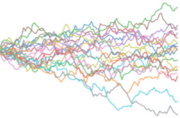
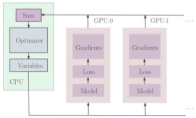
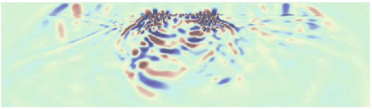
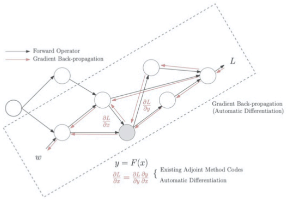
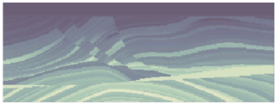
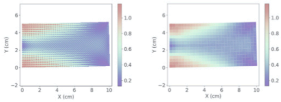
Physics Based Machine Learning for Inverse Problems

Kailai Xu and Eric Darve

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

⋆ The Pathway to Physics Based Machine Learning ⋆

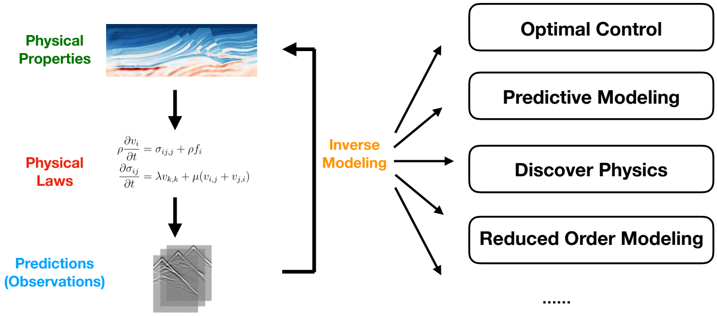





Forward Operator
Gradient Back-propagation
L
$\frac{\partial L}{\partial y}$
$\frac{\partial L}{\partial x}$
w
Gradient Back-propagation (Automatic Differentiation)
$y = F(x)$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$
Existing Adjoint Method Codes
Automatic Differentiation



Outline

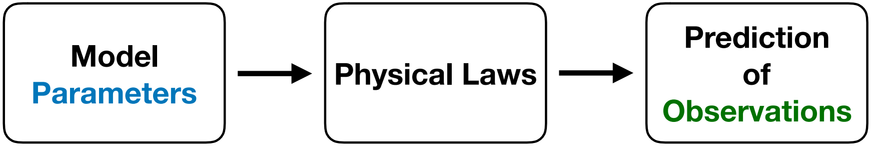
Inverse Modeling

- **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

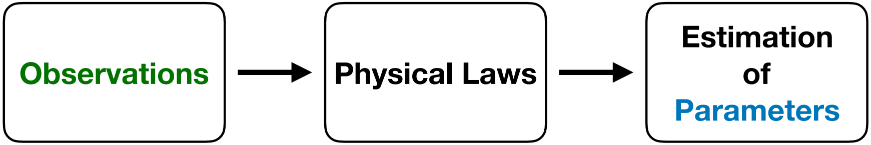

Physical Properties
Physical Laws
$\rho\frac{\partial v_i}{\partial t} = \sigma_{ij,j} + \rho f_i$
$\frac{\partial \sigma_{ij}}{\partial t} = \lambda v_{k,k} + \mu(v_{i,j} + v_{j,i})$
Predictions (Observations)
Inverse Modeling
Optimal Control
Predictive Modeling
Discover Physics
Reduced Order Modeling
......

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?

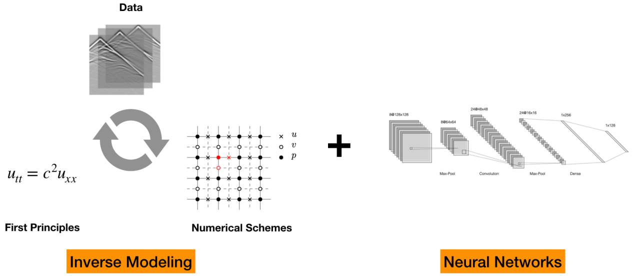
- Koopman operator in dynamical systems.
- Constitutive relations in solid mechanics.
- Turbulent closure relations in fluid mechanics.
- ...

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.

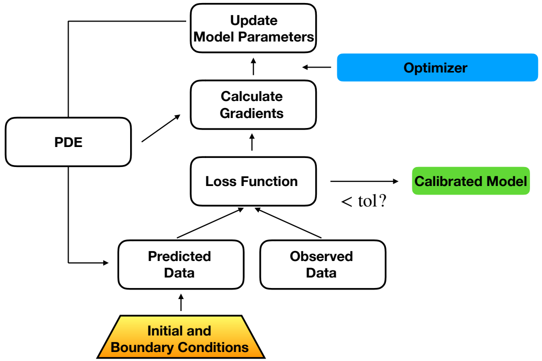


Gradient Based Optimization

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

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

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



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.

