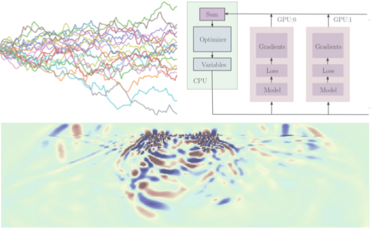
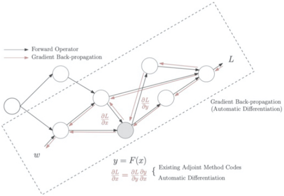
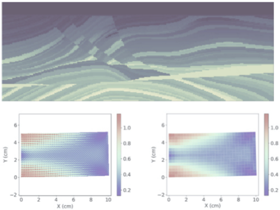
Machine Learning for Inverse Problems in Computational Engineering

Kailai Xu
Stanford University



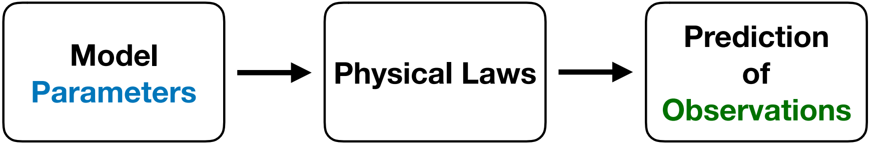




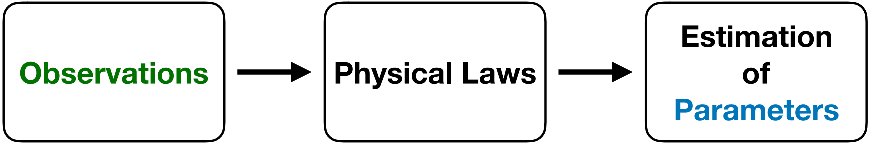
Outline

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 $u_{\rm obs}$, e.g., $L_h(u_h) = \|u_h - u_{\rm obs}\|_2^2$.
- 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.
- ...

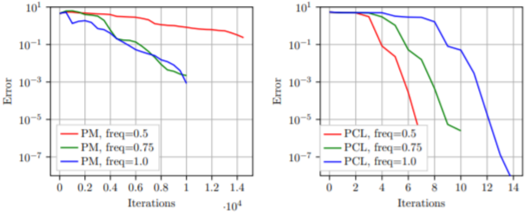
The candidate solution space is infinite dimensional.

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;



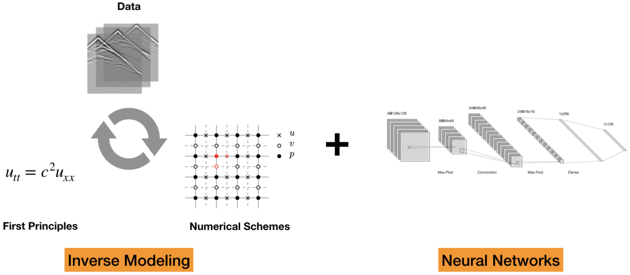
  - High dimensional optimization problem; both θ and u_h are variables.

Machine Learning for Computational Engineering

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

1. Use a deep neural network to approximate the (high dimensional) unknown function;
2. Solve u_h from the physical constraint using a numerical PDE solver;
3. Apply an unconstrained optimizer to the reduced problem

$$\min_{\theta} L_h(u_h(\theta))$$

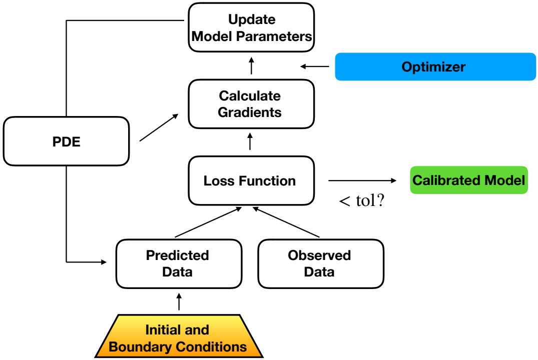


Gradient Based Optimization

$$\min_{\theta} L_h(u_h(\theta))$$

- Steepest descent method:

$$\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla_\theta L_h(u_h(\theta_k))$$



