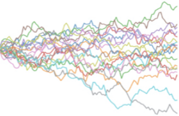
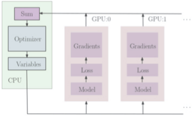
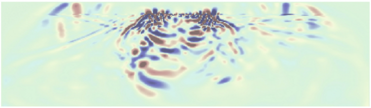
Machine Learning for Inverse Problems in Computational Engineering

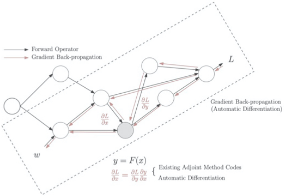
Kailai Xu and Eric Darve

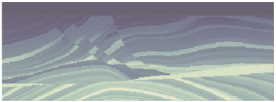
https://github.com/kailaix/ADCME.jl

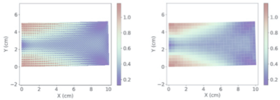












Outline

Inverse Modeling

Forward Problem

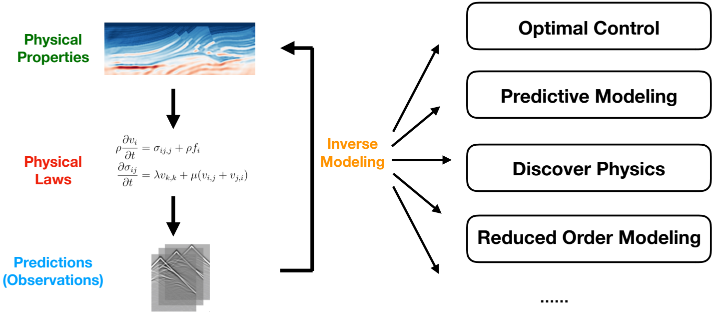
Model Parameters → **Physical Laws** → **Prediction of Observations**

Inverse Problem

Observations → **Physical Laws** → **Estimation of Parameters**

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

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

- 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 or differential algebraic equations (DAEs); e.g.,

$$F_h(\theta, u_h) = \mathrm{A}(\theta)u_h - f_h = 0$$

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.
- ...

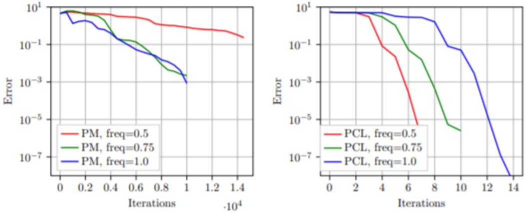
The candidate solution space is infinite dimensional.

Penalty Methods

- Parametrize f with f_θ and incorporate the physical constraint as a penalty term (regularization, prior, ...) in the loss function.

$$\min_{\theta,\, u_h} L_h(u_h) + \lambda \| F_h(f_\theta, u_h) \|_2^2$$

  - May not satisfy physical constraint $F_h(f_\theta, u_h) = 0$ accurately;
  - Slow convergence for stiff problems;

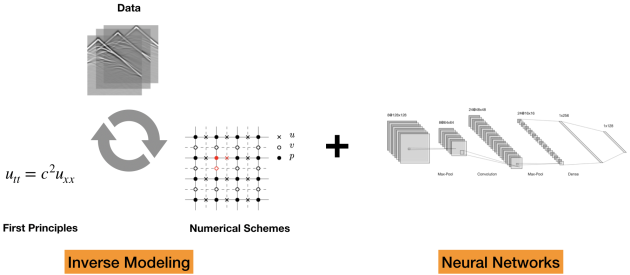


  - High dimensional optimization problem; both θ and u_h are variables.

Machine Learning for Computational Engineering

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

- Deep neural networks exhibit capability of approximating high dimensional and complicated functions.
- **Machine Learning for Computational Engineering**: the unknown function is approximated by a deep neural network, and the physical constraints are enforced by numerical schemes.
- Satisfy the physics to the largest extent.