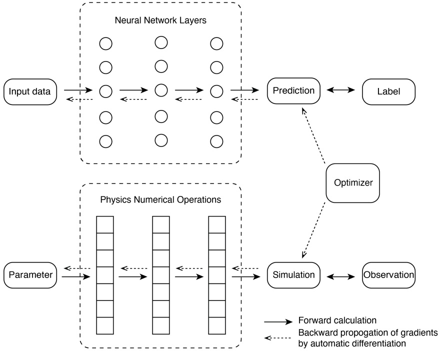
Back-propagation

||

Reverse-mode Automatic Differentiation

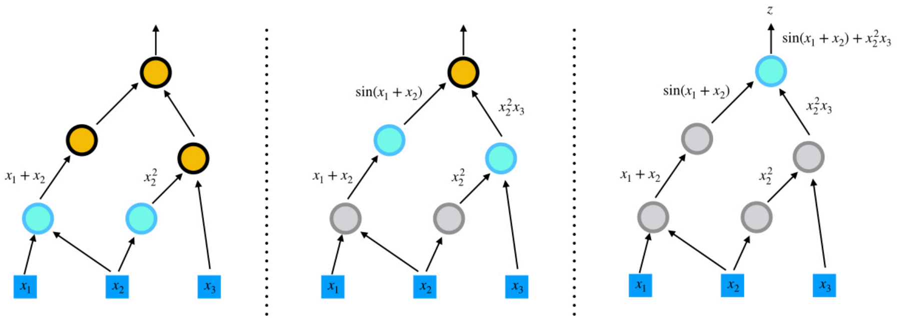
||

Discrete Adjoint-State Method



Automatic Differentiation: Computational Graph

- A computational graph is a functional description of the required computation. In the computational graph, an edge represents data, such as a scalar, a vector, a matrix or a tensor. A node represents a function (operator) whose input arguments are the the incoming edges and output values are are the outcoming edges.
- How to build a computational graph for $z = \sin(x_1 + x_2) + x_2^2 x_3$?

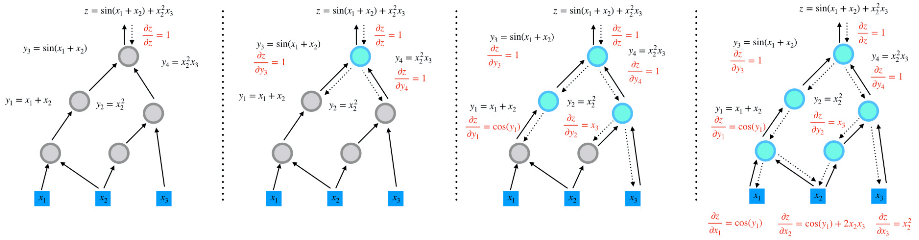


Automatic Differentiation: Reverse-Mode

- The reverse-mode automatic differentiation relies on the chain rule

$$\frac{\partial f \circ g(x)}{\partial x} = \frac{\partial f' \circ g(x)}{\partial g}\frac{\partial g'(x)}{\partial x}$$

- The reverse-mode automatic differentiation stores all intermediate variables in the forward computation.
- Let's see how to compute $\frac{\partial z}{\partial x_i}$, $i = 1, 2, 3$ for $z = \sin(x_1 + x_2) + x_2^2 x_3$

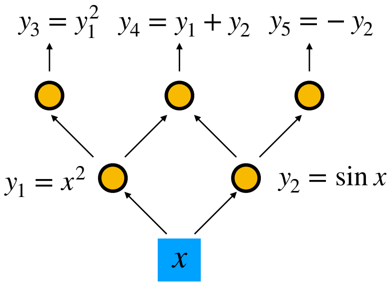

$z = \sin(x_1+x_2)+x_2^2x_3$
$\frac{\partial z}{\partial z} = 1$
$y_3 = \sin(x_1+x_2)$
$y_4 = x_2^2x_3$
$y_1 = x_1+x_2$
$y_2 = x_2^2$
x_1
x_2
x_3
$\frac{\partial z}{\partial y_3} = 1$
$\frac{\partial z}{\partial y_4} = 1$
$\frac{\partial z}{\partial y_1} = \cos(y_1)$
$\frac{\partial z}{\partial y_2} = x_3$
$\frac{\partial z}{\partial x_1} = \cos(y_1)$
$\frac{\partial z}{\partial x_2} = \cos(y_1)+2x_2x_3$
$\frac{\partial z}{\partial x_3} = x_2^2$

Automatic Differentiation: Forward-Mode

- The forward-mode automatic differentiation also uses the chain rule to propagate the gradients.

$$\frac{\partial f \circ g(x)}{\partial x} = \frac{\partial f' \circ g(x)}{\partial g} \frac{\partial g'(x)}{\partial x}$$

- In the forward-mode automatic differentiation, at every stage we evaluate the operator as well as its gradient.



$$(y_1, y_1') = (x^2, 2x)$$
$$(y_2, y_2') = (\sin x, \cos x)$$

$$(y_3, y_3') = (y_1^2, 2y_1y_1') = (x^4, 4x^3)$$
$$(y_4, y_4') = (y_1 + y_1, y_1' + y_2') = (x^2 + \sin x, 2x + \cos x)$$
$$(y_5, y_5') = (-y_2, -y_2') = (-\sin x, -\cos x)$$

Automatic Differentiation: Comparison

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

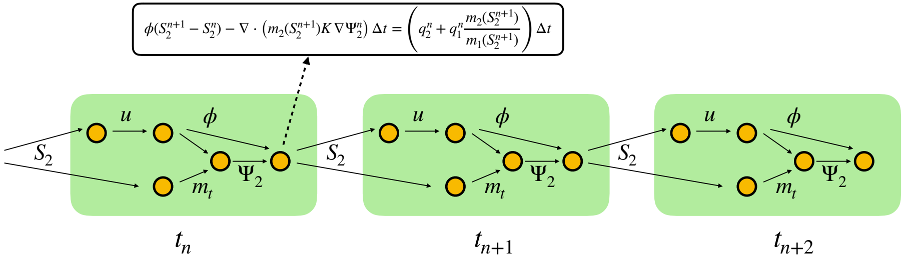
Mode	Suitable for ...	Complexity[1]	Application
Forward	$m \gg n$	$\leq 2.5\ \mathrm{OPS}(f(x))$	UQ
Reverse	$m \ll n$	$\leq 4\ \mathrm{OPS}(f(x))$	Inverse Modeling

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.



