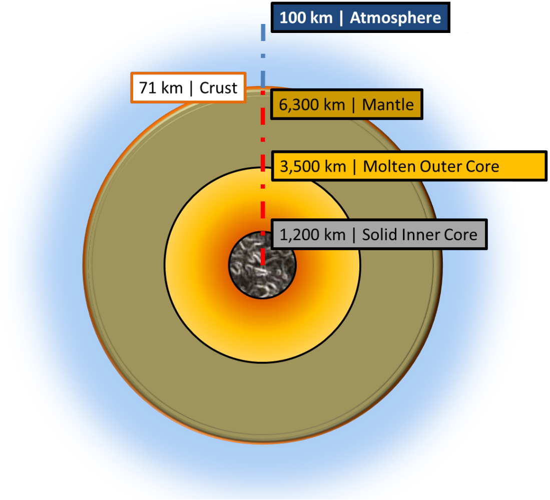
Machine learning of subgrid-scale contribution to the Earth dynamo

Hugo Frezat (IPGP), Nathanaël Schaeffer (ISTerre), Alexandre Fournier (IPGP), Thomas Gastine (IPGP)

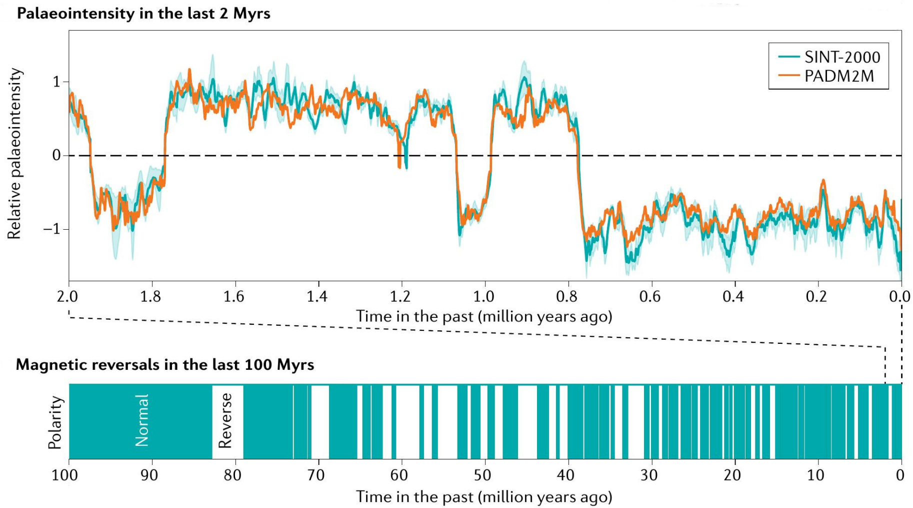
ChEESE2 CoE

Numpex, Saclay, 8 November 2023

Structure of the Earth



Polarity reversals of Earth's magnetic field



From Landeau+ 2022, Nature Review Earth & Environment

Scientific questions

Earth's magnetic field is generated in its liquid core by a dynamo effect ($=$ self-induction of a magnetic field).

- Main questions:
  - Magnetic field reversals?
  - Role of turbulence?
- Difficulties:
  - Broad range of length-scales (from 1 to 10^6 meters)
  - Broad range of time-scales (from 1 day to million years)
  - Turbulent motion (very high Reynolds number $Re \gtrsim 10^8$).

Basic rotating MHD in planetary cores

Navier-Stokes equation

$$\partial_t \mathbf{u} + (2/E\,\mathbf{e_z} + \nabla \times \mathbf{u}) \times \mathbf{u} = -\nabla p + \Delta \mathbf{u} + (\nabla \times \mathbf{b}) \times \mathbf{b} - Ra/Pr\; T\; \vec{r}$$

Induction equation

$$\partial_t \mathbf{b} = \nabla \times (\mathbf{u} \times \mathbf{b}) + 1/Pm\, \Delta \mathbf{b}$$

Temperature equation

$$\partial_t T + \mathbf{u}.\nabla T = 1/Pr\, \Delta T$$

$$E = \nu / D^2 \Omega \sim 10^{-15}$$
$$Pm = \nu \mu_0 \sigma \sim 10^{-5}$$

$$Ra = \Delta T \alpha g D^3 / \kappa \nu \gg 1$$
$$Pr = \nu / \kappa \sim 1$$

the XSHELLS code: resistive MHD in the sphere

A high performance simulation code for rotating incompressible flows and magnetic fields in spherical shells.

