

# Welcome to 18.335J / 6.337J!

Course info/materials: [github.com/mitmath/18.335J](https://github.com/mitmath/18.335J)

- Psets posted on Canvas
- Discussion forums 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

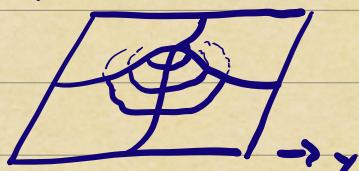
a) Poisson's equation

$$\Delta u = f$$

$$u|_{\partial\Omega} = g$$

"source"

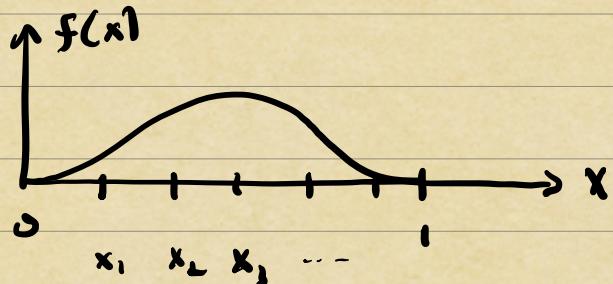
$$f(x,y)$$



How do we solve on a computer?

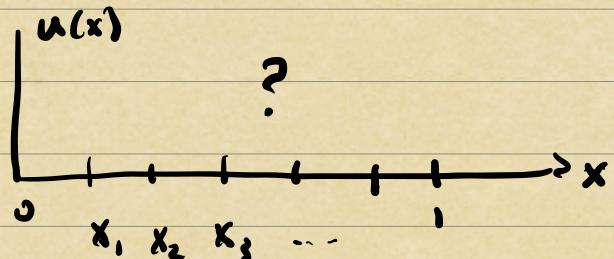
Illustrate in 1D

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



- Discretize w/ finite differences

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



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

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

- Solve by LU factorization (Gaussian Elim.)

### b) Schrödinger Equation

Quantized  
energies =  
eigenvalues

PDE

$$-\Delta u + V u = E u$$

Eigenvalue

Problem

- Discretize by finite differences

$$\frac{1}{h^2} \begin{bmatrix} 2 + V_1 h^2 & -1 & & \\ -1 & 2 + V_2 h^2 & & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 + V_N h^2 \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} = E \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}$$

- Solve by QR Algorithm

Standard workflow: "discretize-then-solve"

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

"discretize"  $\Downarrow$  → see 18.336J Fast methods  
for PDE; IE

$$A x = b$$

"solve"



18.335J,  
e.g. NLA

? → see 18.330 Intro to numerical  
analysis

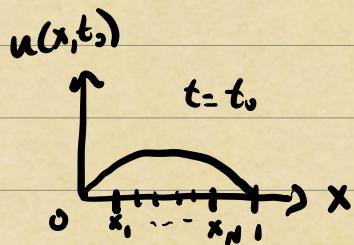
## Remarks:

- 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 structure (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 structure to design fast algorithms

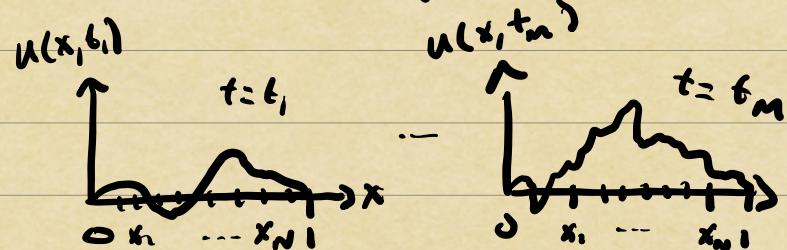
## Example 2: Numerical models from data

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

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

$$X = \begin{bmatrix} | & | \\ u(x_1, t_1) & \cdots & u(x_1, t_{m-1}) \\ | & | \end{bmatrix}$$

Find  $A$  s.t.

$$\Rightarrow X' = AX$$

$$X' = \begin{bmatrix} | & | \\ u(x_2, t_1) & \cdots & u(x_2, t_m) \\ | & | \end{bmatrix}$$

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

"size" of

Idea: minimize  $\|X' - AX\|$  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)

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

$\Rightarrow$  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.)

$$-\Delta u + V(u) u = Eu$$

↑ potential depends on atomic orbitals!

