

Welcome to 18.335J / 6.337J!

Course info/materials: 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

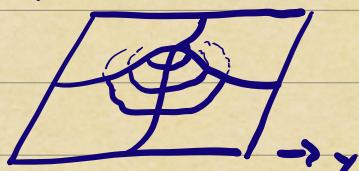
a) Poisson's equation

$$\Delta u = f$$

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

"source"

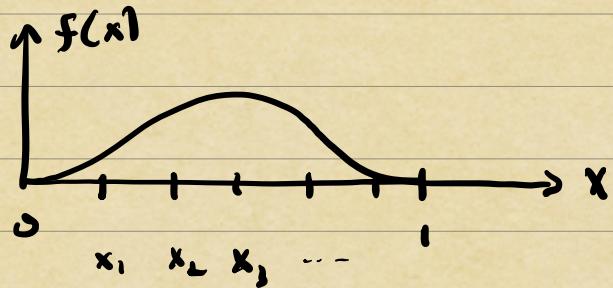
$$f(x,y)$$



How do we solve on a computer?

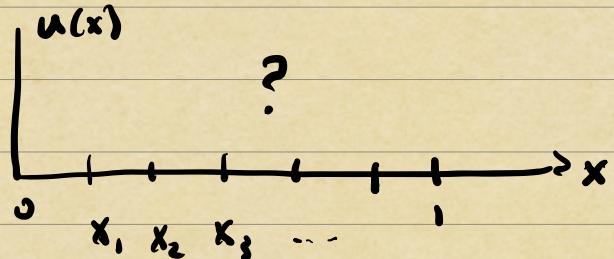
Illustrate in 1D

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



- 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

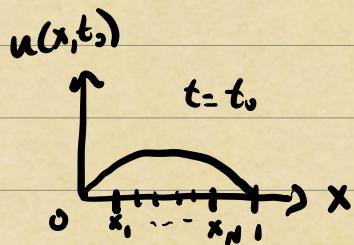
Remarks:

- 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

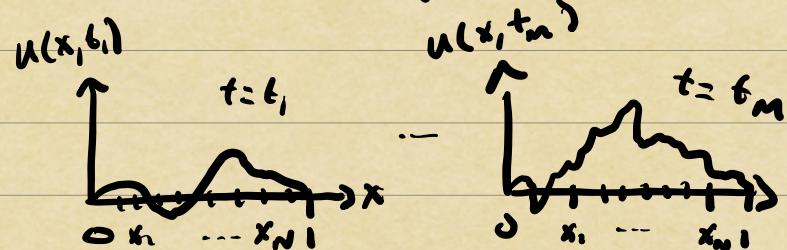
Example 2: Numerical models from data

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

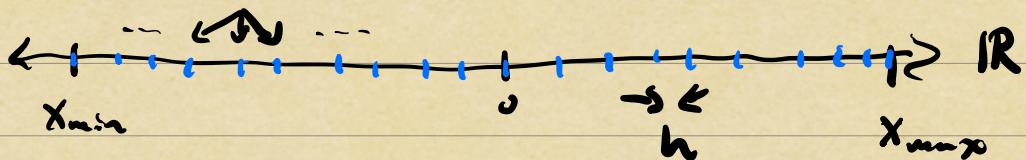
Floating Pt. Numbers

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

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

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 α . 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

an $x' \in \mathbb{F}$ such that $|x' - x| \leq \epsilon_{\text{mach}} |x|$

$$\text{where } \epsilon_{\text{mach}} = \frac{1}{2} \beta^{1-p}$$

In Double prec. $\epsilon_{\text{mach}} = 2^{-52} \approx 2.22 \times 10^{-16}$

Single prec. $\epsilon_{\text{mach}} = 2^{-23} \approx 1.19 \times 10^{-7}$

Floating Pt. Arithmetic

Real Arithmetic

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

F.P. Arithmetic

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

$f_1 : \mathbb{R} \rightarrow \mathbb{F}$, $f_1(x)$ rounds x to closest $x' \in \mathbb{F}$.

Exact rounding: $x \odot y = f_1(x \cdot y)$

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
$$(x \odot y) \oplus z \neq x \oplus (y \oplus z)$$
- Over/Underflow, Inf, NaN
- • Catastrophic cancellation
- • Accumulation of Rounding Errors

New topic

Accuracy : Stability of FP algorithms

The algorithms of NLA are usually implemented in floating point arithmetic. They are built on $\{\oplus, \ominus, \otimes, \div\}$.

A key concern: how do rounding errors accumulate?

Suppose we want to compute $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$

Ideally, we would like to have an algorithm $\tilde{f}: \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that for any $x \in \mathbb{R}^n$

$$\|\tilde{f}(x) - f(x)\| = \|f(x)\| \mathcal{O}(\text{small})$$

↑ Big-O notation
"on the order of"

I.e., relative error in the output is "small."