- Written in **C++**
- Free & Open-source software `https://nschaeff.bitbucket.io/xshells`
- Dependencies: FFTW (or MKL) and SHTns
- Parallelization: domain decomposition with MPI + OpenMP.
- Works well on GPU (cuda/hip).
- Experimental mixed-precision mode on GPU.

XSHELLS code is freely available `https://gricad-gitlab.univ-grenoble-alpes.fr/schaeffn/xshells`

Spatial discretization (1)

- Vector fields are divergenceless: use two scalar representation (Poloidal/Toroidal):

$$\mathbf{u} = \nabla \times (T\mathbf{r}) + \nabla_{\times}\nabla \times (P\mathbf{r})$$

  - reduce the number of degrees of freedom (memory footprint and bandwidth);
  - ensures incompressibility;
  - ensures magnetic field is divergenceless.

Spatial discretization (2)

- Scalars are decomposed into spherical harmonics:

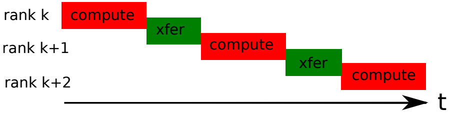
$$S(r,\theta,\phi) = \sum_{\ell} \sum_{m\leq\ell} s_{\ell,m}(r) Y_{\ell,m}(\theta,\phi)$$

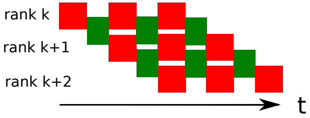
  - pros of spectral method: good accuracy with reduced degrees of freedom;
  - allows to write a local boundary condition for magnetic field matching a potential field;
  - no obvious domain decomposition for parallelization;
  - no "fast" algorithm in practices, but efficient implementation
  - diagonlizes Laplace operator (diffusive terms)

Algorithmic Motif AM1: spherical harmonic transform already highly optimized in SHTns library (vectorized CPU and CUDA/HIP).
Possible improvement: distribute SHTs accross multiple GPUs (MPI and/or NCCL).
`https://gricad-gitlab.univ-grenoble-alpes.fr/schaeffn/shtns`

Spatial discretization (3)

- Use finite differences in radial direction:
  - local formulation: easy domain decomposition;
  - fast solves (Thomas algorithm for banded matrices)
- Combination of spherical harmonics and finite differences leads to a large collection of independent linear solves, with many rhs.
  - straightforward parallelization;
  - software pipelining to reduce latency.





Algorithmic Motif AM2: collection of banded matrix solves

Possible improvements:

- using SPIKE solver (distributed)
- using local transpose (e.g. whithin node).

