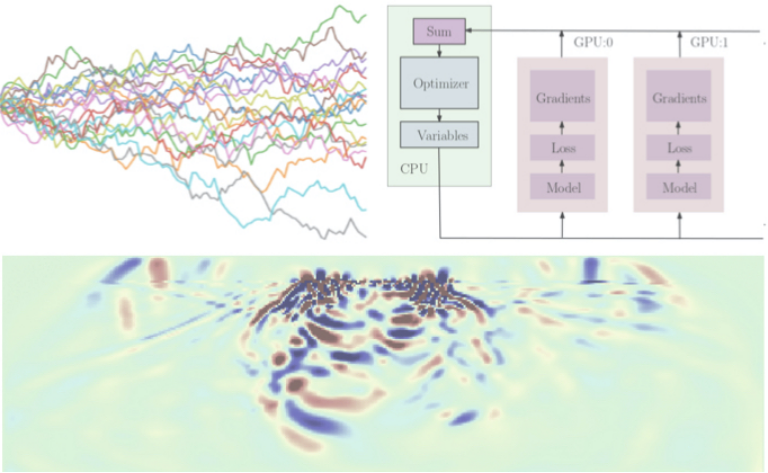
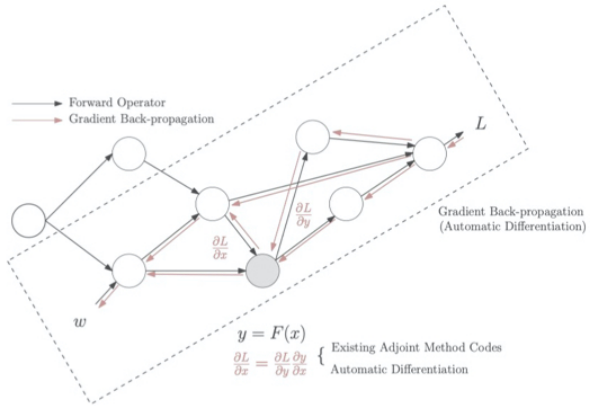
Physics Based Machine Learning for Inverse Problems

