The parallel derivative on structured grids

M. Wiesenberger

July 27, 2018

1 Semi-Lagrangian schemes

In this section we show how to numerically treat parallel derivatives in a non field aligned coordinate system. We introduce the method in Section 1.1 and discuss some problems arising from the boundaries of the computational domain. Finally, we propose a field line mapping for variable initialization in Section 4.

1.1 Discretization of parallel derivatives

Given is a vector field $\mathbf{v}(R,Z)$ in cylindrical coordinates R,Z,φ independent of φ and we want to discretize the derivative $\mathbf{v}\cdot\nabla f \equiv \nabla_\parallel f$. This $\mathbf{v}$ might be the magnetic unit vector field but the algorithm works for any vector field $\mathbf{v}$ with $v^\varphi \neq 0$, in especially $\mathbf{v}$ does not need to have unit length.

We begin with the formulation of a field-aligned discretization. If s denotes the field line following coordinate, then the one-dimensional discrete derivative along the field line reads

$$\frac{\mathrm{d}f}{\mathrm{d}s} \to \frac{f_{k+1}-f_{k-1}}{s_{k+1}-s_{k-1}}. \tag{1}$$

Note here that s does NOT denote the distance (especially since we do not require the existence of a metric at this point).

To every smooth vector field $\mathbf{v}(\mathbf{x})$ there is a unique curve of which the tangent in a point p is the value of $\mathbf{v}(p)$ at that point. It is given by the solution of the differential equation

$$\frac{\mathrm{d}z^i}{\mathrm{d}s} = v^i(\mathbf{z}) \tag{2}$$

where z^i is one of (R,Z,φ) and v^i are the contravariant components of $\mathbf{v}$ in cylindrical coordinates. Moreover, by definition we have

$$\frac{\mathrm{d}f(\mathbf{z}(s))}{\mathrm{d}s} = \mathbf{v}\cdot\nabla f|_{\mathbf{z}(s)} \tag{3}$$

along a field line parameterized by s, i.e. instead of $\nabla_\parallel f$ we can choose to discretize $\frac{\mathrm{d}f}{\mathrm{d}s}$.

Let us divide the φ direction into N_φ equidistant planes of $\Delta\varphi$. Unfortunately, from Eq. (2) we cannot easily determine Δs for given $\Delta\varphi$. It is better to use the transformation $\mathrm{d}t = v^\varphi \mathrm{d}s$

$$\frac{\mathrm{d}z^i}{\mathrm{d}t} = \frac{v^i}{v^\varphi} \tag{4}$$

since in this case $\mathrm{d}\varphi/\mathrm{d}t = 1 \Rightarrow t = \varphi$. We get

$$\frac{\mathrm{d}R}{\mathrm{d}\varphi} = \frac{v^R}{v^\varphi}, \tag{5a}$$

$$\frac{\mathrm{d}Z}{\mathrm{d}\varphi} = \frac{v^Z}{v^\varphi}, \tag{5b}$$

together with the equation

$$\frac{\mathrm{d}s}{\mathrm{d}\varphi} = \frac{1}{|v^\varphi|} \tag{5c}$$

for s. Eqs. (5) are integrated from $\varphi = 0$ to $\varphi = \pm\Delta\varphi$. We characterize the flow generated by $\mathbf{v}/v^\varphi$ by

$$\mathcal{T}^\pm_{\Delta\varphi}\mathbf{z} := \mathcal{T}^\pm_{\Delta\varphi}[R, Z, \varphi] := (R(\pm\Delta\varphi), Z(\pm\Delta\varphi), \varphi \pm \Delta\varphi), \tag{6}$$

where $(R(\varphi), Z(\varphi), s(\varphi))$ is the solution to Eqs. (5) with initial condition

$$(R(0), Z(0), s(0)) = (R, Z, 0). \tag{7}$$

Obviously we have $\mathcal{T}^-_{\Delta\varphi} \circ \mathcal{T}^+_{\Delta\varphi} = \mathbf{1}$, but $\mathcal{T}^\pm_{\Delta\varphi}$ is not unitary since $\mathbf{v}/v^\varphi$ is not divergence free.