Time discretization

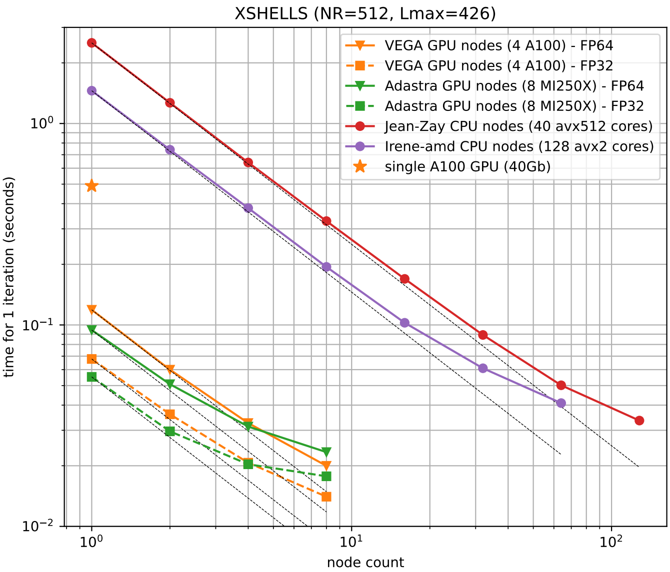
Implcit-Excplicit (IMEX) time stepping

- Diffusion treated implicitily (important because of thin diffusive dboundary layers)
- Other terms treated explicitly
- Experimental: explicit terms can be computed using single precision on GPU (mixed-precision mode)

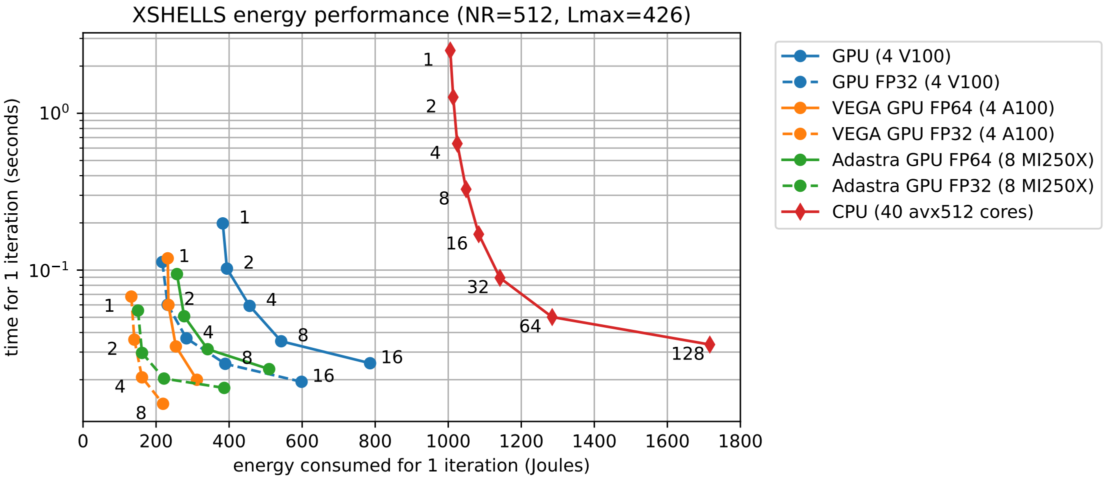
Several efficient time-stepping schemes, with automatic time-step adjustment:

- IMEX Backwards Difference Formula (BDF), order 2 or 3
- Crank-Nicolson (implicit) Adams-Bashforth (explicit), order 2
- special Predictor-Corrector, order 2, highly stable
- Some Additive Runge-Kutta (namely BPR353, order 3, good stability)

Current performance



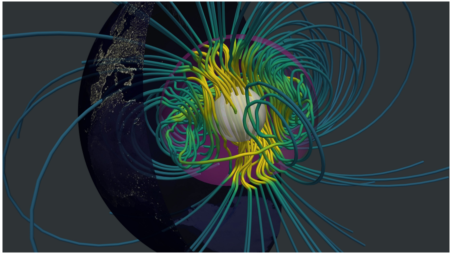
Current energy performance



State of the art simulation (DNS) and Objective

2017 : Schaeffer et. al.

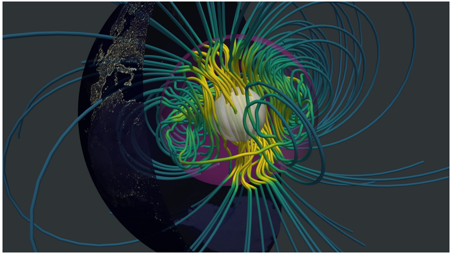
- 2688 x 1504 x 1280
- $E = 10^{-7}$, **Re=5000**, Pm=0.1
- many emergent Earth-like features
- short time-span (5% of magnetic diffusion time)



State of the art simulation (DNS) and Objective

2017 : Schaeffer et. al.

