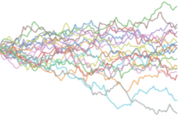
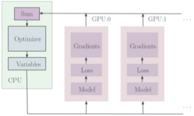
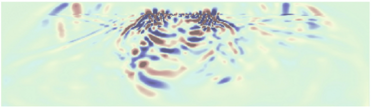
Machine Learning for Inverse Problems in Computational Engineering

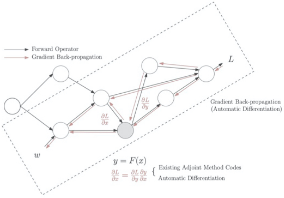
Kailai Xu and Eric Darve

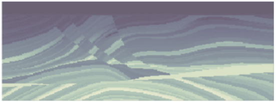
https://github.com/kailaix/ADCME.jl

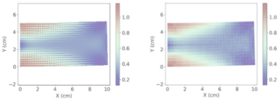












Outline

Inverse Modeling

Forward Problem

Model Parameters → Physical Laws → Prediction of Observations

Inverse Problem

Observations → Physical Laws → Estimation of Parameters

Inverse Modeling

We can formulate inverse modeling as a PDE-constrained optimization problem

$$\min_{\theta} L_h(u_h) \quad \text{s.t. } F_h(\theta, u_h) = 0$$

- The loss function L_h measures the discrepancy between the prediction u_h and the observation u_{obs}, e.g., $L_h(u_h) = \|u_h - u_{\text{obs}}\|_2^2$.
- θ is the model parameter to be calibrated.
- The physics constraints $F_h(\theta, u_h) = 0$ are described by a system of partial differential equations or differential algebraic equations (DAEs); e.g.,

$$F_h(\theta, u_h) = A(\theta)u_h - f_h = 0$$

Function Inverse Problem

$$\min_f L_h(u_h) \quad \text{s.t. } F_h(f, u_h) = 0$$

What if the unknown is a function instead of a set of parameters?

- Koopman operator in dynamical systems.
- Constitutive relations in solid mechanics.
- Turbulent closure relations in fluid mechanics.
- ...

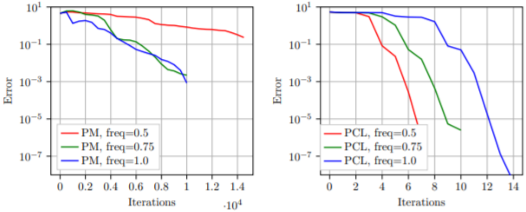
The candidate solution space is infinite dimensional.

Penalty Methods

- Parametrize f with f_θ and incorporate the physical constraint as a penalty term (regularization, prior, ...) in the loss function.

$$\min_{\theta,\, u_h} L_h(u_h) + \lambda \|F_h(f_\theta, u_h)\|_2^2$$

  - May not satisfy physical constraint $F_h(f_\theta, u_h) = 0$ accurately;
  - Slow convergence for stiff problems;



  - High dimensional optimization problem; both θ and u_h are variables.

Machine Learning for Computational Engineering

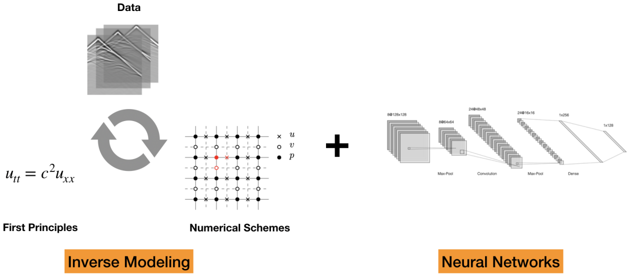
1. Approximate the unknown function with a deep neural network

$$\min_{\theta} L_h(u_h) \quad \text{s.t.} \; F_h(NN_\theta, u_h) = 0$$

2. Reduce the constrained optimization problem to an unconstrained optimization problem by solving the physical constraint numerically

