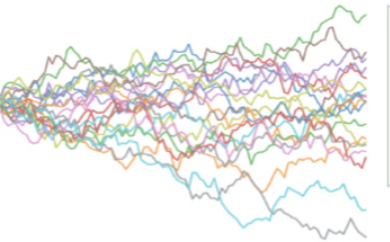
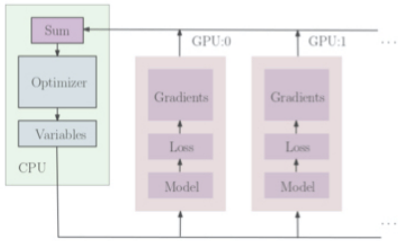
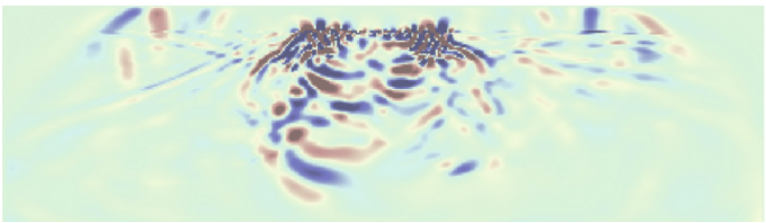
Machine Learning for Inverse Problems in Computational Engineering

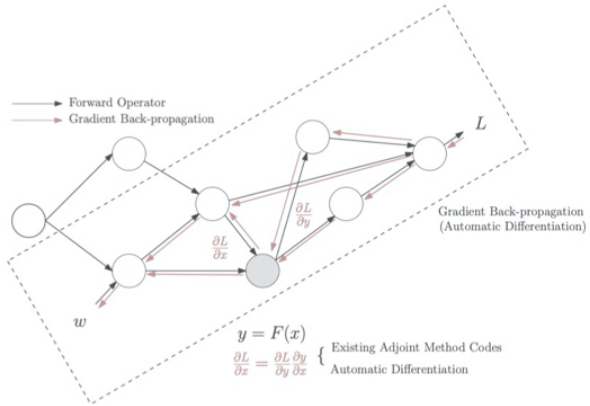
Kailai Xu and Eric Darve

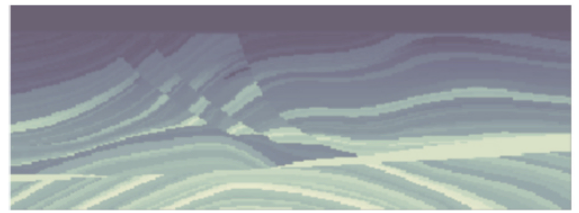
https://github.com/kailaix/ADCME.jl

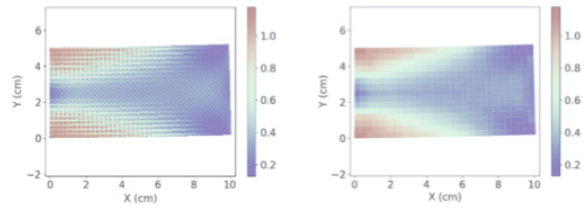








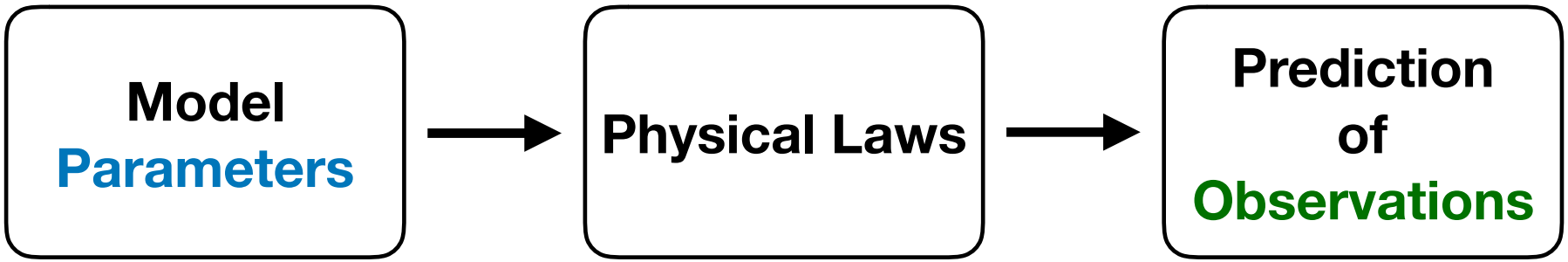




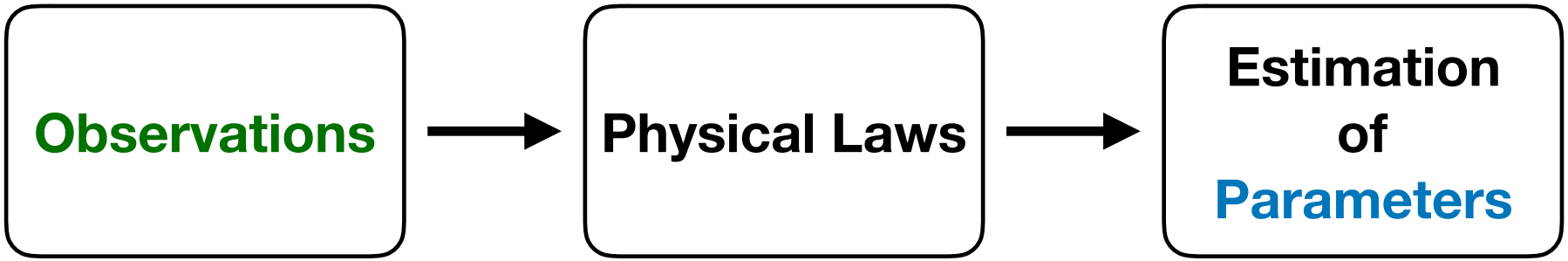
Outline

Inverse Modeling

Forward Problem

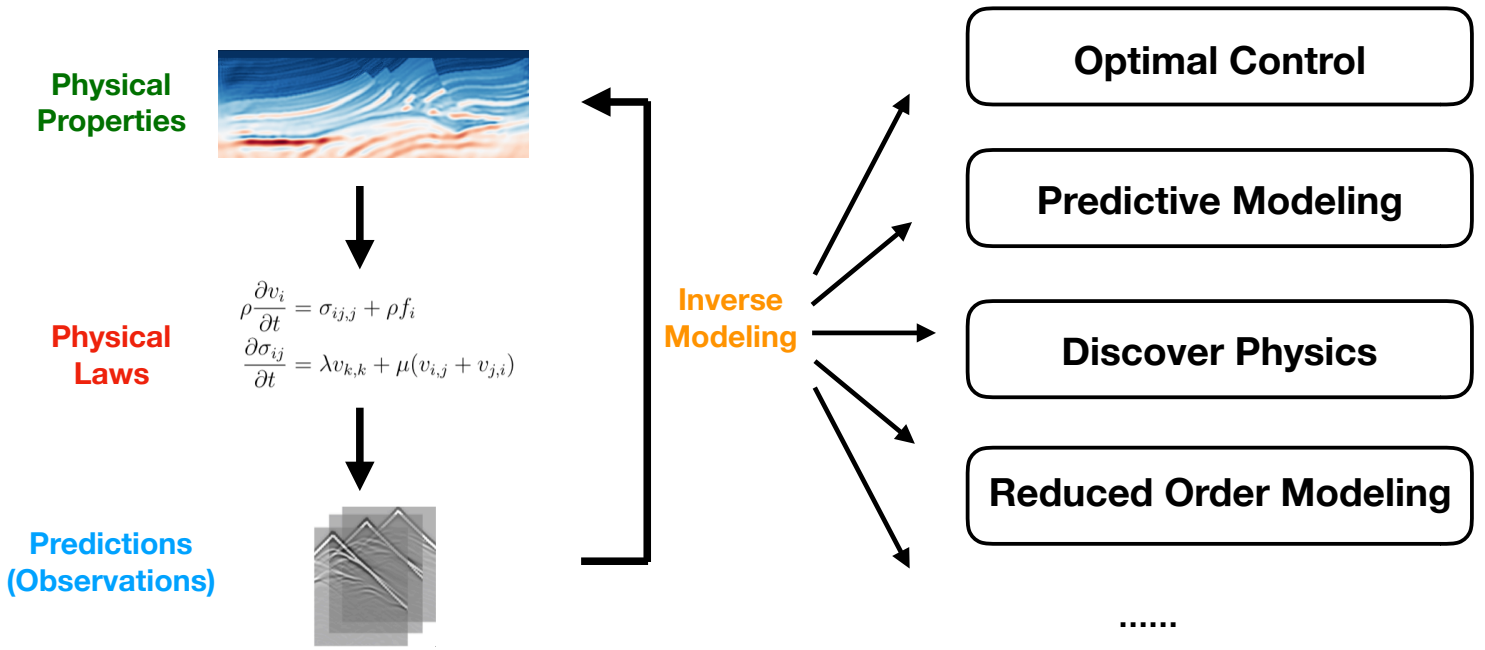


Inverse Problem



Inverse Modeling

- **Inverse modeling** identifies a certain set of parameters or functions with which the outputs of the forward analysis matches the desired result or measurement.
- Many real life engineering problems can be formulated as inverse modeling problems: shape optimization for improving the performance of structures, optimal control of fluid dynamic systems, etc.



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.,

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

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.
- ...

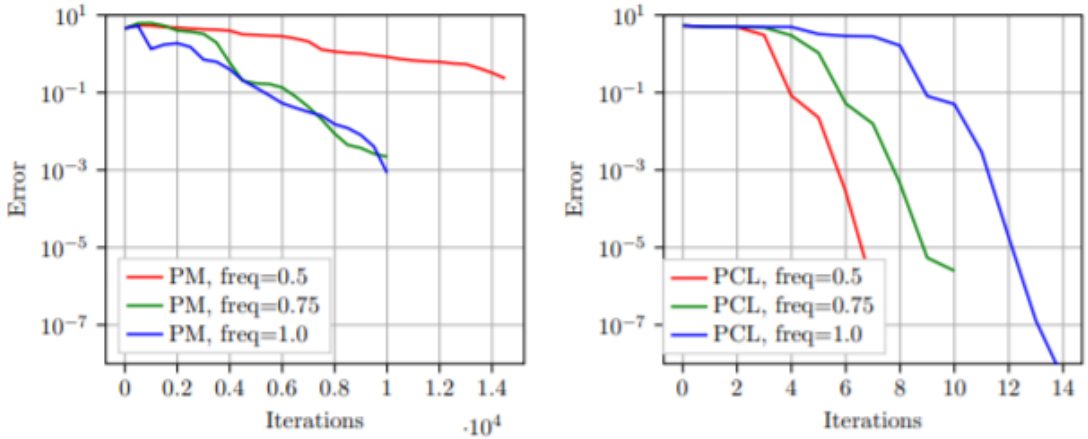
The candidate solution space is infinite dimensional.

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;

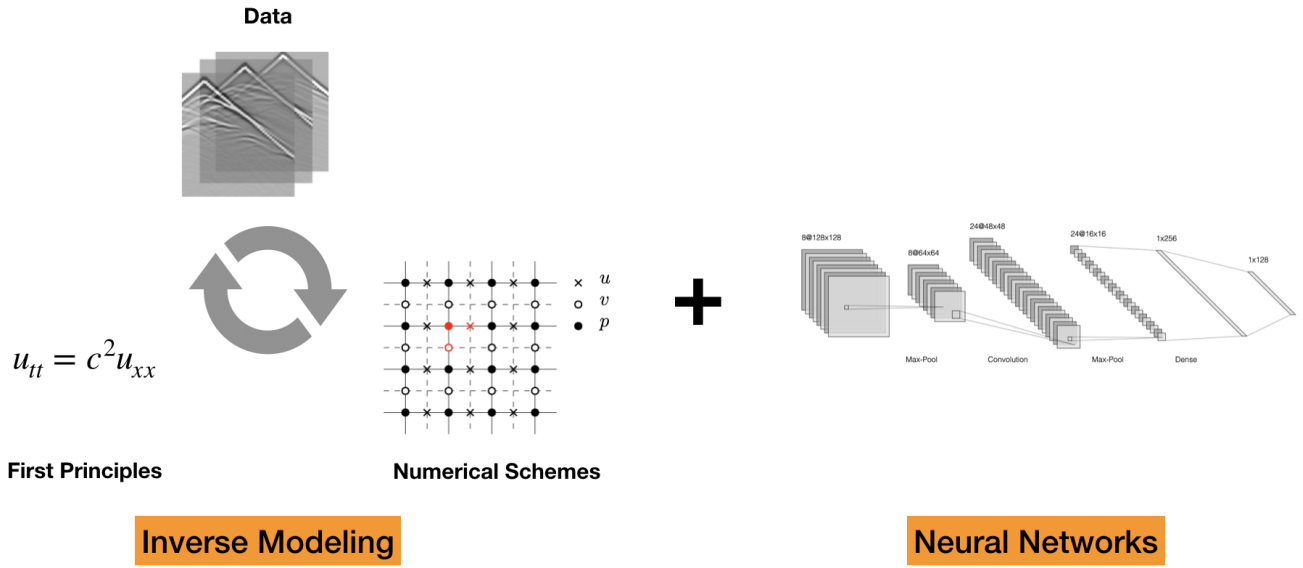


  - High dimensional optimization problem; both θ and u_h are variables.

Machine Learning for Computational Engineering

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

- Deep neural networks exhibit capability of approximating high dimensional and complicated functions.
- **Machine Learning for Computational Engineering**: the unknown function is approximated by a deep neural network, and the physical constraints are enforced by numerical schemes.
- Satisfy the physics to the largest extent.