- 2688 x 1504 x 1280
- $E = 10^{-7}$, **Re=5000**, Pm=0.1
- many emergent Earth-like features
- short time-span (5% of magnetic diffusion time)



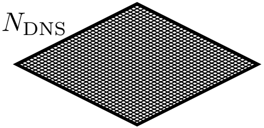
Objective O1: comparable parameters, but x100 to x1000 time-span to study reversals

- single-node optimizations: not much left, close to memory bandwidth limit.
- parallelization: some room to improve scalability (SPIKE solver, better domain decomposition, ...)
- We need subgrid-scale modelling!

Accelerating simulations and reaching higher resolutions

Direct numerical simulation (DNS):

$$\frac{\partial \mathbf{y}}{\partial t} = f(\mathbf{y})$$



Grids on domain length L and corresponding energy spectrum.

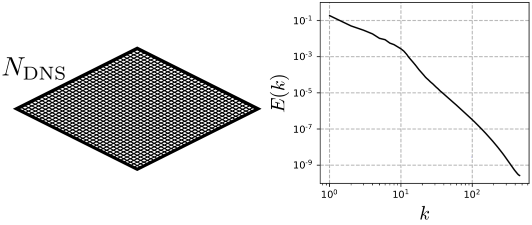
Hardware and optimizations:

- Exascale computing.
- Hybrid GPU architectures.

Accelerating simulations and reaching higher resolutions

Direct numerical simulation (DNS):

$$\frac{\partial \mathbf{y}}{\partial t} = f(\mathbf{y})$$



Grids on domain length L and corresponding energy spectrum.

Hardware and optimizations:

- Exascale computing.
- Hybrid GPU architectures.

Reduced equations (LES):

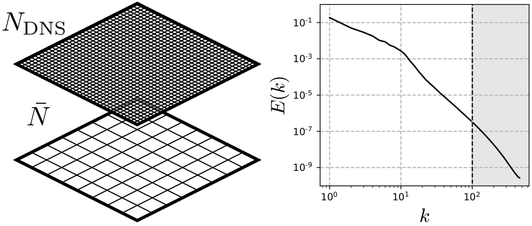
- Universal small-scale dynamics.
- Applying projection $\mathcal{T}(\mathbf{y}) = \bar{\mathbf{y}}$.
- Typically using a filter.

$$\frac{\partial \bar{\mathbf{y}}}{\partial t} = f(\bar{\mathbf{y}}) + \underbrace{\tau(\mathbf{y})}_{\mathcal{T}(f(\mathbf{y})) - f(\mathcal{T}(\mathbf{y}))}$$

Accelerating simulations and reaching higher resolutions

Direct numerical simulation (DNS):

$$\frac{\partial \mathbf{y}}{\partial t} = f(\mathbf{y})$$



Grids on domain length L and corresponding energy spectrum.

Hardware and optimizations:

- Exascale computing.
- Hybrid GPU architectures.

Reduced equations (LES):

- Universal small-scale dynamics.
- Applying projection $\mathcal{T}(\mathbf{y}) = \bar{\mathbf{y}}$.
- Typically using a filter.

$$\frac{\partial \bar{\mathbf{y}}}{\partial t} = f(\bar{\mathbf{y}}) + \underbrace{\tau(\mathbf{y})}_{\mathcal{T}(f(\mathbf{y})) - f(\mathcal{T}(\mathbf{y}))}$$

State of the art: historical

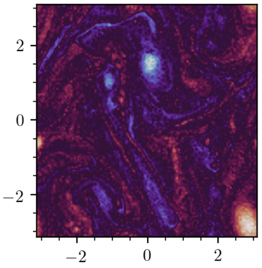
"Historical" – or physical turbulence models (*Sagaut, 2006*):

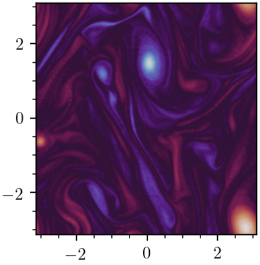
- Mathematical developments (*Clark et al., 1979*): **Structural**.

