The parallel derivative on structured grids

M. Wiesenberger and M. Held

February 16, 2021

Abstract

This write-up shows how we numerically treat parallel derivatives in a non field aligned coordinate system. It is mainly based on References [1, 3, 4] and contains parts from References [2, 5]

1 Discretization of parallel derivatives

We introduce the method and discuss some problems arising from the boundaries of the computational domain.

1.1 The flux coordinate independent approach

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

We begin with the formulation of a field-aligned discretization. To every smooth vector field $\boldsymbol{v}(\boldsymbol{x})$ there is a unique curve of which the tangent in a point p is the value of $\boldsymbol{v}(p)$ at that point. It is given by the solution of the differential equation

$$\frac{\mathrm{d}z^i}{\mathrm{d}s} = v^i(\boldsymbol{z})|_{\boldsymbol{z}(s)} \tag{1}$$

where z^i is one of (R, Z, φ) and v^i are the contravariant components of $\boldsymbol{v}$ in cylindrical coordinates. Note here that s does NOT necessarily denote the distance (especially since we do not require the existence of a metric at this point). Moreover, by definition we have

$$\frac{\mathrm{d}f(\boldsymbol{z}(s))}{\mathrm{d}s} = \boldsymbol{v} \cdot \nabla f|_{\boldsymbol{z}(s)} \tag{2}$$

along a field line parameterized by s. This means that instead of $\boldsymbol{v} \cdot \nabla f$ we can choose to discretize $\mathrm{d}f/\mathrm{d}s$.

Let us divide the φ direction into N_φ equidistant planes of $\Delta\varphi$. Unfortunately, from Eq. (1) we cannot easily determine Δs for given $\Delta\varphi$. Thus, it is better to reparameterize Eq. (2) with φ instead of s:

$$\frac{\mathrm{d}R}{\mathrm{d}\varphi} = \frac{v^R}{v^\varphi}, \tag{3a}$$

$$\frac{\mathrm{d}Z}{\mathrm{d}\varphi} = \frac{v^Z}{v^\varphi}, \tag{3b}$$

$$\frac{\mathrm{d}s}{\mathrm{d}\varphi} = \frac{1}{v^\varphi} \tag{3c}$$

We integrate Eqs. (3) from $\varphi = \varphi_k$ to $\varphi = \varphi_k \pm \Delta\varphi$ with initial condition

$$(R(\varphi_k), Z(\varphi_k), s(\varphi_k)) = (R, Z, s_k). \tag{4}$$

where k is the numbering of the planes and s_k is arbitrary. Let us characterize the solution $(R(\varphi_k \pm \Delta\varphi), Z(\varphi_k \pm \Delta\varphi))$ to Eqs. (3) as the flow generated by $\boldsymbol{v}/v^\varphi$

$$\mathcal{T}^\pm_{\Delta\varphi}\boldsymbol{z} \equiv \mathcal{T}^\pm_{\Delta\varphi}[R, Z, \varphi] := (R(\pm\Delta\varphi), Z(\pm\Delta\varphi), \varphi \pm \Delta\varphi), \tag{5}$$

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

We now fit a second order polynomial through (R, Z, φ), $\mathcal{T}^+_{\Delta\varphi}[R, Z, \varphi]$ and $\mathcal{T}^-_{\Delta\varphi}[R, Z, \varphi]$. We approximate the first and second derivative at (R, Z, φ) by evaluating the derivative of the polynomial at that point.

$$\begin{aligned}\nabla_\parallel f \equiv \frac{df}{ds} \to &\left(\frac{1}{s_{k+1} - s_{k-1}} - \frac{1}{s_k - s_{k-1}}\right) f_{k-1} \\ &+ \left(\frac{1}{s_k - s_{k-1}} - \frac{1}{s_{k+1} - s_k}\right) f_k \\ &+ \left(\frac{1}{s_{k+1} - s_k} - \frac{1}{s_{k+1} - s_{k-1}}\right) f_{k+1}\end{aligned} \tag{6}$$

where $\boldsymbol{z}_k = (R, Z, \varphi_k)$ and $f_k = f(\boldsymbol{z}_k)$ and $f_{k\pm1} := f(\mathcal{T}^\pm_{\Delta\varphi}\boldsymbol{z}_k)$. Note that Eq. (6) reduces to the familiar centered difference formula in case of equidistant spacings. Eq. (6) is slightly different from Reference [1], where the $\mathrm{d}f/\mathrm{d}\varphi v^\varphi$ is discretized to avoid integrating s. However, our discretization has the advantage that it can be used for higher order derivatives as well (e.g. $\mathrm{d}^2 f/\mathrm{d}\varphi^2$), which is not possible otherwise due to the chain rule for v^φ.

