

CS450: Numerical Analysis¹

Howard Hao Yang

Assistant Professor, ZJU-UIUC Institute



¹These slides have been modified from Michael T. Heath's material for the book "Scientific Computing: An Introductory Survey"

Lecture 1

Introduction²

² "Drink a cup of coffee. Test whose lecture you can follow during 30 mn. This defines a cut in the set of mathematicians. This cut measures the strength of the coffee." – *Coffee Meter*, constructed by Adrien Douady (1935—2006)



Scientific Computing

A little bit information about the instruction team

- Instructor: Howard H. Yang
 - ▶ Joined ZJUI in Dec. 2020
 - ▶ Research interests: (stochastic) models for the understanding and design of Internet of Things networks and distributed machine learning systems
 - ▶ I like things that *scales*, namely, they shall be tiny, low-cost, and easy-to-be-used, hence easily to spread
- The teaching assistants
 - ▶ Chenhao Wang, master student at ZJUI
 - ▶ Zeshen Li, PhD student at ZJUI

Scientific Computing

How about you?

- Why did you take this course?
- What do you want to learn from it?

Motivating Examples

Computation is the resource that has most transformed our age...



- Video game rendering.

Motivating Examples

Computation is the resource that has most transformed our age...

Default (GPT-3.5)

RO

I am going to introduce ChatGPT in my numerical analysis course, can give me a motivation to do that?



Introducing ChatGPT into your numerical analysis course can be a highly motivating and beneficial decision for both you as an instructor and your students. Here are some compelling reasons to consider:



1. Real-world Relevance: Numerical analysis is a fundamental discipline in computational mathematics with numerous real-world applications. Incorporating ChatGPT allows you to connect your course to cutting-edge technologies and their practical use. Many industries, including finance, engineering, and data science, rely on numerical analysis, and AI-driven tools like ChatGPT are increasingly part of the toolkit.
2. Hands-On Experience: ChatGPT can serve as a hands-on tool for students to apply numerical analysis concepts in practical scenarios. You can design exercises and projects that involve using ChatGPT to solve numerical problems or analyze data, providing students with a tangible application of their learning.

- Large Language models: extremely expensive in training.

Motivating Examples

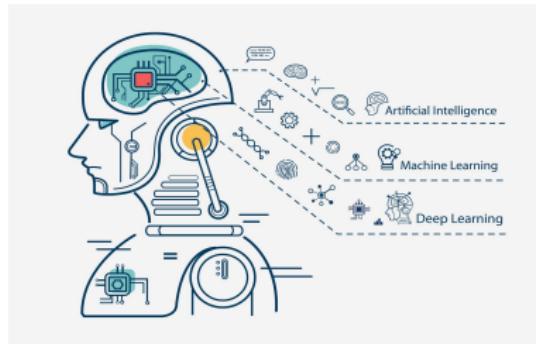
Computation is the resource that has most transformed our age...



- Computing is everything (?)

Motivating Examples

Computation is the resource that has most transformed our age...



- Machine learning models
 - ▶ Much of the computation consumption is due to problems like solving linear equations, evaluating integrals/derivatives, or finding the minimum of (highly) nonlinear functions.
 - ▶ What is common in these problems is that their solutions have no analytic form. Methods to compute such numbers, by a computer or on paper, are of an approximate nature.

Outline

1) Scientific Computing

2) Approximation and Error

3) Floating-Point Arithmetic

Scientific Computing

Introduction

- What is this course all about?
 - ▶ 'Scientific Computing' (a.k.a. 'Numerical Analysis') describes a family of approaches to obtain approximate solutions to problems once they've been stated mathematically.
- Relevant applications
 - ▶ Engineering simulations: weather forecasting, drag from flow over airplane wings, behavior of photonic devices,...
 - ▶ Machine learning: train statistical models based on a collection of data samples with unknown parameters
 - ▶ Image/audio processing: enlargement or filtering

Scientific Computing

What problems do we study, and how?

- Problems with real number, i.e., *continuous* problems
 - ▶ As opposed to *discrete* problems.
 - ▶ Including: How can we put a real number into a computer? (and with what restrictions?)
- What's the general approach?
 - ▶ Pick a general representation (e.g., a polynomial).
 - ▶ Existence/uniqueness?
 - ▶ Develop an algorithm to solve it.

Scientific Computing

Two important questions.

- **How good** of an answer can we expect to our problem?

- ▶ Can't even represent numbers exactly.
- ▶ Answers will always be approximate.
- ▶ It is natural to ask *how far off the mark* we really are.

- **How fast** can we expect the computation to complete?

- ▶ What algorithms do we use? (what is an algorithm, BTW?)
- ▶ What is the cost of these algorithms?
- ▶ Are they efficient (i.e., do they make good use of available machine time)?

- Many times, numerical analysis is about the trade-off between accuracy and efficiency.

Scientific Computing

Course Structure

- Complex problems are reduced to simpler problems
 - ▶ continuous → discretized
 - ▶ nonlinear → linear
 - ▶ complicated problems → polynomials
 - ▶ differential equations → linear equations
- Correspondingly, the contents we shall cover are
 - ▶ error analysis
 - ▶ matrix computing
 - ▶ numerical optimization
 - ▶ ordinary differential equation (ODE) and partial differential equation (PDE)

Scientific Computing

Grading Policy

- Assessment is based on
 - ▶ Attendance (record & quizzes): 10%
 - ▶ Homework (approximately bi-weekly, posted on BB): 30%
 - ▶ Midterm (open book): 15%
 - ▶ Project (paper reading, presentation, and report): 10%
 - ▶ Final (open book): 35%

Scientific Computing

Prerequisites

- Not much specific courses, but...
 - ▶ A solid foundation on linear algebra
 - ▶ A certain level of (multi-variable) calculus skill
 - ▶ Strong programming skills
 - ▶ Not scared of mathematics: Many derivations in class

Groups

Wechat group for communication and Dingding group for...

CS450 Ding



群聊: CS450 FA23



该二维码7天内(9月27日前)有效，重新进入将更新



- Strange rules but we have to abide...

Scientific Computing

What problems can we study in the first place?

- Problem is *well-posed* if solution
 - ▶ exists ✓
 - ▶ is unique ✗
 - ▶ depends continuously on problem data ✓

Otherwise, problem is *ill-posed*

- These are termed the three *Hadamard criteria*

▶ Proposed by Jacques-Salomon Hadamard in 1923 when he considered the issue of delineating between useful problems and those that weren't worth anything scientifically (at that time)

- Examples of ill-posed problems

▶ Solution to $\frac{df(t)}{dt} = \frac{3}{2}(f(t))^{1/3}$, $f(0) = 0$

▶ Solution to $\frac{df(t)}{dt} = \frac{2-f(t)}{t}$

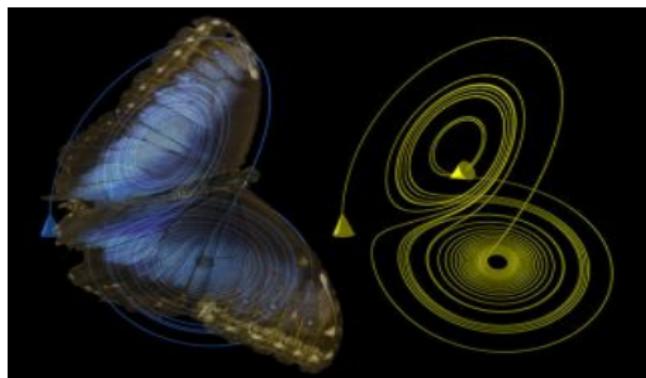
$(f(x))^{\frac{3}{2}}$

$f(t) = ? - t^{\frac{3}{2}}$

no initial point

Scientific Computing

Caveat about well-posedness.