Research Question

Overall

> *Develop algorithms and tools for solving inverse problems by combining DNNs and numerical PDE solvers.*

Specifically, I discuss my contributions on:

1. how to reconcile gradient calculations in both numerical PDE solvers and deep neural networks using automatic differentiation;
2. how to calculate gradients for implicit and/or iterative operators in sophisticated numerical solvers;
3. how to accelerate convergence and improve accuracy with Hessian information;
4. how to solve stochastic inverse problems.

Outline

Question

How deep neural networks communicate gradients with numerical PDE solvers?

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

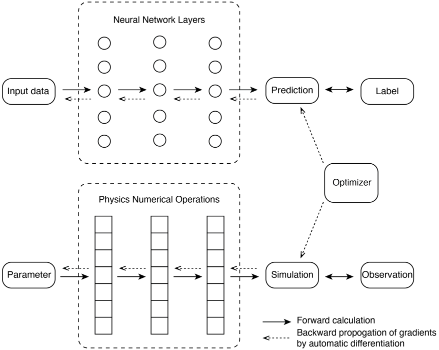
Back-propagation

||

Reverse-mode Automatic Differentiation

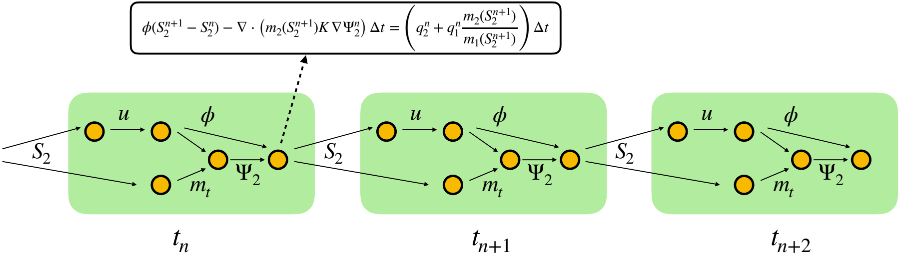
||

Discrete Adjoint-State Method

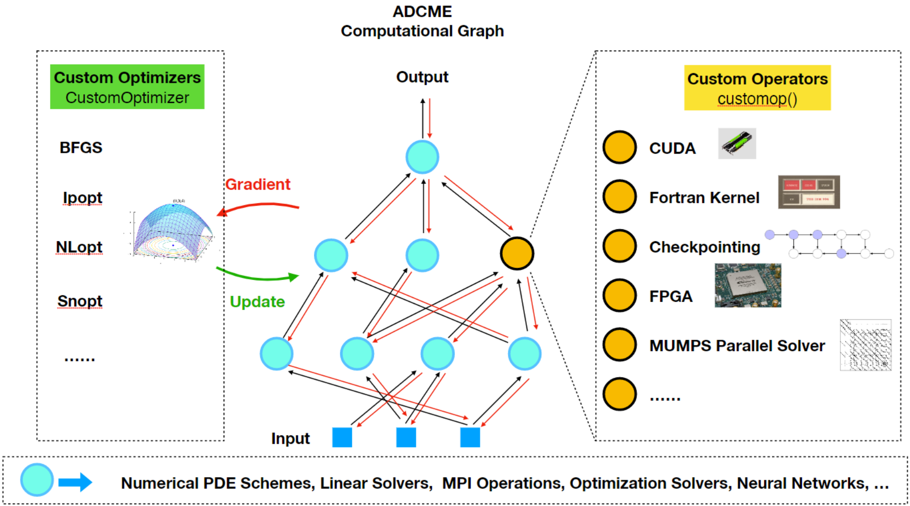


Computational Graph for Numerical Schemes

- 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

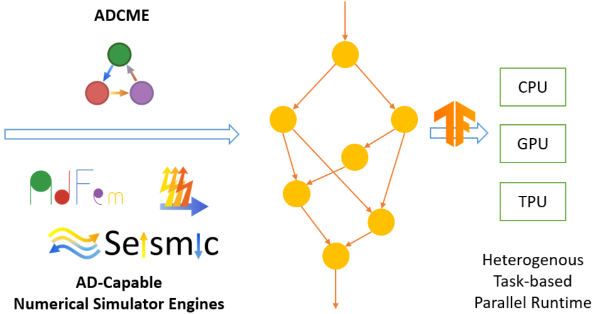


How ADCME works

- 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
```



Summary

- Mathematically equivalent techniques for calculating gradients:
  - gradient back-propagation (DNN)
  - discrete adjoint-state methods (PDE)
  - reverse-mode automatic differentiation
- Computational graphs bridge the gap between gradient calculations in numerical PDE solvers and DNNs.
- ADCME extends the capability of TensorFlow to PDE solvers, providing users a single piece of software for numerical simulations, deep learning, and optimization.

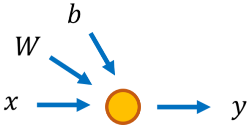
Outline

Motivation

- 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.

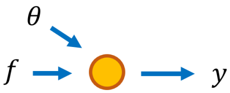
Linear/Nonlinear	Explicit/Implicit	Expression
Linear	Explicit	$y = Ax$