	Structural
Stability	-
Forward	+
Backward	+

Potential difficulties:

- Accumulation of small-scale energy: numerical instabilities.
- Incorrect representation of the unresolved dynamics.





Difficulties in SGS modeling for two-dimensional turbulent systems.

State of the art: historical

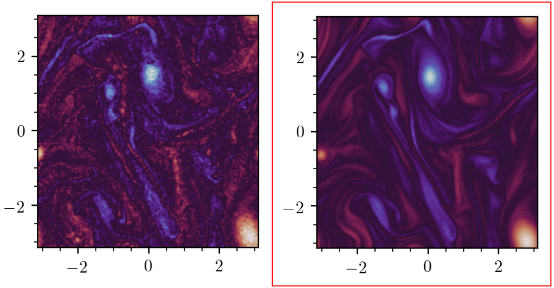
"Historical" – or physical turbulence models (*Sagaut, 2006*):

- Mathematical developments (*Clark et al., 1979*): **Structural**.
- First principles (*Smagorinsky, 1963, Leith, 1996*): **Functional**.

	Structural	**Functional**
Stability	-	+
Forward	+	-
Backward	+	-

Potential difficulties:

- Accumulation of small-scale energy: numerical instabilities.
- Incorrect representation of the unresolved dynamics.



Difficulties in SGS modeling for two-dimensional turbulent systems.

State of the art: machine learning

Current models:

- Exclusive on stability **and** correct transfers.
- Machine learning as an alternative (*Brunton et al., 2020*).

Solving a problem from data:

- Inputs $\bar{\mathbf{y}}$.
- Output τ.
- Model $\mathcal{M} : \bar{\mathbf{y}} \to \tau$.
- "Static".

Sub-grid modelling for two-dimensional turbulence using neural networks

R. Maulik[1], O. San[1]†, A. Rasheed[2], P. Vedula[3]

[1]School of Mechanical & Aerospace Engineering, Oklahoma State University, Stillwater, OK 74078, USA

[2]CSE Group, Applied Mathematics and Cybernetics, SINTEF Digital, N-7465 Trondheim, Norway

[3]School of Aerospace & Mechanical Engineering, The University of Oklahoma Norman, OK 73019, USA

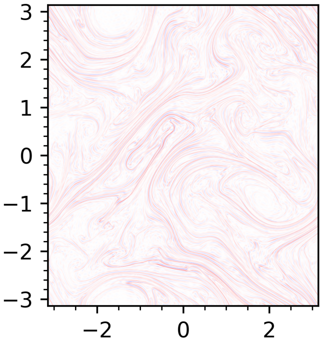
Initial experiments on two-dimensional turbulence (*Maulik et al., 2019*).

	Structural	Functional	ML
Stability	-	+	-
Forward	+	-	++
Backward	+	-	++

Turbulence evaluation metrics

a priori metrics

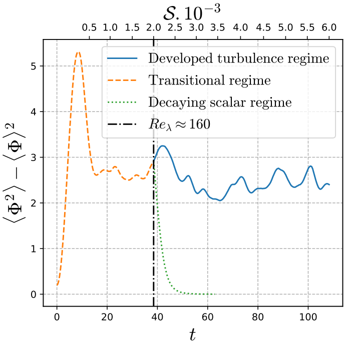
Prediction of the missing term on a **fixed time-step**.



Instantaneous subgrid contribution.

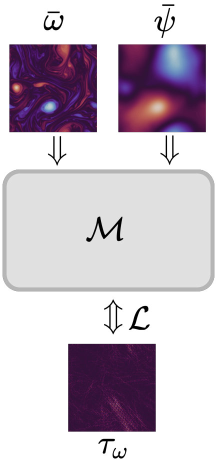
a posteriori metrics

Prediction of the simulation's trajectory over a **temporal horizon**.



