MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

Struphy - solving plasma physics PDEs within the Python ecosystem

Struphy development team
presented by Stefan Possanner

Max Planck Institute for Plasma Physics, Germany

```python
from struphy.pic.particles import Particles3D
from struphy.feec.psydac_derham import Derham
from mpi4py import MPI

Nel = [128, 128, 16]  # number of elements
p = [2, 2, 3] # spline degrees
spl_kind = [False, True, True] # periodic splines?
comm = MPI.COMM_WORLD
derham = Derham(Nel, p, spl_kind, comm=comm)
particles = Particles3D('ions', comm=comm)
particles.draw_markers()
```

✓ 4.6s

EUROfusion

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Outline

Why Python? - Philospohy of Struphy development

Data structures

Geometric methods and examples

Get in touch

Table of Contents

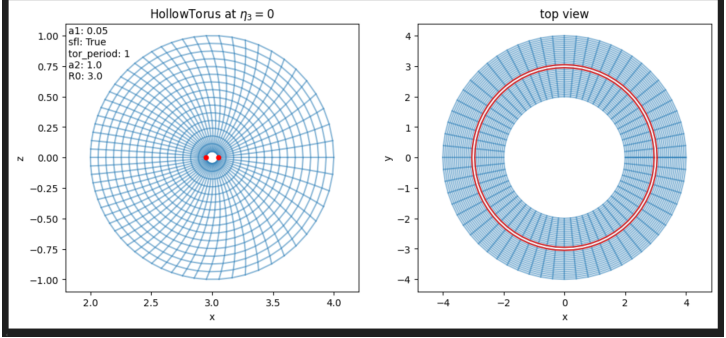
Why Python? - Philospohy of Struphy development

1. `Python` is the English of programming languages.
2. **Clarity, usability and flexibility first**.
3. Be easily accessible/usable by all scientists, for free.
4. Seamless integration with `Python` ecosystem:
   - `psydac` : MPI-distributed high-order spline library
   - `pyccel` : translate and compile in C or Fortran, use OpenMP
   - `mpi4py` : use MPI from Python
   - `numpy` : fast linear algebra
   - `desc-opt` : Stellarator equilibria
   - `pytest` : unit testing
   - and countless more ...
5. Provide an abstract framework for adding **new physics models, quickly**.

How Struphy works

1. Through the command line: `$ struphy run LinearMHD --mpi 8 -o DIR`
2. Through the Struphy API:

```
1 from struphy.geometry import domains
2 domain = domains.HollowTorus(a1=.05, sf1=True, tor_period=1)
3 domain.show()
✓ 1.0s
```



3. Add a new model yourself via the `StruphyModel` abstract base class

What a `StruphyModel` looks like 1/2

1. Inherit abstract base class:

```
class VlasovMaxwellOneSpecies(StruphyModel):
    r'''Vlasov-Maxwell equations for one species.
```

2. Define species (i.e variables) and corresponding data structures in memory:

```
@staticmethod
def species():
    dct = {'em_fields': {}, 'fluid': {}, 'kinetic': {}}

    dct['em_fields']['e_field'] = 'Hcurl'
    dct['em_fields']['b_field'] = 'Hdiv'
    dct['kinetic']['ions'] = 'Particles6D'
    return dct
```

3. Add propagators (push variables $t \to t + \Delta t$):

```
@staticmethod
def propagators_dct():
    return {propagators_fields.Maxwell: ['e_field', 'b_field'],
            propagators_markers.PushEta: ['ions'],
            propagators_markers.PushVxB: ['ions'],
            propagators_coupling.VlasovAmpere: ['e_field', 'ions']}
```

This will auto-generate the parameter input file for the model on first call.