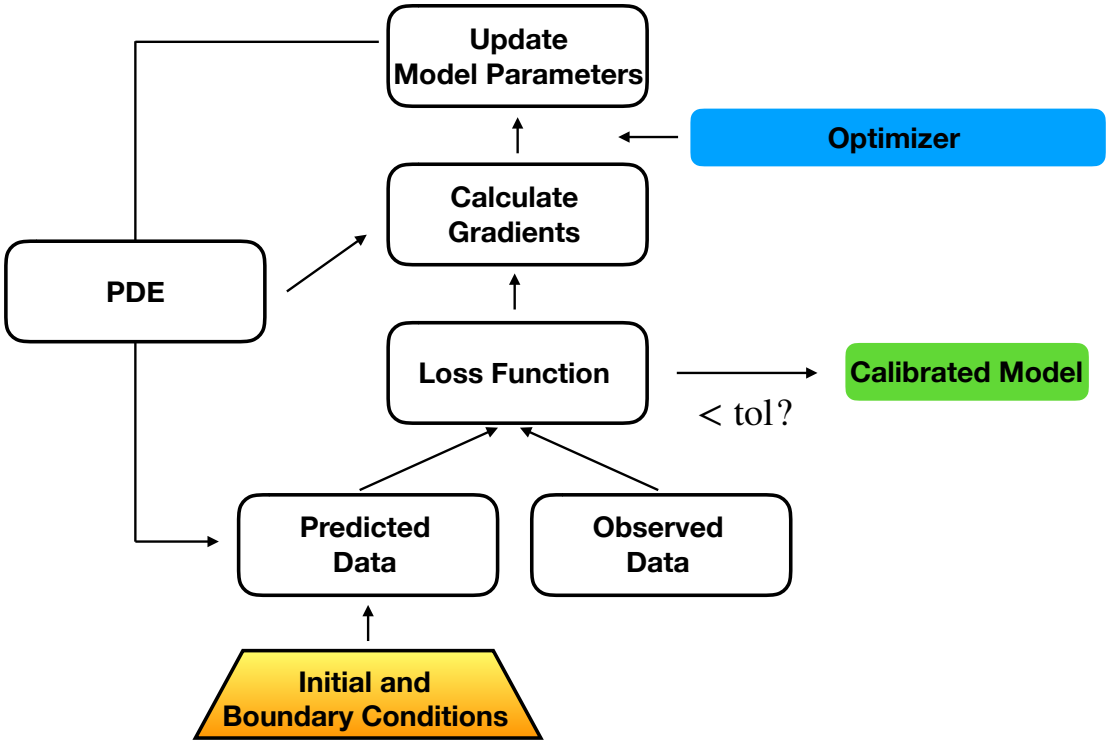


Gradient Based Optimization

$$\min_{\theta} L_h(u_h) \quad \text{s.t. } F_h(\theta, u_h) = 0 \tag{1}$$

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

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



Outline

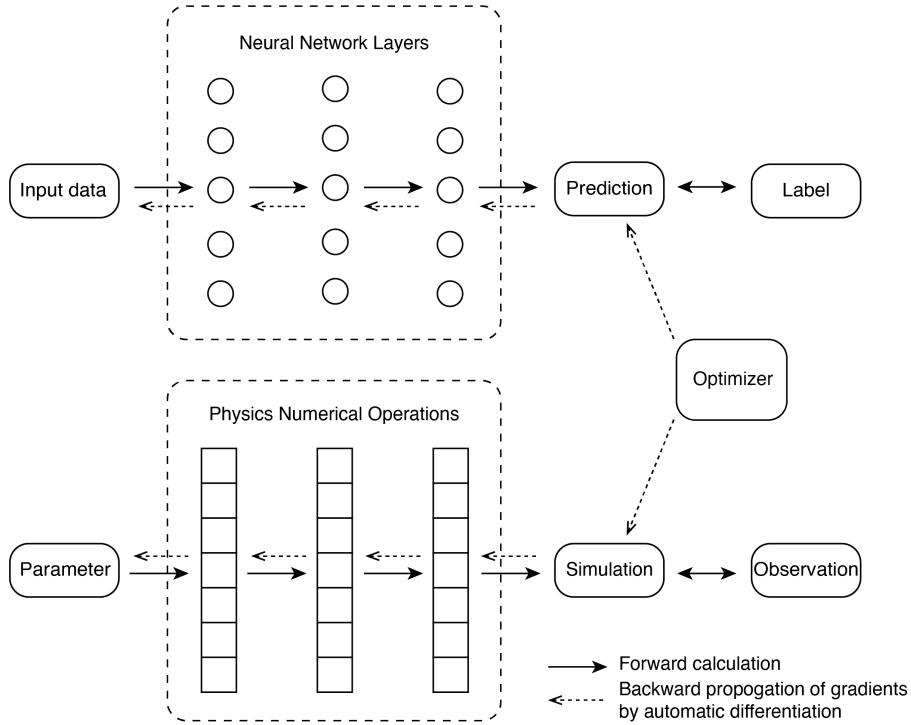
Automatic Differentiation

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

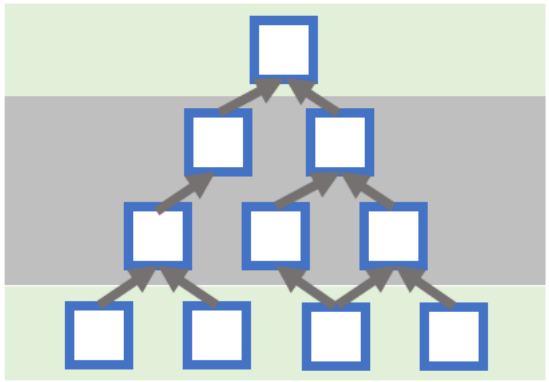
- Deep learning (and many other machine learning techniques) and numerical schemes share the same computational model: composition of individual operators.

Mathematical Fact

Back-propagation
||
Reverse-mode
Automatic Differentiation
||
Discrete
Adjoint-State Method



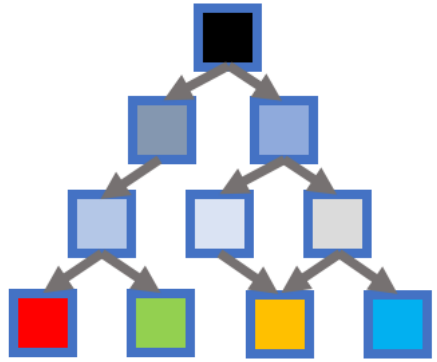
Automatic Differentiation: Forward-mode and Reverse-mode



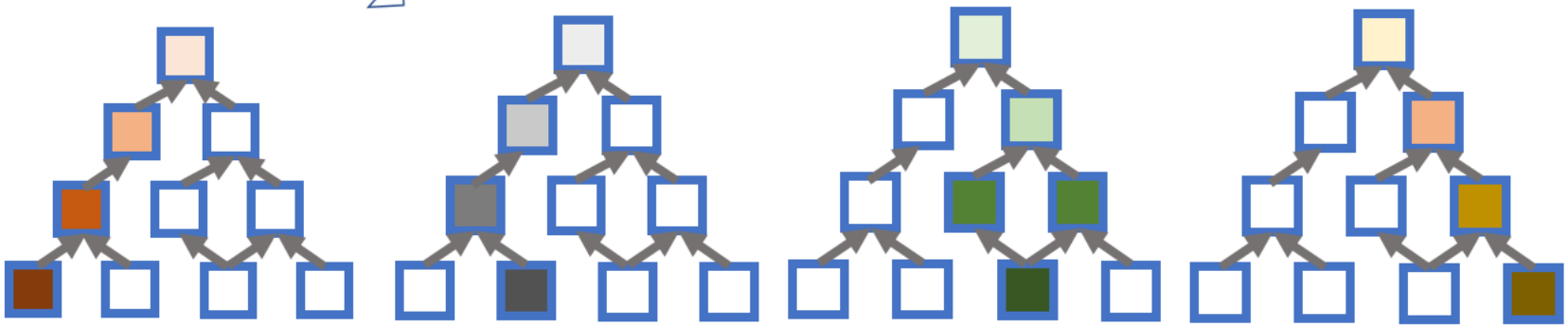
Loss

Intermediate Values

Inputs



Reverse Mode Automatic Differentiation



Forward Mode Automatic Differentiation

What is the Appropriate Model for Inverse Problems?

- In general, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$

Mode	Suitable for ...	Complexity[1]	Application
Forward	$m \gg n$	$\leq 2.5\ \mathrm{OPS}(f(x))$	UQ
Reverse	$m \ll n$	$\leq 4\ \mathrm{OPS}(f(x))$	Inverse Modeling

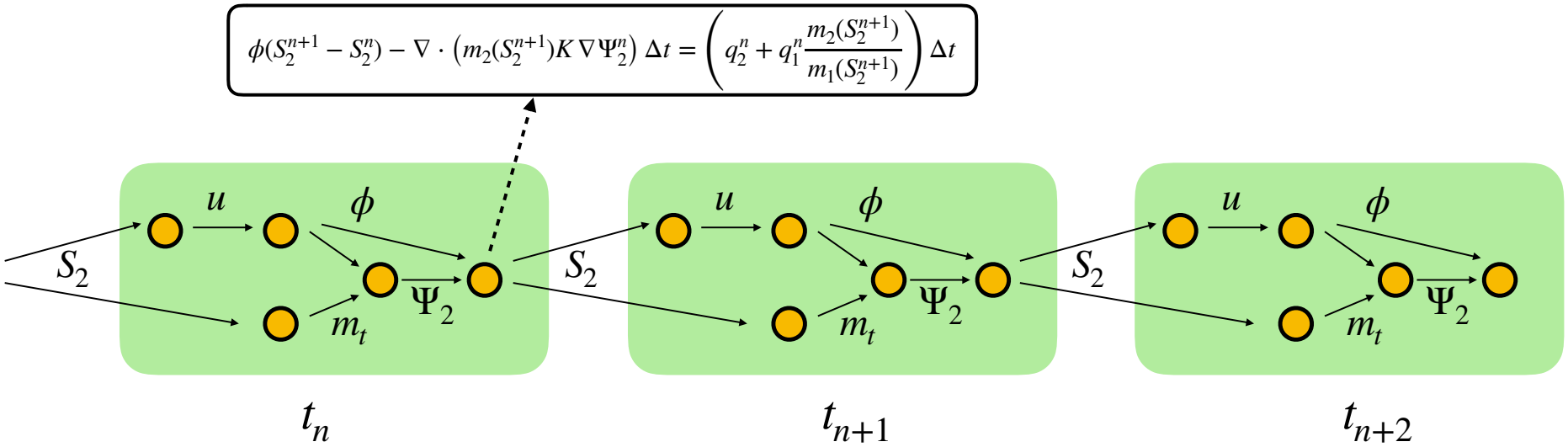
- There are also many other interesting topics
  - Mixed mode AD: many-to-many mappings.
  - Computing sparse Jacobian matrices using AD by exploiting sparse structures.

Margossian CC. A review of automatic differentiation and its efficient implementation. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2019 Jul;9(4):e1305.

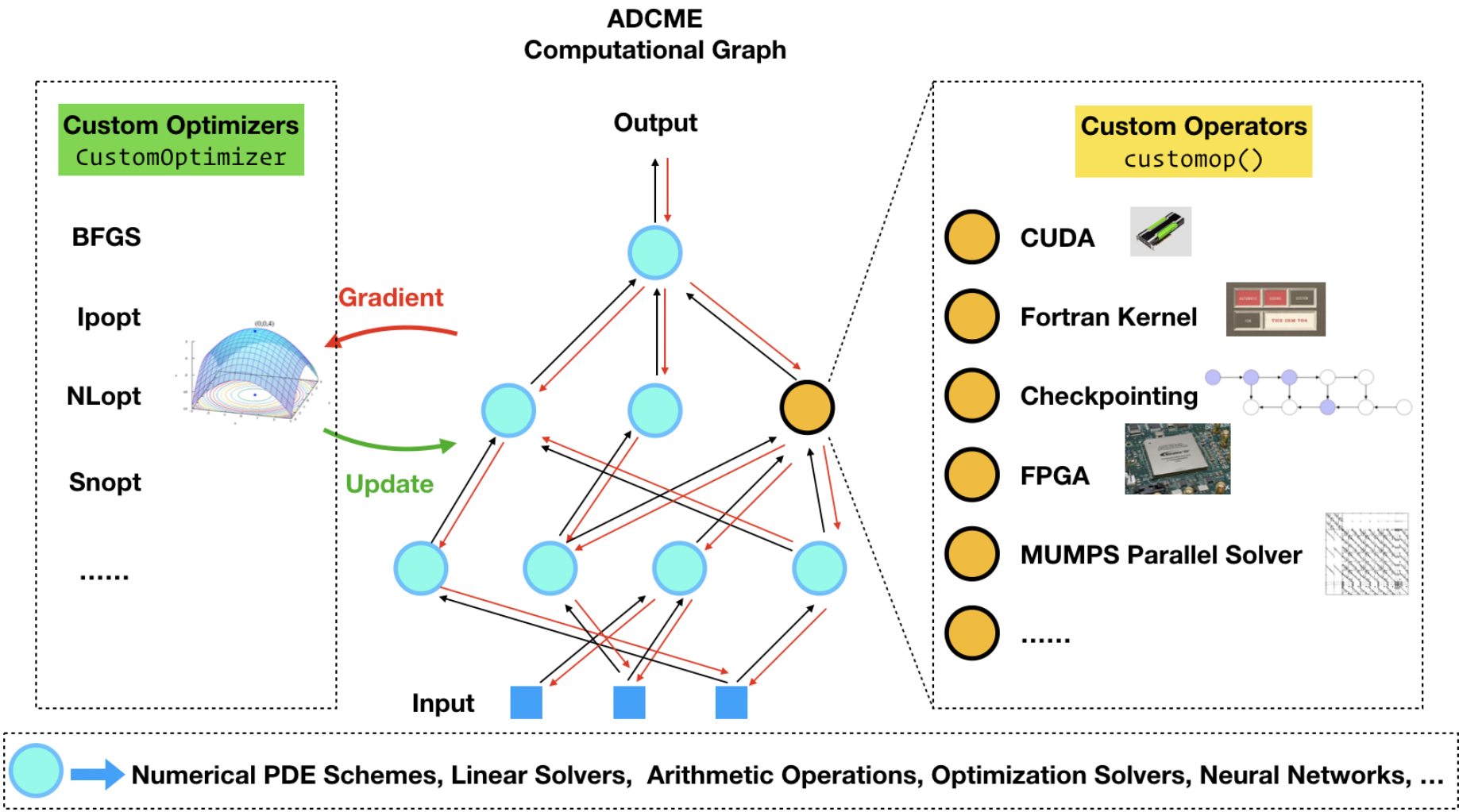
[1] OPS is a metric for complexity in terms of fused-multiply adds.

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

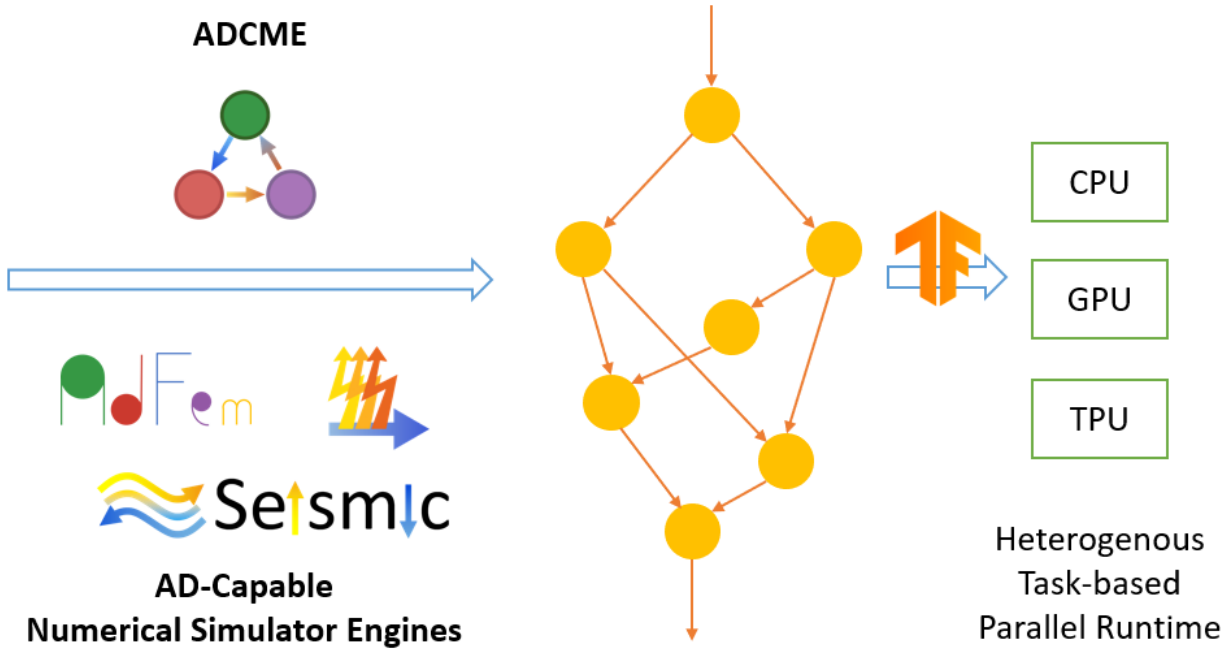


How ADCME works

- 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.