Nonlinear	Explicit	$y = F(x)$
Linear	**Implicit**	$Ay = x$
Nonlinear	**Implicit**	$F(x, y) = 0$

DNN: Explicit



$y = \sigma(Wx + b)$

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

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

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

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

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

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

Theorem

The condition number of A_λ *is*

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

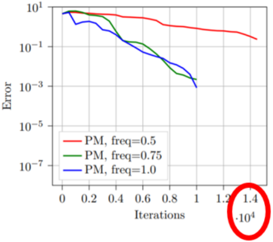
and therefore, the condition number of the unconstrained optimization problem from the penalty method is equal to the square of the condition number of the PCL asymptotically.

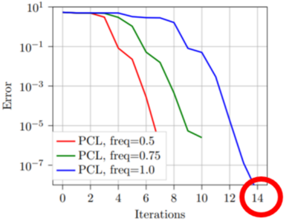
Physics Constrained Learning for Stiff Problems

Parameter Inverse Problem

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

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

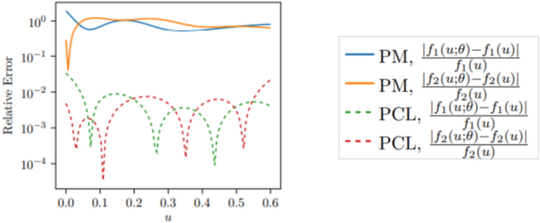




Approximate Unknown Functions using DNNs

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

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



Summary

- Implicit and iterative operators are ubiquitous in numerical PDE solvers. These operators are insufficiently treated in deep learning software and frameworks.
- Penalty methods suffer from slow convergence for stiff problems.
- First order physics constrained learning helps you calculate gradients of implicit/iterative operators efficiently.
- PCL leads to faster convergence and better accuracy compared to penalty methods.

Outline

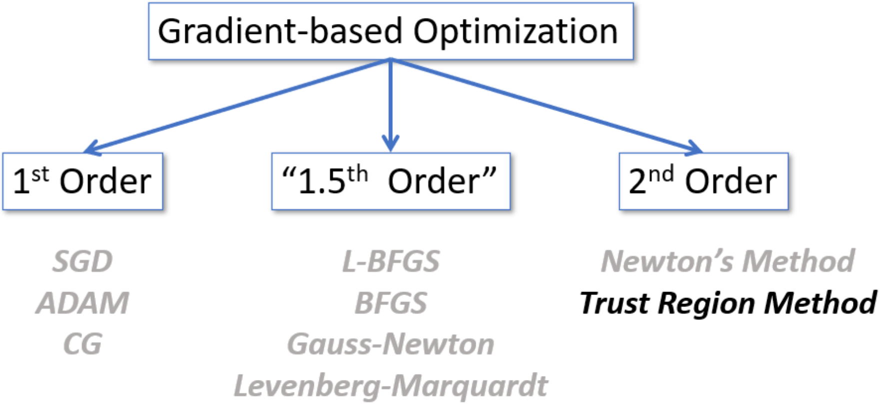
Motivation

- First order methods and quasi-Newton methods are workhorse of physics informed machine learning
