Lecture 2 Wed 1/21

Overview

- HW1 out
- Example FDM code + PINN code
- Explaining DNN vs poly.

Finite differences (0)

Consider $\Omega = [0, 1]$

$$\partial_t u = \alpha \, \partial_{xx} u$$

$$u(0) = u(1) \quad \text{(periodic BC)}$$

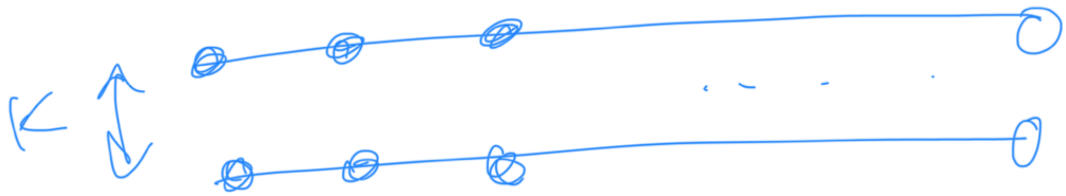
Discretize on uniform grid $X_h \subseteq \Omega$

$$h = 1/N, \quad X_h = \{ih\}_{i=0}^{N-1}$$

Def grid function

$u_i^n = u(x_i, t_n)$, $t_n = kn$

$\mathcal{U} = \{u_i^n\}_{i,n}$

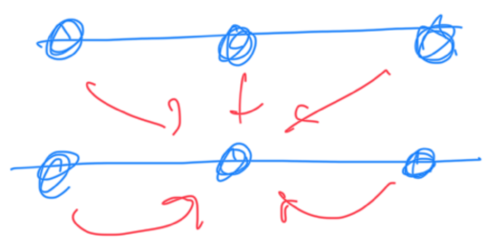


<u>IDEA</u>

Build up a polynomial approx w/ Taylor series, approx. PDE

Def <u>stencil</u>

$\mathcal{S}(x_i^n) = \{y \in \mathcal{U}, y \sim x_i^n\}$



Want to approx a derivative

$$Du_i^n = \sum_{u_j \in \mathcal{S}(x_i^n)} \alpha_j u_j$$

such that $$Dp_i^n = \sum \alpha_j p_j \quad \forall p \in \mathcal{P}^n(\Omega \times [0,T])$$

n^{th} order polynomials

Examples

1. $D = \partial_t$

$$u(t+k,x) = u(t,x) + k\,\partial_t u + O(k^2)$$

$$\frac{u(t+k,x) - u(t,x)}{k} = \partial_t u + O(k)$$

$$\Rightarrow \quad Du_i^n = \frac{u_i^{n+1} - u_i^n}{k}$$

Verify $u = A + Bx + Ct$

$$Du_i^n = [A + Bx_i + C(t_i + k) - (A + Bx_i + Ct_i)]/k$$

$$= 1 \quad \checkmark$$

(2) $\partial_{xx} u_j^n = \dfrac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}$

First finite diff scheme

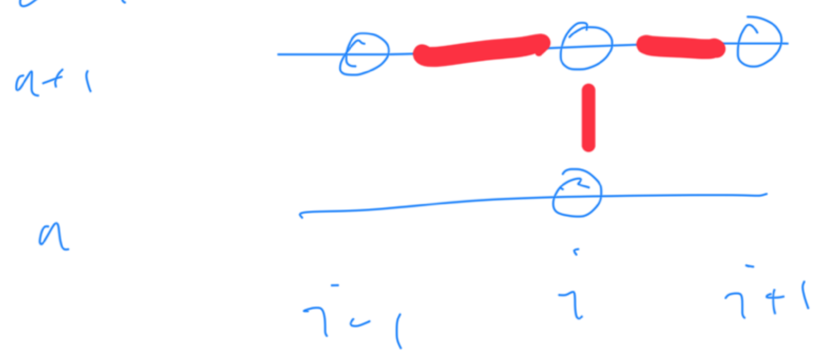
Implicit Euler

$$\partial_t u = \alpha \partial_{xx} u$$

$$u_i^{n+1} - u_i^n \quad (\;{}^{n+1} \quad {}^{n+1} \quad {}^{n+1})$$

$$\frac{u_i^{n+1} - u_i^{n}}{k} = \alpha \frac{(u_{i+1} - 2u_i + u_{i-1})}{h^2}$$

$$:= \quad D_t u_i = D_{xx}$$



Exercise

$$\lim_{k,h \downarrow 0} \left| D_t u - D_{xx} - (\partial_t u - \partial_{xx} u) \right| = 0$$

