

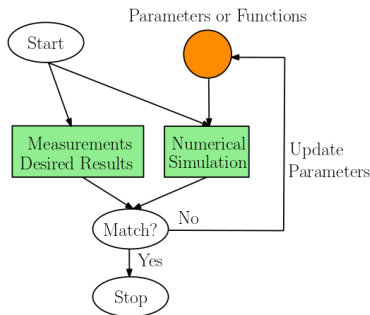
ADCME: Automatic Differentiation Library for Computational and Mathematical Engineering

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

November 26, 2019

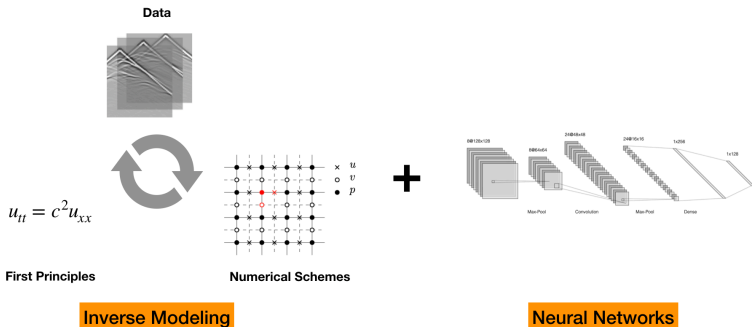
Inverse Modeling

- **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

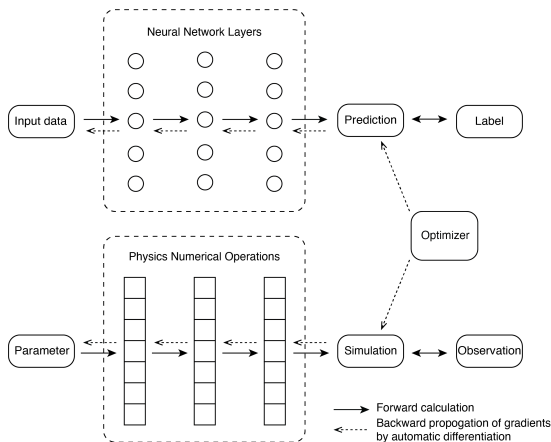
- Traditional inverse modeling methods utilize efficient numerical schemes and incorporate physical knowledge (first principles); deep learning learns statistical relations from large amounts of training data.
- We combine the best of the two worlds and invent **physics based machine learning**.



Automatic Differentiation

- Deep learning and inverse modeling have the same computational model but are 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 powerful parallelism and scalability provided by TensorFlow and Julia itself.
- Gradients are computed automatically.

```
function one_step(param::T, t::T, PropagatorParams, w::PyObject, wold::PyObject, φ, ψ,
    c::PyObject, v::PyObject, cc::PyObject)
    Δt = param.DELTA_T
    hx, hy = param.DELTA_X, param.DELTA_Y
    I1, I2, In1, I3p, I2n, In2n, In3p, In2n =
        param.I1, param.I2, param.In1, param.I3p, param.I2n, param.In2n, param.I3pn, param.I2pn, param.In2pn, param.In3pn
    u = (2 - α[I1]+α[I2]*Δt+2 - 2Δt^2/hx^2 + α[I3] - 2Δt^2/hy^2 + c[I1]) * u[I1] +
        (c[I2] + Δt/hx)^2 * w[In1] +
        (c[I3] + Δt/hy)^2 * w[I3pn] +
        (Δt^2/(2hx)) * (w[I3p]-q[I3n]) +
        (Δt^2/(2hy)) * (w[I2p]-q[I2n]) -
        ((1 - α[I1]+α[I2]*Δt/2) * wold[I1] -
            (1 - α[I1] + c[I1]*Δt)/2 * h_x) *
            u = vector{I1, u, (param.NX+2)*(param.NY+2)}
    q = (1 - α[note(I2)]) * q[I2] + Δt * c[I3] * (c[I1] - α[I1]) / 2 * h_y *
        (u[I3p] - u[In])
    q = (1 - α[note(I3)]) * q[I3] + Δt * c[I3] * (α[I1] - α[I1]) / 2 * h_x *
        (u[I2p] - u[In])
    q = vector{I1, q, (param.NX+2)*(param.NY+2)}
    ψ = vector{I1, ψ, (param.NX+2)*(param.NY+2)}
    u, ψ, φ
```