The proposed centered discretization (1) for the parallel derivative then reads

$$\nabla_\parallel f \equiv \frac{df}{ds} = \frac{df}{d\varphi}\frac{d\varphi}{ds} \to \frac{f\left(T^+_{\Delta\varphi}\mathbf{z}\right) - f\left(T^-_{\Delta\varphi}\mathbf{z}\right)}{s(+\Delta\varphi) - s(-\Delta\varphi)}, \tag{8}$$

which is slightly different from Reference [1], where the relation (5c) was used to replace $\mathrm{d}\varphi/\mathrm{d}s$. Since the R and Z coordinates are still discretized in the dG framework we note that in our work the interpolation of f on the transformed points $\mathcal{T}^\pm_{\Delta\varphi}\mathbf{z}$ is naturally given by interpolating the base polynomials. Let us for a moment omit the Z coordinate for ease of notation. If (R_{nj}, φ_k) are the grid points, we call $(R^+_{nj}, \varphi_{k+1}) := \mathcal{T}^+_{\Delta\varphi}[R_{nj}, \varphi_k]$ and $(R^-_{nj}, \varphi_{k-1}) := \mathcal{T}^-_{\Delta\varphi}[R_{nj}, \varphi_k]$ the transformed coordinates along the field lines. We then have

$$f(\mathcal{T}^+_{\Delta\varphi}\mathbf{z}) = f(R^+_{nj}, \varphi_{k+1}) = \bar{f}^{ml}_{k+1} p_{ml}(R^+_{nj}) =: (I^+)^{ml}_{nj} f_{(k+1)ml}, \tag{9a}$$

$$f(\mathcal{T}^-_{\Delta\varphi}\mathbf{z}) = f(R^-_{nj}, \varphi_{k-1}) = \bar{f}^{ml}_{k-1} p_{ml}(R^-_{nj}) =: (I^-)^{ml}_{nj} f_{(k-1)ml}, \tag{9b}$$

where the backward transformations of $\bar{\mathbf{f}}$ are hidden in I. Thus the interpolation of all the necessary points can simply be written as a matrix-vector product, where the interpolation matrices I^+ and I^- are independent of time since the field lines are constant in time. The order of this

interpolation is given by P, the number of polynomial coefficients. A consistency check is the relation $I^+ \circ I^- = \mathbf{1}$.

The discretization (8) can now be written as a matrix vector product

$$\nabla_\parallel f \to S \circ \left[\mathbf{1}^+ \otimes I^+ - \mathbf{1}^- \otimes I^-\right] \mathbf{f}, \tag{10}$$

where S is the diagonal matrix that contains the entries $1/(s(+\Delta\varphi) - s(-\Delta\varphi))$. This discretization is not skew-symmetric since the field lines are not volume-preserving, or $(I^+)^{\mathrm{T}} \neq I^-$. In fact, the adjoint of the parallel derivative is

$$\nabla_\parallel^\dagger f = -\nabla \cdot (\mathbf{v}\ f) \neq -\nabla_\parallel f. \tag{11}$$

Note that with this relation we can define the parallel diffusion operator as

$$\Delta_\parallel := -\nabla_\parallel^\dagger \nabla_\parallel = (\nabla \cdot \hat{\mathbf{v}})\nabla_\parallel + \nabla_\parallel^2, \tag{12}$$

which is indeed the parallel part of the full Laplacian $\Delta = \nabla \cdot (\hat{\mathbf{v}}\nabla_\parallel + \nabla_\perp)$. $\hat{\mathbf{v}}$ is the unit vector $\mathbf{v}/|\mathbf{v}|$. The second order derivative $\nabla_\parallel^2$ can be discretized using

$$\begin{aligned}\frac{\mathrm{d}^2 f}{\mathrm{d}s^2} \to \frac{2f_{k+1}}{(s_{k+1}-s_k)(s_{k+1}-s_{k-1})} - \frac{2f_k}{(s_{k+1}-s_k)(s_k-s_{k-1})} \\ + \frac{2f_{k-1}}{(s_k-s_{k-1})(s_{k+1}-s_{k-1})}\end{aligned} \tag{13}$$

and repeating the procedure of this section.

1.2 Change of coordinates

In principle the above considerations hold in any coordinate system η, ζ, φ, since the directional derivative is an intrinsic operation. The only question is how to integrate the field lines in the η, ζ, φ system since we assumed that our vector field $\mathbf{v}(\mathbf{x})$ was given analytically in cylindrical coordinates. There are two possibilities. First, interpolate $R(\zeta_i, \eta_i), Z(\zeta_i, \eta_i)$ for all i, then integrate $\mathbf{v}$ in (R, Z) space and finally use Newton iteration to find $\zeta(R_i^\pm, Z_i^\pm), \eta(R_i^\pm, Z_i^\pm)$. The downside here is that it is difficult to tell when and where the fieldline leaves the simulation domain.

