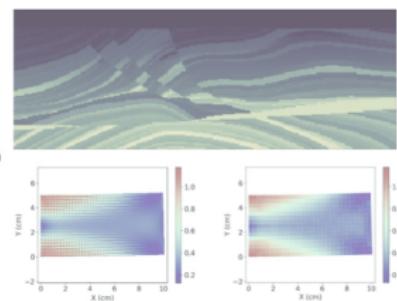
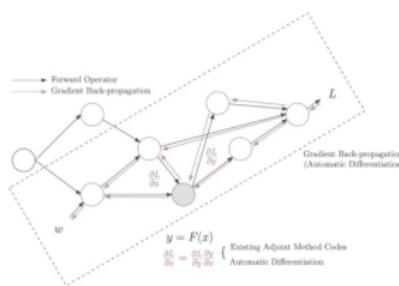
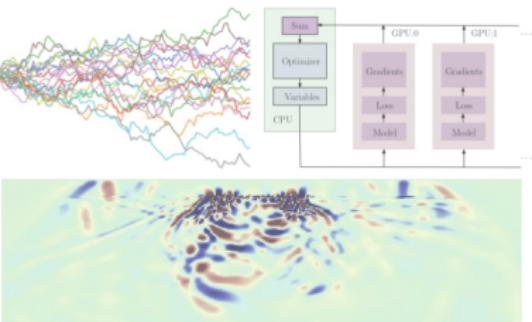


Machine Learning for Inverse Problems in Computational Engineering

Kailai Xu, and Eric Darve

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



Outline

1 Inverse Modeling

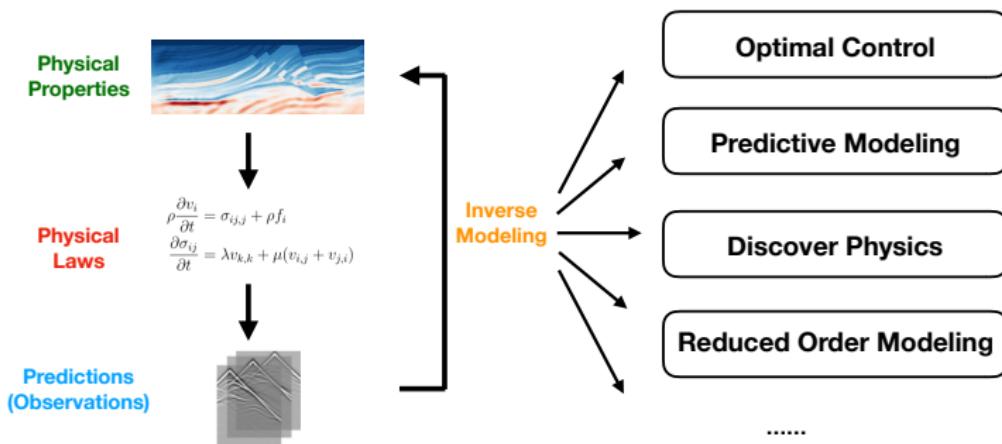
2 Automatic Differentiation

3 Code Example

4 Applications

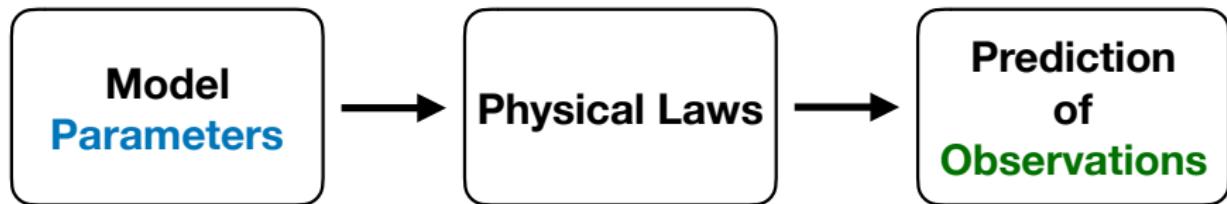
Inverse Modeling

- **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.t

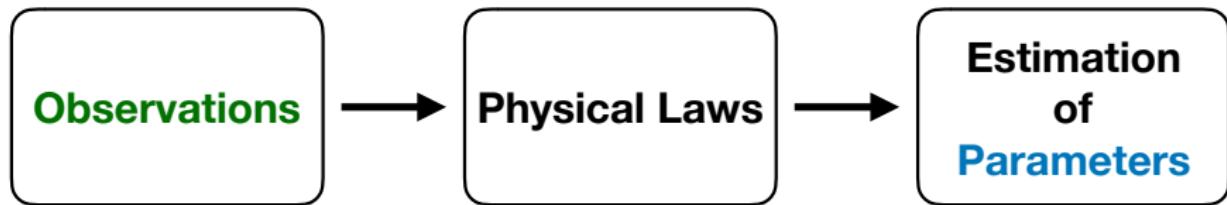


Inverse Modeling

Forward Problem



Inverse Problem



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. Solving for u_h may require solving linear systems or applying an iterative algorithm such as the Newton-Raphson method.

Function Inverse Problem

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

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

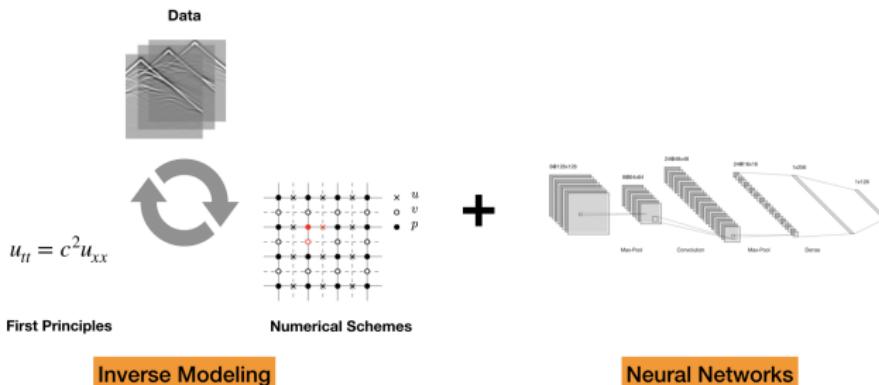
- Koopman operator in dynamical systems.
- Constitutive relations in solid mechanics.
- Turbulent closure relations in fluid mechanics.
- ...

The candidate solution space is **infinite dimensional**.

Machine Learning for Computational Engineering

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

- 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.