$$\min_{\theta} \tilde{L}_h(\theta) := L_h(u_h(\theta))$$

Satisfy the physics to the largest extent

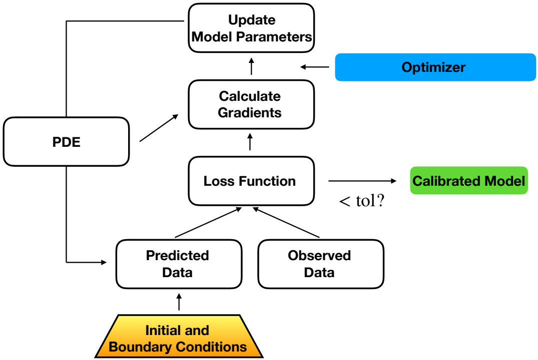


Gradient Based Optimization

$$\min_{\theta} \tilde{L}_h(\theta) := L_h(u_h(\theta)) \tag{1}$$

- We can now apply a gradient-based optimization method to (1).
- The key is to calculate a descent direction g^k

$$\theta^{k+1} \leftarrow \theta^k - \alpha g^k$$



Outline

1. Inverse Modeling
2. ADCME: Automatic Differentiation for Computational and Mathematical Engineering
3. Distributed Computing via MPI
4. Physics Constrained Learning
5. Applications: Constitutive Modeling

Automatic Differentiation

The fact that bridges the technical gap between machine learning and inverse modeling:

- Deep learning (and many other machine learning techniques) and numerical schemes share the same computational model: composition of individual operators.