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 $\mathcal{T}^\pm_{\Delta\varphi}\boldsymbol{z}$ 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

$$f(\mathcal{T}^+_{\Delta\varphi}\boldsymbol{z}) = f(R^+_{nj}, \varphi_{k+1}) = \bar{f}^{ml}_{k+1} p_{ml}(R^+_{nj}) =: (I^+)^{ml}_{nj} f_{(k+1)ml}, \tag{7a}$$

$$f(\mathcal{T}^-_{\Delta\varphi}\boldsymbol{z}) = f(R^-_{nj}, \varphi_{k-1}) = \bar{f}^{ml}_{k-1} p_{ml}(R^-_{nj}) =: (I^-)^{ml}_{nj} f_{(k-1)ml}, \tag{7b}$$

where the backward transformations of $\bar{\boldsymbol{f}}$ 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 (6) can now be written as a matrix vector product

$$\nabla_\parallel f \to \left[S^+ \circ \mathbf{1}^+ \otimes I^+ + S^0 + S^- \circ \mathbf{1}^- \otimes I^-\right] \boldsymbol{f}, \tag{8}$$

where S^+, S^0 and S^- are the diagonal matrices that contain the prefactors in Eq. (6). 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

$$\nabla_\parallel^\dagger f = -\nabla\cdot(\boldsymbol{v}\; f) \neq -\nabla_\parallel f. \tag{9}$$

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

$$\Delta_\parallel := -\nabla_\parallel^\dagger \nabla_\parallel = (\nabla\cdot\hat{\boldsymbol{v}})\nabla_\parallel + \nabla_\parallel^2, \tag{10}$$

which is indeed the parallel part of the full Laplacian $\Delta = \nabla\cdot(\hat{\boldsymbol{v}}\nabla_\parallel + \nabla_\perp)$. $\hat{\boldsymbol{v}}$ is the unit vector $\boldsymbol{v}/|\boldsymbol{v}|$.

Note that the second order derivative $\nabla_\parallel^2$ can be discretized using

$$\frac{\mathrm{d}^2 f}{\mathrm{d}s^2} \to \frac{2f_{k+1}}{(s_{k+1}-s_k)(s_{k+1}-s_{k-1})} - \frac{2f_k}{(s_{k+1}-s_k)(s_k-s_{k-1})} + \frac{2f_{k-1}}{(s_k-s_{k-1})(s_{k+1}-s_{k-1})} \tag{11}$$

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 $\boldsymbol{v}(\boldsymbol{x})$ 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 $\boldsymbol{v}$ 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. (However this is true also for the following approach, See Section 1.3)

The second possibility (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

$$v^\zeta(\zeta,\eta) = \left(\frac{\partial\zeta}{\partial R}v^R + \frac{\partial\zeta}{\partial Z}v^Z\right)_{R(\zeta,\eta),Z(\zeta,\eta)} \tag{12}$$

$$v^\eta(\zeta,\eta) = \left(\frac{\partial\eta}{\partial R}v^R + \frac{\partial\eta}{\partial Z}v^Z\right)_{R(\zeta,\eta),Z(\zeta,\eta)} \tag{13}$$

$$v^\varphi(\zeta,\eta) = v^\varphi(R(\zeta,\eta), Z(\zeta,\eta)) \tag{14}$$

The fieldline equations (3) are still

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\varphi} = \frac{v^\zeta}{v^\varphi} \tag{15a}$$

$$\frac{\mathrm{d}\eta}{\mathrm{d}\varphi} = \frac{v^\eta}{v^\varphi} \tag{15b}$$

$$\frac{\mathrm{d}s}{\mathrm{d}\varphi} = \frac{1}{v^\varphi} \tag{15c}$$

The issue here is that when integrating fieldlines we have to interpolate the vector field $\boldsymbol{v}$ at arbitrary points instead of simply evaluating the exact values. However, the interpolation error vanishes with order P in the perpendicular plane so in order to mitigate this error we transform $\boldsymbol{v}$ 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 $\boldsymbol{v} \cdot \nabla$ is entirely unchanged and Eq. (6) still holds.