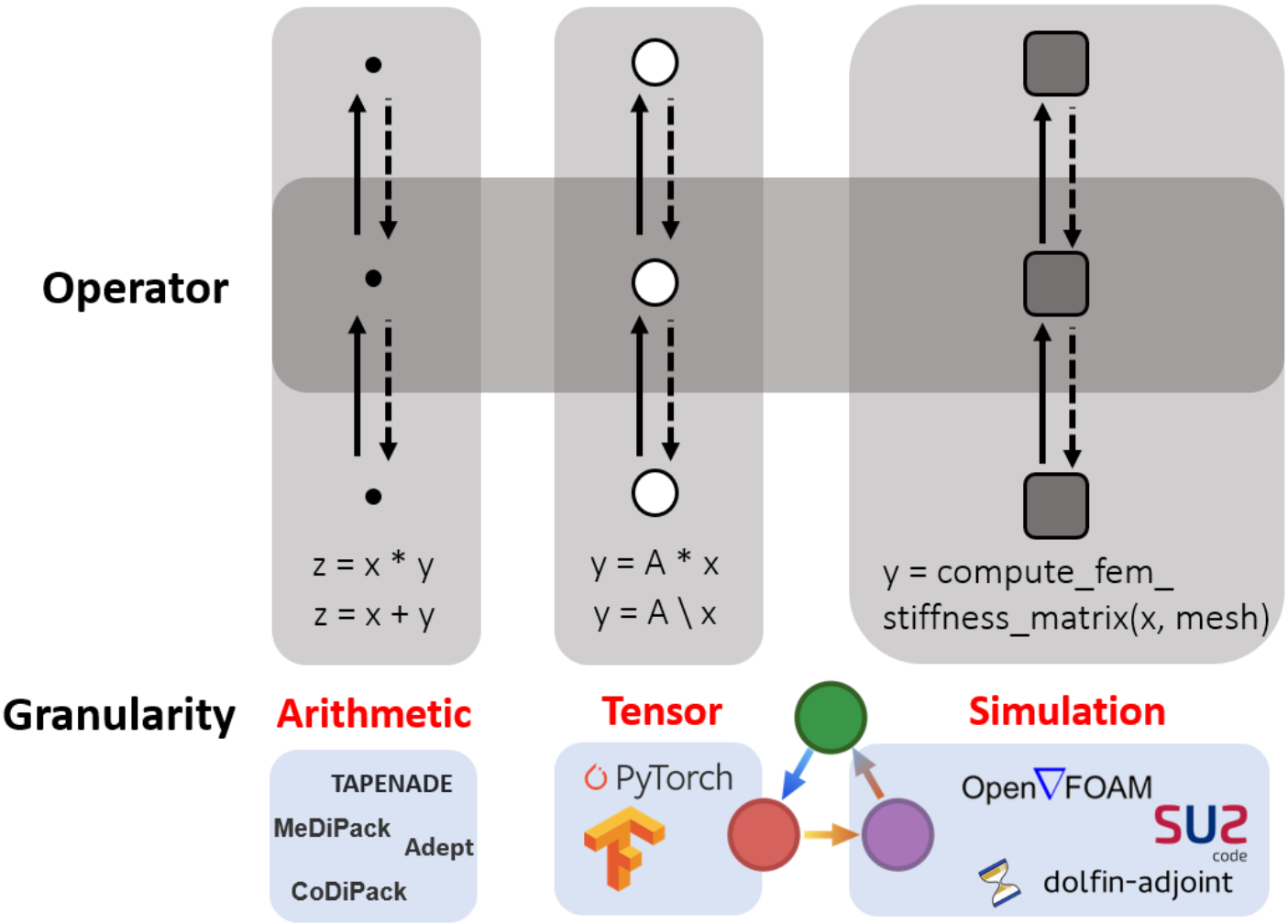
$$\begin{aligned} \operatorname{div} \sigma(u) &= f(x) && x \in \Omega \\ \sigma(u) &= C\varepsilon(u) \\ u(x) &= u_0(x) && x \in \Gamma_u \\ \sigma(x)n(x) &= t(x) && x \in \Gamma_n \end{aligned}$$

```
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
```



Granularity of Automatic Differentiation

- Coarser granularity gives researchers more control over gradient back-propagation.

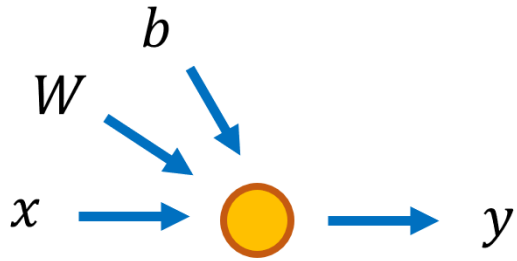


Outline

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.

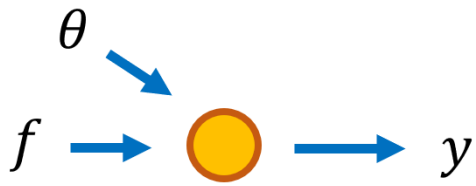
DNN: Explicit



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

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

Numerical Schemes: Implicit, Iterative



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

Example

- Consider a function $f : x \to y$, which is implicitly defined by

$$F(x, y) = x^3 - (y^3 + y) = 0$$

If not using the cubic formula for finding the roots, the forward computation consists of iterative algorithms, such as the Newton's method and bisection method

$y^0 \leftarrow 0$
$k \leftarrow 0$
while $|F(x, y^k)| > \epsilon$ **do**
  $\delta^k \leftarrow F(x, y^k)/F'_y(x, y^k)$
  $y^{k+1} \leftarrow y^k - \delta^k$
  $k \leftarrow k + 1$
end while
Return y^k

$l \leftarrow -M$, $r \leftarrow M$, $m \leftarrow 0$
while $|F(x, m)| > \epsilon$ **do**
  $c \leftarrow \frac{a+b}{2}$
  if $F(x, m) > 0$ **then**
    $a \leftarrow m$
  else
    $b \leftarrow m$
  end if
end while
Return c

Example

- An efficient way to do automatic differentiation is to apply the implicit function theorem. For our example, $F(x, y) = x^3 - (y^3 + y) = 0$; treat y as a function of x and take the derivative on both sides

$$3x^2 - 3y(x)^2 y'(x) - y'(x) = 0 \Rightarrow y'(x) = \frac{3x^2}{3y^2 + 1}$$

The above gradient is exact.

Can we apply the same idea to inverse modeling?

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

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

Physics Constrained Learning for Stiff Problems

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

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

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

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

Theorem

The condition number of A_λ *is*

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

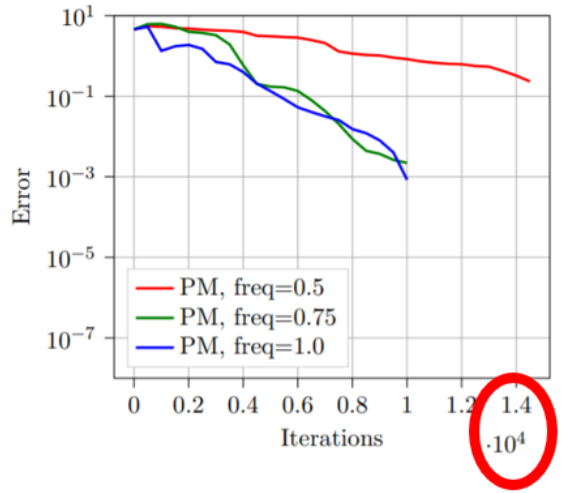
and therefore, the condition number of the unconstrained optimization problem from the penalty method is equal to the the square of the condition number of the PCL asymptotically.

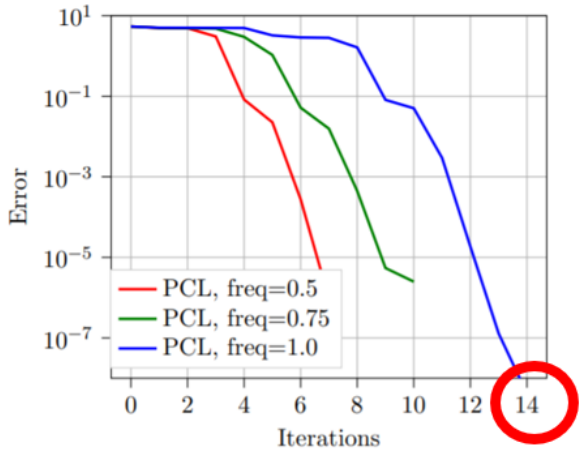
Physics Constrained Learning for Stiff Problems

Parameter Inverse Problem

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

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

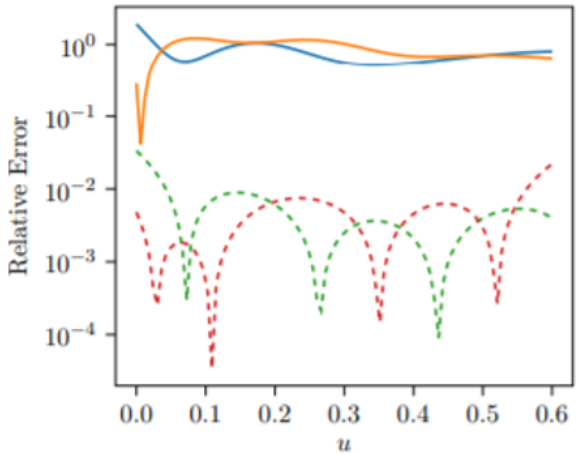




Approximate Unknown Functions using DNNs

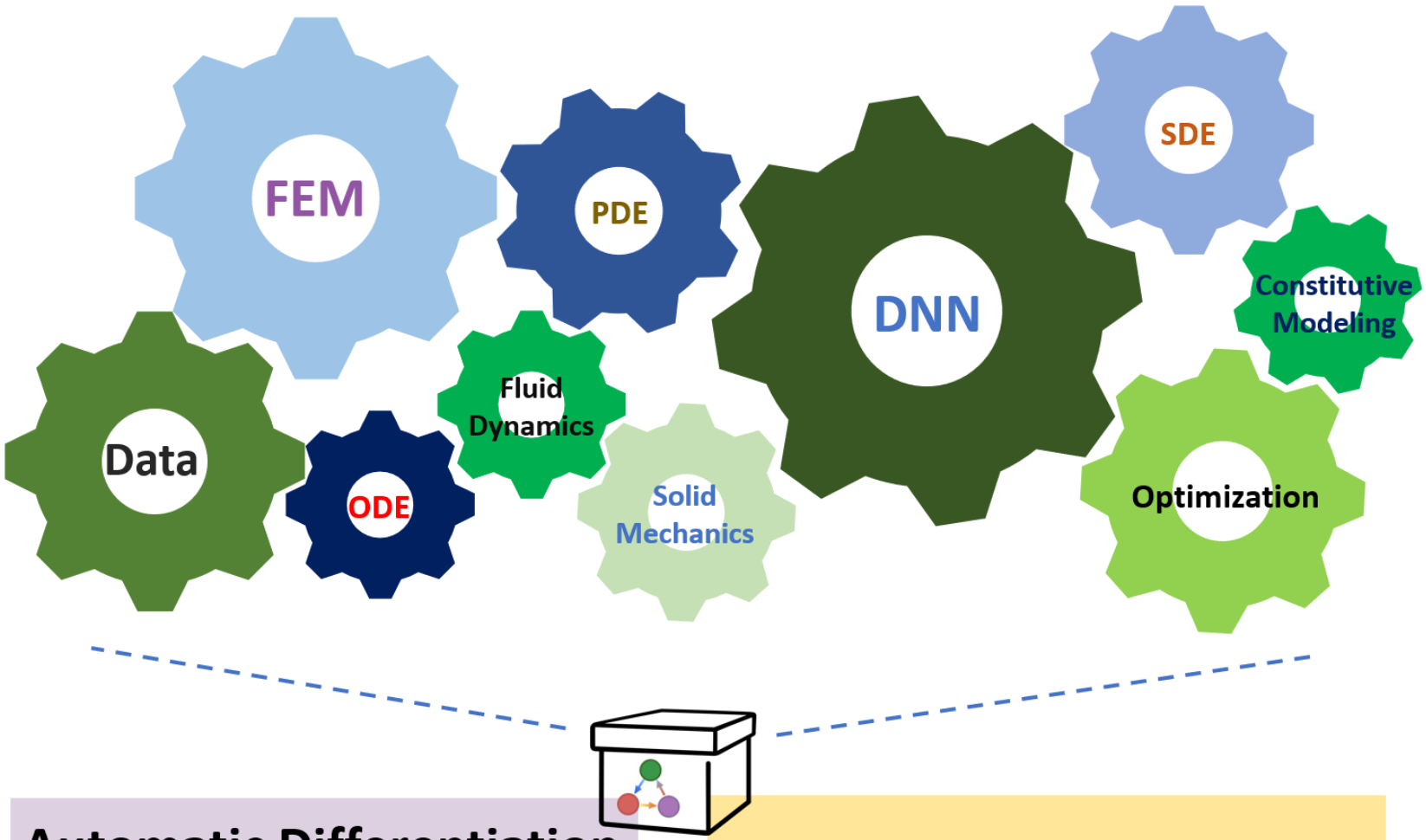
$$-\nabla \cdot (\boldsymbol{f}(u)\nabla u) = h(\boldsymbol{x})$$

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



PM, $\frac{|f_1(u;\theta)-f_1(u)|}{f_1(u)}$
PM, $\frac{|f_2(u;\theta)-f_2(u)|}{f_2(u)}$
PCL, $\frac{|f_1(u;\theta)-f_1(u)|}{f_1(u)}$
PCL, $\frac{|f_2(u;\theta)-f_2(u)|}{f_2(u)}$

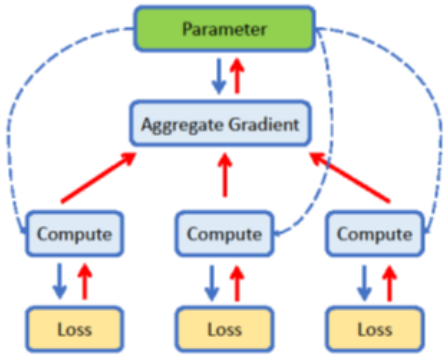
PCL: Backbone of the ADCME Infrastructure