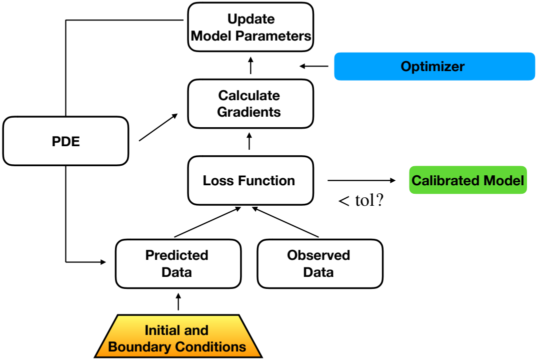


Gradient Based Optimization

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

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

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



Outline

1. Inverse Modeling
2. Automatic Differentiation
3. Physics Constrained Learning

Automatic Differentiation

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

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

Mathematical Fact

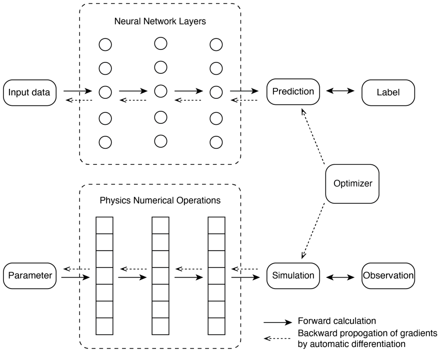
Back-propagation

$\|$

Reverse-mode
Automatic Differentiation

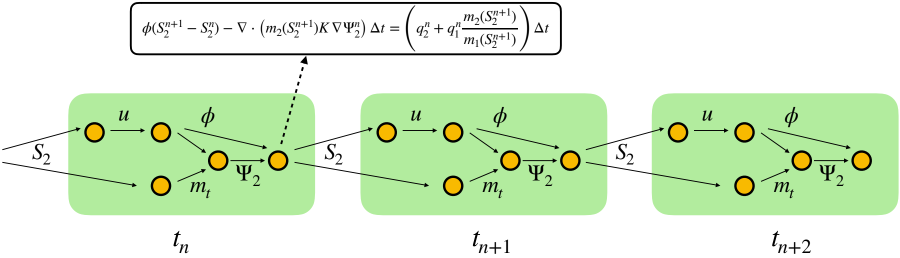
$\|$

Discrete
Adjoint-State Method

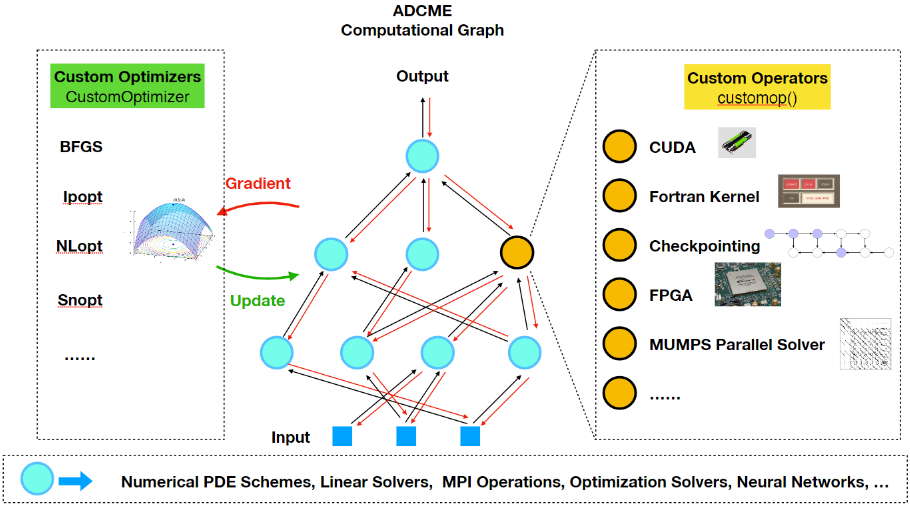


Computational Graph for Numerical Schemes

- To leverage automatic differentiation for inverse modeling, we need to express the numerical schemes in the "AD language": computational graph.
- No matter how complicated a numerical scheme is, it can be decomposed into a collection of operators that are interlinked via state variable dependencies.