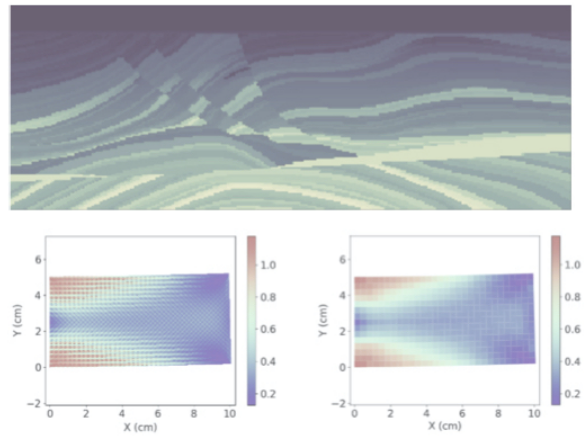
Kailai Xu and Eric Darve

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

⋆ The Pathway to Physics Based Machine Learning ⋆



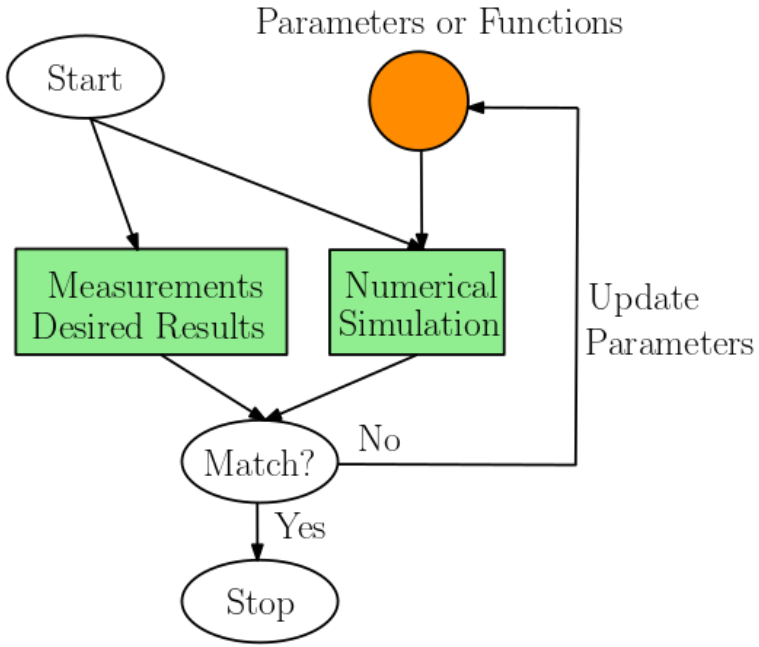


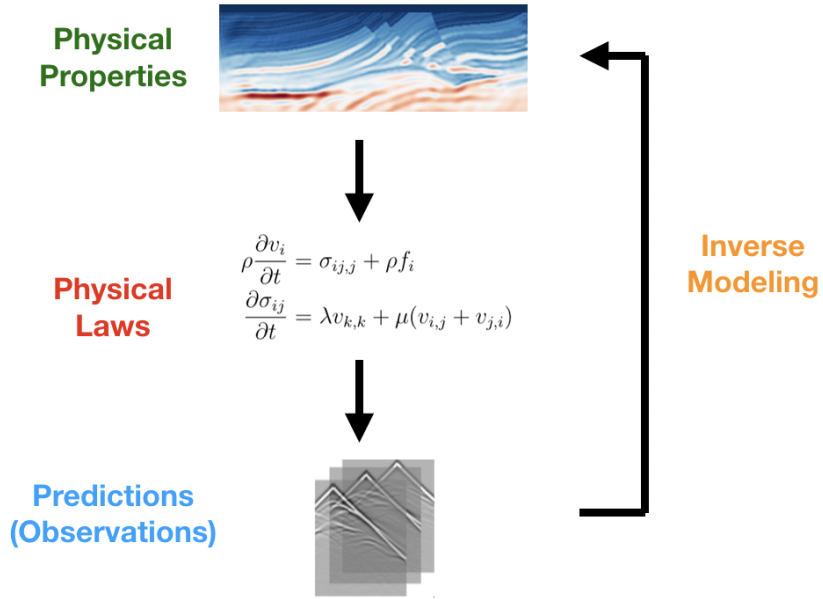


Outline

Inverse Modeling

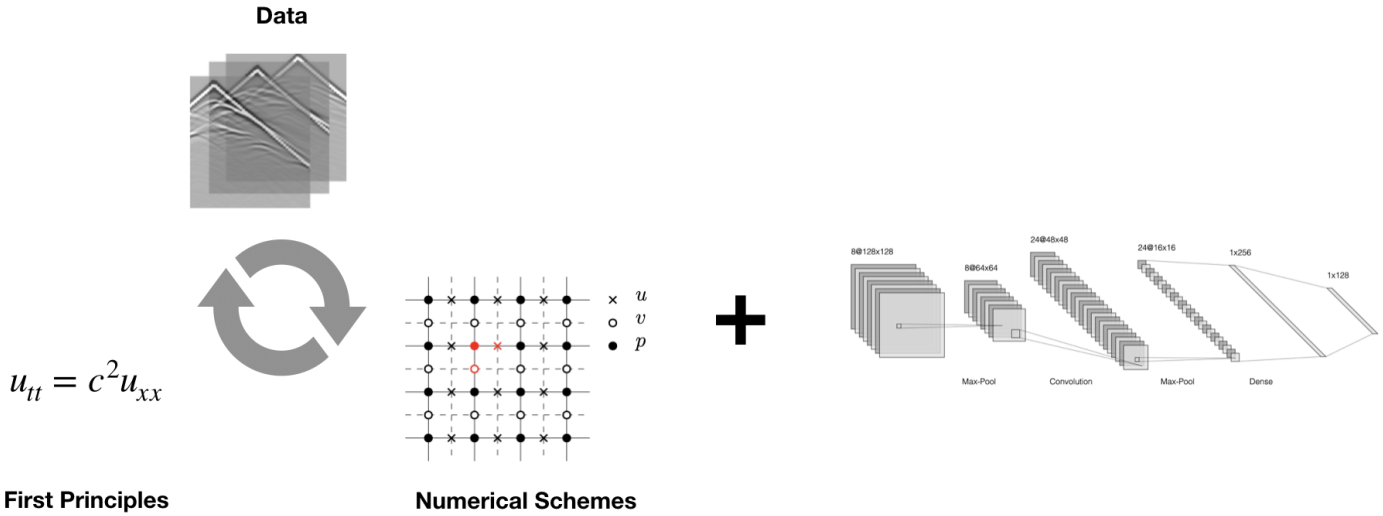
- **Inverse modeling** identifies a certain set of parameters or functions with which the outputs of the forward analysis matches the desired result or measurement.
- Many real life engineering problems can be formulated as inverse modeling problems: shape optimization for improving the performance of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





Physics Based Machine Learning

- Classical physical modeling relies on efficient numerical schemes for conservation laws derived from first principles; deep learning learns statistical relations from large amounts of training data.
- We combine the best of the two worlds and invent **physics based machine learning**: only the unknown is modeled with deep neural networks, and the known physical laws are solved with numerical schemes.