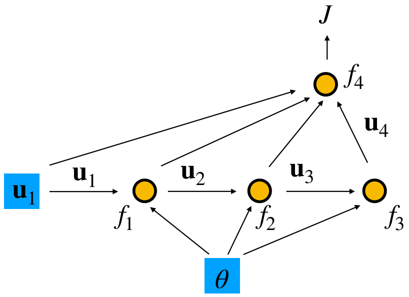
$$\boxed{\phi(S_2^{n+1}-S_2^n) - \nabla\cdot\left(m_2(S_2^{n+1})K\nabla\Psi_2^n\right)\Delta t = \left(q_2^n + q_1^n\frac{m_2(S_2^{n+1})}{m_1(S_2^{n+1})}\right)\Delta t}$$

The Relationship between reverse-mode Automatic Differentiation and KKT Condition

Consider a concrete PDE-constrained optimization problem:

$$\begin{aligned}\min_{\mathbf{u}_1,\boldsymbol{\theta}}\ & J = f_4(\mathbf{u}_1,\mathbf{u}_2,\mathbf{u}_3,\mathbf{u}_4),\\ \text{s.t. } & \mathbf{u}_2 = f_1(\mathbf{u}_1,\boldsymbol{\theta}),\\ & \mathbf{u}_3 = f_2(\mathbf{u}_2,\boldsymbol{\theta}),\\ & \mathbf{u}_4 = f_3(\mathbf{u}_3,\boldsymbol{\theta}).\end{aligned}$$

- f_1, f_2, f_3 are PDE constraints
- f_4 is the loss function
- $\mathbf{u}_1$ is the initial condition
- $\boldsymbol{\theta}$ is the model parameter



The Relationship between reverse-mode Automatic Differentiation and KKT Condition

Solving the constrained optimization method using adjoint-state methods:

- The Lagrange multiplier is

$$\mathcal{L} = f_4(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4) + \boldsymbol{\lambda}_2^T(f_1(\mathbf{u}_1, \boldsymbol{\theta}) - \mathbf{u}_2) + \boldsymbol{\lambda}_3^T(f_2(\mathbf{u}_2, \boldsymbol{\theta}) - \mathbf{u}_3) + \boldsymbol{\lambda}_4^T(f_3(\mathbf{u}_3, \boldsymbol{\theta}) - \mathbf{u}_4)$$

- Therefore, the first order KKT condition of the constrained PDE system is

$$\begin{aligned}
\boldsymbol{\lambda}_4^T &= \frac{\partial f_4}{\partial \mathbf{u}_4}\\
\boldsymbol{\lambda}_3^T &= \frac{\partial f_4}{\partial \mathbf{u}_3} + \boldsymbol{\lambda}_4^T \frac{\partial f_3}{\partial \mathbf{u}_3}\\
\boldsymbol{\lambda}_2^T &= \frac{\partial f_4}{\partial \mathbf{u}_2} + \boldsymbol{\lambda}_3^T \frac{\partial f_2}{\partial \mathbf{u}_2}\\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} &= \boldsymbol{\lambda}_2^T \frac{\partial f_1}{\partial \boldsymbol{\theta}} + \boldsymbol{\lambda}_3^T \frac{\partial f_2}{\partial \boldsymbol{\theta}} + \boldsymbol{\lambda}_4^T \frac{\partial f_3}{\partial \boldsymbol{\theta}} \Rightarrow \text{Sensitivity } \frac{\partial J}{\partial \boldsymbol{\theta}}
\end{aligned}$$

The Relationship between reverse-mode Automatic Differentiation and KKT Condition

How do we implement reverse-mode automatic differentiation for computing the gradients?

- Consider the operator f_2, we need to implement two operators

$$\text{Forward: } \mathbf{u}_3 = f_2(\mathbf{u}_2, \boldsymbol{\theta})$$
$$\text{Backward: } \frac{\partial J}{\partial \mathbf{u}_2}, \frac{\partial J}{\partial \boldsymbol{\theta}} = b_2\left(\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}, \mathbf{u}_2, \boldsymbol{\theta}\right)$$

  $\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}$ is the "total" gradient $\mathbf{u}_3$ received from the downstream in the computational graph.

- The backward operator is implemented using the chain rule

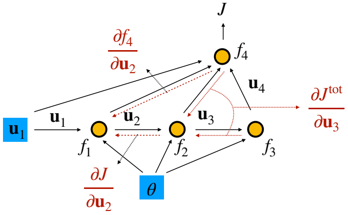
$$\frac{\partial J}{\partial \mathbf{u}_2} = \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}\frac{\partial f_2}{\partial \mathbf{u}_2} \qquad \frac{\partial J}{\partial \boldsymbol{\theta}} = \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}\frac{\partial f_2}{\partial \boldsymbol{\theta}}$$

What are $\frac{\partial J}{\partial \mathbf{u}_2}$, $\frac{\partial J}{\partial \boldsymbol{\theta}}$, and $\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}$ exactly?

The Relationship between reverse-mode Automatic Differentiation and KKT Condition

The total gradient $\mathbf{u}_2$ received is

$$\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_2} = \frac{\partial f_4}{\partial \mathbf{u}_2} + \frac{\partial J}{\partial \mathbf{u}_2} = \frac{\partial f_4}{\partial \mathbf{u}_2} + \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}\frac{\partial f_2}{\partial \mathbf{u}_2}$$



J
$\frac{\partial f_4}{\partial \mathbf{u}_2}$
f_4
$\mathbf{u}_4$
$\frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_3}$
$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
f_1
f_2
f_3
$\frac{\partial J}{\partial \mathbf{u}_2}$
θ

The dual constraint in the KKT condition

$$\boldsymbol{\lambda}_2^T = \frac{\partial f_4}{\partial \mathbf{u}_2} + \boldsymbol{\lambda}_3^T\frac{\partial f_2}{\partial \mathbf{u}_2}$$

The following equality can be verified

$$\boxed{\boldsymbol{\lambda}_i^T = \frac{\partial J^{\text{tot}}}{\partial \mathbf{u}_i}}$$

In general, the reverse-mode AD is back-propagating the Lagrange multiplier (adjoint variables).