The second possibiliy (the one currently implemented) is to integrate entirely in the transformed coordinate system ζ, η, φ. The magnetic field can be easily transformed since we have the Jacobian of the coordinate transformation

$$v^\zeta(\zeta,\eta) = \left(\frac{\partial\zeta}{\partial R}v^R + \frac{\partial\zeta}{\partial Z}v^Z\right)_{R(\zeta,\eta),Z(\zeta,\eta)} \tag{14}$$

$$v^\eta(\zeta,\eta) = \left(\frac{\partial\eta}{\partial R}v^R + \frac{\partial\eta}{\partial Z}v^Z\right)_{R(\zeta,\eta),Z(\zeta,\eta)} \tag{15}$$

$$v^\varphi(\zeta,\eta) = v^\varphi(R(\zeta,\eta), Z(\zeta,\eta)) \tag{16}$$

The fieldline equations (5) are still

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\varphi} = \frac{v^\zeta}{v^\varphi} \tag{17a}$$
$$\frac{\mathrm{d}\eta}{\mathrm{d}\varphi} = \frac{v^\eta}{v^\varphi} \tag{17b}$$
$$\frac{\mathrm{d}s}{\mathrm{d}\varphi} = \frac{1}{|v^\varphi|} \tag{17c}$$

The issue here is that when integrating fieldlines we have to interpolate the vector field $\mathbf{v}$ at arbitrary points. However, the interpolation error vanishes with order P in the perpendicular plane. In order to mitigate this error we transform $\mathbf{v}$ on a finer grid/higher order polynomials for more accurate integration. Apart from the issue of how to get the transformed vector field the remaining algorithm for $\mathbf{v}\cdot\nabla$ is entirely unchanged.

1.3 Boundary conditions

The question is what to do when a fieldline intersects with the boundary of the simulation domain before reaching the next plane. Boundary conditions are formulated by either setting a value on the boundary of the domain (Dirichlet) or by fixing the derivative perpendicularly to the boundary (Neumann), or a combination of both (Robin).

The problem with a Dirichlet boundary condition is that we need to find the exact place where the fieldline reaches the boundary. We have to find φ_b such that the result of the integration of Eq. (5) from 0 to φ_b lies on the boundary. The angle φ_b can be found by a bisection algorithm knowing that $0 < \varphi_b < \Delta\varphi$. This kind of procedure is known as a shooting method. Secondly, for Dirichlet boundaries the small distance of a point to the wall seriously deteriorates the CFL condition. (To ease the CFL condition was the reason to devise the algorithm in the first place)

The problem with Neumann boundaries is that they are usually given perpendicularly to the boundary and that the fieldlines are not necessarily perpendicular to the boundary.

1.4 Avoiding boundary conditions

An obvious way to avoid boundary conditions is to align the simulation domain to the vector field such that the fieldlines never intersect the boundary. This is possible in FELTOR (read the geometry section).

When computing in non-aligned coordinate systems one idea to avoid boundary conditions is to simply cut the contribution from field lines that leave the computational domain. While this works in practice it is unclear what numerical and physical side-effects this procedure might have.

Another solution would be to change the vector field $\mathbf{v}$ and only retain the toroidal part of $\mathbf{v}$ on the boundary ($v^R|_{\partial\Omega} = v^Z|_{\partial\Omega} = 0$). The fieldlines then have a kink on the boundary $\partial\Omega$. On the other hand we can implement boundary conditions consistent with the perpendicular ones since the fieldlines never leave the domain. We simply interpolate the quantity to derive on the inner side of the domain boundary (Neumann conditions = "No boundary condition") or set the value to zero (Dirichlet condition).

1.5 Poloidal limiters

A poloidal limiter can simply be implemented via a boundary condition in φ. As long as the form of the limiter is aligned with a flux-function we do not have to integrate a field line in order to determine which points lie in the limiter-shadow. It is therefore straightforward to implement ghost-cells in that case.