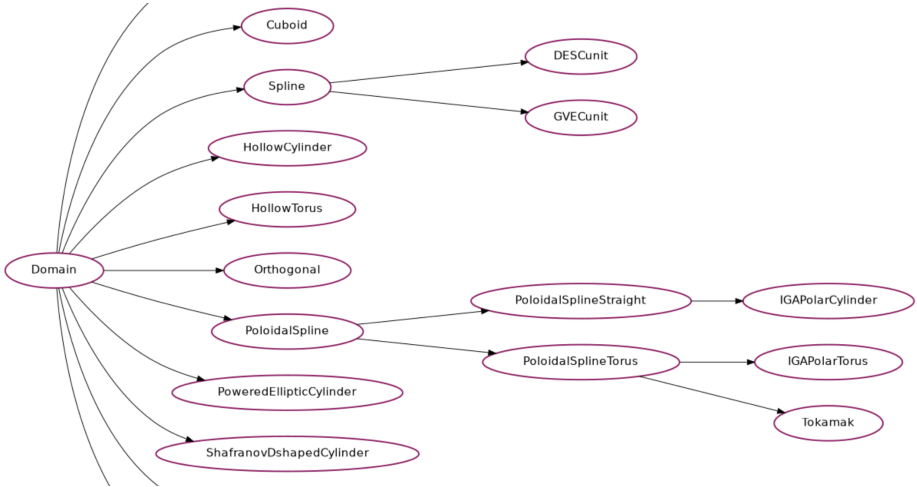
What a `StruphyModel` looks like 2/2

4. Pass simulation parameters to propagators:

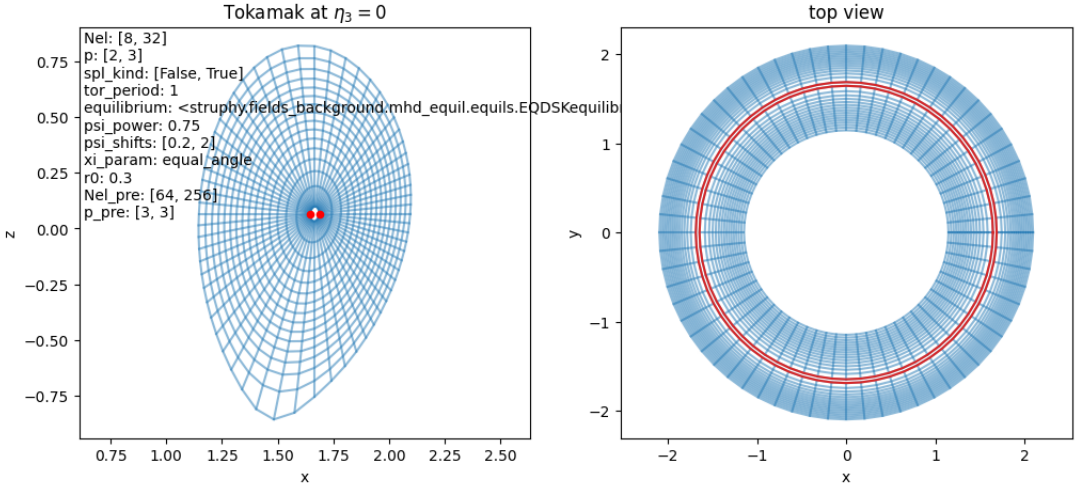
```
# set keyword arguments for propagators
self._kwargs[propagators_fields.Maxwell] = {'solver': params_maxwell}

self._kwargs[propagators_markers.PushEta] = {'algo': algo_eta,
                                             'bc_type': ions_params['markers']['bc']['type']}

self._kwargs[propagators_markers.PushVxB] = {'algo': algo_vxb,
                                             'scale_fac': 1./self._epsilon,
                                             'b_eq': b_backgr,
                                             'b_tilde': self.pointer['b_field']}

self._kwargs[propagators_coupling.VlasovAmpere] = {'c1': self._alpha**2/self._epsilon,
                                                   'c2': 1./self._epsilon,
                                                   'solver': params_coupling}

# Initialize propagators used in splitting substeps
self.init_propagators()
```

The propagators are the building blocks of each model. They can be combined to design new models. There are currently 41 propagators coded in Struphy.
New propagators can be added through the `Propagator` abstract base class.

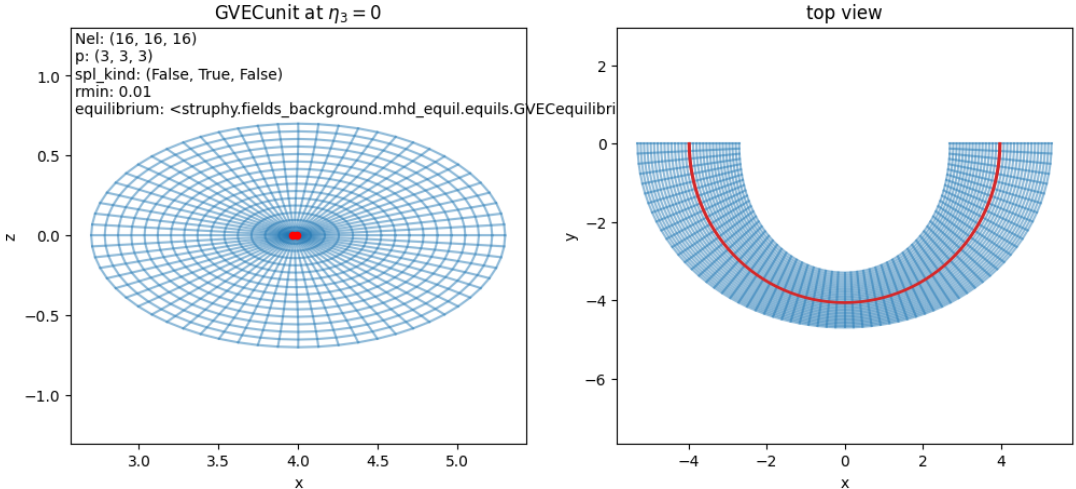
`StruphyModel` can be launched in curvilinear coordinates