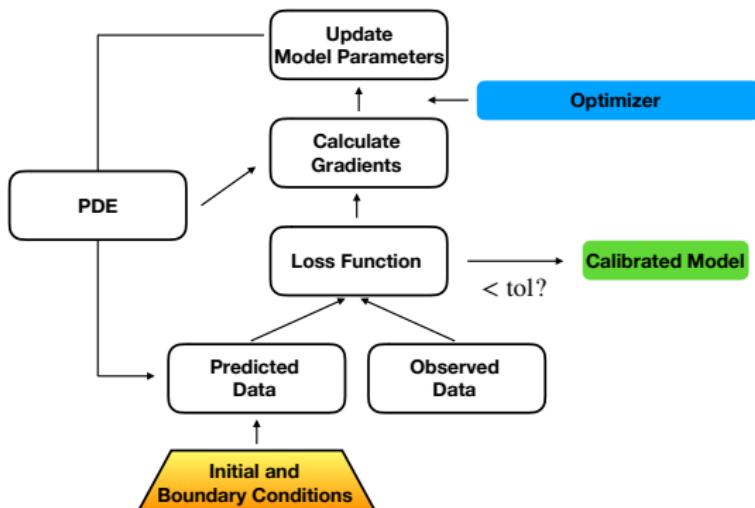


Gradient Based Optimization

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

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

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



Outline

1 Inverse Modeling

2 Automatic Differentiation

3 Code Example

4 Applications

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

||

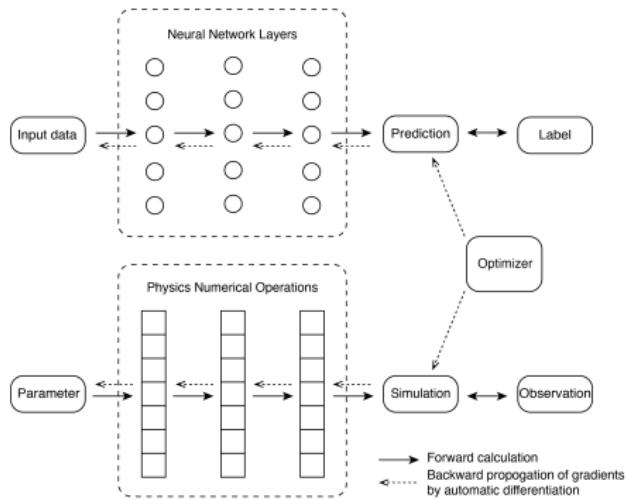
Reverse-mode

Automatic Differentiation

||

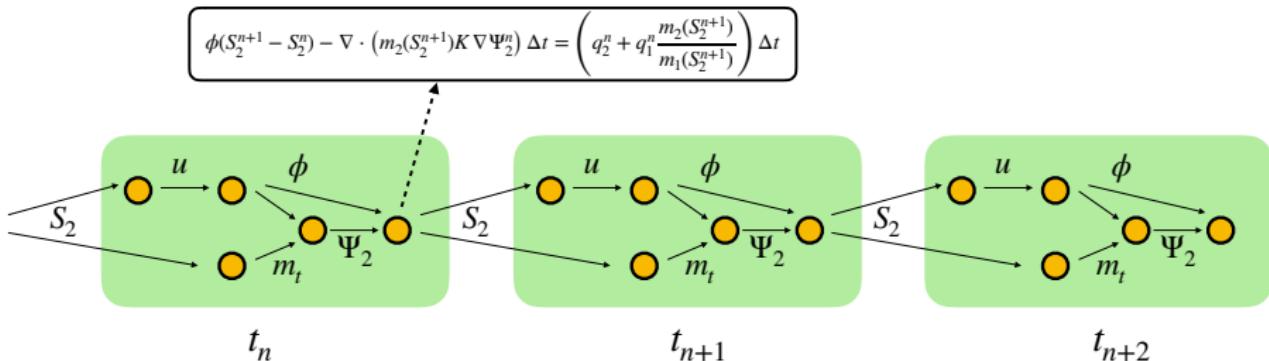
Discrete

Adjoint-State Method

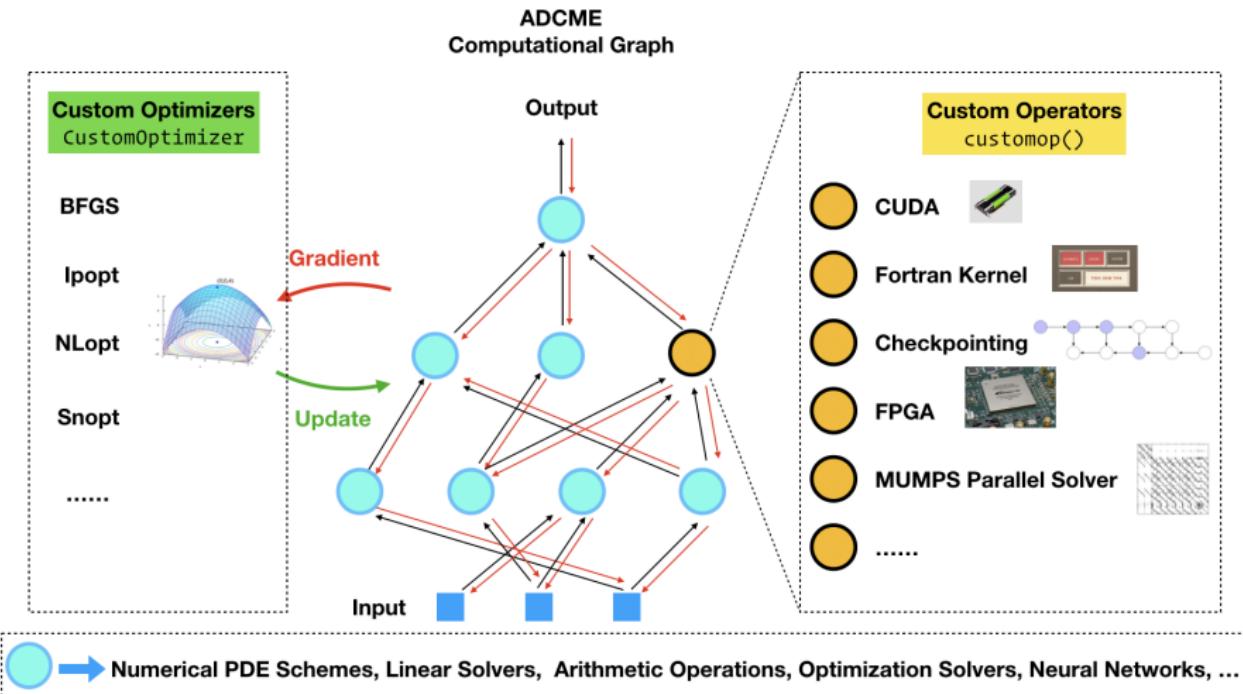


Computational Graph for Numerical Schemes

- 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.

