Physics Based Machine Learning for Inverse Problems

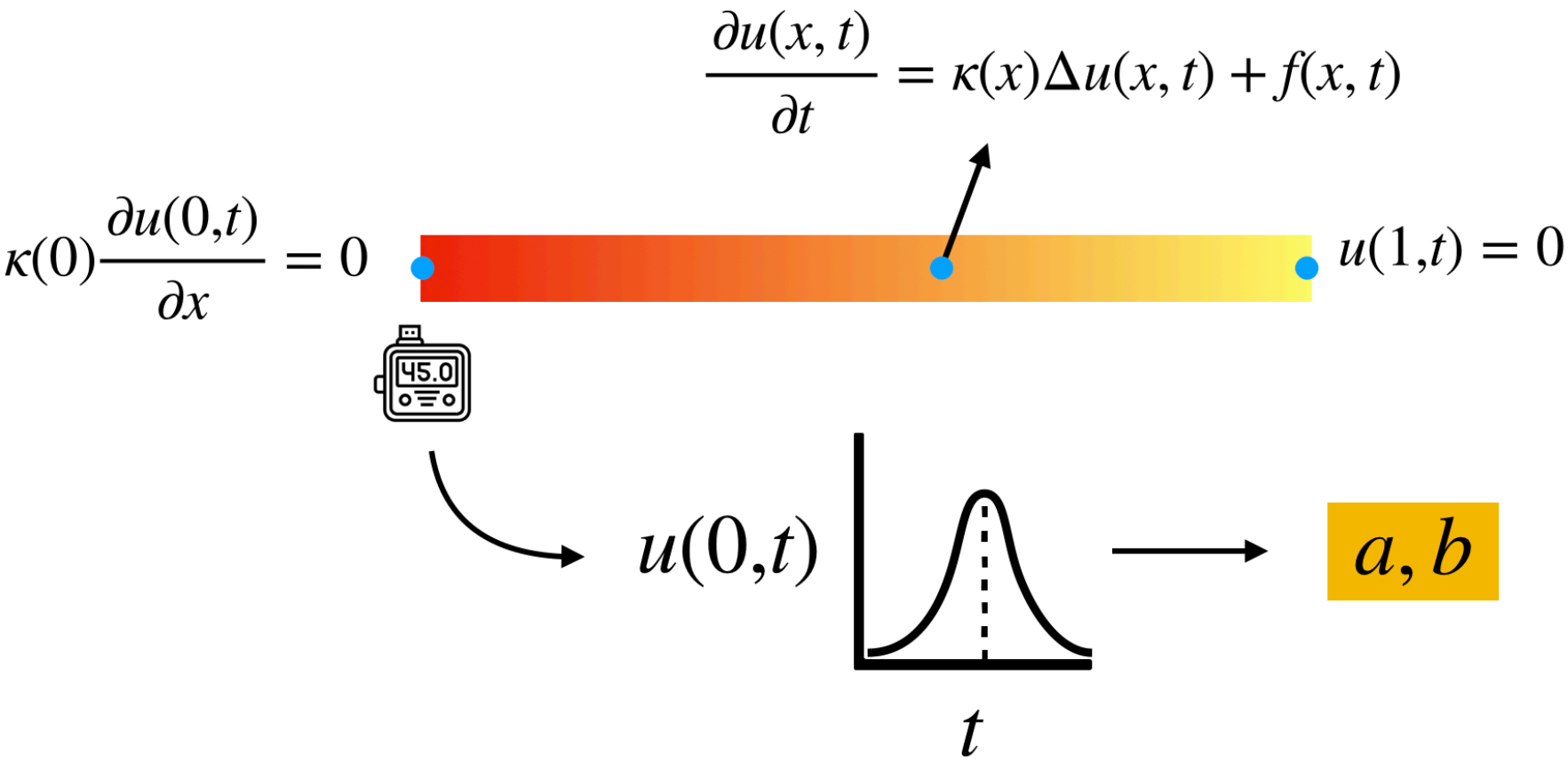
Kailai Xu and Eric Darve

Outline

Recap: Inverse Problem in Heat Transfer

- Goal: calibrate a and b from $u_0(t) = u(0, t)$

$$\kappa(x) = a + bx$$



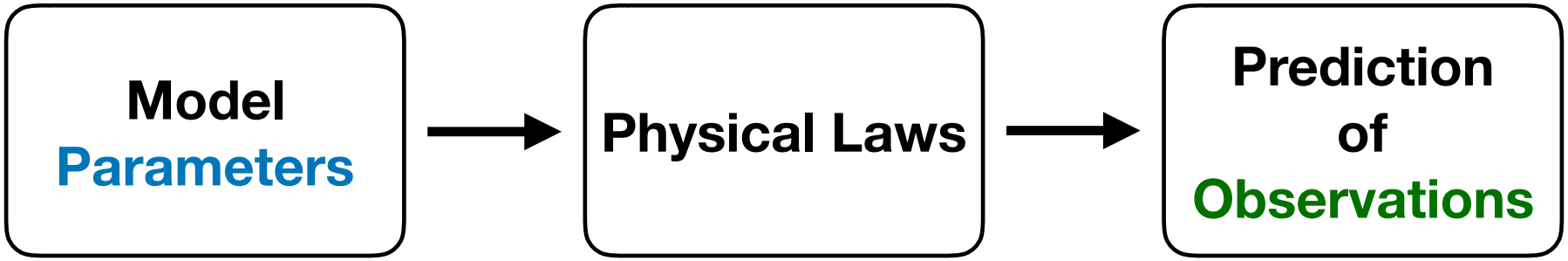
Mathematical Points of View

- This problem is a standard inverse problem. We can formulate the problem as a PDE-constrained optimization problem

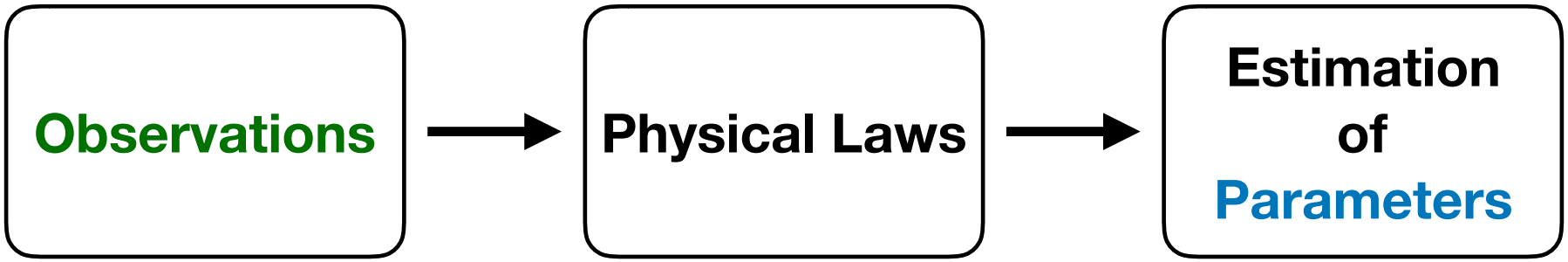
$$
\begin{aligned}
&\min_{a,b} \int_0^t (u(0,t) - u_0(t))^2 dt \\
&\text{s.t. } \frac{\partial u(x,t)}{\partial t} = \kappa(x)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1) \\
&\quad - \kappa(0)\frac{\partial u(0,t)}{\partial x} = 0, t > 0 \\
&\quad u(1,t) = 0, t > 0 \\
&\quad u(x,0) = 0, x \in [0,1] \\
&\quad \kappa(x) = ax + b
\end{aligned}
$$