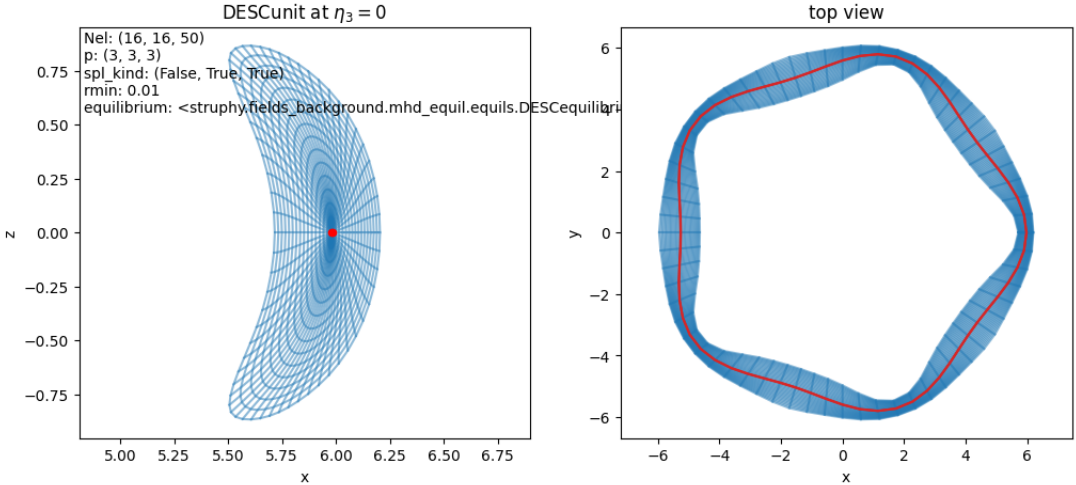
Grid example: from EQDSK file



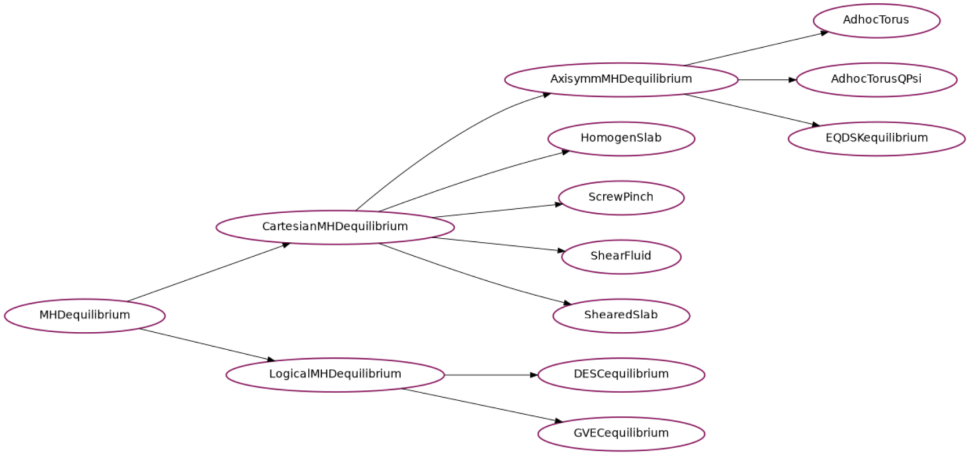
Grid example: from GVEC output



Grid example: from DESC output (W7-X)

