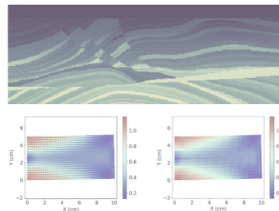
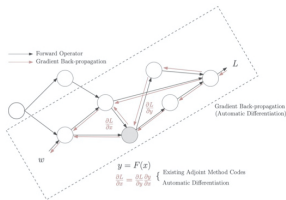
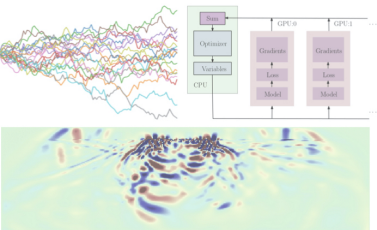


Machine Learning for Inverse Problems in Computational Engineering

Kailai Xu, and Eric Darve

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



Outline

- 1 Inverse Modeling
- 2 Automatic Differentiation
- 3 Code Example
- 4 Applications

Forward Problem

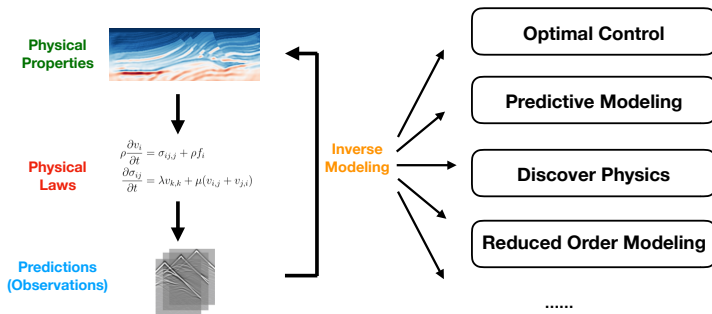


Inverse Problem



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

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

- 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

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

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

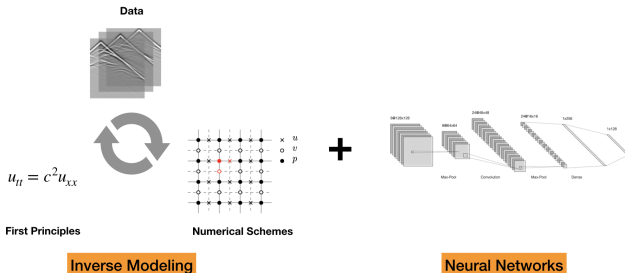
- Koopman operator in dynamical systems.
- Constitutive relations in solid mechanics.
- Turbulent closure relations in fluid mechanics.
- ...

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

Machine Learning for Computational Engineering

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

- 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.