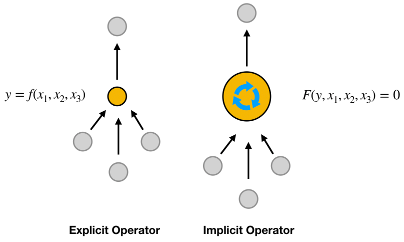
The Relationship between reverse-mode Automatic Differentiation and KKT Condition

- The well-established adjoint-state method is equivalent to solving the KKT system.
- The adjoint-state methods are challenging to implement, mainly due to the time-consuming and difficult process of deriving the gradients in a complex system.
- Using reverse-mode automatic differentiation is equivalent to solving the inverse modeling problem using discrete adjoint-state methods, but in a more manageable way.
- Computational graph based implementation also allows for automatic compilation time optimization and parallelization.

Outline

Challenges in AD

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



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

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

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 in the forward computation, we solve for $u_h = G_h(\theta)$ in $F_h(\theta, u_h) = 0$, 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

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

Step 1: Calculate w by solving a linear system (never invert the matrix!)

$$w^T = \underbrace{\frac{\partial L_h(u_h)}{\partial u_h}}_{1\times N}\underbrace{\left(\frac{\partial F_h}{\partial u_h}\Big|_{u_h=G_h(\theta)}\right)^{-1}}_{N\times N}$$

Step 2: Calculate the gradient by automatic differentiation

$$w^T\underbrace{\frac{\partial F_h}{\partial \theta}\Big|_{u_h=G_h(\theta)}}_{N\times p} = \frac{\partial (w^T\; F_h(\theta, u_h))}{\partial \theta}\Bigg|_{u_h=G_h(\theta)}$$

Physics Constrained Learning: Linear System

- Many physical simulations require solving a linear system

$$A(\theta_2)u_h = \theta_1$$

- The corresponding PDE constraint in our formulation is

$$F_h(\theta_1, \theta_2, u_h) = \theta_1 - A(\theta_2)u_h = 0$$

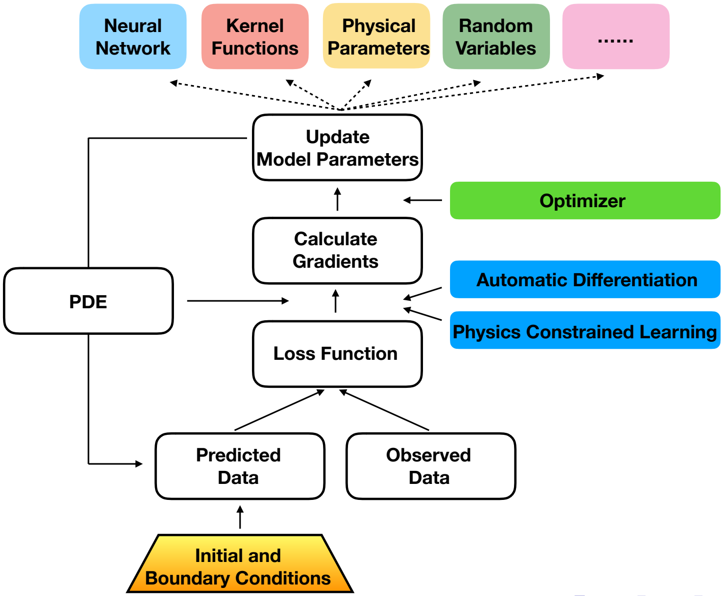
- The backpropagation formula

$$\begin{aligned} p &:= \frac{\partial \tilde{L}_h(\theta_1, \theta_2)}{\partial \theta_1} = \frac{\partial L_h(u_h)}{\partial u_h} A(\theta_2)^{-1} \\ q &:= \frac{\partial \tilde{L}_h(\theta_1, \theta_2)}{\partial \theta_2} = -\frac{\partial L_h(u_h)}{\partial u_h} A(\theta_2)^{-1} \frac{\partial A(\theta_2)}{\partial \theta_2} \end{aligned}$$

which is equivalent to

$$A^T p^T = \left(\frac{\partial L_h(u_h)}{\partial u_h}\right)^T \quad q = -p\frac{\partial A(\theta_2)}{\partial \theta_2}$$

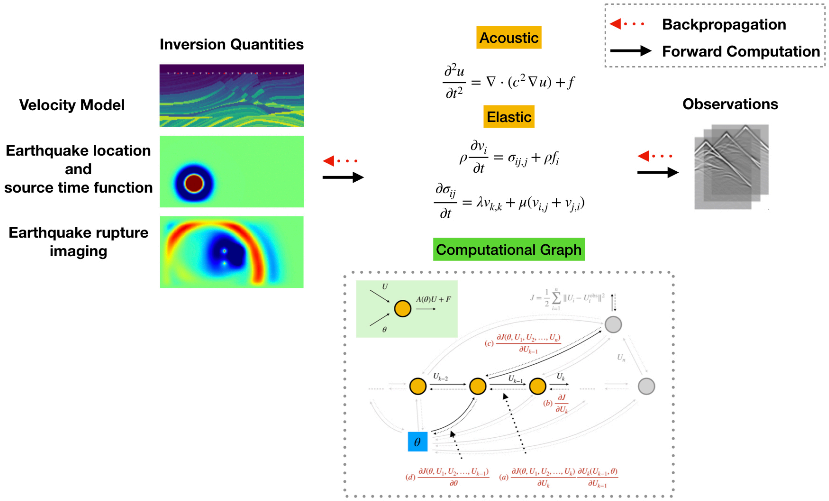
Methodology Summary



Outline

ADSeismic.jl: A General Approach to Seismic Inversion

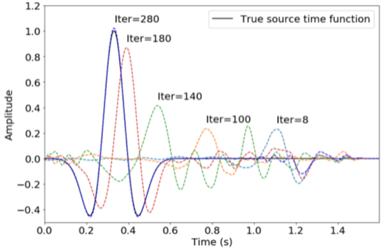
- Many seismic inversion problems can be solved within a unified framework.