Outline

Automatic Differentiation

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

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

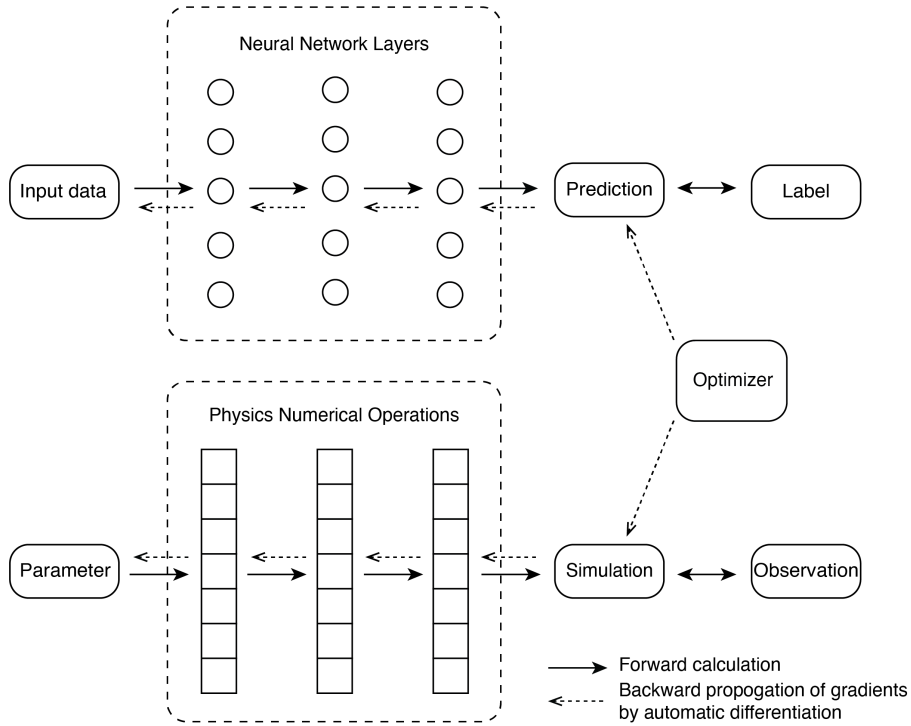
Back-propagation

=

Automatic Differentiation

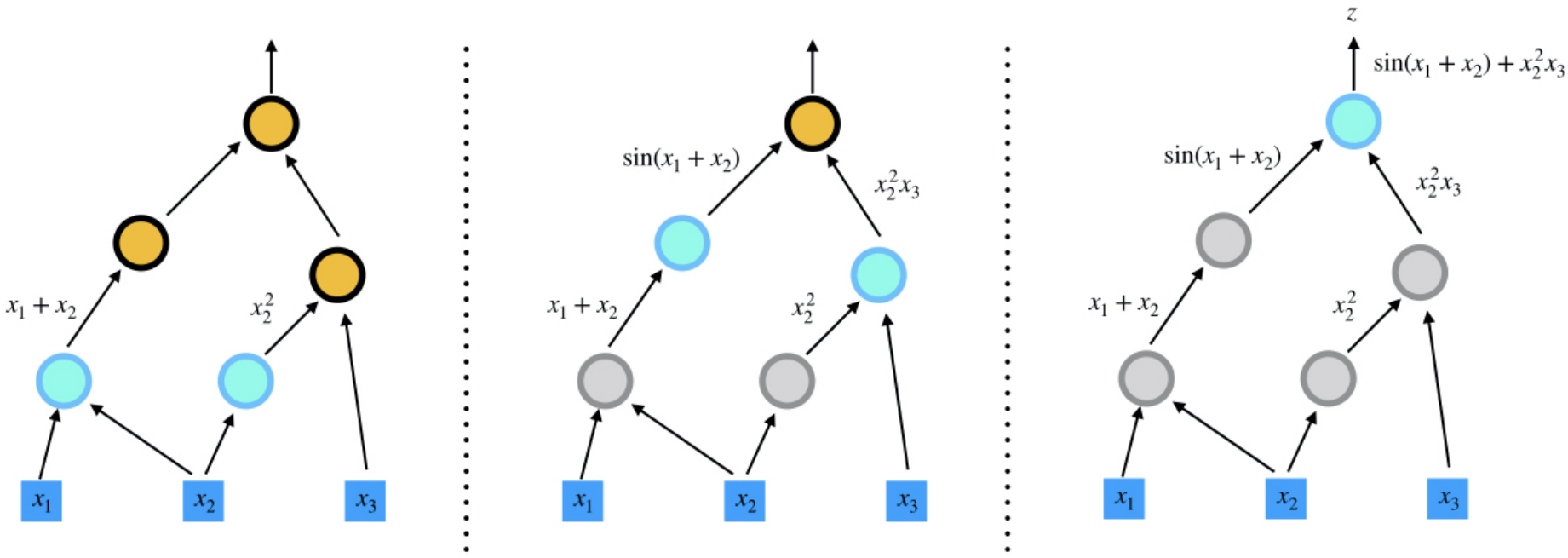
=

Adjoint-State Method



Automatic Differentiation: Computational Graph

- A computational graph is a functional description of the required computation. In the computationall 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 $z = \sin(x_1 + x_2) + x_2^2 x_3$?

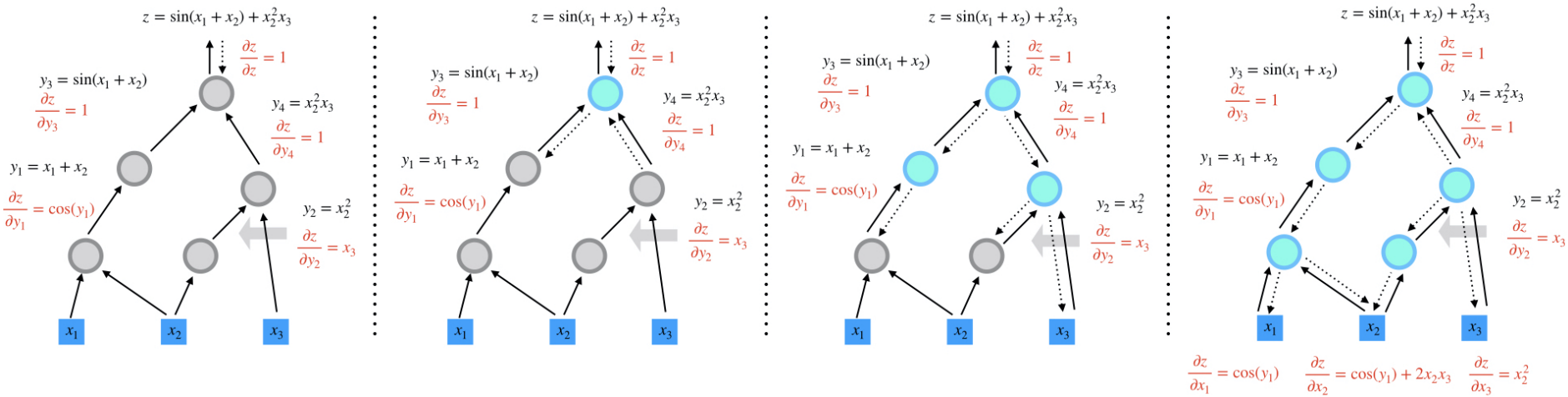


Automatic Differentiation: Reverse-Mode

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

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

- Let's see how to compute $\frac{\partial z}{\partial x_i}$, $i = 1, 2, 3$ for $z = \sin(x_1 + x_2) + x_2^2 x_3$

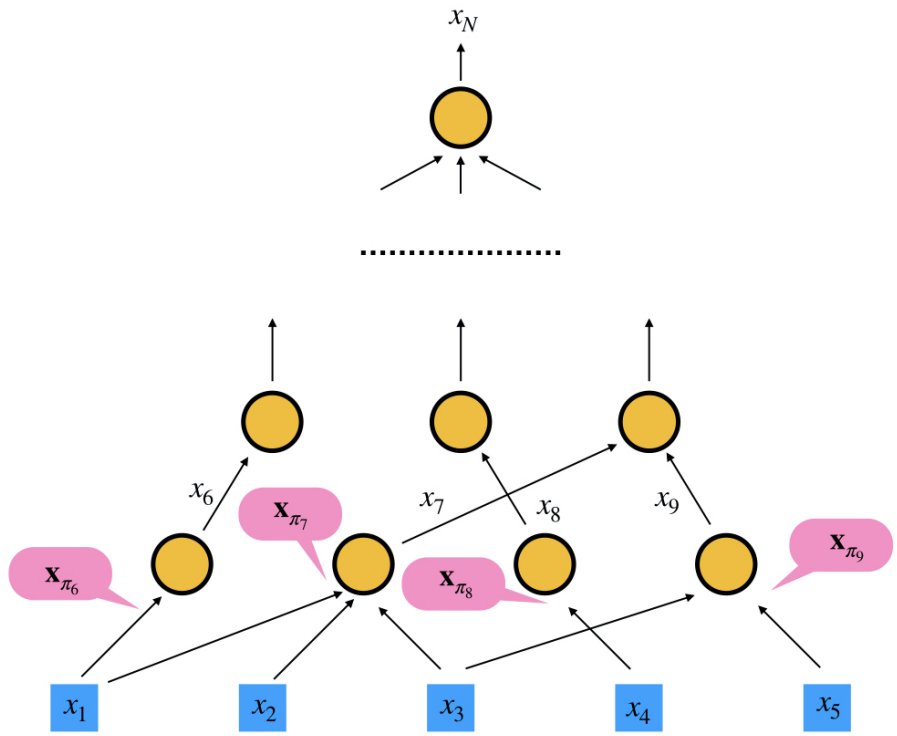


Automatic Differentiation: Mathematical Description

- A computational graph starts from independent variables x_1, x_2, ..., x_n, which are transformed by the composition of operators f_k, $k = n+1, n+2, \ldots, N$

$$x_{n+1} = f_{n+1}(\mathbf{x}_{\pi(n+1)})$$
$$x_{n+2} = f_{n+2}(\mathbf{x}_{\pi(n+2)})$$
$$\ldots$$
$$x_N = f_N(\mathbf{x}_{\pi(N)})$$

