_b\right)\alpha\hat{\Phi}_1(\theta,\zeta)}$$
$$-\hat{Z}_b/\hat{T}_b\alpha\ \hat{C}_{ab}\left\{f_{a1}, f_{bM}\right\} e^{-\hat{Z}_b/\hat{T}_b\alpha\hat{\Phi}_1(\theta,\zeta)} - \hat{Z}_a/\hat{T}_a\alpha\hat{\Phi}_1(\theta,\zeta)\hat{C}_{ab}\left\{f_{aM}, f_{b1}\right\} e^{-\hat{Z}_a/\hat{T}_a\alpha\hat{\Phi}_1(\theta,\zeta)}. \tag{4}$$

Apart from the different pre-factor, the term in Eq. (4) differs from the term in $\delta\hat{C}_{ab}^{L:f0}/\delta\hat{f}_{1s}$ by acting on the first order distribution function f_{1s}. For example, in the first case, terms such as $\partial f_{1s}/\partial x_s$ become

$$\frac{\partial}{\partial f_{1s}}\frac{\partial f_{1s}}{\partial x_s} = \frac{\partial}{\partial x_s},$$

whereas in the second case, when we take the derivative with respect to Φ_1, we have to include the derivative $\partial f_{1s}/\partial x_s$ (and $\partial f_{0s}/\partial x_s$ for temperature equilibration, which again is not present in the first contribution), as well.

Implementation in the Code

To implement Eq. (3) and Eq. (4) we need to modify the evaluation of the residual and the Jacobian. The residual is evaluated in `evaluateResidual.F90` and the Jacobian in `evaluateJacobian.F90`. Both these two routines call the functions in `PopulateMatrix.F90`, where the actual assembly of the matrices used to compute the residual (by multiplying the matrix with the state-vector) and Jacobian, is done. As a consequence of this structure, we only need to modify `PopulateMatrix.F90` in order to include poloidal density variation in the collision operator.

Residual

In the residual we have to include the extra pre-factor appearing in Eq. (3). Since this factor is species specific, we have to keep track of which species density is appearing in the equations. The pitch-angle scattering and the $\hat{C}_{ab}^{E}\left\{f_{a1}\right\}$ parts of the collision operator (see Ref. [3]) both uses the density `nHats(iSpeciesB)` while the other contributions to the total collision operator `CHat` uses `nHats(iSpeciesA)` instead.

Rather than creating many if-statements in the current implementation of the collision operator, for sake of clarity we separate the two implementations with one if-statement directly before the various contributions to the collision operator are calculated. If `Phi1` is included and `poloidalVariationInCollisionOperator = .true.` the contribution from the collision operator is calculated taking poloidal density variation into account.

```
if (poloidalVariationInCollisionOperator .and. includePhi1 .and.
    includePhi1InKineticEquation) then
```

Otherwise the second block is executed, which contains the original code.
Since we now will have a `Phi1Hat` dependence in the various terms of the collision operator we will have to iterate over `itheta` and `izeta` when calculating the various contributions. Before calculating each term, we define a pre-factor (`PreFactor`) as is specified in Eq. (3), taking the correct species index into account.

```
do itheta=ithetaMin,ithetaMax
   do izeta=izetaMin,izetaMax
      ! Generate preFactor for nHats(iSpeciesA) terms
      preFactor =
          exp(-Zs(iSpeciesA)*alpha*Phi1Hat(itheta,izeta)/Thats(iSpeciesA))
```

When required, this pre-factor is overwritten using the charge and temperature of `iSpeciesB` instead. Because of the `itheta`, `izeta` dependence we have replaced `CECD` and `nuDHat` to their equivalents `CECDpol` and `nuDHatpol` with space for their `itheta` and `izeta` components.

The rest of this block is the same as in the original code, except that we do the second iteration over `itheta`, `izeta` slightly earlier, and include the appropriate `PreFactor` in front of additional contributions to the total collision operator `CHat`.