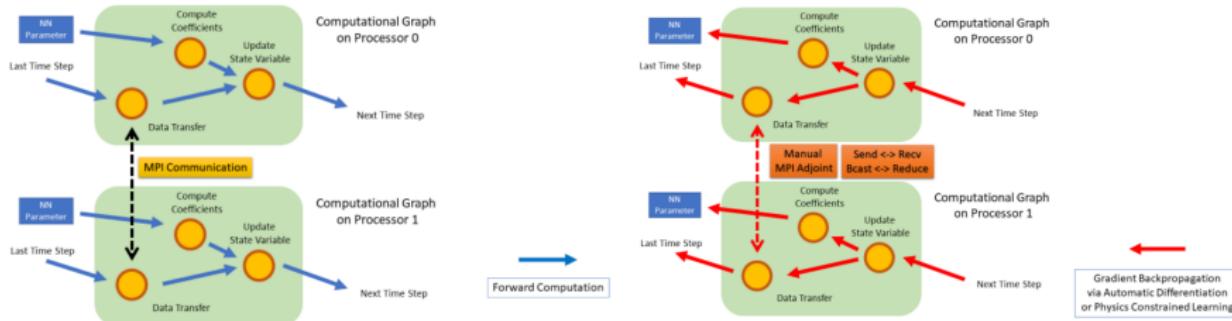


ADCME: Computational-Graph-based Numerical Simulation



Distributed Optimization

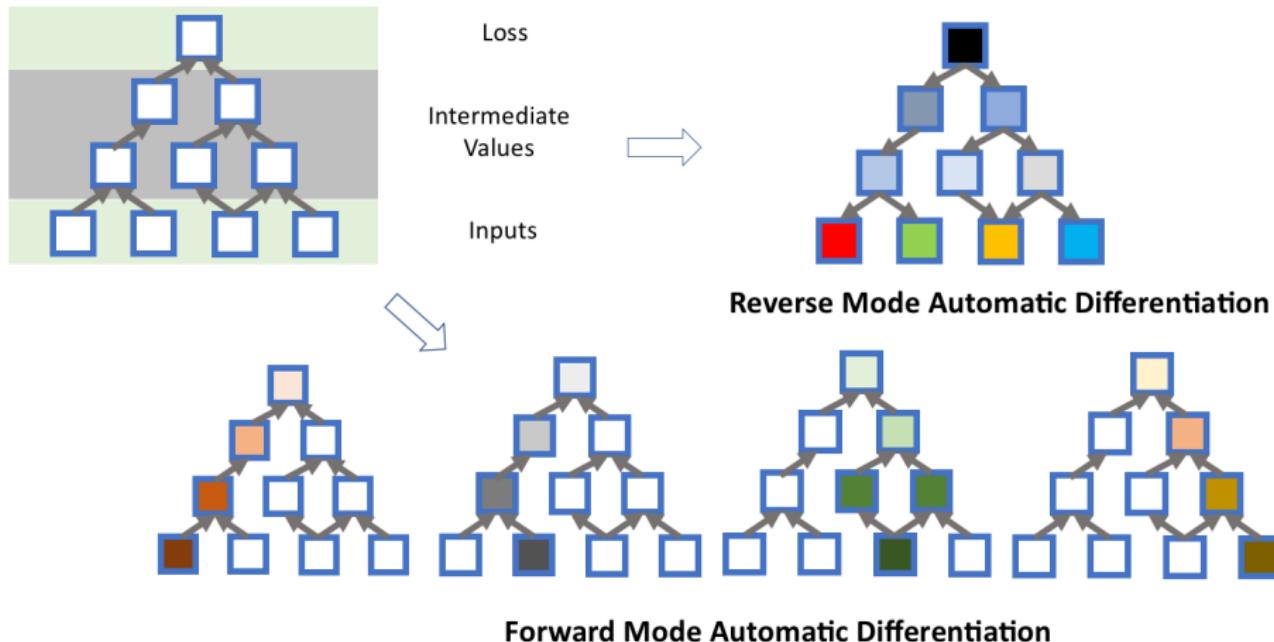
- ADCME also supports MPI-based distributed computing. The parallel model is designed specially for scientific computing.



- 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, ...`

Automatic Differentiation: Forward-mode and Reverse-mode



What is the Appropriate Model for Inverse Problems?

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

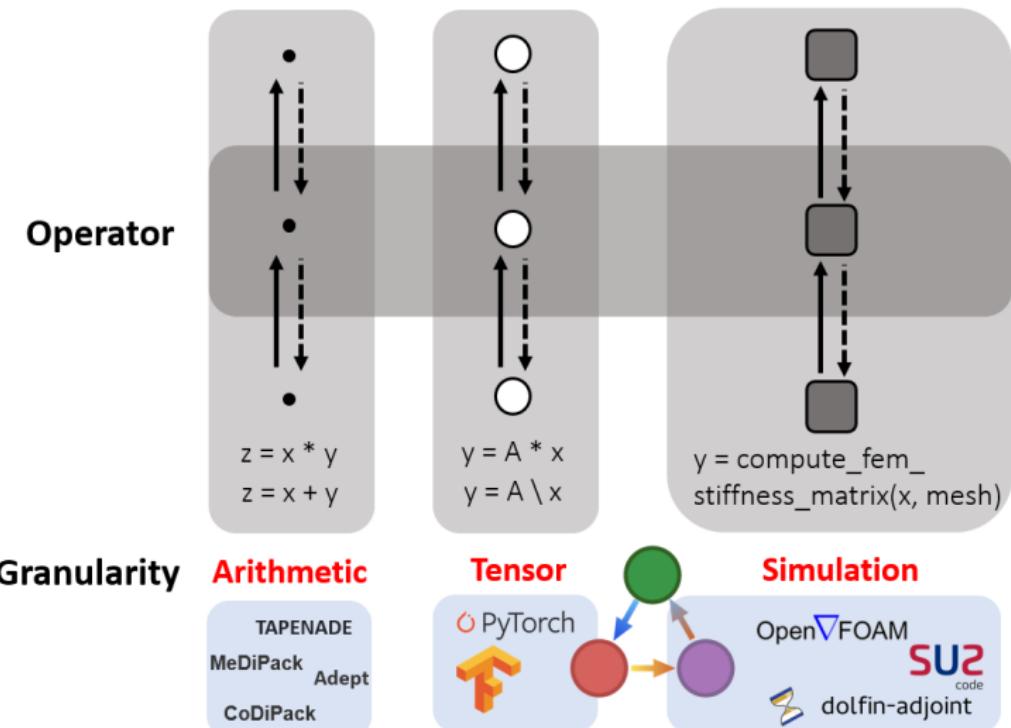
Mode	Suitable for ...	Complexity ¹	Application
Forward	$m \gg n$	$\leq 2.5 \text{ OPS}(f(x))$	UQ
Reverse	$m \ll n$	$\leq 4 \text{ OPS}(f(x))$	Inverse Modeling

- 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.