Such an algorithm is called forward stable.

Big "O" notation $f(t) = \mathcal{O}(g(t))$ as $t \rightarrow \infty$
if there is a constant $C > 0$ and $t_0 > 0$ such

that $|f(t)| \leq Cg(t)$ for all $t \in \mathbb{R}$.

E.g. 1 $p(t) = t + t^2 \leq 2t$ for all $t \leq 1$

$$\Rightarrow p(t) = O(t)$$

E.g. 2 $f(t) = \sin t = t - \frac{t^3}{6} + \frac{t^5}{120} + \dots$
 $= t + O(t^3)$

Sensitivity and conditioning

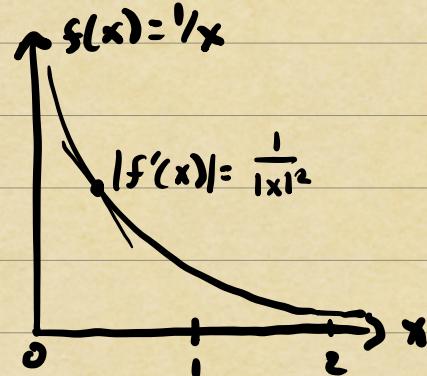
Unfortunately, forward stability is often impossible to achieve. This is because functions can be highly sensitive to small perturbations in their inputs - we call such a function ill-conditioned.

E.g. $x \in \mathbb{R} \xrightarrow{\text{round}} \tilde{x} \in \mathbb{F}$ $|x - \tilde{x}| \leq |x|(1 + \epsilon_{\text{mach}})$
 $\begin{pmatrix} \text{Scalar} \\ \text{functions} \end{pmatrix} \downarrow$
 $f(x) \quad \downarrow$
 $f(\tilde{x}) = f(x + \varepsilon) \quad |\varepsilon| \leq \epsilon_{\text{mach}}$

if f continuously differentiable

$$f(x) - f(\tilde{x}) \approx f'(x)(x - \tilde{x})$$

$$\text{error: } |f(x) - f(x+\epsilon)| \approx |f'(x)| |x| \frac{|x-\tilde{x}|}{|\tilde{x}|}$$



$\leq \underbrace{|f'(x)| |x|}_{\text{measures sensitivity}} \epsilon_{\text{mach}}$

* in floating point, we will usually be more concerned with relative errors in f.

Backward Stability

In light of perturbations due to inputs, we usually can't get forward stable algorithms for problem classes that include ill-conditioned functions. How should we judge the quality of our algorithms, then and understand their accumulation of rounding errors?

=> Ask that the algorithm behaves as if it computed exactly the right solution for a perturbed input.

We want $\tilde{f}: \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that for any $x \in \mathbb{R}^n$

$$\tilde{f}(x) = f(\tilde{x}) \text{ for some } \tilde{x}$$

$$\text{s.t. } \|x - \tilde{x}\| \leq \|x\| \Omega(\epsilon_{\text{mach}})$$

We say the algorithm \tilde{f} is backward stable.

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

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

Algorithm

Pseudocode:

$$s = 0$$

for $i = 1, \dots, n$

$$s = s + x_i$$

end

floating point

$$\tilde{s}_1 = x_1$$

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

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

Claim: Algorithm is backward stable.

PF

We'll show $\tilde{f}(x) = \sum_{i=1}^n x_i \prod_{k=1}^{i-1} (1 + \varepsilon_k)$

with $|\varepsilon_k| \leq \varepsilon_{\text{mach}}$, using induction on n .

$n=1$ $\tilde{f}(x) = x_1$ (WLOG, assume $x_1 \in \mathbb{F}$)

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

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

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

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

$$\text{So, } \tilde{f}(x) = \sum_{i=1}^n x_i (1+\delta_i) \text{ where}$$

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

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

This means $\tilde{f}(x) = f(\tilde{x})$ where $\tilde{x}_i = x_i (1+\delta_i)$

$$\text{So } |x_i - \tilde{x}_i| = |x_i| |\delta_i| \leq |x_i| O(\varepsilon_{\text{mach}})$$

$$\|x - \tilde{x}\| = \sqrt{\sum_{i=1}^n |x_i - \tilde{x}_i|^2} \leq \underbrace{\sqrt{\sum_{i=1}^n |x_i|^2}}_{\|x\|} O(\varepsilon_{\text{mach}}) \quad \square$$

Accuracy

What can we say about the accuracy of a back-and-stable algorithm?

key idea: Backward stable + well-conditioned
algorithm problem

\hat{x} = "small" forward error

Then: Compute $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ with condition number K at input $x \in \mathbb{R}^n$ on computer satisfying fundamental floating point axioms.

$$\|\tilde{f}(x) - f(x)\| = \|f(x)\| \mathcal{O}(k(x) \varepsilon_{\text{approx}})$$

The condition # K in the theorem measures the sensitivity of f to perturbed inputs.