Temperature equilibration term

In the original version of the code, the temperature equilibration term is calculated in `evaluateResidual.F90` by using `whichMatrix = 2` when calling `PopulateMatrix.F90`. The output is then multiplied with f_{0s}. Since the original version of `SFINCS` already includes Φ_1 in the definition of f_{0s} (in the subroutine `init_f0()` inside `PopulateMatrix.F90`), the same routines as were presented above can be used also if `includeTemperatureEquilibrationTerm = .true.`.

Jacobian

If the modifications explained in the previous section are implemented, the $\delta C/\delta f_{1s}$ contribution to the Jacobian is at this point already taking the density variation into account. We need to add the calculation of $\delta C/\delta \Phi_1$.

The $\delta C/\delta \Phi_1$ term

We begin with treating the two latter terms appearing in Eq. (4) and will then look at the temperature equilibration term.

To implement the derivative with respect to Φ_1 we need to redo the calculation of the various contributions to the collision operator, but including a new pre-factor, `PreFactorJ` as specified in Eq. (4). This should only be done when the Jacobian (`whichmatrix =1`) or the preconditioner (`whichmatrix =0`) is calculated.

```
if (whichMatrix == 1 .or. whichMatrix == 0) then
   ! Generate preFactorJ for nHats(iSpeciesB) terms
   preFactorJ = (-Zs(iSpeciesA)*alpha/Thats(iSpeciesA)) &
   *exp(-Zs(iSpeciesA)*alpha*Phi1Hat(itheta,izeta)/Thats(iSpeciesA))
```

Again, the same is done using the charge and temperature of `iSpeciesB` instead, whenever required. The result is saved in the new matrices `nuDHatpolJ`, `CECDpolJ` etc. Same calculations are then repeated for these terms (except for the different pre-factor), and the result is saved in `CHatJ`.

In contrary to the residual (and the $\delta C/\delta f_{1s}$ terms), when calculating $\delta C/\delta \Phi_1$ we have to include the first order distribution function before saving the result into the main matrix. For this purpose we first have to initiate this distribution function, by reading the appropriate terms in the current state vector.

```
 do ix= max(ixMinCol,min_x_for_L(L)),Nx
    ! Generate f1b from state vector
    index = getIndex(iSpeciesB,ix,L+1,itheta,izeta,BLOCK_F)
    f1b(ix) = stateArray(index + 1)
```

We multiply the terms in the collision operator with this distribution function.

```
  CHatTimesf = matmul(CHatJ,f1b)
```

The result is then saved into the $\delta\hat{C}/\delta\hat{\Phi}_1$ block of the main matrix,

```
do ix_row=max(ixMin,min_x_for_L(L)),Nx
   rowIndex=getIndex(iSpeciesA,ix_row,L+1,itheta,izeta,BLOCK_F)
   ! Get column index for the d/dPhi1 terms
   colIndex=getIndex(1,1,1,itheta,izeta,BLOCK_QN)
   ! Save into the main matrix, note that here we only use ix_row since
      CHatTimesf is now a vector
   call MatSetValue(matrix, rowIndex, colIndex, &
   -nu_n*CHatTimesf(ix_row), ADD_VALUES, ierr)
   ! need to use MatSetValue, otherwise petsc gives error
end do ! ix_row
```

Temperature equilibration

If the temperature equilibration terms should be included, we have to add additional terms into the main matrix, when calculating $\delta\hat{C}/\delta\hat{\Phi}_1$. These terms are different from the other contributions since now we need to multiply the output with $f_{0s} = f_{Ms}(\psi)e^{-Z_s e\Phi_1(\theta,\zeta)/T_s}$ (instead of the first order distribution function f_{1s}). Because of this, we need to generate the Maxwellian distribution function for `iSpeciesB`.

```
if (includeTemperatureEquilibrationTerm .and. L==0) then
   fM(ix) =
      sqrt(mhats(iSpeciesB)/Thats(iSpeciesB))*mhats(iSpeciesB)/Thats(iSpeciesB)&
   * nhats(iSpeciesB)/(pi*sqrtpi)*expxb2(ix)
end if
```