2 The adjoint methods

The idea is to discretize the operation $\nabla \cdot (\mathbf{v}.)$ by taking the adjoint of the discretization for $\nabla_\parallel$. In order to understand what the adjoint operators do let us denote $\mathcal{T}^+_{\Delta\varphi}$ as the push-forwward operator. Then we have

$$\int f(\mathbf{x})\mathcal{T}^+_{\Delta\varphi}h(\mathbf{x})\sqrt{g(\mathbf{x})}\mathrm{d}^3x = \int \frac{1}{\sqrt{g(\mathbf{x}')}}\mathcal{T}^-_{\Delta\varphi}\left[J^{-1}(\mathbf{x}')\sqrt{g(\mathbf{x}')}f(\mathbf{x}')\right]h(\mathbf{x}')\sqrt{g(\mathbf{x}')}\mathrm{d}^3x' \tag{18}$$

$$\equiv \int (\mathcal{T}^+_{\Delta\varphi})^\dagger\left[f(\mathbf{x})\right]h(\mathbf{x})\sqrt{g(\mathbf{x})}\mathrm{d}^3x \tag{19}$$

J is the determinant of the Jacobian $\partial(\mathbf{x}')/\partial(\mathbf{x})$ with $\mathbf{x}' = \mathcal{T}^-_{\Delta\varphi}\mathbf{x}$. In the last step we simply replaced the dummy variable $\mathbf{x}'$ with $\mathbf{x}$ again and identified the relevant terms as the adjoint operator:

$$(\mathcal{T}^+_{\Delta\varphi})^\dagger f(\mathbf{x}) := \frac{1}{\sqrt{g(\mathbf{x})}}\mathcal{T}^-_{\Delta\varphi}\left[\sqrt{g(\mathbf{x})}J^{-1}(\mathbf{x})f(\mathbf{x})\right] \tag{20}$$

This means that numerically the adjoint of the push-forward operator should be a valid discretization of its inverse. Note that $\sqrt{g}J^{-1}(\mathbf{x}) = \sqrt{g'(\mathcal{T}^-_{\Delta\varphi}\mathbf{x})}$. With this we can write

$$(\mathcal{T}^+_{\Delta\varphi})^\dagger f(\mathbf{x}) := \sqrt{\frac{g'(\mathbf{x})}{g(\mathbf{x})}}\mathcal{T}^-_{\Delta\varphi}\left[f(\mathbf{x})\right] \tag{21}$$

Also note that $\mathcal{T}^+_{\Delta\varphi}[fh] = \mathcal{T}^+_{\Delta\varphi}f\mathcal{T}^+_{\Delta\varphi}h$ might not hold on the discrete level. Also the question is how J enters on the discrete level. We have to multiply $\sqrt{g}$ artificially when we form the adjoint. Theoretically J could be hidden somehow when we integrate the fieldlines, so the information could be contained in the discrete version? (Maybe in the back-projection?)

If the streamlines are divergence free, we have $J = 1$. A numerical test could be (if we neglect the volume form in the adjoint)

$$(\mathcal{T}^+_{\Delta\varphi})^\dagger\left[J(\mathbf{x})\mathcal{T}^+_{\Delta\varphi}f(\mathbf{x})\right] - f(\mathbf{x}) = 0 \tag{22}$$

The numerical computation of J might a bit tricky at the boundaries. In a flux-aligned ζ, η it should be feasible but in cylindrical coordinates I don't know how. Maybe we can simply cut the last few cells before the boundary. Even easier might be

$$\left(\mathcal{T}^+_{\Delta\varphi}\right)^\dagger J(\mathbf{x}) = 1 \tag{23}$$

If we integrate streamlines of the vector field $\mathbf{B}/B^\varphi$, then we have

$$\frac{\mathrm{d}J}{\mathrm{d}\varphi} = J(\mathbf{x})\nabla\cdot(\mathbf{B}/B^\varphi) \tag{24}$$

along these streamlines. Also we have that $\mathrm{d}s = B/B^\varphi \mathrm{d}\varphi$ and the interpolation/projection of $\triangle s$ can probably be neglected. (We want to neglect it because it's memory intensive to store all combinations) Also this means that when we transpose $\nabla_\parallel$ we get a multiplication by B^φ/B in the beginning. In any case this means that we want to have

$$\left(\mathcal{T}^+_{\Delta\varphi}\right)^\dagger B^\varphi = B^\varphi \tag{25}$$