Inverse Modeling

Forward Problem

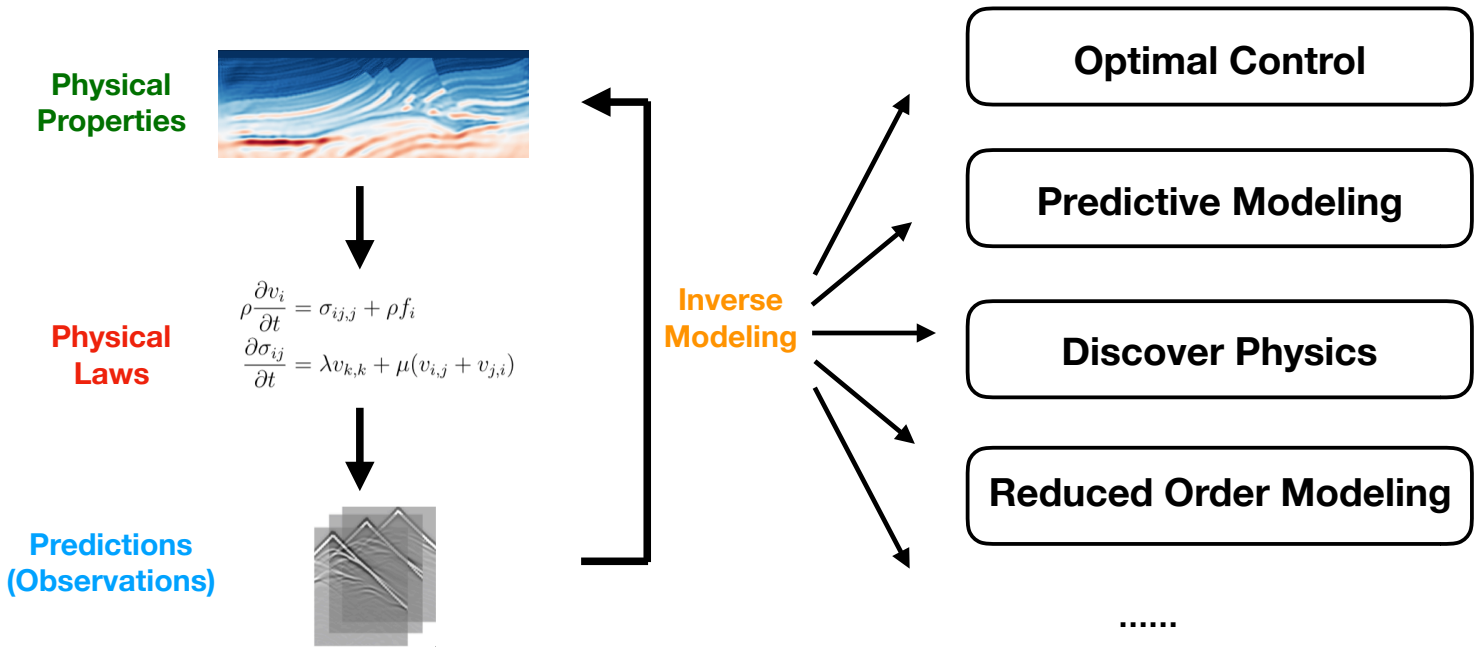


Inverse Problem



Inverse Modeling

- 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.



Parameter Inverse Problem

- The problem we consider so far falls into the category of parameter inverse problem, where the unknown is one or more parameters.

$$
\begin{aligned}
&\min_{a,b} \int_0^t (u(0,t) - u_0(t))^2 dt \\
&\text{s.t. } \frac{\partial u(x,t)}{\partial t} = \kappa(x)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1) \\
&\quad - \kappa(0)\frac{\partial u(0,t)}{\partial x} = 0, t > 0 \\
&\quad u(1,t) = 0, t > 0 \\
&\quad u(x,0) = 0, x \in [0,1] \\
&\quad \kappa(x) = ax + b
\end{aligned}
$$

Function Inverse Problem: Independent Case

- Another wide category of inverse problem is the so called function inverse problem, where the unknown is a function.
- First let us consider the case where κ is independent of the state variable u.

$$\min_{\kappa(x)} \int_0^t (u(0,t) - u_0(t))^2 dt$$

$$\text{s.t. } \frac{\partial u(x,t)}{\partial t} = \kappa(x)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1)$$

$$-\kappa(0)\frac{\partial u(0,t)}{\partial x} = 0, t > 0$$

$$u(1,t) = 0, t > 0$$

$$u(x,0) = 0, x \in [0,1]$$

Function Inverse Problem: Dependent Case.

- Another interesting scenario is that κ is dependent on u.

$$\min_{\kappa(x,u)} \int_0^t (u(0,t) - u_0(t))^2 dt$$

$$\text{s.t. } \frac{\partial u(x,t)}{\partial t} = \kappa(x,u)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1)$$

$$-\kappa(0,u(0))\frac{\partial u(0,t)}{\partial x} = 0, t > 0$$

$$u(1,t) = 0, t > 0$$

$$u(x,0) = 0, x \in [0,1]$$

Stochastic Inverse Problem

- In the fourth category, the unknown is a random variable $\kappa(\varpi)$, where ϖ is the outcome in the probability space.

$$\min_{\kappa(\varpi)} \int_0^t (u(0,t) - u_0(t))^2 dt$$

$$\text{s.t. } \frac{\partial u(x,t)}{\partial t} = \kappa(\varpi)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1)$$

$$-\kappa(\varpi)\frac{\partial u(0,t)}{\partial x} = 0, t > 0$$

$$u(1,t) = 0, t > 0$$

$$u(x,0) = 0, x \in [0,1]$$

Four Types of Inverse Problems

- Parameter Inverse Problem. The unknowns are constant scalars, vectors, matrices, or tensors. $\kappa(x) = a + bx$.
- Function Inverse Problem. The unknowns are functions:
  - The unknown function is independent of state variables. No function form of $\kappa(x)$ is given.
  - The unknown function is dependent on state variables. No function form of $\kappa(x, u)$ is given.
- Stochastic Inverse Problem. The unknown is a random variable. $\kappa(\varpi)$.

This lecture: function inverse problem.

Outline

Function Forms

- The key to solve function inverse problem is to parametrize the unknown function $\kappa(x)$ or $\kappa(x,u)$ using a function form

$$\kappa(x) \approx \kappa_\theta(x) \qquad \kappa(x,u) \approx \kappa_\theta(x,u)$$

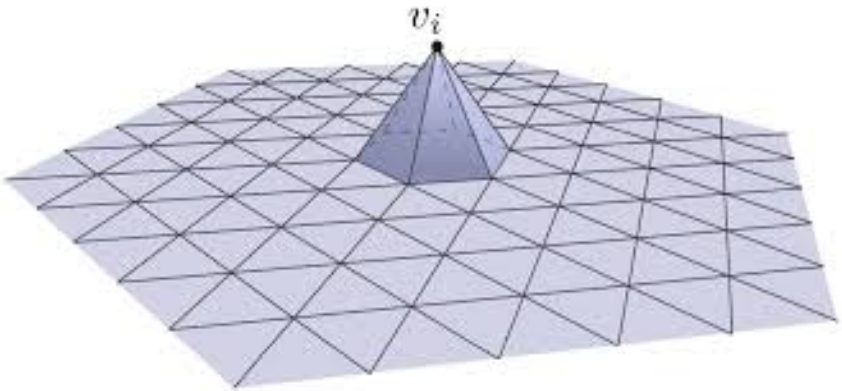
$$\begin{aligned}
\min_{\theta} \quad & \int_0^t (u(0,t) - u_0(t))^2 dt \\
\text{s.t.} \quad & \frac{\partial u(x,t)}{\partial t} = \kappa_\theta(x)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1) \\
& -\kappa_\theta(x)\frac{\partial u(0,t)}{\partial x} = 0, t > 0 \\
& u(1,t) = 0, t > 0 \\
& u(x,0) = 0, x \in [0,1]
\end{aligned}$$

