Welcome to 18.335J / 6.337J!

Course info/materials: github.com/mitmath/18.335J

→ Psets posted on Canvas
→ Discussion forum on Piazza

18.335J is an advanced intro to NLA + select topics

- Direct methods
- Iterative methods
- Randomized NLA
- Optimization, nonlinear problems, NLA in inf. dim. spaces, ...

Numerical methods (& analysis)

"Numerical analysis is the study of algorithms for the problems of continuous mathematics." - L.N. Trefethen

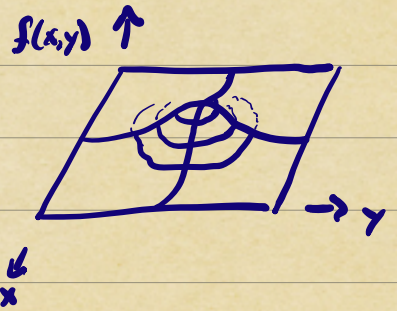
At the heart of those algorithms:

Numerical Linear Algebra (NLA)

Example 1: Numerical methods for PDEs

a) Poisson's equation $\Delta u = f$ $\quad u|_{\partial\Omega} = g$

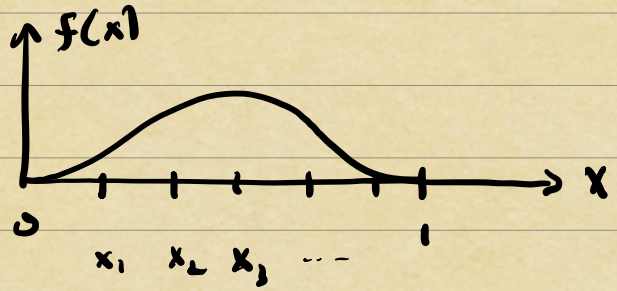
"source"



How do we solve on a computer?

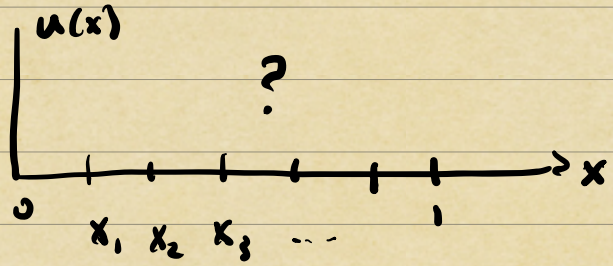
Illustrate in 1D

$$\frac{d^2}{dx^2} u(x) = f(x)$$



- Discretize w/ finite differences

$f_k = f(x_k)$, $u_k = u(x_k)$, $h = x_{k+1} - x_k$



$$u''(x_k) \approx \frac{u_{k+1} - 2u_k + u_{k-1}}{h^2}$$

$$\frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & & 1 & -2 \end{bmatrix}\begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix}$$

- Solve by LU factorization (Gaussian Elim.)

b) Schrodinger Equation

Quantized energies = eigenvalues

PDE Eigenvalue Problem

$$-\Delta u + Vu = Eu$$

- Discretize by finite differences

$$\frac{1}{h^2}\begin{bmatrix} 2+V_1h^2 & -1 & & \\ -1 & 2+V_2h^2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2+V_Nh^2 \end{bmatrix}\begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} = E\begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}$$

- Solve by QR Algorithm

Standard workflow: "discretize-then-solve"

$\Delta u = f$ s.t. $u|_{\partial\Omega} = g$

"discretize" ⇓ → see 18.336J Fast methods for PDE; IE

$Ax = b$

"solve" ⇓

18.335J, e.g. NLA

? → see 18.330 Intro to numerical analysis

$x \approx u$

Remarks:

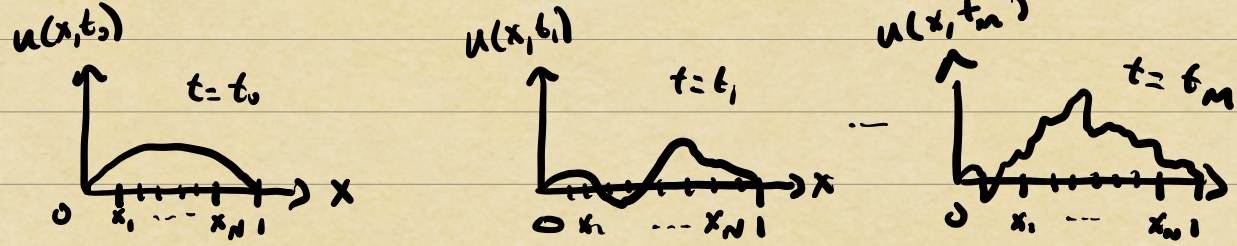
- In 2 and 3D, A can be HUGE!
- Standard factorization techniques become too expensive
- but A arising from PDE disc. (and many other settings) often has <u>structure</u> (e.g. tridiagonal, block tridiagonal, etc.) that allows us to compute Ax or $A^{-1}b$, or $A = X \Lambda X^{-1}$ very efficiently
- A central theme in modern NLA is finding and exploiting <u>structure</u> to design fast algorithms

Example 2: Numerical models from data

What if we don't know the PDE for a complex system, but we have lots of experimental data? Can we "learn" a model?