It is independent of the algorithm used!!

We'll introduce condition #1's and prove the theorem for differentiable functions first, then discuss the non-differentiable case.

Two pieces of notation will be useful.

Little "o" notation

We say $f(t) = o(g(t))$ as $t \rightarrow 0$

if $\lim_{t \rightarrow 0} \frac{f(t)}{g(t)} = 0$

In other words, $f(t)$ decays faster than $g(t)$ as $t \rightarrow 0$.

E.g. $1 - \cos(t) = o(t)$ b/c $1 - \cos(t) = \frac{t^2}{2} + O(t^4)$
↑
"big-O"

Norms of vectors : matrices

So far we have measured the size of vectors by their Euclidean length,

$$\|v\| = \left[\sum_{i=1}^n v_i^2 \right]^{1/2}, \quad v \in \mathbb{R}^n.$$

The induced matrix norm is

$$\|A\| = \sup_{x \in \mathbb{R}^n} \frac{\|Ax\|}{\|x\|}, \quad A \in \mathbb{R}^{m \times n}.$$

The induced matrix norm measures the size of the output relative to the size of the input - it describes the largest possible

amplification of inputs when multiplied by A.

$$\Rightarrow \|Ax\| \leq \|A\| \|x\| \text{ for any } x \in \mathbb{R}^n$$

The definition holds for any vector norm,
i.e., $\|\cdot\|: \mathbb{R}^n \rightarrow \mathbb{R}$ that satisfies

- a) $\|v\| \geq 0$ and $\|v\| = 0 \Leftrightarrow v = 0$
- b) $\|v+w\| \leq \|v\| + \|w\|$
- c) $\|\alpha v\| = |\alpha| \|v\|$

Note by c), we can conveniently rewrite

$$\|A\| = \sup_{x \in \mathbb{R}^n} \frac{\|Ax\|}{\|x\|} = \sup_{\substack{x \in \mathbb{R}^n \\ \|x\|=1}} \frac{\|Ax\| / \|x\|}{\|x\| / \|x\|}$$

$$= \max_{\substack{x \in \mathbb{R}^n \\ \|x\|=1}} \|Ax\|.$$

E.g. $\|x\|_1 = \sum_{k=1}^n |x_k|$ is the "1-norm."

$$Ax = \begin{bmatrix} 1 & 1 \\ a_1 & \dots & a_n \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ a_1 \\ 1 \end{bmatrix} + \dots + x_n \begin{bmatrix} 1 \\ a_n \\ 1 \end{bmatrix}$$

$$\|Ax\|_1 \leq \sum_{k=1}^n |x_k| \|a_k\|_1 \leq \max_{1 \leq k \leq n} \|a_k\|_1 \sum_{k=1}^n |x_k|$$

$\left[\max_{1 \leq k \leq n} \|a_k\|_1 \right] \|x\|_1$

" = " holds for unit vector $\rightarrow = \|A\|_1$

$$e_j = [0 \dots 0 \underset{\uparrow}{1} 0 \dots 0]^T$$

picks out max
 $\| \cdot \|_1$, column of A

The "induced" 1-norm of a matrix is the maximum column 1-norm.

Now we're ready to define condition #'s

Condition # of a differentiable function

If $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is differentiable

$$\begin{cases} \delta f_1 = \frac{\partial f_1}{\partial x_1} \Big|_{x_1} \delta x_1 + \dots + \frac{\partial f_1}{\partial x_n} \Big|_{x_n} \delta x_n + o(\delta x) \\ \vdots \\ \delta f_m = \frac{\partial f_m}{\partial x_1} \Big|_{x_1} \delta x_1 + \dots + \frac{\partial f_m}{\partial x_n} \Big|_{x_n} \delta x_n + o(\delta x) \end{cases}$$

\downarrow goes to zero faster than δx

where $x, \delta x \in \mathbb{R}^n$ and $\delta f = f(x + \delta x) - f(x)$

\Downarrow In matrix notation

$$\begin{bmatrix} \delta f_1 \\ \vdots \\ \delta f_m \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}}_x \begin{bmatrix} \delta x_1 \\ \vdots \\ \delta x_n \end{bmatrix}$$

Jacobian matrix J_f

$$\delta f = J_f|_x \delta x + o(\delta x)$$

Jacobian $J_f|_x$ relates small changes in input x to small changes in output $f(x)$. If $\tilde{x} = x + \delta x$, we have

$$\|f(\tilde{x}) - f(x)\| \leq \|J_f|_x \delta x\| + o(\delta x)$$

$$\leq \|J_f|_x\| \|\delta x\| + o(\delta x)$$

For a backward stable algorithm,

$$\|\delta x\| = \|\tilde{x} - x\| = \|x\| O(\epsilon_{\text{mach}})$$

$$\Rightarrow \|f(\tilde{x}) - f(x)\| \leq \|f(x)\| \left[\frac{\|J_f|_x\|}{\|f(x)\|/\|x\|} \right] \frac{\|\tilde{x} - x\|}{\|x\|}$$

$$\leq \|f(x)\| k(x) \Theta(\epsilon_{\text{mach}})$$

where $\kappa(x) = \left[\frac{\|\delta f_x\|}{\|f(x)\|/\|x\|} \right]$ is the

relative condition # of f .