    - First order methods: stochastic gradient descent (SGD) and ADAM
    - Quasi-Newton methods: BFGS and L-BFGS
- Second order methods, such as trust region methods, are desirable for accelerating convergence and improving accuracy.
    - It requires us to calculate the curvature information (Hessians).

Goal

Accelerate convergence and improve accuracy with Hessian information

Optimization Algorithms



Trust Region vs. Line Search

Trust Region

- Approximate $f(x_k + p)$ by a model quadratic function

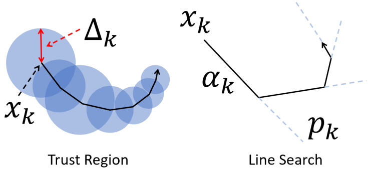
$$m_k(p) = f_k + g_k^T p + \frac{1}{2} p^T B_k p$$

$$f_k = f(x_k), g_k = \nabla f(x_k), B_k = \nabla^2 f(x_k)$$

- Solve the optimization problem within a trust region $\|p\| \leq \Delta_k$

$$p_k = \arg\min_p \; m_k(p) \quad \text{s.t. } \|p\| \leq \Delta_k$$

- If decrease in $f(x_k + p_k)$ is sufficient, then update the state $x_{k+1} = x_k + p_k$; otherwise, $x_{k+1} = x_k$ and improve Δ_k.



Line Search

- Determine a descent direction p_k
- Determine a step size α_k that sufficiently reduces $f(x_k + \alpha_k p_k)$
- Update the state $x_{k+1} = x_k + \alpha_k p_k$

Second-order Physics Constrained Learning

- Consider a composite function with a vector input x and scalar output

$$v = f(G(x)) \tag{1}$$

- Define

$$f_{,k}(y) = \frac{\partial f(y)}{\partial y_k}, \quad f_{,kl}(y) = \frac{\partial^2 f(y)}{\partial y_k \partial y_l}$$
$$G_{k,l}(x) = \frac{\partial G_k(x)}{\partial x_l}, \quad G_{k,lr}(x) = \frac{\partial^2 G_k(x)}{\partial x_l \partial x_r}$$

- Differentiate Equation (1) with respect to x_i

$$\frac{\partial v}{\partial x_i} = f_{,k} G_{k,i} \tag{2}$$

- Differentiate Equation (2) with respect to x_j

$$\boxed{\frac{\partial^2 v}{\partial x_i \partial x_j} = f_{,kr} G_{k,i} G_{r,j} + f_{,k} G_{k,ij}}$$

Second-order Physics Constrained Learning

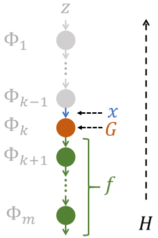
