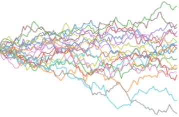
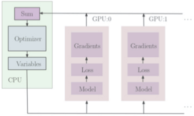
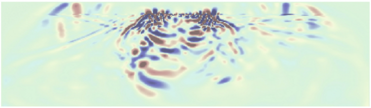
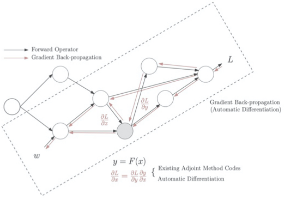
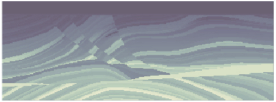
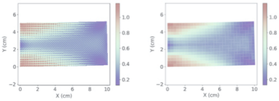
ADCME.jl
Physics Based Machine Learning for Inverse Problems

Kailai Xu (许开来)
https://github.com/kailaix/ADCME.jl

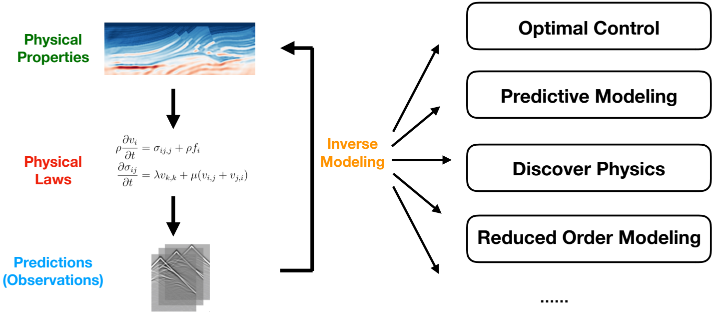



This work (ADCME.jl and its ecosystem) is a result of collective efforts of many of my Ph.D. collaborators together with our faculty advisors. In chronological order they are:

- Ph.D. collaborators: Daniel (Zhengyu) Huang, Dongzhuo Li, Weiqiang Zhu, and Tiffany (Li) Fan.
- Faculty supervisors: Eric Darve, Charbel Farhat, Jerry M. Harris, and Gregory C. Beroza.
- Many other fellow researchers from Julia language and scientific computing communities, who provide valuable inputs.

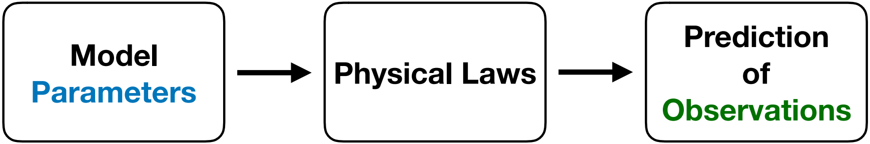
Outline

- **Inverse modeling** (逆建模) 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.

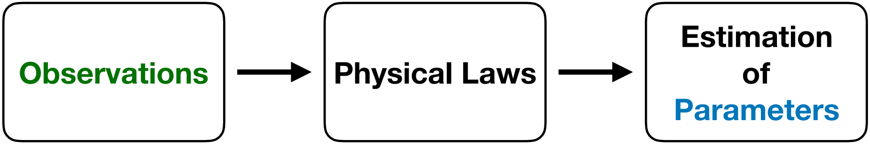


Inverse Modeling

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

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

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

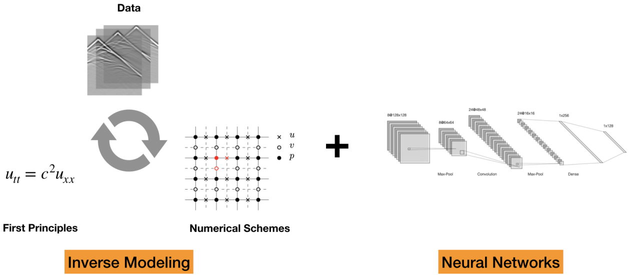
- Koopman operator in dynamical systems.
- Constitutive relations in solid mechanics.
- Turbulent closure relations in fluid mechanics.
- ...

The candidate solution space is infinite dimensional.