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).

1.3.1 Boundary conditions perpendicular to the wall

The issue with Neumann boundary conditions is that they usually prescribe derivatives perpendicular to the boundary while the fieldlines are in general not perpendicular to the boundary. One possible approach is to introduce ghostcells at the places where fieldlines end. The value of the ghostcells are as if we Fourier transformed the fields on the simulation domain with the correct boundary conditions and thus have a periodic extension of the fields beyond the boundaries. For example, for Neumann boundary conditions the field is effectively mirrored at the boundary, while for Dirichlet boundary conditions the values are the negative mirror image. In one dimension this reads

$$f(x) = \pm f(2x_b - x) \tag{16}$$

where x_b is the closest boundary $+$ is for Neumann, and $-$ for Dirichlet conditions. This is very easily implemented (at least as long as we only allow homogeneous Boundary conditions) since we don't actually have to perform the Fourier transformation, we only need to mirror the end coordinates at the boundary and then choose either the positive (Neumann) or negative (Dirichlet) interpolation.

The above procedure works for cylindrical coordinates where we can evaluate and thus integrate the vector field $\boldsymbol{v}$ even outside the domain. This is unfortunately not true for transformed coordinates. For now we have to rely on the fieldlines being aligned to the boundary in these cases to avoid boundary conditions altogether.

The downside of this approach is that the mirrored point of a field-line can lie very far away from the wall if the resolution in φ is low (which is the motivation to implement FCI in the first place) and in addition also on a different flux surface. In principle we need to increase the φ resolution until we resolve the perpendicular direction, in order to reliably converge with this method. In practice we can often run a simulation stably with low resolution nevertheless.

In order to avoid coupling different flux surfaces through the boundary condition, we have the possibility to mirror also the flux surfaces at the boundary. This approach converges well and works stably if the flux surface is (or is close to) perpendicular to the wall.

1.3.2 Boundary conditions parallel to the fieldline

The natural way to implement boundary conditions is to work them into the interpolating polynomial. The boundary condition then replaces a third point by setting either a predefined value (Dirichlet) or derivative (Neumann) on the boundary. Evaluating the interpolating polynomial at the middle point then yields (assuming that $\mathcal{T}_{\Delta\varphi}^{+}\boldsymbol{z}$ lies outside the boundary and $s_k < s_b^+ < s_{k+1}$ is where the boundary lies)

$$\begin{aligned}\frac{df}{ds} \underset{\text{NEU}}{\rightarrow} &\left(\frac{1}{s_k - s_{k-1}} - \frac{1}{2(s_b^+ - s_k) + (s_k - s_{k-1})}\right)(f_k - f_{k-1}) \\ &+ \left(\frac{s_k - s_{k-1}}{2(s_b^+ - s_k) + (s_k - s_{k-1})}\right) f'_{b+} \end{aligned} \tag{17}$$

$$\frac{\mathrm{d}^2 f}{\mathrm{d}s^2} \underset{\text{NEU}}{\rightarrow} \frac{2}{2(s_b^+ - s_k) + (s_k - s_{k-1})}\left(f'_{b+} - \frac{1}{s_k - s_{k-1}}(f_k - f_{k-1})\right) \tag{18}$$

where f'_{b+} is the value of the derivative on the boundary (currently we only allow homogeneous boundary conditions, i.e. $f'_{b+} \equiv 0$). If the boundary lies at $s_{k-1} < s_b^- < s_k$ then we have

$$\begin{aligned}\frac{df}{ds} \underset{\text{NEU}}{\rightarrow} &\left(\frac{1}{s_{k+1} - s_k} - \frac{1}{2(s_k - s_b^-) + (s_{k+1} - s_k)}\right)(f_{k+1} - f_k) \\ &+ \left(\frac{s_{k+1} - s_k}{2(s_k - s_b^-) + (s_{k+1} - s_k)}\right) f'_{b-} \end{aligned} \tag{19}$$