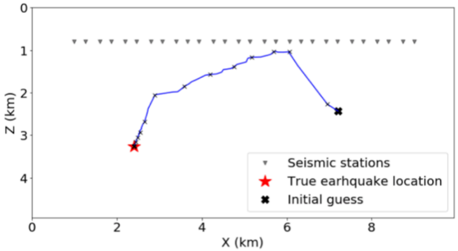

Backpropagation
Forward Computation
Inversion Quantities
Velocity Model
Earthquake location and source time function
Earthquake rupture imaging
Acoustic
$\frac{\partial^2 u}{\partial t^2} = \nabla \cdot (c^2 \nabla u) + f$
Elastic
$\rho \frac{\partial v_i}{\partial t} = \sigma_{ij,j} + \rho f_i$
$\frac{\partial \sigma_{ij}}{\partial t} = \lambda v_{k,k} + \mu(v_{i,j} + v_{j,i})$
Observations
Computational Graph
U
θ
$A(\theta)U + F$
$J = \frac{1}{2}\sum_{i=1}^{n} \|U_i - U_i^{\text{obs}}\|^2$
(c) $\frac{\partial J(\theta, U_1, U_2, \ldots, U_n)}{\partial U_{k-1}}$
U_n
U_{k-2}
U_{k-1}
U_k
(b) $\frac{\partial J}{\partial U_k}$
θ
(d) $\frac{\partial J(\theta, U_1, U_2, \ldots, U_{k-1})}{\partial \theta}$
(a) $\frac{\partial J(\theta, U_1, U_2, \ldots, U_k)}{\partial U_k} \frac{\partial U_k(U_{k-1}, \theta)}{\partial U_{k-1}}$

ADSeismic.jl: Earthquake Location Example

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

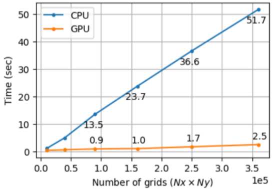
$$f(x,t) = \frac{g(t)}{2\pi\sigma^2}\exp\left(-\frac{||x-x_0||^2}{2\sigma^2}\right)$$



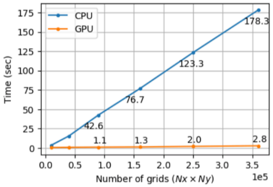


ADSeismic.jl: Benchmark

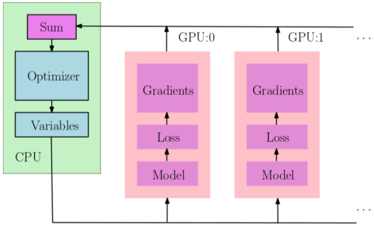
- ADCME makes the heterogeneous computation capability of TensorFlow available for scientific computing.



(a)



(b)



(c)

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: Robustic 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: Robustic Constitutive Modeling

- Weak form of balance equations of linear momentum

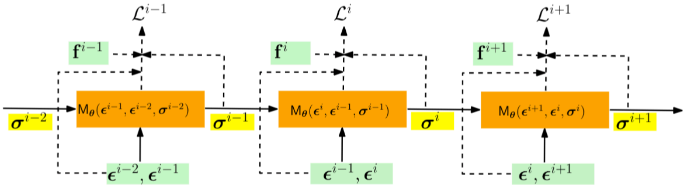
$$P_i(\theta) = \int_V \rho \ddot{u}_i \delta u_i dVt + \int_V \underbrace{\sigma_{ij}(\theta)}_{\text{embedded neural network}} \delta \varepsilon_{ij} dV$$

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

- Train the neural network by

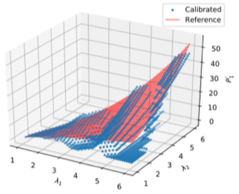
$$\boxed{L(\theta) = \min_\theta \sum_{i=1}^N (P_i(\theta) - F_i)^2}$$

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

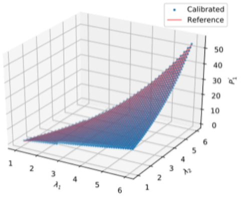


$\mathcal{L}^{i-1}$
$\mathcal{L}^{i}$
$\mathcal{L}^{i+1}$
$\mathbf{f}^{i-1}$
$\mathbf{f}^{i}$
$\mathbf{f}^{i+1}$
$M_\theta(\epsilon^{i-1}, \epsilon^{i-2}, \sigma^{i-2})$
$M_\theta(\epsilon^{i}, \epsilon^{i-1}, \sigma^{i-1})$
$M_\theta(\epsilon^{i+1}, \epsilon^{i}, \sigma^{i})$
σ^{i-2}
σ^{i-1}
σ^{i}
σ^{i+1}
$\epsilon^{i-2}, \epsilon^{i-1}$
$\epsilon^{i-1}, \epsilon^{i}$
$\epsilon^{i}, \epsilon^{i+1}$

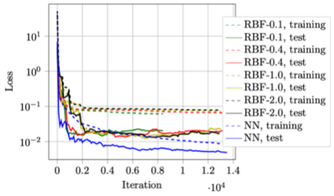
NNFEM.jl: Robustic Constitutive Modeling