- Even if the problem is well posed, a solution may still be *sensitive* to input data
 - ▶ Is a chaotic system well-posed or ill-posed?
- Computational algorithm should not make sensitivity worse

Approximation

Example: Approximations



- When designing the atomic bomb, scientists need to collaborate and do tons of calculations/approximations
 - ▶ Leslie Groves: “Are you saying that there’s a chance that when we push that button... we destroy the world?”
 - ▶ J. Robert Oppenheimer: “The chances are near zero...”
 - ▶ Leslie Groves: “Near zero?”
 - ▶ J. Robert Oppenheimer: “What do you want from theory alone?”
 - ▶ Leslie Groves: “Zero would be nice!”

Approximation

Example: Approximations

- Computing surface area of Earth using formula $A = 4\pi r^2$ involves several approximations
 - ▶ Earth is modeled as sphere, while its true shape is not
 - ▶ Value for radius is based on empirical measurements and previous computations
 - ▶ Value for π requires truncating infinite process
 - ▶ Values for input data and results of arithmetic operations are rounded in computer

Approximation

Sources of Approximation

- Before computation
 - ▶ modeling (George Box: All models are wrong, but some are useful)
 - ▶ empirical measurements
 - ▶ previous computations
- During computation
 - ▶ truncation or discretization
 - ▶ rounding
- Accuracy of final result reflects all these
- Uncertainty in input may be amplified by problem
- Perturbations during computation may be amplified by algorithm

Approximation

How do we measure error?

$$\frac{100000100 - 100000000}{= 100}$$

- Absolute error: approximate value – true value
- Relative error: $\frac{\text{absolute error}}{\text{true value}}$
- Equivalently, approx value = $(\underbrace{\text{true value}}_{}) \times (1 + \underbrace{\text{rel error}}_{})$
- True value usually unknown, so we estimate or bound error rather than compute it exactly
- Relative error often taken relative to approximate value, rather than (unknown) true value

Approximation

Data Error and Computational Error

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

- Example: Calculate $\sin(\pi/8)$ without a computer/calculator

① $\pi?$ or $\frac{\pi}{8}?$

$$\pi \approx 3.14$$

← approximation

② $\sin(x) \approx x$

← approximation

$$\sin\left(\frac{\pi}{8}\right) \approx \frac{\pi}{8} \approx \frac{3.14}{8}$$

$$\approx 0.3927$$

$$x = \pi/8 \quad \hat{x} = 3.14/8$$

Approximation

Data Error and Computational Error

$$\sin\left(\frac{\pi}{8}\right)$$

- Typical problem: compute value of function $f : \mathbb{R} \rightarrow \mathbb{R}$ for given argument

- x = true value of input
- $f(x)$ = desired result
- \hat{x} = approximate (inexact) input
- \hat{f} = approximate function actually computed

- Total error: $\hat{f}(\hat{x}) - f(x)$
- approximate value of function*
- $$\hat{f}(\hat{x}) - f(\hat{x}) + f(\hat{x}) - f(x)$$
- computational error* + *propagated data error*

Approximation

Data Error and Computational Error

- Back to example: Calculate $\sin(\pi/8)$ without a computer/calculator
 - ▶ true value of input: $x = \pi/8$
 - ▶ desired result: $f(x) = \sin(x)$
 - ▶ approximate (inexact) input: $\hat{x} = 3.14/8$
 - ▶ approximate function actually computed: $\hat{f}(x) = x$
- Total error:
$$\hat{f}(\hat{x}) - f(x) = 3.14/8 - \sin(\pi/8) = 0.3927 - 0.3827 = 0.01$$
 - ▶ computational error: $3.14/8 - \sin(3.14/8) = 0.0100$
 - ▶ propagated data error: $\sin(3.14/8) - \sin(\pi/8) = 1.8394 \times 10^{-4}$

Approximation

Truncation Error and Rounding Error

Taking only part of sum

- Computational error, i.e., errors made during computations, usually consists of truncation error and rounding error
- **Truncation error** : difference between true result (for actual input) and result produced by given algorithm using exact arithmetic
 - ▶ Due to approximations such as truncating infinite series or terminating iterative sequence before convergence
- **Rounding error** : difference between result produced by given algorithm using exact arithmetic and result produced by same algorithm using limited precision arithmetic
 - ▶ Due to the inexact representation of real numbers and arithmetic operations upon them

Approximation

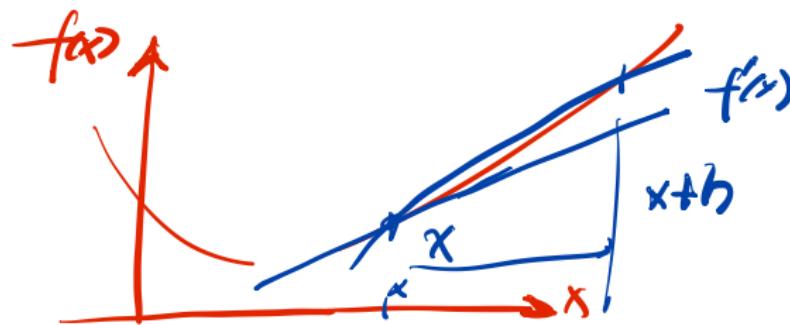
Example: Finite Difference Approximation

- Error in finite difference approximation

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

exhibits a tradeoff between rounding error and truncation error

- Elementary calculus tells us: let $h \rightarrow 0$, we can have the derivative. Is that true?



Approximation

Example: Finite Difference Approximation (cont'd)

- Error in finite difference approximation

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

- Taylor expansion of $f(x)$ gives

$$\underline{f(x+h)} = \underline{f(x)} + f'(x)h + \frac{f''(\theta)h^2}{2}, \quad \theta \in [x, x+h]$$

- Suppose evaluating f incurs an error of ϵ and denote by

$$\underline{M} = \max_{\theta \in [x, x+h]} |f''(\theta)|$$

- Total error is

$$\frac{f(x+h) - f(x)}{h} - f'(x) \leq \frac{2\epsilon}{h} + \frac{Mh}{2}$$

- The error is minimized when $h = \sqrt{M\epsilon}$

$$\underline{3.14} \approx 71$$

- \square

$$h \rightarrow 0 \quad 0.00$$
$$\sin(\frac{\pi}{4}) \quad \sin(\frac{\pi}{8})$$
$$3.1415 \dots$$

CS450: Numerical Analysis

Howard Hao Yang

Assistant Professor, ZJU-UIUC Institute

22/09/2023

Approximation Errors

- Last lecture: we want to evaluate $\sin\left(\frac{\pi}{8}\right)$, but the requirement is without using a computer/calculator
 - We have to opt for approximations
 - Using $\sin\left(\frac{\pi}{8}\right) \approx \frac{\pi}{8} \approx \frac{3.14}{8} = 0.3927$, we have an approximated solution
 - Two types of errors:
 - Truncation error, caused by approximating $\sin(x)$ by x
 - Rounding error, caused by approximating π by 3.14
 - There could be other errors, making things complicated... we want something more unified...