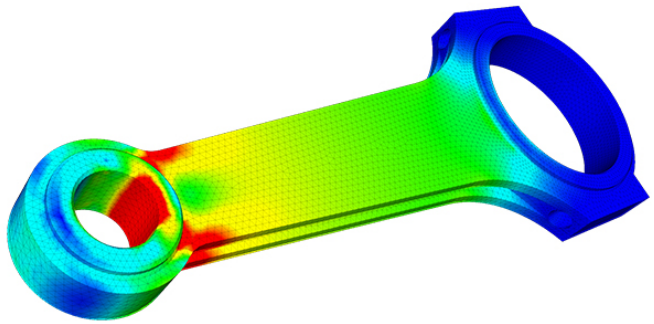
- Let's see a few function form examples.

Piecewise Linear Function

- Linear combination of piecewise linear basis functions ("hat functions").
- The building bricks of finite element analysis (FEA); FEA is the workhorse of many engineering applications (solid mechanics, structural engineering, etc.).

$$\kappa_\theta(x) = \sum_{i=1}^{n} c_i \varphi_i(x) \qquad \theta = \{c_i\}_{i=1}^{n}$$





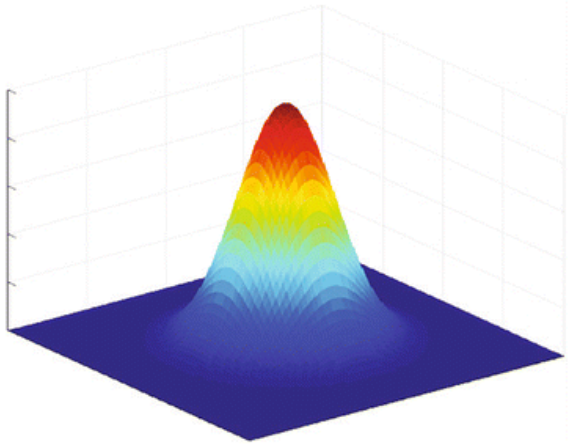
Radial Basis Function

- A radial basis function depends only on the radial distance form the "center" v_i (σ is the shape parameter)

$$\varphi_i(x) = g_\sigma(\|x - v_i\|)$$

- Linear combination of radial basis functions

$$\kappa_\theta(x) = \sum_{i=1}^{n} c_i \varphi_i(x) \qquad \theta = \{c_i\}_{i=1}^{n}$$



Other Classical Function Approximators

- Most classical function approximators are generalized linear models. Consider the 1D case

$$\kappa_\theta(x) = \sum_{i=1}^{n} c_i \varphi_i(x) \qquad \theta = \{c_i\}_{i=1}^{n}$$

  - Polynomial regression

$$\varphi_i(x) = x^{i-1}$$

  - Chebyshev polynomials

$$\varphi_i(x) = \begin{cases} \cos\left(n \arccos x\right), & \text{if } |x| \leq 1 \\ \cosh\left(n \operatorname{arcosh} x\right), & \text{if } x \geq 1 \\ (-1)^n \cosh\left(n \operatorname{arcosh}(-x)\right), & \text{if } x \leq -1 \end{cases}$$

  - B-splines

$$\varphi_i(x) = B_{i,p}(x)$$

Neural Networks

- Feed-forward neural network

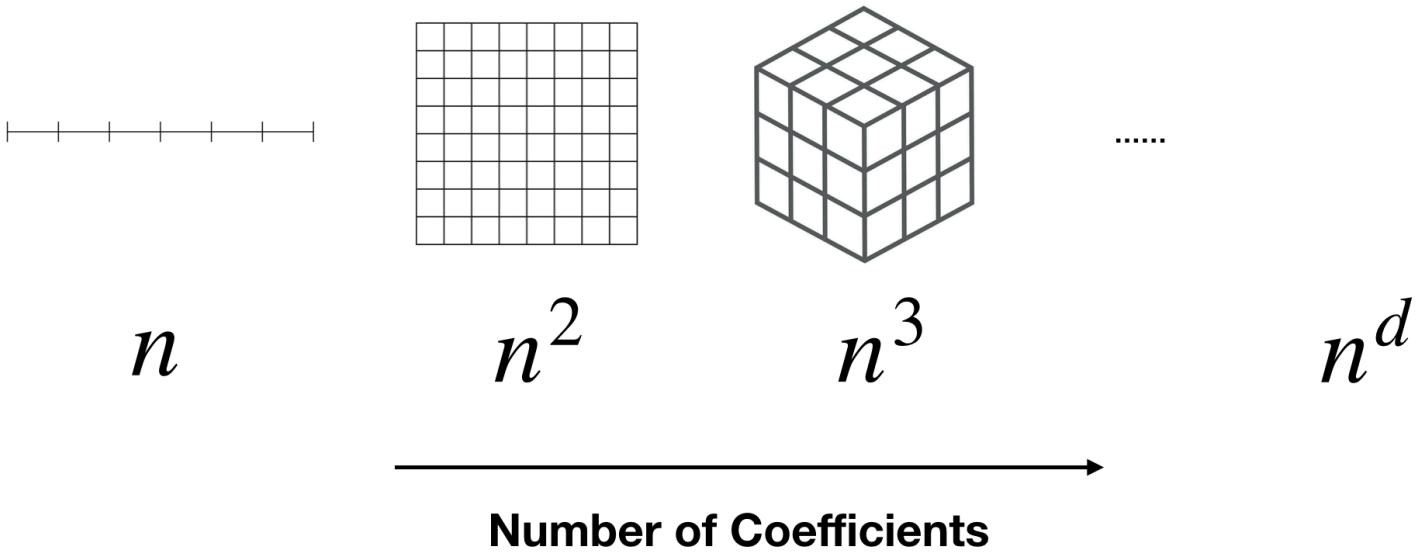
$$\left.\begin{aligned} \mathbf{y}_1 &= \tanh(\mathbf{W}_1 x + \mathbf{b}_1) \\ \mathbf{y}_2 &= \tanh(\mathbf{W}_2 \mathbf{y}_1 + \mathbf{b}_2) \\ &\dots \\ \mathbf{y}_{n-1} &= \tanh(\mathbf{W}_{n-1} \mathbf{y}_{n-2} + \mathbf{b}_{n-1}) \\ y &= \mathbf{W}_n \mathbf{y}_{n-1} + \mathbf{b}_n \end{aligned}\right\} \Rightarrow y = \kappa_\theta(x)$$

$$\theta = [\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \dots, \mathbf{W}_n, \mathbf{b}_n]$$

- It is another function approximator.
- It is NOT a linear combination of basis functions: composition of linear functions and nonlinear activation function.