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

Granularity of Automatic Differentiation



Outline

1 Inverse Modeling

2 Automatic Differentiation

3 Code Example

4 Applications

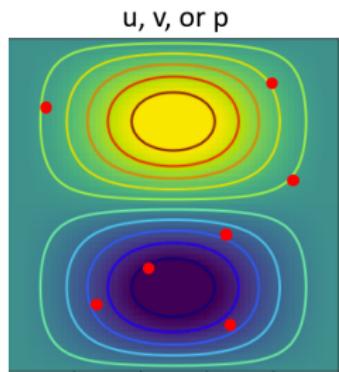
Inverse Modeling of the Stokes Equation

- The governing equation for the Stokes problem

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

- The weak form is given by

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



Inverse Modeling of the Stokes Equation

```
nu = Variable(0.5)
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  
  
sol = Z\rhs
```

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, nu)
```

- Optimization with a one-liner:

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



PoreFlow/ADCME



Simulation Program

Outline

1 Inverse Modeling

2 Automatic Differentiation

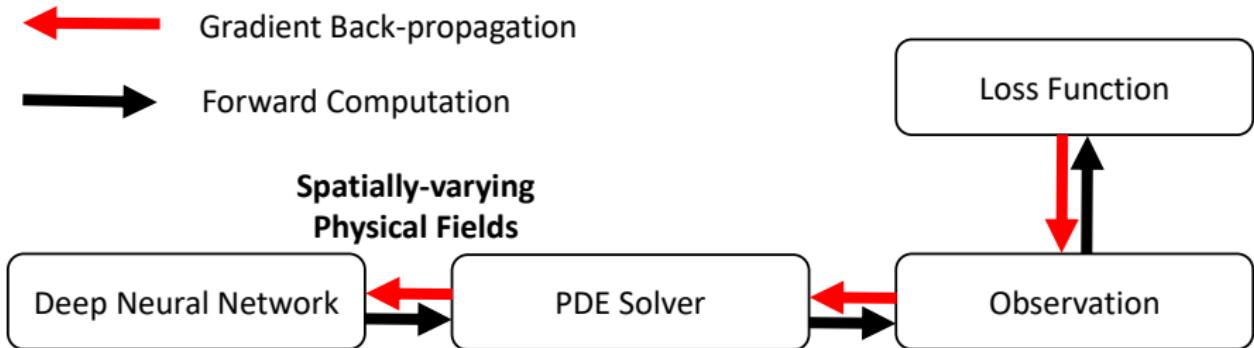
3 Code Example

4 Applications

Learning spatially-varying physical parameters using deep neural networks

- It is easy to adopt ADCME for modeling spatially-varying physical parameters using deep neural networks with a PDE solver.

DNN + PDE + Data = Physics Constrained Data-driven Modeling



Linear Elasticity

DNN + Linear Elasticity + Displacement Data

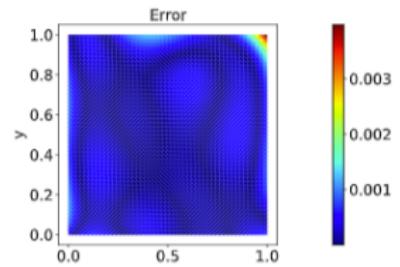
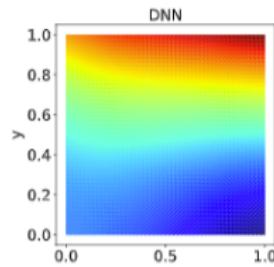
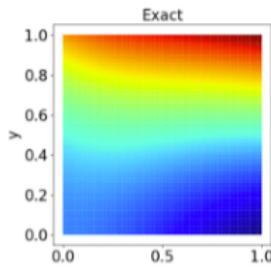
$$\sigma_{ij,j} + b_i = 0, \quad x \in \Omega$$

$$\varepsilon_{ij} = \frac{1}{2}(\textcolor{blue}{u}_{j,i} + \textcolor{blue}{u}_{i,j}), \quad x \in \Omega$$