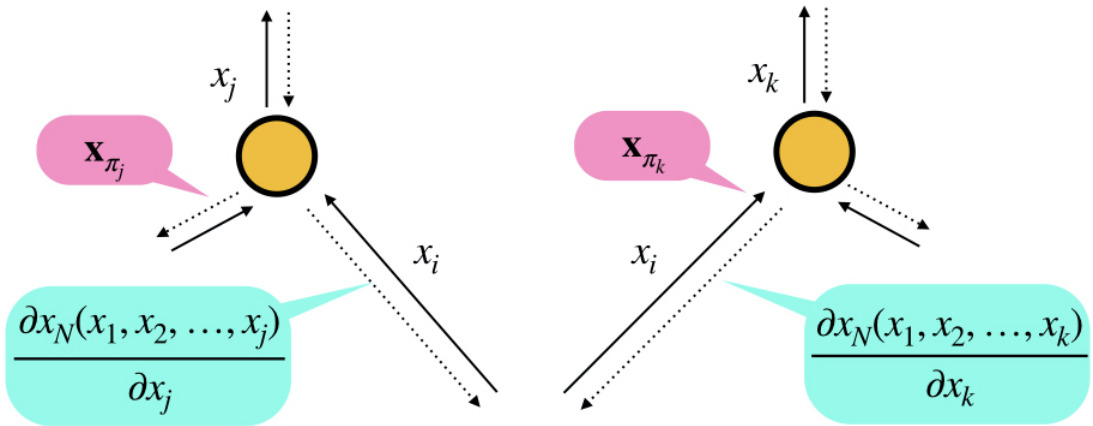
where $\pi(i) \in \{1, 2, \ldots, i-1\}$.



Automatic Differentiation: Mathematical Description

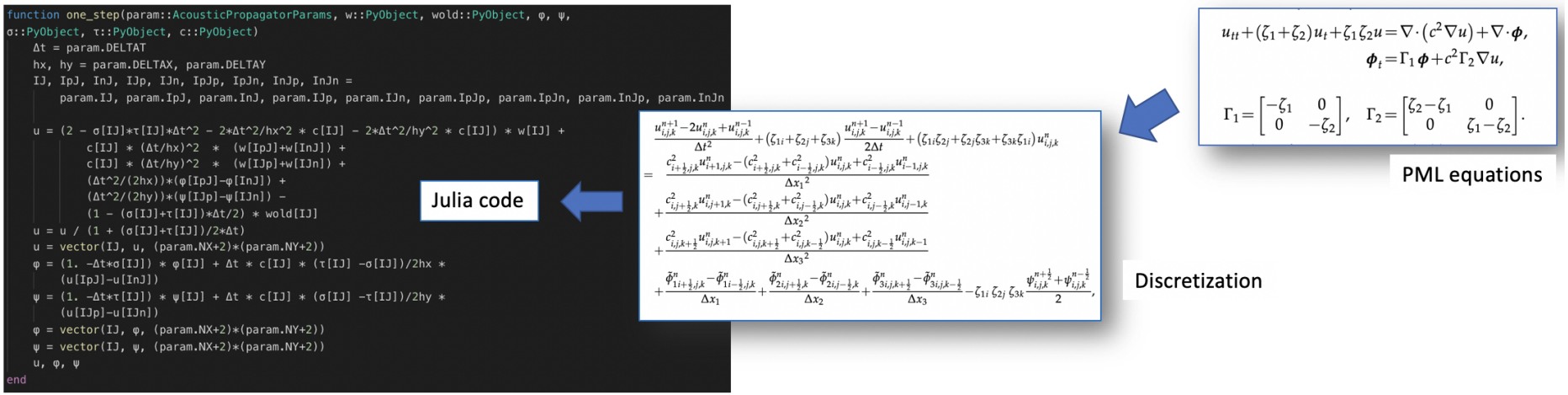
Theorem (Chain Rule)

$$\frac{\partial x_N(x_1, x_2, \ldots, x_i)}{\partial x_i} = \sum_{j\,:\, i \in \pi(j)} \frac{\partial x_N(x_1, x_2, \ldots, x_j)}{\partial x_j} \frac{\partial x_j(x_1, x_2, \ldots, x_{j-1})}{\partial x_i}$$