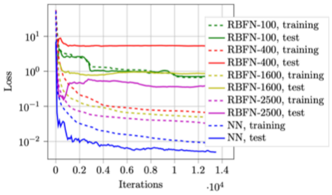
Piecewise Linear



Neural Network



Radial Basis Functions
vs.
Neural Network

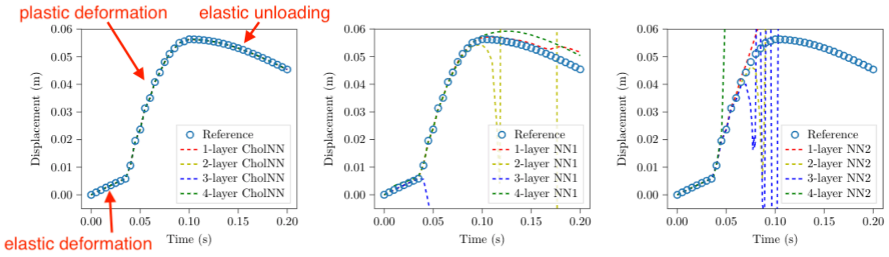


Radial Basis Function Networks
vs.
Neural Network

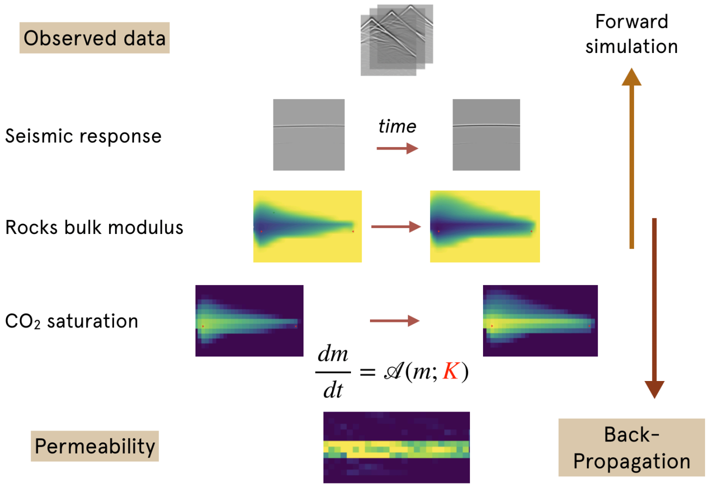
NNFEM.jl: Robustic Constitutive Modeling

- Comparison of different neural network architectures

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



FwiFlow.jl: Elastic Full Waveform Inversion for subsurface flow problems



FwiFlow.jl: Fully Nonlinear Implicit Schemes

- The governing equation is a nonlinear PDE

$$\frac{\partial}{\partial t}(\phi S_i \rho_i) + \nabla \cdot (\rho_i \mathbf{v}_i) = \rho_i q_i, \quad i = 1, 2$$

$$S_1 + S_2 = 1$$

$$\mathbf{v}_i = -\frac{K k_{ri}}{\tilde{\mu}_i}(\nabla P_i - g\rho_i \nabla Z), \quad i = 1, 2$$

$$k_{r1}(S_1) = \frac{k_{r1}^o S_1^{L_1}}{S_1^{L_1} + E_1 S_2^{T_1}}$$

$$k_{r2}(S_1) = \frac{S_2^{L_2}}{S_2^{L_2} + E_2 S_1^{T_2}}$$

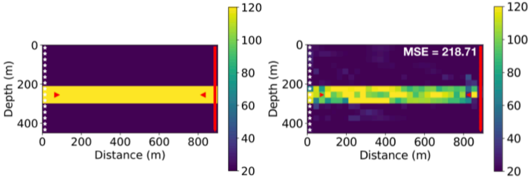
- For stability and efficiency, implicit methods are the industrial standards.

$$\phi(S_2^{n+1} - S_2^n) - \nabla \cdot \left(m_2(S_2^{n+1}) K \nabla \Psi_2^n\right) \Delta t = \left(q_2^n + q_1^n \frac{m_2(S_2^{n+1})}{m_1(S_2^{n+1})}\right) \Delta t \quad m_i(s) = \frac{k_{ri}(s)}{\tilde{\mu}_i}$$

- It is impossible to express the numerical scheme directly in an AD framework. Physics constrained learning is used to enhance the AD framework for computing gradients.

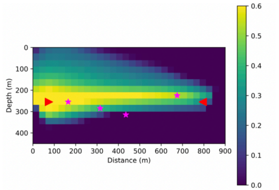
FwiFlow.jl: Showcase

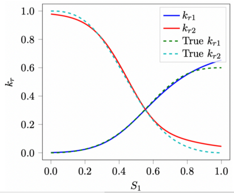
- Task 1: Estimating the permeability from seismic data
  B.C. + Two-Phase Flow Equation + Wave Equation $\Rightarrow$ Seismic Data



- Task 2: Learning the rock physics model from sparse saturation data. The rock physics model is approximated by neural networks

$$f_1(S_1;\theta_1) \approx k_{r1}(S_1) \qquad f_2(S_1;\theta_2) \approx k_{r2}(S_1)$$





FwiFlow.jl: Showcase

- Task 3: Learning the nonlocal (space or time) hidden dynamics from seismic data. This is very challenging using traditional methods (e.g., the adjoint-state method) because the dynamics is history dependent.