$\|\delta f_x\|$ by itself is the absolute condition # of f . It describes absolute changes in outputs from small absolute changes in inputs.

For non-differentiable functions, the relative condition # is

$$\kappa(x) = \limsup_{\delta \rightarrow 0} \frac{\left[\frac{\|f+\delta f\|}{\|f\|} / \frac{\|\delta x\|}{\|x\|} \right]}{\|\delta x\| \approx \delta}$$

The idea is that (similar to diff. case)

$$\text{and } \frac{\|f+\delta f\|}{\|f\|} \leq k \frac{\|\delta x\|}{\|x\|} + o(\|\delta x\|) \quad \begin{matrix} \downarrow \text{negligible for} \\ \text{small } \delta x \end{matrix}$$

↑ forward error ↑ backward error

See p. 111 in Trefethen for pf. of this in this general case.

(Revisit summation example)

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

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

Algorithm

Pseudocode:

$$s = 0$$

for $i = 1, \dots, n$

$$s = s + x_i$$

end

floating point

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

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

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

We showed $\tilde{f}(\underline{x}) = f(\tilde{\underline{x}})$ where

$$|\tilde{x}_i - x_i| \leq \|x_i\| |\delta_i| \quad |\delta_i| \leq n \epsilon_{\text{mach}} + O(\epsilon_m^2)$$

In the 1-norm, backward error is

$$\|\tilde{\underline{x}} - \underline{x}\|_1 = \sum_{i=1}^n |\tilde{x}_i - x_i| \leq \sum_{i=1}^n \|x_i\| |\delta_i|$$

$$\leq \left[\sum_{i=1}^n \|x_i\| \right] [n \epsilon_{\text{mach}} + O(\epsilon_m^2)]$$

$$\leq \|\underline{x}\|_1 (n \epsilon_{\text{mach}} + O(\epsilon_m^2))$$

Now, let's calculate the condition # of $f(x)$ and convert backward error bounds to forward error bounds.

1) Calculate Jacobian of $f(x) = \sum_{i=1}^n x_i$

$$\frac{\partial f}{\partial x_i} = 1 \Rightarrow J = \underbrace{[1 \cdots 1]}_{n \text{ times}}$$

$$\|J\|_1 = 1 \quad (\text{see 1-matrix norm e.g. above})$$

$$\Rightarrow K = \frac{\|J\|_1}{\|f(x)\|/\|x\|_1} = \frac{1}{|\sum_{i=1}^n x_i| / |\sum_{i=1}^n x_i|} = \frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|}$$

$$\left[\frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|} \right] = \text{condition } \#$$

$$= 1 \text{ when } x_i \geq 0$$

$$(\text{cancellation}) \gg 1 \text{ when } |\sum_{i=1}^n x_i| \ll \sum_{i=1}^n |x_i|$$

By this

$$\|\tilde{f}(x) - f(x)\|_1 = \|f(x)\|_1 \mathcal{O}(K(x) \epsilon_{\text{mach}})$$

Again

backward stable algorithm
+ well conditioned problem
building blocks for accurate NLA

The above process of estimating the backward error $\|\tilde{x} - x\|/\|x\|$ is called **backward error analysis**. Combining backward error bounds for an algorithm \tilde{f} with condition κ bounds for a problem f yields forward error bounds for the accuracy of the outputs.

In the simple summation example, we can also do a direct forward error analysis:

$$f(x) - \tilde{f}(x) = f(x) - f(\tilde{x}) = \sum_{i=1}^n x_i \delta_i$$

$$\Rightarrow |f(x) - \tilde{f}(x)| \leq \sum_{i=1}^n |x_i| |\delta_i| \leq n \epsilon_{mach} \sum_{i=1}^n |x_i|$$

$$\Rightarrow |f(x) - \tilde{f}(x)| \leq n \epsilon_{mach} |f(x)| \left[\frac{\sum_{i=1}^n |x_i|}{\sum_{i=1}^n \tilde{x}_i} \right] + O(\epsilon_{mach}^2)$$

Notice how the condition ~~ϵ~~ still appears
in the forward error bound!