2.1 A grid refinement approach

While the idea of the last section sounds appealing the problem is that the discretization does not converge. The idea of the sandwich method [**?**] is to bracket the parallel derivative by interpolation and projection matrices:

$$\nabla^c_\parallel = P\nabla^f_\parallel Q \tag{26}$$
$$\nabla^{c\dagger}_\parallel = P\nabla^{f\dagger}_\parallel Q \tag{27}$$

In this way the projection integrals

$$\int \mathrm{dV}\,(\nabla_\parallel f) p_i(x) p_j(y) \tag{28}$$

are computed more precisely. The size of the fine grid should therefore be as large as possible. We first notice that the one interpolation matrix can be absorbed in the parallel derivative since this also consists of interpolation operations.

$$\nabla^c_\parallel = P\nabla^{fc}_\parallel \tag{29}$$
$$\nabla^{c\dagger}_\parallel = \nabla^{fc\dagger}_\parallel Q \tag{30}$$

Note that the matrix-matrix multiplications in Eq. (30) can be precomputed and stored. The memory requirements in the final computations are therefore the same as in the old version. (Not entirely, since the diagonal $1/\Delta s$ matrix does not commute with Q or P).

Finally remember that the adjoint of a matrix in the modified geometry involves the volume element. This means that after you've adjoined the parallel derivative the normal way simply bracket the result by $1/\sqrt{g}$ and $\sqrt{g}$.

3 Algorithm

Given are the components $v^i(R,Z)$ for $i \in \{R,Z,\varphi\}$ and a compuational grid (in the following the "coarse grid")

- generate a fine grid by multiplying the cell numbers of the given coarse grid (only topologcially, metric and Jacobian are not needed)
- integrate the fieldlines for the fine grid:
  - evaluate the starting points on the coarse grid in computational space
  - For a curvilinear grid set up a (higher order) grid for the interpolation of the vector components v^i and push forward the vector components to the curvilinear coordinate system
  - Integrate the fieldline equations

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\varphi} = \frac{v^\zeta}{v^\varphi} \tag{31a}$$
$$\frac{\mathrm{d}\eta}{\mathrm{d}\varphi} = \frac{v^\eta}{v^\varphi} \tag{31b}$$
$$\frac{\mathrm{d}s}{\mathrm{d}\varphi} = \frac{1}{|v^\varphi|} \tag{31c}$$

    with the given starting points and $s(0) = 0$ from $\varphi = 0$ until $\varphi = \pm\Delta\varphi$.
  - create an interpolation matrix that interpolates from the coarse grid to the fine grid
  - use the interpolation matrix to generate the plus/minus points for the fine grid
- create the interpolation matrices that interpolate from the given coarse grid to the plus/minus points
- create a projection matrix that projects from the fine grid to the coarse grid
- compute the matrix-matrix multiplications $P \cdot I^\pm$ as well as the transpose
- project the s vectors to the coarse grid

3.1 Notes on the MPI implmentation

We note that we normally construct $\nabla_\parallel^{fc}$ as a column distributed matrix. The advantage is then that the gather operation is bijective, i.e. the transpose of the gather matrix is its inverse. This advantage is lost in the present problem. It turns out that it is advantageous to construct $\nabla_\parallel^{fc}$ as s row-distributed matrix with global indices. This is because a column distributed matrix can be easily (without mpi-communication) multiplied with a row distributed matrix especially if the indices are global indices. Each process just multiplies its local matrices.

$$M = C \cdot R \tag{32}$$

This is not true the other way round. The result is then a row distributed matrix with global indices. From the global indices the gather map/matrix and the local indices can be constructed. We note here that we even don't need to construct the gather matrix for $\nabla_\parallel^{fc}$, only the one for $\nabla_\parallel^{c}$ is needed.