Iterative solution: linearization + NLA

Choose  $u = u_0$  (initial guess)

Solve  $-\Delta u_i + V(u_0) u_i = Eu_i$ , as above,

solve  $-\Delta u_2 + V(u_1) u_2 = Eu_2$

(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

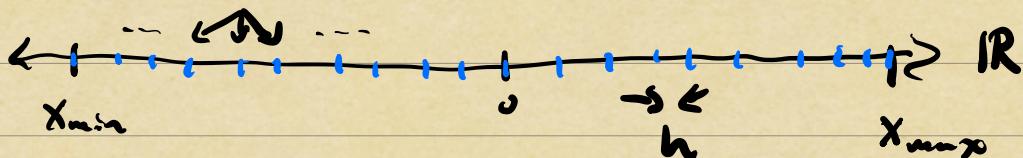
## Floating Pt. Numbers

To do linear algebra, we need to do arithmetic (+ - × ÷) on  $\mathbb{R}$  - the "real #'s" - or  $\mathbb{C}$  (complex #'s). We'll focus on  $\mathbb{R}$ , but not much changes for  $\mathbb{C}$ .

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

[Intuitively, The challenge is that the real #'s are unbounded and form a continuum.]

Idea 1: "Discretize" the continuum into equally spaced points



Mathematically,  $F_h = \{x_{min} + (integer) \times h \leq x_{max}\}$

On computer, store  $\frac{\pm}{\uparrow} \text{-----} \times h$   
each bit stores  
a digit of the integer

Example:  $h = 10^{-2}$ , 7 digit integers

$\pm \underline{1} \underline{2} \underline{0} \underline{0} \underline{4} . \underline{7} \underline{2}$

↑  $h$  controls position  
of decimal pt.

"Fixed point" arithmetic

⇒ Round result of  $\{+, -, \times, \div\}$  to nearest

$x \in F_h$

$$.02 \times .01 = .0002$$

$$.02 \boxtimes .01 = F_h(.0002) = 0$$

Lost all significant digits!

Idea 2: "Floating Point"

bits for exponent

$\frac{\pm}{\uparrow} \text{-----} \text{-----} \times h$

Sign each bit stores

a digit of the  
Significand (also, "mantissa",  
"fraction")

exponent

mimics scientific notation:  $\underline{1.263} \times 10^{-5}$

Significant  
digits

Mult.  
w/sci.not.

$$\begin{aligned} .02 \times .01 &= (2 \times 10^{-2}) \times (1 \times 10^{-2}) \\ &= 2 \times 10^{-4} \leftarrow \text{update exp.} \\ &\quad \leftarrow \text{update integer.} \end{aligned}$$

Decimal location changes or "floats" depending on size of  $\alpha$ . This frees up all bits of the mantissa to preserve sig. digits during multiplication/division.

Floating point #'s ( $F$ ) have the form

$$x = \pm \underbrace{d_0.d_1 \dots d_{p-1}}_{\substack{\text{Significand} \\ p-\text{precision} \\ (\# \text{ digits})}} \times \beta^e \quad 0 \leq d_x < \beta$$

In practice, finite storage for exponent means  
 $e_{\min} \leq e \leq e_{\max}$  (overflow  
underflow)

(There is also  $\mathcal{O}$ ,  $\pm \infty$ , NaN.)

On most computers IEEE single : double prec.  
 $\beta = 2$  (binary)

single (32 bits)

double (64)

Sign 1

1

exp 8

11

mantissa 23

52

Floating point #'s are not equally spaced!

The spacing changes relative to size of #

key property: For all  $x \in \mathbb{R}$ , there is

a  $x' \in \mathbb{F}$  such that  $|x' - x| \leq \frac{1}{2} \epsilon_{\text{mach}} |x|$

where  $\epsilon_{\text{mach}} = \beta^{1-p}$

In Double prec.  $\epsilon_{\text{mach}} = 2^{-53} \approx 1.11 \times 10^{-16}$

Single prec.  $\epsilon_{\text{mach}} = 2^{-24} \approx 5.96 \times 10^{-8}$

### Floating Pt. Arithmetic

#### Real Arithmetic

$$+, -, \times, \div : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

#### F.P. Arithmetic

$$\oplus, \ominus, \otimes, \div : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$$

$f| : \mathbb{R} \rightarrow \mathbb{F}$ ,  $f|(x)$  rounds  $x$  to closest  $x' \in \mathbb{F}$ .