Physics Based Machine Learning

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

- Deep neural networks exhibit capability of approximating high dimensional and complicated functions.
- **Physics based machine learning**: 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.



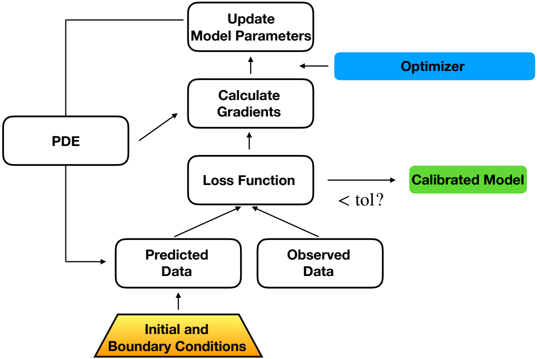
Outline

Gradient Based Optimization

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

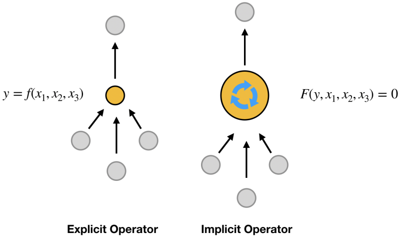
- We can now apply a gradient-based optimization method to (1).
- The key is to calculate the gradient descent direction g^k

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



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 ⇒ Physics Constrained Learning (PCL)



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

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

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

- 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

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

Scientific Machine Learning Inverse Modeling Toolkit

ADCME

High Performance

Solves large-scale problems with TensorFlow backend and MPI-based distributed optimization for scientific computing.

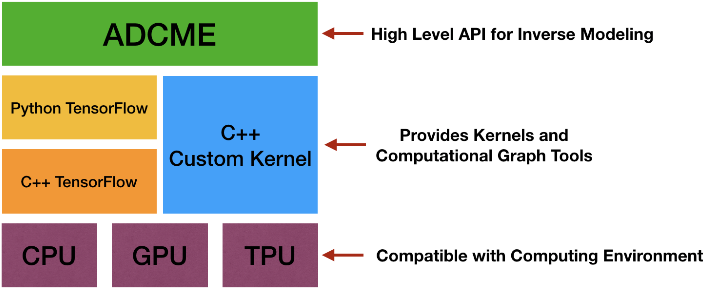
Easy to Use

Provides high-level syntax, which is compatible with Julia syntax, for implementing numerical schemes.

Broad Applicability

Constructs multiple physical models using toolkits from ADCME ecosystem and extends capabilities by custom

ADCME Architecture

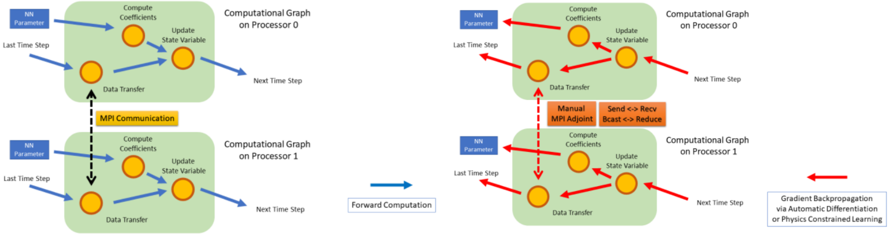


Targeting at scientific computing:

- Sparse linear algebra;
- MPI-based distributed computing;
- Domain specific numerical schemes: seismic inversion (ADSeismic.jl), fluid dynamics (FwiFlow.jl), geomechanics (PoreFlow.jl), solid mechanics (NNFEM.jl), . . .

Distributed Optimization

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



Estimating Coefficients from Data using ADCME

$$-bu''(x) + u(x) = f(x), \quad x \in [0,1], \quad u(0) = u(1) = 0$$
$$f(x) = 8 + 4x - 4x^2$$

- Data: $u(0.5) = 1$
- Finite difference:

$$-b\frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} + u_i = f(x_i)$$
$$\mathbf{B}\mathbf{u} = \mathbf{f}$$