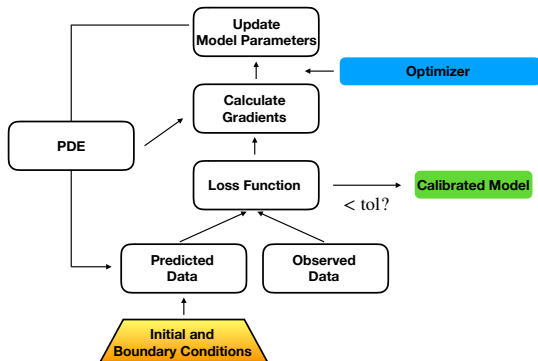


Gradient Based Optimization

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

- 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

||

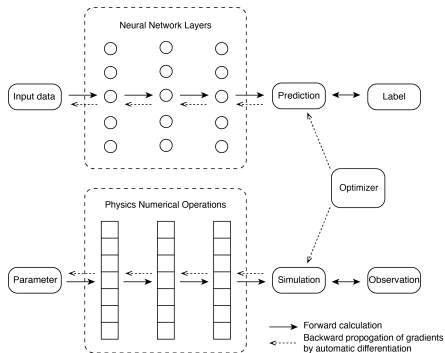
Reverse-mode

Automatic Differentiation

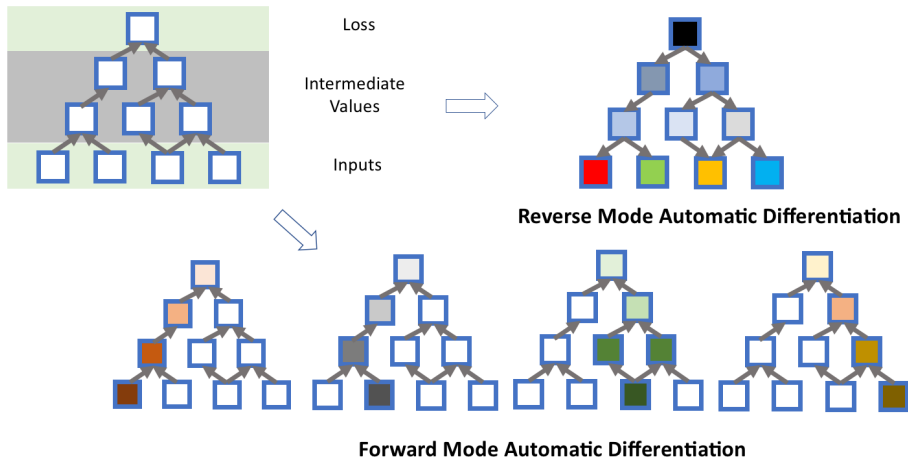
||

Discrete

Adjoint-State Method



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

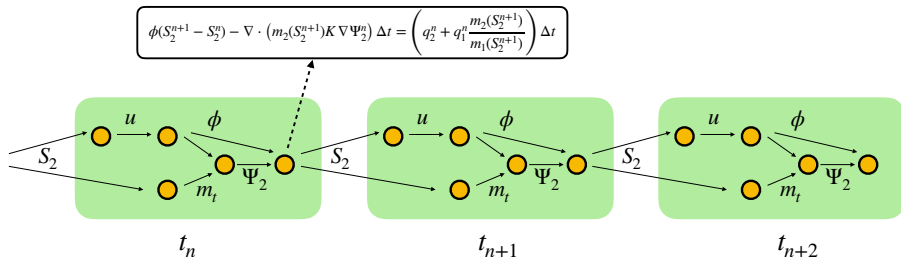
- There are also many other interesting topics
 - Mixed mode AD: many-to-many mappings.
 - Computing sparse Jacobian matrices using AD by exploiting sparse structures.

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.

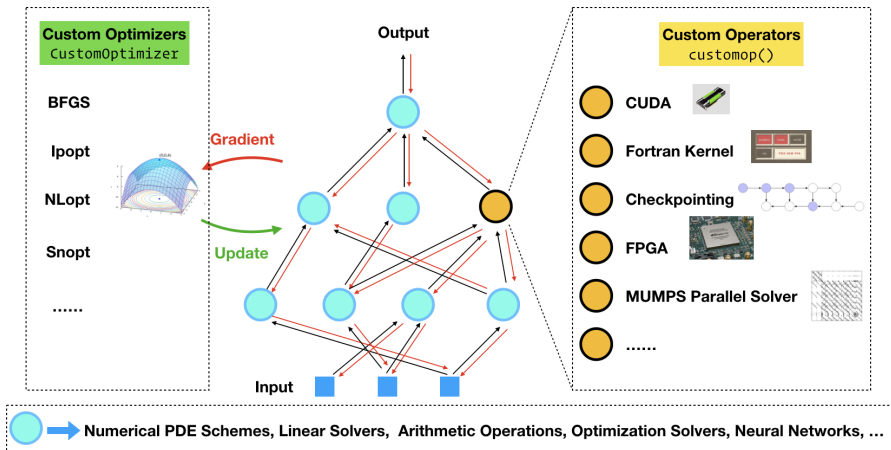
Computational Graph for Numerical Schemes

- 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.



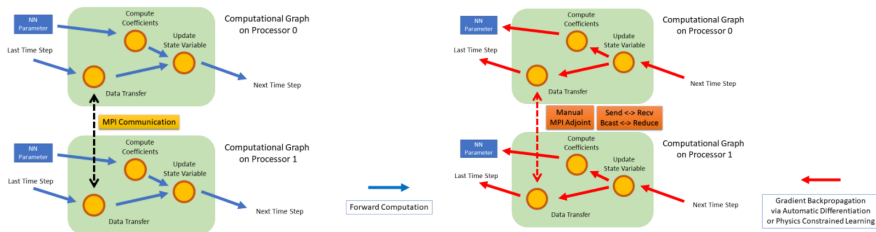
ADCME: Computational-Graph-based Numerical Simulation