1D Illustration

Data (= "Snapshots")



Want to learn linear model that maps

$$A: u(x, t_k) \to u(x, t_{k+1})$$

Data matrix $(X)_{i,j} = u(x_i, t_j)$ ← Experimental data

$$X = \begin{bmatrix} | & & | \\ u(x,t_1) & \cdots & u(x,t_{m-1}) \\ | & & | \end{bmatrix}$$

$$X' = \begin{bmatrix} | & & | \\ u(x,t_2) & \cdots & u(x,t_m) \\ | & & | \end{bmatrix}$$

$\Rightarrow$ Find A s.t. $X' = AX$

(May have infinitely many, one, or no solutions)

Idea: minimize ↑ "size" of $X' - AX$ over $N \times N$ matrices A. (Usually $N \gg M$.)

Measure "size" of $X' - AX$ with a matrix norm, e.g., the Frobenius norm (generalizes Euclidean dist.)

$$\|M\|_F = \sqrt{\sum_{i,j} |M_{ij}|^2}$$

$\Rightarrow$ Least-squares problem. Solve by QR factorization or SVD decomp.

NLA is the workhorse under most numerical methods, but it works best in tandem with other fields of math.

Example 3: Nonlinear problems

For interacting quantum systems (e.g. Quantum Chem.)

$$-\Delta u + V(u)\,u = E u$$

↑ potential depends on atomic orbitals!

Iterative solution: linearization + NLA

Choose $u = u_0$ (initial guess)

Solve $-\Delta u_1 + V(u_0)\,u_1 = E u_1$ as above,

Solve $-\Delta u_2 + V(u_1)\,u_2 = E u_2$

(repeat) ⋮

Convergence?

Remarks

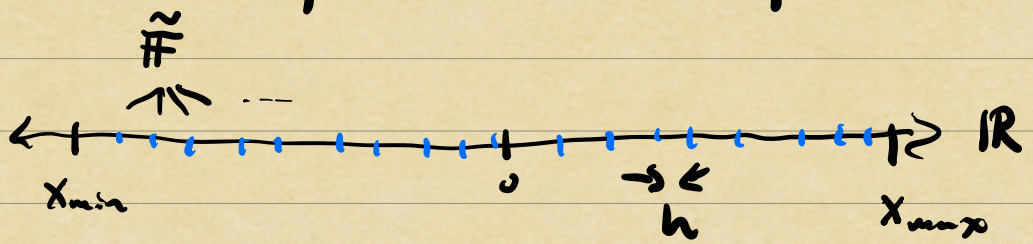
=> Using Calculus tools to "linearize" and then applying NLA is a powerful paradigm for nonlinear problems.

=> It is a key ingredient in many optimization algorithms.

New Topic / # Floating Pt. Arithmetic

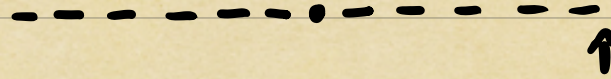
Q: Computers have finite memory, so how do we work with real #s?

Naive idea: store a fixed # of digits
=> "Fixed point" or "fixed precision"



Approximates continuum by equally spaced points (similar to our function approx. scheme earlier!).

Mathematically, (fixed max # of digits; fixed 5 digits of precision)



$x \in \mathbb{R}$ (really between $[x_{min}, x_{max}]$), $x \in \tilde{\mathbb{F}}$ s.t. $|x - x'| \leq h$

Problem: lose <u>significant digits</u> rapidly, if we are working with a range of big and small #s.

Significant digits: $\underline{1.2340000} \times 10^{-5}$

⇒ Nicely expressed w/ scientific notation

Floating point mimics scientific notation to efficiently represent a huge array of real #'s!

$$\mathbb{F} = \{0, \pm (m/\beta^t)\beta^e\} \quad \text{w/} \quad \beta^{t-1} \le m \le \beta^t - 1$$

integer, and integer exponent.

(In practice, e is limited by storage, i.e. overflow & underflow.)

$$\text{sign}\ \boxed{\pm}\ 1.\underbrace{\_\_\_\_\_}_{\text{Significand}} \times \beta^{e}$$

↖ base

↙ exponent

Significand: $p-1$ digits, $0 \le \_ < \beta$

p is the precision: # of sig. figs.

We can construct algorithms by approx. $\mathbb{R}$ by $\mathbb{F}$, replacing arithmetic on $\mathbb{R}$ by arithmetic over $\mathbb{F}$. To understand the errors in this approx, we need to get a few facts about $\mathbb{F}$.