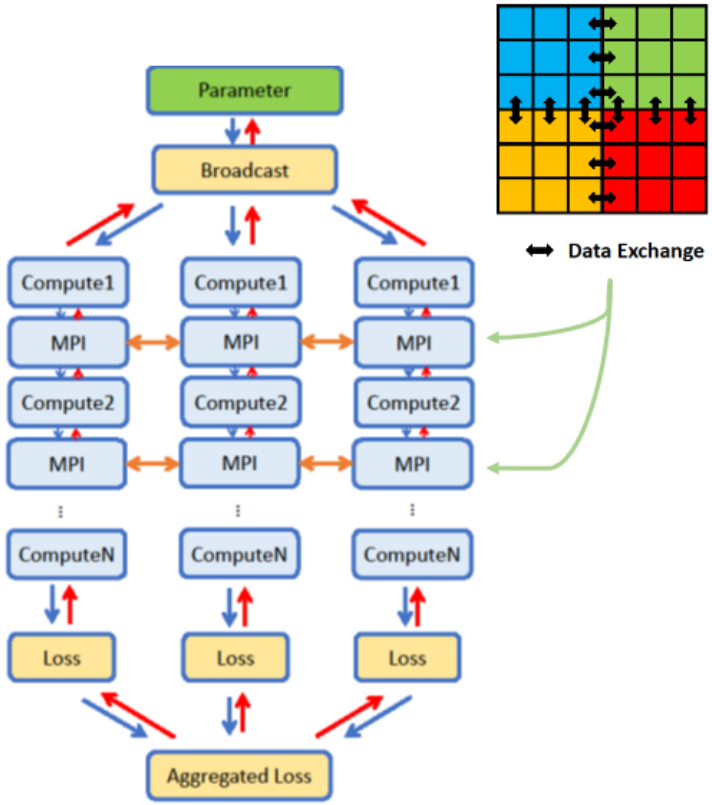
Outline

Parallel Computing

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



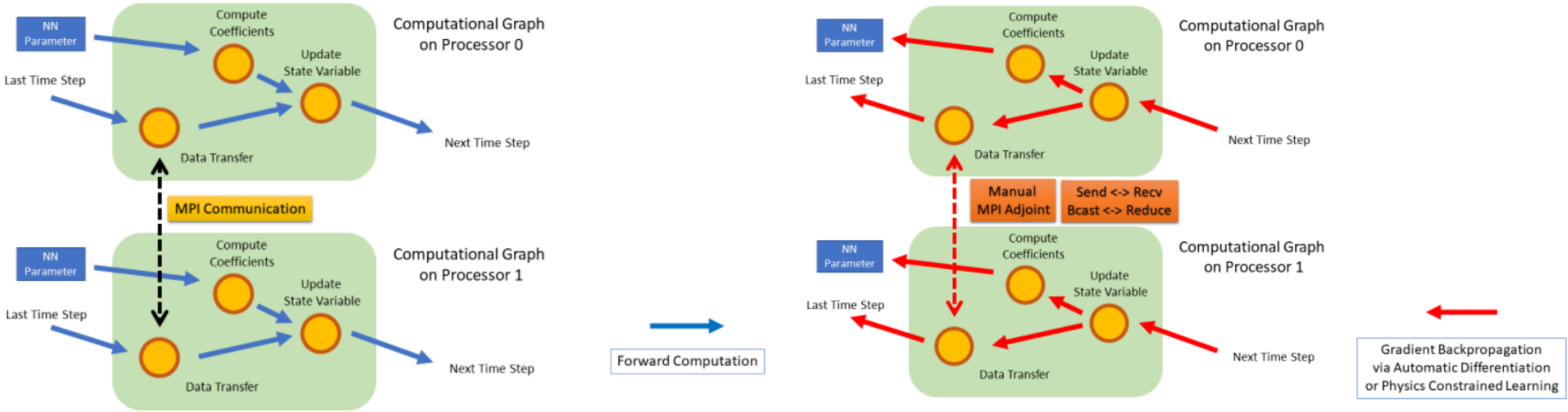
Deep Learning Data/Model Parallelism



Scientific Computing Mixed Parallelism

Distributed Optimization

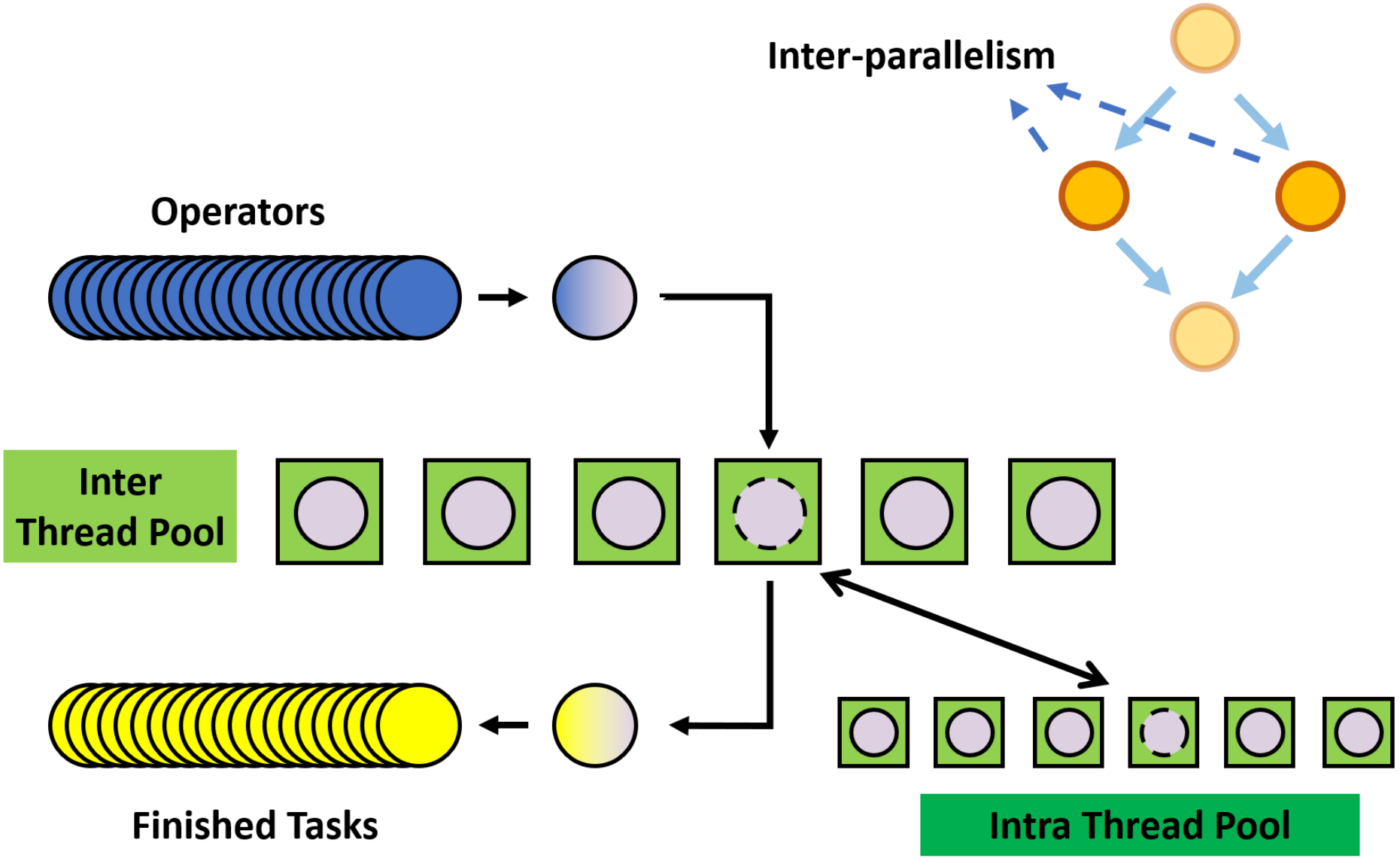
- ADCME also supports MPI-based distributed computing. The parallel model is designed specially for scientific computing.



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

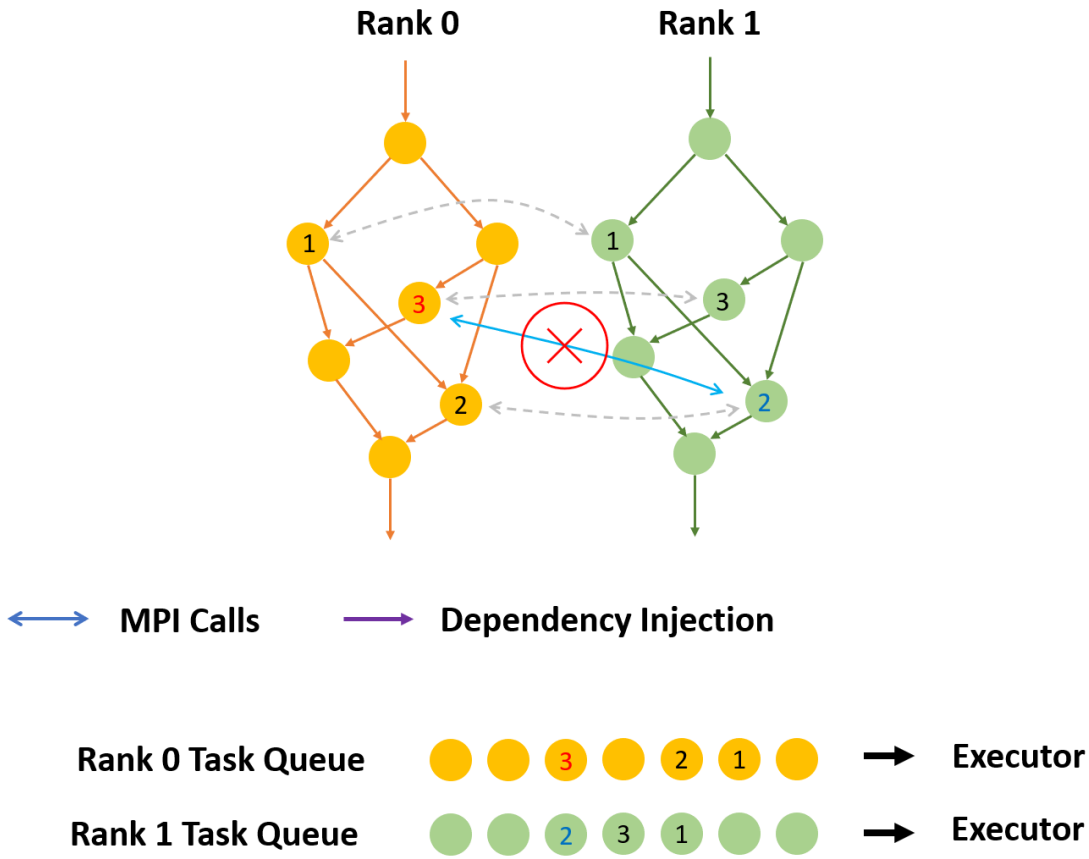
  `mpi_bcast`, `mpi_sum`, `mpi_send`, `mpi_recv`, `mpi_halo_exchange`, ...

Parallel Computing Model for a Single Processor



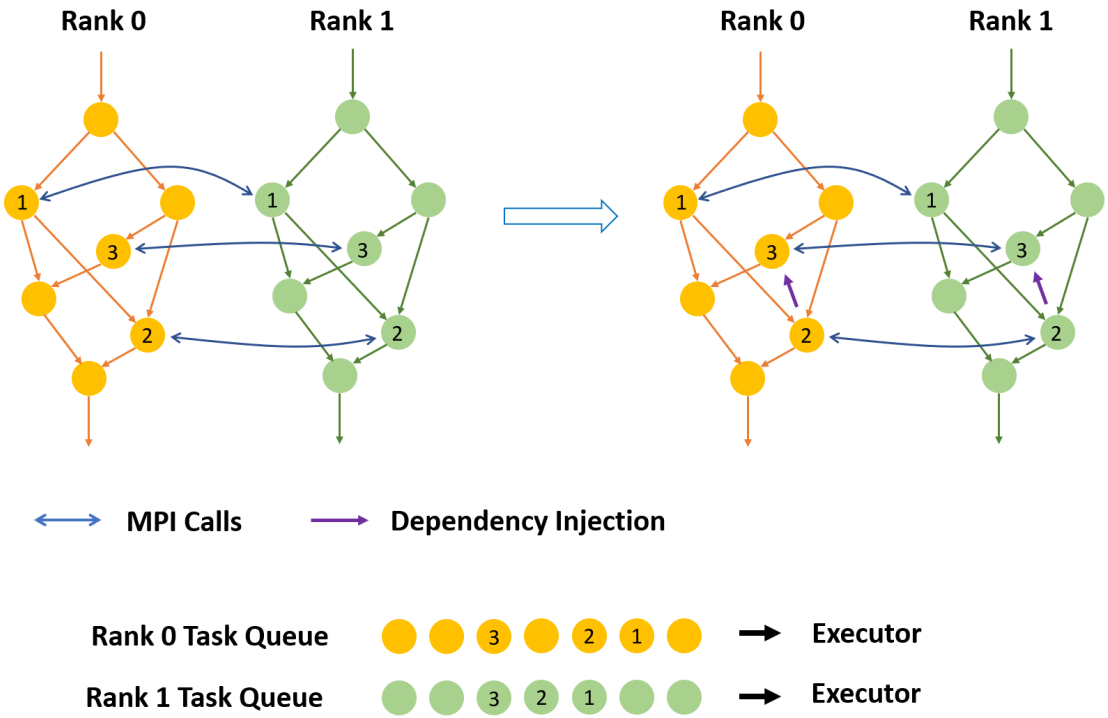
Mismatched MPI Calls in Hybrid Parallel Computing

- Without any additional synchronization mechanisms, we may encounter mismatched MPI calls.



Hybrid Parallel Computing

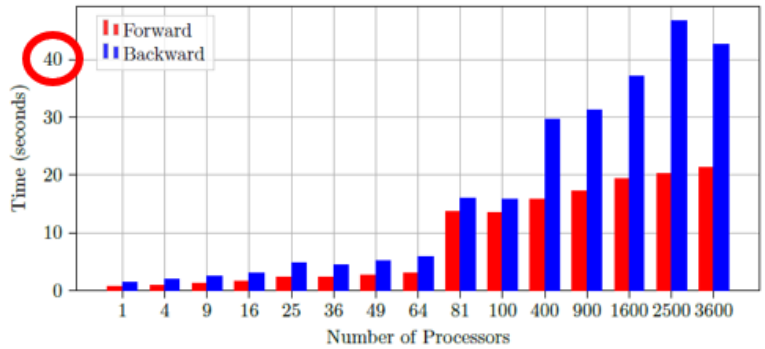
We use **dependency injection** techniques to ensure consistency.

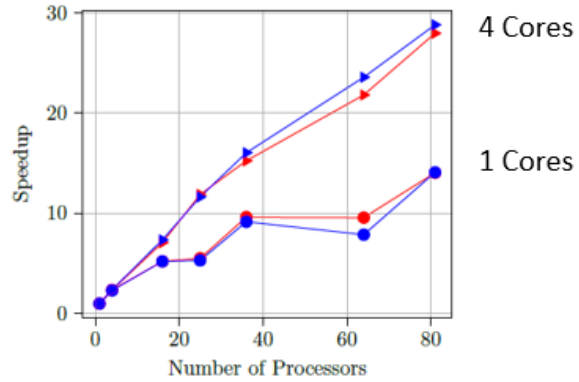


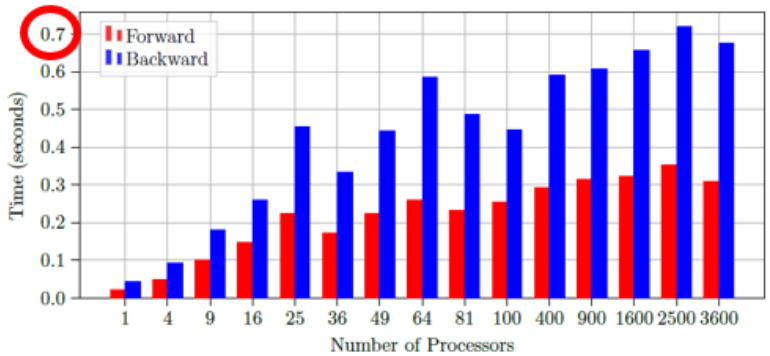
Interoperability with Hypre

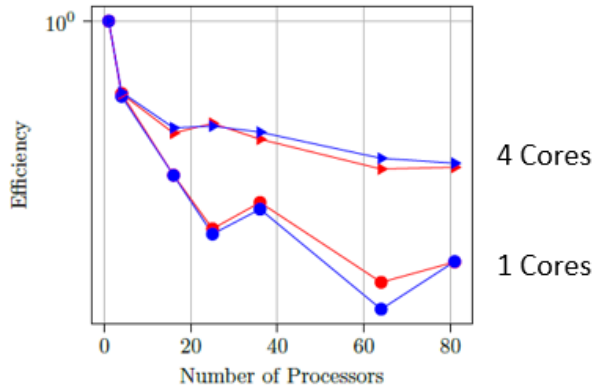
$$\nabla \cdot (\mathrm{NN}_\theta(x)\nabla u(x)) = f(x) \qquad x \in \Omega$$
$$u(x) = 0 \qquad x \in \partial\Omega$$

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









Weak Scalability **Strong Scalability**

Outline

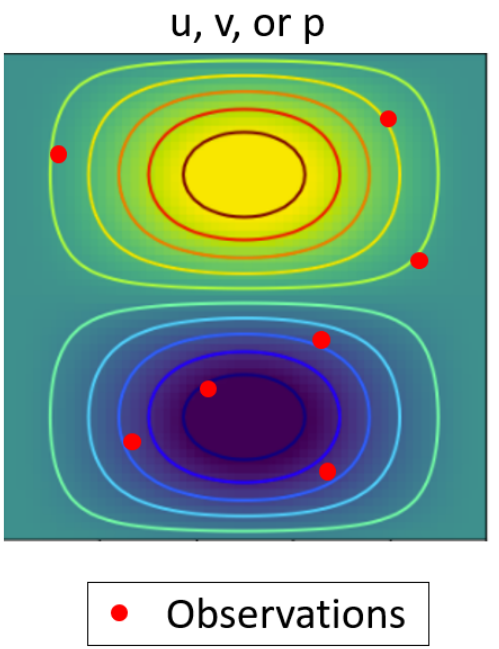
Inverse Modeling of the Stokes Equation

- The governing equation for the Stokes problem

$$\begin{aligned} -\nu(\mathrm{x})\Delta \mathrm{u} + \nabla p &= \mathrm{f} && \text{in } \Omega \\ \nabla \cdot \mathrm{u} &= 0 && \text{in } \Omega \\ \mathrm{u} &= 0 && \text{on } \partial\Omega \end{aligned}$$