ADCME: Computational-Graph-based Numerical Simulation

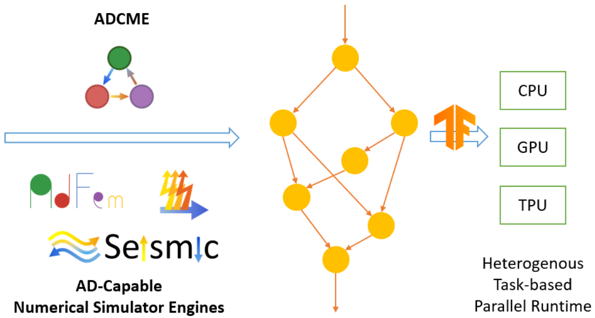


How ADCME works

- ADCME translates your numerical simulation codes to computational graph and then the computations are delegated to a heterogeneous task-based parallel computing environment through TensorFlow runtime.

$$\begin{aligned}
\operatorname{div} \sigma(u) &= f(x) && x \in \Omega \\
\sigma(u) &= C\varepsilon(u) \\
u(x) &= u_0(x) && x \in \Gamma_u \\
\sigma(x)n(x) &= t(x) && x \in \Gamma_n
\end{aligned}$$

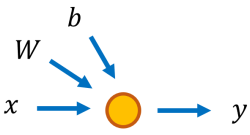
```
mmesh = Mesh(50, 50, 1/50, degree=2)

left = bcnode((x,y)->x<1e-5, mmesh)
right = bcedge((x1,y1,x2,y2)->(x1>0.049-1e-5) && (x2>0.049-1e-5), mmesh)

t1 = eval_f_on_boundary_edge((x,y)->1.0e-4, right, mmesh)
t2 = eval_f_on_boundary_edge((x,y)->0.0, right, mmesh)
rhs = compute_fem_traction_term(t1, t2, right, mmesh)

nu = 0.3
x = gauss_nodes(mmesh)
E = abs(fc(x, [20, 20, 20, 1])|>squeeze)
# E = constant(eval_f_on_gauss_pts(f, mmesh))

D = compute_plane_stress_matrix(E, nu*ones(get_ngauss(mmesh)))
K = compute_fem_stiffness_matrix(D, mmesh)

bdval = [eval_f_on_boundary_node((x,y)->0.0, left, mmesh);
        eval_f_on_boundary_node((x,y)->0.0, left, mmesh)]
DOF = [left;left .+ mmesh.ndof]
K, rhs = impose_Dirichlet_boundary_conditions(K, rhs, DOF, bdval)
u = K\rhs
```



Outline

- 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.

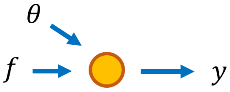
DNN: Explicit



$y = \sigma(Wx + b)$

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

Numerical Schemes: Implicit, Iterative



$A(y, \theta)y = f$

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 (PCL)

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

- 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

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

Physics Constrained Learning for Stiff Problems

- For stiff problems, better to resolve physics using PCL.
- Consider a model problem

$$\min_\theta \|u - u_0\|_2^2 \qquad \text{s.t. } Au = \theta y$$

$$\text{PCL}: \qquad \min_\theta \tilde{L}_h(\theta) = \|\theta A^{-1} y - u_0\|_2^2 = (\theta - 1)^2 \|u_0\|_2^2$$

$$\text{Penalty Method}: \qquad \min_{\theta, u_h} \tilde{L}_h(\theta, u_h) = \|u_h - u_0\|_2^2 + \lambda \|A u_h - \theta y\|_2^2$$

Theorem

The condition number of A_λ *is*

$$\liminf_{\lambda \to \infty} \kappa(\mathrm{A}_\lambda) = \kappa(A)^2, \qquad \mathrm{A}_\lambda = \begin{bmatrix} I & 0 \\ \sqrt{\lambda} A & -\sqrt{\lambda} y \end{bmatrix}, \qquad \mathrm{y} = \begin{bmatrix} u_0 \\ 0 \end{bmatrix}$$

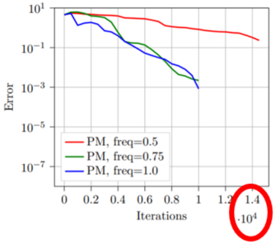
and therefore, the condition number of the unconstrained optimization problem from the penalty method is equal to the square of the condition number of the PCL asymptotically.