$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + \mu (\varepsilon_{ij} + \varepsilon_{ji}), \quad x \in \Omega$$

$$\sigma_{ij} n_j = t_j, \quad x \in \Gamma_N; \quad \textcolor{blue}{u}_i = (u_0)_i, \quad x \in \Gamma_D$$

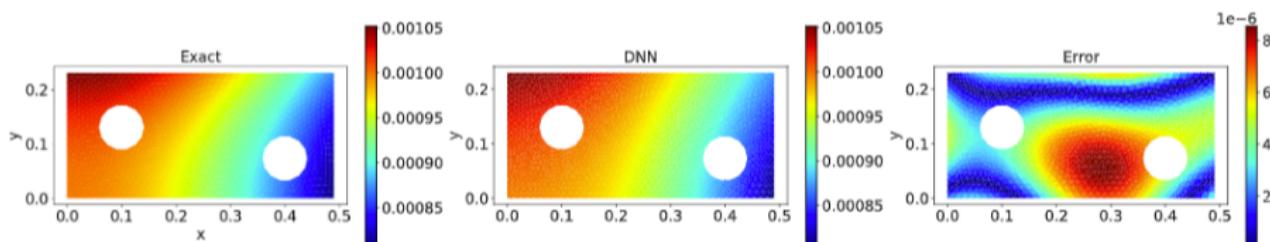
$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \mu = \frac{E\nu}{1-\nu^2}$$



Stokes' Problem

DNN + Stokes' Problem + Pressure Data

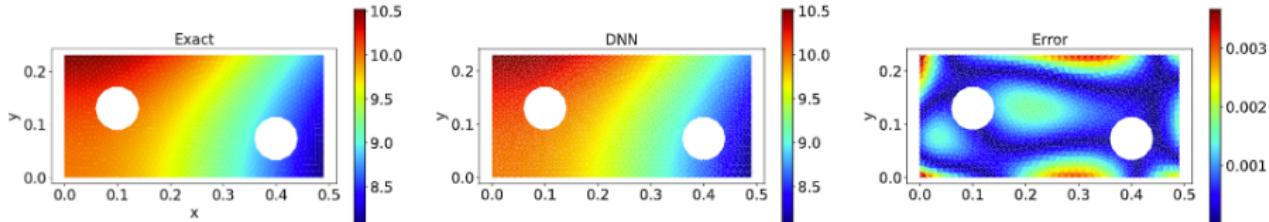
$$\begin{aligned} -\nabla \cdot (\nu \nabla u) + \nabla p &= f \quad \text{in } \Omega \\ \nabla \cdot u &= 0 \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$



Hyperelasticity

DNN + Hyperelasticity + Displacement Data

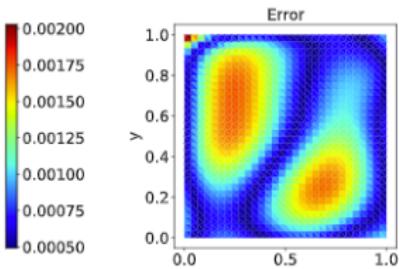
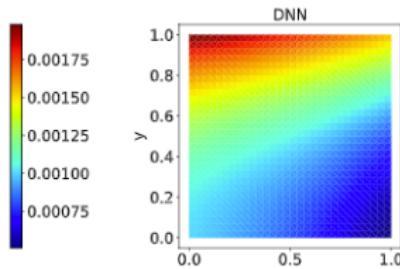
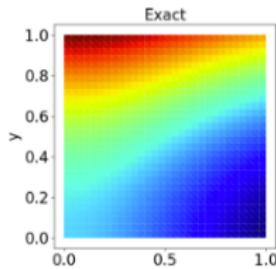
$$\min_u \psi = \frac{\mu}{2}(I_c - 2) - \frac{\mu}{2} \log(J) + \frac{\lambda}{8} \log(J)^2$$
$$F = I + \nabla u, \quad C = F^T F, \quad J = \det(C), \quad I_c = \text{trace}(C)$$
$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}$$



Burgers' Equation

DNN + Burgers' Equation + Velocity Data

$$\begin{aligned}\frac{\partial \textcolor{blue}{u}}{\partial t} + \textcolor{blue}{u} \frac{\partial \textcolor{blue}{u}}{\partial x} + \textcolor{blue}{v} \frac{\partial \textcolor{blue}{u}}{\partial y} &= \nabla \cdot (\nu \nabla \textcolor{blue}{u}) \\ \frac{\partial \textcolor{blue}{v}}{\partial t} + \textcolor{blue}{u} \frac{\partial \textcolor{blue}{v}}{\partial x} + \textcolor{blue}{v} \frac{\partial \textcolor{blue}{v}}{\partial y} &= \nabla \cdot (\nu \nabla \textcolor{blue}{v})\end{aligned}\quad (x, y) \in \Omega, t \in (0, T)$$