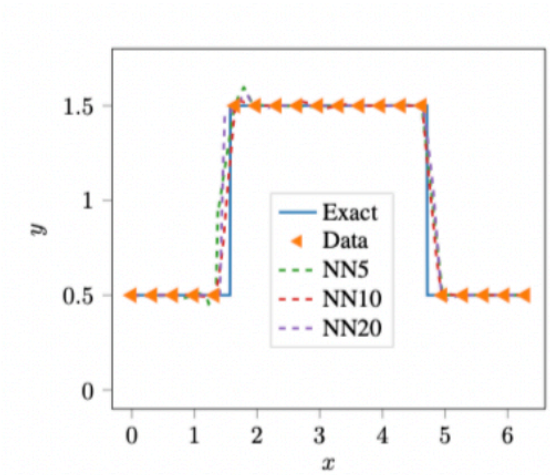
Neural Networks

- For generalized linear models, the number of coefficients typically grow exponentially in the dimension d.
- Implementing high dimensional generalized linear models is nontrivial, but it's extremely easy to extend neural networks to any high dimensional input/output space.

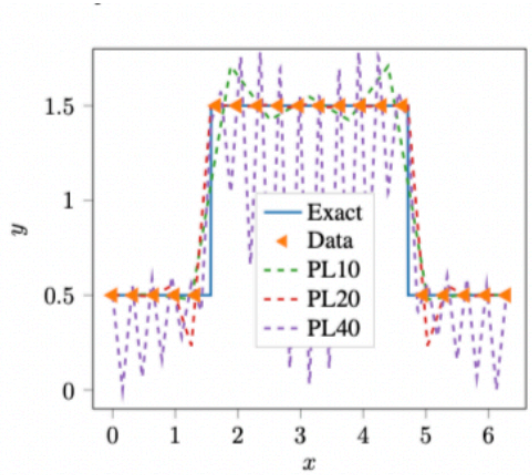


Neural Networks

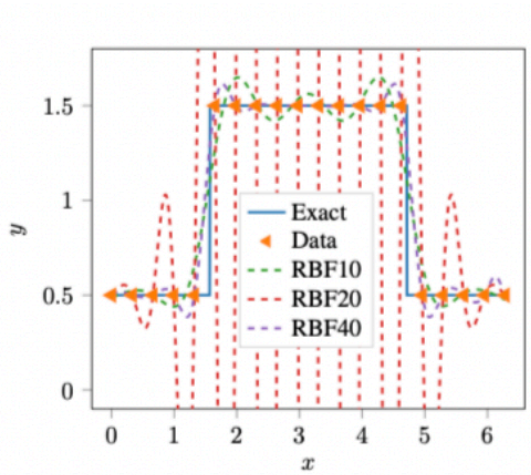
- Neural network is adaptive to discontinuities.



Neural Network



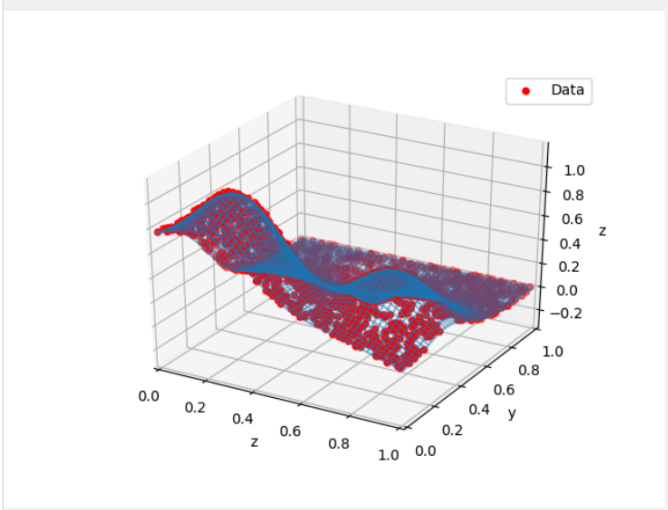
Piecewise Linear



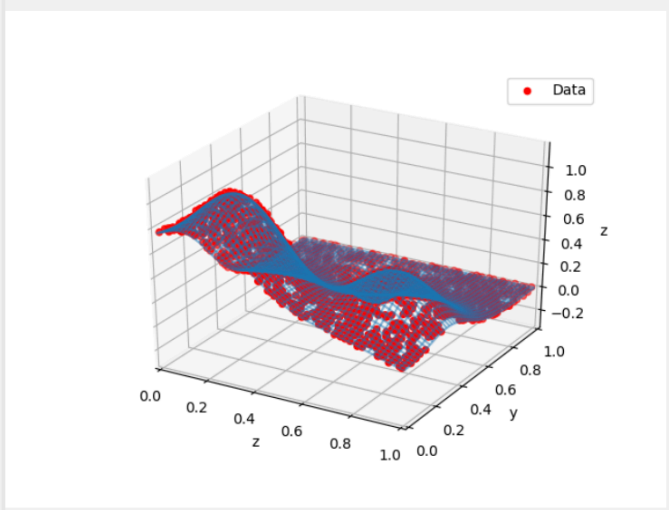
Radial Basis Functions

Neural Networks

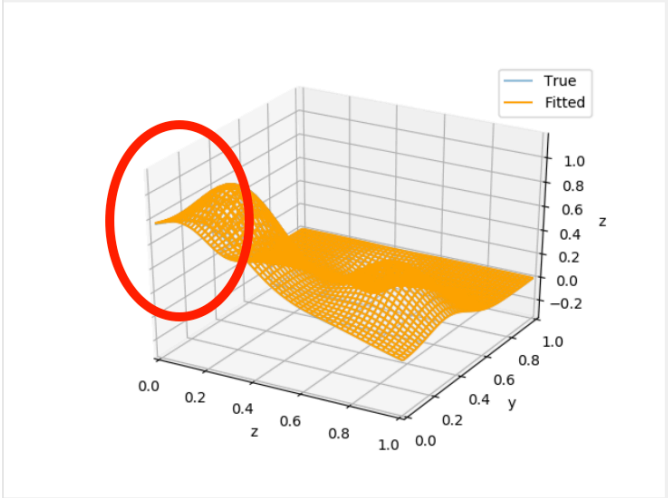
- Neural network is robust to noise. Left: NN; right: RBF.