Temporal evolution of kinetic energy.

a priori learning



Instantaneous loss computation.

Instantaneous (classical) loss

$$\mathcal{L} := \left\langle \ell(\mathcal{M}(\bar{\psi}, \bar{\omega}), \tau_\omega) \right\rangle_{\mathbf{x}}$$

Optimize only on the **next** temporal increment $t + \Delta t$.

Not perfect: errors can either lead to stable or unstable predictions.

Examples:

$$\ell := (\mathcal{M}(\bar{\psi}, \bar{\omega}) - \tau_\omega)^2$$

$$\vdots$$

$$\ell := \tau_\omega(\log \tau_\omega - \mathcal{M}(\bar{\psi}, \bar{\omega}))$$

a posteriori learning

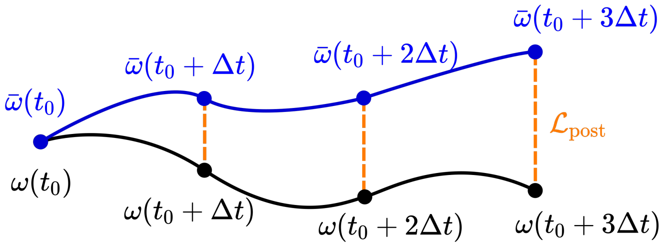
a posteriori loss

$$\mathcal{L} := \langle \ell(\bar{\mathbf{y}}_{\text{pred}}(t), \mathbf{y}(t)) \rangle_{\mathbf{x},\mathbf{t}}$$
$$\bar{\mathbf{y}}_{\text{pred}} \equiv \{\bar{\omega}_{\text{pred}}, \mathcal{M}\}$$
$$\mathbf{y} \equiv \{\omega, \tau_\omega\}$$

Temporal component in loss function.

Required to form a **continuous** trajectory.

Integrate during training: **solver performance** is important.



Visual sketch of an *a posteriori* training on one trajectory.

a posteriori learning

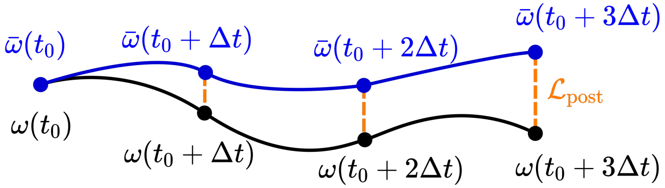
a posteriori loss

$$\mathcal{L} := \langle \ell(\bar{\mathbf{y}}_{\text{pred}}(t), \mathbf{y}(t)) \rangle_{\mathbf{x},\mathbf{t}}$$
$$\bar{\mathbf{y}}_{\text{pred}} \equiv \{\bar{\omega}_{\text{pred}}, \mathcal{M}\}$$
$$\mathbf{y} \equiv \{\omega, \tau_\omega\}$$

Temporal component in loss function.

Required to form a **continuous** trajectory.

Integrate during training: **solver performance** is important.



Visual sketch of an *a posteriori* training on one trajectory.

a posteriori learning

a posteriori loss

$$\mathcal{L} := \langle \ell(\bar{\mathbf{y}}_{\text{pred}}(t), \mathbf{y}(t)) \rangle_{\mathbf{x},\mathbf{t}}$$

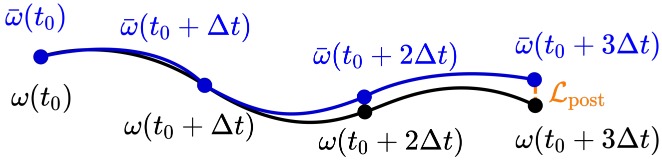
$$\bar{\mathbf{y}}_{\text{pred}} \equiv \{\bar{\omega}_{\text{pred}}, \mathcal{M}\}$$

$$\mathbf{y} \equiv \{\omega, \tau_\omega\}$$

Temporal component in loss function.

