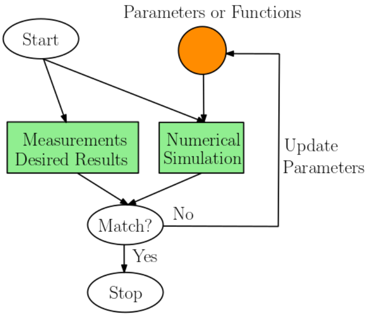
ADCME: Automatic Differentiation Library for Computational and Mathematical Engineering

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

November 7, 2019

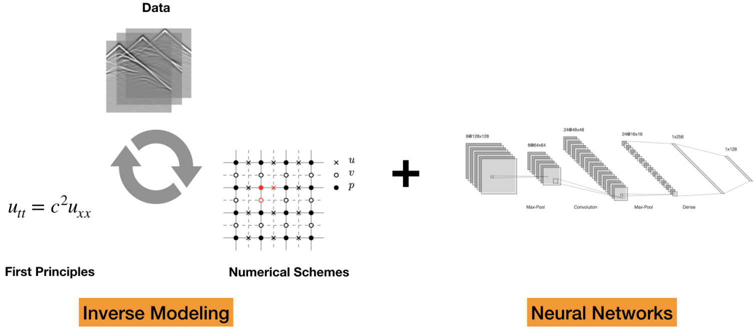
Inverse Modeling

- **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

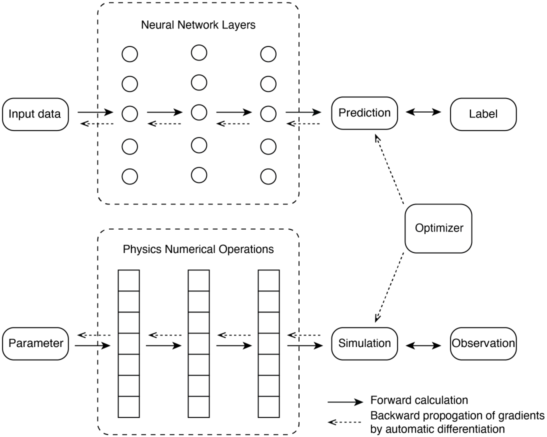
- Traditional inverse modeling methods utilizes efficient numerical schemes and incorporates physical knowledge (first principles); deep learning learns statistical relations from large amounts of training data.
- We combine the best of the two worlds and create **physics based machine learning**.



Automatic Differentiation

- Deep learning and inverse modeling have the same computational model but disguised under different terminologies.

Back-propagation = Automatic Differentiation = Discrete Adjoint State Method



AD Implementation in ADCME

- ADCME allows users to use high level script language Julia to implement numerical simulation codes, but obtain the extremely powerful parallelism and scalability provided by TensorFlow and Julia itself.
- Gradients are computed automatically.

```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
```

Julia code

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

Discretization

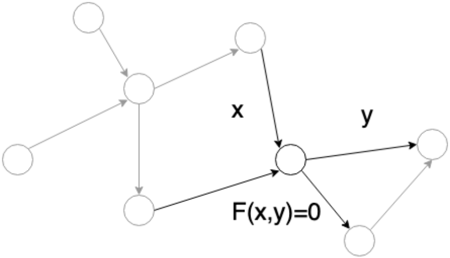
$$
\begin{aligned}
u_{tt}+(\zeta_1+\zeta_2)u_t+\zeta_1\zeta_2u&=\nabla\cdot(c^2\nabla u)+\nabla\cdot\boldsymbol{\phi},\\
\boldsymbol{\phi}_t&=\Gamma_1\boldsymbol{\phi}+c^2\Gamma_2\nabla u,
\end{aligned}
$$

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

PML equations

Challenges in AD

- ADCME aims to solve the nonlinear implicit operator case via custom operators.
- Another ongoing effort is automatic calculation of Jacobian (`IGACS.jl`).



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

Framework

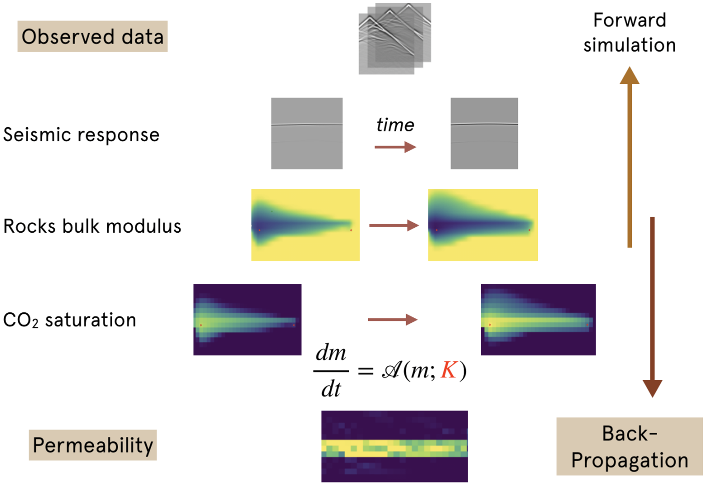
- Most inverse modeling problems can be classified into 4 categories. For example, the PDE for describing physics is

$$\nabla \cdot (X \nabla u) = 0 \quad \mathcal{BC}(u) = 0 \tag{1}$$

We observe some quantities depending on the solution u and want to estimate X.