$$\frac{\mathrm{d}^2 f}{\mathrm{d}s^2} \underset{\text{NEU}}{\rightarrow} \frac{2}{2(s_k - s_b^-) + (s_{k+1} - s_k)}\left(-f'_{b-} + \frac{1}{s_{k+1} - s_k}(f_{k+1} - f_k)\right) \tag{20}$$

while if the fieldline intersects the wall on both ends we have

$$\frac{df}{ds} \underset{\text{NEU}}{\rightarrow} \frac{1}{s_b^+ - s_b^-}\left[(s_k - s_b^-)f'_{b+} + (s_b^+ - s_k)f'_{b-}\right] \tag{21}$$

$$\frac{\mathrm{d}^2 f}{\mathrm{d}s^2} \underset{\text{NEU}}{\rightarrow} \frac{1}{s_b^+ - s_b^-}(f'_{b+} - f'_{b-}) \tag{22}$$

The formulas for Dirichlet boundary conditions are the same as the original formulas, with s_{k+1}, f_{k+1} replaced by s_b^+, f_b^+ and/or s_{k-1}, f_{k-1} replaced by s_b^-, f_b^-.

There is no difference between those formulas and actually evaluating the interpolating polynomial at a ghost point and then using that point in the original formulas. The reason is that the interpolating polynomial is unique and thus is the value of its derivatives at s_k.

For the above formulas to work we need to find the exact place where the fieldline intersects the boundary. We have to find φ_b such that the result of the integration of Eq. (3) from φ to φ_b lies on the boundary. The angle φ_b can be found by a bisection algorithm knowing that $\varphi_k < \varphi_b < \varphi_k + \Delta\varphi$. This kind of procedure is known as a shooting method. Another possibility is to trick the fieldline integrator into finding the point for us. We do this by setting $\boldsymbol{v} \equiv 0$ on all points outside the simulation box, which makes the ODE integrator stop once it crosses the domain boundary. This works fairly well with the adaptive embedded Runge Kutta method that we use and in particular also works in transformed coordinates.

The advantage of this method is that the parallel derivative does not couple different field lines through the boundary conditions. Thus the dynamics on each field-line can be completely independent.

1.3.3 Avoiding boundary conditions in non-aligned systems

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 might work in practice it is **highly unclear** what numerical and physical side-effects this procedure might have.

Another solution would be to change the vector field $\boldsymbol{v}$ and only retain the toroidal part of $\boldsymbol{v}$ 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). **Unfortunately, when testing this procedure with an analytical solution the error does not converge neither for Dirichlet nor for Neumann.**

1.4 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

2.1 A grid refinement approach

The idea is to discretize the operation $\nabla \cdot (\boldsymbol{v}.)$ by taking the adjoint of the discretization for $\nabla_\parallel$ i.e. Eq. (6). Remember that the adjoint of a matrix involves the volume element (including dG weights). This means that after you've transposed the parallel derivative Eq. (6), simply bracket the result by $1/\sqrt{g}$ and $\sqrt{g}$ to get the adjoint.

While the idea of simply transposing the discretization matrices sounds appealing the problem is that the resulting discretization does not converge. One idea to solve this problem [4] is to bracket the parallel derivative by interpolation (Q) and projection (P) matrices:

$$\nabla_\parallel^c = P\nabla_\parallel^f Q \tag{23}$$

$$\nabla_\parallel^{c\dagger} = P\nabla_\parallel^{f\dagger} Q \tag{24}$$

where f and c denote fine and coarse grid respectively. In this way the projection integrals

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

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

$$\nabla_\parallel^c = P\nabla_\parallel^{fc} \tag{25}$$

$$\nabla_\parallel^{c\dagger} = \nabla_\parallel^{fc\dagger} Q \tag{26}$$

Note that the matrix-matrix multiplications in Eq. (26) 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 $v^\varphi/\Delta\varphi$ matrix does not commute with Q or P).