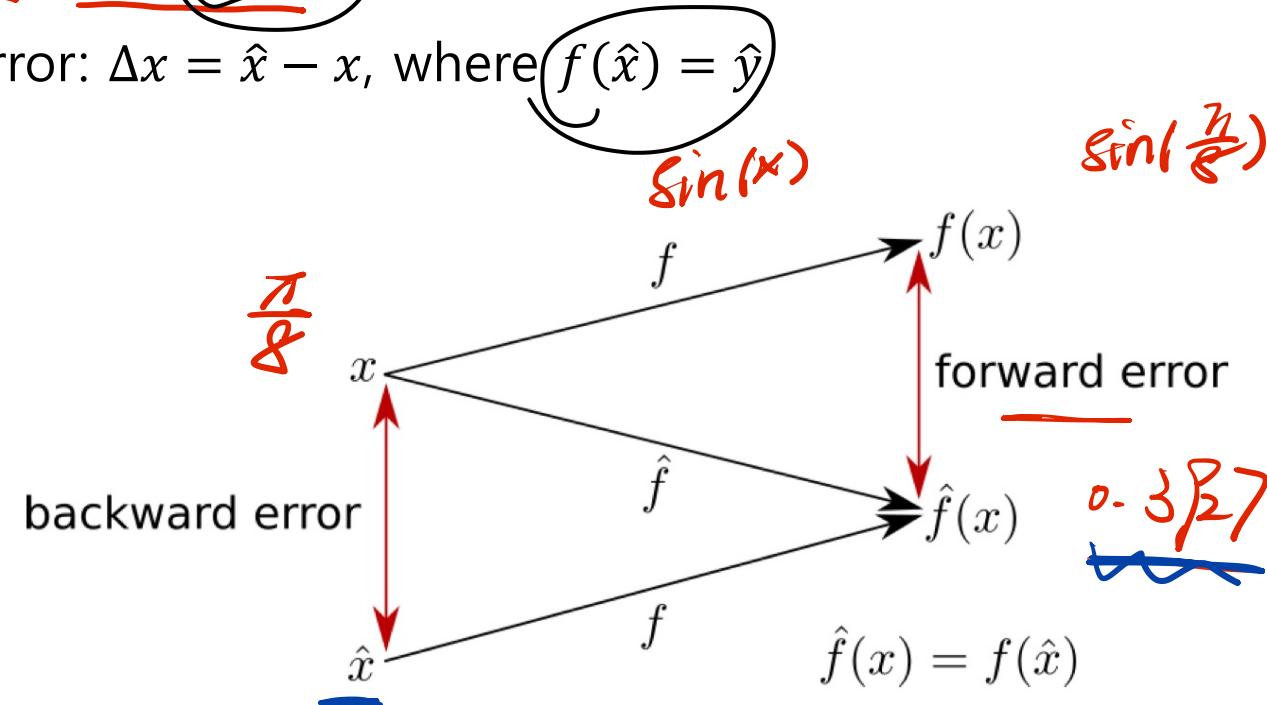
$$y = (1 + \exp(\sin(\frac{\pi}{8})))$$

Approximation and Errors

★ Forward and backward error

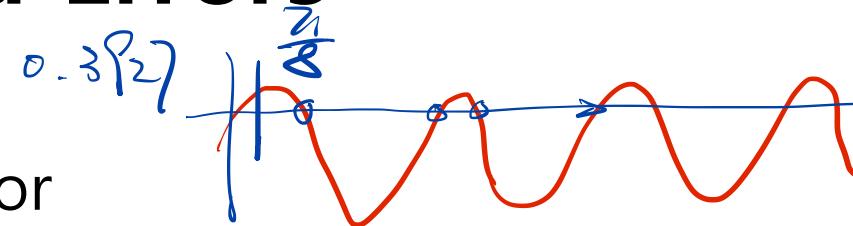
$$\sin\left(\frac{\pi}{8}\right) \rightarrow \boxed{0.3827}$$

- Suppose we want to compute $y = f(x)$, where $f: R \rightarrow R$, but obtain approximate value \hat{y}
- Forward error: $\Delta y = \hat{y} - y$
- Backward error: $\Delta x = \hat{x} - x$, where $f(\hat{x}) = \hat{y}$



Approximation and Errors

- Forward and backward error



- Suppose we want to compute $\sin\left(\frac{\pi}{8}\right)$, and the system produces $\sin\left(\frac{\pi}{8}\right) \approx \frac{\pi}{8} \approx \frac{3.14}{8} =$

0.3927

- What is the forward error?: $\Delta y = \hat{y} - y = \underbrace{0.3927 - \sin\left(\frac{\pi}{8}\right)}_{= 0.0100}$
- What is the backward error?: $\Delta x = \hat{x} - x = \underbrace{\arcsin(0.3927) - \frac{\pi}{8}}_{= 0.0109}$
- Is backward error unique?
- Why we need backward error?

$$\begin{aligned} & \text{forward error: } \Delta y = \hat{y} - y = 0.3927 - \sin\left(\frac{\pi}{8}\right) = 0.0100 \\ & \text{backward error: } \Delta x = \hat{x} - x = \arcsin(0.3927) - \frac{\pi}{8} = 0.0109 \\ & \hat{x} = \sin^{-1}(0.3927) \end{aligned}$$

Approximation and Errors

- Forward and backward error
 - What is forward error?
 - The computational error of an algorithm
 - Essentially, this is what we really want for an algorithm, but usually hard to obtain...
 - What is backward error?
 - Backward error enables us to measure computational error with respect to data propagation error



Conditioning

- Absolute condition number

- The absolute condition number is a property of the problem, measuring its sensitivity to perturbations in input
- How much a small change in the input leads to changes in the output $f(x)$.
- Formally, defined by the ratio of absolute errors at output and input

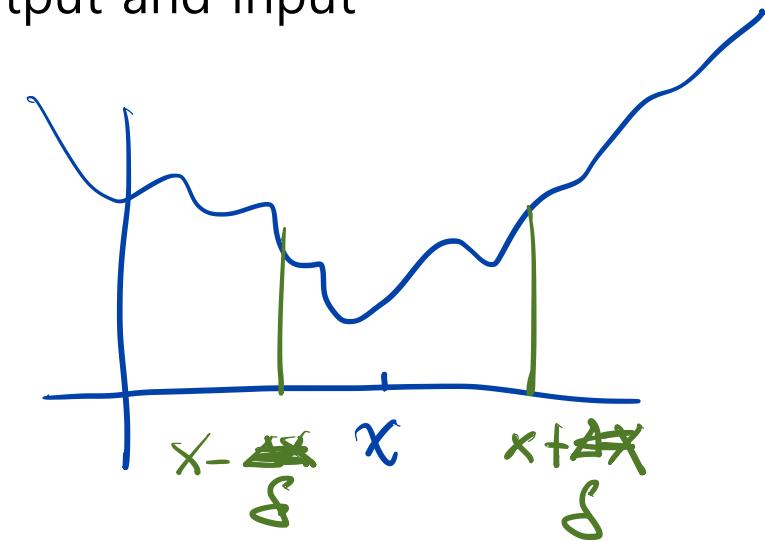
$$f: \mathbb{R} \rightarrow \mathbb{R}$$

| perturbation output)
 | perturbation input)

$\sin(x)$
 $\sin\left[\frac{\pi}{8}\right] \leftrightarrow \frac{\pi}{8} \pm 0.00001$

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|f(x + \Delta x) - f(x)\|}{\|\Delta x\|}$$

$$\max_{|\Delta x| < \delta} \frac{|f(x + \Delta x) - f(x)|}{|\Delta x|}$$



Conditioning

- Exercise
 - Absolute condition number, defined by the ratio of absolute errors at output and input

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|f(x + \Delta x) - f(x)\|}{\|\Delta x\|}$$

