CME216 Lecture: Automatic Differentiation for Computational Engineering

Kailai Xu and Eric Darve

Outline

Overview

- Gradients are useful in many applications
  - Mathematical Optimization

    $$\min_{x \in \mathbb{R}^n} f(x)$$

    Using the gradient descent method:

    $$x_{n+1} = x_n - \alpha_n \nabla f(x_n)$$

  - Sensitivity Analysis

    $$f(x + \Delta x) \approx f'(x)\Delta x$$

  - Machine Learning
    Training a neural network using automatic differentiation (back-propagation).
  - Solving Nonlinear Equations

    Solve a nonlinear equation $f(x) = 0$

    Using Newton's method

    $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Terminology

- Deriving and implementing gradients are a challenging and all-consuming process.
- Automatic differentiation: a set of techniques to numerically evaluate the derivative of a function specified by a computer program (Wikipedia). It also bears other names such as autodiff, algorithmic differentiation, computational differentiation, and back-propagation.
- There are a lot of AD softwares
  1. TensorFlow and PyTorch: deep learning frameworks in Python
  2. Adept-2: combined array and automatic differentiation library in C++
  3. autograd: efficiently derivatives computation of NumPy code.
  4. ForwardDiff.jl, Zygote.jl: Julia differentiable programming packages
- This lecture: how to compute gradients using automatic differentiation (AD)
  - Forward mode, reverse mode, and AD for implicit solvers

Finite Differences

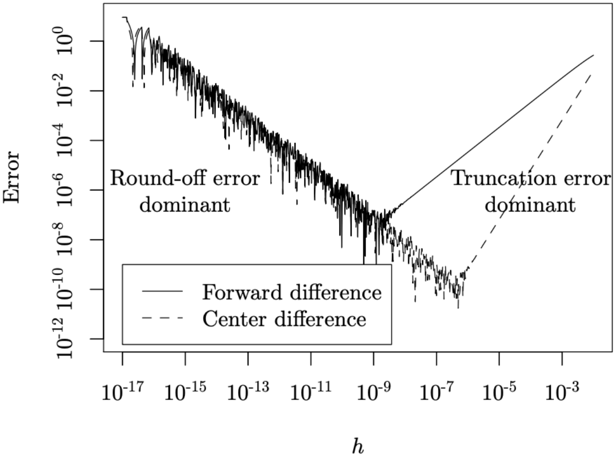
$$f'(x) \approx \frac{f(x+h)-f(x)}{h}, \quad f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$$

- Derived from the definition of derivatives

$$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$$

- Conceptually simple.
- Curse of dimensionalties: to compute the gradients of $f : \mathbb{R}^m \to \mathbb{R}$, you need at least $\mathcal{O}(m)$ function evaluations.
- Huge numerical error.

Finite Difference



Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. The Journal of Machine Learning Research, 18(1), 5595-5637.

- Symbolic differentiation computes exact derivatives (gradients): there is no approximation error.
- It can lead to complex and redundant expressions

```
>> sigmoid = @(x) 1/(1+exp(-x));
>> syms x w1 w2 w3 b1 b2 b3
>> diff(w3*sigmoid(w2*sigmoid(w1*x+b1)+b2)+b3, x)

ans =

(w1*w2*w3*exp(- b1 - w1*x)*exp(- b2 - w2/(exp(- b1 - w1*x) + 1)))/((exp(- b1
```

Figure: Symbolic differentiation for a neural network with two hidden layers in MATLAB.

- There may not exist convenient expressions for the analytical gradients of some functions.

Automatic Differentiation

- AD is neither finite difference nor symbolic differentiation.
- It evaluates numerically gradients of "function units" using symbolic differentiation, and chains the computed gradients using the chain rule

$$\frac{df(g(x))}{dx} = f'(g(x))g'(x)$$

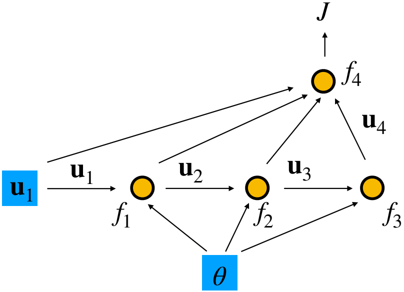
- It is efficient (linear in the cost of computing the function itself) and numerically stable.