2.2 Some Thoughts

In order to understand what the adjoint operators do let us denote $\mathcal{T}^+_{\Delta\varphi}$ as the push-forwward operator. Then we have

$$\int f(\boldsymbol{x})\mathcal{T}^+_{\Delta\varphi}h(\boldsymbol{x})\sqrt{g(\boldsymbol{x})}\mathrm{d}^3x = \int \frac{1}{\sqrt{g(\boldsymbol{x}')}}\mathcal{T}^-_{\Delta\varphi}\left[J^{-1}(\boldsymbol{x}')\sqrt{g(\boldsymbol{x}')}f(\boldsymbol{x}')\right]h(\boldsymbol{x}')\sqrt{g(\boldsymbol{x}')}\mathrm{d}^3x'$$
$$\equiv \int (\mathcal{T}^+_{\Delta\varphi})^\dagger\left[f(\boldsymbol{x})\right]h(\boldsymbol{x})\sqrt{g(\boldsymbol{x})}\mathrm{d}^3x \quad (27)$$

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

$$(\mathcal{T}^+_{\Delta\varphi})^\dagger f(\boldsymbol{x}) := \frac{1}{\sqrt{g(\boldsymbol{x})}}\mathcal{T}^-_{\Delta\varphi}\left[\sqrt{g(\boldsymbol{x})}J^{-1}(\boldsymbol{x})f(\boldsymbol{x})\right] \quad (28)$$

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

$$(\mathcal{T}^+_{\Delta\varphi})^\dagger f(\boldsymbol{x}) := \sqrt{\frac{g'(\boldsymbol{x})}{g(\boldsymbol{x})}}\mathcal{T}^-_{\Delta\varphi}\left[f(\boldsymbol{x})\right] \quad (29)$$

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 we integrate streamlines of the any vector field $\boldsymbol{v}$, then we have

$$\frac{\mathrm{d}J}{\mathrm{d}\varphi} = J(\boldsymbol{x})\nabla\cdot\boldsymbol{v} \quad (30)$$

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

$$(\mathcal{T}^+_{\Delta\varphi})^\dagger\left[J(\boldsymbol{x})\mathcal{T}^+_{\Delta\varphi}f(\boldsymbol{x})\right] - f(\boldsymbol{x}) = 0 \quad (31)$$

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

$$\left(\mathcal{T}^+_{\Delta\varphi}\right)^\dagger J(\boldsymbol{x}) = 1 \quad (32)$$

Finally, let us assume that $\boldsymbol{v} = \hat{\boldsymbol{b}}$ is the magnetic field unit vector. Then we have analytically $\nabla\cdot\boldsymbol{B} = \nabla^\dagger_\parallel B = 0$. Numerically, this is true if we have

$$\left(\mathcal{T}^+_{\Delta\varphi}\right)^\dagger B^\varphi = \left(\mathcal{T}^-_{\Delta\varphi}\right)^\dagger B^\varphi = B^\varphi \quad (33)$$

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 topologcially (metric and Jacobian of the fine grid 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, currently 7) 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

$$\frac{\mathrm{d}\zeta}{\mathrm{d}\varphi} = \frac{v^\zeta}{v^\varphi} \tag{34a}$$

$$\frac{\mathrm{d}\eta}{\mathrm{d}\varphi} = \frac{v^\eta}{v^\varphi} \tag{34b}$$

$$\frac{\mathrm{d}s}{\mathrm{d}\varphi} = \frac{1}{v^\varphi} \tag{34c}$$

    with the given starting points and $s = 0$ from $\varphi = 0$ until $\varphi = \pm\Delta\varphi$. (Currently we use a Prince-Dormand method with stepsize control for this step).
  - store the results in s_{k+1} and s_{k-1}.
  - 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 of the fine grid
- 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 their transposes

Notes on the MPI implmentation 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.

$$M = C \cdot R \tag{35}$$

This is not true if we started with a column distributed matrix. 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

$$f_0(R,Z,\varphi) = F(R,Z,\varphi)\exp\left(-\frac{(\varphi-\varphi_0)^2}{2\sigma_\parallel^2}\right), \tag{36}$$