ADCME
Computational Graph



Distributed Optimization

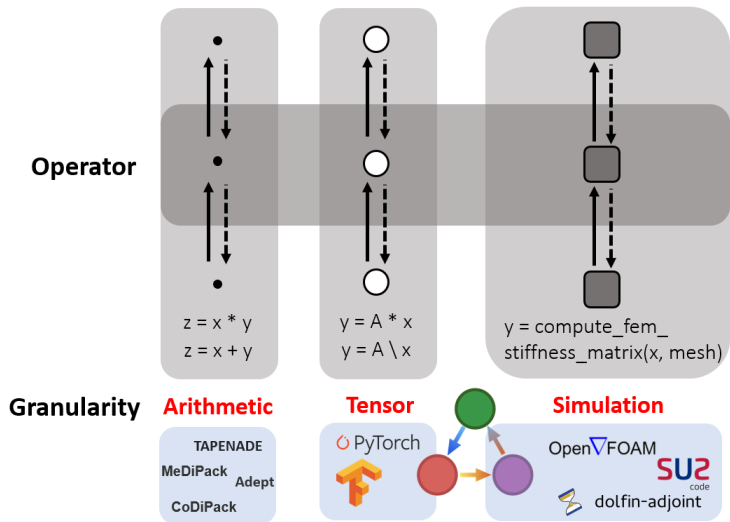
- ADCME also supports MPI-based distributed computing. The parallel model is designed specially for scientific computing.



- 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`, ...

Granularity of Automatic Differentiation



Outline

- 1 Inverse Modeling
- 2 Automatic Differentiation
- 3 Code Example**
- 4 Applications

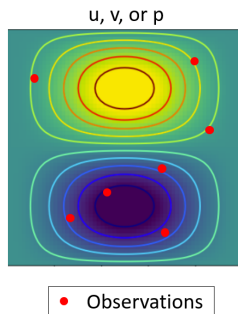
Inverse Modeling of the Stokes Equation

- The governing equation for the Stokes problem

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

- The weak form is given by

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



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

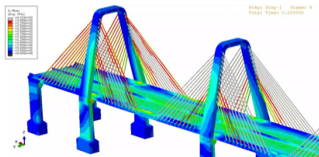


Simulation Program

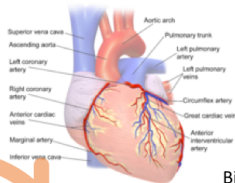
Outline

- 1 Inverse Modeling
- 2 Automatic Differentiation
- 3 Code Example
- 4 Applications**

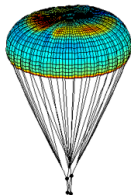
Constitutive Modeling



Civil Engineering



Biology



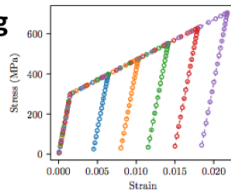
Aeronautics & Astronautics

Constitutive Modeling

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

$$\sigma + \frac{\eta}{E} \dot{\sigma} = \frac{E}{1 + \eta \dot{\epsilon}}$$


Geomechanics



Theoretical Mechanics

Governing Equations

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

- **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

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

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

Residual Learning using Full-field Data

- Weak form of balance equations of linear momentum

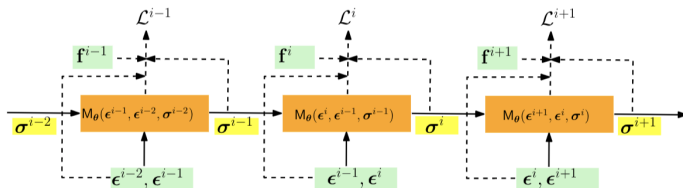
$$P_i(\theta) = \int_V \rho \ddot{u}_i \delta u_i dV + \int_V \underbrace{\sigma_{ij}(\theta)}_{\text{embedded neural network}} \delta \varepsilon_{ij} dV$$