Mathematical Fact

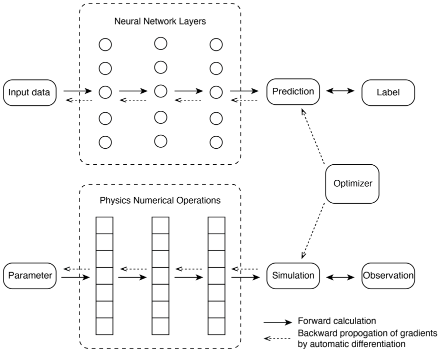
Back-propagation

||

Reverse-mode Automatic Differentiation

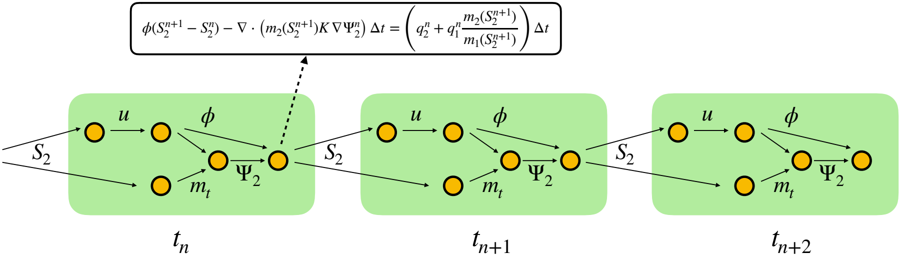
||

Discrete Adjoint-State Method

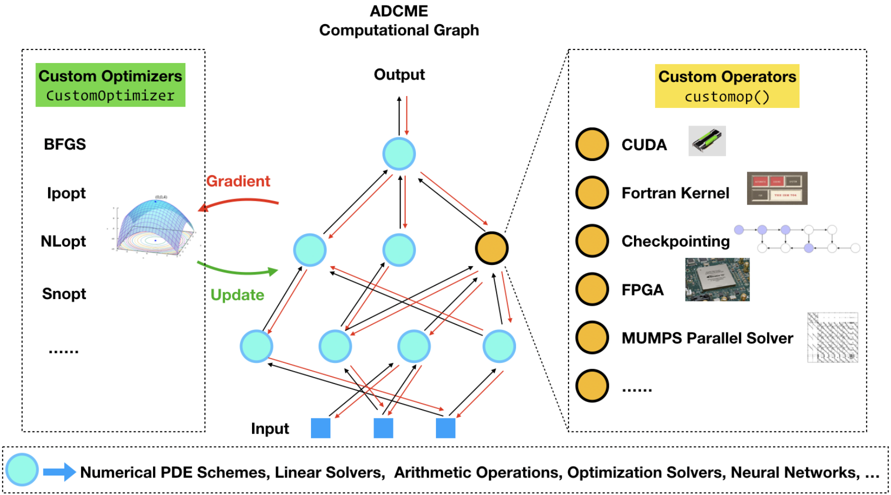


Computational Graph for Numerical Schemes

- To leverage automatic differentiation for inverse modeling, we need to express the numerical schemes in the "AD language": computational graph.
- No matter how complicated a numerical scheme is, it can be decomposed into a collection of operators that are interlinked via state variable dependencies.



ADCME: Computational-Graph-based Numerical Simulation

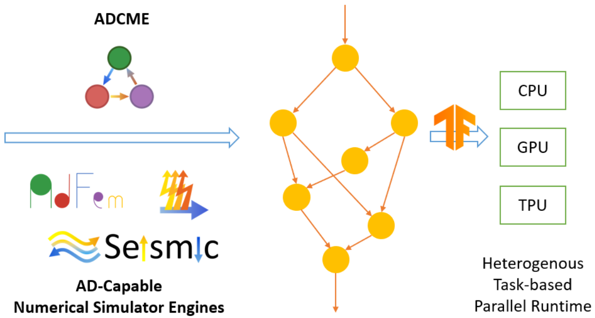


How ADCME works

- ADCME translates your numerical simulation codes to computational graph and then the computations are delegated to a heterogeneous task-based parallel computing environment through TensorFlow runtime.

$$
\begin{aligned}
\operatorname{div} \sigma(u) &= f(x) && x \in \Omega \\
\sigma(u) &= C\varepsilon(u) && \\
u(x) &= u_0(x) && x \in \Gamma_u \\
\sigma(x)n(x) &= t(x) && x \in \Gamma_n
\end{aligned}
$$

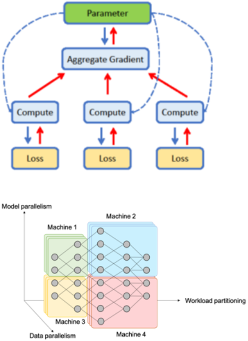
```
mmesh = Mesh(50, 50, 1/50, degree=2)

left = bcnode((x,y)->x<1e-5, mmesh)
right = bcedge((x1,y1,x2,y2)->(x1>0.049-1e-5) && (x2>0.049-1e-5), mmesh)

t1 = eval_f_on_boundary_edge((x,y)->1.0e-4, right, mmesh)
t2 = eval_f_on_boundary_edge((x,y)->0.0, right, mmesh)
rhs = compute_fem_traction_term(t1, t2, right, mmesh)

nu = 0.3
x = gauss_nodes(mmesh)
E = abs(fc(x, [20, 20, 20, 1])|>squeeze)
# E = constant(eval_f_on_gauss_pts(f, mmesh))

D = compute_plane_stress_matrix(E, nu*ones(get_ngauss(mmesh)))
K = compute_fem_stiffness_matrix(D, mmesh)

bdval = [eval_f_on_boundary_node((x,y)->0.0, left, mmesh);
        eval_f_on_boundary_node((x,y)->0.0, left, mmesh)]
DOF = [left;left .+ mmesh.ndof]
K, rhs = impose_Dirichlet_boundary_conditions(K, rhs, DOF, bdval)
u = K\rhs
```



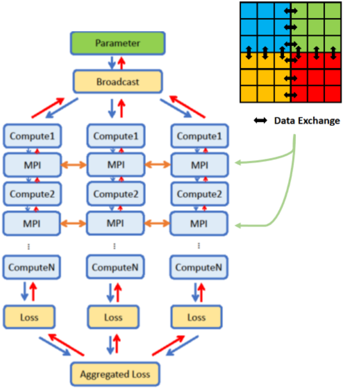
Outline

Parallel Computing

- Parallel computing is essential for accelerating simulation and satisfying demanding memory requirements.