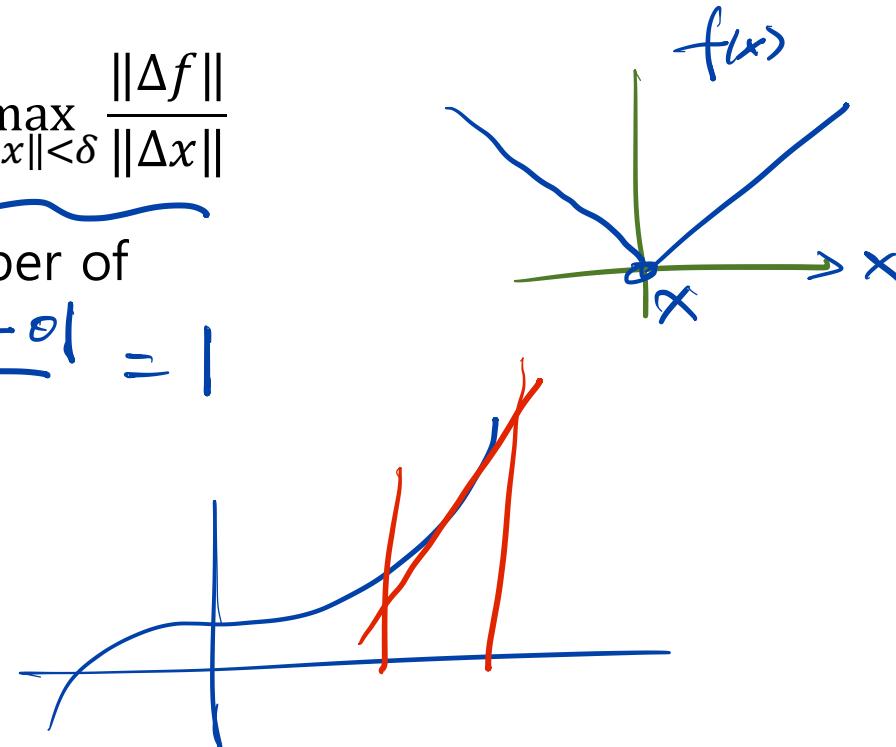
- Calculate the absolute condition number of

?

- $f(x) = |x|$ at $x = 0$
- $f(x) = e^x$ at $x = 1$

$$\max_{\|\Delta x\| < \delta} \frac{|f(x + \Delta x) - f(x)|}{|\Delta x|} = 1$$

- What do we observe?



Conditioning

- Exercise
 - Absolute condition number, defined by the ratio of absolute errors at output and input

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|\Delta f\|}{\|\Delta x\|}$$

- If f is differentiable at x , the absolute condition number is essentially the derivative at that point

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|\Delta f\|}{\|\Delta x\|} = \left\| \frac{df}{dx} \right\|$$

Conditioning

$$\frac{|\Delta y|}{|\Delta x|} \rightarrow \frac{|\Delta y/y|}{|\Delta x/x|}$$

- Relative condition number

- Formally, defined by the ratio of relative errors at output and input

$$\kappa_{rel}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|\Delta f/f(x)\|}{\|\Delta x/x\|} = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\left\| \frac{f(x + \Delta x) - f(x)}{f(x)} \right\|}{\left\| \frac{\Delta x}{x} \right\|}$$

$$= \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|f(x + \Delta x) - f(x)\|}{\|\Delta x\|} \frac{\|x\|}{\|f(x)\|} = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{|x| \cdot \kappa_{abs}(f)}{|f(x)|}$$

Conditioning

- Exercise
 - Relative condition number, defined by the ratio of relative errors at output and input

$$\kappa_{rel}(f) = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{|y - \hat{y}|}{|\hat{x} - x|}$$

- Calculate the absolute condition number of

○ $f(x) = \sqrt{x}$ at $x = 2$

○ $f(x) = e^x$ at $x = 1$

→

$$\frac{|x|}{|f'(x)|} \cdot \left| \frac{1}{\sqrt{x}} - \frac{1}{\sqrt{x+1}} \right| = \frac{1}{\sqrt{x}}$$

Conditioning

- Interpretation

- (Relative) condition number

$$\begin{aligned}K(\tan) &= \frac{|x|}{|\tan x|} \cdot |\tan'(x)| \\&= \frac{|x|}{|\tan x|} \cdot \frac{1}{|\cos^2 x|} = \frac{|x|}{|\sin(x) \cdot \cos(x)|}\end{aligned}$$

$$x_1 = \frac{\pi}{2} - 0.001 \quad) \\ \approx \infty$$

$$\kappa(f) = \frac{|x \cdot f'(x)|}{|f(x)|}$$

- Condition number indicates how much the system will be amplifying the errors in the input: suppose we are to evaluate $f(x) = \tan(x)$ for x near $\frac{\pi}{2}$
- Suppose $x_1 = \frac{\pi}{2} - 0.001$ and $x_2 = \frac{\pi}{2} - 0.002$, $|x_1 - x_2| = 0.001$ but $|f(x_1) - f(x_2)| = 500$
- What is the condition number?

x_1

$$x_1 + \Delta x = x_2$$

$$\tan(x)$$

Conditioning

- Interpretation

- (Relative) condition number

$$\kappa(f) = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{\text{forward_error}}{\text{backward_error}}$$

- Linking the forward and backward error

$$\underbrace{\text{forward_error}}_{\text{forward_error}} = \kappa(f) \times \underbrace{\text{backward_error}}_{\text{backward_error}}$$

- Condition number indicates how much the system will be amplifying the errors in the input
 - Forward error may not be easy to obtain, but condition number and backward errors are accessible

Conditioning

- Example

- Evaluate the value of $\sqrt{2}$
- Algorithm: make guess on t such that $\underbrace{t^2 \leq 2}$ until reaching some desire precision

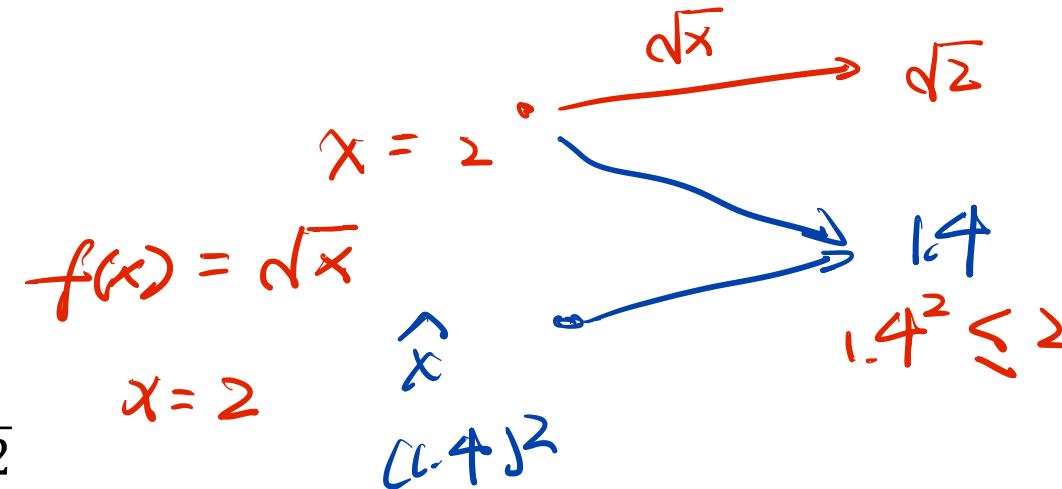
- Equivalently, it means evaluating $f(x) = \sqrt{x}$ at $x = 2$

- To start with, we approximate $\sqrt{2}$ by $t = 1.4$

- Forward error: $\underbrace{\sqrt{2} - 1.4}_{= ?} = ?$ $\frac{1}{2} \times 0.04 = 0.02$