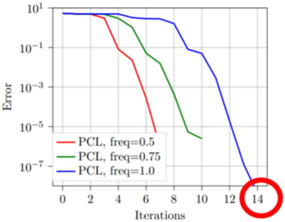
Physics Constrained Learning for Stiff Problems

Parameter Inverse Problem

$$\Delta u + k^2 g(x)u = 0$$
$$g(x) = 5x^2 + 2y^2$$

$$g_\theta(x) = \theta_1 x^2 + \theta_2 y^2 + \theta_3 xy + \theta_4 x + \theta_5 y + \theta_6$$

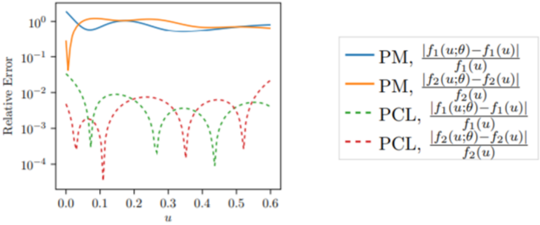




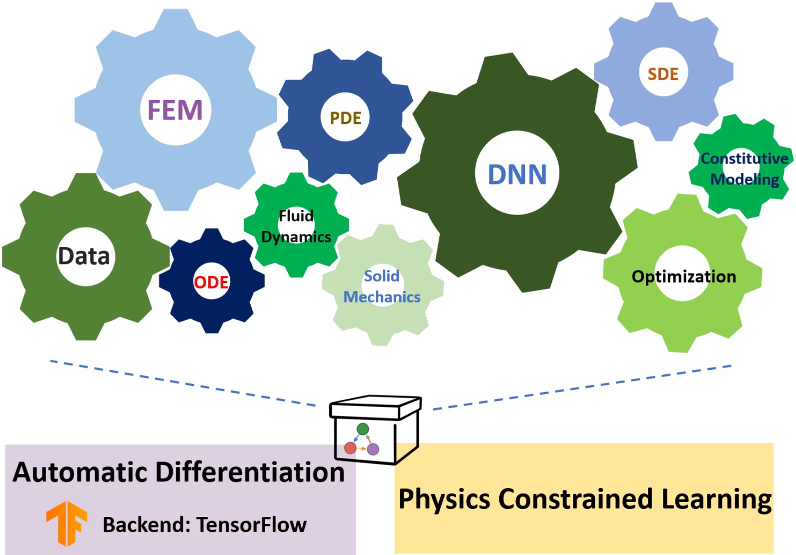
Approximate Unknown Functions using DNNs

$$-\nabla \cdot (\boldsymbol{f}(u)\nabla u) = h(\boldsymbol{x})$$

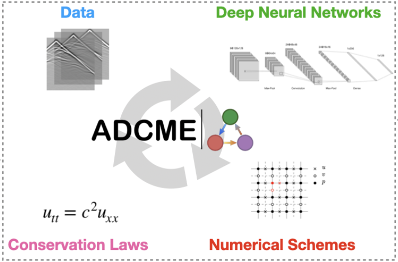
$$\boldsymbol{f}(u) = \begin{bmatrix} NN(u;\theta_1) & 0 \\ 0 & NN(u;\theta_2) \end{bmatrix}$$



PCL: Backbone of the ADCME Infrastructure



A General Approach to Inverse Modeling



AdFem.jl
Finite Element Library
for Inverse Modeling
https://github.com/kailaix/AdFem.jl

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