Django-Jorek code: a numerical box for MHD discretization and JOREK

A. Ratnani[2],B. Nkonga[35], E. Franck[1],C. Caldini-Queiros[2], L. Mendoza[2], G. Latu[4], V. Grandgirard[4], E. Sonnendrücker[26], M. Hölzl[2], H. Guillard [5]

, ,

16 September 2015

[1]INRIA Nancy Grand-Est and IRMA Strasbourg, TONUS team, France
[2]Max-Planck-Institut für Plasmaphysik, Garching, Germany
[3]Nice university, Nice, France
[4]CEA/DSM/IRFM, Cadarache, France
[5]INRIA Sophia-Antipolis, CASTOR team, France
[6]TUM center of mathematics, Garching, Germany

Outline

Context and the Django code

Physical context : MHD and ELM's

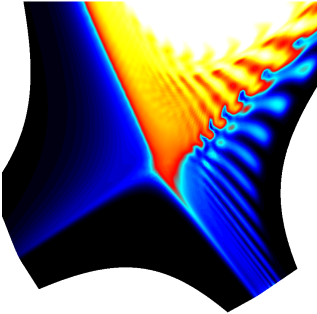
- In the tokamak plasma instabilities can appear.
- The simulation of these instabilities is an important subject for ITER.

- Examples of Instabilities in the tokamak :
  - **Disruptions**: Violent instabilities which can seriously damage the tokamak.
  - **Edge Localized Modes (ELM's)**: Periodic edge instabilities which can damage wall components due to their extremely high energy transfer rate.

- These instabilities are described by MHD models like

$$
\begin{cases}
\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0 \\
\rho \partial_t \mathbf{u} + \rho \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p = \mathbf{J} \times \mathbf{B} - \nabla \cdot \overline{\overline{\mathbf{\Pi}}} \\
\frac{1}{\gamma - 1} \partial_t p + \frac{1}{\gamma - 1} \mathbf{u} \cdot \nabla p + \frac{\gamma}{\gamma - 1} p \nabla \cdot \mathbf{u} + \nabla \cdot \mathbf{q} \\
= \frac{1}{\gamma - 1} \frac{m_i}{e\rho} \mathbf{J} \cdot \left(\nabla p_e - \gamma p_e \frac{\nabla \rho}{\rho} \right) - \overline{\overline{\mathbf{\Pi}}} : \nabla \mathbf{u} + \eta |\mathbf{J}|^2 \\
\partial_t \mathbf{B} = -\nabla \times \left(-\mathbf{u} \times \mathbf{B} + \eta \mathbf{J} - \frac{m_i}{\rho e} \nabla p_e + \frac{m_i}{\rho e} (\mathbf{J} \times \mathbf{B}) \right) \\
\mu_0 \nabla \times \mathbf{B} = \mathbf{J}, \quad \nabla \cdot \mathbf{B} = 0
\end{cases}
$$

- ELM's Simulation



Aim and principle of DJANGO project

Aim of DJANGO project

- Develop a library to test and validate the numerical methods which we use in the MHD codes with
  - more simple models,
  - more simple geometries and meshes,
  - more simple cases.
- Validate the future numerical heart for JOREK 3.0

Numerical heart of DJANGO

- **Full and reduced MHD** with bi-fluids and diamagnetic terms.
- **Arbitrary high-order and stable Splines** on quadrangular and triangular meshes using Bernstein formalism with refinement.
- **New toroidal basis** or flexible toroidal discretization.
- **Adaptive preconditioning** and Jacobian-free method.
- **Possible coupling with kinetic codes** like Selalib.

Models in DJANGO

Models in Django

A model in Django is defined by

- The specific parameters of the model or the scheme.
- The weak forms (which depend also of the time scheme).
- The diagnostics computation (norm, energy, mass).
- The algorithm to solve the problem which can solve successively some operators (initialization, time loop or solving, diagnostics, etc).

Current models implemented in Django

- **Elliptic models**: 2D-3D Laplacian, 2D Grad-Div operator, 2D Bi-Laplacian and 2D Grad-Shafranov
- **Diffusion models**: 2D diffusion equation, 3D anisotropic diffusion equation.
- **Mixed hyperbolic-parabolic models**: 2D Cartesian and cylindrical Current Hole, 2D damped wave equations.