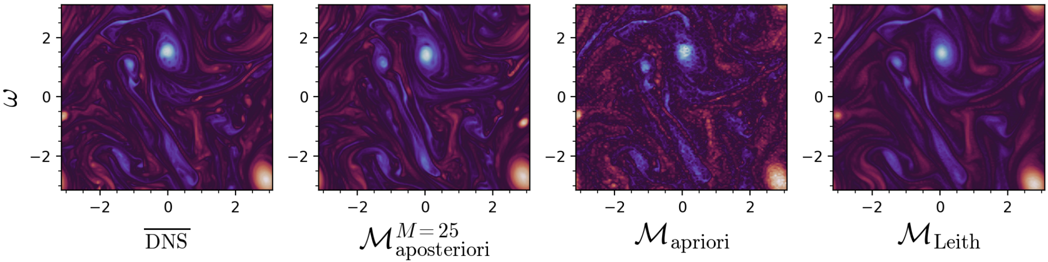
Required to form a **continuous** trajectory.

Integrate during training: **solver performance** is important.



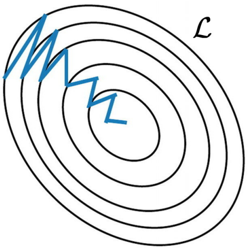
Visual sketch of an *a posteriori* training on one trajectory.

Numerical experiments: forced turbulence



Forced turbulence (*Graham et al., 2013*)

The differentiability requirement



Loss function minimization using gradient descent (classical for NNs).

Gradient-based mathematical optimization:

$$\mathcal{M}(\mathbf{y} \mid \theta) : \arg\min_{\theta} \mathcal{L}$$

$$\Downarrow$$

$$\theta_{n+1} = \theta_n - \gamma \nabla_\theta \mathcal{L}$$

a priori vs *a posteriori*: losses

The differentiability requirement

a priori loss gradient:

$$
\begin{aligned}
&\nabla_\theta \ell_{\text{prio}}(\mathcal{M}, \tau_\omega) \\
&\quad = \frac{\partial \ell_{\text{prio}}}{\partial \tau_\omega}\frac{\partial \tau_\omega}{\partial \theta} + \frac{\partial \ell_{\text{prio}}}{\partial \mathcal{M}}\frac{\partial \mathcal{M}}{\partial \theta} \\
&\quad = \frac{\partial \ell_{\text{prio}}}{\partial \mathcal{M}}\underbrace{\frac{\partial \mathcal{M}}{\partial \theta}}_{\text{AD}}
\end{aligned}
$$

a posteriori loss gradient:

$$
\begin{aligned}
&\nabla_\theta \ell_{\text{post}}(\bar{\mathbf{y}}_{\text{pred}}(t), \mathbf{y}(t)) \\
&\quad = \frac{\partial \ell_{\text{post}}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \theta} + \frac{\partial \ell_{\text{post}}}{\partial \bar{\mathbf{y}}_{\text{pred}}}\frac{\partial \bar{\mathbf{y}}_{\text{pred}}}{\partial \theta} \\
&\quad = \frac{\partial \ell_{\text{post}}}{\partial \bar{\mathbf{y}}_{\text{pred}}}\left(\int_{t_0}^{t} \frac{\partial\, \text{solver}}{\partial \theta} + \underbrace{\frac{\partial \mathcal{M}}{\partial \theta}}_{\text{AD}} \mathrm{d}t'\right)
\end{aligned}
$$

a priori vs *a posteriori*: losses

The differentiability requirement

a priori loss gradient:

$$\nabla_\theta \ell_{\text{prio}}(\mathcal{M}, \tau_\omega) = \frac{\partial \ell_{\text{prio}}}{\partial \tau_\omega}\frac{\partial \tau_\omega}{\partial \theta} + \frac{\partial \ell_{\text{prio}}}{\partial \mathcal{M}}\frac{\partial \mathcal{M}}{\partial \theta} = \frac{\partial \ell_{\text{prio}}}{\partial \mathcal{M}} \underbrace{\frac{\partial \mathcal{M}}{\partial \theta}}_{\text{AD}}$$