1. Compute u_i as a function of b;
2. Minimize $(u_k - u(0.5))^2$, here $x_k = 0.5$.

Estimating Coefficients from Data using ADCME

```julia
using LinearAlgebra
using ADCME

n = 101 # number of grid nodes in [0,1]
h = 1/(n-1)
x = LinRange(0,1,n)[2:end-1]

b = Variable(10.0)
A = diagm(0=>2/h^2*ones(n-2),
          -1=>-1/h^2*ones(n-3), 1=>-1/h^2*ones(n-3))
B = b*A + I  # I stands for the identity matrix
f = @. 4*(2 + x - x^2)
u = B\f # solve the equation using built-in linear solver
ue = u[div(n+1,2)] # extract values at x=0.5
loss = (ue-1.0)^2

# Optimization
sess = Session(); init(sess)
BFGS!(sess, loss)
```

Domain Specific Numerical Schemes

$$-\nabla\cdot(\kappa\nabla u) = f, \qquad u|_{\partial\Omega} = 0$$

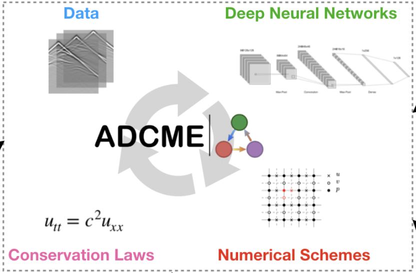
- Weak form

$$\int_\Omega \kappa\nabla u\cdot\nabla v d\mathbf{x} = \int_\Omega f v d\mathbf{x}$$

- The variational problem is transcribed into numerical simulation using domain specific implementations from PoreFlow.jl:

```
A = constant(compute_fem_laplace_matrix1(kappa, m, n, h))
F = eval_f_on_gauss_pts(f, m, n, h)
bd = bcnode("all", m, n, h)
A, _ = fem_impose_Dirichlet_boundary_condition1(A, bd,
                                                 m, n, h)
rhs = compute_fem_source_term1(F, m, n, h)
rhs[bd] .= 0.0
sol = A\rhs
```

A General Approach to Inverse Modeling



PoreFlow.jl
Geomechanics
Viscoelasticity
Multiphase Flow
Multiphysics
* coming soon

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

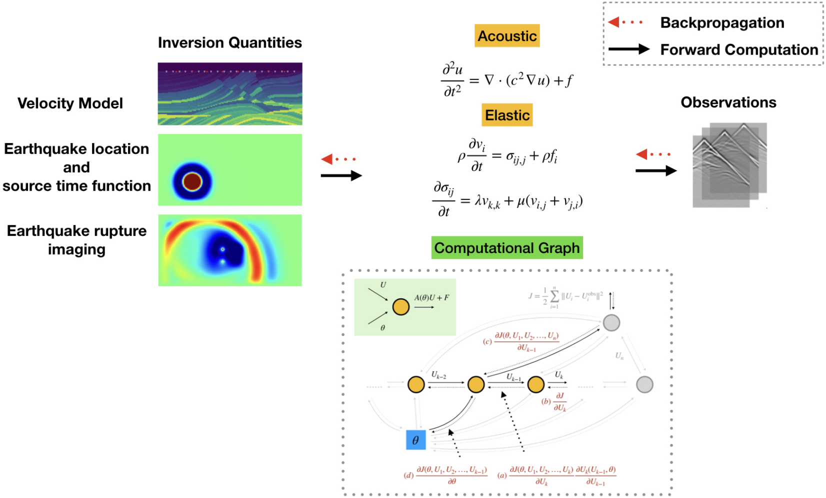
ADSeismic.jl
General Seismic Inversion
https://github.com/kailaix/ADSeismic.jl

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

Outline

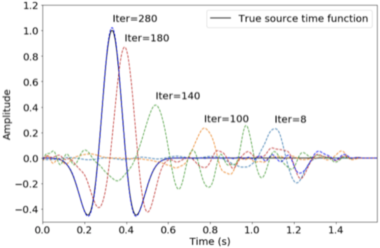
ADSeismic.jl: A General Approach to Seismic Inversion

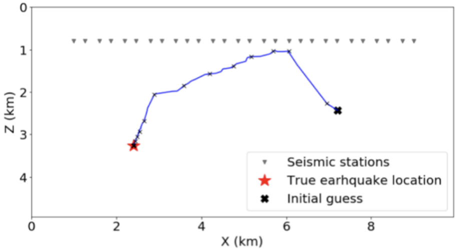
- Many seismic inversion problems can be solved within a unified framework.