- Backward error: $2 - (1.4)^2 = ?$ $2 - 1.96 = \underline{0.04}$

- Condition number: $\kappa(f) = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{|x|}{|\sqrt{x}|} \cdot \frac{1}{2} \cdot \frac{1}{\sqrt{x}} = \frac{1}{2}$



Conditioning

- Example

- Evaluate the value of $\sqrt{2}$
- Algorithm: make guess on t such that $\underbrace{t^2 \leq 2}$ until reaching some desire precision
- Equivalently, it means evaluating $f(x) = \sqrt{x}$ at $x = 2$
- Next, we approximate $\sqrt{2}$ by $t = 1.41$

- Forward error: $\sqrt{2} - 1.41 \leftarrow 0.0119 \times \frac{1}{2} = 0.0059$

- Backward error: $2 - (1.41)^2 = ?$ $2 - (1.41)^2 = 2 - 1.9881 = 0.0119$

- Condition number: $\kappa(f) = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{1}{2}$

Conditioning

- Remark
 - (Relative) condition number

$$\kappa(f) = \frac{|x \cdot f'(x)|}{|f(x)|}$$

- Depends on the function f itself, not on anything else
- By applying an algorithm, we evaluate $f(x)$ by $\hat{f}(x) = \hat{y}$
- Using backward error analysis, we assume $\hat{f}(x) = f(\hat{x})$ for some \hat{x} that is close to x ,
i.e., our approximated solution to the original problem is the exact solution to a
“nearby” point

Conditioning

$$\sin\left(\frac{\pi}{8}\right)$$

$$\tan\left(\frac{\pi}{2} - 0.001\right)$$



ZJU-UIUC INSTITUTE
Zhejiang University/University of Illinois at Urbana-Champaign Institute

- Remark
 - Condition number provides a sense about errors due to round-off, which might result in useless/unreliable results
 - If condition number is 1, then 1% input error → 1% output error
 - If condition number is 1000, then 0.1% input error → 100 amplification in output
- Conditioning tells only half the story...
 - Even if a problem is well conditioned, does it mean that we are bound to evaluate it well?

Algorithm Stability

- Example

- To evaluate $a_n = 2^{-n}$, use the following recursion

- $\circ a_0 = 1 + 2 \times 10^{-16}, a_1 = 1/2$

- $\circ a_{n+1} = \frac{5}{2}a_n - a_{n-1}$

- A piece of code in Matlab

```
delta = 2e-16;  
  
A(1) = 1+delta; a0  
A(2) = 1/2; a1  
  
N = 100;  
  
for i = 2:N  
    A(i+1) = 5/2*A(i) - A(i-1);  
end
```

$$a_{64} = \frac{1}{2^{+64}}$$

a₅₁

51	52	53	54	55	56	57	58	59	60	61	62	63	64
-0.0833	-0.1667	-0.3333	-0.6667	-1.3333	-2.6667	-5.3333	-10.6667	-21.3333	-42.6667	-85.3333	-170.6667	-341.3333	-682.6667

a₆₄

Algorithm Stability

- Characterization
 - Algorithm is stable if result produced is relatively insensitive to perturbations during computation
 - Stability of algorithms is analogous to conditioning of problems: Small changes in the initial data produce correspondingly small changes in the final results
 - Accuracy: closeness of computed solution to true solution of the problem
 - Applying stable algorithm to well-conditioned problem yields accurate solution

Algorithm Stability

- Characterizing stability

- Suppose that $E_0 > 0$ denotes an error introduced at some stage in the calculation and E_n represents the magnitude of the error after n subsequent operations

- If $E_n \approx CnE_0$, where C is a constant independent of n , then the growth of error is linear;

- If $E_n \approx C^n E_0$, for some $C > 1$, then the growth of error is called exponential

- Linear growth of error is usually unavoidable, and when E_0 and C are small, the results are generally acceptable

- Exponential growth of error should be avoided because the terms C^n becomes large for even relatively small values of n

Algorithm Stability

- Example

- To evaluate $a_n = 2^{-n}$, use the following recursion

- $a_0 = 1, a_1 = 1/2 \quad \times$

- $a_{n+1} = \frac{5}{2}a_n - a_{n-1}$

- What would be the growth of error?

- Step 1: check that for any constants c_1 and c_2 , $a_n = c_1 \cdot \left(\frac{1}{2}\right)^n + c_2 \cdot 2^n$ is a solution to the above recursion

$$\begin{aligned} & \frac{5}{2} \left(c_1 \cdot \left(\frac{1}{2}\right)^n + c_2 \cdot 2^n \right) \\ & - \left(c_1 \cdot \left(\frac{1}{2}\right)^{n-1} + c_2 \cdot 2^{n-1} \right) \end{aligned}$$

$$\begin{aligned} &= \left(\frac{5}{2}c_1 - 2c_1 \right) \left(\frac{1}{2}\right)^n \\ &+ \frac{5}{2}c_2 \cdot 2^n - \frac{c_2}{2} \cdot 2^n \\ &= c_1 \left(\frac{1}{2}\right)^{n+1} + c_2 \cdot 2^{n+1} \\ &= a_{n+1} \end{aligned}$$



Algorithm Stability

- Example

- To evaluate $a_n = 2^{-n}$, use the following recursion

- $a_0 = 1, a_1 = 1/2$ ✓

- $a_{n+1} = \frac{5}{2}a_n - a_{n-1}$

- Step 2: check that for any constants c_1 and c_2 , $\underbrace{a_n = c_1 \cdot \left(\frac{1}{2}\right)^n + c_2 \cdot 2^n}$ is a solution to the above recursion; and if $a_0 = 1, a_1 = 1/2$, then $c_1 = 1$ and $c_2 = 0$

$$a_n = C_1 \left(\frac{1}{2}\right)^n$$

Algorithm Stability

- Example

- To evaluate $a_n = 2^{-n}$, use the following recursion

- $a_0 = 1, a_1 = 1/2$

- $a_{n+1} = \frac{5}{2}a_n - a_{n-1}$

- Step 3: check that for any constants c_1 and c_2 , $a_n = c_1 \cdot \left(\frac{1}{2}\right)^n + c_2 \cdot 2^n$ is a solution to the above recursion; and if $\underbrace{a_0 = 1 + \delta}, \underbrace{a_1 = 1/2}$, then $c_1 = ?$ and $c_2 = ?$

$$c_1 = 1 + \frac{4}{3}\delta$$

$$c_2 = -\frac{\delta}{3}$$

$$1 + \delta = c_1 + c_2$$

$$\frac{1}{2} = \frac{c_1}{2} + 2c_2$$

Algorithm Stability

- Example

- To evaluate $a_n = 2^{-n}$, use the following recursion

- $a_0 = 1, a_1 = 1/2$

- $a_{n+1} = \frac{5}{2}a_n - a_{n-1}$

- Step 4: if $a_0 = 1 + \delta, a_1 = 1/2$, then $c_1 = ?$ and $c_2 = ?$; what is the growth of error?

$$\underline{a_n = \left(\frac{4}{3}\delta + 1\right) \cdot \frac{1}{2^n} - \frac{\delta}{3} \cdot 2^n} \quad (0^{-16})$$

Learning Objectives

- Forward error and backward error
- Conditioning number of a problem
- Algorithm stability