a posteriori loss gradient:

$$\nabla_\theta \ell_{\text{post}}(\bar{\mathbf{y}}_{\text{pred}}(t), \mathbf{y}(t)) = \frac{\partial \ell_{\text{post}}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \theta} + \frac{\partial \ell_{\text{post}}}{\partial \bar{\mathbf{y}}_{\text{pred}}}\frac{\partial \bar{\mathbf{y}}_{\text{pred}}}{\partial \theta} = \frac{\partial \ell_{\text{post}}}{\partial \bar{\mathbf{y}}_{\text{pred}}}\left(\int_{t_0}^{t} \underbrace{\frac{\partial\, \text{solver}}{\partial \theta}}_{\text{Not available}} + \underbrace{\frac{\partial \mathcal{M}}{\partial \theta}}_{\text{AD}} \mathrm{d}t'\right)$$

a priori vs *a posteriori*: losses

Technical alternatives

Gradient of the solver w.r.t. model parameters:

- Estimates using numerical derivatives.
- Manually implement adjoint.
- **Re-implementation using auto-differentiation languages or libraries.**
- **Deep Differentiable Emulators** (*Frezat et al., 2023, Nonnenmacher and Greenberg, 2021*)

a posteriori loss gradient:

$$\frac{\partial \ell_{\mathrm{post}}}{\partial \bar{\mathbf{y}}_{\mathrm{pred}}} \left(\int_{t_0}^{t} \underbrace{\frac{\partial\, \mathrm{solver}}{\partial \theta}}_{\text{Not available}} + \underbrace{\frac{\partial \mathcal{M}}{\partial \theta}}_{\text{AD}} \, \mathrm{d}t' \right)$$

Differentiable programming libraries in Julia and Python.

Technical alternatives

Gradient of the solver w.r.t. model parameters:

- Estimates using numerical derivatives.
- Manually implement adjoint.
- **Re-implementation using auto-differentiation languages or libraries.**
- **Deep Differentiable Emulators** (*Frezat et al., 2023, Nonnenmacher and Greenberg, 2021*)

a posteriori loss gradient:

$$\frac{\partial \ell_{\mathrm{post}}}{\partial \bar{\mathbf{y}}_{\mathrm{pred}}} \left(\int_{t_0}^{t} \underbrace{\frac{\partial\, \mathrm{solver}}{\partial \theta}}_{\text{AD}} + \underbrace{\frac{\partial \mathcal{M}}{\partial \theta}}_{\text{AD}} \mathrm{d}t' \right)$$

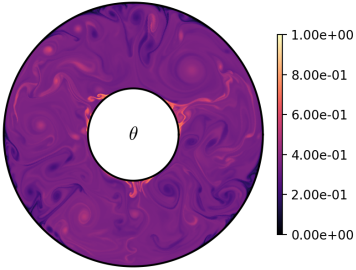
Zygote

Differentiable programming libraries in Julia and Python.

Building efficient solvers with JAX

"High-performance numerical computing and large-scale machine learning research"

- `.grad()`: **Computes the gradient of the function w.r.t. any parameters (auto differentiation).**
- `.jit()`: Compiles and optimizes on the fly for different architectures.
- `.pmap()`: Parallelize over many GPUs / CPU nodes.
- Writing vectorized code.
- Lacking important scientific algorithms (sparses solvers for e.g.).



A 2d convection code in an annulus geometry built with JAX in 100-200 LoC, benchmark TBD.

Objectives

O0. Running dynamo simulation at $Re = 5000$ for multiple viscous time.

O1. Provide banded solvers with their adjoint for GPUs (and/or JAX)

O2. Optional: couple trained model in Python with high performance solver

O3. Questions ?