Navier-Stokes Equation

- Steady-state Navier-Stokes equation

$$(u \cdot \nabla) u = -\frac{1}{\rho} \nabla p + \nabla \cdot (\nu \nabla u) + g$$
$$\nabla \cdot u = 0$$

- Inverse problem are ubiquitous in fluid dynamics:

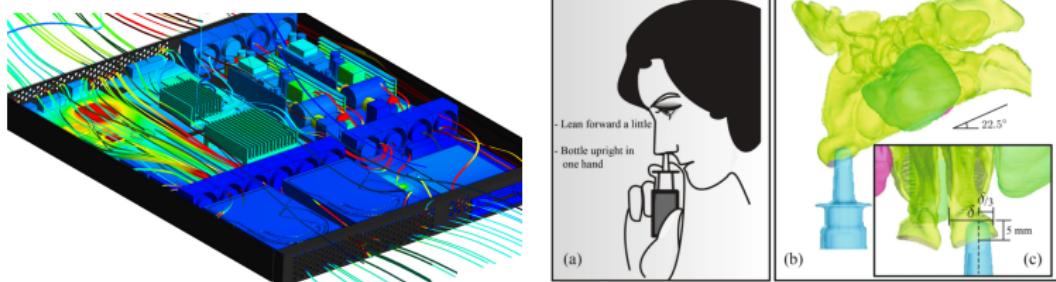
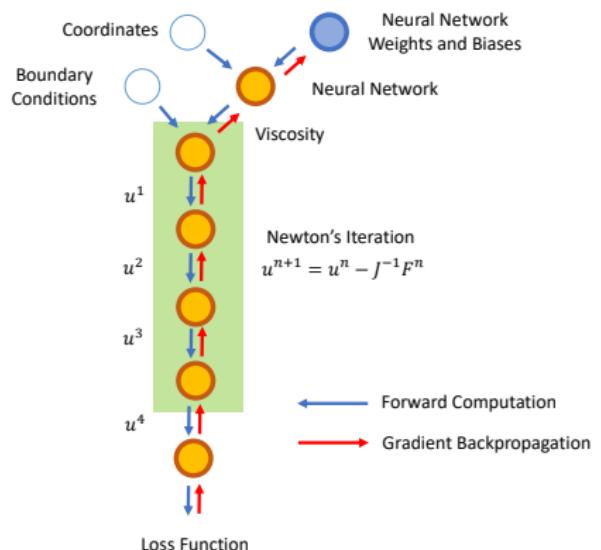
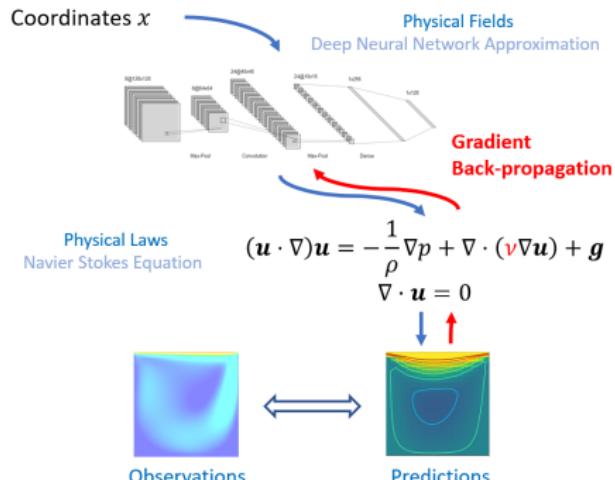


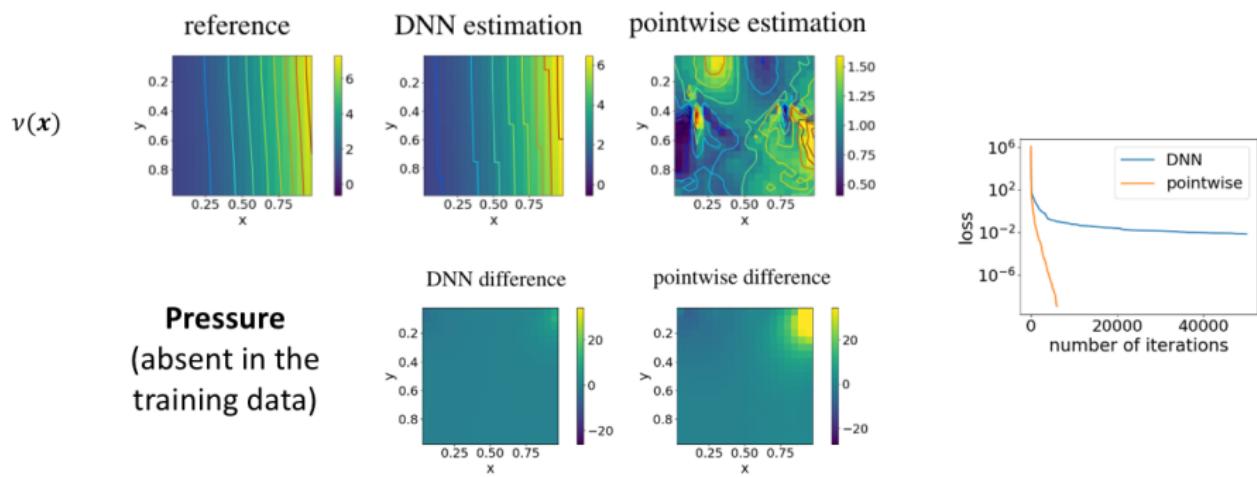
Figure: Left: electronic cooling; right: nasal drug delivery.

Navier-Stokes Equation



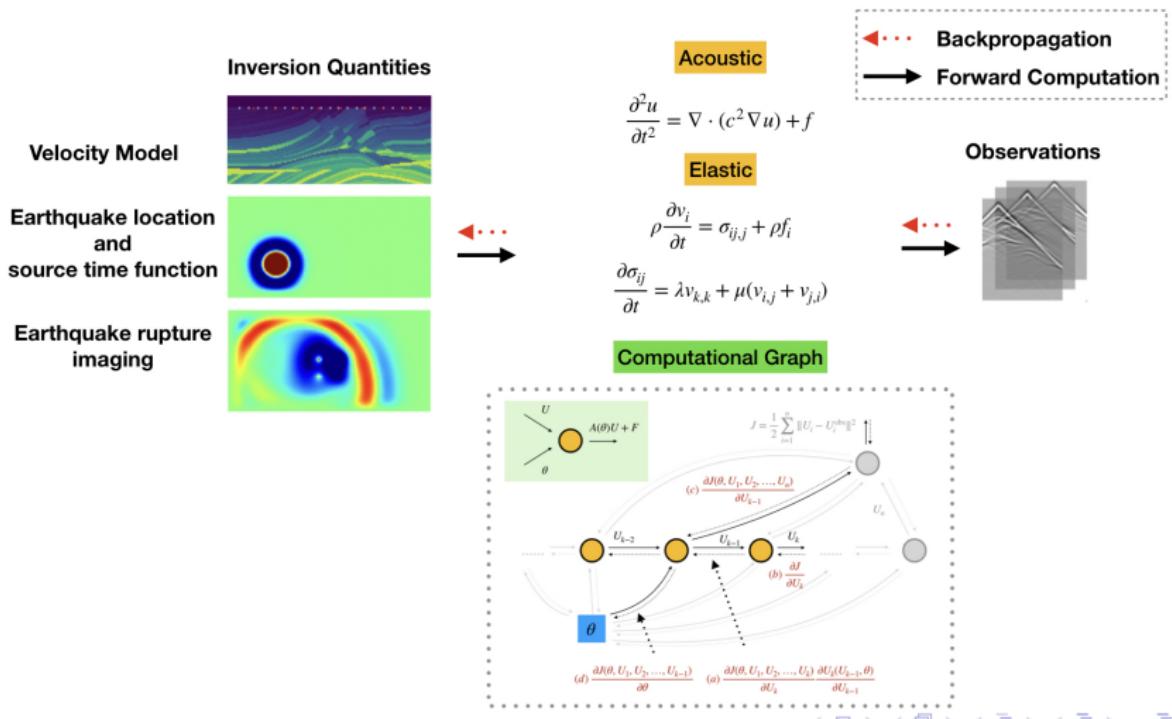
Navier-Stokes Equation

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



ADSeismic.jl: A General Approach to Seismic Inversion

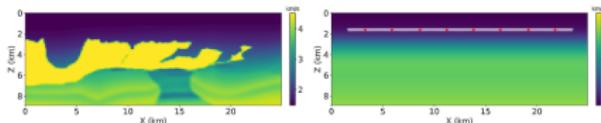
- Many seismic inversion problems can be solved within a unified framework.



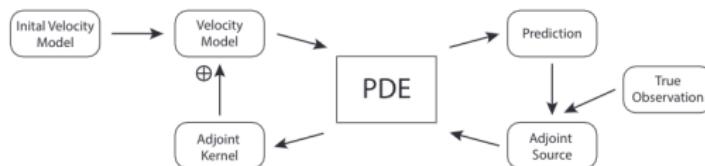
NNFWI: Neural-network-based Full-Waveform Inversion

- Estimate velocity models from seismic observations.

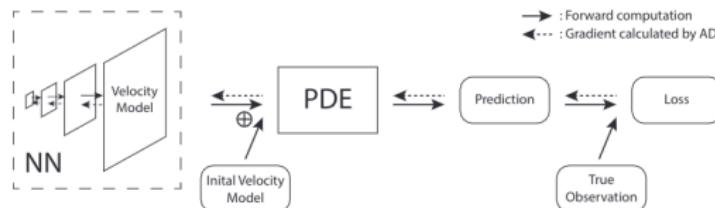
$$\frac{\partial^2 u}{\partial t^2} = \nabla \cdot (\mathbf{m}^2 \nabla u) + f$$



(a) Traditional FWI:

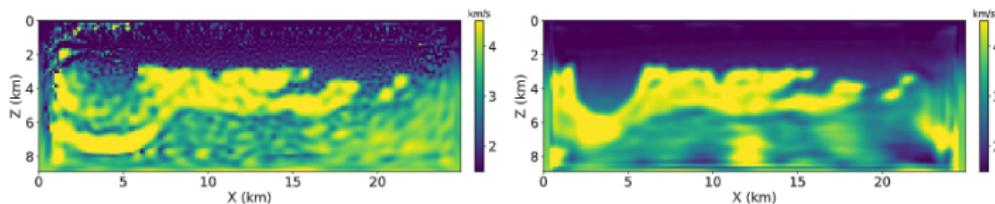


(b) NNFWI:

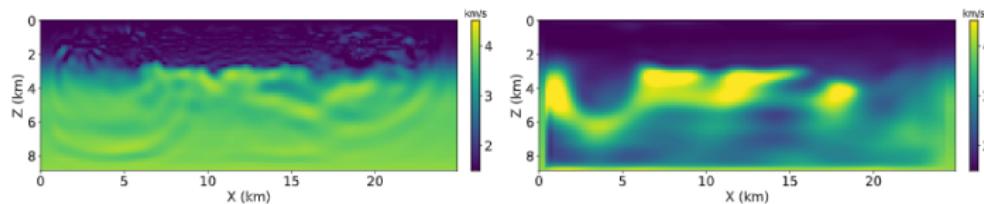


NNFWI: Neural-network-based Full-Waveform Inversion

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