Outline

Computational Graph

- The "language" for automatic differentiation is computational graph.
  - The computational graph is a directed acyclic graph (DAG).
  - Each edge represents the data: a scalar, a vector, a matrix, or a high dimensional tensor.
  - Each node is a function that consumes several incoming edges and outputs some values.

$$
\begin{aligned}
J &= f_4(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4), \\
\mathbf{u}_2 &= f_1(\mathbf{u}_1, \boldsymbol{\theta}), \\
\mathbf{u}_3 &= f_2(\mathbf{u}_2, \boldsymbol{\theta}), \\
\mathbf{u}_4 &= f_3(\mathbf{u}_3, \boldsymbol{\theta}).
\end{aligned}
$$

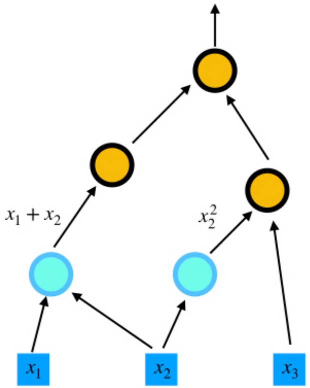


- Let's build a computational graph for computing

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$

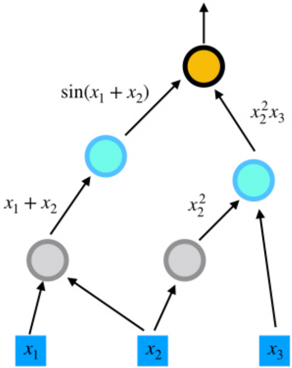
Building a Computational Graph

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$



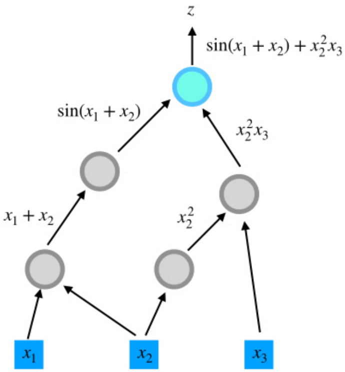
Building a Computational Graph

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$



Building a Computational Graph

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$

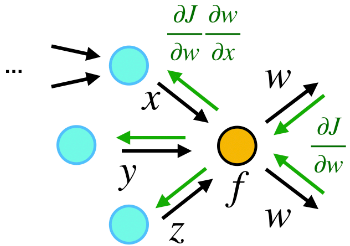


Outline

Reverse Mode AD

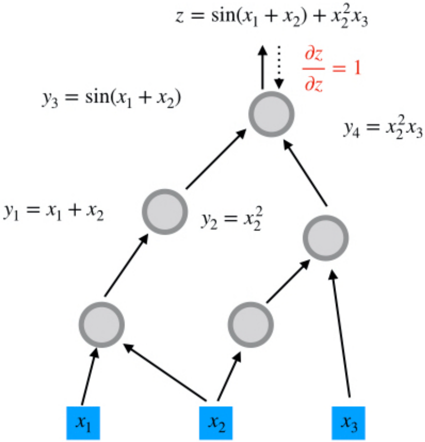
$$\frac{df(g(x))}{dx} = f'(g(x))g'(x)$$

- Computing in the reverse order of forward computation.
- Each node in the computational graph
  - Aggregate all the gradients from down-streams
  - Back-propagate the gradient to upstream nodes.