In the vector form,

$$\boxed{\nabla^2 v = (\nabla G)^T \nabla^2 f (\nabla G) + \nabla^2(\bar{G}^T G) \qquad \bar{G} = \nabla f}$$

- Consider a function composed of a sequence of computations

$$v = \Phi_m(\Phi_{m-1}(\cdots(\Phi_1(z))))$$

Initialize $H \leftarrow 0$
for $k = m-1, m-2, \ldots, 1$ **do**
  Define $f := \Phi_m(\Phi_{m-1}(\cdots(\Phi_{k+1}(\cdot))))$, $G := \Phi_k$
  Calculate the gradient (Jacobian) $J \leftarrow \nabla G$
  Extract $\bar{G}$ from the saved gradient back-propagation data.
  Calculate $Z = \nabla^2(\bar{G}^T G)$
  Update $H \leftarrow J^T H J + Z$
end for



Numerical Benchmark

- We consider the heat equation in $\Omega = [0,1]^2$

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \nabla \cdot (\kappa(x,y)\nabla u)) + f(x,y) & x \in \Omega \\
u(x,y,0) &= x(1-x)y^2(1-y)^2 & (x,y) \in \Omega \\
u(x,y,t) &= 0 & (x,y) \in \partial\Omega
\end{aligned} \tag{3}$$

- The diffusivity coefficient κ and exact solution u are given by

$$\begin{aligned}
\kappa(x,y) &= 2x^2 - 1.05x^4 + x^6 + xy + y^2 \\
u(x,y,t) &= x(1-x)y^2(1-y)^2 e^{-t}
\end{aligned} \tag{4}$$

- We learn a DNN approximation to κ using full-field observations of u

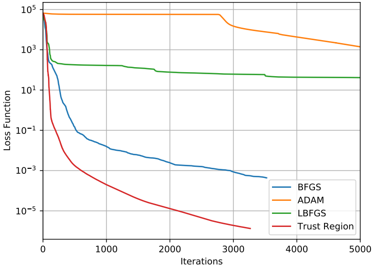
$$\kappa(x,y) \approx \mathrm{NN}_\theta(x,y)$$

Convergence

- The optimization problem is given by

$$\min_{\theta} \; L(\theta) = \sum_{n} \sum_{i,j} \left(\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} - F_{i,j}(u^{n+1};\theta) - f_{i,j}^{n+1} \right)^2 \qquad (5)$$

Here $F_{i,j}(u^{n+1};\theta)$ is the 4-point finite difference approximation to the Laplacian $\nabla \cdot (\mathrm{NN}_\theta \nabla u)$.

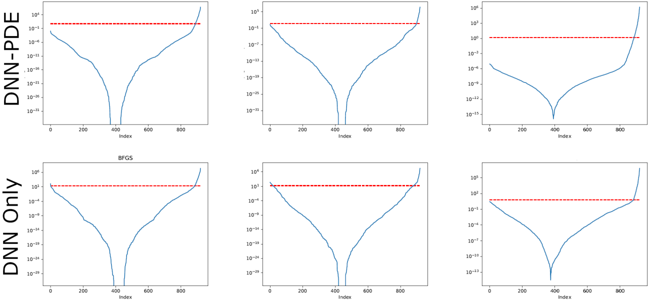


Effect of PDEs

- Consider the loss function excluding the effects of PDEs

$$l(\theta) = \sum_{i,j} (\mathrm{NN}_\theta(x_{i,j}, y_{i,j}) - \kappa(x_{i,j}, y_{i,j}))^2$$