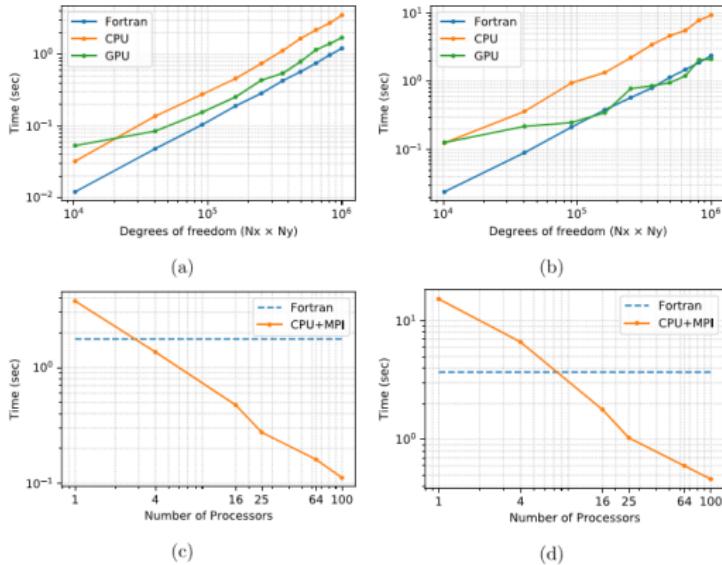


- Inversion results for the same loss function value:

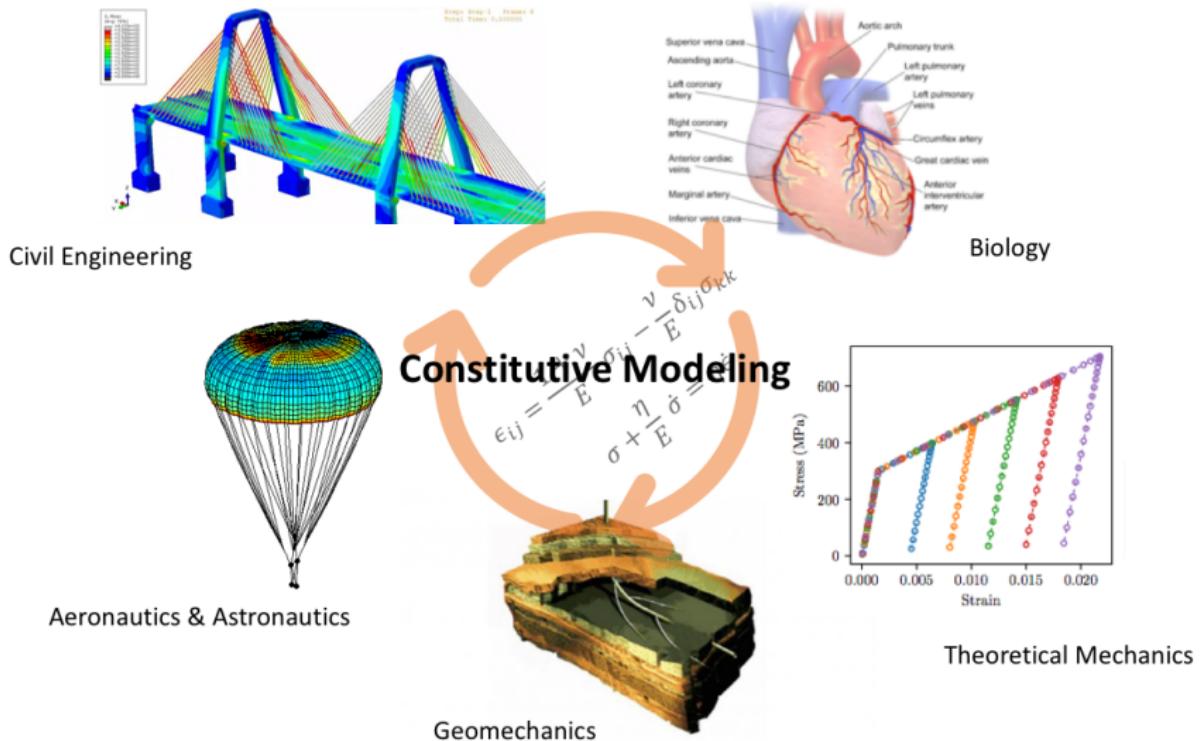


ADSeismic.jl: Performance Benchmark

- 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_CPM1



Constitutive Modeling



Poroelasticity

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

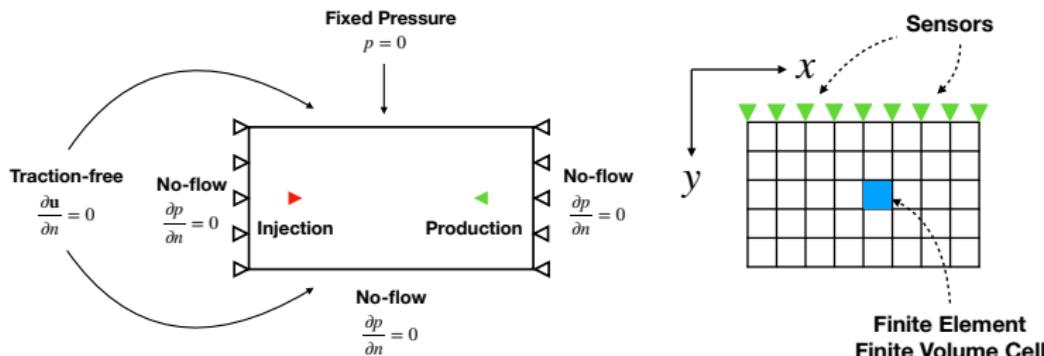
$$\operatorname{div}\boldsymbol{\sigma}(\mathbf{u}) - b\nabla p = 0$$

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

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

- Approximate the constitutive relation by a neural network

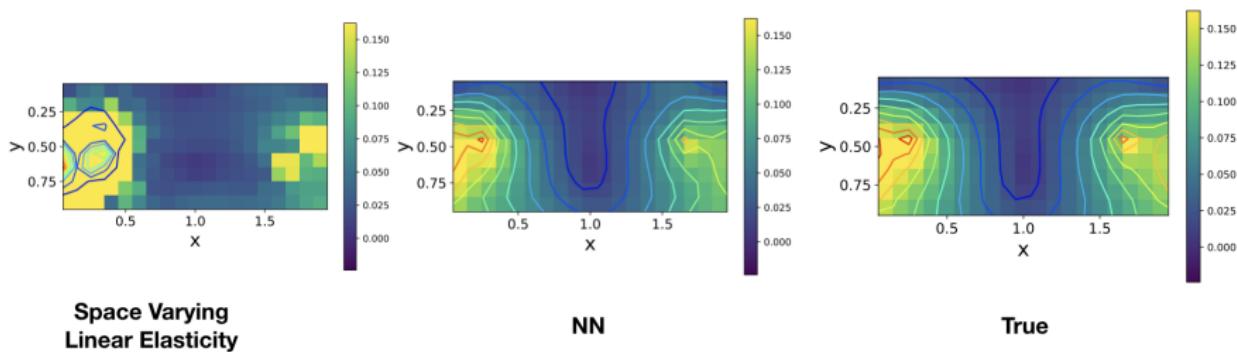
$$\boldsymbol{\sigma}^{n+1} = \mathcal{NN}_{\theta}(\boldsymbol{\sigma}^n, \boldsymbol{\epsilon}^n) + H\boldsymbol{\epsilon}^{n+1}$$



Poroelasticity

- Comparison with space varying linear elasticity approximation

$$\sigma = H(x, y)\epsilon$$

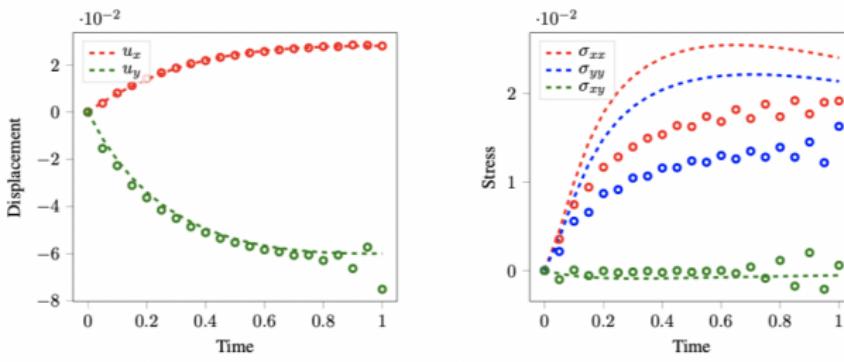


Space Varying
Linear Elasticity

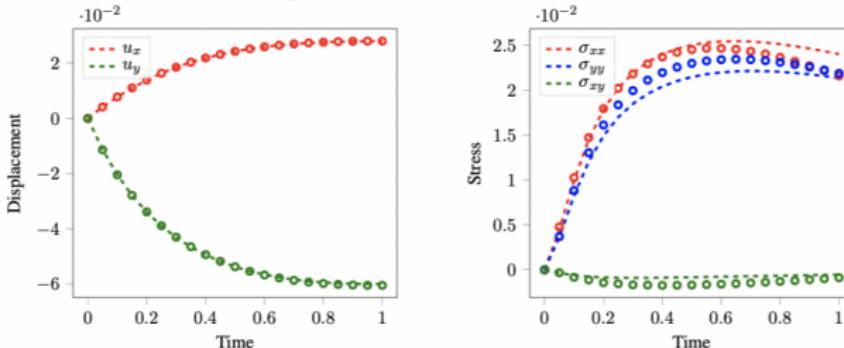
NN

True

Poroelasticity



(a) Space Varying Linear Elasticity



(b) NN-based Viscoelasticity

A Paradigm for Inverse Modeling

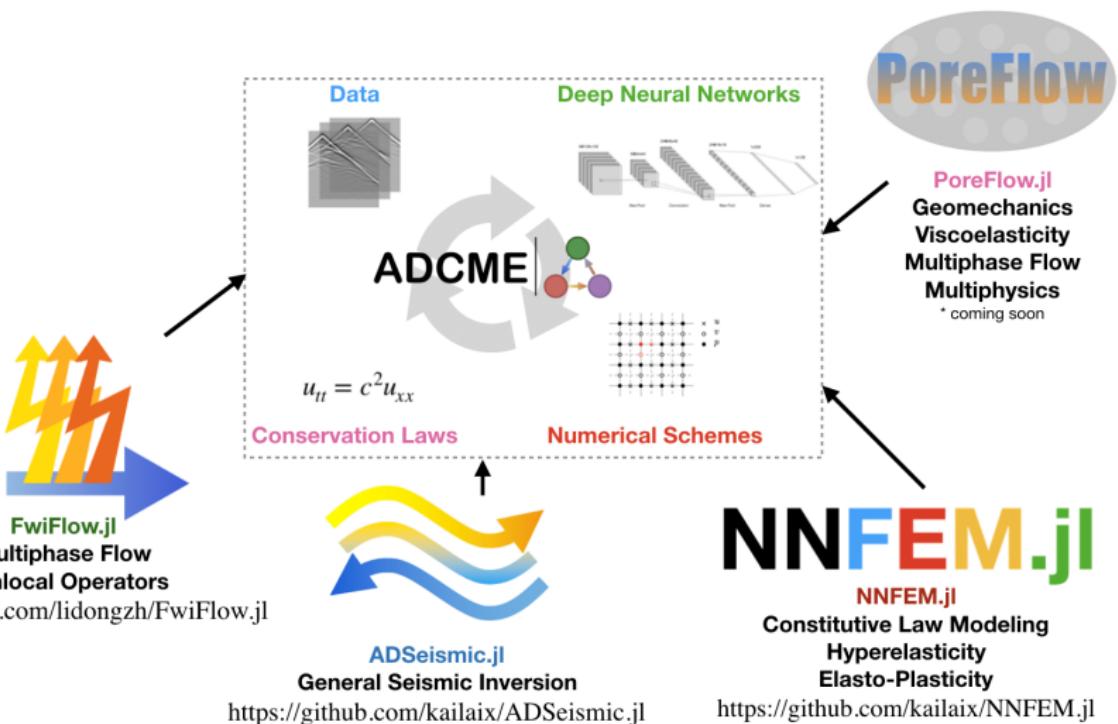
- Most inverse modeling problems can be classified into 4 categories.
To be more concrete, consider the PDE for describing physics

$$\nabla \cdot (\theta \nabla u(x)) = 0 \quad \mathcal{BC}(u(x)) = 0 \quad (2)$$

We observe some quantities depending on the solution u and want to estimate θ .