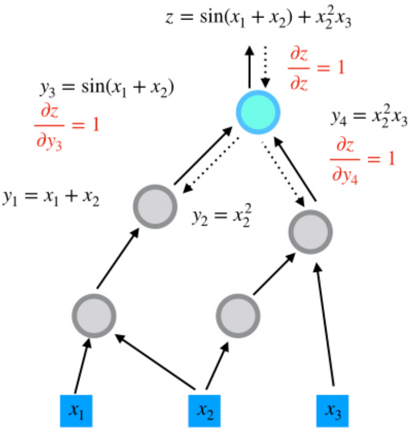
Example: Reverse Mode AD

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$



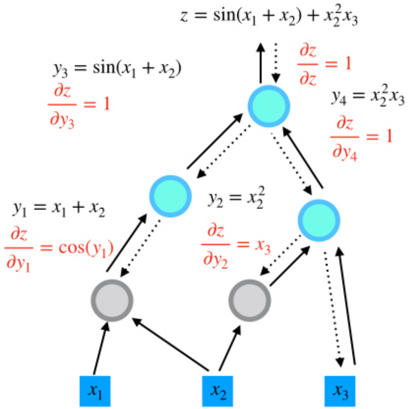
Example: Reverse Mode AD

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$



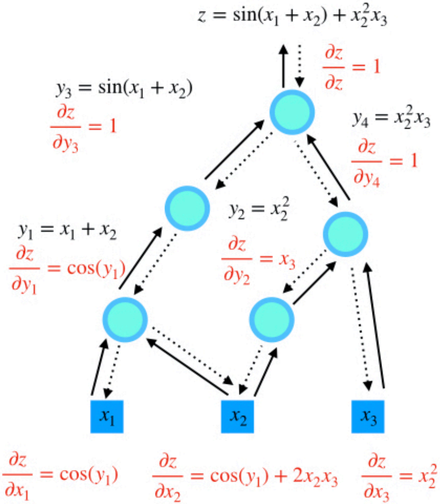
Example: Reverse Mode AD

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$



Example: Reverse Mode AD

$$z = \sin(x_1 + x_2) + x_2^2 x_3$$



Summary

- Reverse mode AD reuses gradients from down-streams. Therefore, this mode is useful for many-to-few mappings

$$f : \mathbb{R}^n \to \mathbb{R}^m, n \gg m$$

- Typical application:
  - Deep learning: $n =$ total number of weights and biases of the neural network, $m = 1$ (loss function).
  - Mathematical optimization: usually there are only a single objective function.

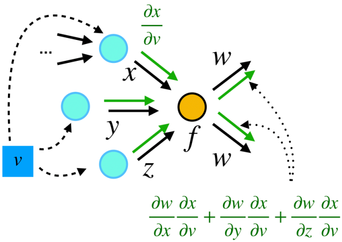
Outline

Automatic Differentiation: Forward Mode AD

- The forward-mode automatic differentiation also uses the chain rule to propagate the gradients.

$$\frac{\partial f \circ g(x)}{\partial x} = \frac{\partial f' \circ g(x)}{\partial g} \frac{\partial g'(x)}{\partial x}$$

- Compute in the same order as function evaluation.
- Each node in the computational graph
  - Aggregate all the gradients from up-streams.
  - Forward the gradient to down-stream nodes.



Example: Forward Mode AD

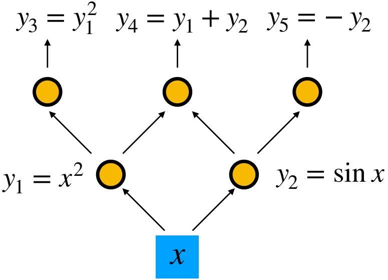
- Let's consider a specific way for computing

$$f(x) = \begin{bmatrix} x^4 \\ x^2 + \sin(x) \\ -\sin(x) \end{bmatrix}$$

Example: Forward Mode AD

- Let's consider a specific way for computing

$$f(x) = \begin{bmatrix} x^4 \\ x^2 + \sin(x) \\ -\sin(x) \end{bmatrix}$$

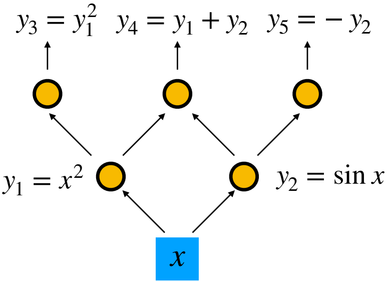


$$(y_1, y_1') = (x^2, 2x)$$
$$(y_2, y_2') = (\sin x, \cos x)$$

Example: Forward Mode AD

- Let's consider a specific way for computing

$$f(x) = \begin{bmatrix} x^4 \\ x^2 + \sin(x) \\ -\sin(x) \end{bmatrix}$$



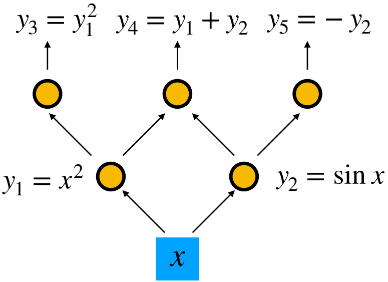
$$(y_1, y_1') = (x^2, 2x)$$
$$(y_2, y_2') = (\sin x, \cos x)$$

$$(y_3, y_3') = (y_1^2, 2y_1y_1') = (x^4, 4x^3)$$

Example: Forward Mode AD

- Let's consider a specific way for computing

$$f(x) = \begin{bmatrix} x^4 \\ x^2 + \sin(x) \\ -\sin(x) \end{bmatrix}$$



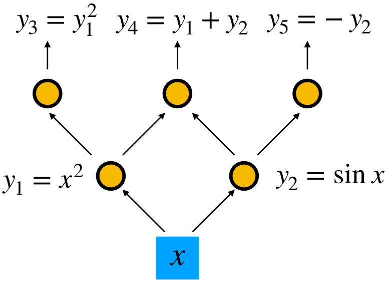
$$(y_1, y_1') = (x^2, 2x)$$
$$(y_2, y_2') = (\sin x, \cos x)$$

$$(y_3, y_3') = (y_1^2, 2y_1 y_1') = (x^4, 4x^3)$$
$$(y_4, y_4') = (y_1 + y_1, y_1' + y_2')$$
$$= (x^2 + \sin x, 2x + \cos x)$$