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

$$\mathcal{T}^+_{\Delta\varphi}F(\boldsymbol{z}) = F(\mathcal{T}^+_{\Delta\varphi}\boldsymbol{z}) \overset{!}{=} F(\boldsymbol{z}) \text{ (pull-back)}, \tag{37a}$$
$$\mathcal{T}^-_{\Delta\varphi}F(\boldsymbol{z}) = F(\mathcal{T}^-_{\Delta\varphi}\boldsymbol{z}) \overset{!}{=} F(\boldsymbol{z}) \text{ (push-forward)}. \tag{37b}$$

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

$$F(R,Z,\varphi_{k+1}) = \mathcal{T}^-_{\Delta\varphi}F(R,Z,\varphi_{k+1}) = F(R^-,Z^-,\varphi_k), \tag{38a}$$
$$F(R,Z,\varphi_{k-1}) = \mathcal{T}^+_{\Delta\varphi}F(R,Z,\varphi_{k-1}) = F(R^+,Z^+,\varphi_k), \tag{38b}$$

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. (7) applies for the required interpolation procedures.

5 Numerical tests

The test programs for the parallel derivative are located in `path/to/feltor/inc/geometries`. To every shared memory test `*_t.cu` there is a corresponding distributed memory `*_mpit.cu` program. `ds_t.cu` tests the cylindrical grid with boundary conditions. `ds_guenther_t.cu` tests the cylindrical grid without boundary conditions i.e. no fieldline leaves the domain. `ds_curv_t.cu` tests the implementation on a flux-aligned grid again with no fieldline leaving the domain. Finally, `ds_straight_t.cu` tests the implementation of completely straight fieldlines and the boundary conditions in the parallel direction.

The magnetic field in FELTOR is given by

$$\boldsymbol{B} = \frac{R_0}{R}\left(I(\psi_p)\hat{e}_\varphi + \nabla\psi_p\times\hat{e}_\varphi\right) \tag{39}$$

This gives rise to magnetic field strength and components

$$B = \frac{R_0}{R}\sqrt{I^2 + (\nabla\psi_p)^2} \tag{40}$$
$$B^R = \frac{R_0}{R}\frac{\partial\psi_p}{\partial Z} \quad B^Z = -\frac{R_0}{R}\frac{\partial\psi_p}{\partial R} \quad B^\varphi = \frac{R_0 I}{R^2} \tag{41}$$
$$\nabla\cdot\hat{\boldsymbol{b}} = -\nabla_\parallel \ln B = -\frac{R_0}{RB^2}[B,\psi_p] \tag{42}$$

where $[.,.]$ is the Poisson bracket. Note that in order to compute analytical testfunctions we use

$$\nabla_\parallel f = b^R\partial_R f + b^Z\partial_Z f + b^\varphi\partial_\varphi f \tag{43}$$
$$\Delta_\parallel f = \nabla\cdot(\hat{\boldsymbol{b}}\hat{\boldsymbol{b}}\cdot\nabla f) = (\nabla\cdot\hat{\boldsymbol{b}})\nabla_\parallel f + (\nabla_\parallel b^j)\partial_j f + b^i b^j\partial_i\partial_j f \tag{44}$$

with

$$\nabla_\parallel b^R = (\nabla\cdot\hat{\boldsymbol{b}})b^R + \frac{\psi_Z(\psi_{RZ}-\psi_Z/R)-\psi_{ZZ}\psi_R}{I^2+(\nabla\psi)^2} \tag{45}$$

$$\nabla_\parallel b^Z = (\nabla\cdot\hat{\boldsymbol{b}})b^Z + \frac{\psi_R(\psi_{RZ}+\psi_Z/R)-\psi_{RR}\psi_Z}{I^2+(\nabla\psi)^2} \tag{46}$$

$$\nabla_\parallel b^\varphi = (\nabla\cdot\hat{\boldsymbol{b}})b^\varphi + \frac{\psi_Z(I_R/R-2I/R^2)-I_Z\psi_R/R}{I^2+(\nabla\psi)^2} \tag{47}$$

where we used $\psi_R \equiv \partial\psi_p/\partial R$ and $I_R \equiv \partial I(\psi_p)/\partial R$.