- The earthquake source function is parameterized by ($g(t)$ and x_0 are unknowns)

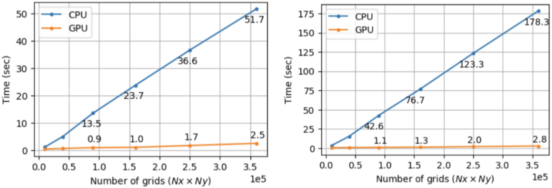
$$f(x,t) = \frac{g(t)}{2\pi\sigma^2}\exp\left(-\frac{||x-x_0||^2}{2\sigma^2}\right)$$



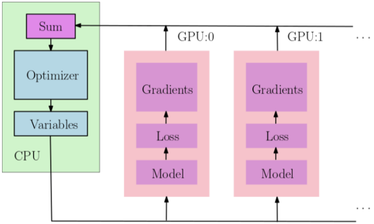


ADSeismic.jl: Benchmark

- ADCME makes the heterogeneous computation capability of TensorFlow available for scientific computing.



(a) (b)

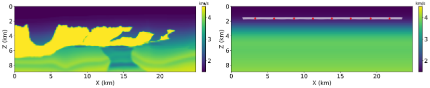


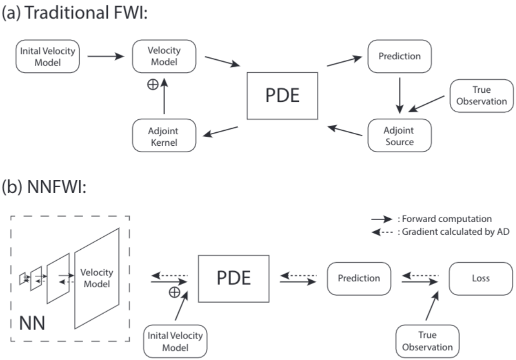
(c)

NNFWI: Neural-network-based Full-Waveform Inversion

- Estimate velocity models from seismic observations.

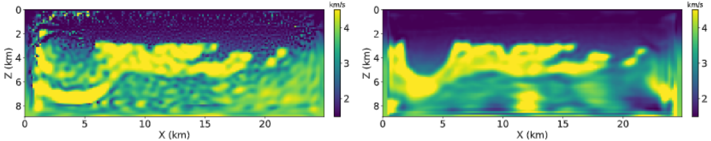
$$\frac{\partial^2 u}{\partial t^2} = \nabla\cdot({\color{red}m}^2\nabla u)+f$$



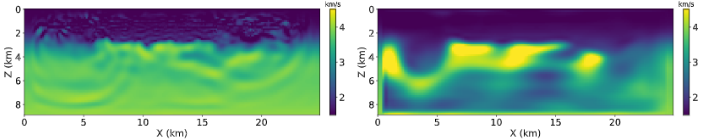


NNFWI: Neural-network-based Full-Waveform Inversion

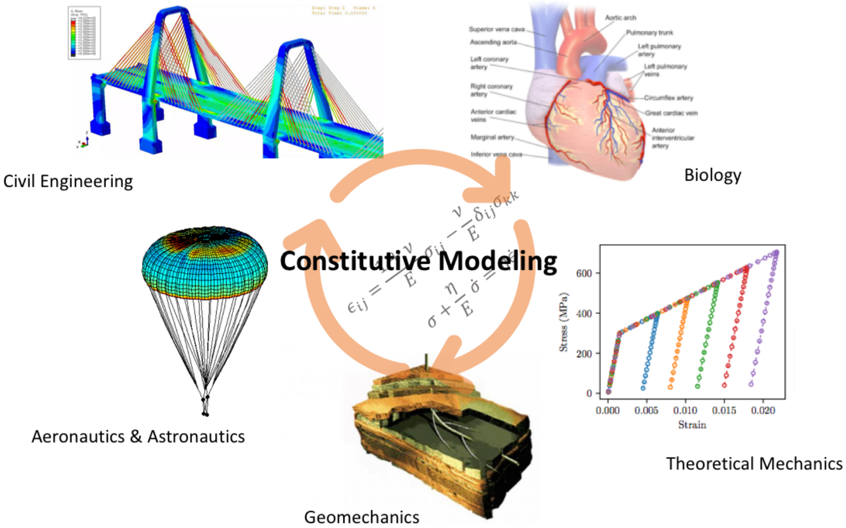
- Inversion results with a noise level $\sigma = \sigma_0$