CS450: Numerical Analysis

Howard Hao Yang

Assistant Professor, ZJU-UIUC Institute

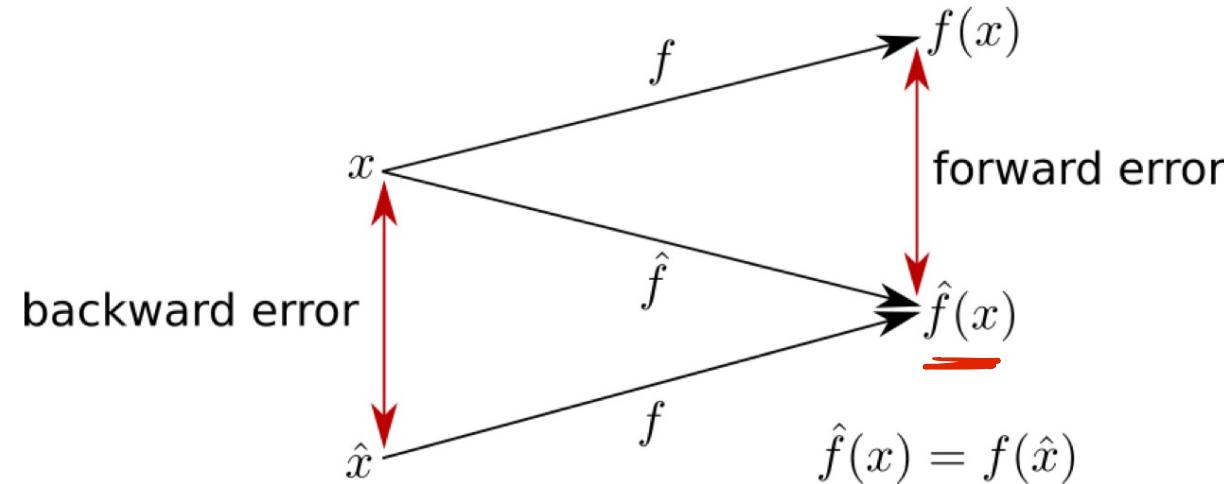
27/09/2023

Recap: Approximation and Errors

- Forward and backward error

- Suppose we want to compute $y = f(x)$, where $f: R \rightarrow R$, but obtain approximate value \hat{y}
- Forward error: $\Delta y = \hat{y} - y$
- Backward error: $\Delta x = \hat{x} - x$, where $f(\hat{x}) = \hat{y}$

$$\frac{\text{fwd-err}}{\text{bk-err}} = \text{end. \#}$$



Recap: Conditioning

- Absolute condition number

- The absolute condition number is a **property of the problem**, measuring its sensitivity to perturbations in input
- Absolute condition number, defined by the ratio of absolute errors at output and input

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|f(x + \Delta x) - f(x)\|}{\|\Delta x\|}$$

$$\kappa_{abs} = \frac{f(x)}{b}$$

- Relative condition number, defined by the ratio of relative errors at output and input

$$\kappa_{rel}(f) = \frac{|x \cdot f'(x)|}{|f(x)|}$$

$$\begin{aligned} f(x) &> 0 \\ \sin(x) &\leftarrow ? \\ x &= \pi \end{aligned}$$

Condition Number

- What if we cannot directly obtain the value of conditioning number?
 - Estimate/bound it
 - Example: evaluate the conditioning number of $f(x) = e^{\sqrt{x}}$ at $x = 2$

$$\kappa_{rel}(f) = \frac{|x \cdot f'(x)|}{|f(x)|} = \frac{|x \cdot (e^{\sqrt{x}})'|}{|e^{\sqrt{x}}|} = \frac{1}{2}\sqrt{x} = \frac{1}{2}\sqrt{2} \quad \text{if } \frac{1}{2} \times 2 = 1$$

0.1% error \rightarrow input

\rightarrow 0.1% output

Refreshing Questions

- True or false?
 - A problem is ill-conditioned if its solution is highly sensitive to small changes in the problem data.
 - Using higher-precision arithmetic will make an ill-conditioned problem better conditioned. → *Example: evaluate $\tan(\frac{\pi}{2} - 0.000001)$*
 - The conditioning of a problem depends on the algorithm used to solve it.
 - If a computational problem has a condition number of 1, is this good or bad? Why?



Computer Arithmetic

- When conducting computations, we encounter the following

$$2 + 2 = 4, 4 \times 8 = 32, (\sqrt{3})^2 = 3\dots$$

- Would it hold?: $2 + 2 = 4$
- Would it hold?: $4 \times 8 = 32$
- Would it hold?: $(\sqrt{3})^2 = 3$ 


Floating Point Numbers

$$1200 = 1.2 \times 10^3$$



- Scientific notation, format like the following

$$\underbrace{2.103}_{\text{precision}} \times 10^7 \rightarrow 21,030,000$$

- Why we need this?
- Scientific-notation provides a unique representation of any real number for a given amount of 'precision' (number of significant digits)
- Useful for computer representation & storage
- Floating point numbers are a computational realization of scientific notation

Floating Point Numbers (cont'd)



ZJU-UIUC INSTITUTE
Zhejiang University/University of Illinois at Urbana-Champaign Institute



- Floating point number system is characterized by four integers

- Base/radix: $\beta = 2$, binary

64 bits

- Precision: $p = 16$

- Exponent range: $[L, U]$

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

$\overbrace{\phantom{d_0 + \frac{d_1}{\beta} + \dots + \frac{d_{p-1}}{\beta^{p-1}}}}$

fraction

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Sign, exponent, and mantissa are stored in separate fixed-width fields

$$\begin{array}{c} 1.010111 \\ \hline \beta=2 \\ 1 + \frac{0}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{0}{2^4} + \frac{1}{2^5} \end{array}$$

6-bits

1-bit : sign

5-bits : $\beta = 10$

P.E

$$\begin{array}{c} 1 \\ \hline 2 = 4 \end{array}$$

3 bit $\rightarrow P$

2 bit $\rightarrow E$

$$\underbrace{d_0.d_1.d_2}_{\substack{3 \text{ bit} \\ \rightarrow P}} \times 10^4$$

$$\frac{9.99 \times 10^4}{4 \text{ bit} \rightarrow P}$$

1 bit $\rightarrow E$

$$d_0.d_1.d_2 \times 10^2 = 2$$

$$9.999 \times 10^2$$

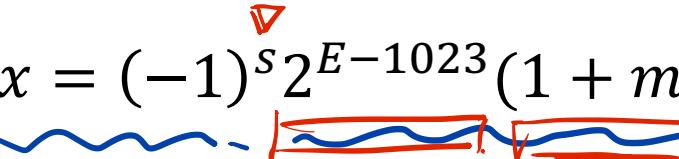
Floating Point Numbers (cont'd)

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Sign, exponent, and mantissa are stored in separate fixed-width fields
- A 64-bit (binary digit $\beta = 2$) representation $x = (-1)^s 2^{E-1023} (1 + m)$
 - The first bit is used for sign indication $s \in \{0, 1\}$
 - Followed by 11-bit exponent E
 - And a 52-bit mantissa $m = d_0.d_1d_2 \cdots d_{p-1}$



$1+11+52 = 64$

Floating Point Numbers (cont'd)

- A 64-bit (binary digit $\beta = 2$) representation $x = (-1)^s 2^{E-1023} (1 + m)$
- Example:
 - The left most bit: $s = 0$
 - Exponent: $E = 1 \cdot 2^{10} + 0 \cdot 2^9 + \dots + 0 \cdot 2^2 + \underline{1 \cdot 2^1} + 1 \cdot 2^0 = 1024 + 2 + 1 = \underline{\underline{1027}}$
 - Mantissa: $m = 1 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^4 + 1 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^8 + 1 \cdot \left(\frac{1}{2}\right)^{12}$
- The value is $x = (-1)^0 \cdot 2^{2017-2013} \left(1 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\right)^8 + \left(\frac{1}{2}\right)^{12}\right) = \underline{\underline{27.56640625}}$