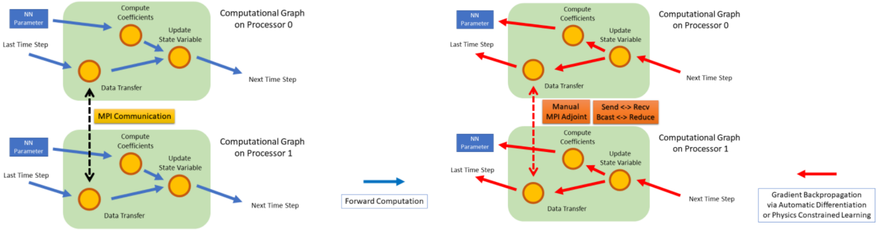
Deep Learning Data/Model Parallelism



Scientific Computing Mixed Parallelism

Distributed Optimization

- ADCME also supports MPI-based distributed computing. The parallel model is designed specially for scientific computing.

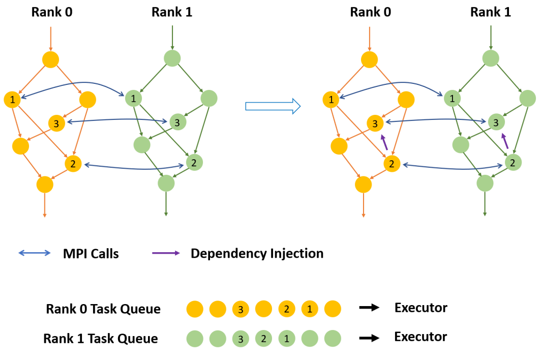


- Key idea: Everything is an operator. Computation and communications are converters of data streams (tensors) through the computational graph.

`mpi_bcast`, `mpi_sum`, `mpi_send`, `mpi_recv`, `mpi_halo_exchange`, ...

Hybrid Parallel Computing

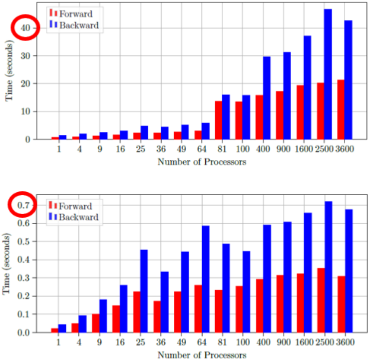
We use **dependency injection** techniques to ensure consistency.



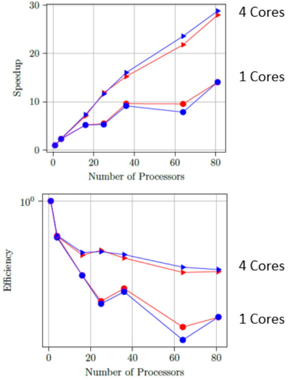
Interoperability with Hypre

$$\nabla \cdot ({\color{red}\mathrm{NN}_\theta(\mathrm{x})} \nabla u(\mathrm{x})) = f(\mathrm{x}) \qquad \mathrm{x} \in \Omega$$
$$u(\mathrm{x}) = 0 \qquad \mathrm{x} \in \partial\Omega$$

The discretization leads to a linear system, which is solved using Hypre.



Weak Scalability



Strong Scalability

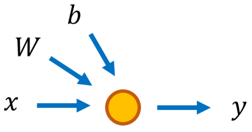
Outline

Challenges in AD

- Most AD frameworks only deal with explicit operators, i.e., the functions that has analytical derivatives, or composition of these functions.
- Many scientific computing algorithms are iterative or implicit in nature.