5.1 Cylindrical grid and boundary conditions

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

$$\psi_p = \frac{1}{2}\left((R-R_0)^2+Z^2\right) \equiv \frac{1}{2}r^2 \tag{48}$$

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

$$f(R,Z,\varphi) = (\cos(\pi(R-R_0))+1)(\cos(\pi Z/2)+1)\sin(\varphi) \tag{49}$$

On the chosen domain f respects both Neumann and Dirichlet boundary conditions both along the field and perpendicular to the wall. In Tables 1 and 2 we show the convergence of the solution for various operators and Dirichlet boundary conditions. Note that the same tables for Neumann boundary conditions exhibit similar values. Apparently, the discretization for the divergence does not converge even if we increase the refinement.

			$\nabla_\parallel f$ Eq. (6)		$\nabla_\parallel^2 f$ Eq. (11)		$\nabla\cdot(\hat{\boldsymbol{b}}f)$		$\Delta_\parallel^{-1}f$	
			error	order	error	order	error	order	error	order
m	**N**	**Nz**								
1	**10**	**5**	2.61e-01	n/a	2.40e-01	n/a	4.18e-01	n/a	5.08e-03	n/a
	16	**10**	7.84e-02	1.74	1.04e-01	1.21	6.88e-01	-0.72	3.24e-03	0.65
	26	**20**	2.16e-02	1.86	4.01e-02	1.37	1.40e+00	-1.02	4.04e-03	-0.32
	40	**40**	5.65e-03	1.93	1.48e-02	1.44	3.23e+00	-1.21	3.78e-03	0.10
	64	**80**	1.44e-03	1.97	5.36e-03	1.47	6.35e+00	-0.98	3.04e-03	0.31
	100	**160**	3.67e-04	1.97	1.96e-03	1.45	1.22e+01	-0.94	2.37e-03	0.36

Table 1: Convergence Table for Dirichlet boundary conditions and $m = 1$, $N_R = N_Z = N$. Centered discretization with boundary conditions along the fieldline. The table for Neumann conditions exhibits similar numbers.

5.2 The Günther field

A simple choice for the flux function that makes fieldlines stay inside a square box is

$$\psi_p = \cos(\pi(R-R_0)/2)\cos(\pi Z/2) \tag{50}$$

			$\nabla_\parallel f$ Eq. (6)		$\nabla^2_\parallel f$ Eq. (11)		$\nabla\cdot(\hat{\boldsymbol{b}} f)$		$\Delta^{-1}_\parallel f$	
			error	order	error	order	error	order	error	order
m	**N**	**Nz**								
10	**10**	**5**	2.61e-01	n/a	2.40e-01	n/a	2.61e-01	n/a	5.07e-03	n/a
	16	**10**	7.84e-02	1.74	1.06e-01	1.18	8.00e-02	1.71	2.44e-03	1.06
	26	**20**	2.16e-02	1.86	4.01e-02	1.40	3.74e-02	1.10	1.02e-03	1.26
	40	**40**	5.66e-03	1.93	1.48e-02	1.44	7.81e-02	-1.06	4.16e-04	1.29
	64	**80**	1.45e-03	1.96	5.38e-03	1.46	3.46e-01	-2.15	2.15e-04	0.95
	100	**160**	3.67e-04	1.98	2.09e-03	1.36	9.81e-01	-1.50	1.89e-04	0.19

Table 2: Convergence Table for Dirichlet boundary conditions and $m = 10$, $N_R = N_Z = N$. Centered discretization with boundary conditions along the fieldline. The table for Neumann conditions exhibits similar numbers.

Again, 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

$$f_1(R,Z,\varphi) = -\psi_p(R,Z)\cos(\varphi) \tag{51}$$

$$f_2(R,Z,\varphi) = -\psi_p(R,Z)\cos(\varphi) + (R-R_0)^2/4 + Z(R-R_0)/4 \tag{52}$$

			$\nabla_\parallel f$ Eq. (6)		$\nabla^2_\parallel f$ Eq. (11)		$\nabla\cdot(\hat{\boldsymbol{b}} f)$		$\Delta^{-1}_\parallel f$	
			error	order	error	order	error	order	error	order
m	**N**	**Nz**								
10	**6**	**5**	2.77e-01	n/a	1.93e-01	n/a	2.79e-01	n/a	5.28e-03	n/a
	12	**10**	7.80e-02	1.83	5.81e-02	1.73	8.24e-02	1.76	1.95e-03	1.43
	18	**20**	2.02e-02	1.95	1.55e-02	1.91	5.20e-02	0.66	5.76e-04	1.76
	24	**40**	5.10e-03	1.99	3.94e-03	1.97	1.10e-01	-1.08	1.56e-04	1.89
	36	**80**	1.28e-03	2.00	1.03e-03	1.93	3.07e-01	-1.49	8.87e-05	0.81
	60	**160**	3.20e-04	2.00	3.56e-04	1.53	7.45e-01	-1.28	1.57e-04	-0.83

Table 3: Convergence Table for $m = 10$ and $N_R = N_Z = N$ and the Günther field. Centered discretizations, no boundary condition.