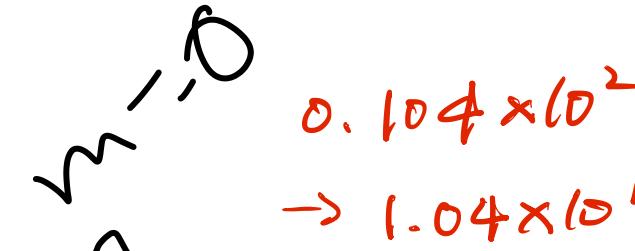
Typical Floating-Point Systems

- Parameters for typical floating-point systems

System	β	p	L	U
IEEE SP	2	24	-126	127
IEEE DP	2	53	-1,022	1,023
Cray	2	48	-16,383	16,384
HP calculator	10	12	-499	499
IBM mainframe	16	6	-64	63

- Most modern computers use binary ($\beta = 2$) arithmetic
- IEEE floating-point systems are now almost universal in digital computers

Normalization



0.104 × 10¹
→ 1.04 × 10¹

- Number x is represented as

$$x = \pm \left(\underline{d_0} + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Floating-point system is normalized if the leading digit d_0 is always non-zero unless number represented is zero ($d_0 \neq 0$)
- In normalized systems, the mantissa $m = \boxed{d_0.d_1d_2 \dots d_{p-1}}$ always satisfies $1 \leq m < \beta$ (why?)
- Why we do normalization?

$$(1-\beta) + 0. \times \beta + \dots < 1 + \beta - 1 = \beta$$

$\Leftrightarrow \beta = 10 \cdot [\leq 9.999 \dots] < 10$

Properties

β

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E \quad \triangleleft \text{FP system}$$

$L \leq E \leq U$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- How many numbers can be represented by this system?
- The number of normalized floating-point numbers is

$$2(\beta - 1)\beta^{p-1}(U - L + 1) + 1$$

- Two choices of sign
- $\beta - 1$ choices for leading digit, β choices for the $p - 1$ remaining digits
- $U - L + 1$ possible values for the exponent

Properties (cont'd)

- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Largest and smallest representable numbers are

- Underflow level (smallest positive normalized number): $\text{UFL} = \beta^L$
- Overflow level (largest floating-point number): $\text{OFL} = \beta^{U+1}(1 - \beta^{-p})$

Exceptional Values

- IEEE floating-point standard provides special values to indicate two exceptional situations
- Largest and smallest representable numbers are
 - Inf, which stands for “infinity”, results from dividing a finite number by zero (e.g., $1/0$)
 - NaN, which stands for “not a number”, results from undefined or indetermined operations such as $0/0$, $0 * \text{Inf}$, or Inf/Inf
- Inf and NaN are implemented in IEEE arithmetic through special reserved values of exponent field

Rounding Error

Excel害人事件大赏

Original 陈薛 果壳 2022-09-19 16:00 Posted on 北京

虽然看起来操作人员只是把数字和文字填进表格里，但Excel软件似乎总是有自己的想法。

自动把身份证号转换成科学计数法、把一大堆数字变成了日期.....更不要提复制粘贴表格这种高难度动作了。Excel的每一个错误结果仿佛都写满了“叛逆”。

你永远也不知道按下回车后，刚刚填进单元格里的内容会变成什么样。

A3		X	✓	fx	
1	电话				
2	1.38E+10				
3					
4					

Excel, 谢谢你自作多情的聪明

在网上随便一搜，你能看到各种Excel答疑解惑和学习资料包。也许你曾经也觉得“不就是填个表吗”，但当你看过下面这些Excel害人事件之后，可能真的想去报个班了。

Rounding

$$d_0.d_1 \dots d_{p-1} \times 10^{\beta}$$

- Not all real numbers can be represented exactly by floating-point systems
- The numbers representable are called machine numbers
- The numbers not representable are approximated by “nearby” floating-point numbers
- Two commonly used rounding rules
 - Chop: truncate base- β expansion of x after $(p - 1)$ -th precision digit; also called round toward zero
 - Round to nearest: fl(x) is nearest floating-point number to x
- Round to nearest is most accurate and is default in IEEE systems

~~1.414~~

Rounding (cont'd)

- Two commonly used rounding rules
 - Chop: truncate base- β expansion of x after $(p - 1)$ -th precision digit; also called round toward zero
 - Round to nearest: $\text{fl}(x)$ is nearest floating-point number to x
- What is the five-digit value of π under chopping and round to nearest?

$\pi = \underline{\underline{3.1415}}\underline{8267\dots}$

3.1415
3.1416
- The error that results from replacing a number with its floating-point form is called **round-off error** regardless of whether the rounding to nearest or chopping is applied

Example: Machine Precision

- Task: evaluate the following sum

$$\sum_{n=1}^{\infty} \frac{1}{n}$$

- Ideally, what is the result?
- When doing computation...
- What happens?

```
In [2]: import numpy as np

In [*]: n = int(0)

float_type = np.float32

my_sum = float_type(0)

while True:
    n += 1
    last_sum = my_sum
    my_sum += float_type(1 / n)

    if n % 200000 == 0:
        print("1/n = %g, sum0 = %g%(1.0/n, my_sum))
```

```
1/n = 5e-06, sum0 = 12.7828
1/n = 2.5e-06, sum0 = 13.4814
1/n = 1.66667e-06, sum0 = 13.8814
1/n = 1.25e-06, sum0 = 14.1666
1/n = 1e-06, sum0 = 14.3574
1/n = 8.33333e-07, sum0 = 14.5481
1/n = 7.14286e-07, sum0 = 14.7388
1/n = 6.25e-07, sum0 = 14.9296
1/n = 5.55556e-07, sum0 = 15.1203
1/n = 5e-07, sum0 = 15.3111
1/n = 4.54545e-07, sum0 = 15.4037
1/n = 4.16667e-07, sum0 = 15.4037
1/n = 3.84615e-07, sum0 = 15.4037
1/n = 3.57143e-07, sum0 = 15.4037
1/n = 3.33333e-07, sum0 = 15.4037
1/n = 3.125e-07, sum0 = 15.4037
1/n = 2.94118e-07, sum0 = 15.4037
1/n = 2.77778e-07, sum0 = 15.4037
1/n = 2.63158e-07, sum0 = 15.4037
1/n = 2.5e-07, sum0 = 15.4037
1/n = 2.38095e-07, sum0 = 15.4037
1/n = 2.27273e-07, sum0 = 15.4037
1/n = 2.17391e-07, sum0 = 15.4037
1/n = 2.08333e-07, sum0 = 15.4037
```

Machine Precision

- Definition 1: The machine precision or machine epsilon is the smallest number ϵ such that $\text{fl}(1 + \epsilon) > 1$, formally,

$$\epsilon_{\text{mach}} = \operatorname{argmin}_{\epsilon > 0} \{ \text{fl}(1 + \epsilon) \neq 1 \}$$

- Under FP system: $1 + 0.001 = 1$
- Under FP system: $1 + 0.001 = 1.001 \neq 1$
- Which is larger, the machine epsilon or underflow level?