- The weak form is given by

$$\begin{aligned} (\nu(\mathrm{x})\nabla u, \nabla v) - (p, \nabla \cdot v) &= (f, v) \\ (\nabla \cdot u, q) &= 0 \end{aligned}$$



Inverse Modeling of the Stokes Equation

```
xy = gauss_nodes(m, n, h)
nu = squeeze(fc(xy, [20,20,20,1]))
K = nu*constant(compute_fem_laplace_matrix(m, n, h))
B = constant(compute_interaction_matrix(m, n, h))
Z = [K -B'
-B spdiag(zeros(size(B,1)))]

# Impose boundary conditions
bd = bcnode("all", m, n, h)
bd = [bd; bd .+ (m+1)*(n+1); ((1:m) .+ 2(m+1)*(n+1))]
Z, _ = fem_impose_Dirichlet_boundary_condition1(Z, bd, m, n, h)

# Calculate the source term
F1 = eval_f_on_gauss_pts(f1func, m, n, h)
F2 = eval_f_on_gauss_pts(f2func, m, n, h)
F = compute_fem_source_term(F1, F2, m, n, h)
rhs = [F;zeros(m*n)]
rhs[bd] .= 0.0
```

Inverse Modeling of the Stokes Equation

- The distinguished feature compared to traditional forward simulation programs: the model output is differentiable with respect to model parameters!

```
loss = sum((sol[idx] - observation[idx])^2)
g = gradients(loss, get_collection())
```

- Optimization with a one-liner:

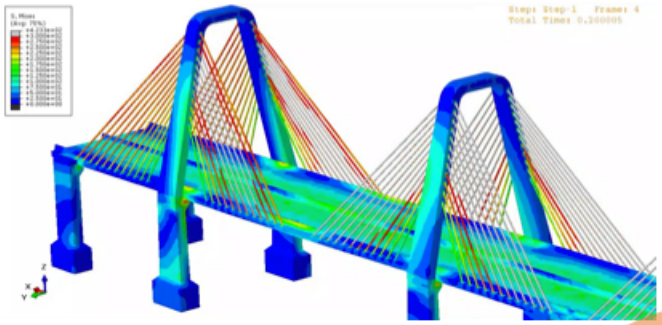
```
BFGS!(sess, loss)
```

ADCME/AdFem

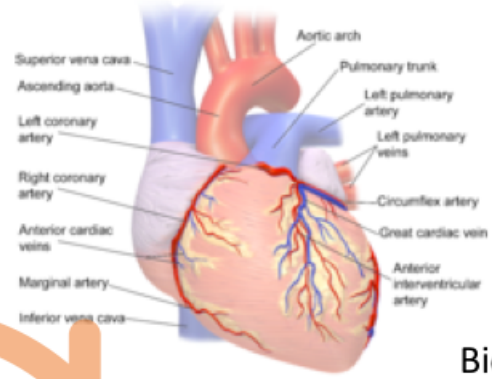
Simulation Program

Outline

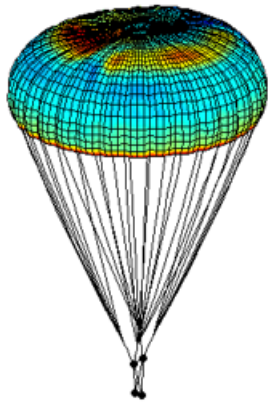
Constitutive Modeling



Civil Engineering



Biology

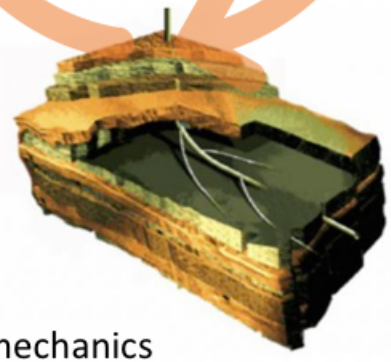


Aeronautics & Astronautics

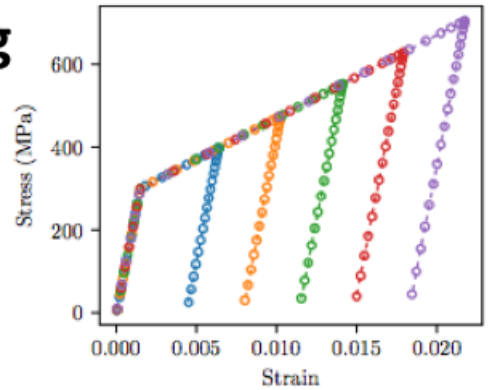
Constitutive Modeling

$$\epsilon_{ij} = \frac{1+\nu}{E}\sigma_{ij} - \frac{\nu}{E}\delta_{ij}\sigma_{kk}$$

$$\sigma + \frac{\eta}{E}\dot{\sigma} =$$



Geomechanics



Theoretical Mechanics

Governing Equations

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

- **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. **??** and Eq. **??**.

Residual Learning using Full-field Data

- Weak form of balance equations of linear momentum

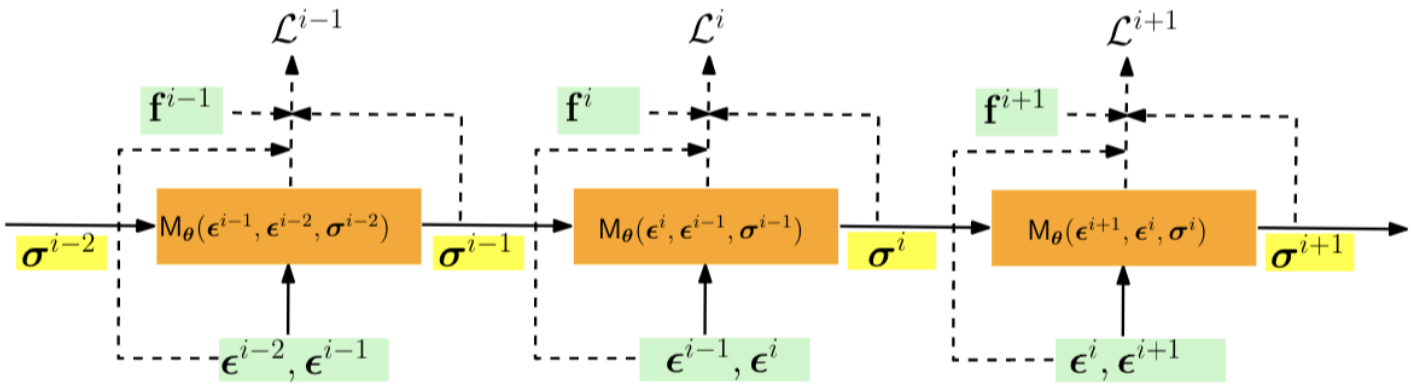
$$P_i(\theta) = \int_V \rho \ddot{u}_i \delta u_i dVt + \int_V \underbrace{\sigma_{ij}(\theta)}_{\text{embedded neural network}} \delta \varepsilon_{ij} dV$$

$$F_i = \int_V \rho b_i \delta u_i dV + \int_{\partial V} t_i \delta u_i dS$$

- Train the neural network by

$$\boxed{L(\theta) = \min_\theta \sum_{i=1}^{N} (P_i(\theta) - F_i)^2}$$

The gradient $\nabla L(\theta)$ is computed via automatic differentiation.



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 $\boldsymbol{\epsilon}^{n+1} \to \boldsymbol{\epsilon}^n$

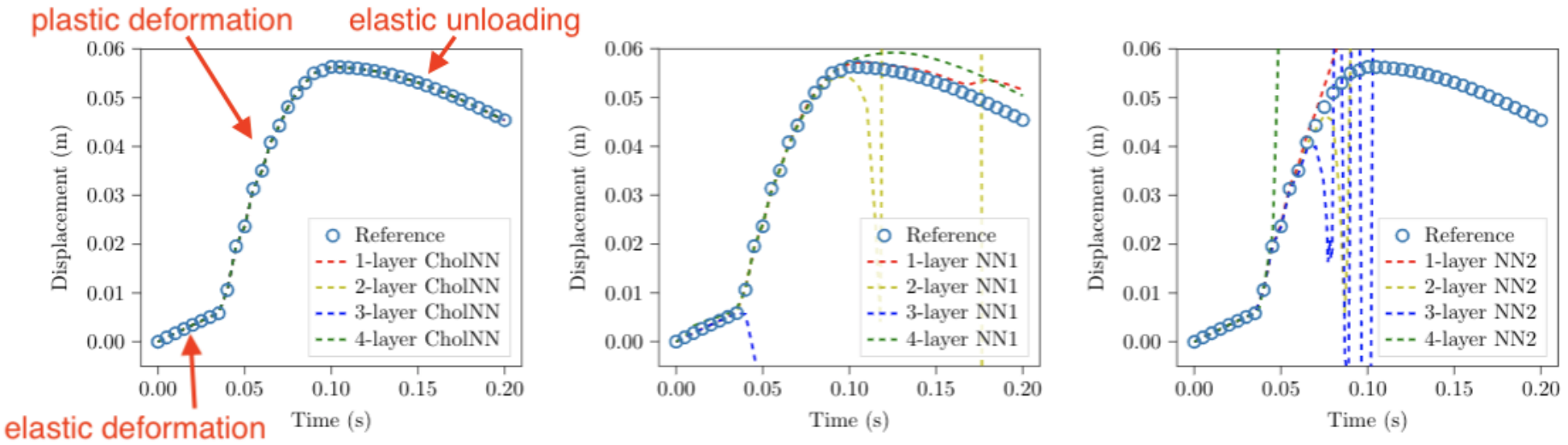
Modeling Elasto-plasticity

- Comparison of different neural network architectures

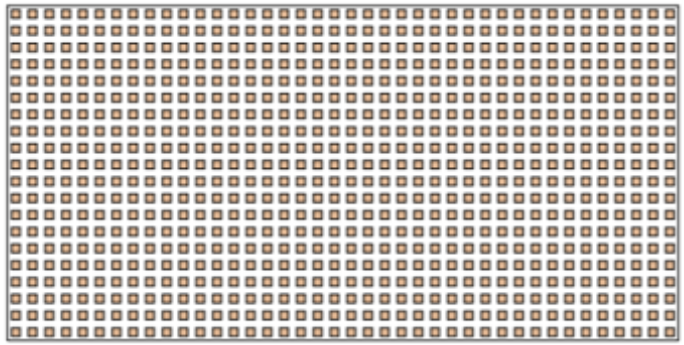
$$\boldsymbol{\sigma}^{n+1} = \mathrm{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^{n}, \boldsymbol{\sigma}^{n})\mathrm{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^{n}, \boldsymbol{\sigma}^{n})^{T}(\boldsymbol{\epsilon}^{n+1} - \boldsymbol{\epsilon}^{n})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

$$\boldsymbol{\sigma}^{n+1} = \mathrm{NN}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^{n}, \boldsymbol{\sigma}^{n})$$

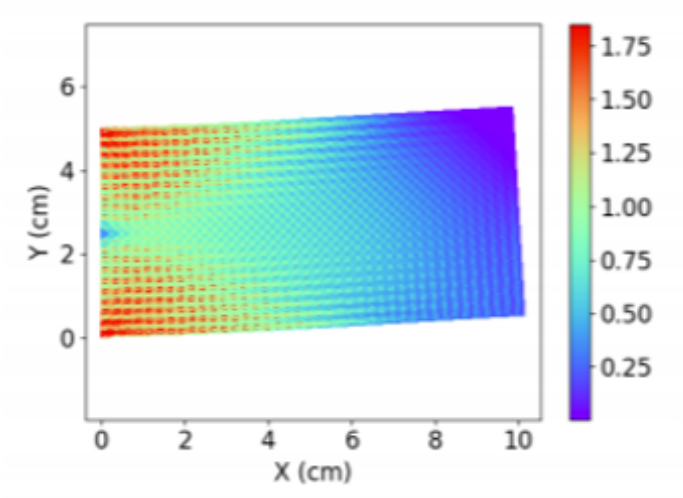
$$\boldsymbol{\sigma}^{n+1} = \mathrm{NN}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^{n}, \boldsymbol{\sigma}^{n}) + \boldsymbol{\sigma}^{n}$$



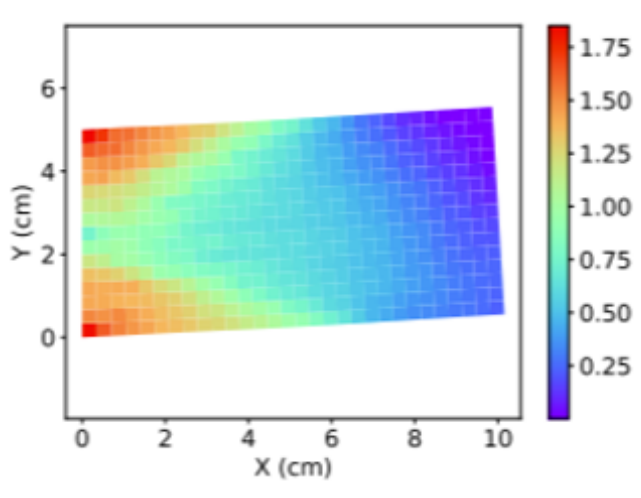
Modeling Elasto-plasticity: Multi-scale



Fiber Reinforced Thin Plate



Reference von Mises stress



SPD-NN

Static Hyperelasticity Problem

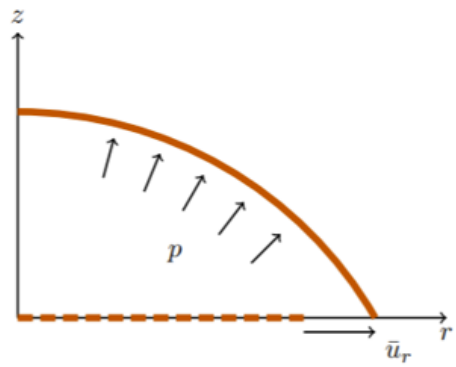
- Consider an axisymmetric Mooney-Rivlin hyperelastic incompressible material with an energy density function

$$W(\lambda_1, \lambda_2, \lambda_3) = \mu(\lambda_1^2 + \lambda_2^2 + \lambda_3^2 - 3) + \alpha(\lambda_1^2\lambda_2^2 + \lambda_2^2\lambda_3^2 + \lambda_3^2\lambda_1^2 - 3)$$
$$J = \lambda_1\lambda_2\lambda_3 = 1$$

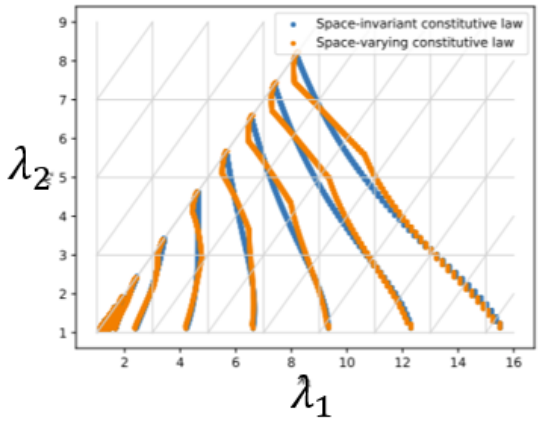
- The constitutive relations is modeled as

$$\mathcal{N}_\theta : (\lambda_1, \lambda_2) \to (P_1, P_2)$$

Here (P_1, P_2) is the stress tensor.