4 Field aligned initialization

An important aspect of our simulations is a judicious initialization of the fields. We want structures to be field-aligned in the beginning of the simulation with a possible modulation along the direction of the field line. If a Gaussian shape is used, we call $\sigma_\parallel$ the extension in parallel direction and write

$$f_0(R,Z,\varphi) = F(R,Z,\varphi)\exp\left(-\frac{(\varphi-\varphi_0)^2}{2\sigma_\parallel^2}\right), \tag{33}$$

where F is a function that is invariant under the field line transformations

$$\mathcal{T}^+_{\Delta\varphi}F(\mathbf{z}) = F(\mathcal{T}^+_{\Delta\varphi}\mathbf{z}) \overset{!}{=} F(\mathbf{z}) \text{ (pull-back)}, \tag{34a}$$

$$\mathcal{T}^-_{\Delta\varphi}F(\mathbf{z}) = F(\mathcal{T}^-_{\Delta\varphi}\mathbf{z}) \overset{!}{=} F(\mathbf{z}) \text{ (push-forward)}. \tag{34b}$$

We can use these relations to construct aligned structures by active transformations of some given field. Our idea is to initialize a two-dimensional field $F(R,Z,\varphi_k)$ in a given plane k and transform this field to all other planes using the recursive relations

$$F(R,Z,\varphi_{k+1}) = \mathcal{T}^-_{\Delta\varphi}F(R,Z,\varphi_{k+1}) = F(R^-,Z^-,\varphi_k), \tag{35a}$$

$$F(R,Z,\varphi_{k-1}) = \mathcal{T}^+_{\Delta\varphi}F(R,Z,\varphi_{k-1}) = F(R^+,Z^+,\varphi_k), \tag{35b}$$

which is the statement that F in the next plane equals the push-forward and F in the previous plane equals the pull-back of F in the current plane. Note here that Eq. (9) applies for the required interpolation procedures.

References

[1] F. Hariri, P. Hill, M. Ottaviani, and Y. Sarazin. The flux-coordinate independent approach applied to x-point geometries. *Physics of Plasmas*, 21(8):082509, Aug. 2014. doi:10.1063/1.4892405.

5 Numerical test programs

The essential test programs for the parallel derivative are located in `path/to/feltor/inc/geometries`. `ds_t.cu` and `ds_mpit.cu` test the cylindrical grid for shared and distributed memory systems. `ds_curv_t.cu` and `ds_curv_mpit.cu` test the implementation on a flux-aligned grid. The magnetic field in FELTOR is given by

$$\mathbf{B} = \frac{R_0}{R}(I(\psi_p)\hat{e}_\varphi + \nabla\psi_p \times \hat{e}_\varphi) \tag{36}$$

This gives rise to magnetic field strength and components

$$B = \frac{R_0}{R}\sqrt{I^2 + (\nabla\psi_p)^2} \tag{37}$$

$$B^R = \frac{R_0}{R}\frac{\partial\psi_p}{\partial Z} \quad B^Z = -\frac{R_0}{R}\frac{\partial\psi_p}{\partial R} \quad B^\varphi = \frac{R_0 I}{R^2} \tag{38}$$

$$\nabla\cdot\hat{\boldsymbol{b}} = -\nabla_\parallel \ln B = -\frac{R_0}{RB^2}[B,\psi_p] \tag{39}$$

where $[.,.]$ is the Poisson bracket.

5.1 Cylindrical grid and circular flux surfaces

A very simple non-trivial choice for the poloidal flux is

$$\psi_p = \frac{1}{2}\left((R-R_0)^2 + Z^2\right) \equiv \frac{1}{2}r^2 \tag{40}$$

for which

$$\mathbf{b}^p = \frac{1}{\sqrt{I^2+r^2}}\begin{pmatrix} Z \\ -(R-R_0)\end{pmatrix} \tag{41}$$

$$b^\varphi = \frac{1}{\sqrt{I^2+r^2}} I/R \tag{42}$$

$$\nabla\cdot\hat{\boldsymbol{b}} = \frac{Z}{R(I^2+r^2}} \tag{43}$$

for the poloidal and toroidal parts of the magnetic unit vector. We choose $R_0 = 10$ and $I = 20$ in order to keep the q-factor for the $r = 1$ flux surface at 2. We set up a domain $R \in [R_0-1, R_0+1]$, $Z \in [-1,1]$ and $\varphi \in [0, 2\pi]$ and choose

$$f(R,Z,\varphi) = ((R-R_0)^2 + Z^2)\sin(\varphi) \tag{44}$$

$$\hat{\boldsymbol{b}}\cdot\nabla f = b^\varphi\cos(\varphi) \tag{45}$$

This is advantageous since the solution is correct even if we modify the $\hat{\boldsymbol{b}}$ field to be toroidal on the boundary of the domain in order to avoid boundary conditions. Also the function is parabolic, which means the dG polynomials of degree greater than 2 should be exact.

5.2 Curvilinear grid