Julia code

$$\begin{aligned}
& \frac{u_{i,j}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1}}{\Delta t^3} + (\zeta_{1i} + \zeta_{2j}) \frac{u_{i,j}^{n+1} - u_{i,j}^{n-1}}{2\Delta t} + (\zeta_{1i}\zeta_{2j} + \zeta_{2i}\zeta_{1j} + \zeta_{3i}\zeta_{1j})u_{i,j}^{n+1} \\
&= \frac{c_{i,j}^{n+1}u_{i,j}^{n+1} - (c_{i,j}^{n+1} + c_{i,j}^{n-1})u_{i,j}^n + c_{i,j}^{n-1}u_{i,j}^{n-1}}{\Delta t_1^2} \\
&+ \frac{c_{i,j}^{n+1} + u_{i,j}^{n+1} - (c_{i,j}^{n+1} + u_{i,j}^{n-1})u_{i,j}^n + c_{i,j}^{n-1}u_{i,j}^{n-1}}{\Delta t_2^2} \\
&+ \frac{c_{i,j}^{n+1} + u_{i,j}^{n+1} - (c_{i,j}^{n+1} + u_{i,j}^{n-1})u_{i,j}^n + c_{i,j}^{n-1}u_{i,j}^{n-1}}{\Delta t_3^2} \\
&+ \frac{\theta_{i,j}^{n+1} - \theta_{i,j}^{n-1}}{\Delta t_1} + \frac{\theta_{i,j}^{n+1} - \theta_{i,j}^{n-1}}{\Delta t_2} + \frac{\theta_{i,j}^{n+1} - \theta_{i,j}^{n-1}}{\Delta t_3} - \zeta_{1i}\zeta_{2j}\zeta_{3i}\frac{u_{i,j}^{n+1} + u_{i,j}^{n-1}}{2}
\end{aligned}$$

$$u_{tt} + (\zeta_1 + \zeta_2)u_t + \zeta_1 \zeta_2 u = \nabla \cdot (c^2 \nabla u) + \nabla \cdot \phi,$$

$$\phi_t = \Gamma_1 \phi + c^2 \Gamma_2 \nabla u,$$

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

PML equations

Discretization

Code Example

- Find b such that $u(0.5) = 1.0$ and

$$-bu''(x) + u(x) = 8 + 4x - 4x^2, x \in [0, 1], u(0) = u(1) = 0$$

```
using LinearAlgebra
using ADCME

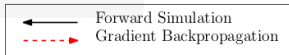
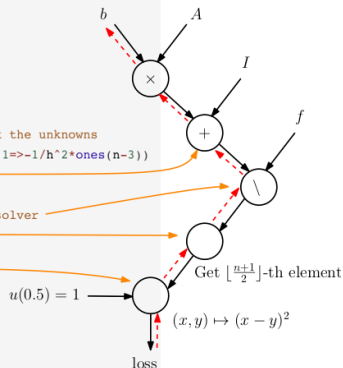
n = 101 # number of grid nodes in [0,1]
h = 1/(n-1)
x = LinRange(0,1,n)[2:end-1]
```

```
b = Variable(10.0) # we use Variable keyword to mark the unknowns
A = diagm(0=>2/h^2*ones(n-2), -1=>-1/h^2*ones(n-3), 1=>-1/h^2*ones(n-3))
B = b*A + I # I stands for the identity matrix
f = @. 4*(2 + x - x^2)
u = B\f # solve the equation using built-in linear solver
ue = u[div(n+1,2)] # extract values at x=0.5
```

```
loss = (ue-1.0)^2
```

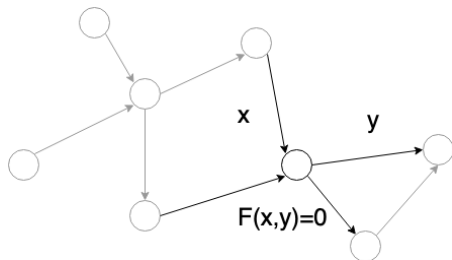
```
# Optimization
sess = Session(); init(sess)
BFGS!(sess, loss)
```

```
println("Estimated b = ", run(sess, b))
```