Because of the `Phi1Hat` dependence in the zeroth order distribution function we get a different contribution to the Jacobian, as specified in the first term of Eq. (4). In the code, we need to use both the residual term `CHat` and the Jacobian terms `CHatJ` to obtain the correct result.

```
CHatTimesf = matmul((CHatJ + CHat*(-Zs(iSpeciesB)*alpha/Thats(iSpeciesB))&
 )*exp(-Zs(iSpeciesB)*alpha*Phi1Hat(itheta,izeta)/Thats(iSpeciesB)),fM)
```

Written in terms of operators we have that `CHat` is $\hat{C}_{ab}\left\{f_{aM},\cdot\right\}e^{-\hat{Z}_a/\hat{T}_a\alpha\hat{\Phi}_1(\theta,\zeta)}$ and `CHatJ` is $\hat{C}_{ab}\left\{f_{aM},\cdot\right\}\left(-\hat{Z}_a/\hat{T}_a\alpha\right)e^{-\hat{Z}_a/\hat{T}_a\alpha\hat{\Phi}_1(\theta,\zeta)}$. Here $\cdot$ means that the operator does not act on anything. What is left is to add this term into the main matrix, using the the same procedure as was explained in the previous section.

Pure pitch angle scattering operator

The implementation of poloidal density variation in the case the pure pitch angle scattering collision operator is used (corresponding to the case `collisionoperator = 1`), is slightly easier because of the existing loop over `itheta`, `izeta` and since we are working with a scalar `CHat_element`. Also, since $L > 0$ always for this term, no temperature equilibration has to be included. The only block we need to change is when we save the collision operator into the main matrix.

We introduce the factor `preFactor` which is in normal cases equal to one, but when `poloidalVariationInCollisionOperator = .true.` it is equal to the exponential prefactor containing Φ_1 in Eq. (3).

Before saving into the main matrix, we multiply the collision operator with this factor.

```
call MatSetValue(matrix, index, colIndex, &
  -nu_n*CHat_element*preFactor, ADD_VALUES, ierr)
```

Jacobian, the $\delta C/\delta\Phi_1$ term

Just as before, the $\delta C/\delta f_1$ terms are the same as in the residual. What is different is the $\delta C/\delta\Phi_1$ term. The procedure to calculate this term is however very similar, to the case with the residual. The only difference is the different pre-factor which now also is multiplied with the distribution function from the state vector.

```
preFactor = -Zs(iSpeciesA)*alpha/Thats(iSpeciesA)*exp(-Zs(iSpeciesA) &
        *alpha*Phi1Hat(itheta,izeta)/Thats(iSpeciesA))*stateArray(index + 1)
```

Current state of the project

Presently, the modifications discussed in the previous section have been implemented into the “poloidalVariationInCollisionOperator” branch of `SFINCS`. What remains is to verify that the new implementation is correct. First, we try to make simple tests and compare the output from the new code (with poloidal density variation included in the collision operator), with the original code (without poloidal density variation in the collision operator). This is discussed in the first subsection below. Next, we attempt to benchmark the output from the code with theoretical predictions. This is done in the second subsection.

Tokamak geometry has been used in all cases considered here.

Tests of new code

First, for simple scenarios (such as one species cases, or two species but with low impurity charge, $Z \leq 4$), comparison between the new and old version of the code gives exactly the same results. The time it takes to run a simulation is practically the same.