Expression	Description	ADCME Solution	Note
$\nabla \cdot (a \nabla u) = 0$	Parameter Inverse Problem	Discrete Adjoint State Method	Direct Optimize a
$\nabla \cdot (f(x) \nabla u) = 0$	Functional Inverse Problem	Neural Network Functional Approximator	$f(x) \approx f_\theta(x)$
$\nabla \cdot (f(u) \nabla u) = 0$	Relation Inverse Problem	Deep Learning for Indirect Data	$f(u) \approx f_\theta(u)$
$\nabla \cdot (\varpi \nabla u) = 0$	Stochastic Inverse Problem	Adversarial Numerical Analysis	Generative Neural Nets for ϖ (unknown random processes)

Parameter Inverse Problem: Learning Hidden Geophysical Processes



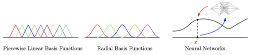
Functional Inverse Problem: Calibrating Lévy Processes

$$\phi(\boldsymbol{\xi}) = \mathbb{E}[e^{\mathrm{i}\langle \boldsymbol{\xi}, \mathbf{X}_t\rangle}] =$$

$$\exp\left[t\left(\mathrm{i}\langle \mathbf{b}, \boldsymbol{\xi}\rangle - \frac{1}{2}\langle \boldsymbol{\xi}, \mathbf{A}\boldsymbol{\xi}\rangle + \int_{\mathbb{R}^d}\left(e^{\mathrm{i}\langle \boldsymbol{\xi}, \mathbf{x}\rangle} - 1 - \mathrm{i}\langle \boldsymbol{\xi}, \mathbf{x}\rangle \mathbf{1}_{\|\mathbf{x}\|\leq 1}\right)\nu(d\mathbf{x})\right)\right]$$

Model

$$\nu(\mathbf{x}) \approx \nu_{\boldsymbol{\theta}}(\mathbf{x})$$

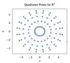


$$\downarrow$$

$$\phi(\boldsymbol{\xi}) \approx \phi_{\boldsymbol{\theta}}(\boldsymbol{\xi}) := \exp\left[\Delta t \sum_{i=1}^{n_q}\left(e^{\mathrm{i}\langle \boldsymbol{\xi}, \mathbf{x}_i\rangle} - 1 - \mathrm{i}\langle \boldsymbol{\xi}, \mathbf{x}_i\rangle \mathbf{1}_{\|\mathbf{x}_i\|\leq 1}\right)\nu_{\boldsymbol{\theta}}(\mathbf{x}_i) w_i\right]$$

$$\downarrow$$

$$\boxed{\min_{\boldsymbol{\theta}} \frac{1}{m}\sum_{i=1}^{m} \|\hat{\phi}_n(\boldsymbol{\xi}_i) - \phi_{\boldsymbol{\theta}}(\boldsymbol{\xi}_i)\|^2}$$



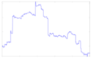
$$\uparrow$$

Data

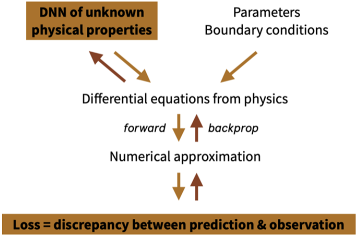
$$\hat{\phi}_n(\boldsymbol{\xi}) := \frac{1}{n}\sum_{i=1}^{n} \exp(\mathrm{i}\langle \boldsymbol{\xi}, \mathbf{X}_{i\Delta t} - \mathbf{X}_{(i-1)\Delta t}\rangle),\ \boldsymbol{\xi} \in \mathbb{R}^d$$

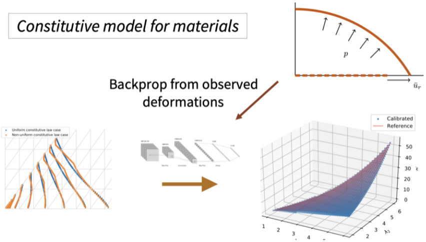
$$\uparrow$$

$$\{\mathbf{X}_{i\Delta t}\}_i$$



Relation Inverse Problem: Learning Constitutive Relations

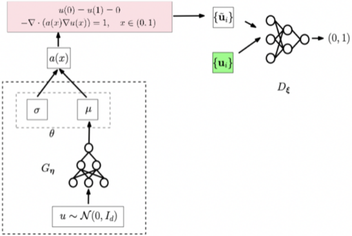


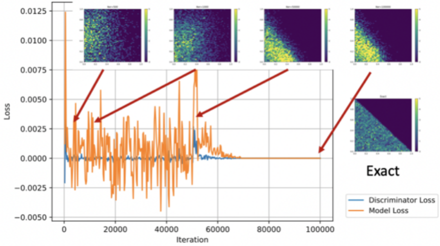


Probability Inverse Problem: Adversarial Numerical Analysis

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

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





A Cool Application: `ADSeismic.jl`

- An Open Source High Performance Package for General Seismic Inversion Problems
- Problems include:
  - Full waveform inversion (FWI);
  - Rupture inversion;
  - Source-time inversion.
- Features:
  - (Multi-)GPU support;
  - Easy-to-use;
  - Easily extendable.