Future models to be implemented in Django

- **Elliptic models**: 2D or 3D stokes and stokes-MHD models.
- **Hyperbolic models**: 3D Maxwell equations, 2D Euler equations.
- **Mixed hyperbolic-parabolic models**: 3D full and reduced MHD (199 and 303 version).

Spatial discretization and meshes in DJANGO

Meshes and discretization in DJANGO

3D geometry

- Currently the code allows to use 3D geometries: cylinder or torus.
- **Current strategy**: Tensor product between 2D poloidal meshes and 1D toroidal uniform meshes.
- **Future strategies**: Tensor product or real 3D triangular or quadrangular meshes.

2D poloidal Meshes

- Triangular or quadrangular meshes generated by CAID (code of A. Ratnani) based on isoparametric and isogeometric approach.

Remarks

- Currently the poloidal and toroidal discretization are separated.
- **future works and researches**: would be realized on the non-singular meshes in the isoparametric or isogeometric context.

Current discretization in DJANGO

Philosophy of the discretization

- **Isoparametric and isogeometric** approachs: the physical function are represented with the same basis functions of used to represent the geometry.
- These approaches allow to align and adapt the meshes to the physical flows (surfaces aligned mesh).

Current discretization in Django

- **Hermite Bezier** finite element basis (used in JOREK) with different quadrature rules implemented.
- **B-Splines** finite element basis with arbitrary order (1 to 5 actually) and regularity (C^0 to C^{p-1}).
- **Remark**: the B-Splines on triangles and quadrangles are unified using Bernstein formalism (A. Ratnani)
- **Box-Splines** finite element: Splines with quasi-interpolation on triangle used also in Selalib for transport problems on Hexagonal meshes (L. Mendoza).

Results for B-Splines and Bezier elements in DJANGO

- Convergence of poloidal discretizations.

	Cells	2D laplacian		2D Bi-Laplacian		2D-Wave	
		Err	Order	Err	Order	Err	Order
HBezier	16*16	2.9E-5	-	3.4E-5	-	2.8E-5	-
	32*32	1.9E-6	3.9	2.1E-6	4.0	1.8E-6	3.95
	64*64	1.2E-7	4.0	1.6E-7	3.8	1.2E-7	3.9
B-S2 c^0	16*16	-	-	-	-	-	-
	32*32	-	-	-	-	-	-
	64*64	-	-	-	-	-	-
B-S2 c^1	16*16	-	-	-	-	-	-
	32*32	-	-	-	-	-	-
	64*64	-	-	-	-	-	-
B-S5 c^0	16*16	-	-	-	-	-	-
	32*32	-	-	-	-	-	-
	64*64	-	-	-	-	-	-
B-S5 c^4	16*16	-	-	-	-	-	-
	32*32	-	-	-	-	-	-
	64*64	-	-	-	-	-	-

- Efficiency and conditioning of poloidal basis (Mesh 64*64).

	Nb dof		time solving	
	C^0	C^{p-1}	C^0	C^{p-1}
BS p=3	36481	4225	-	-
BS p=4	65205	4356	-	-
Hezier	16384	-	4.4E-3	-

Future poloidal discretization in DJANGO

Aim:

- **Arbitrary high-order and stable Splines** on quadrangular and triangular mesh using the Bernstein formalism with refinement.

Future works with new PhD student:

- **B-Splines** on quadrangular and triangular mesh using Bernstein formalism (A. Ratnani)
- **Refinement** of the mesh, order and regularity for B-Splines (A. Ratnani, E. Franck + PhD) also to construct low-order and adaptive preconditioning (next section).
- **Compatible finite element** using the DeRham sequence to obtain a stable discretization for non-coercive problems or for problems with involutive constraints (A. Ratnani, E. Franck, E. Sonnendrücker + PhD).
- **Theoretical study** of the stability and convergence of these elements (A. Ratnani, E. Franck, E. Sonnendrücker + PhD).

Other works:

- **Stabilization** for convective problems using Petrov-Galerkin methods (B. Nkonga).

Future and current toroidal discretization in DJANGO

Aim:

- **New toroidal basis** or flexible toroidal discretization.