Next, we try to increase the effect of poloidal density variation in the collision operator, by increasing the impurity charge. For $Z \leq 16$ this leads to a much longer computation time, but eventually the solver converges and we get a solution. When attempting to run with $Z = 17$, the nonlinear solver diverges. It is possible to get a solution by using `reusePreconditioner = .false.` in the input to `SFINCS` and by changing the relative and absolute tolerances used in the iterations. This can be done by providing the flags `-ksp_atol` and `-snes_rtol` when running the `SFINCS` simulation. Example of values that can be used are $1e-8$ for `-ksp_atol` and $1.0e-6$ for `-snes_rtol`. With these options, the nonlinear solver should eventually converge. Note that the other input parameters, such as geometry, species and resolution parameters, all have a strong impact on whether the solver will converge or not. Typical input parameters used here will be presented in the next subsection, further below.

Even for these more complicated cases, with high impurity charge, there is still practically no difference between using the new code and the original version. This is demonstrated in Fig. 1 where the perturbed impurity density is shown for $Z = 1$ and $Z = 17$ respectively. Both cases have been computed using the new and original version of the code. As is seen in the figure, the output from the old and new version of the code is almost identical.

Other cases, when changing for example impurity density, have also been studied. All have the same conclusion that poloidal density variation does not seem to matter in the collision operator, at least when Tokamak geometry is considered. Further work with Stellarators might be more interesting and should be done.

Comparing with model

The second approach we take to validate the code, is to benchmark the `SFINCS` output with theoretical predictions, using for example the model described in Ref. [4][1].

The first step is to test the model with the output from the original `SFINCS` code, without including poloidal density variation. By setting the temperature gradient to zero we remove the effect from the collision operator and expect therefore the output from `SFINCS` to agree reasonably well with the predictions from the model. Example of typical `SFINCS`-input used to get the best agreement so far is:

[1]Note that in this model the collision operator is not linearised around a poloidally varying zeroth order distribution function (as is done here), but the poloidal variation enters through the perturbed distribution function in the ion-impurity collision operator, when computing the parallel friction force.