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	$F(x, y) = 0$

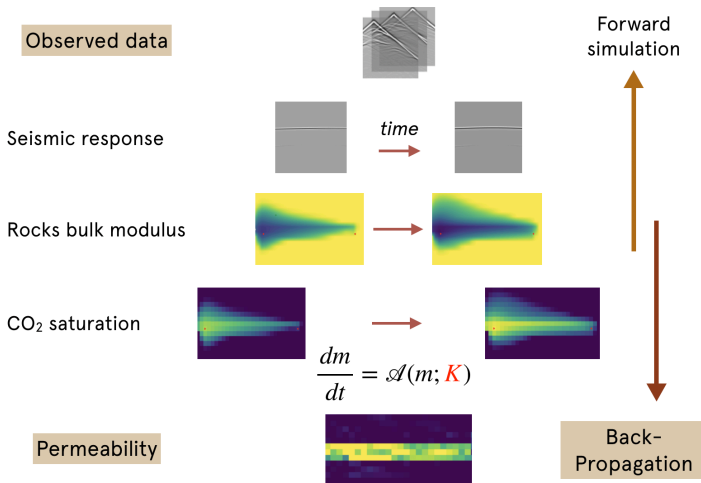
- Most inverse modeling problems can be classified into 4 categories. For example, the PDE for describing physics is

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

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

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

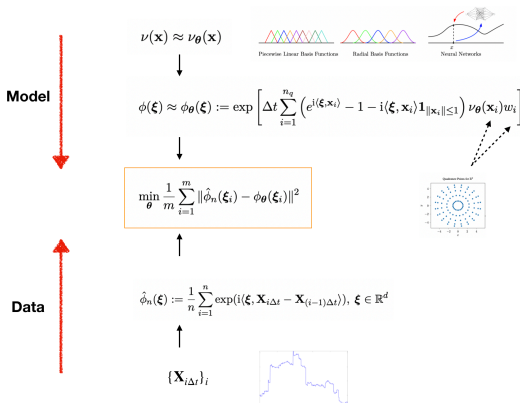
Parameter Inverse Problem: Learning Hidden Geophysical Processes



Functional Inverse Problem: Calibrating Lévy Processes

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

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



Relation Inverse Problem: Learning Constitutive Relations

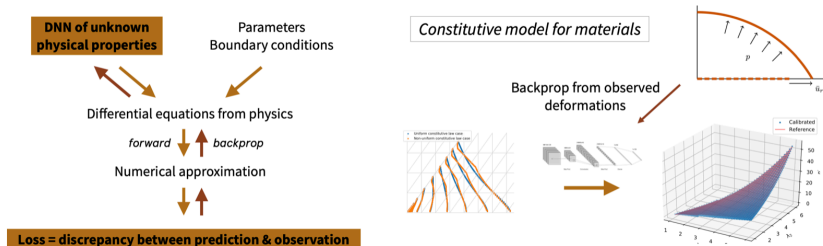
- Equilibrium equation

$$\mathcal{P}(u(\mathbf{x}), \mathcal{M}(u(\mathbf{x}), \dot{u}(\mathbf{x}), \mathbf{x})) = \mathcal{F}(u(\mathbf{x}), \mathbf{x}, p)$$

- Neural Network Approximation:

$$\mathcal{M}_\theta(\mathbf{u}) \approx \mathcal{M}(u(\mathbf{x}), \dot{u}(\mathbf{x}), \mathbf{x})$$

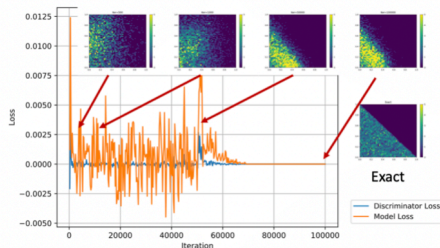
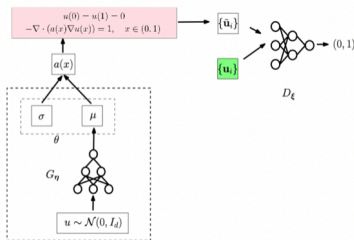
$$\min_{\theta} \|\mathcal{P}(\mathbf{u}, \mathcal{M}_\theta(\mathbf{u})) - \mathcal{F}(\mathbf{u}, \mathbf{x}, p)\|_2^2$$



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 (Coming soon)

- 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.