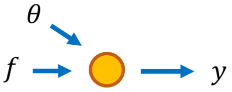
Linear/Nonlinear	Explicit/Implicit	Expression
Linear	Explicit	$y = Ax$
Nonlinear	Explicit	$y = F(x)$
Linear	**Implicit**	$Ay = x$
Nonlinear	**Implicit**	$F(x, y) = 0$

DNN: Explicit



$$y = \sigma(Wx + b)$$

Numerical Schemes: Implicit, Iterative



$$A(y, \theta)y = f$$

Physics Constrained Learning (PCL)

$$\min_{\theta}\ L_h(u_h) \quad \text{s.t.}\ \ F_h(\theta, u_h) = 0$$

- Assume that we solve for $u_h = G_h(\theta)$ with $F_h(\theta, u_h) = 0$, and then

$$\tilde{L}_h(\theta) = L_h(G_h(\theta))$$

- Applying the implicit function theorem

$$\frac{\partial F_h(\theta, u_h)}{\partial \theta} + \frac{\partial F_h(\theta, u_h)}{\partial u_h}\frac{\partial G_h(\theta)}{\partial \theta} = 0 \Rightarrow \frac{\partial G_h(\theta)}{\partial \theta} = -\Big(\frac{\partial F_h(\theta, u_h)}{\partial u_h}\Big)^{-1}\frac{\partial F_h(\theta, u_h)}{\partial \theta}$$

- Finally we have

$$\boxed{\frac{\partial \tilde{L}_h(\theta)}{\partial \theta} = \frac{\partial L_h(u_h)}{\partial u_h}\frac{\partial G_h(\theta)}{\partial \theta} = -\frac{\partial L_h(u_h)}{\partial u_h}\Big(\frac{\partial F_h(\theta, u_h)}{\partial u_h}\Big|_{u_h = G_h(\theta)}\Big)^{-1}\frac{\partial F_h(\theta, u_h)}{\partial \theta}\Big|_{u_h = G_h(\theta)}}$$

Theoretical Analysis

- For stiff problems, better to resolve physics using PCL.
- Consider a model problem

$$\min_\theta \|u - u_0\|_2^2 \qquad \text{s.t.} \;\; Au = \theta y$$

$$\text{PCL}: \qquad \min_\theta \tilde{L}_h(\theta) = \|\theta A^{-1} y - u_0\|_2^2 = (\theta - 1)^2 \|u_0\|_2^2$$

$$\text{Penalty Method}: \qquad \min_{\theta, u_h} \tilde{L}_h(\theta, u_h) = \|u_h - u_0\|_2^2 + \lambda \|Au_h - \theta y\|_2^2$$

Theorem

The condition number of A_λ *is*

$$\liminf_{\lambda\to\infty} \kappa(\mathrm{A}_\lambda) \geq \kappa(A)^2, \qquad \mathrm{A}_\lambda = \begin{bmatrix} I & 0 \\ \sqrt{\lambda} A & -\sqrt{\lambda} y \end{bmatrix}, \qquad \mathrm{y} = \begin{bmatrix} u_0 \\ 0 \end{bmatrix}$$

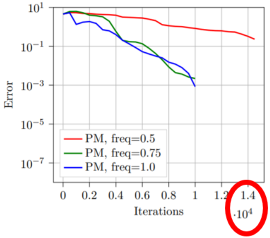
and therefore, the condition number of the unconstrained optimization problem from the penalty method is equal to the the square of the condition number of the PCL asymptotically.