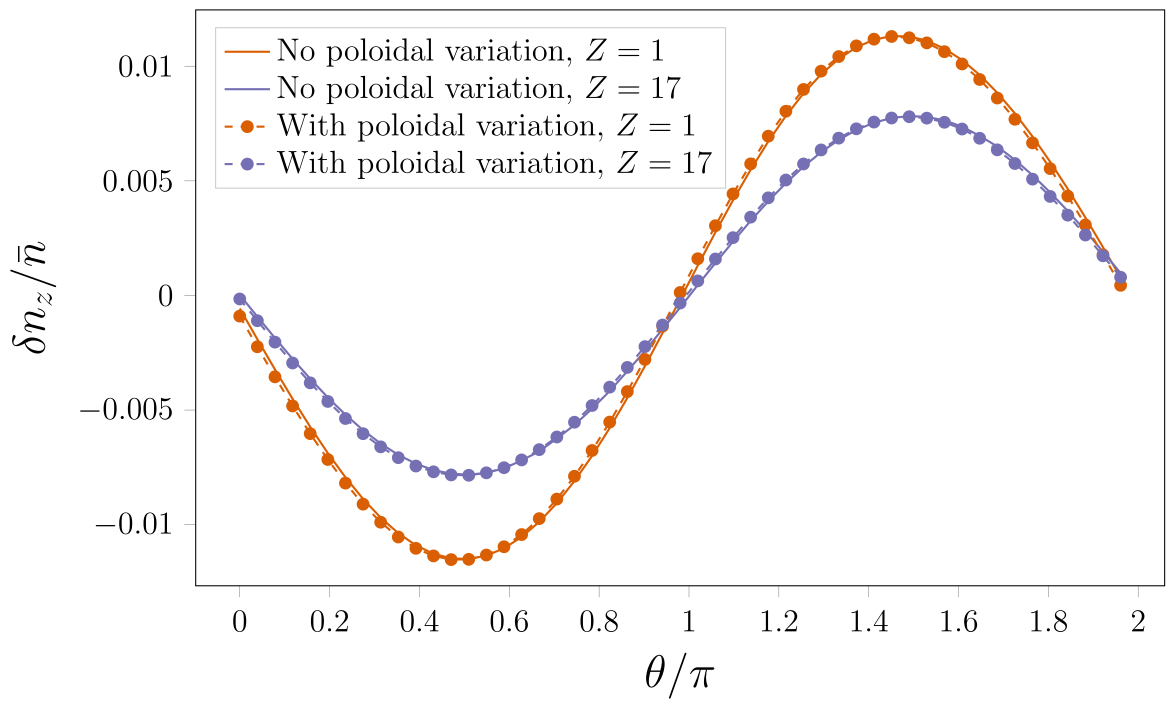

Figure 1: Normalised perturbed impurity density for impurity charge $Z = 1$ (orange) and $Z = 17$ (violet) respectively. The result is shown for the case when the original code is used (solid lines) and when the new code is used instead, with poloidal density variation included in the collision operator(dashed line with circles).

```
 &speciesParameters
 Zs = 1.0, 20.0
 mHats = 1.0, 20.0
 nHats = 4.0, 0.012
 THats = 2.0, 2.0
 dnHatdrNs = -40.0, -0.12
 dTHatdrNs = -0.0, -0.0

 &geometryParameters
 inputRadialCoordinate = 3
 inputRadialCoordinateForGradients = 3

 rN_wish = 0.3

 geometryScheme = 1
 B0OverBBar = 60.0
 GHat = 180.0
 IHat = 0.9
 iota = 0.5
 epsilon_t = 0.1
 epsilon_h = 0
 epsilon_antisymm = 0
 helicity_l = 1
 helicity_n = 1

 psiAHat = 30.0
 aHat = 1.0
/
```

For this input, the orderings of various parameters described in Ref. [4] are reasonably satisfied (we have $\Delta \sim 0.3, \delta \sim 0.15, \epsilon \sim 0.1, n_z Z^2/n_i \sim 1.2$ and $\nu^* \sim 0.11$). If we use this input for varying magnetic field strength B_0 (as an example) and calculate the sine and cosine parts of the perturbed impurity density, we get Fig. 2 and Fig. 3 respectively. Solid lines correspond to the case when SFINCS output is used to calculate the perturbed density while dashed lines correspond to the analytical model. The two outputs show good agreement for the cosine parts while the sine is slightly deviating.

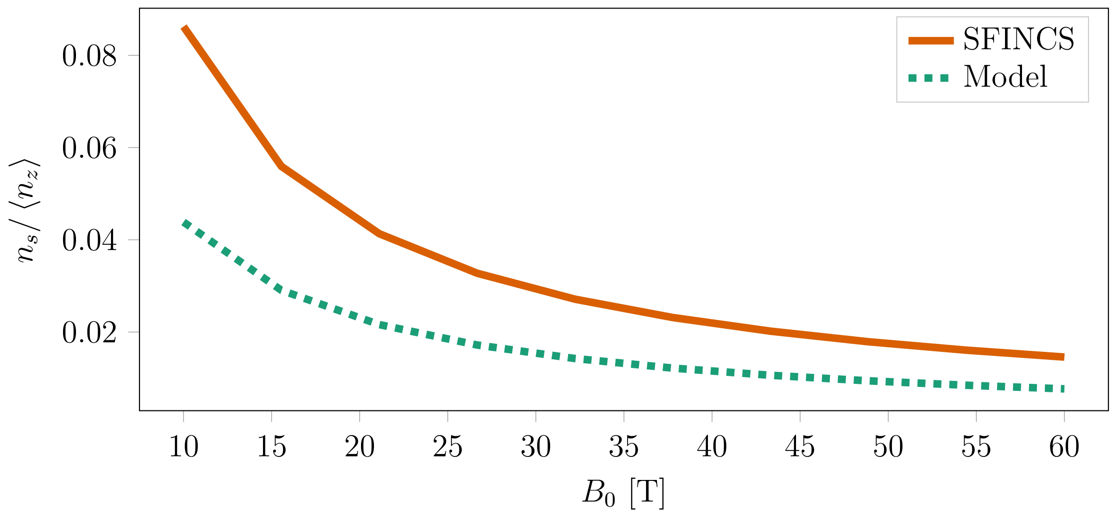

Figure 2: Sine part of the perturbed density for a scan in the magnetic field strength B_0. Solid line shows result calculated from `SFINCS` output while the dashed line is the prediction from the model.