Example: Forward Mode AD

- Let's consider a specific way for computing

$$f(x) = \begin{bmatrix} x^4 \\ x^2 + \sin(x) \\ -\sin(x) \end{bmatrix}$$



$$(y_1, y_1') = (x^2, 2x)$$
$$(y_2, y_2') = (\sin x, \cos x)$$

$$(y_3, y_3') = (y_1^2, 2y_1 y_1') = (x^4, 4x^3)$$
$$(y_4, y_4') = (y_1 + y_1, y_1' + y_2')$$
$$= (x^2 + \sin x, 2x + \cos x)$$
$$(y_5, y_5') = (-y_2, -y_2') = (-\sin x, -\cos x)$$

Summary

- Reverse mode AD reuses gradients from upstreams. Therefore, this mode is useful for few-to-many mappings

$$f : \mathbb{R}^n \to \mathbb{R}^m, n \ll m$$

- Applications: sensitivity analysis, uncertainty quantification, etc.
  - Consider a physical model $f : \mathbb{R}^n \to \mathbb{R}^m$, let $x \in \mathbb{R}^n$ be the quantity of interest (usually a low dimensional physical parameter), uncertainty propagation method computes the perturbation of the model output (usually a large dimensional quantity, i.e., $m \gg 1$)

$$f(x + \Delta x) \approx f'(x)\Delta x$$

Summary

- In general, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$

Mode	Suitable for ...	Complexity[1]	Application
Forward	$m \gg n$	$\leq 2.5\ \mathrm{OPS}(f(x))$	UQ
Reverse	$m \ll n$	$\leq 4\ \mathrm{OPS}(f(x))$	Inverse Modeling

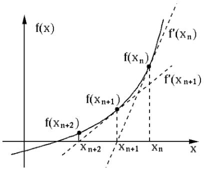
- There are also many other interesting topics
  - Mixed mode AD: many-to-many mappings.
  - Computing sparse Jacobian matrices using AD by exploiting sparse structures.

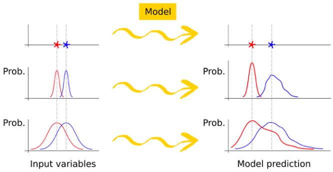
Margossian CC. A review of automatic differentiation and its efficient implementation. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2019 Jul;9(4):e1305.

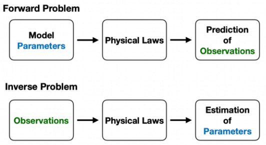
[1] OPS is a metric for complexity in terms of fused-multiply adds.

Outline

The Demand for Gradients in Physical Simulation







- Solving nonlinear equations
- Uncertainty quantification/sensitivity analysis
- Inverse problems

Image source:
https://mirams.wordpress.com/2016/11/23/uncertainty-in-risk-prediction/,
http://fourier.eng.hmc.edu/e176/lectures/ch2/node5.html

Inverse Problem and Mathematical Optimization

- Consider a bar under heating with a source term $f(x,t)$. The right hand side has fixed temperature and the left hand side is insulated.
- The governing equation for the temperature $u(x,t)$ is

$$\begin{aligned}
\frac{\partial u(x,t)}{\partial t} &= \kappa \Delta u(x,t) + f(x,t), \quad t \in (0,T), x \in \Omega \\
u(1,t) &= 0 \\
\kappa \frac{\partial u(x,t)}{\partial x} &= 0
\end{aligned}$$

- The diffusivity coefficient is given by

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

where a and b are design parameters.