- Eigenvalue magnitudes of $\nabla^2 L(\theta)$ and $\nabla^2 l(\theta)$



BFGS LBFGS Trust Region

Effect of PDEs

- Most of the eigenvalue directions at the local landscape of loss functions are "flat" $\Rightarrow$ "effective degrees of freedom (DOFs)".
- Physical constraints (PDEs) further cannibalize effective DOFs:

	BFGS	LBFGS	Trust Region
DNN-PDE	**31**	**22**	**35**
DNN Only	34	41	38

Effect of Widths and Depths

- The ratio of zero eigenvalues **increases** as
  - the number of hidden layers increase for a fixed number (20) of neurons per layer

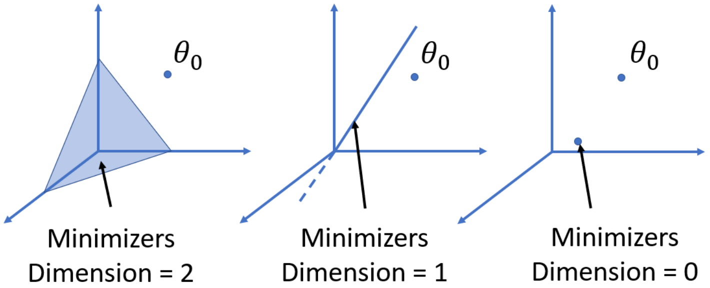
# Hidden Layers	LBFGS	BFGS	Trust Region
1	76.54	72.84	77.78
2	98.2	94.41	93.21
3	98.7	98.15	96.09

  - the number of neurons per layer increases for a fixed number (3) of hidden layers

# Neurons per Layer	LBFGS	BFGS	Trust Region
5	93.83	85.19	69.14
10	97.7	83.52	89.66
20	96.2	97.39	96.42

Effect of Widths and Depths

- Implications for overparametrization: **the minimizer lies on a relatively higher dimensional manifold of the parameter space**.



Summary

- Trust region methods converge significantly faster compared to first order/quasi second order methods by leveraging Hessian information.
- Second order physics constrained learning helps you calculate Hessian matrices efficiently.
- The local minimum of DNNs have small effective degrees of freedom compared to DNN sizes.

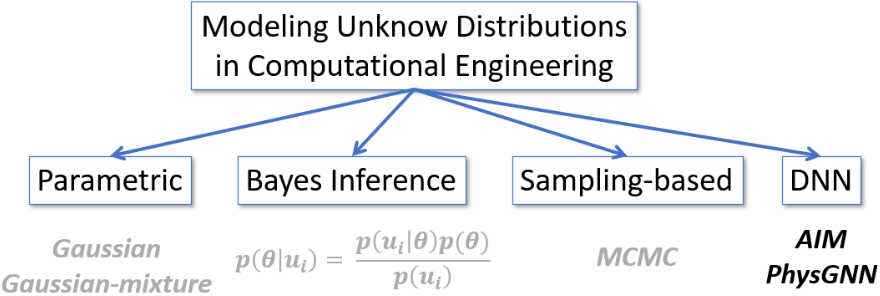
Outline

Motivation

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

- What if f is a random variable?
  - uncertainty estimation
- What if F_h is a stochastic model?
  - stochastic differential equations (SDE)

Methodologies



Modeling Stochasticity using Deep Neural Networks

- We consider a map F

$$F : (w, \theta) \mapsto u(\cdot, \cdot)$$