Expression	Description	ADCME Solution	Note
$\nabla \cdot (\textcolor{red}{c} \nabla u(x)) = 0$	Parameter Inverse Problem	Discrete Adjoint State Method	c is the minimizer of the error functional
$\nabla \cdot (\textcolor{red}{f}(x) \nabla u(x)) = 0$	Function Inverse Problem	Neural Network Functional Approximator	$f(x) \approx \mathcal{NN}_w(x)$
$\nabla \cdot (\textcolor{red}{f}(u) \nabla u(x)) = 0$	Relation Inverse Problem	Residual Learning Physics Constrained Learning (PCL)	$f(u) \approx \mathcal{NN}_w(u)$
$\nabla \cdot (\varpi \nabla u(x)) = 0$	Stochastic Inverse Problem	Physical Generative Neural Networks (PhysGNN)	$\varpi = \mathcal{NN}_w(v_{\text{latent}})$

A General Approach to Inverse Modeling



Reference

- Methodology and Implementation:
 - Physics Constrained Learning for Data-driven Inverse Modeling from Sparse Observations (**Core techniques!**)
 - A General Approach to Seismic Inversion with Automatic Differentiation
 - Time-lapse Full-waveform Inversion for Subsurface Flow Problems with Intrusive Automatic Differentiation
- Constitutive Modeling:
 - Learning Constitutive Relations from Indirect Observations Using Deep Neural Networks
 - Learning Constitutive Relations using Symmetric Positive Definite Neural Networks
 - Inverse Modeling of Viscoelasticity Materials using Physics Constrained Learning
- Learning Spatially-varying Fields:
 - Solving Inverse Problems in Steady State Navier-Stokes Equations using Deep Neural Networks