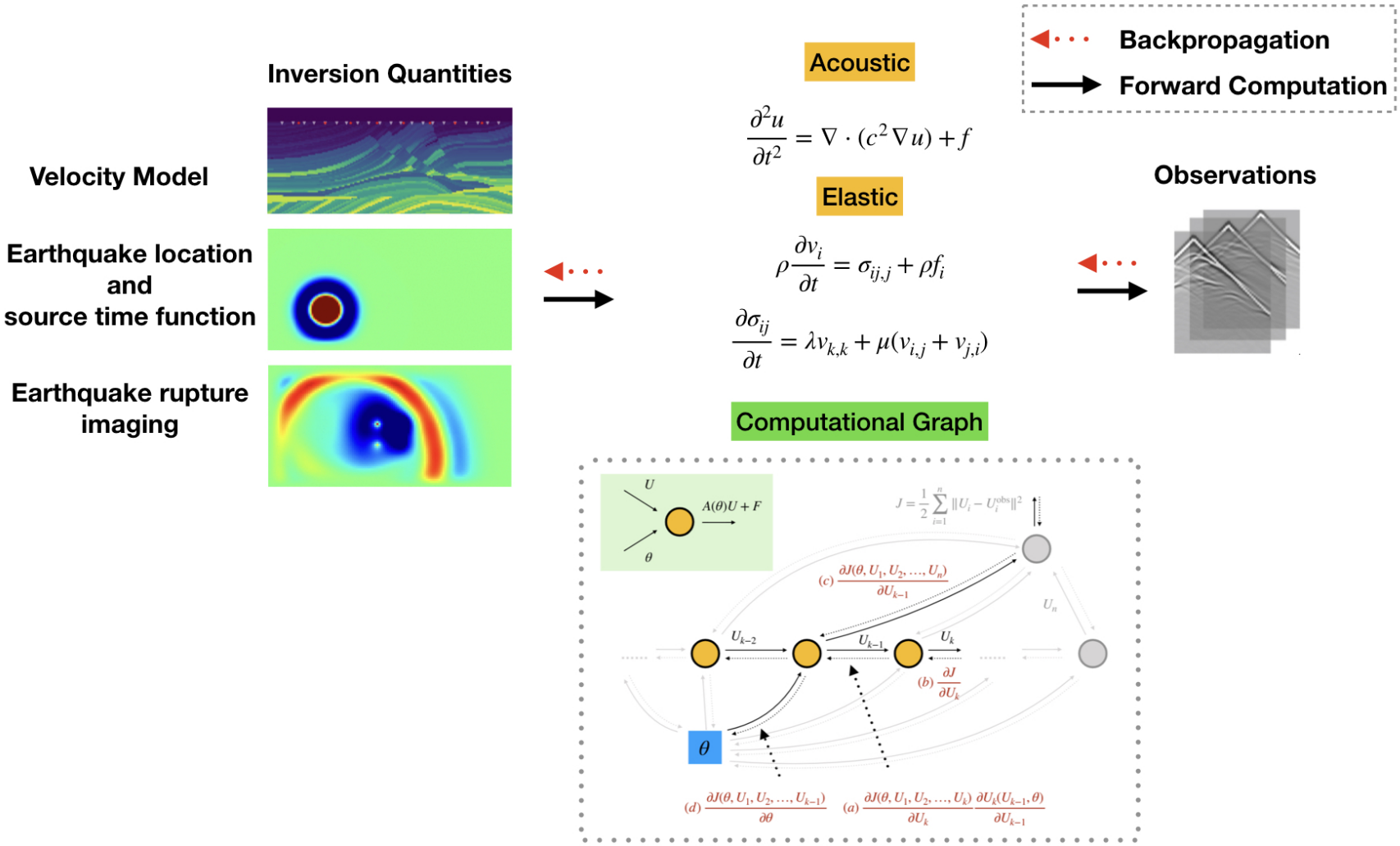
ADCME

- 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.
- ADCME simplifies programming and improves performance.



ADSeismic.jl: A General Approach to Seismic Inversion

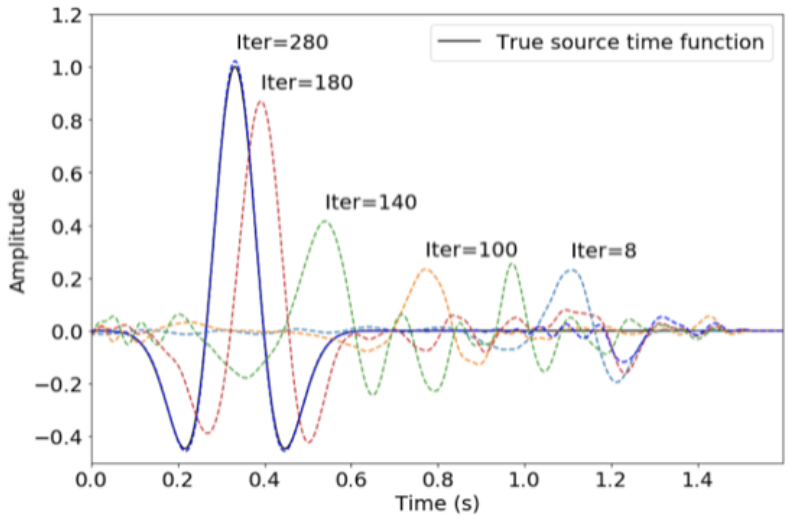
- Many seismic inversion problems can be solved within a unified framework.

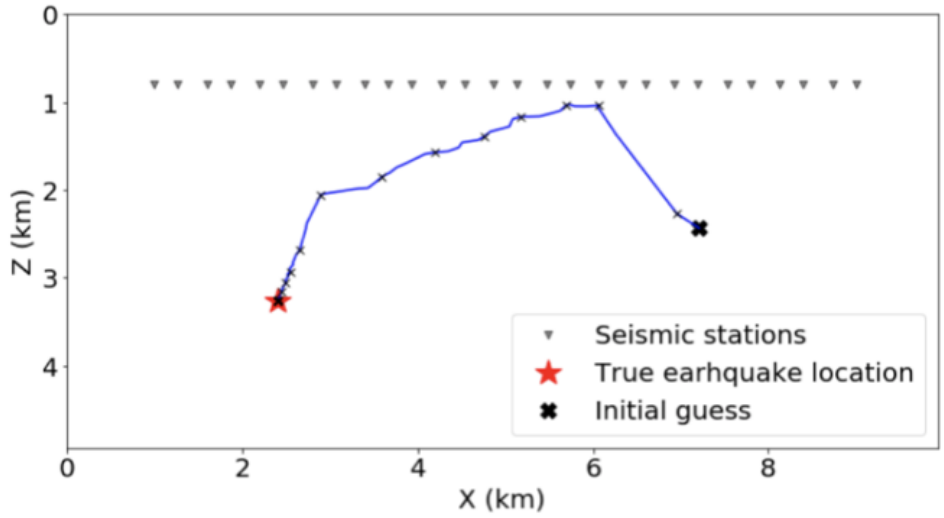


ADSeismic.jl: Earthquake Location Example

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

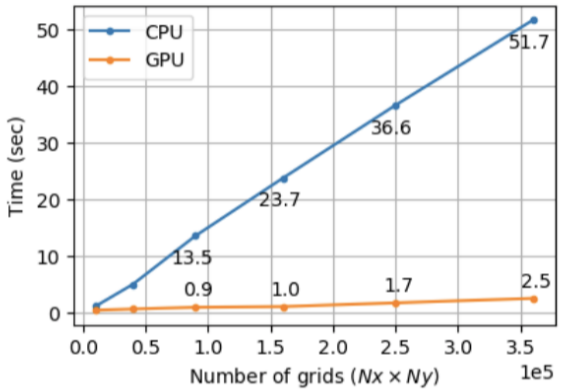
$$f(x,t) = \frac{g(t)}{2\pi\sigma^2} \exp\left(-\frac{||x - x_0||^2}{2\sigma^2}\right)$$



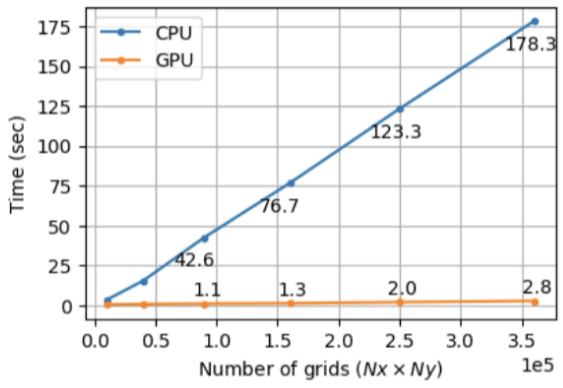


ADSeismic.jl: Benchmark

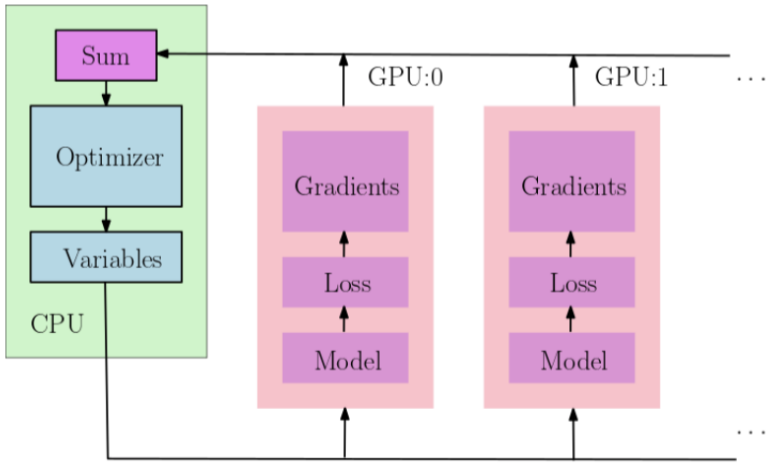
- ADCME makes the heterogeneous computation capability of TensorFlow available for scientific computing.