5.3 Curvilinear grid

Here we choose the general Solov'ev equilibrium for ψ_p and choose to use f_1 or f_2 again. We use the simple orthogonal flux to construct grids on the ring bounded by $\psi_p = -20$ and $\psi_p = -4$.

			$\nabla_\parallel f$ Eq. (6)		$\nabla_\parallel^2 f$ Eq. (11)		$\nabla\cdot(\hat{\boldsymbol{b}}f)$		$\Delta_\parallel^{-1} f$	
			error	order	error	order	error	order	error	order
m	**N**	**Nz**								
1000	**30**	**5**	4.88e-01	n/a	3.83e-01	n/a	5.41e-01	n/a	2.11e-06	n/a
	60	**10**	1.64e-01	1.57	1.26e-01	1.60	1.84e-01	1.56	1.09e-06	0.95
	90	**20**	4.48e-02	1.87	3.42e-02	1.88	5.01e-02	1.88	3.62e-07	1.59
	120	**40**	1.15e-02	1.97	8.75e-03	1.97	1.31e-02	1.93	9.85e-08	1.88
	180	**80**	2.88e-03	1.99	2.26e-03	1.95	7.11e-03	0.89	2.67e-08	1.88
	300	**160**	7.22e-04	2.00	6.60e-04	1.78	1.27e-02	-0.84	1.90e-08	0.49

Table 4: Convergence Table for $m = 1000$, $N_R = 2$, $N_Z = N$ on a curvilinear grid. Centered discretizations, no boundary condition.

6 Performance considerations

The performance of the main matrix-vector multiplication depends, as do all our routines, on the number of memory loads and stores per vector element. A dG interpolation matrix in two dimensions has n^2 points per line since it uses all n^2 points in one dG-cell to compute the result. A matrix-vector multiplication with the interpolation matrix thus needs $n^2 + 3$ vector loads and stores, with the 3 coming from loading the input vector and loading and storing the output vector. However, since we use the refinement-projection aproach to our discretization the number of points per line of the interpolation matrix increases to typically $4n^2$ since now also neighboring cells are used for the computation.

For the discretization of $\nabla_\parallel$ this in total means that the number of memory operations per element is

$$m = 2\cdot(4n^2+3)+5 \quad \text{(refined)} \tag{53}$$

$$m = 2\cdot(n^2+3)+5 \quad \text{(unrefined)} \tag{54}$$

with n the number of polynomial coefficients. This is comparable to one iteration of a CG method in three dimensions.

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, 2014. doi:10.1063/1.4892405.

[2] M. Held. *Full-F gyro-fluid modelling of the tokamak edge and scrape-off layer*. PhD thesis, University of Innsbruck, 2016. URL http://resolver.obvsg.at/urn:nbn:at:at-ubi:1-6853.

[3] M. Held, M. Wiesenberger, and A. Stegmeir. Three discontinuous galerkin schemes for the anisotropic heat conduction equation on non-aligned grids. *Computer Physics Communications*, 199:29–39, 2016. doi:10.1016/j.cpc.2015.10.009.

[4] A. Stegmeir, O. Maj, D. Coster, K. Lackner, M. Held, and M. Wiesenberger. Advances in the flux-coordinate independent approach. *Comput. Phys. Commun.*, 213:111 – 121, 2017. doi:10.1016/j.cpc.2016.12.014.

[5] M. Wiesenberger. *Gyrofluid computations of filament dynamics in tokamak scrape-off layers*. PhD thesis, University of Innsbruck, 2014. URL http://resolver.obvsg.at/urn:nbn:at:at-ubi:1-1799.