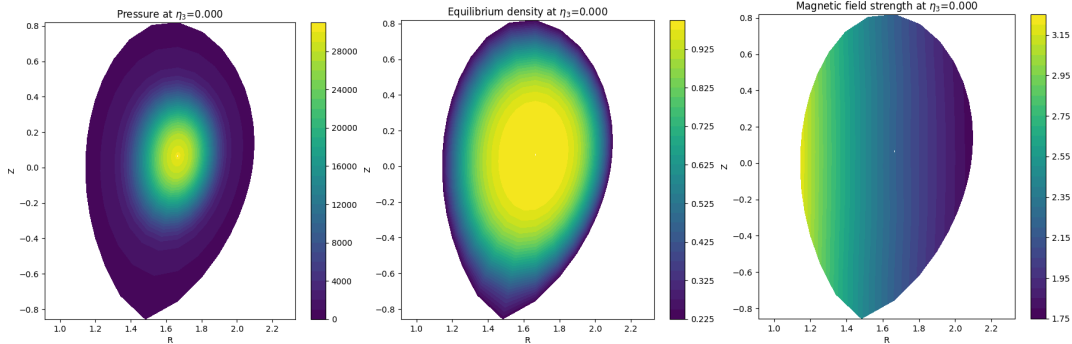


`StruphyModel` can load MHD equilibria



MHD equilibrium example: from EQDSK file

```
1  mhd_equil = equils.EQDSKequilibrium()
2  mhd_equil.domain = domains.Tokamak(equilibrium=mhd_equil)
3  mhd_equil.show()
```



Purpose of Struphy

1. Provide a fast, easy-to-use `Python` framework for plasma physics PDEs
2. Test the newest numerical algorithms in 1D, 2D or 3D fusion scenarios
3. Provide maintainable, accessible and extensible code
4. Spread the open-source paradigm in the academic community

Current developers

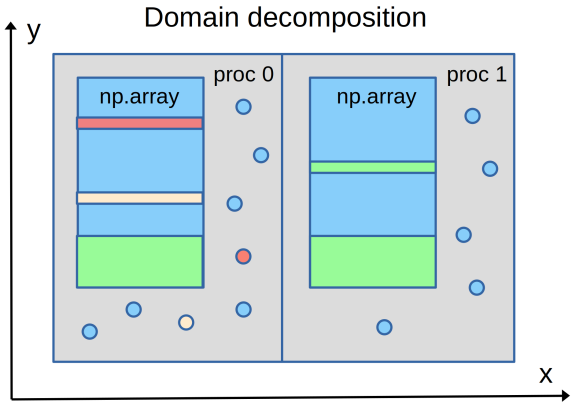
- Dominik Bell (PhD) - energy-conserving δf-methods (with Eric, Martin)
- Valentin Carlier (PhD) - variational MHD (with Martin)
- Ori Saporta-Katz (nt-Tao fusion startup) - MHD in time-varying coil fields
- Yingzhe Li (Post-doc) - 6D Vlasov astrophysics hybrid model (with Eric)
- Max Lindqvist (ACH) - domain cloning, Struphy-Psydac integration (with Roman)
- Nathan Marín (PhD) - quasi-inter-/histopolation, shear-Alfvén preconditioning
- Byung-Kyu Na (PhD) - MHD-drift-kinetic hybrid models (with Xin)
- Philipp Thalhofer (Master) - quasi-neutral fluids with Lagrange multiplier (Omar)
- Matilde Tozzi (Master) - using the `pyAmrex` backend in Struphy (with Eric)

Table of Contents

Struphy data structures

- All `StruphyModels` are implemented in 3D configuration space
- Velocity dimensions can range from 0-3 (at present)
- Lower-dim. simulations can be carried out by setting `Nel = [128, 128, 1]`
- **Particle data structures**: 2D `numpy` arrays with communication buffer
- **Grid data structures**: `Stencil`-format from `psydac` with ghost regions
- Use of matrix-free operators (only `dot` product needed)