Solving this on a computer

$$\underbrace{(I - \alpha k D_{xx})}_{A} u^{n+1} = u^n$$

$$A \quad u^{n+1} = u^n$$

Solve for

Linear ... next step

Let $\lambda = \frac{\alpha k}{h^2}$

$$A = \begin{bmatrix} \ddots & \ddots & & & & -\lambda \\ \ddots & \ddots & \ddots & & & \\ & -\lambda & (1+2\lambda) & -\lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \ddots \\ -\lambda & & & & \ddots & \ddots \end{bmatrix}$$

periodic BCs

Code gives example of this in PyTorch. HW1 due in 1 week will have you running this code

Building Toward

Learning physics

$$\partial_t u = N[u; \Theta]$$

time stencil

trainable stencil

Polynomials vs Neural Networks

As you go out in the world, many will argue whether a neural network can solve PDE's "better" than a standard method.

Both polynomials & DNN's have a universal approx theorem

Thm Universal Approx.

Let $f \in [a,b] \to \mathbb{R}$ be a cont. func. and $\varepsilon > 0$

Poly There exists a polynomial $p(x)$ such that

$$\max_{x \in [a,b]} |f(x) - p(x)| < \varepsilon$$

NNs There is a one-layer ReLU network $f_{NN}(x)$

$$\max_{x \in [a,b]} |f(x) - f_{NN}(x)| < \varepsilon$$

Polynomials

Thm Existence of interp. poly.

Given N pts $(x_i, y_i)_{i=0}^{N-1}$

there is a unique polynomial of degree at most $N-1$ satisfying $p(x_i) = y_i$

PF $p(x) = c_0 + c_1 x + \dots + c_{n-1} x^{n-1}$

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & \vdots & & & \\ & \vdots & & & \end{bmatrix}}_{V} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} y_i \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

called a vandermonde matrix

- poly. exists if V is invertible

$\Leftarrow$) $\det(V) \neq 0$

- Assume $\det(V) = \prod_{\substack{i,j \\ i \neq j}} (x_j - x_i)$

Details in notes. Need induction + cofactors to show

- Invertible for $x_j \neq x_i$

$\blacksquare$

This means that on the grid

$$\|p - f\|_\infty = \|p - f\|_2 = 0$$

To interpolate between grid pts we can express p w/ Lagrange interpolant

$$L_j(x) = \prod_{k=0}^{n-1} \frac{x - x_k}{x_j - x_k}$$

$k \neq j$

ex $\quad \dfrac{x - x_0}{x_1 - x_0}, \quad \dfrac{x - x_1}{x_0 - x_1}$

Lemma

we have the property

$L_j(x_i) = \delta_{ij}$

Implying $p(x) = \sum_j p(x_j) L_j(x)$

is an explicit formula for interpolant

Pf 1. If $i = j$

num = den

If $i \neq j$

get zero

2. From vandermonde

we know we interpolate polynomials

$$\sum_j p(x_j)\, L_j(x_i)$$

$$= \sum_j p(x_j)\, \delta_{ij}$$

$$= p(x_i)$$

Error Analysis Pf

Expand

$$f(x_j) = f(x) + f'(x)(x_j - x) + \dots$$

Plug into Lagrange

$$p(x) = \sum_j L_j(x)\left[f(x) + f'(x)(x_j - x) \ldots\right]$$

From $\sum L_j(x) = 1$

$$\sum L_j(x)\, f(x) = f(x)$$

For higher moments

$$\sum L_j(x)\, f'(x)\,(x_j - x)$$

$$= f'(x)\,(x - x) = 0$$

So we're left with remainder in Taylor series

$$|f - p|(x) = \left| \sum_j \frac{f^{(n+1)}(\xi)}{(n+1)!} (x_j - x)^{n+1} L_j(x) \right|$$