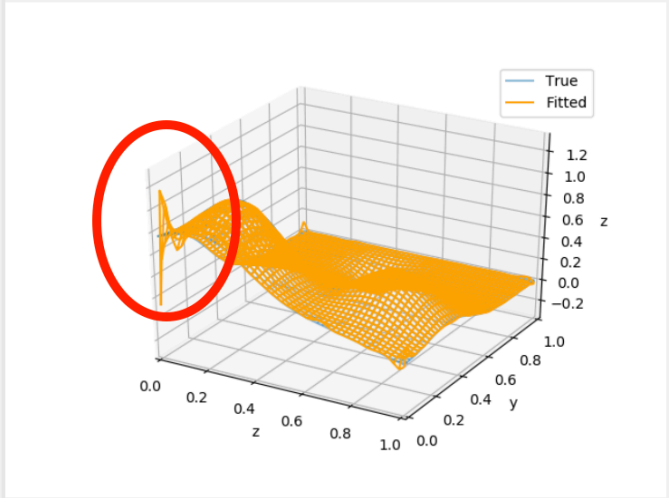
True Data



True Data



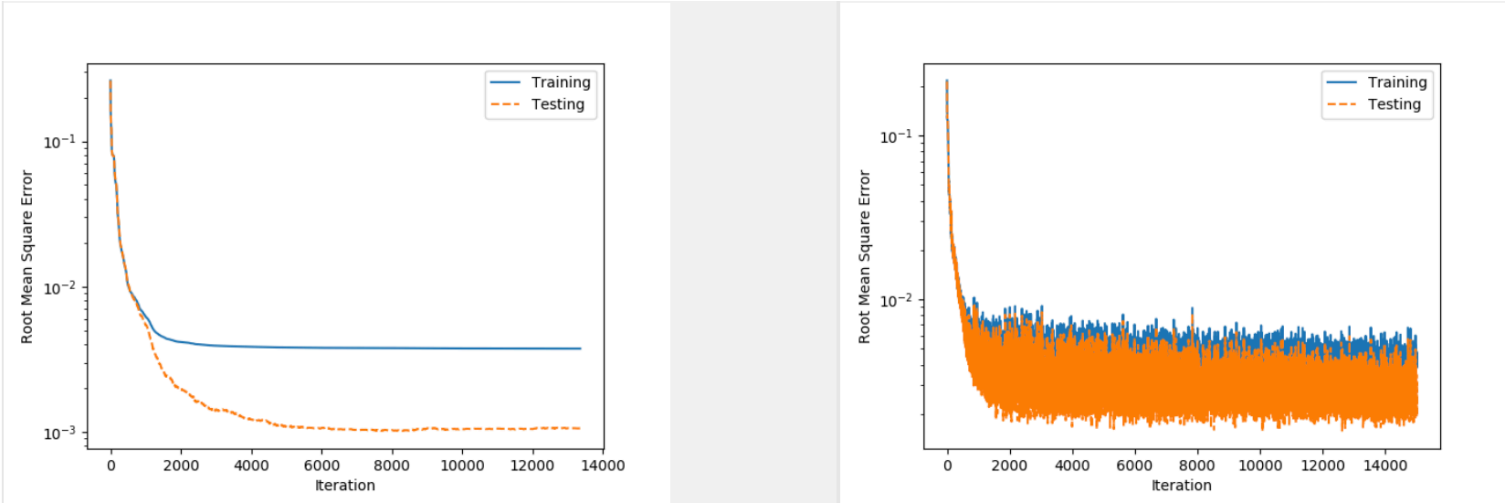
Fitted Surface



Fitted Surface

Neural Networks

- Quasi-Newton optimization (when it is affordable) is more efficient than stochastic gradient descent methods. Left: BFGS; right: SGD.



Take-Home Messages

- Neural network is easily extended to high dimensions.
- Neural network exhibits adaptiveness for discontinuous functions, compared to function approximators with fixed basis functions.
- Neural network is more robust to noise than traditional global basis functions such as radial basis functions.
- (Quasi-)second-order optimizers converge faster and are more stable than stochastic gradient descent methods, as long as you can afford the computational and memory cost.

Use Neural Networks to Parametrize Unknown Functions

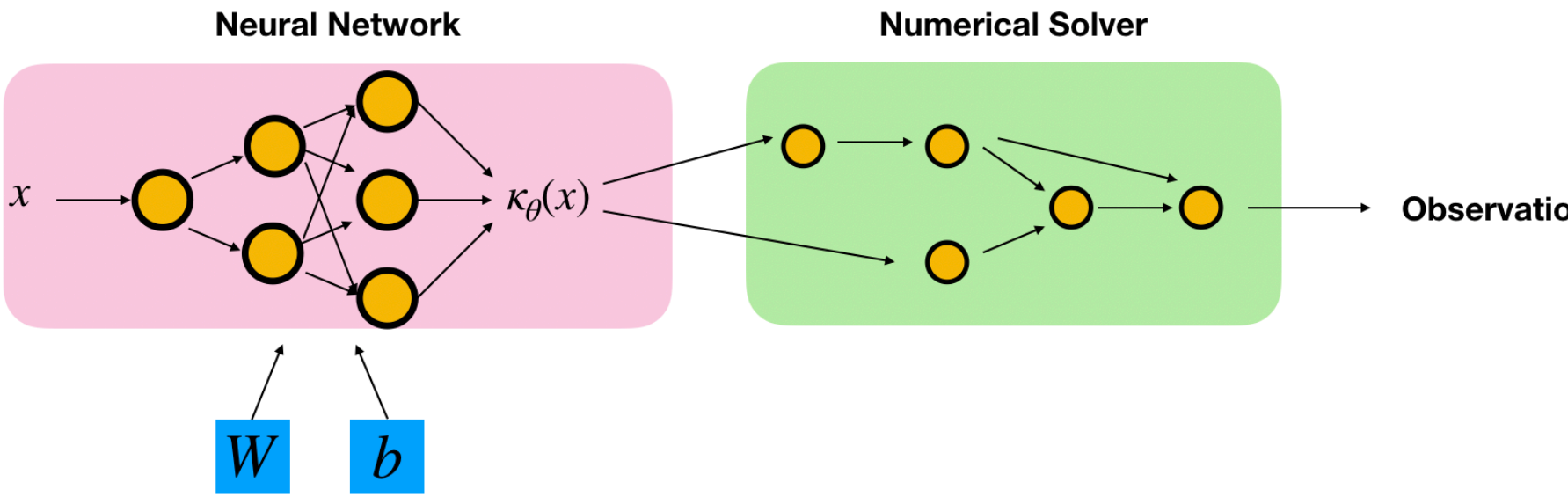
- Substitute $\kappa(x)$ with a neural network $\kappa_\theta(x)$, and then solve the PDE-constrained optimization problem:

$$\begin{aligned}
&\min_{\theta} \int_0^t (u(0,t) - u_0(t))^2 dt \\
&\text{s.t. } \frac{\partial u(x,t)}{\partial t} = \kappa_\theta(x)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1) \\
&\quad -\kappa_\theta(0)\frac{\partial u(0,t)}{\partial x} = 0, t > 0 \\
&\quad u(1,t) = 0, t > 0 \\
&\quad u(x,0) = 0, x \in [0,1]
\end{aligned}$$

- Major technical difficulty: how to train the neural networks (estimate θ)?

Computational Graph

- The computational graphs of a neural network and a numerical solver are coupled.



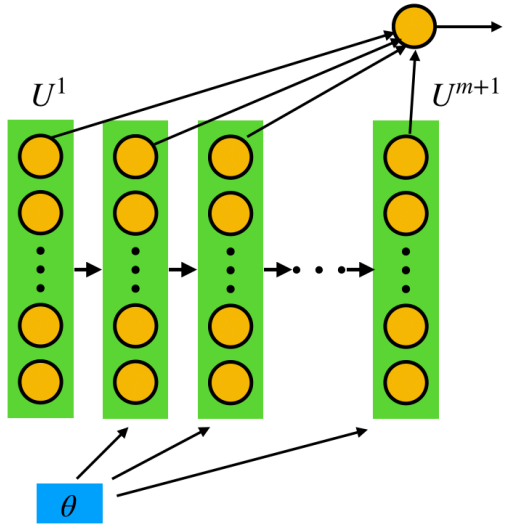
- Training Neural Networks: estimating the weights and biases of the neural network W, b by running gradient descent on the computational graph.

Computational Graph for Numerical Schemes

- The discretized optimization problem is

$$\min_{\theta} \sum_{k=1}^{m} (u_1^k - u_0((k-1)\Delta t))^2$$
$$\text{s.t. } A(\theta)U^{k+1} = U^k + F^{k+1}, k = 1, 2, \ldots, m$$
$$U^0 = 0$$

- The computational graph for the forward computation (evaluating the loss function) is



Outline

Direct Training

- If the input and output pairs

$$\{(u_i, x_i), \kappa_i\}_{i=1}^n$$

to $\kappa_\theta(u, x)$ are available, we can train the neural network using the standard supervised learning method

$$\min_\theta \sum_{i=1}^n (\kappa_\theta(u_i, x_i) - \kappa_i)^2$$

- Pros:
  - Extremely easy to implement using a deep learning software.
  - No insight of the PDE is required.
- Cons:
  - Input-output pair data may not be available.
  - Leads to nonphysical κ_θ by ignoring the PDE.