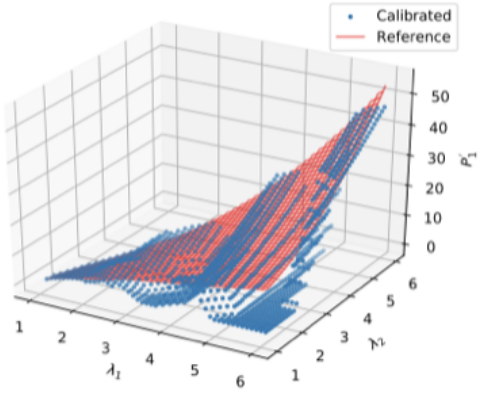


Rubber Membrane

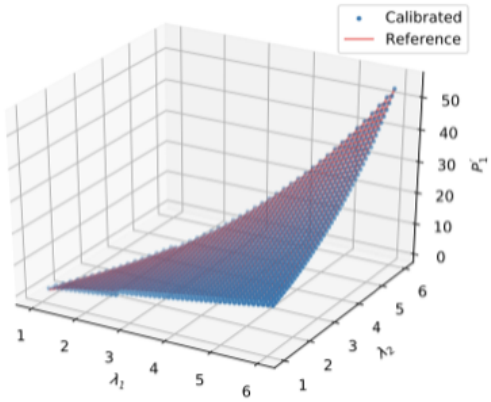


(λ_1, λ_2) Distribution

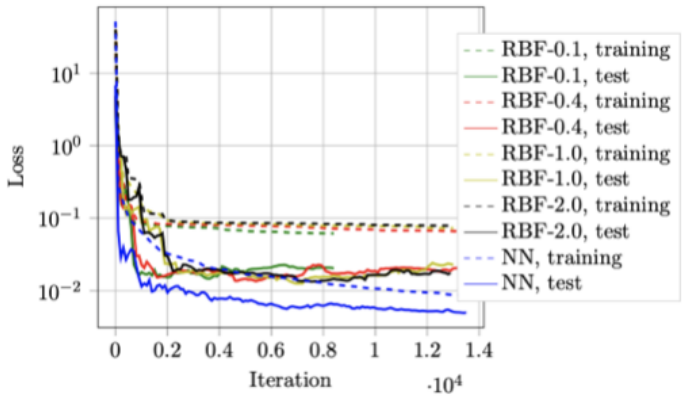
Comparison with Traditional Basis Functions



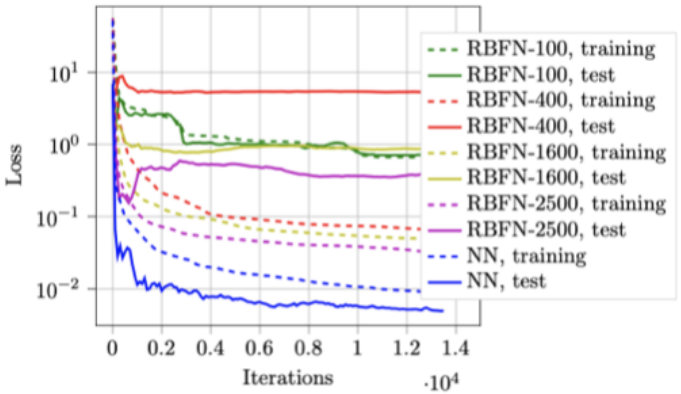
Piecewise Linear



Neural Network



Radial Basis Functions vs. Neural Network



Radial Basis Function Networks vs. Neural Network

Learning Spatially-varying fields

- Hyperelasticity: minimizing the neo-Hookean stored energy

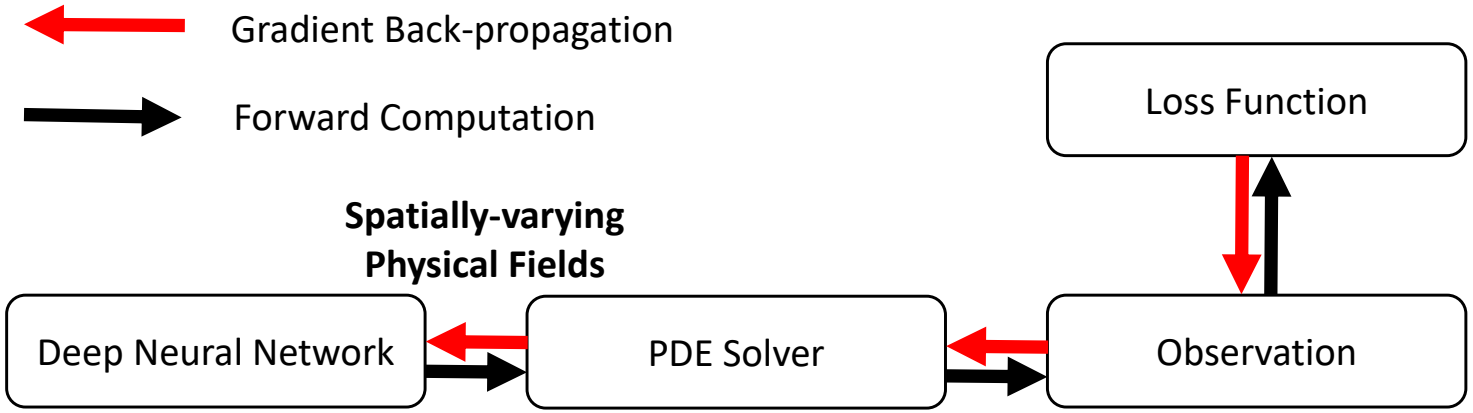
$$\min_u \psi = \frac{\mu}{2}(I_c - 2) - \frac{\mu}{2}\log(J) + \frac{\lambda}{8}\log(J)^2$$

where

$$F = I + \nabla u,\ C = F^T F,\ J = \det(C),\ I_c = \mathrm{trace}(C)$$

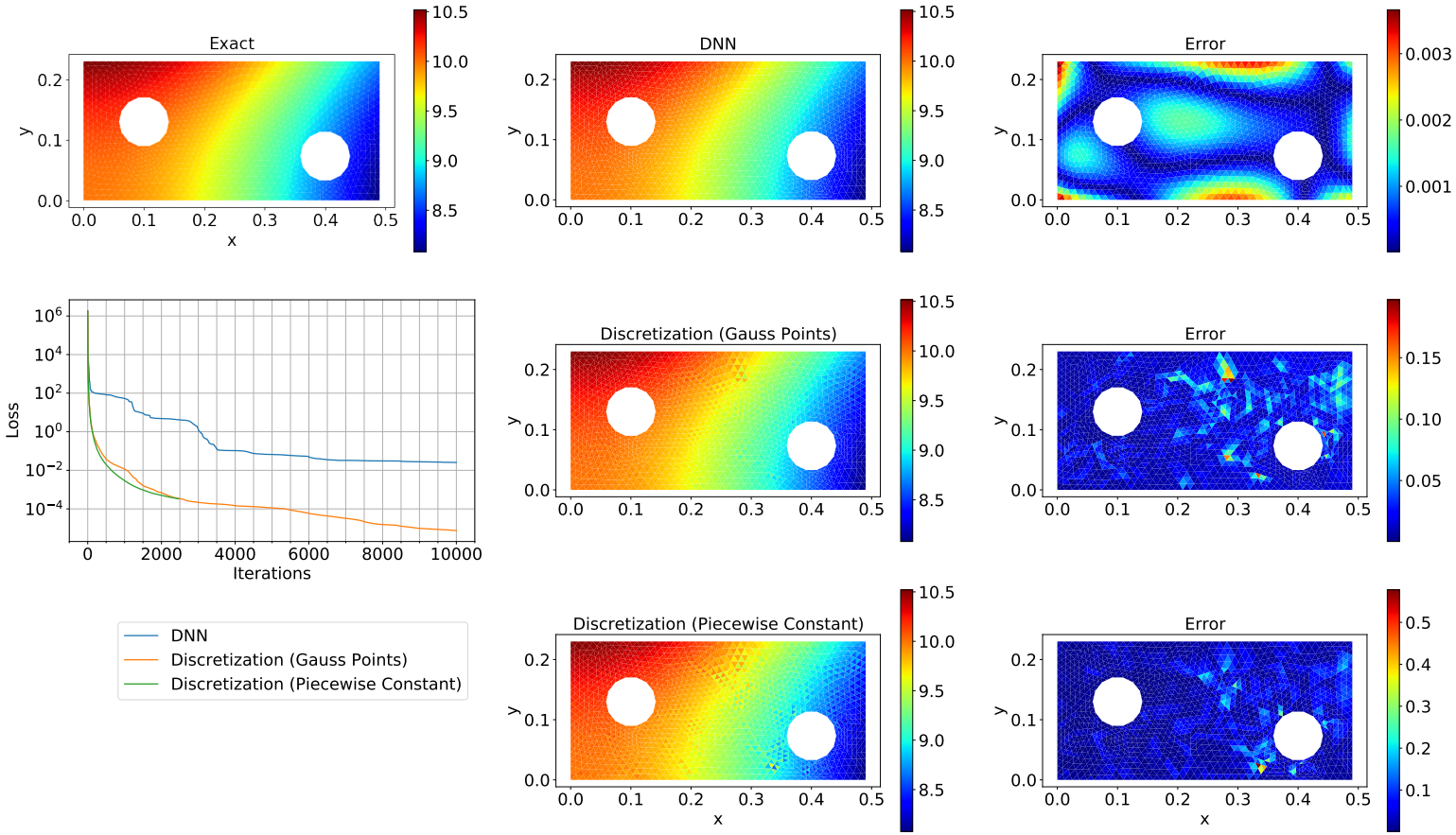
- Lamé parameters

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}$$



Learning Spatially-varying fields

- DNN provides expressive data-driven models and regularization (e.g., spatial dependencies).



Poroelasticity

- Multi-physics Interaction of Coupled Geomechanics and Multi-Phase Flow Equations

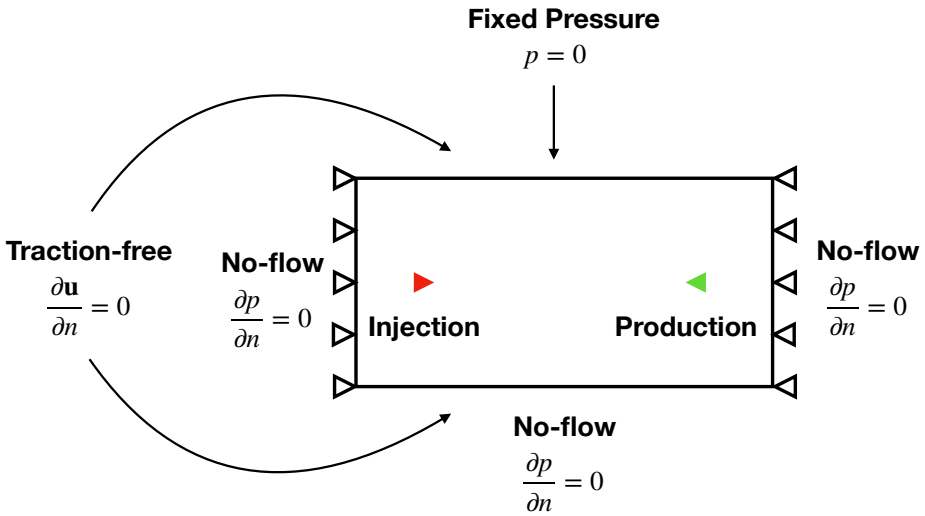
$$\mathrm{div}\boldsymbol{\sigma}(\mathrm{u}) - b\nabla p = 0$$

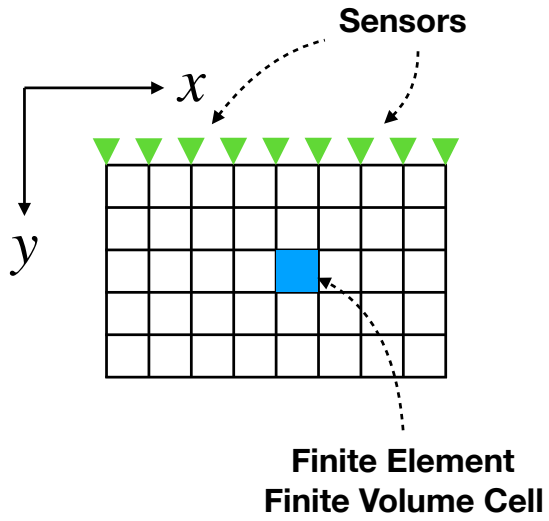
$$\frac{1}{M}\frac{\partial p}{\partial t} + b\frac{\partial \epsilon_v(\mathrm{u})}{\partial t} - \nabla\cdot\left(\frac{k}{B_f\mu}\nabla p\right) = f(x,t)$$

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{\epsilon}, \dot{\boldsymbol{\epsilon}})$$

- Approximate the constitutive relation by a neural network

$$\boldsymbol{\sigma}^{n+1} = \mathcal{NN}_{\boldsymbol{\theta}}(\boldsymbol{\sigma}^n, \boldsymbol{\epsilon}^n) + H\boldsymbol{\epsilon}^{n+1}$$





Neural Networks: Inverse Modeling of Viscoelasticity

- We propose the following form for modeling viscosity (assume the time step size is fixed):

$$\boldsymbol{\sigma}^{n+1} - \boldsymbol{\sigma}^n = \mathcal{NN}_{\boldsymbol{\theta}}(\boldsymbol{\sigma}^n, \boldsymbol{\epsilon}^n) + H(\boldsymbol{\epsilon}^{n+1} - \boldsymbol{\epsilon}^n)$$

- H is a free optimizable symmetric positive definite matrix (SPD). Hence the numerical stiffness matrix is SPD.
- Implicit linear equation

$$\boldsymbol{\sigma}^{n+1} - H\boldsymbol{\epsilon}^{n+1} = -H\boldsymbol{\epsilon}^n + \mathcal{NN}_{\boldsymbol{\theta}}(\boldsymbol{\sigma}^n, \boldsymbol{\epsilon}^n) + \boldsymbol{\sigma}^n := \mathcal{NN}^*_{\boldsymbol{\theta}}(\boldsymbol{\sigma}^n, \boldsymbol{\epsilon}^n)$$

- Linear system to solve in each time step $\Rightarrow$ good balance between numerical stability and computational cost.
- Good performance in our numerical examples.