Exact rounding:  $x \odot y = f|(x \cdot y)$

for any  $x, y \in \mathbb{F}$  and  $\odot = +, -, \times, \div$

This means that for any  $x, y \in F$ , we have

$$x \odot y = (x \cdot y)(1 + \varepsilon)$$

where  $|\varepsilon| \leq \epsilon_{\text{mach}}$ .

## "Fundamental Axiom of Floating Point Arithmetic"

\* This ignores over/underflow, when numbers are too big/small for  $c_{\max}, c_{\min}$  to capture.

Important points: (see iJulia notebook)

- Decimal input / output means rounding on inputs & outputs!

→ • F-P-A is not associative

$$(x \odot y) \oplus z \neq x \oplus (y \odot z)$$

- Over/Underflow, Inf, NaN

- • Catastrophic cancellation

- Accumulation of Rounding Errors

Example: Sum  $n$  numbers  $\underbrace{x_1, \dots, x_n}_{\underline{x}} \in \mathbb{F}$

$$f(\underline{x}) = \sum_{i=1}^n x_i$$

Pseudocode:

$$s = 0$$

for  $i = 1, \dots, n$

$$s = s + x_i$$

end

floating point

$$\tilde{s}_i = x_i$$

$$\tilde{s}_i = \tilde{s}_{i-1} \oplus x_i$$

$$\tilde{f}(\underline{x}) = \tilde{s}_n$$

Error:  $|f(\underline{x}) - \tilde{f}(\underline{x})| = ?$  We'll look for an upper bound

useful notation

$f(\varepsilon^2) = O(\varepsilon^2)$  if here is  
per.  
const  
C and  
 $\varepsilon_0$ , s.t.  $f(\varepsilon) \leq C \varepsilon^2$  for all  $\varepsilon < \varepsilon_0$

Claim:  $|\tilde{f}(\underline{x}) - f(\underline{x})| \leq n \varepsilon_{\text{mach}} \sum_{i=1}^n |x_i| + O(\varepsilon_{\text{mach}})$

1) First, show that  $\tilde{f}(\underline{x}) = \sum_{i=1}^n x_i \prod_{k=1}^n (1 + \varepsilon_k)$

with  $|\varepsilon_k| \leq \varepsilon_{\text{mach}}$ .

Inductive proof:  $n=1$   $\tilde{f}(\underline{x}) = x_1$

$$\underline{n \geq 1} \quad \text{Suppose } \tilde{s}_{n-1} = \sum_{i=1}^{n-1} x_i \prod_{k=i}^{n-1} (1+\varepsilon_k)$$

$$\tilde{s}_n = \tilde{s}_{n-1} + x_n = (\tilde{s}_{n-1} + x_n)(1+\varepsilon_n)$$

$$= \left[ \sum_{i=1}^{n-1} x_i \prod_{k=i}^{n-1} (1+\varepsilon_k) \right] (1+\varepsilon_n) \quad |\varepsilon_n| < \varepsilon_{\text{mach}}$$

$$= \sum_{i=1}^n x_i \prod_{k=i}^n (1+\varepsilon_k) \quad + x_n (1+\varepsilon_n) \quad \checkmark$$

$$2) \text{ So } \tilde{f}(x) = \sum_{i=1}^n x_i (1+\delta_i) \quad 1+\delta_i = \prod_{k=i}^n (1+\varepsilon_k)$$

$$= 1 + \sum_{k=i}^n \varepsilon_k + O(\varepsilon_{\text{mach}}^2)$$

$$|\delta_i| = \left| \sum_{k=i}^n \varepsilon_k + O(\varepsilon_{\text{mach}}^2) \right| \leq \sum_{k=i}^n |\varepsilon_k| + O(\varepsilon_{\text{mach}}^2)$$

$$3) \text{ Putting it together} \quad |\tilde{f}(x) - f(x)| = \left| \sum_{i=1}^n x_i \delta_i \right| \leq \sum_{i=1}^n |x_i| |\delta_i| \leq n \varepsilon_{\text{mach}} \sum_{i=1}^n |x_i| + O(\varepsilon_{\text{mach}}^2)$$

$$\leq n \varepsilon_{\text{mach}} \sum_{i=1}^n |x_i|$$

$$+ O(\varepsilon_{\text{mach}}^2) \quad \checkmark$$