(a)



(b)



(c)

NNFEM.jl: Constitutive Modeling

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

- **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

$$\boxed{\text{stress} = \mathcal{M}_\theta(\text{strain}, \ldots)} \tag{2}$$

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

  Hyperelasticity (Static) $\Rightarrow \boldsymbol{\sigma} = \mathcal{M}_\theta(\boldsymbol{\epsilon})$

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

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

- Weak form of balance equations of linear momentum

$$P_i(\theta) = \underbrace{\int_V \rho \ddot{u}_i \delta u_i dV}_{\text{equals 0 for static problems}} + \int_V \sigma_{ij}\delta\varepsilon_{ij} dV$$

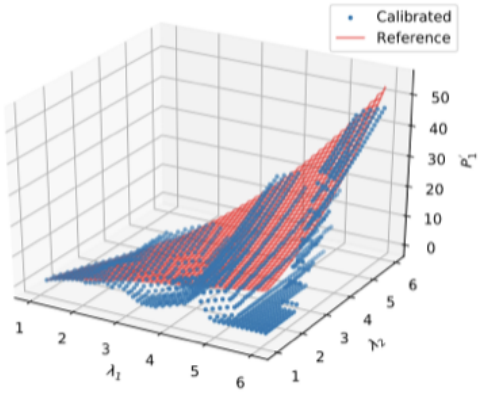
$$F_i = \int_V \rho b_i \delta u_i dV + \int_{\partial V} t_i \delta u_i dS$$

- Train the neural network by

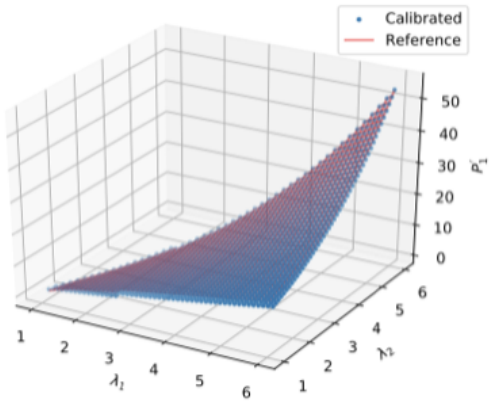
$$\boxed{L(\theta) = \min_\theta \sum_{i=1}^N (P_i(\theta) - F_i)^2}$$

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

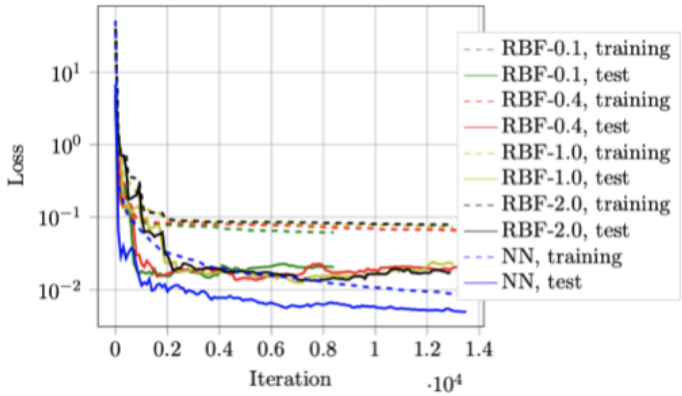
NNFEM.jl: Robustic Constitutive Modeling



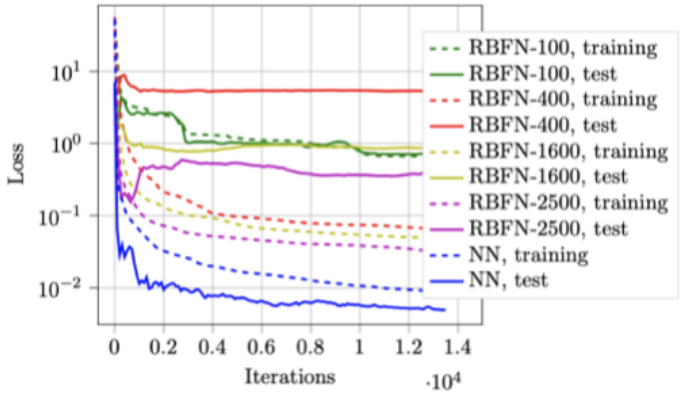
Piecewise Linear



Neural Network



Radial Basis Functions
vs.
Neural Network



Radial Basis Function Networks
vs.
Neural Network

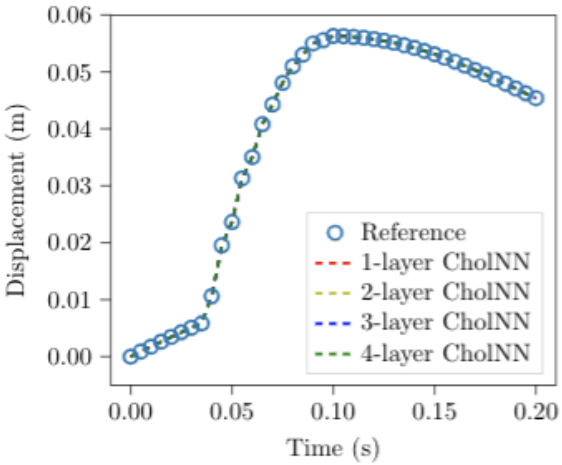
NNFEM.jl: Robustic Constitutive Modeling

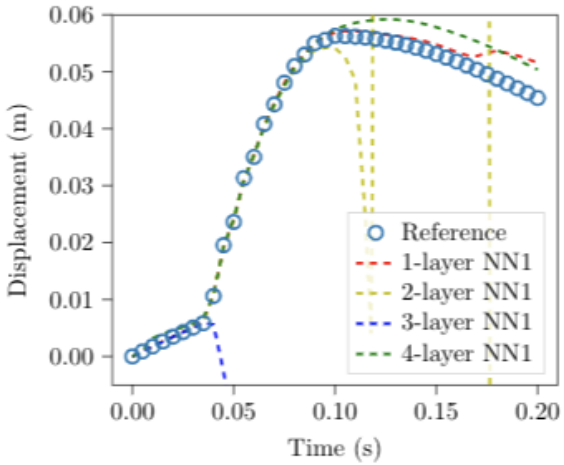
- Comparison of different neural network architectures