Particle data structures

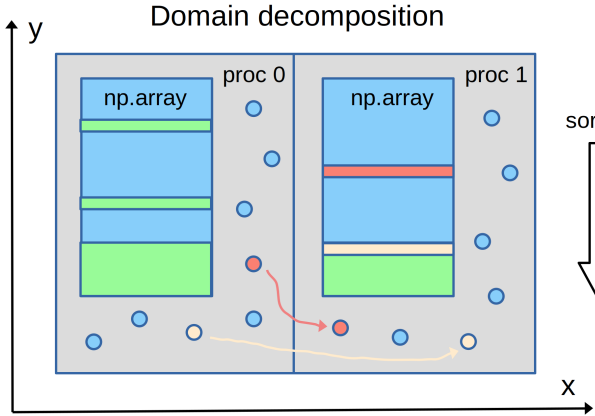


→ each row is a marker

→ columns are marker attributes

→ buffer (green) to manage load imbalance

Particle data structures



sorting

→ sorting with certain frequency

→ boundary conditions: periodic, reflect, remove

→ more info in the `Particles` abstract base class

Grid data structures

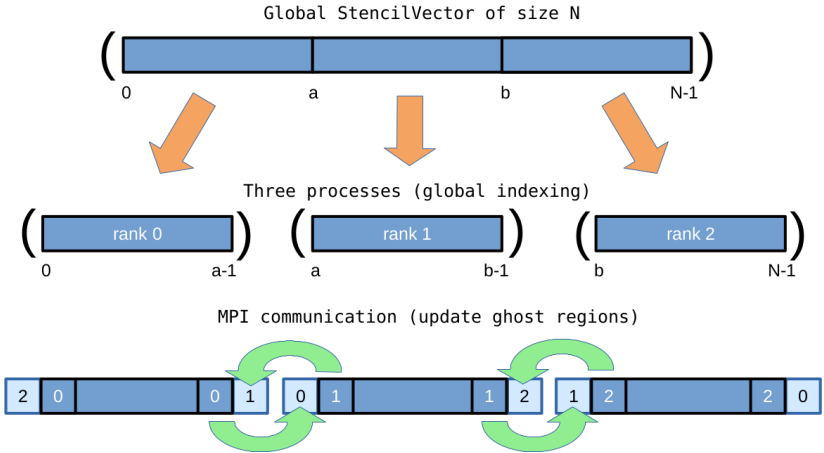
Struphy uses the grid data structures provided by `psydac`:
`https://github.com/pyccel/psydac`
`psydac` main developers: Yaman Güclü (NMPP) and Said Hadjout (former student).

Tensor-product grids on the logical (computational) domain.
Multi-patch with different resolutions.

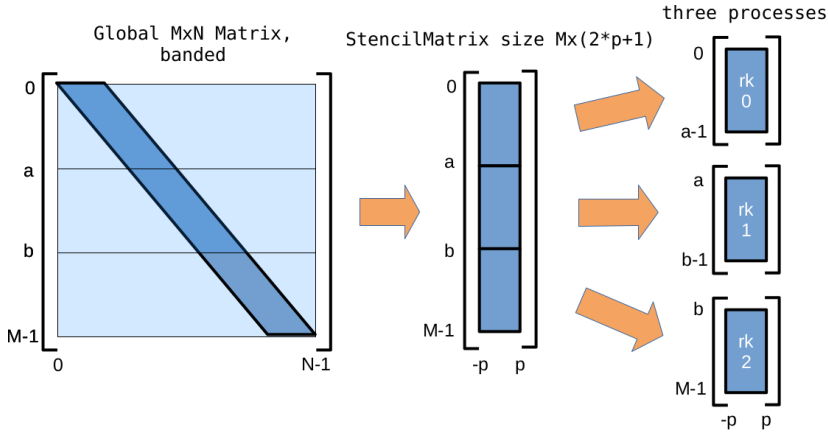
- `StencilVectorSpace`
- `StencilVector` holds the FE coefficients (or other grid data)
- `StencilMatrix` operators on `StencilVector`
- `LinearOperators` are used for matrix-free operations
- particle-to-grid accumulation into `StencilMatrix` or `StencilVector`

These objects have MPI communication via ghost regions.

StencilVector



StencilMatrix



Boundary conditions: periodic, hom. Dirichlet, Neumann (traces of de Rham spaces)

Table of Contents

Lagrangian methods: PIC

Discretization of conservation laws of the form

$$\partial_t f + \nabla_q \cdot (G(f)\, f) = S(f) \qquad q \in \Omega \subset \mathbb{R}^n \,,$$

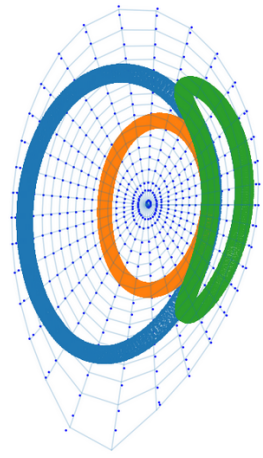
where $\Omega \subset \mathbb{R}^n$ open, some (curvilinear) coordinates $q \in \Omega$ and $G(f)$ a (nonlinear) flow velocity.

$$f \approx f_h(t,q) = \frac{1}{N} \sum_{k=1}^{N} w_k(t)\, \delta(q - q_k(t)) \,.$$