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

- Train the neural network by

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

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



Representation of Constitutive Relations

- Proper form of constitutive relation is crucial for numerical stability

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

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

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

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

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

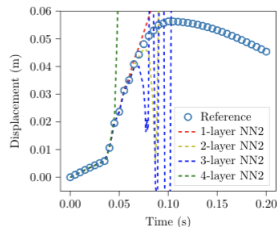
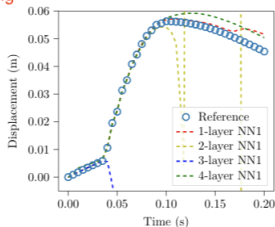
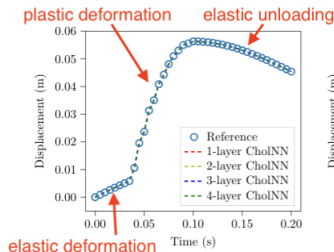
Modeling Elasto-plasticity

- Comparison of different neural network architectures

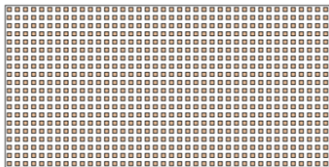
$$\sigma^{n+1} = \mathbf{L}_\theta(\epsilon^{n+1}, \epsilon^n, \sigma^n) \mathbf{L}_\theta(\epsilon^{n+1}, \epsilon^n, \sigma^n)^T (\epsilon^{n+1} - \epsilon^n) + \sigma^n$$

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

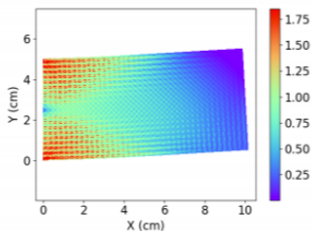
$$\sigma^{n+1} = \text{NN}_\theta(\epsilon^{n+1}, \epsilon^n, \sigma^n) + \sigma^n$$



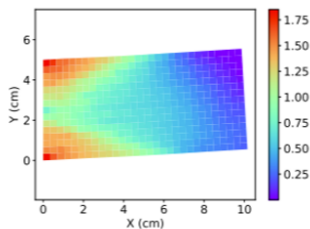
Modeling Elasto-plasticity: Multi-scale



Fiber Reinforced Thin Plate



Reference von Mises stress



SPD-NN

Static Hyperelasticity Problem

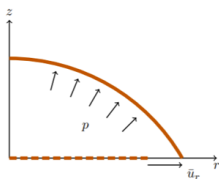
- Consider an axisymmetric Mooney-Rivlin hyperelastic incompressible material with an energy density function

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

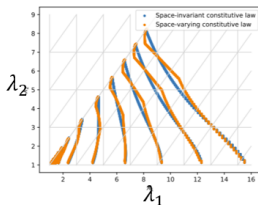
- The constitutive relations is modeled as

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

Here (P_1, P_2) is the stress tensor.

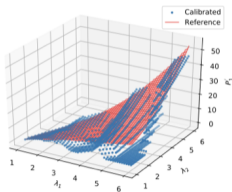


Rubber Membrane

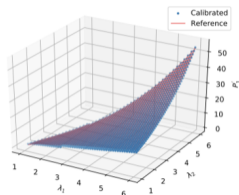


(λ_1, λ_2) Distribution

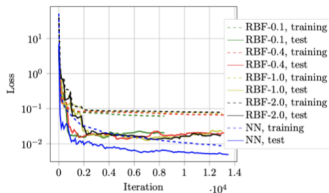
Comparison with Traditional Basis Functions



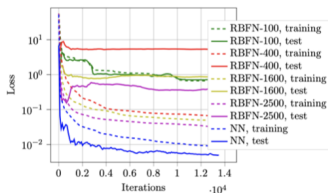
Piecewise Linear



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

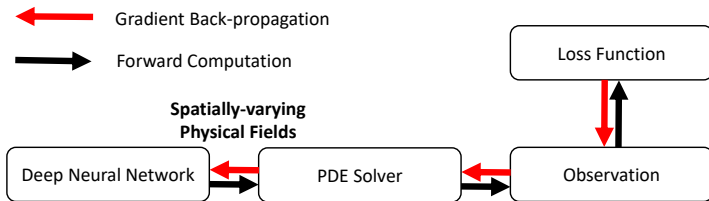
$$\min_u \psi = \frac{\mu}{2}(I_c - 2) - \frac{\mu}{2} \log(J) + \frac{\lambda}{8} \log(J)^2$$