$$\leq \sum C\left(f^{(n+1)}\right) |x_j - x|^{n+1} \, |L_j(x)|$$

Two key scenarios

Are we on a

① Fixed domain $[0,1]$

→ $|x_j - x| < 1 - \frac{1}{N}$

→ Need $|L_j| < C$

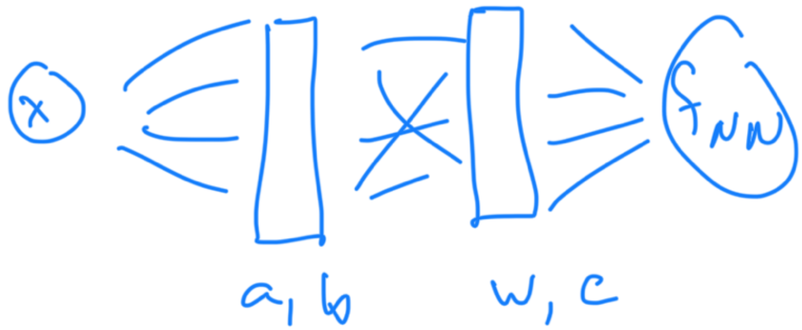
→ Take $m \to \infty$

② Localized domain $[0,h]$ like a finite difference stencil so we can pick h small (i.e refine stencil)

$$\boxed{\text{On to neural Networks!}}$$

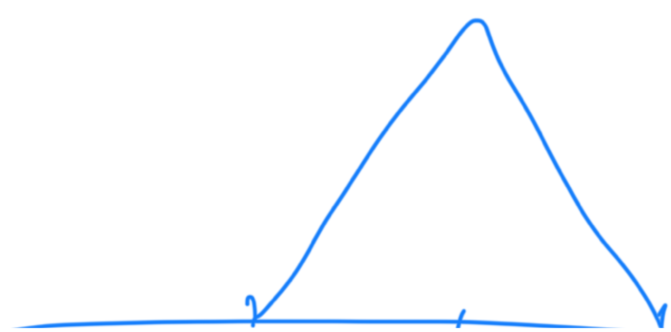
ReLU activation $\sigma(x) = \max(0,x)$

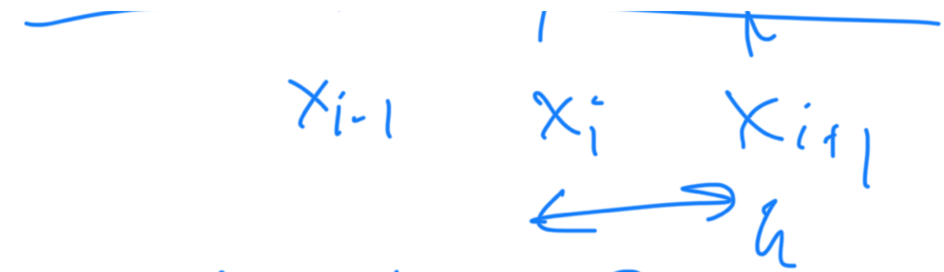
$$f_{NN}(x) = \sum_j w_j \, \sigma(a_j x + b_j) + c$$



We can construct a linear interpolant through careful choices of a, b

$$\phi_i(x) = \frac{1}{h}\Big[\sigma(h(x-x_{i-1})) - 2\sigma(h(x-x_i)) + \sigma(h(x-x_{i+1}))\Big]$$





Note $\phi_i(x_j) = \delta_{ij}$

We can choose linear layer so

$$f_{NN}(x) = \sum_i f_i \, \phi_i(x)$$

Note this interpolates

$$f_{NN}(x_j) = \sum_i f_i \, \phi_i(x_j)$$

$$= f_j \quad ✓$$

and $\sum \phi_i = 1$

E

$$= |f - f_{NN}|(x) = \left| f(x) - \sum_i f_i \, \phi_i(x) \right|$$

mult by r $= \left| \sum_i \phi_i f(x) - \phi_i f_i \right|$

$$\leq \sum_i \phi_i \left| f(x) - f_i \right|$$

On each little piecewise linear subdomain

$$\left| f(x) - f_i \right| \leq C h^2$$

$$E \leq C \sum_i \phi_i h^2$$

$$= C h^2$$

So if we take enough neurons $h \to 0$

Punchline

- Polynomials can get tricky, but easily converge on a refinable domain like a finite diff stencil

- MLPs can "refine themselves" by picking neurons that localize the basis.

- Zero guarantees that

you get those localized

bases when searching

w/ GD.