  - w is sampled from a **know** stochastic process (stochasticity intrinsic to the model).
  - θ is a random variable of interest.
  - $u(x, t)$ is the output of the mapping, which depends on the location x and time t. u is also a random variable.

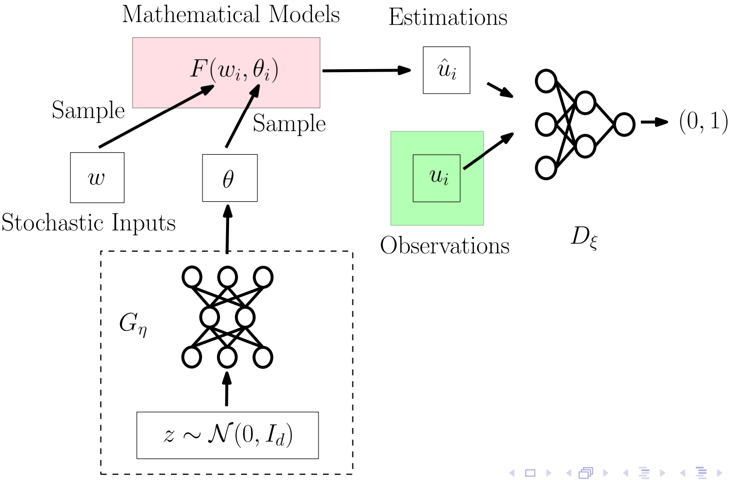
- Examples:
  - CIR process (a short-term interest rate model)

$$dr_t = \kappa(\tau - r_t)dt + \sqrt{r_t}\sigma dW_t$$

  Here $\theta = (\kappa, \tau, \sigma)$, $w = W_t$, and $u = r_t$.

Adversarial Inverse Modeling (AIM)

- Approximate the unknown distribution θ with a DNN G_η (generator);
- A discriminator D_ξ tries to distinguish between generated and observed samples.



Adversarial Inverse Modeling

- Find the **Nash equilibrium** by solving a min-max problem

$$\min_{\eta} \max_{\xi} \; V(D_\xi, G_\eta)$$

  Here V is the reward to the discriminator D for distinguishing generated and observed samples.
- Inspired by generative adversarial nets (GAN): iteratively minimizing discriminator and generator loss functions
  - Example: Kullback-Leibler (KL) GAN

$$L^G(\{\hat{u}_i\}; \eta) = \frac{1}{n}\sum_{i=1}^{n} \log \frac{1 - D_\xi(\hat{u}_i(\eta))}{D_\xi(\hat{u}_i(\eta))}$$

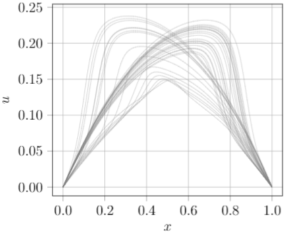
$$L^D(\{u_i\}, \{\hat{u}_i\}; \xi) = -\frac{1}{n}\sum_{i=1}^{n} (\log D_\xi(\hat{u}_i) + \log(1 - D_\xi(u_i)))$$

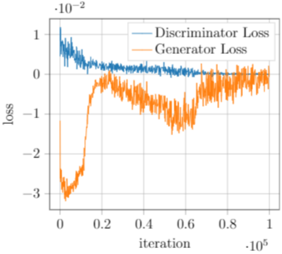
Numerical Benchmarks

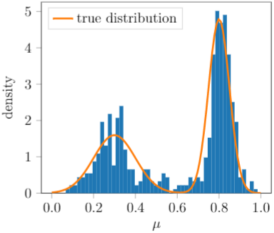
- We consider a Poisson equation

$$\begin{cases} -\nabla \cdot (a(x)\nabla u(x)) = 1 & x \in (0,1) \\ u(0) = u(1) = 0 & \text{otherwise} \end{cases}$$

$$a(x) = 1 - 0.9 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$







Optimal Transport

- Solve the transport matrix from a linear programming problem