B.C. + Time-/Space-fractional PDE + Wave Equation ⇒ Seismic Data

Governing Equation	$\sigma = 0$	$\sigma = 5$
${}_0^C D_t^{\mathbf{0.8}} m = 10\Delta m$	$a/a^* = 1.0000$ $\alpha = \mathbf{0.8000}$	$a/a^* = 0.9109$ $\alpha = \mathbf{0.7993}$
${}_0^C D_t^{\mathbf{0.2}} m = 10\Delta m$	$a/a^* = 0.9994$ $\alpha = \mathbf{0.2000}$	$a/a^* = 0.3474$ $\alpha = \mathbf{0.1826}$
$\frac{\partial m}{\partial t} = -10(-\Delta)^{\mathbf{0.2}} m$	$a/a^* = 1.0000$ $s = \mathbf{0.2000}$	$a/a^* = 1.0378$ $s = \mathbf{0.2069}$
$\frac{\partial m}{\partial t} = -10(-\Delta)^{\mathbf{0.8}} m$	$a/a^* = 1.0000$ $s = \mathbf{0.8000}$	$a/a^* = 1.0365$ $s = \mathbf{0.8093}$

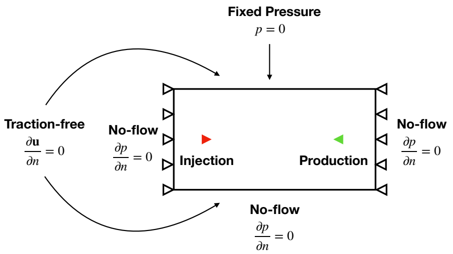
PoreFlow.jl: Inverse Modeling of Viscoelasticity

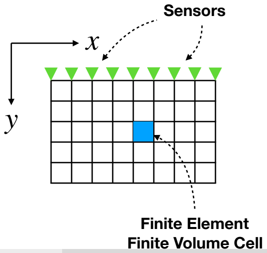
- Multi-physics Interaction of Coupled Geomechanics and Multi-Phase Flow Equations

$$\begin{aligned} \mathrm{div}\boldsymbol{\sigma}(\mathbf{u}) - b\nabla p &= 0 \\ \frac{1}{M}\frac{\partial p}{\partial t} + b\frac{\partial \epsilon_v(\mathbf{u})}{\partial t} - \nabla\cdot\left(\frac{k}{B_f\mu}\nabla p\right) &= f(x,t) \\ \boldsymbol{\sigma} &= \boldsymbol{\sigma}(\boldsymbol{\epsilon}, \dot{\boldsymbol{\epsilon}}) \end{aligned}$$

- Approximate the constitutive relation by a neural network

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

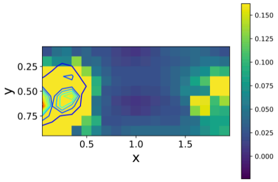

Fixed Pressure
$p = 0$
Traction-free
$\frac{\partial \mathbf{u}}{\partial n} = 0$
No-flow
$\frac{\partial p}{\partial n} = 0$
Injection
Production
No-flow
$\frac{\partial p}{\partial n} = 0$
No-flow
$\frac{\partial p}{\partial n} = 0$



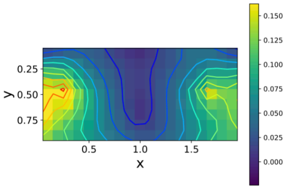
PoreFlow.jl: Inverse Modeling of Viscoelasticity

- Comparison with space varying linear elasticity approximation

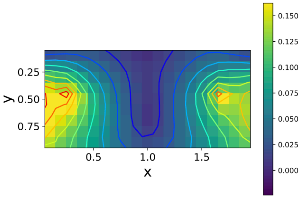
$$\boldsymbol{\sigma} = H(x, y)\epsilon \tag{4}$$



Space Varying Linear Elasticity

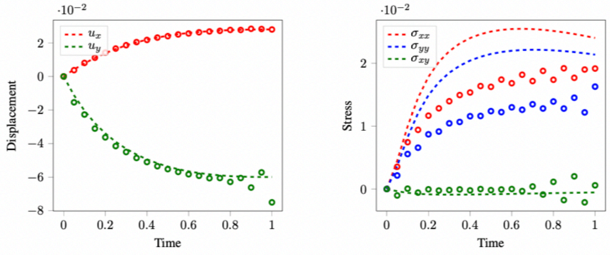


NN

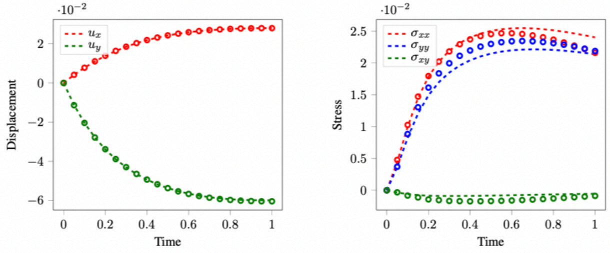


True

PoreFlow.jl: Inverse Modeling of Viscoelasticity



(a) Space Varying Linear Elasticity



(b) NN-based Viscoelasticity

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

$$\nabla \cdot (\theta \nabla u(x)) = 0 \quad \mathcal{BC}(u(x)) = 0 \tag{5}$$

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.

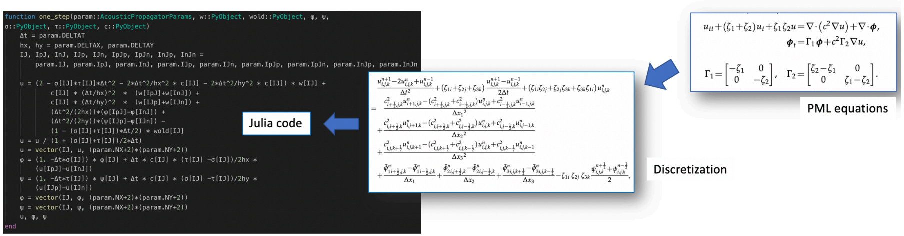
Automatic Differentiation: the core technique of physics based machine learning.

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.

ADCME

- ADCME is the materialization of the physics based machine learning concept.
- ADCME allows users to use high performance and mathematical friendly programming language Julia to implement numerical schemes, and obtain the comprehensive automatic differentiation functionality, heterogeneous computing capability, parallelism and scalability provided by the TensorFlow backend.