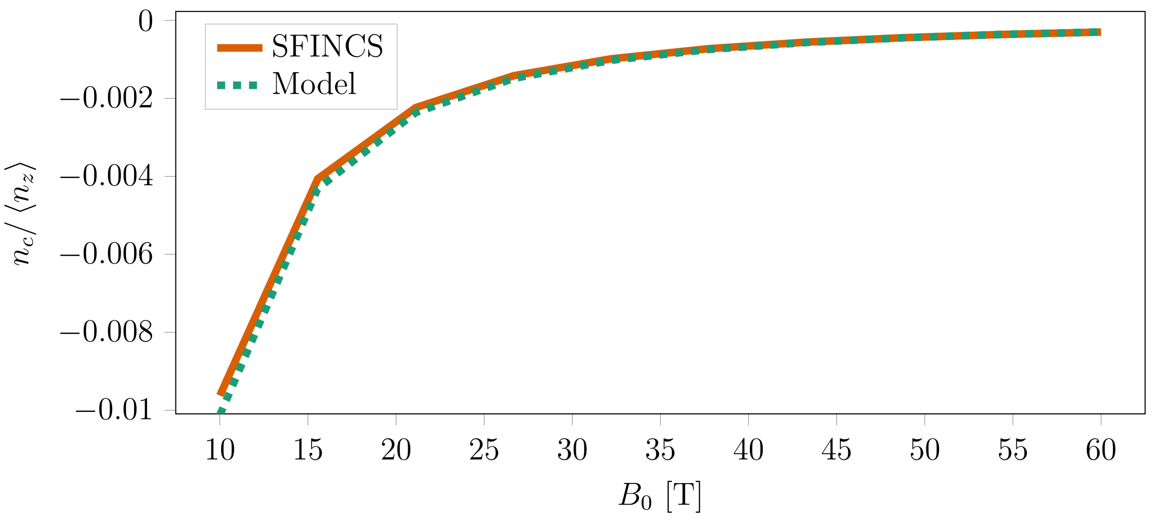

Figure 3: Cosine part of the perturbed density for a scan in the magnetic field strength B_0. Solid line shows result calculated from `SFINCS` output while the dashed line is the prediction from the model.

Scans in ϵ shows a similar result, but a scan in impurity charge and density show more disagreement between model predictions and output from `SFINCS`. Similar tests using the new version of the code (with nonzero temperature gradients) show the same result.

It appears to be challenging to get good agreement between the model and `SFINCS`. A possible explanation is the difficulty to get asymptotic in the various quantities required for the model to be valid, while maintaining a sufficiently large density perturbation to make a meaningful comparison. As mentioned, here we use $\Delta \sim 0.3, \delta \sim 0.15, \epsilon \sim$

$0.1, n_zZ^2/n_i \sim 1.2$ and $\nu^* \sim 0.11$ which might not be good enough for the theoretical model to be valid.

Comparison between `SFINCS` and the model has been done using a series of python scripts available on GitHub. Further work is required and welcome.

References

[1] S. Buller, *Poloidal variation in collision operator* (2017).

[2] A. Mollén, *Implementation of Φ_1 in SFINCS* (2016).

[3] M. Landreman, *Technical Documentation for version 3 of SFINCS* (2014).

[4] T. Fülöp, P. Helander *Phys. Plasmas* **6**, 3066 (1999)

.