Residual Minimization

- Assumption: the full field data of the state variable $u(x, t)$ are available. Possible in laboratories.

- Solve an unconstrained optimization problem:

$$\min_{\theta} \sum_{j} \sum_{i=1}^{n} \left(\frac{\partial u}{\partial t}\Big|_{x=x_i, t=t_j} - \left(\kappa_\theta(x)\Delta u + f\right)\Big|_{x=x_i, t=t_j} \right)^2$$

Residual Minimization

- Pros:
  - No insight is required into numerical solvers; however, insight into the PDE is required.
  - Do not require input-output pair data.
- Cons:
  - Full field data is required.
  - Does not enforce the PDE constraints (the residual may not be zero due to local minimum).

Penalty Method

- Solve the constrained optimization problem using the penalty method

$$\min_{u,\theta} \int_0^t (u(0,t) - u_0(t))^2 dt$$
$$+ \lambda_1 \int_0^t \int_0^1 \left(\frac{\partial u(x,t)}{\partial t} - \kappa_\theta(x,u)\Delta u(x,t) - f(x,t) \right)^2 dtdx$$
$$+ \lambda_2 \int_0^t \int_0^1 \left(-\kappa_\theta(x,u) \frac{\partial u(0,t)}{\partial x} \right)^2 dtdx$$
$$+ \lambda_3 \int_0^t u(1,t)^2 dt + \lambda_4 \int_0^1 u(x,0)^2 dx$$

Here λ_1, λ_2, λ_3 and λ_4 are positive penalty parameters.

Penalty Method

- Pros:
  - No insight is required into numerical solvers; however, insight into the PDE is required.
  - Do not require input-output pair data.
- Cons:
  - The number of free optimization variables increase by the degrees of freedom (DOF) of state variable. This may be an issue for dynamic problems, where the DOF is very large (solution vectors at each time step must be added to free optimization variables).
  - Does not enforce the PDE constraints.
  - Selecting appropriate λ_i's is challenging.
  - Convergence is an issue for stiff problems.

Physics Constrained Learning

The most efficient and powerful approach.

- Physics constrained learning (PCL) solves for u first in the PDE constrained optimization.

Step1 Solve for u

$$\frac{\partial u(x,t)}{\partial t} = \kappa_\theta(x)\Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in (0,1)$$
$$-\kappa_\theta(x)\frac{\partial u(0,t)}{\partial x} = 0, t > 0$$
$$u(1,t) = 0, t > 0$$
$$u(x,0) = 0, x \in [0,1]$$

$$u = u_\theta(x,t)$$

Physics Constrained Learning

Step2 Solve the unconstrained optimization problem

$$\min_{\theta} \ \tilde{L}_h(\theta) := \int_0^t (u_\theta(0,t) - u_0(t))^2 dt$$

This step requires computing the gradients

$$\boxed{\frac{\partial \tilde{L}_h(\theta)}{\partial \theta}}$$

However, wse cannot express u_θ analytically in terms of θ.
Challenge: how to compute the gradient efficiently and automatically in a computational graph?

Physics Constrained Learning

- Let's consider a simple example

$$\min_{\theta} \|u - u_0\|^2$$
$$\text{s.t. } B(\theta)u = y$$

By definition:

$$\tilde{L}_h(\theta) := \|u_\theta - u_0\|^2$$

where u_θ is the solution to

$$B(\theta)u = y$$

Recap: Implicit Function Theorem

- Consider a function $f : x \mapsto y$, implicitly defined by

$$x^3 - (y^3 + y) = 0$$

- Treat y as a function of x and take the derivative on both sides

$$3x^2 - 3y(x)^2 y'(x) - 1 = 0$$

- Rearrange the expression and we obtain

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

Physics Constrained Learning

1. $$\frac{\partial \tilde{L}_h(\theta)}{\partial \theta} = 2(u_\theta - u_0)^T \frac{\partial u_\theta}{\partial \theta}$$

2. To compute $\frac{\partial u_\theta}{\partial \theta}$, consider the PDE constraint (θ is a scalar)

   $$B(\theta) u_\theta = y$$

   Take the derivative with respect to θ on both sides

   $$\frac{\partial B(\theta)}{\partial \theta} u_\theta + B(\theta) \frac{\partial u_\theta}{\partial \theta} = 0 \Rightarrow \frac{\partial u_\theta}{\partial \theta} = -B(\theta)^{-1} \frac{\partial B(\theta)}{\partial \theta} u_\theta$$

3. Finally,

   $$\frac{\partial \tilde{L}_h(\theta)}{\partial \theta} = -2(u_\theta - u_0)^T B(\theta)^{-1} \frac{\partial B(\theta)}{\partial \theta} u_\theta$$

Physics Constrained Learning

1. Remember: in reverse-mode AD, gradients are always back-propagated from downstream (objective function) to upstream (unknowns).
2. The following quantity is computed first:

$$g^T = 2(u_\theta - u_0)^T B(\theta)^{-1}$$

   which is equivalent to solve a linear system

$$B(\theta)^T g = 2(u_\theta - u_0)$$

3. In the gradient back-propagation step, a linear system with an adjoint matrix (compared to the forward computation) is solved.
4. Finally,

$$\frac{\partial \tilde{L}_h(\theta)}{\partial \theta} = -2(u_\theta - u_0)^T B(\theta)^{-1} \frac{\partial B(\theta)}{\partial \theta} u_\theta = -g^T \frac{\partial B(\theta)}{\partial \theta} u_\theta$$

Physics Constrained Learning

- A trick for evaluating $g^T B u_\theta$: consider g and u_θ as independent of θ in the computational graph, then

$$g^T \frac{\partial B(\theta)}{\partial \theta} u_\theta = \frac{\partial (g^T B(\theta) u_\theta)}{\partial \theta}$$

- $g^T B(\theta) u_\theta$ is a scalar, thus we can apply reverse-mode AD to compute $\frac{\partial (g^T B(\theta) u_\theta)}{\partial \theta}$.
- Declaring independence of variables can be done with `tf.stop_gradient` in TensorFlow or `independent` in ADCME.