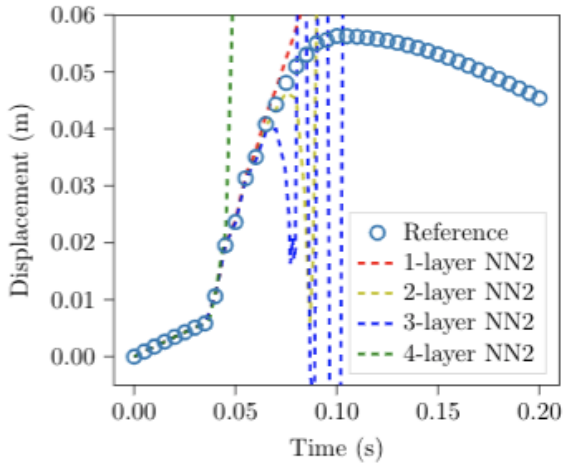
$$\boldsymbol{\sigma}^{n+1} = \mathsf{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^{n}, \boldsymbol{\sigma}^{n})\mathsf{L}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}^{n+1}, \boldsymbol{\epsilon}^{n}, \boldsymbol{\sigma}^{n})^{T}(\boldsymbol{\epsilon}^{n+1} - \boldsymbol{\epsilon}^{n})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

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

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



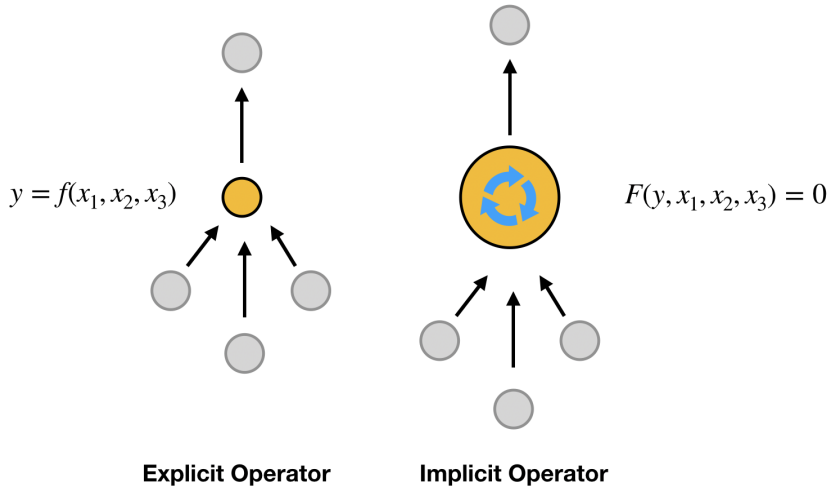




Outline

Challenges in AD

- Most AD frameworks only deal with explicit operators, i.e., the operators that can be expressed by differentiable library functions.
- A large portion of scientific computing algorithms involve implicit schemes or iterative procedures to some extent.



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

Physics Constrained Learning

- Let L_h be a error functional, F_h be the corresponding nonlinear implicit operator, θ be all the input to this operator and u_h be all the output of this node.

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

- 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

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

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

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

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

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

Step 2: Calculate the gradient

$$\underbrace{\frac{\partial L_h(u_h)}{\partial u_h}}_{1\times N} z$$

Which strategy should we use?