Current toroidal discretization in DJANGO:

- 1D B-splines (the 3D basis is obtained by tensor product).
- Classical Fourier expansion.

Future toroidal discretization:

- **Two possibilities**: find the more adapted basis (actually it is not clear) or propose different toroidal discretizations and switch depending on the test case.
- Possible discretizations
  - 3D B-Splines on triangular meshes.
  - 3D B-Splines on mixed triangular-quadrangular (aligned ?) meshes.
  - Fourier method. Classical Fourier method (current JOREK method) or Mapped Fourier method (H. Guillard)

Preconditioning and solver in DJANGO

Implicit scheme for wave equation

- Damping wave equation (baby problem used for Inertial fusion confinement)

$$\begin{cases} \partial_t p + c\nabla \cdot \mathbf{u} = 0 \\ \partial_t \mathbf{u} + c\nabla p = \varepsilon \Delta \mathbf{u} \end{cases}$$

- This problem is stiff in time for fast waves. CFL condition close to $\Delta t \leq C_1 \frac{h}{c}$.
- Simple way to solve this: implicit scheme but the model is ill-conditioned.
- Two sources of ill-conditioning: the stiff terms (which depend of ε) and the hyperbolic structure.

Philosophy : Divide, reformulate, approximate and solve

- **Divise**: use splitting method to separate the full coupling system between simple operators (advection, diffusion etc).
- **Reformulate**: rewrite the coupling terms as second order operator simple to invert.
- **Approximate**: use approximations in the previous step to obtain well-posed and well-conditioning simple operators.
- **Solve**: solve the suitability of sub-systems to obtain an approximation of the solution.

Principle of the preconditioning

- The implicit system is given by

$$\begin{pmatrix} M & U \\ L & D \end{pmatrix} \begin{pmatrix} p^{n+1} \\ \mathbf{u^{n+1}} \end{pmatrix} = \begin{pmatrix} R_p \\ R_{\mathbf{u}} \end{pmatrix}$$

with $M = I_d$, $D = \begin{pmatrix} I_d - c\theta\varepsilon\Delta & 0 \\ 0 & I_d - c\theta\varepsilon\Delta \end{pmatrix}$, $L = \begin{pmatrix} \theta c \Delta t \partial_x \\ \theta c \Delta t \partial_y \end{pmatrix}$ and $U = L^t$.

- The solution of the system is given by

$$\begin{pmatrix} p^{n+1} \\ \mathbf{u^{n+1}} \end{pmatrix} = \begin{pmatrix} I & M^{-1}U \\ 0 & I \end{pmatrix} \begin{pmatrix} M^{-1} & 0 \\ 0 & P_{schur}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -LM^{-1} & I \end{pmatrix} \begin{pmatrix} R_p \\ R_{\mathbf{u}} \end{pmatrix}$$

with $P_{schur} = D - LM^{-1}U$.

- Using the previous Schur decomposition, we can solve the implicit wave equation with the algorithm:

$$\begin{cases} \text{Predictor}: & Mp^* = R_p \\ \text{Velocity evolution}: & P\mathbf{u}^{n+1} = (-Lp^* + R_{\mathbf{u}}) \\ \text{Corrector}: & Mp^{n+1} = M_h p^* - U\mathbf{u}_{n+1} \end{cases}$$

- with the matrices:
  - P discretize the positive and symmetric operator :

$$P_{Schur} = I_d - c\varepsilon\theta\Delta I_d - \nabla(\nabla \cdot I_d) = I_d - c\theta\varepsilon\Delta I_d - c2\theta^2\Delta t^2 \begin{pmatrix} \partial_{xx} & \partial_{xy} \\ \partial_{yx} & \partial_{yy} \end{pmatrix}$$

Results for the PC with pressure Schur

- Results for classical Preconditioning (no diffusion).

	Cells	Jacobi		ILU(2)		ILU(4)		ILU(8)	
		iter	Err	iter	Err	iter	Err	iter	Err
$c\Delta t$=1	16*16	-	-	140	2.8E-1	55	4.8E-1	90	1.4E+0
	32*32	-	-	-	-	-	-	180	5.E+0
	64*64	-	-	-	-	-	-	-	-
$c\Delta t$=100	16*16	-	-	88	2.4E-1	58	4.9E-1	88	1.4E+0
	32*32	-	-	-	-	-	-	110	5.6E+0
	64*64	-	-	-	-	-	-	2000	8.8E+1

- Results for the new preconditioning.