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

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

Summary

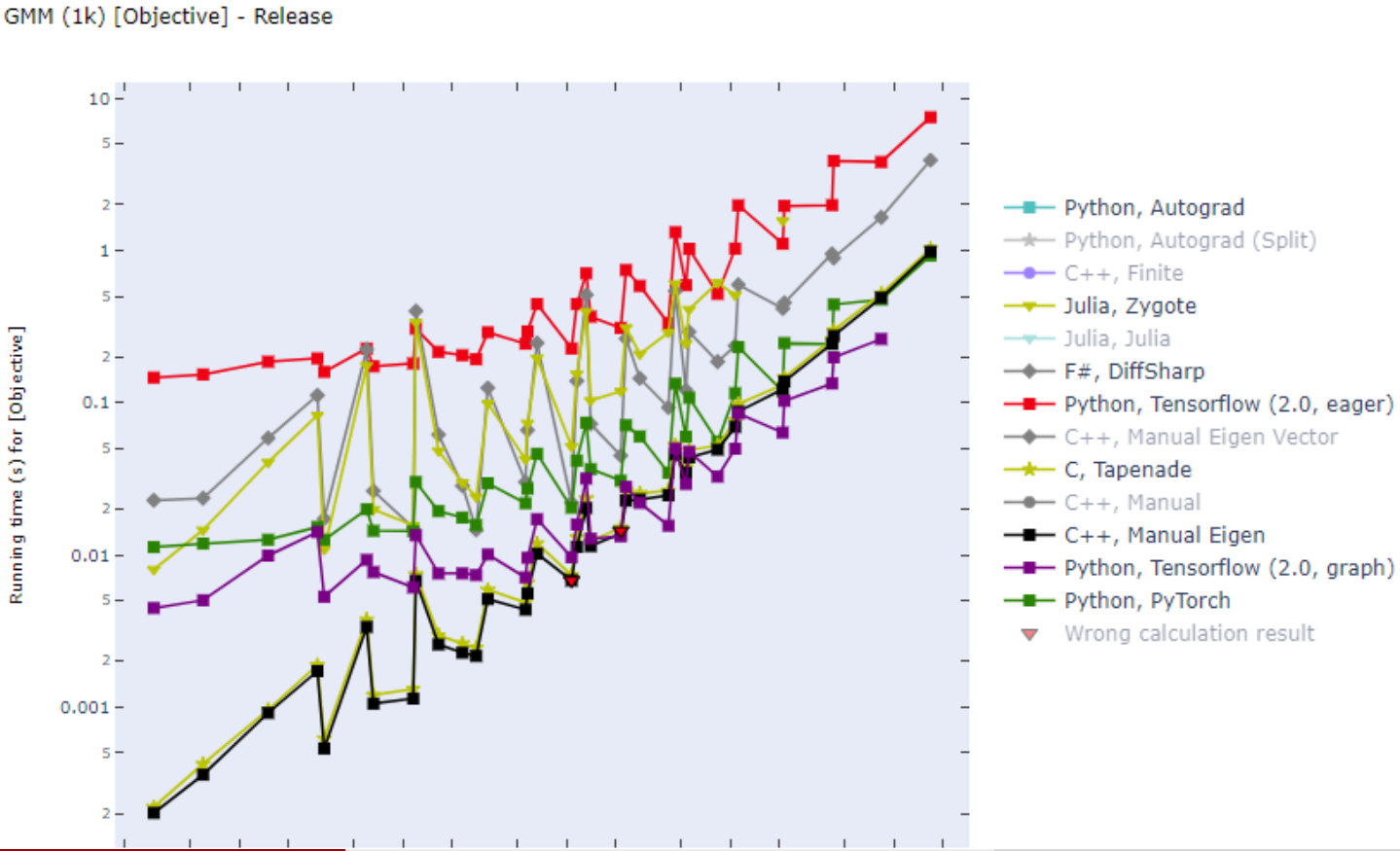
We compare the residual minimization method, penalty method, and physics constrained learning (PCL) from several aspects. We exclude the direct training method due to its limitation to input-output pairs.

Method	Residual Minimization	Penalty Method	Physics Constrained Learning
Sparse Observations	✗	✓	✓
Easy-to-implement	✓	✓	✗
Enforcing Physical Constraints	✗	✗	✓
Fast Convergence	✗	✗	✓
Minimal Optimization Variables	✓	✗	✓

Outline

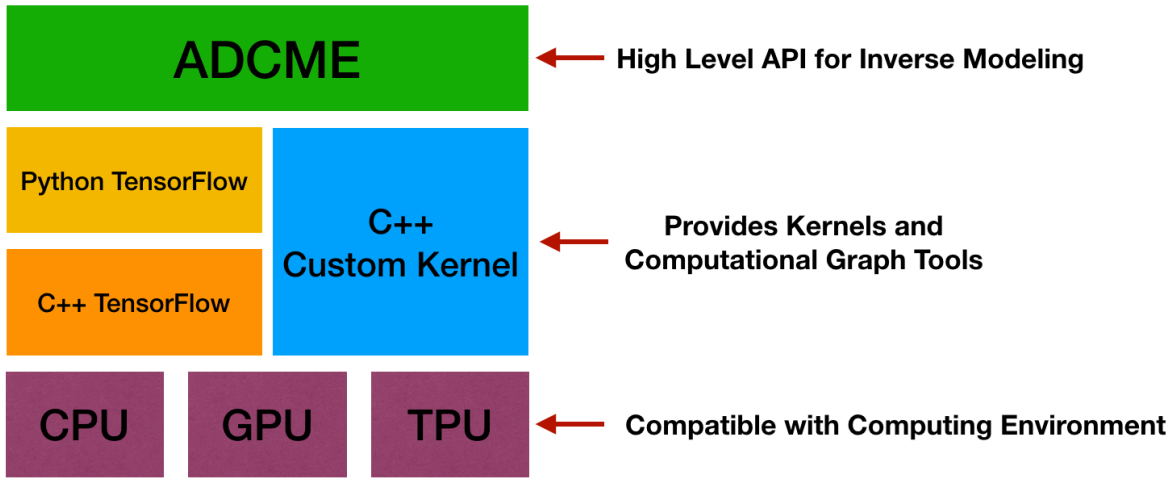
An Overview

- The ADCME library (Automatic Differentiation Library for Computational and Mathematical Engineering) aims at general and scalable inverse modeling in scientific computing with gradient-based optimization techniques.
- The automatic differentiation engine: TensorFlow static graph mode.



How ADCME works?

- Uses TensorFlow for computational graph-based optimization and generation of the computational graph for calculating gradients.
- Provides optimized C++ kernels and interfaces that are essential for scientific computing. Featured modules:
  - Sparse Linear Algebra Library.
  - Custom Optimizer, such as Ipopt and NLopt.
  - Neural Network with Tangent Matrices (sensitivity). See `fc` for details.
  - Probabilistic Metrics for stochastic inverse problems, such as `dtw` and `sinkhorn`.