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



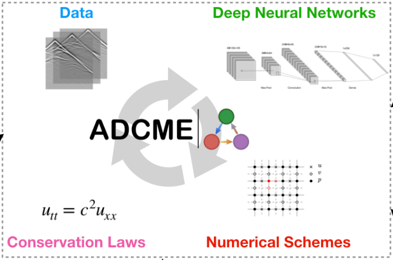
```julia
function one_step(param::AcousticPropagatorParams, w::PyObject, wold::PyObject, φ, ψ,
σ::PyObject, τ::PyObject, c::PyObject)
    Δt = param.DELTAT
    hx, hy = param.DELTAX, param.DELTAY
    IJ, IpJ, InJ, IJp, IJn, IpJp, IpJn, InJp, InJn =
        param.IJ, param.IpJ, param.InJ, param.IJp, param.IJn, param.IpJp, param.IpJn, param.InJp, param.InJn

    u = (2 - σ[IJ]*τ[IJ]*Δt^2 - 2*Δt^2/hx^2 * c[IJ] - 2*Δt^2/hy^2 * c[IJ]) * w[IJ] +
            c[IJ] * (Δt/hx)^2  *  (w[IpJ]+w[InJ]) +
            c[IJ] * (Δt/hy)^2  *  (w[IJp]+w[IJn]) +
            (Δt^2/(2hx))*(φ[IpJ]-φ[InJ]) +
            (Δt^2/(2hy))*(ψ[IJp]-ψ[IJn]) -
            (1 - (σ[IJ]+τ[IJ])*Δt/2) * wold[IJ]
    u = u / (1 + (σ[IJ]+τ[IJ])/2*Δt)
    u = vector(IJ, u, (param.NX+2)*(param.NY+2))
    φ = (1. -Δt*σ[IJ]) * φ[IJ] + Δt * c[IJ] * (τ[IJ] -σ[IJ])/2hx *
        (u[IpJ]-u[InJ])
    ψ = (1. -Δt*τ[IJ]) * ψ[IJ] + Δt * c[IJ] * (σ[IJ] -τ[IJ])/2hy *
        (u[IJp]-u[IJn])
    φ = vector(IJ, φ, (param.NX+2)*(param.NY+2))
    ψ = vector(IJ, ψ, (param.NX+2)*(param.NY+2))
    u, φ, ψ
end
```

$$\begin{aligned}
&\frac{u_{i,j,k}^{n+1}-2u_{i,j,k}^{n}+u_{i,j,k}^{n-1}}{\Delta t^2}+(\zeta_{1i}+\zeta_{2j}+\zeta_{3k})\frac{u_{i,j,k}^{n+1}-u_{i,j,k}^{n-1}}{2\Delta t}+(\zeta_{1i}\zeta_{2j}+\zeta_{2j}\zeta_{3k}+\zeta_{3k}\zeta_{1i})u_{i,j,k}^{n}\\
=&\frac{c^2_{i+\frac12,j,k}u^n_{i+1,j,k}-(c^2_{i+\frac12,j,k}+c^2_{i-\frac12,j,k})u^n_{i,j,k}+c^2_{i-\frac12,j,k}u^n_{i-1,j,k}}{{\Delta x_1}^2}\\
+&\frac{c^2_{i,j+\frac12,k}u^n_{i,j+1,k}-(c^2_{i,j+\frac12,k}+c^2_{i,j-\frac12,k})u^n_{i,j,k}+c^2_{i,j-\frac12,k}u^n_{i,j-1,k}}{{\Delta x_2}^2}\\
+&\frac{c^2_{i,j,k+\frac12}u^n_{i,j,k+1}-(c^2_{i,j,k+\frac12}+c^2_{i,j,k-\frac12})u^n_{i,j,k}+c^2_{i,j,k-\frac12}u^n_{i,j,k-1}}{{\Delta x_3}^2}\\
+&\frac{\bar\phi^n_{1i+\frac12,j,k}-\bar\phi^n_{1i-\frac12,j,k}}{\Delta x_1}+\frac{\bar\phi^n_{2i,j+\frac12,k}-\bar\phi^n_{2i,j-\frac12,k}}{\Delta x_2}+\frac{\bar\phi^n_{3i,j,k+\frac12}-\bar\phi^n_{3i,j,k-\frac12}}{\Delta x_3}-\zeta_{1i}\,\zeta_{2j}\,\zeta_{3k}\frac{\psi^{n+\frac12}_{i,j,k}+\psi^{n-\frac12}_{i,j,k}}{2},
\end{aligned}$$

$$\begin{aligned}
u_{tt}+(\zeta_1+\zeta_2)u_t+\zeta_1\zeta_2 u&=\nabla\cdot(c^2\nabla u)+\nabla\cdot\boldsymbol{\phi},\\
\boldsymbol{\phi}_t&=\Gamma_1\boldsymbol{\phi}+c^2\Gamma_2\nabla u,
\end{aligned}$$

$$\Gamma_1=\begin{bmatrix}-\zeta_1 & 0\\ 0 & -\zeta_2\end{bmatrix},\quad \Gamma_2=\begin{bmatrix}\zeta_2-\zeta_1 & 0\\ 0 & \zeta_1-\zeta_2\end{bmatrix}.$$

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
* coming soon

Acknowledgement

- `NNFEM.jl`: Joint work with Daniel Z. Huang and Charbel Farhat.
- `FwiFlow.jl`: Joint work with Dongzhuo Li and Jerry M. Harris.
- `ADSeismic.jl`: Joint work with Weiqiang Zhu and Gregory C. Beroza.
- `PoreFlow.jl`: Joint work with Alexandre M. Tartakovsky and Jeff Burghardt.