Training Strategy and Numerical Stability

- Physics constrained learning = improved numerical stability in predictive modeling.
- For simplicity, consider two strategies to train an NN-based constitutive relation using direct data $\{(\epsilon_o^n, \sigma_o^n)\}_n$

$$\Delta\sigma^n = H\Delta\epsilon^n + \mathcal{NN}_{\boldsymbol{\theta}}(\sigma^n, \epsilon^n), \quad H \succ 0$$

- Training with input-output pairs

$$\min_{\boldsymbol{\theta}} \sum_n \left(\sigma_o^{n+1} - \left(H\epsilon_o^{n+1} + \mathcal{NN}^*_{\boldsymbol{\theta}}(\sigma_o^n, \epsilon_o^n)\right)\right)^2$$

- Better stability using training on trajectory = physics constrained learning

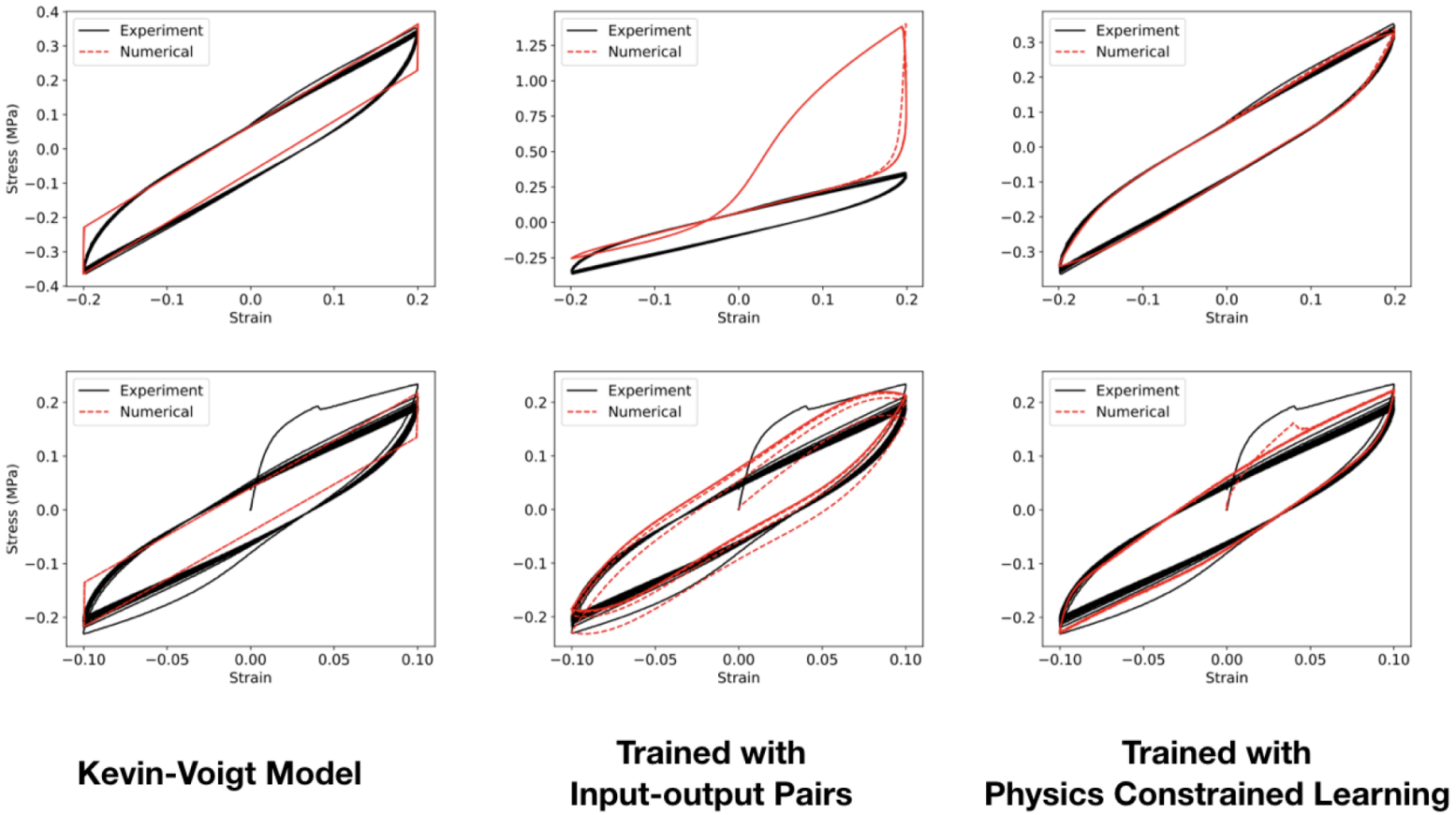
$$\min_{\boldsymbol{\theta}} \sum_n (\sigma^n(\boldsymbol{\theta}) - \sigma_o^n)^2$$

s.t. I.C. $\sigma^1 = \sigma_o^1$ and time integrator $\Delta\sigma^n = H\Delta\epsilon^n + \mathcal{NN}_{\boldsymbol{\theta}}(\sigma^n, \epsilon^n)$

Experimental Data

Dataset 1

Dataset 2



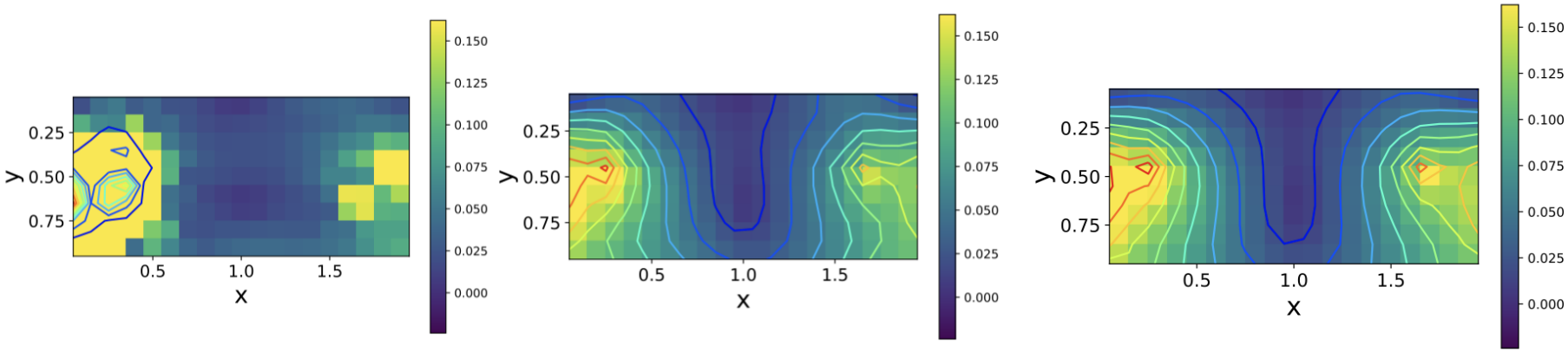
Kevin-Voigt Model | **Trained with Input-output Pairs** | **Trained with Physics Constrained Learning**

Experimental data from: Javidan, Mohammad Mahdi, and Jinkoo Kim. "Experimental and numerical Sensitivity Assessment of Viscoelasticity for polymer composite Materials." Scientific Reports 10.1 (2020): 1–9.

Poroelasticity

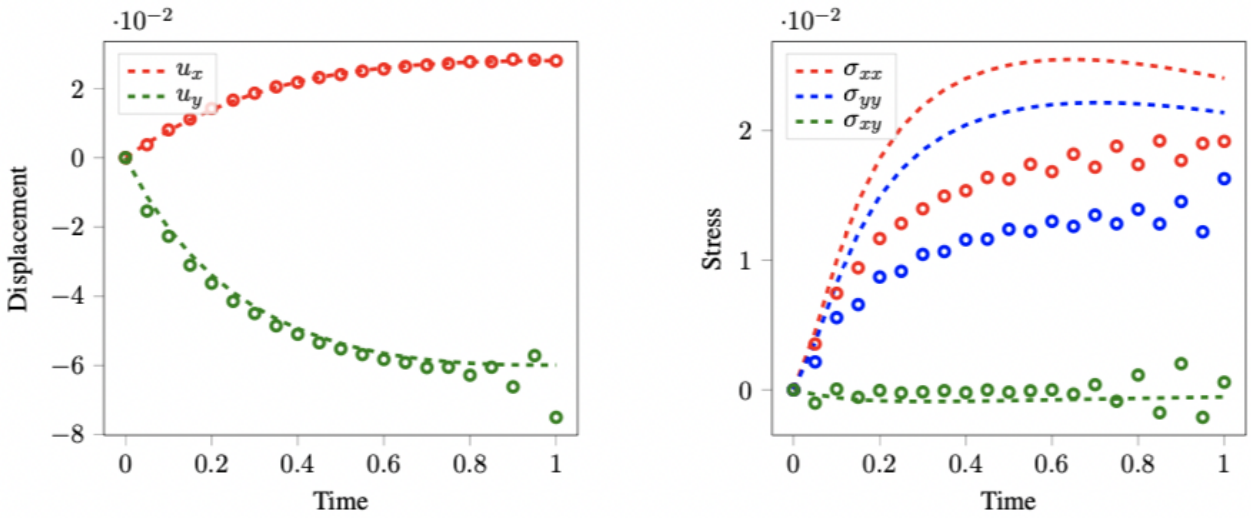
- Comparison with space varying linear elasticity approximation

$$\boldsymbol{\sigma} = H(x, y)\boldsymbol{\epsilon}$$

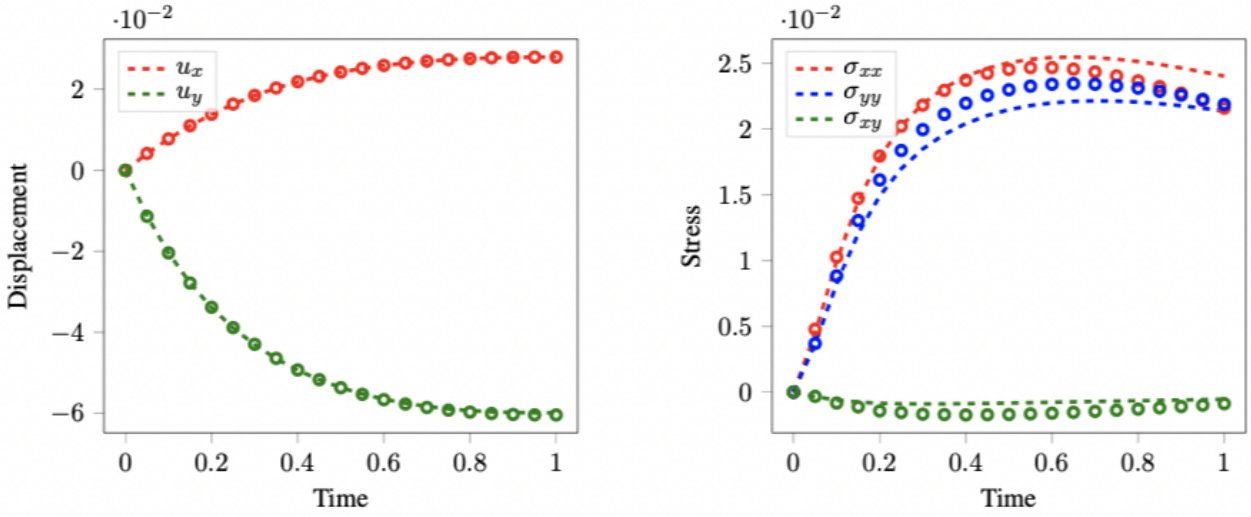


Space Varying Linear Elasticity **NN** **True**

Poroelasticity

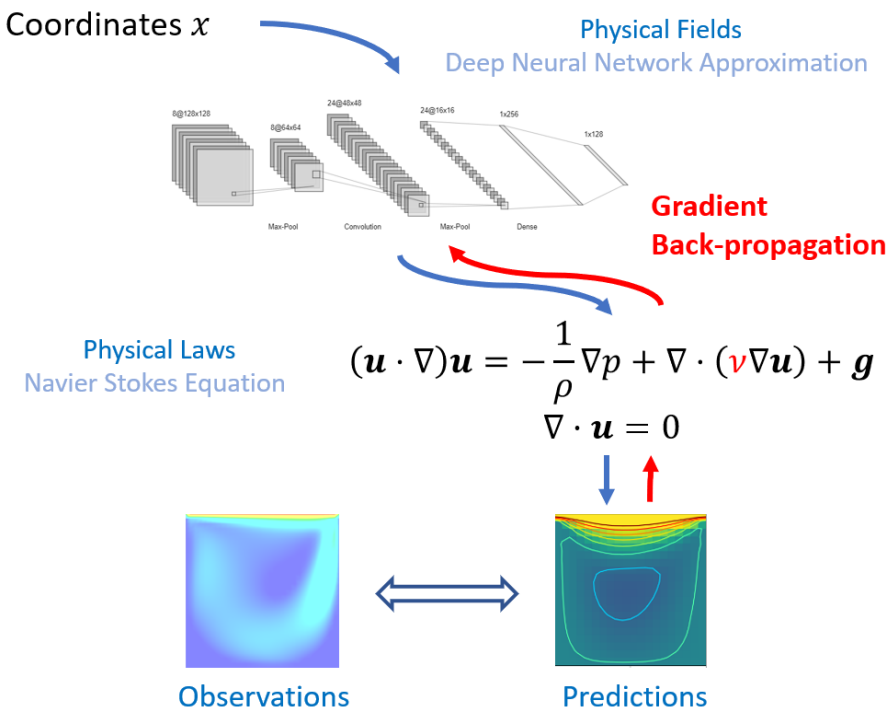


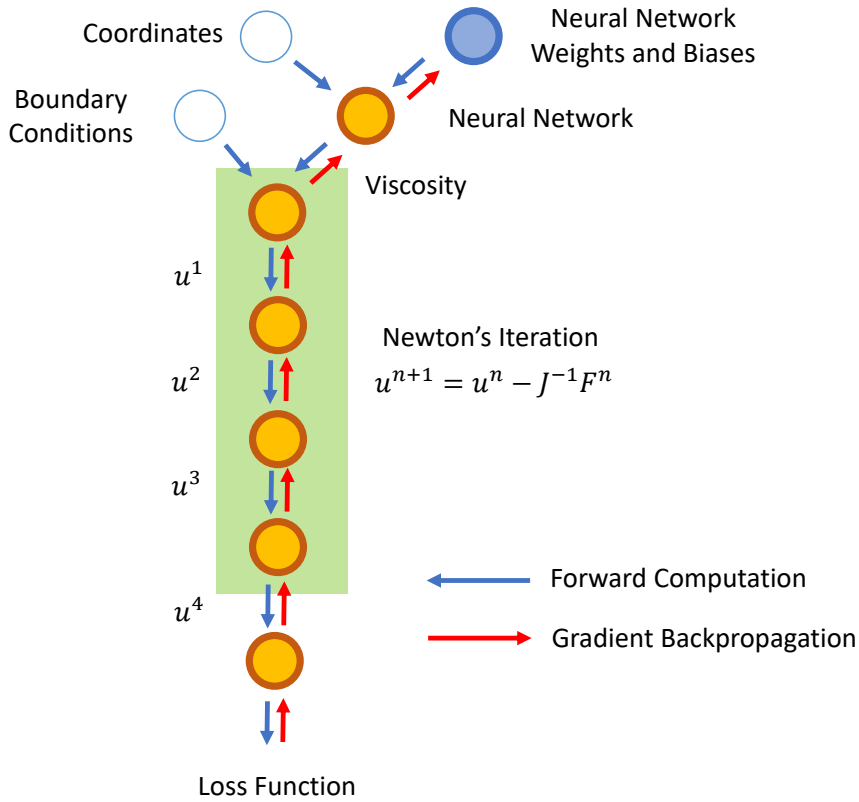
(a) Space Varying Linear Elasticity



(b) NN-based Viscoelasticity

Navier-Stokes Equation

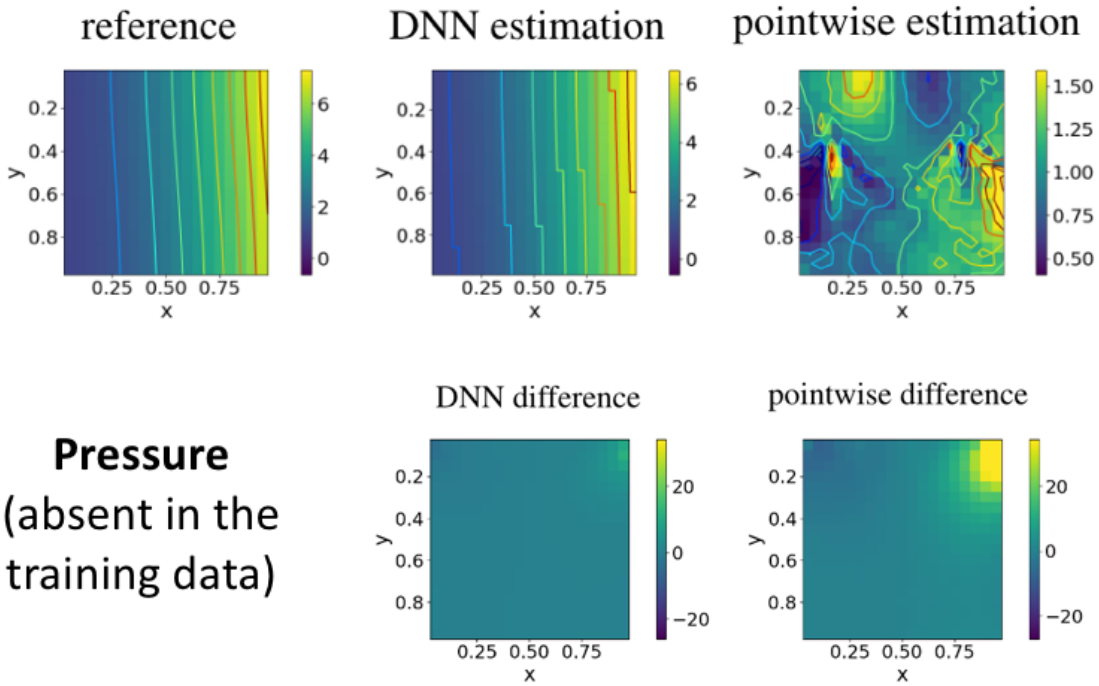


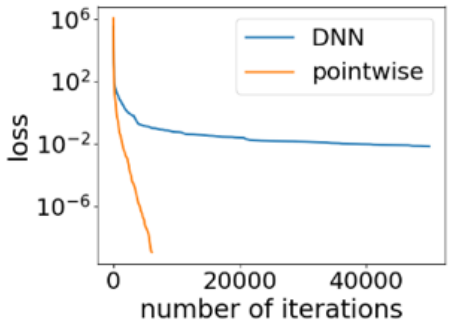


Navier-Stokes Equation

- Data: (u, v)
- Unknown: $\nu(\mathrm{x})$ (represented by a deep neural network)
- Prediction: p (absent in the training data)
- The DNN provides regularization, which generalizes the estimation better!