where

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

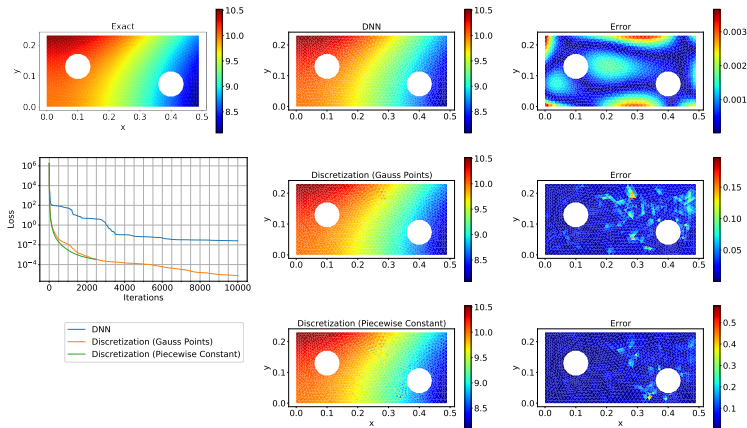
- Lamé parameters

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



Learning Spatially-varying fields

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



Viscoelasticity

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

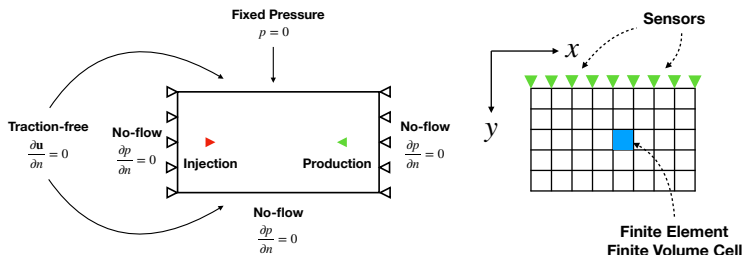
$$\operatorname{div} \boldsymbol{\sigma}(\mathbf{u}) - b \nabla p = 0$$

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

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

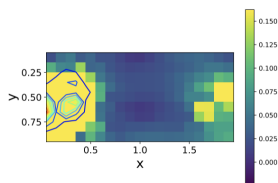
- Approximate the constitutive relation by a neural network

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

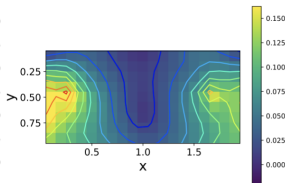


- Comparison with space varying linear elasticity approximation

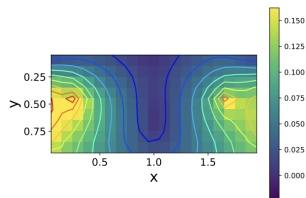
$$\sigma = H(x, y)\epsilon$$



Space Varying
Linear Elasticity

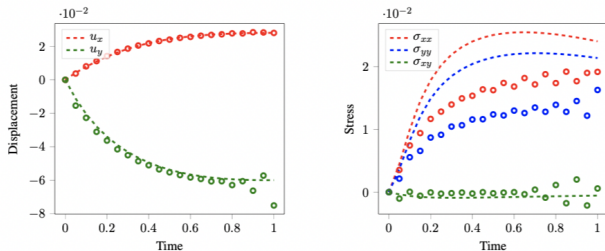


NN

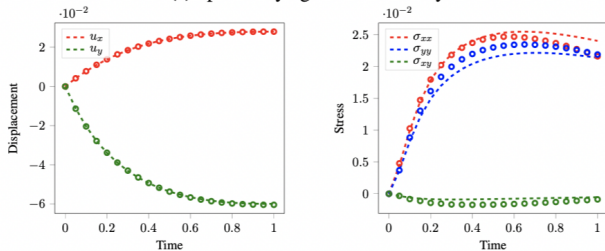


True

Viscoelasticity



(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

$$\nabla \cdot (\theta \nabla u(x)) = 0 \quad \mathcal{BC}(u(x)) = 0 \quad (4)$$

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

Expression	Description	ADCME Solution	Note
$\nabla \cdot (c \nabla u(x)) = 0$	Parameter Inverse Problem	Discrete Adjoint State Method	c is the minimizer of the error functional
$\nabla \cdot (f(x) \nabla u(x)) = 0$	Function Inverse Problem	Neural Network Functional Approximator	$f(x) \approx \mathcal{NN}_w(x)$
$\nabla \cdot (f(u) \nabla u(x)) = 0$	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