$$\begin{aligned} \min_{p_{ij}} W(\mathrm{p}; \mathrm{x}, \mathrm{y}) &= \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}c_{ij} \\ \text{s.t. } \sum_{j=1}^{m} p_{ij} = \frac{1}{n},& \ \sum_{i=1}^{n} p_{ij} = \frac{1}{m}, \ 0 \leq p_{ij} \leq 1 \end{aligned} \tag{6}$$

- Discrete Wasserstein distance between two sets of samples:

$$D(\mathrm{x}, \mathrm{y}) = W(\mathrm{p}^*; \mathrm{x}, \mathrm{y}) = \sum_{1\leq i\leq n, 1\leq j\leq m} p_{ij}^* \|x_i - y_j\|_1$$

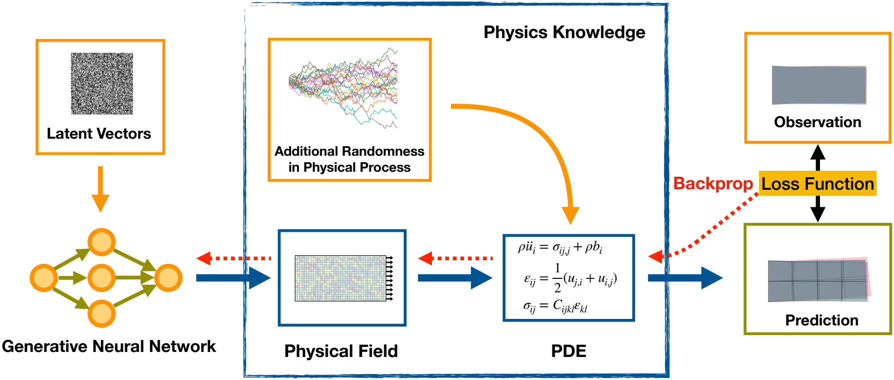
AD for Discrete Wasserstein Distance

Theorem

Assume that $\mathrm{p}^*([\underline{c}_{ij}])$ *is the optimal solution to the linear programming problem given the cost matrix* $[\underline{c}_{ij}]$*, then we have*

$$\frac{\partial W(\mathrm{p}^*([\underline{c}_{ij}]))}{\partial \underline{c}_{ij}} = p^*_{ij}$$

PhysGNN: Training Generative Neural Networks with Discrete Wasserstein Distance



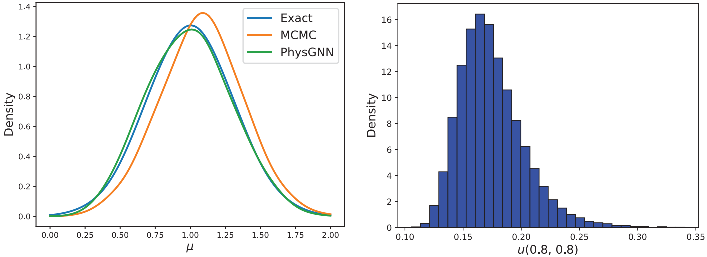
Numerical Benchmarks

- We consider a 2D Poisson's equation

$$\begin{aligned} \nabla\cdot(\kappa\nabla u) &= 2\pi^2\sin(\pi x)\sin(\pi y) && (x,y)\in(0,1)^2 \\ u(x,y) &= 0 && (x,y)\in\partial(0,1)^2 \end{aligned} \tag{7}$$

$$\kappa = |v|, \quad v\sim\mathcal{N}(1.0, 0.3^2)$$

- Observation: $u(0.8, 0.8)$
- $\sim$ 20,000 PDE solves for both MCMC and PhysGNN:

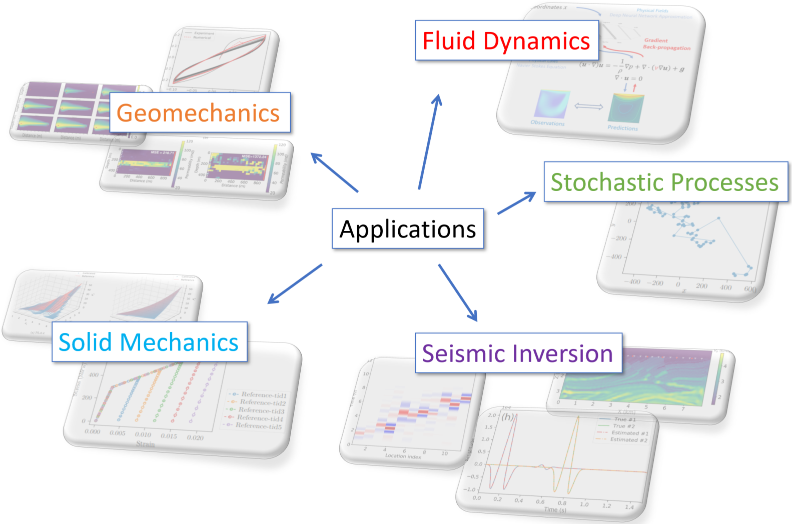


Summary

- Generative neural networks are used to approximate unknown distributions in a stochastic inverse problem.
- Training generative neural networks
  - Adversarial inverse modeling (AIM): solving a min-max optimization problem involving generator and discriminator nets
  - Physics Generative Neural Network (PhysGNN): minimizing a discrete Wasserstein distance

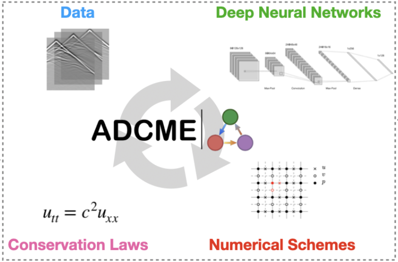
Outline

Applications



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

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

PLACEHOLDER