Physics Constrained Learning: Linear System

- Many physical simulations require solving a linear system

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

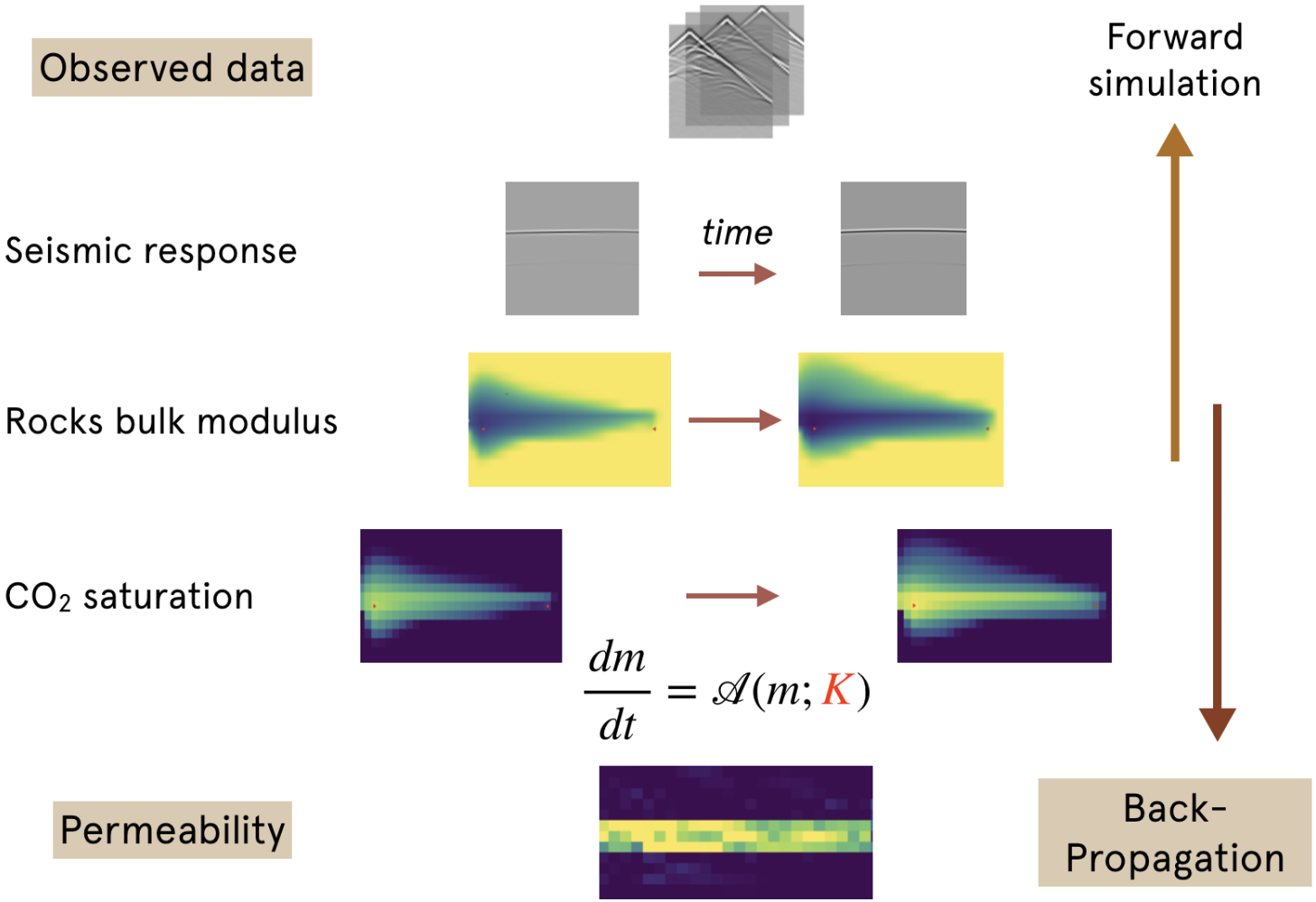
- The backpropagation formula

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

which is equivalent to

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

FwiFlow.jl: Elastic Full Waveform Inversion for subsurface flow problems with intelligent automatic differentiation



FwiFlow.jl: Fully Nonlinear Implicit Schemes

- The governing equation is a nonlinear PDE

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

$$S_1 + S_2 = 1$$

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

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

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

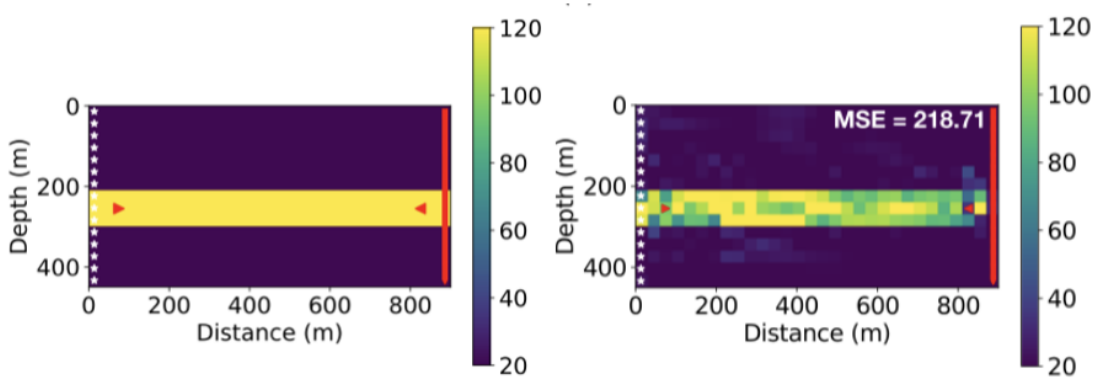
- For stability and efficiency, implicit methods are the industrial standards.

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

- 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.

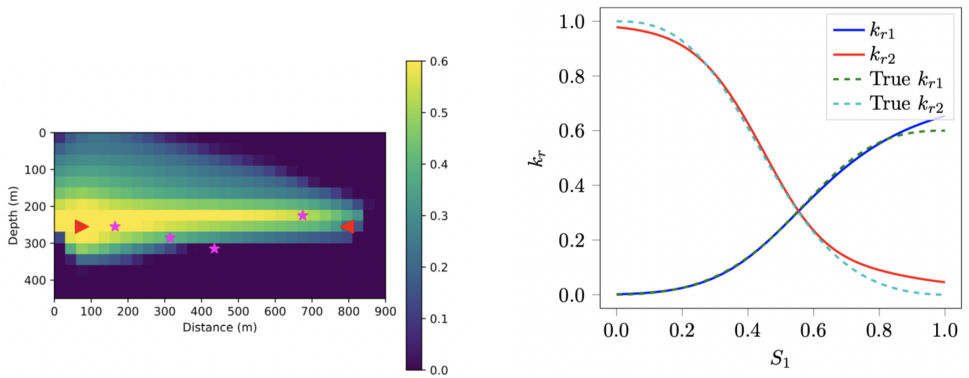
FwiFlow.jl: Showcase

- Task 1: Estimating the permeability from seismic data



- 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.

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

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

$$\nabla \cdot (\theta \nabla u(x)) = 0 \quad \mathcal{BC}(u(x)) = 0 \tag{3}$$

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

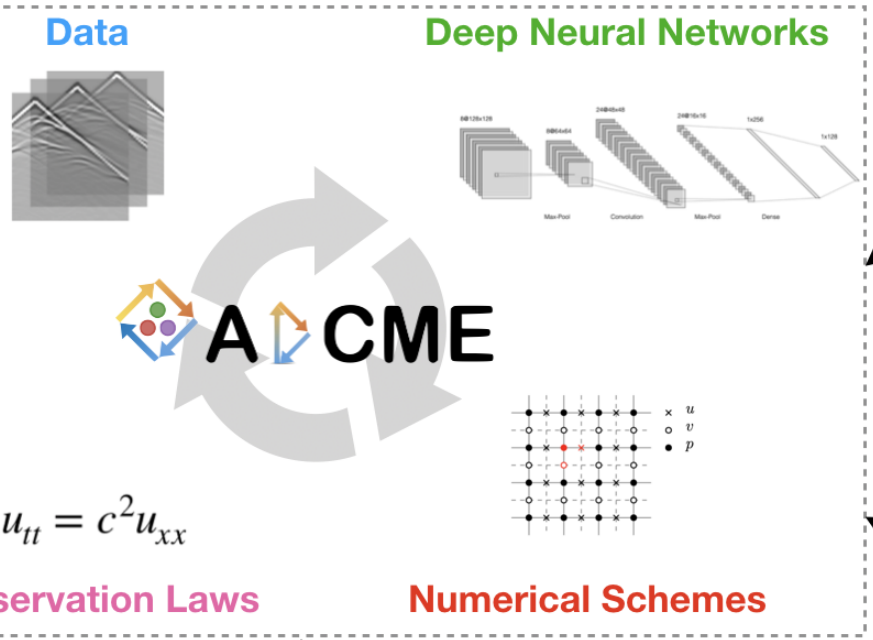
Physics based machine learning: an innovative approach to inverse modeling.

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



PoreFlow

PoreFlow.jl
Geomechanics
Viscoelasticity
Multiphase Flow
Multiphysics
* in production

FwiFlow.jl
Multiphase Flow
Nonlocal Operators

ADSeismic.jl
General Seismic Inversion

NNFEM

NNFEM.jl
Constitutive Law Modeling
Hyperelasticity
Elasto-Plasticity

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.