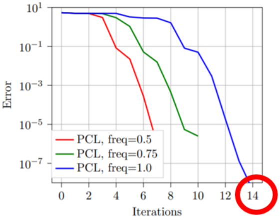
Physics Constrained Learning for Stiff Problems

Parameter Inverse Problem

$$\Delta u + k^2 g(x) u = 0$$
$$g(x) = 5x^2 + 2y^2$$

$$g_\theta(x) = \theta_1 x^2 + \theta_2 y^2 + \theta_3 xy + \theta_4 x + \theta_5 y + \theta_6$$

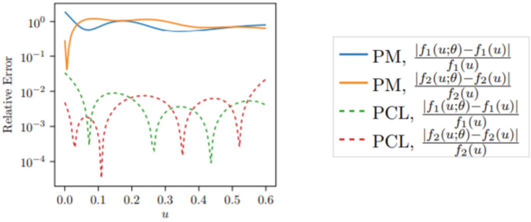




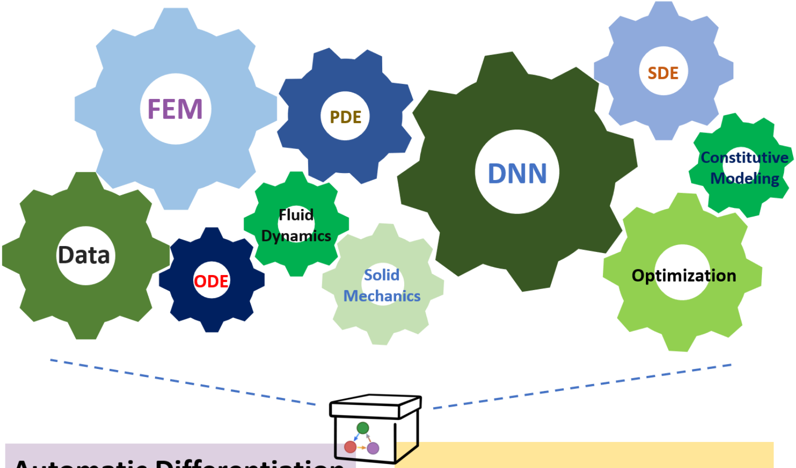
Approximate Unknown Functions using DNNs

$$-\nabla \cdot (\boldsymbol{f}(u) \nabla u) = h(\boldsymbol{x})$$

$$\boldsymbol{f}(u) = \begin{bmatrix} f_1(u) & 0 \\ 0 & f_2(u) \end{bmatrix}$$



PCL: Backbone of the ADCME Infrastructure



Automatic Differentiation

Backend: TensorFlow

Physics Constrained Learning

Outline

Governing Equations

$$
\begin{aligned}
\underbrace{\sigma_{ij,j}}_{\text{stress}} + \rho \underbrace{b_i}_{\text{external force}} &= \rho \underbrace{\ddot{u}_i}_{\text{velocity}} \\
\underbrace{\varepsilon_{ij}}_{\text{strain}} &= \frac{1}{2}(u_{j,i} + u_{i,j})
\end{aligned} \tag{2}
$$

- **Observable**: external/body force b_i, displacements u_i (strains ε_{ij} can be computed from u_i); density ρ is known.
- **Unobservable**: stress σ_{ij}.
- Data-driven Constitutive Relations: modeling the strain-stress relation using a neural network

$$
\boxed{\text{stress} = \mathcal{M}_\theta(\text{strain}, \ldots)} \tag{3}
$$

and the neural network is trained by coupling Eq. 2 and Eq. 3.

Representation of Constitutive Relations

- Proper form of constitutive relation is crucial for numerical stability

$$\text{Elasticity} \Rightarrow \boldsymbol{\sigma} = \mathsf{C}_\theta \boldsymbol{\epsilon}$$

$$\text{Hyperelasticity} \Rightarrow \begin{cases} \boldsymbol{\sigma} = \mathcal{M}_\theta(\boldsymbol{\epsilon}) & \text{(Static)} \\ \boldsymbol{\sigma}^{n+1} = \mathsf{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1})\mathsf{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1})^T(\boldsymbol{\epsilon}^{n+1} - \boldsymbol{\epsilon}^n) + \boldsymbol{\sigma}^n & \text{(Dynamic)} \end{cases}$$

$$\text{Elaso-Plasticity} \Rightarrow \boldsymbol{\sigma}^{n+1} = \mathsf{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^n, \boldsymbol{\sigma}^n)\mathsf{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^n, \boldsymbol{\sigma}^n)^T(\boldsymbol{\epsilon}^{n+1} - \boldsymbol{\epsilon}^n) + \boldsymbol{\sigma}^n$$