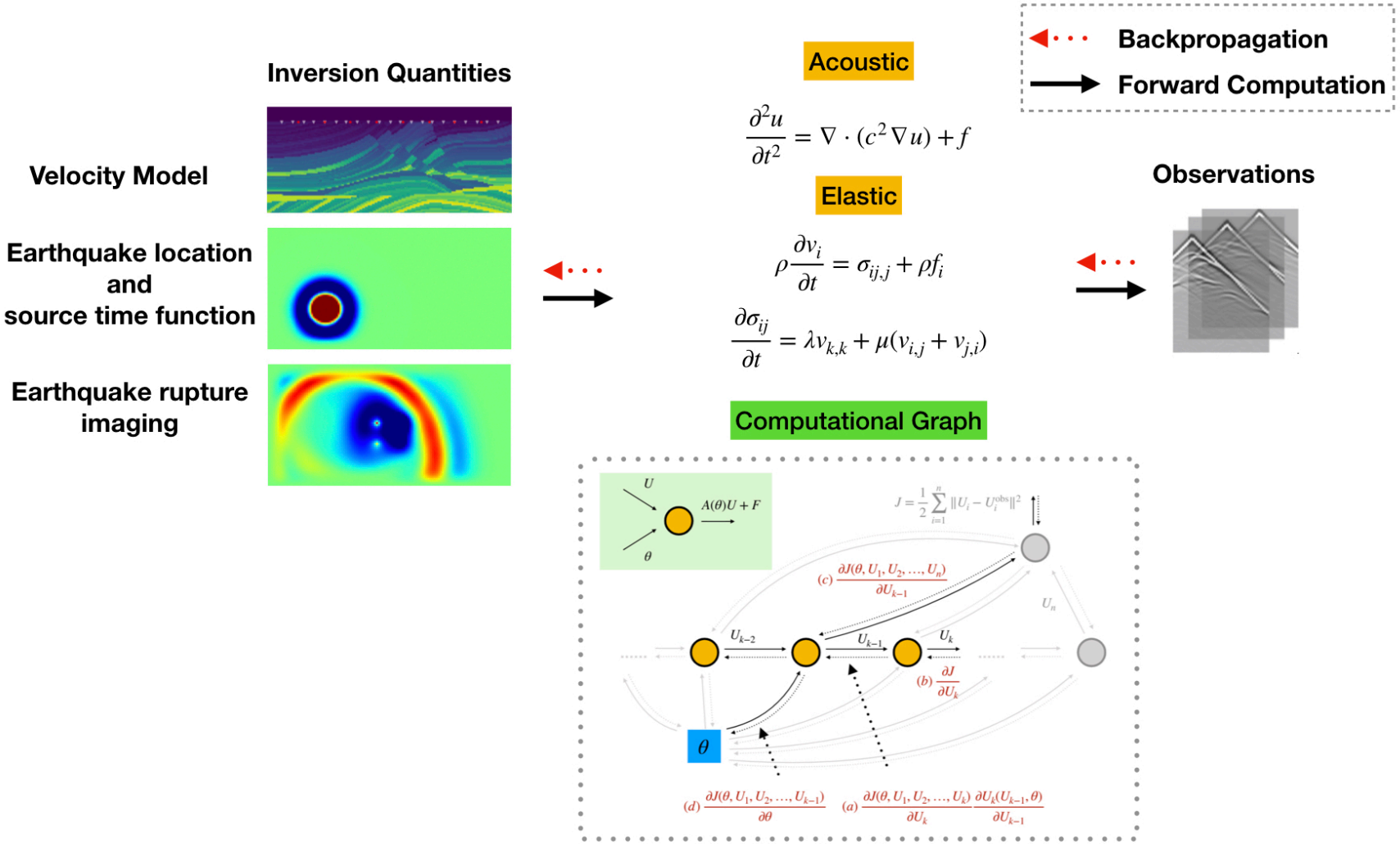


Featured Applications

Before we have some hands-on experience with inverse modeling using ADCME, let's first see some applications.

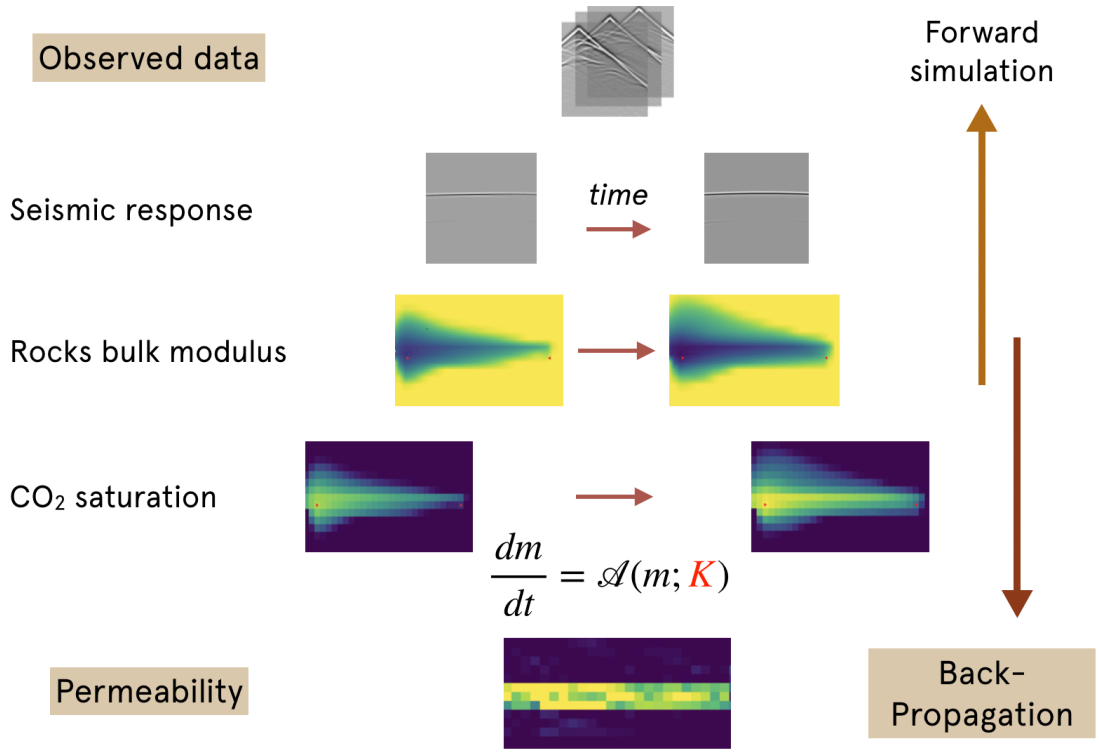
ADSeismic.jl: A General Approach to Seismic Inversion

- Solve seismic inversion problems within a unified framework.



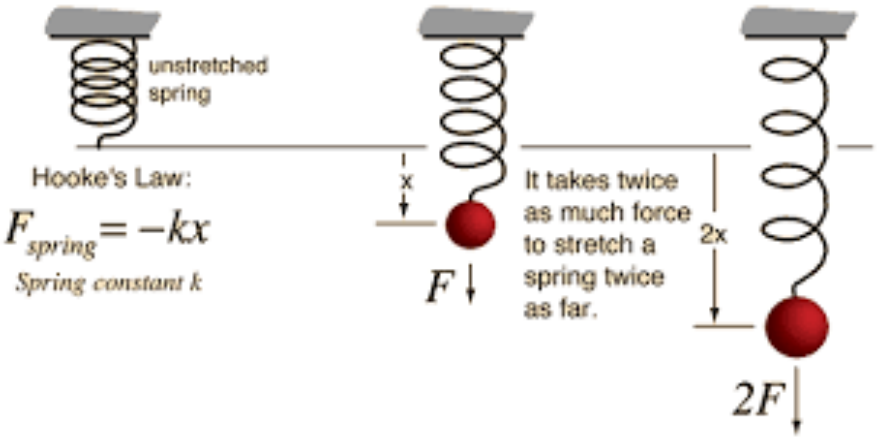
FwiFlow.jl: Coupled Full Waveform Inversion for Subsurface Flow Problems

- Estimating hydrological properties from high resolution geological data (e.g., seismic data).



NNFEM.jl: Robust Constitutive Modeling

- Modeling constitutive relations in dynamic structural equations using neural networks.
- A finite element library built on computational graph.



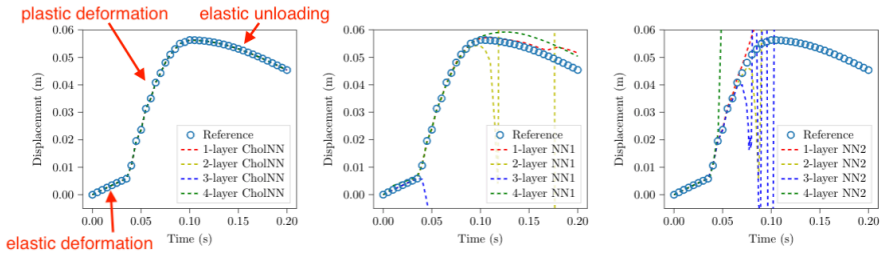


Image source: `http://hyperphysics.phy-astr.gsu.edu/hbase/permot2.html`

ADCME Example

Hands-on Example

Outline

Conclusion

- What's covered in this lecture
  - Four types of inverse problems:
    - Parameter inverse problem;
    - Function inverse problem (covered in this lecture);
    - Stochastic inverse problem.
  - Neural networks as a function approximation form;
  - Training algorithms:
    - Direct training;
    - Residual minimization;
    - Penalty method;
    - Physics constrained learning.
  - ADCME: applications and hands-on examples