System of ODEs (here for incompressible flow $\nabla_q \cdot G = 0$):

$$\dot{q}_k = G(t, f_h) \qquad q_k(0) = q_{k0} \,,$$

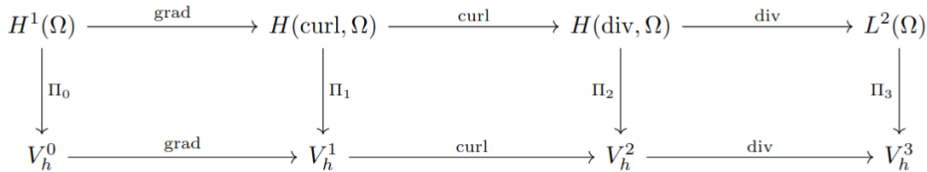
$$\dot{w}_k = \frac{1}{s(t=0, q_{k0})}\, S(f_h) \qquad w_k(0) = \frac{f(t=0, q_{k0})}{s(t=0, q_{k0})} \,.$$



6D Vlasov orbits in EQDSK equilibrium

Finite element exterior calculus (FEEC)

Arnold, D., Falk, R., Winther, R.: Finite element exterior calculus: from Hodge theory to numerical stability. Bulletin of the American mathematical society, 47(2): 281-354, (2010).

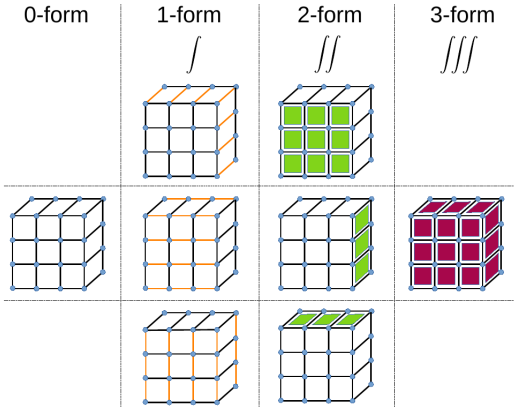


$$u \approx u_h = \sum_{i=1}^{M} u_i \Lambda_i \quad \in V_h\,,$$

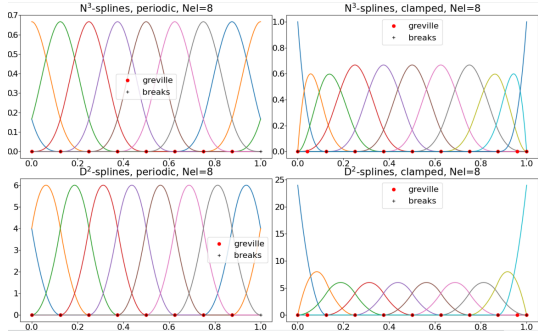
where $\Lambda_i \in V$ are linearly independent basis functions, $(u_i)_i \in \mathbb{R}^M$ are FE coefficients.

- $\Pi_0 : H^1(\Omega) \to V_h^0, \quad \Pi_1 : H(\mathrm{curl},\Omega) \to V_h^1, \quad \Pi_2 : H(\mathrm{div},\Omega) \to V_h^2, \quad \Pi_3 : L^2(\Omega) \to V_h^3$
- $\mathrm{curl}\,\mathrm{grad} = 0\,, \qquad \mathrm{div}\,\mathrm{curl} = 0$
- $\Pi_1 \mathrm{grad} = \mathrm{grad}\,\Pi_0\,, \qquad \Pi_2 \mathrm{curl} = \mathrm{curl}\,\Pi_1\,, \qquad \Pi_3 \mathrm{div} = \mathrm{div}\,\Pi_2$

Geometric DOFs and B-splines



Point values, edge, face and volume integrals



High-order B-splines, tensor product:

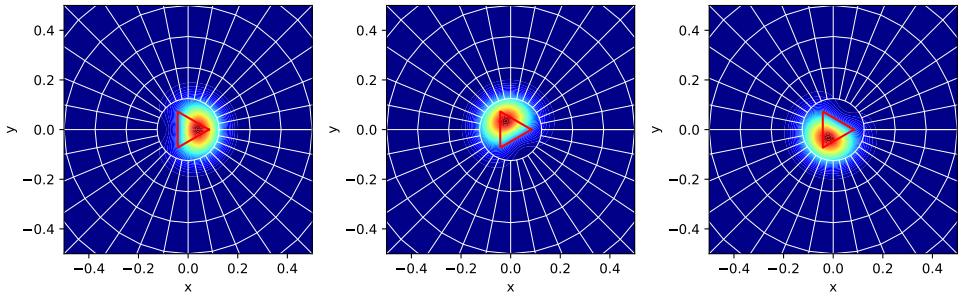
$$\Lambda^{1,1}_{ijk}(\mathbf{x}) = D_i^{p-1}(x) N_j^p(y) N_k^p(z)$$