Inverse Problem and Mathematical Optimization

- How can we design the material of the bar so that $u(0,t)$ has a desired value $u_0(t)$?
- This problem is a standard inverse problem. We can formulate the problem as a PDE-constrained optimization problem

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

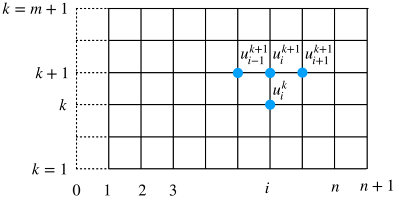
Numerical Partial Differential Equation

- As with many physical modeling techniques, we discretize the PDE using numerical schemes. Here is a finite difference scheme for the PDE $k = 1, 2, \dots, m, i = 1, 2, \dots, n$

$$\frac{u_i^{k+1} - u_i^k}{\Delta t} = \kappa_i \frac{u_{i+1}^{k+1} + u_{i-1}^{k+1} - 2u_i^{k+1}}{\Delta x^2} + f_i^{k+1}$$

For initial and boundary conditions, we have

$$-\kappa_1 \frac{u_2^k - u_0^k}{2\Delta x} = 0$$
$$u_{n+1}^k = 0$$
$$u_i^0 = 0$$



Numerical Partial Differential Equation

- Rewriting the equation as a linear system, we have

$$AU^{k+1} = U^k + F^{k+1}, \quad U^k = \begin{bmatrix} u_1^k \\ u_2^k \\ \vdots \\ u_n^k \end{bmatrix}$$

Here $\lambda_i = -\kappa_i \frac{\Delta t}{\Delta x^2}$ and

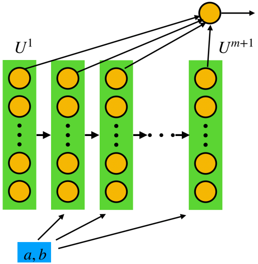
$$A = \begin{bmatrix} -2\lambda_1 - 1 & 2\lambda_1 & & & & \\ \lambda_2 & -2\lambda_2 - 1 & \lambda_2 & & & \\ & \lambda_3 & -2\lambda_3 - 1 & \lambda_3 & & \\ & & \ddots & & & \\ & & & \ddots & \lambda_{n-1} \\ & & & \lambda_n & -2\lambda_n - 1 \end{bmatrix}, \quad F^k = \Delta t \begin{bmatrix} f_1^{k+1} \\ f_2^{k+1} \\ \vdots \\ f_n^{k+1} \end{bmatrix}$$

Computational Graph for Numerical Schemes

- The discretized optimization problem is

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

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



Implementation using an AD system

```
function condition(i, u_arr)
    i<=m+1
end

function body(i, u_arr)
    u = read(u_arr, i-1)
    rhs = u + F[i]
    u_next = A\rhs
    u_arr = write(u_arr, i, u_next)
    i+1, u_arr
end

F = constant(F)
u_arr = TensorArray(m+1)
u_arr = write(u_arr, 1, zeros(n))
i = constant(2, dtype=Int32)
_, u = while_loop(condition, body, [i, u_arr])
u = set_shape(stack(u), (m+1, n))
```

Simulation Loop

You will have chance to Practice in your homework! (TensorFlow/PyTorch, ADCME, or any other AD tools)

```
uc = readdlm("data.txt")[:]
```

```
loss = sum((uc-u[:,1])^2) * 1e10
```

Formulate Loss Function

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

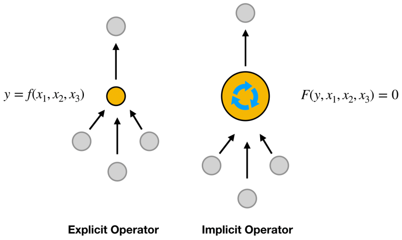
Gradient Computation

Optimization

Outline

Challenges in AD

- 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**	$Ax = y$
Nonlinear	**Implicit**	$F(x, y) = 0$

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 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

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

The above gradient is exact.

Implicit Operators in Physical Modeling

- Return to our bar problem, what if the material property is complex and has a temperature-dependent governing equation?

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

- An implicit scheme is usually a nonlinear equation, and requires an iterative solver (e.g., the Newton-Raphson algorithm) to solve

$$\frac{u_i^{k+1} - u_i^k}{\Delta t} = \kappa(u_i^{k+1}) \frac{u_{i+1}^{k+1} + u_{i-1}^{k+1} - 2u_i^{k+1}}{\Delta x^2} + f_i^{k+1}$$

- Typical AD frameworks cannot handle this operator. We need to differentiate through implicit operators.
- This topic will be covered in a future lecture: physics constrained learning.

Outline

Conclusion

- What's covered in this lecture
  - Reverse mode automatic differentiation;
  - Forward mode automatic differentiation;
  - Using AD to solver inverse problems in physical modeling;
  - Automatic differentiation through implicit operators.

What's Next

- Physics constrained learning: inverse modeling using automatic differentiation through implicit operators;
- Neural networks and numerical schemes: substitute the unknown component in a physical system with a neural network and learn the neural network with AD;
- Implementation of inverse modeling algorithms in ADCME.