(P)review of the FEEC Coldplasma Project

Frederik Schnack (frederik.schnack@tum.de)

17.04.2019

Problem Setting

Our aim is the implementation of a finite element method (FEM) for the following linearized plasma fluid model for electrons in a static ion background,

$$\begin{cases} -\dfrac{\partial \mathbf{E}}{\partial t} + c^2 \nabla \times \mathbf{B} = \dfrac{1}{\epsilon_0}\mathbf{J} & \text{(Ampère's law)}\,, \\ \dfrac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} = 0 & \text{(Faraday's law)}\,, \\ \dfrac{\partial \mathbf{J}}{\partial t} = \omega_{\mathrm{p}}^2\, \mathbf{E} - \mathbf{J} \times \omega_{\mathrm{c}} - \nu_{\mathrm{e}} \mathbf{J} & \text{(electron momentum balance)}\,, \\ \nabla \cdot \mathbf{B} = 0 & \text{(divergence constraint)}\,, \end{cases} \tag{1}$$

whereas, for $\nu_e = 0$, this system has a Hamiltonian structure:

$$\mathcal{H} = \int_{\Omega_3} \left(\frac{\varepsilon_0 |\mathbf{E}|^2}{2} + \frac{|\mathbf{B}|^2}{2\mu_0} + \frac{|\mathbf{J}|^2}{2\omega_{\mathrm{p}}^2} \right) \mathrm{d}^3\mathbf{x}\,. \tag{2}$$

In order to get a geometry conserving code, we build our discretization upon the idea of the finite element exterior calculus (FEEC). In other words, the finite dimensional subspaces and discrete differential operators have to fulfill the commuting diagram as seen in figure 1.

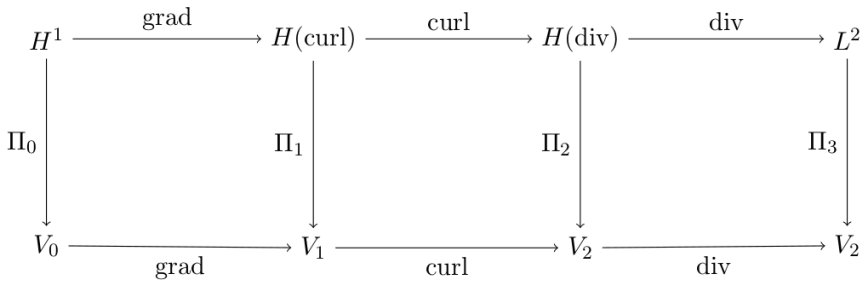

Figure 1: 3D De-Rham differential complex.

Theoretical Aspects

At first, we write the system (1) in the **weak formulation**. Where we can read of the according vectorspace for each unknown, which on the other hand tells us in what finite dimensional subspace the corresponding discrete version lives.
Next up, we introduce the **basis functions** for the finite dimensional subspaces. We decided, that we start in V_0 with a basis function consisting of B-Splines $(N_i^k)_{0\leq i\leq n-1}$ of degree k in $x-$ and $y-$direction, and a single Fourier mode $E(z) = e^{\lambda z}$ in the $z-$direction. From that starting point, we can calculate all the other basis functions and the relation of the coefficients among themselves. These relations define the **discrete differential operators**, which we need for the commuting diagram.
Inserting the discrete functions into the weak formulation leads to the definition of the **mass matrices**. Furthermore, we are now able to write the system as a system of ODEs. Considering the Hamiltonian (2), we can transfer it to our semi-discretization. This allows us to embed the ODE-system into a Hamiltonian system defined by a **Poisson bracket**.
In order to solve this system and preserve different desired properties, we develop a 2-step **Poisson splitting**, via the average vector field, and a $4-$step **Hamiltonian splitting**. Both give similar satisfying results in practice so far.

Practical Aspects

The first task is the **basis creation**, where we used the scipy.interpolate.BSpline routine.
The next part, where we **assemble** the **mass matrices**, is more complicated. Here, we go really into detail in order to build up the structure from the one dimensional case: We want to write our code with help of the **Kronecker-product** structure, which allows very efficient computation. Hence, we have to express our mass matrices and unknowns in this framework. After some calculation, we are able to write these matrices as block-matrices of tensor-products of mass matrices, which are build by the basis functions of a single direction. But here, we have to make our first assumption: Since in general, we have a additional function in two arising stiffness matrices, c.f. ω_c, ω_p, we have to be careful as this may deny the option to write these matrices in the Kronecker-framework. Hence, we assumed that such functions are always products of functions depending on only one variable x, y or z. Another assumption we made, was ω_c being of the form: $\omega_c = (0, 0, \omega_{c,x}(x)\omega_{c,y}(y)\omega_{c,z}(z))^T$.
Obviously, we assemble all matrices in a sparse format, otherwise it wouldn't be even possible to run the code. We also utilize other optimization to make the code more efficient. Beginning with the „small“ mass matrices, which have no additional function in the integral, we can assemble them by calculating one row of the matrix and then augment the rest with help of the **circulant structure**. This structure also allows us to easily invert the matrix, which is very helpful as we need to invert one of the „big“ mass matrices. Since we can write the inverse of this matrix again as blocks of tensor-products of the smaller inverted matrices, we get a very cheap and stable way to calculate this **inverse**. Furthermore, we pre-calculate all matrix products occurring in the time-stepping.

For the **Poisson Splitting**, we have to solve a linear system in every step, where the matrices are constant if we choose a constant step-size. Hence, we calculate the LU-decomposition of both matrices once and solve two triangular systems in every step. The **Hamiltonian Splitting** involves a matrix exponential and another matrix inversion, which can both be pre-calculated and also yield a cheap time stepping. Here, one has to be careful as the matrix that needs to be inverted is ill-conditioned, this leads to numerical problems. We use a **Strang splitting** for these steps, which is applicable as both methods are self-adjoint.
At this point, we are able to propagate the coefficients of our discrete functions through time. In order to get the function values, we introduce the standard **collocation matrix**.

Numerical Results

In order to test the stiffness matrices and the collocation matrix, we implemented a simple **L2-projection** problem. As this test was successful, we went a step further and tested the **discrete curl matrix** with it, which also was successful. This led us to considering the whole system, where first tests were done with the **dispersion relation**.
Here, we ran into first problems. The biggest flaw right now is the instability for $\lambda \neq 0$, where the coefficients grow exponentially. For $\lambda = 0$, both methods show (the same) reasonable results, although for some initial coefficients the coefficients corresponding to the B-field have a linear growth (in x and y component) which looks quite weird at the moment. Note that $\lambda = 0$ corresponds to the 2D embedding of our implementation, but this means that in V_2, where B lives, the first two components have no influence in the system. Still, in this case, when we check the frequencies of the different oscillating function values, we always get a clear peak in frequencies that belong to the dispersion relation. This marks a small success on the way to reach our goal.

Outlook

Different things have yet to be considered, need a closer look or have to be fixed. This is for example:

- Derivation and implementation of the projectors. (Were not required yet.)
- Checking why the B-field is growing linearly for some initial conditions. (Is this a problem? coeff. not considered anyways.)
- Finding the reason for the instability with $\lambda \neq 0$.
- How to deal with the assumption for general functions in the stiffness matrix? Can the tensor product framework still hold? What about ω_c?