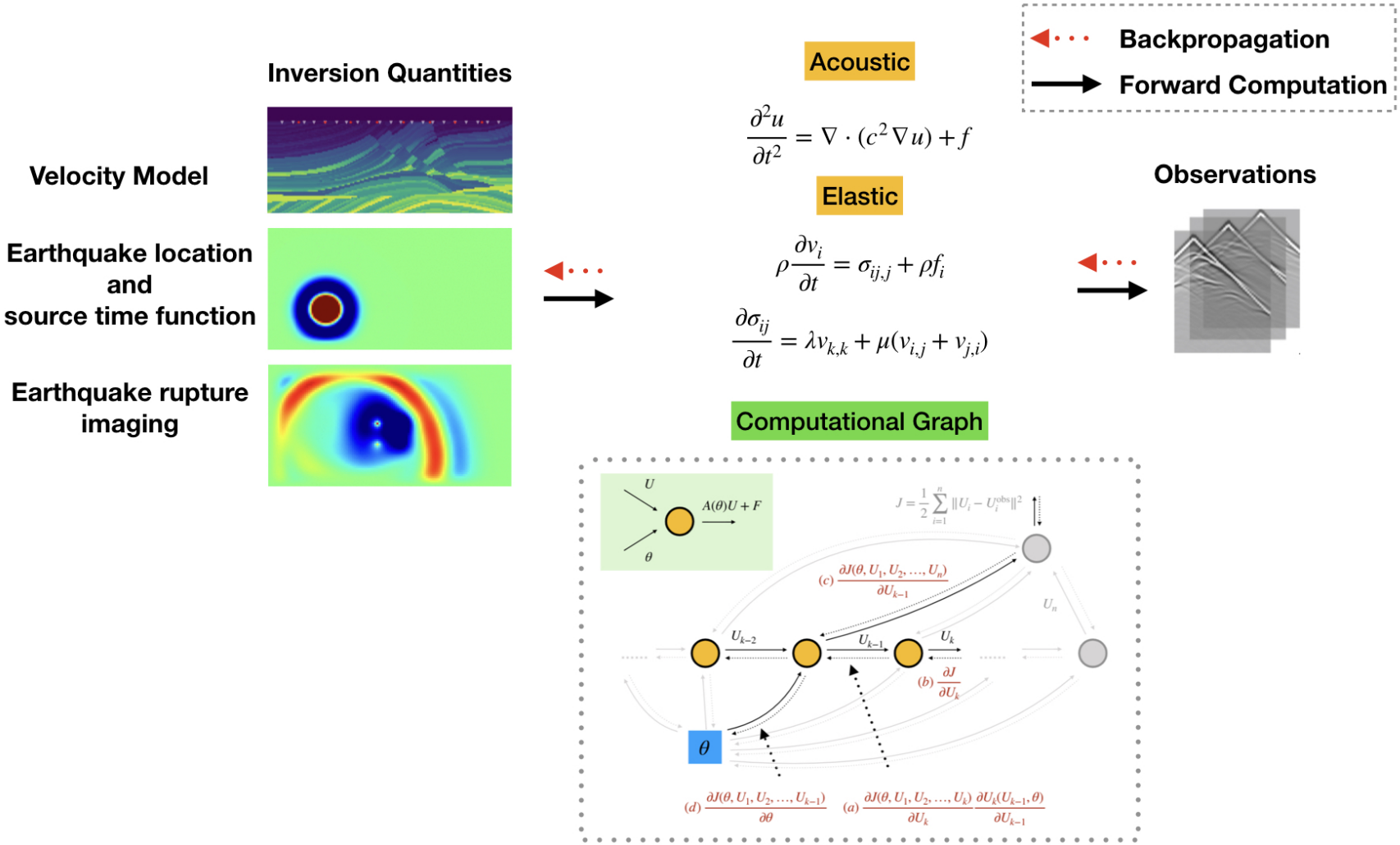
$\nu(\boldsymbol{x})$





ADSeismic.jl: A General Approach to Seismic Inversion

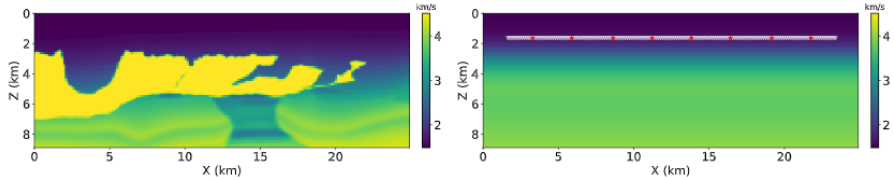
- Many seismic inversion problems can be solved within a unified framework.



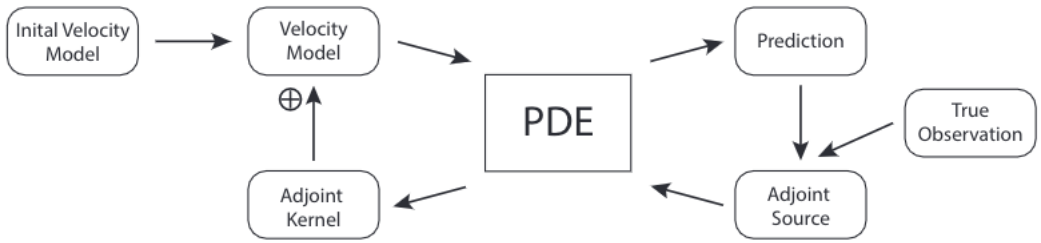
NNFWI: Neural-network-based Full-Waveform Inversion

- Estimate velocity models from seismic observations.

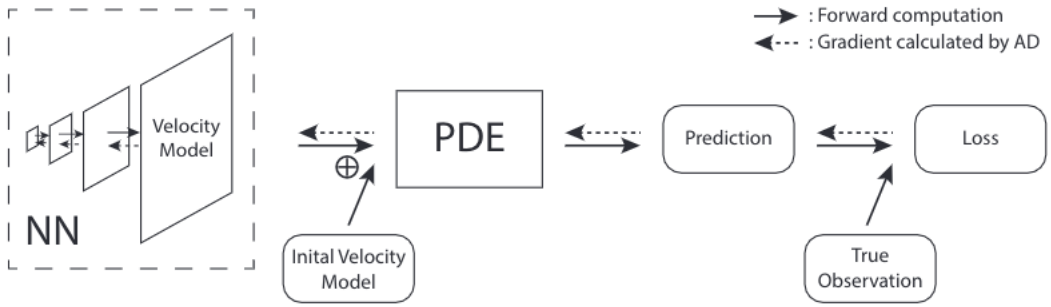
$$\frac{\partial^2 u}{\partial t^2} = \nabla\cdot(m^2\nabla u)+f$$



(a) Traditional FWI:

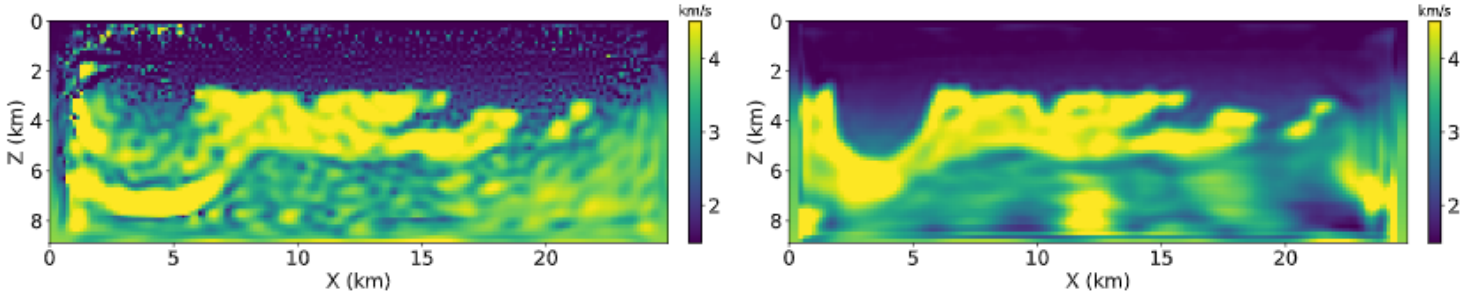


(b) NNFWI:

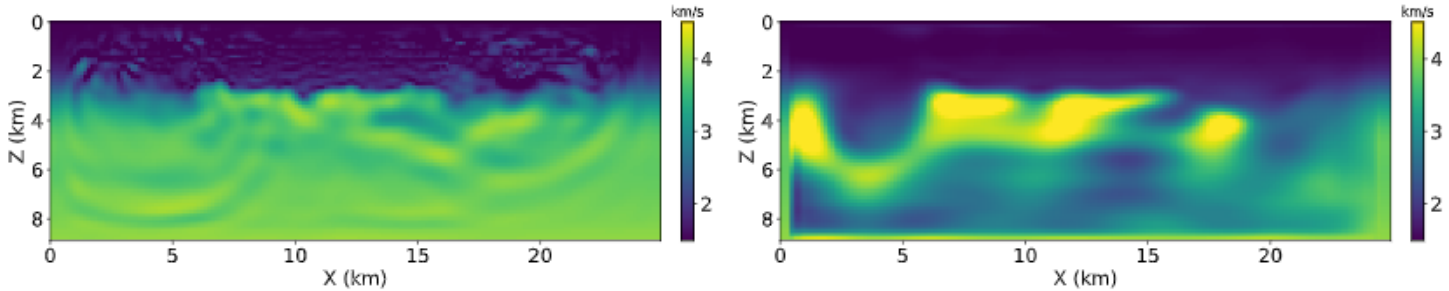


NNFWI: Neural-network-based Full-Waveform Inversion

- Inversion results with a noise level $\sigma = \sigma_0$

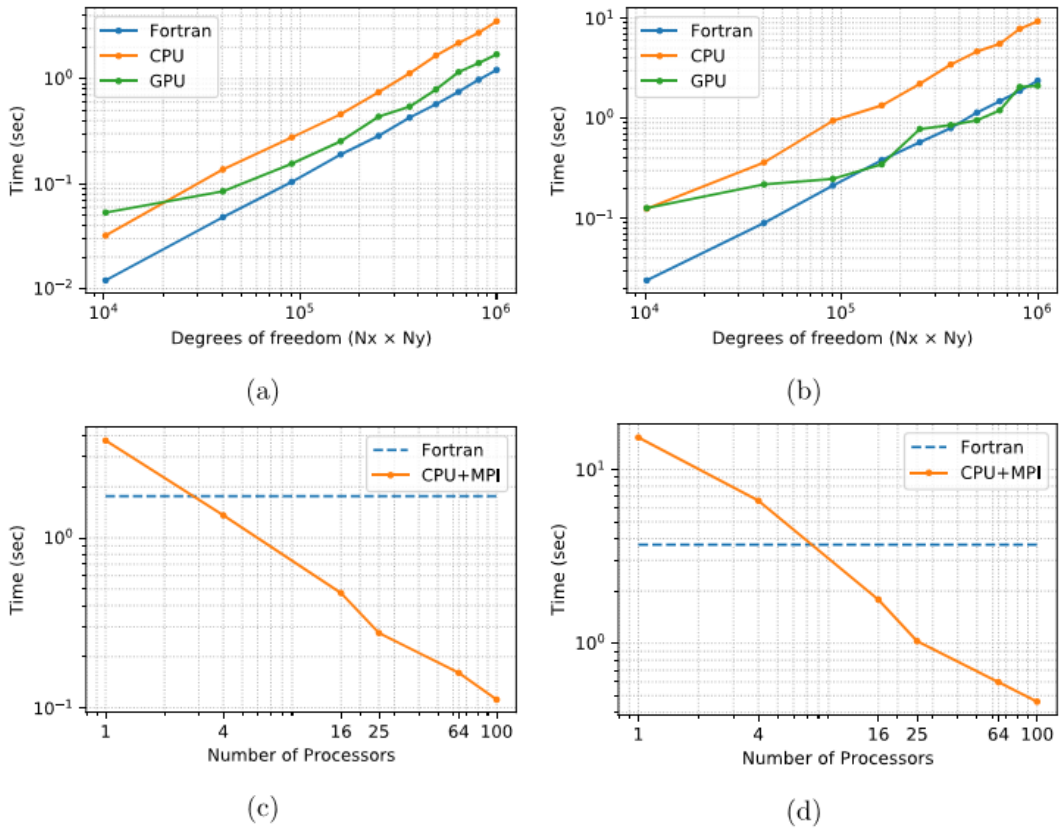


- Inversion results for the same loss function value:

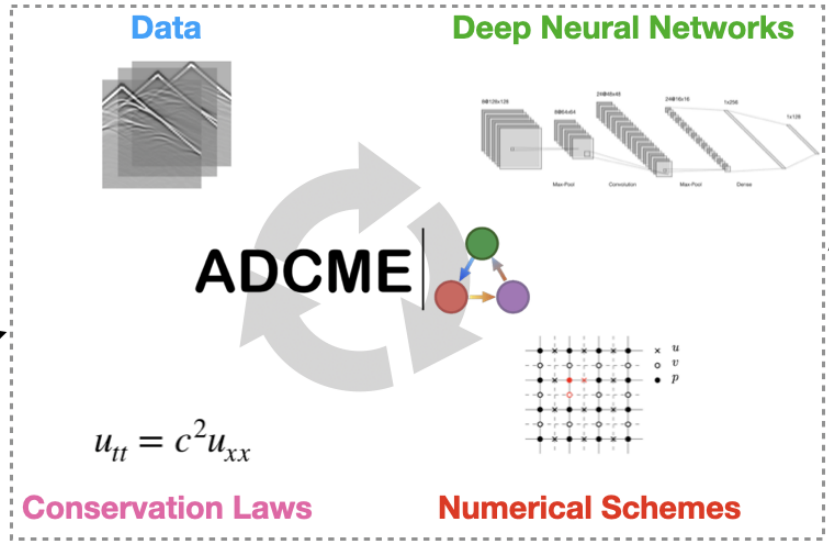


ADSeismic.jl: Performance Benchmark

- Performance is a key focus of ADCME.
- ADCME enables us to utilize heterogeneous (CPUs, GPUs, and TPUs) and distributed (CPU clusters) computing environments.
  Fortran: open-source Fortran90 programs SEISMIC_CPML



A General Approach to Inverse Modeling



FwiFlow.jl
Multiphase Flow
Nonlocal Operators
https://github.com/lidongzh/FwiFlow.jl

ADSeismic.jl
General Seismic Inversion
https://github.com/kailaix/ADSeismic.jl

AdFem.jl
Finite Element Library
for Inverse Modeling
https://github.com/kailaix/AdFem.jl

NNFEM.jl
Constitutive Law Modeling
Hyperelasticity
Elasto-Plasticity
https://github.com/kailaix/NNFEM.jl