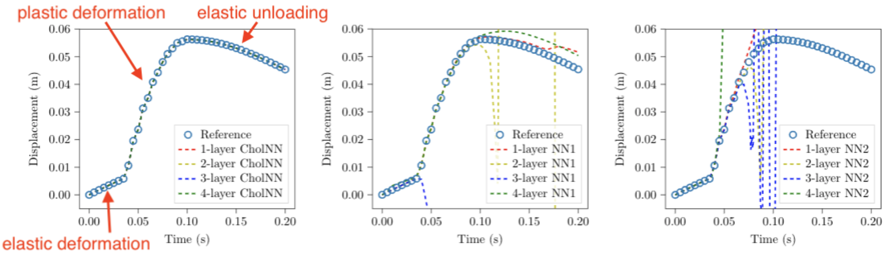
$$\mathsf{L}_{\boldsymbol{\theta}} = \begin{bmatrix} L_{1111} & & & & & \\ L_{2211} & L_{2222} & & & & \\ L_{3311} & L_{3322} & L_{3333} & & & \\ & & & L_{2323} & & \\ & & & & L_{1313} & \\ & & & & & L_{1212} \end{bmatrix}$$

- Weak convexity: $\mathsf{L}_{\boldsymbol{\theta}}\mathsf{L}_{\boldsymbol{\theta}}^T \succ 0$
- Time consistency: $\boldsymbol{\sigma}^{n+1} \to \boldsymbol{\sigma}^n$ when $\boldsymbol{\epsilon}^{n+1} \to \boldsymbol{\epsilon}^n$

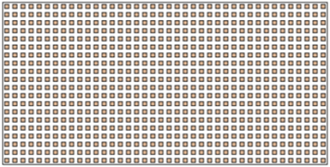
Modeling Elasto-plasticity

- Comparison of different neural network architectures

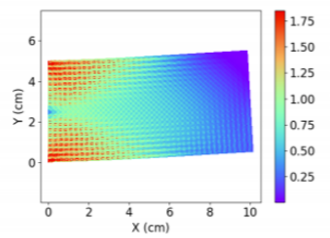
$$\boldsymbol{\sigma}^{n+1} = \mathsf{L}_\theta(\epsilon^{n+1}, \epsilon^n, \boldsymbol{\sigma}^n)\mathsf{L}_\theta(\epsilon^{n+1}, \epsilon^n, \boldsymbol{\sigma}^n)^T(\epsilon^{n+1} - \epsilon^n) + \boldsymbol{\sigma}^n$$
$$\boldsymbol{\sigma}^{n+1} = \mathsf{NN}_\theta(\epsilon^{n+1}, \epsilon^n, \boldsymbol{\sigma}^n)$$
$$\boldsymbol{\sigma}^{n+1} = \mathsf{NN}_\theta(\epsilon^{n+1}, \epsilon^n, \boldsymbol{\sigma}^n) + \boldsymbol{\sigma}^n$$



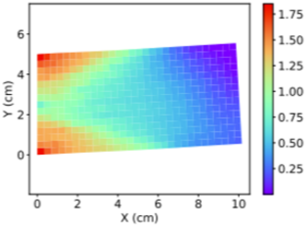
Modeling Elasto-plasticity: Multi-scale



Fiber Reinforced Thin Plate



Reference von Mises stress



SPD-NN

Other Applications

- Time-Lapse Full-Waveform Inversion for Subsurface Flow;
- Seismic Inversion;
- Viscoelasticity Modeling;
- Seismic Inversion;
- Stochastic Differential Equations;
- Navier Stokes Equations;
- ...

See the following slide for more details:

`https://kailaix.github.io/ADCMESlides/2020_11_17.pdf`

A General Approach to Inverse Modeling