Polar splines at magnetic axis

- Polar spline framework for discrete differential forms

  Toshniwal et. al., Comput. Meth. Appl. Mech. Engrg. **376**, 113576 (2021)

- 3 new $\mathcal{C}^1$-continuous polar splines around pole

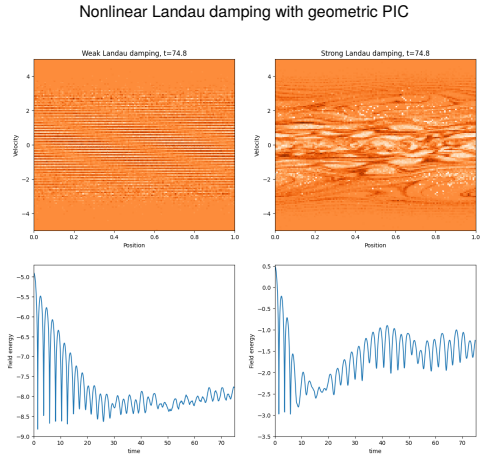


- Polar sub-spaces $\overline{V}_h^n \subset V_h^n$ contain C^0-functions on mapped domain

Wave-particle resonances

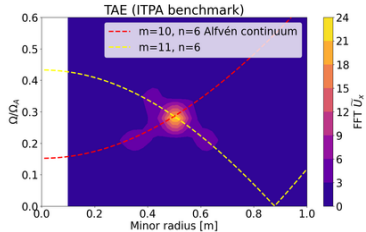
- Study coupling of energetic particles to waves
- Current focus on MHD-kinetic hybrid models
- An improved particle sorting algorithm for nearest neighbour search is being implemented

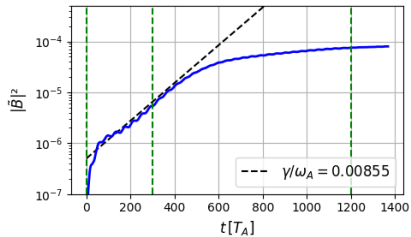
  → completely mesh-free methods (direction SPH)

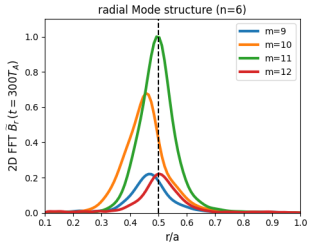


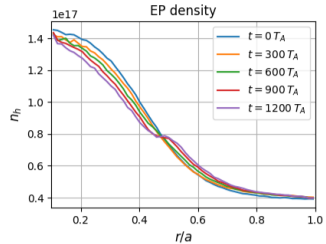
TAE benchmark (courtesy of Byung-Kyu Na) 1/3

full linear MHD-drift-kinetic hybrid model, use of binomial filter:



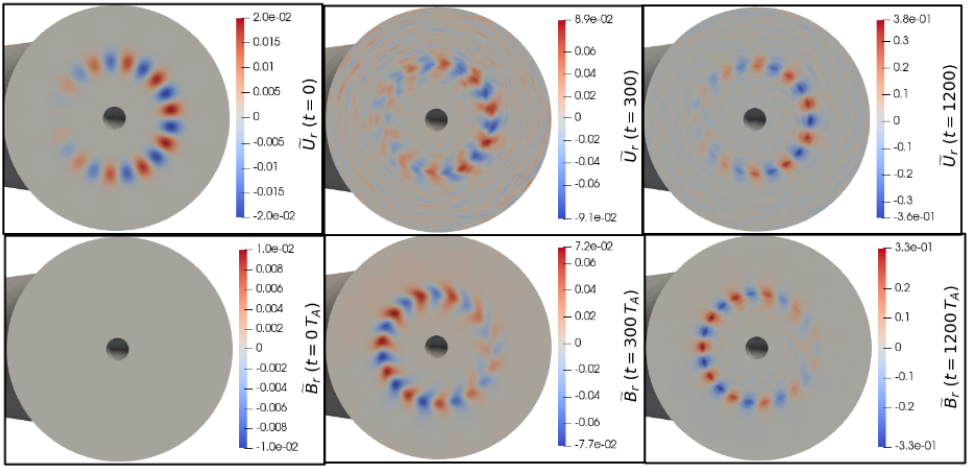






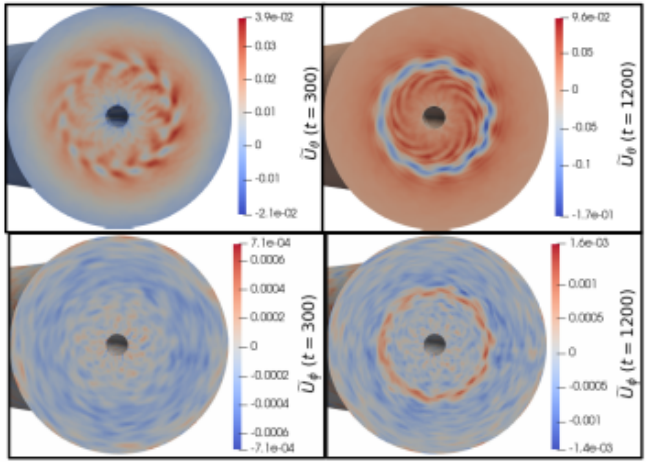
TAE benchmark (courtesy of Byung-Kyu Na) 2/3

Radial velocity and magnetic field:



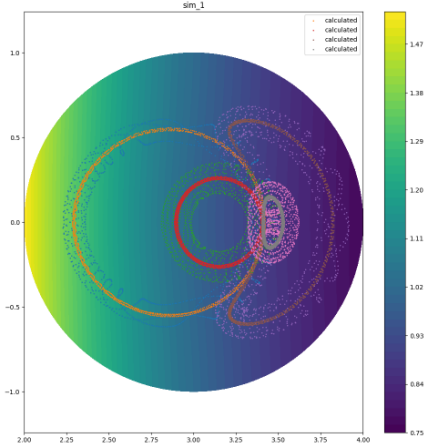
TAE benchmark (courtesy of Byung-Kyu Na) 3/3

Zonal flows? Contact: byungkyu.na@ipp.mpg.de

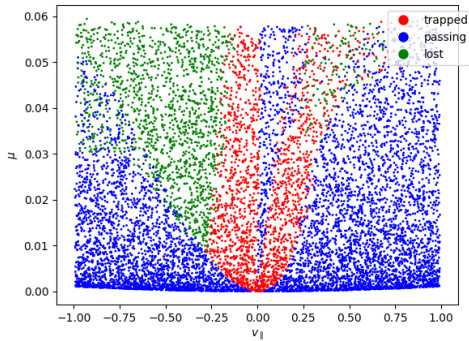


Ongoing study: comparing particle alogrithms

Full-orbit vs. guiding-center, RK4 vs. implicit discrete gradient, etc.



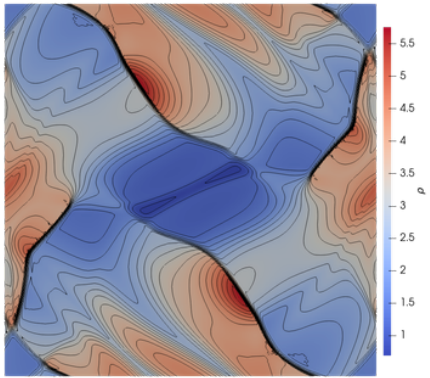
Verification of guiding-center orbits



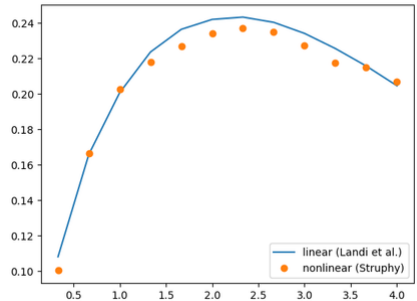
Classifications

Verification of Variational MHD (courtesy of Valentin Carlier)

full nonlinear MHD from variational principle. Contact: Valentin.Carlier@ipp.mpg.de



Orszag-Tang vortex, with artifical viscostiy



Current sheet instability

Table of Contents

Installing Struphy

- Virtual environment:

```
python -m venv env
source env/bin/activate
```

- Current (stable) version:

```
pip install struphy
struphy compile
```

https://struphy.pages.mpcdf.de/struphy/index.html

- Install from source:

```
git clone https://gitlab.mpcdf.mpg.de/struphy/struphy.git
cd struphy
pip install -e .
struphy compile
```

Get in touch

- Google `struphy`
- Documentation: `https://gitlab.mpcdf.mpg.de/struphy/struphy`
- Repository: `https://gitlab.mpcdf.mpg.de/struphy/struphy`
- Ticket system: `https://gitlab.mpcdf.mpg.de/struphy/struphy/-/issues`
- Developer's channel: `https://chat.gwdg.de/channel/struphy-developers`
- Follow Struphy on LinkedIn: `https://www.linkedin.com/company/struphy/`
- Main contacts:
  - Eric Sonnendrücker
  - Xin Wang
  - myself