	Cells	PB_p		PB_u	
		iter	Err	iter	Err
$a\Delta t$=1	16*16	4	4.9E-2	3	6.8E-2
	32*32	2	9.2E-2	1	1.2E-1
	64*64	2	4.2E-1	1	24
$a\Delta t$=100	16*16	7	1.1E-1	8	4.5E-1
	32*32	6	5.3E-1	6	2.8E+0
	64*64	6	1.E+0	-	-

- For each sub-system we use a CG+Jacobi solver.
- Velocity Schur operator (coupled diffusion operator) not easy to invert and generate a large additional cost.
- On fine grid we use CG+MG 2-cycle for velocity Schur operator.

Some remarks

- **Schur complement on the velocity** since In fluid mechanics and plasma physics the velocity couple all the other equations.
- **Problem** : Schur complement on the velocity not so well-conditioned.
- Wave problem of the hyperbolic problem :
  - Pressure and $(\mathbf{u}, \mathbf{n})$ are propagated at the speed $\pm c$,
  - $(\mathbf{u} \times \mathbf{n})$ is propagated at the speed 0.
- **Idea**: split the propagation (static and non static waves) in the Schur complement using the vorticity equation:

$$\partial_t \mathbf{u} + c\nabla p = \mathbf{f}_u \Longrightarrow \partial_t(\nabla \times \mathbf{u}) = \nabla \times \mathbf{f}_u$$

$$\begin{cases} \text{Predictor :} \quad Mp^* = R_p \\ \text{Vorticity evolution :} \quad \mathbf{w}^{n+1} = R(R_{\mathbf{u}}) \\ \text{Velocity evolution :} \quad P\mathbf{u}^{n+1} = (\alpha R(\mathbf{w}^{n+1}) - Lp^* + R_{\mathbf{u}}) \\ \text{Corrector :} \quad Mp^{n+1} = M_h p^* - U\mathbf{u}_{n+1} \end{cases}$$

- with R the matrix of the curl operator, $\alpha = c^2\theta^2\Delta t^2$ and $P_{Schur} = I_d - (\varepsilon c\theta + \alpha)\Delta$.

Remarks

- The method, the propagation properties and the vorticity prediction can be generalized for compressible fluid mechanics.

Future solver in DJANGO

Aim:

Adaptive and efficient preconditioning for mixte hyperbolic-parabolic problems and full or reduced MHD with free-jacobian matrix.

Possible evolution to have more efficiency

- **The Mass Lumping**: replace the mass matrix (in the PC) by diagonal matrix.
- **Optimization**: algorithm where the matrices are assembled together.
- **Jacobian Free**: use the jacobian free method for the full matrix and for the subsystems of the PC when it is possible.
- **Additional Splitting**: If an operator is to complex to invert we can use a operator splitting to invert easier operators.
- **Geometric Multi-grid with B-Splines**: to invert the subsystems in the PC.

Adaptive PC

- Some matrices of the PC cannot be written with Jacobian-Free method.
- **Idea**: use discretization in the PC with a low memory consumption.
- **Possible Solution**: Adaptive preconditioning where the order and the type of discretization is different between the model and the PC.

Conclusion

- Basic models, discretizations and solvers are present and validated in the code.
- A basic MPI parallelization is present but lot of work must be realized to obtain a efficient code.
- **Coupling with JOREK**: The following important step is the coupling of Django and JOREK (using restart files) to validate the numerical method on realistic cases.

Peoples on Django for the new year

- **An engineer (ADT Nice, 2 years)**: triangular Powell-Sabin finite element, Mapped Fourier method and parallelization.
- **An engineer (Bavarian founding in IPP 6 mouth)**: Jacobian Free method, parallelization open-ACC.
- **An PhD (IPP)**: Compatible B-Splines for Maxwell and MHD models, physic-based preconditioning and adaptivity.
- **An Post-doc (IPP)** : On the meshes construction.
- All the current developers and perhaps new peoples if you are interested.