- Inversion results for the same loss function value:



Constitutive Modeling



NNFEM.jl: Constitutive Modeling

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

- **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 (1) and (2).

NNFEM.jl: Robust Constitutive Modeling

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

NNFEM.jl: Robust Constitutive Modeling

- Weak form of balance equations of linear momentum

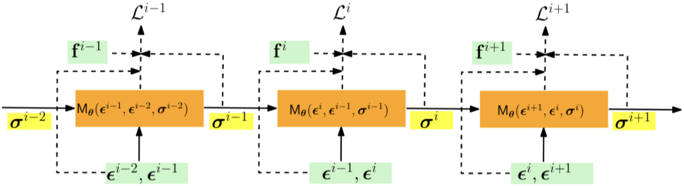
$$P_i(\theta) = \int_V \rho \ddot{u}_i \delta u_i dVt + \int_V \underbrace{\sigma_{ij}(\theta)}_{\text{embedded neural network}} \delta\varepsilon_{ij} dV$$

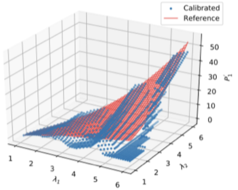
$$F_i = \int_V \rho b_i \delta u_i dV + \int_{\partial V} t_i \delta u_i dS$$

- Train the neural network by

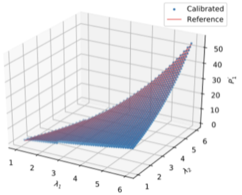
$$\boxed{L(\theta) = \min_\theta \sum_{i=1}^N (P_i(\theta) - F_i)^2}$$

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

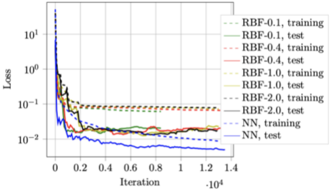




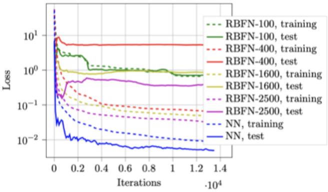
Piecewise Linear



Neural Network



Radial Basis Functions vs. Neural Network

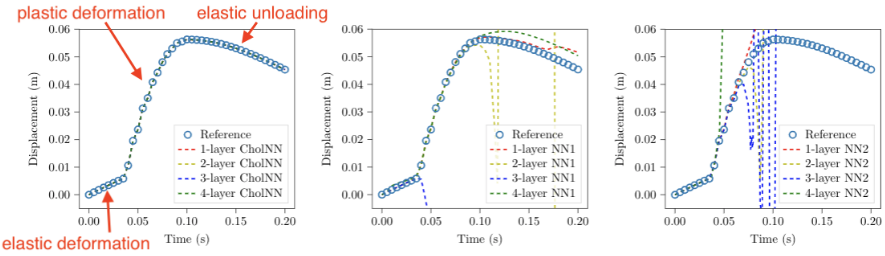


Radial Basis Function Networks vs. Neural Network

NNFEM.jl: Robust Constitutive Modeling

- Comparison of different neural network architectures

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



PoreFlow.jl: FEM/FVM on Structured Grids

- Steady-state Navier-Stokes equation

$$
\begin{aligned}
(\mathbf{u}\cdot\nabla)\mathbf{u} &= -\frac{1}{\rho}\nabla p + \nabla\cdot(\nu\nabla\mathbf{u}) + \mathbf{g}\\
\nabla\cdot\mathbf{u} &= 0
\end{aligned}
$$

- Inverse problem are ubiquitous in fluid dynamics:

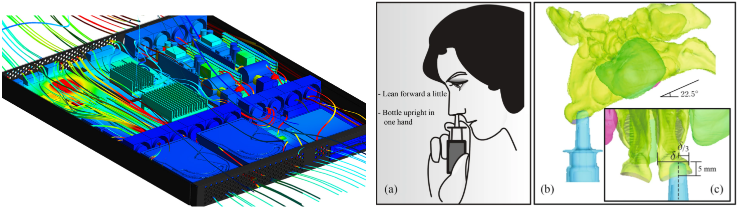
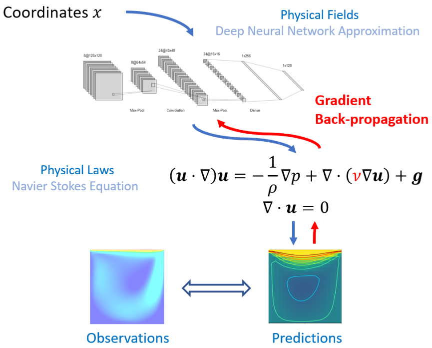
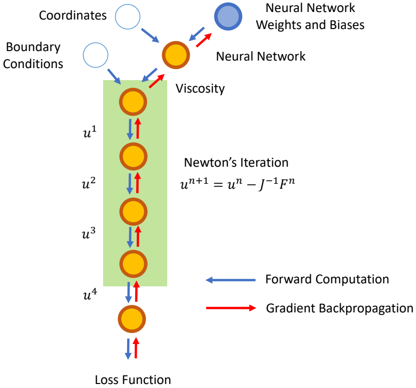


图: Left: electronic cooling; right: nasal drug delivery.

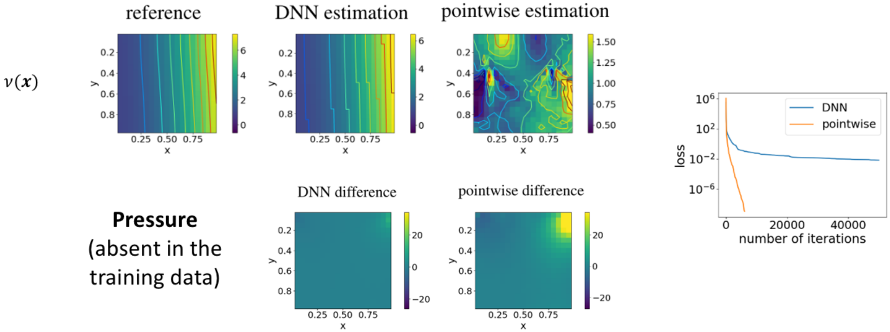
PoreFlow.jl: FEM/FVM on Structure Grids





PoreFlow.jl: FEM/FVM on Structure Grids

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



Outline

A Parameter/Function Learning View of Inverse Modeling

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

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

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	$f(x) \approx f_w(x)$
$\nabla \cdot (f(u)\nabla u(x)) = 0$	Relation Inverse Problem	Residual Learning Physics Constrained Learning	$f(u) \approx f_w(u)$
$\nabla \cdot (\varpi \nabla u(x)) = 0$	Stochastic Inverse Problem	Generative Neural Networks	$\varpi = f_w(v_{\text{latent}})$

Scopes, Challenges, and Future Work

Physics based Machine Learning: an innovative approach to inverse modeling.

1. Deep neural networks provide a novel function approximator that outperforms traditional basis functions in certain scenarios.
2. Numerical PDEs are not on the opposite side of machine learning. By expressing the known physical constraints using numerical schemes and approximating the unknown with machine learning models, we combine the best of the two worlds, leading to efficient and accurate inverse modeling tools.

Automatic Differentiation: the core technique of physics based machine learning.

1. The AD technique is not new; it has existed for several decades and many software exists.
2. The advent of deep learning drives the development of robust, scalable and flexible AD software that leverages the high performance computing environment.
3. As deep learning techniques continue to grow, crafting the tool to incorporate machine learning and AD techniques for inverse modeling is beneficial in scientific computing.
4. However, AD is not a panacea. Many scientific computing algorithms cannot be directly expressed by composition of differentiable operators.

A General Approach to Inverse Modeling