Machine Precision (cont'd)

- Definition 2: (Equivalently,) maximum relative error in representing real number x within range of floating-point system is

$$\left| \frac{\text{fl}(x) - x}{x} \right| \leq \epsilon_{\text{mach}}$$

- Definition (significant digit): The number \hat{x} is said to approximate x to ***t significant digits*** if t is the largest nonnegative integer for which

$$\left| \frac{\hat{x} - x}{x} \right| \leq 5 \times 10^{-t}$$

Machine Precision (cont'd)

- Given a number x represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

- The machine epsilon of such a system is
 - With rounding by chopping, $\epsilon_{\text{mach}} = \beta^{1-p}$
 - With rounding to nearest, $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$
- For IEEE floating-point systems
 - In single precision: $\epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7}$
 - In double precision: $\epsilon_{\text{mach}} = 2^{-53} \approx 10^{-16}$

Machine Precision (cont'd)

- Exercise

Q1a-4⁵. In 1958 the Russians developed a ternary (base-3) computer called *Setun*, after the Setun River that flows near Moscow State University where it was built. In contrast with today's binary computers, this machine used "trits" (ternary bits) whose three possible states can be represented as $\{0, 1, 2\}$. Its floating-point number system was based on 27-trit numbers, with 9 trits reserved for the exponent and 18 for the mantissa. What was the value of machine epsilon ϵ_M for the *Setun*?

- (A) 3^{-19}
- (B) 3^{-18}
- (C) 3^{-9}
- (D) $\frac{1}{3} \cdot 2^{-18}$

Setun – Moscow State University



{ Source: JMS, plus info from <http://homepage.divms.uiowa.edu/~jones/ternary/numbers.shtml> }

Machine Precision (cont'd)

- Machine precision and underflow level
 - Machine epsilon is determined by number of digits in mantissa
 - Underflow level is determined by number of digits in exponent field
- In all practical floating-point systems

$$0 < \text{UFL} < \epsilon_{\text{mach}} < \text{OFL}$$

Floating-Point Arithmetic

- Rule: When adding two numbers, the exponents align before mantissa adds

$$2.414 \times 10^3 + 1.2 \times 10^2 = 2.414 \times 10^3 + 0.12 \times 10^3 = 2.534 \times 10^3$$

- Addition or subtraction: Shifting of mantissa to make exponents match may cause loss of some digits of smaller number, possibly all of them
- Some familiar laws of real arithmetic are not necessarily valid in floating-point system, e.g., floating-point addition and multiplication are commutative but not associative
 - If ϵ is slightly smaller than the machine precision ϵ_{mach} , then $(1 + \epsilon) + \epsilon = 1$, but $1 + (\epsilon + \epsilon) > 1$

Floating-Point Arithmetic (cont'd)

- Example
- Suppose $x = 1.48234$ and $y = 1.48235$, then $x - y = -0.00001$; but...

Let's make two numbers with very similar magnitude:

```
x = 1.48234
y = 1.48235
```

Now let's compute their difference in double precision:

```
x_dbl = np.float64(x)
y_dbl = np.float64(y)
diff_dbl = x_dbl-y_dbl

print(repr(diff_dbl))
```

-1.000000000065512e-05

Floating-Point Arithmetic (cont'd)

- Given machine epsilon ϵ_{mach} , bound the error incurred when two floating-point numbers x and y add up

$$\kappa_{add} = \frac{|x+y-\text{fl}(x)-\text{fl}(y)|}{|x+y|}$$

- What happens if $y \approx -x$?

Floating-Point Arithmetic (cont'd)

- Cancellation often implies serious loss of information
- Operands are often uncertain due to rounding or other previous errors, so relative uncertainty in difference may be large
- It is generally bad idea to compute any small quantity as difference of large quantities, since rounding error is likely to dominate result

Rounding Error in Multiplication

- This operation, in general, is safe...
- Given machine epsilon ϵ_{mach} , bound the error incurred when two floating-point numbers x and y multiply

$$\kappa_{mul} = \frac{|xy - \text{fl}(xy)|}{|xy|}.$$

Exceptional and Subnormal Numbers



- Exceptional numbers
- Subnormal numbers

- {
 - ① remember → attendance -
 - ② Quiz → next Friday (Friday next week -
 - ③ HW1 → extend one week



CS450: Numerical Analysis

Howard Hao Yang

Assistant Professor, ZJU-UIUC Institute

07/10/2023

Floating Point Numbers

- Scientific notation, format like the following

$$\underbrace{2.103}_{\text{ significand}} \times 10^{\text{ exponent}}$$

- Why we need this?
- Scientific-notation provides a unique representation of any real number for a given amount of 'precision' (number of significant digits)
- Useful for computer representation & storage
- **Floating point** numbers are a computational realization of scientific notation

Floating Point Numbers (cont'd)

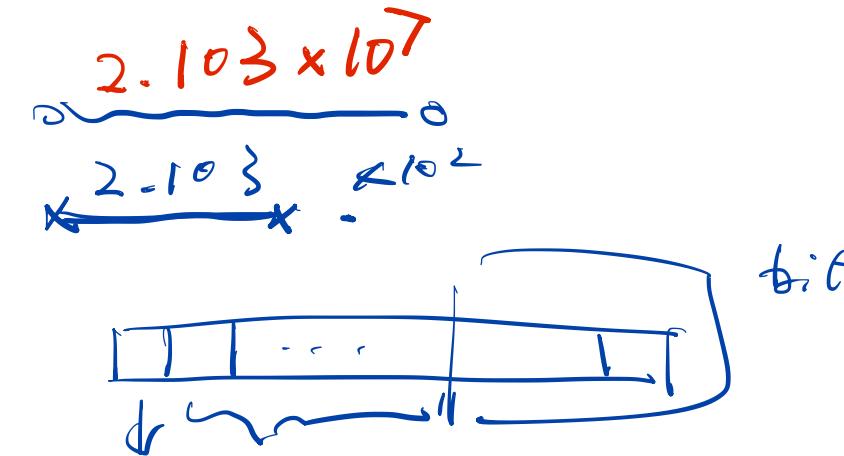
- Floating point number system is characterized by **four integers**

- Base/radix: $\beta = 10$

- Precision: $p = 4$

- Exponent range: $[L, U]$

- Number x is represented as



$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Sign, exponent, and mantissa are stored in separate fixed-width fields

Normalization

$$\begin{aligned}0.010 &\times 10^{-1} \\0.100 &\times 10^{-2} \\1.000 &\times 10^{-3}\end{aligned}$$

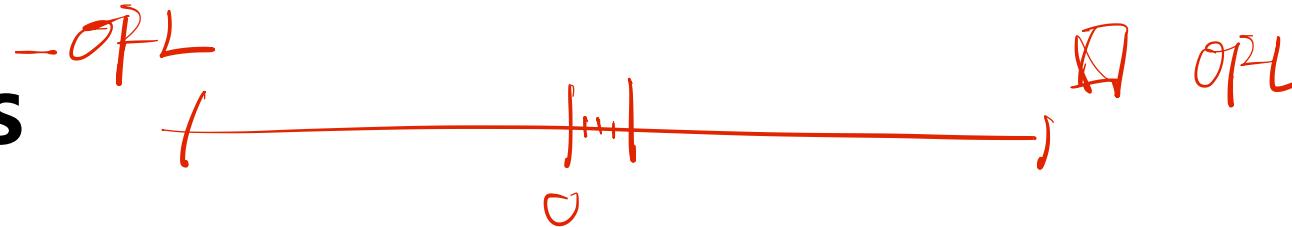
- Number x is represented as

$$x = \pm \left(\cancel{\textcolor{blue}{d}_0} + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Floating-point system is normalized if the leading digit d_0 is always non-zero unless number represented is zero
- In normalized systems, the mantissa $m = d_0.d_1d_2 \cdots d_{p-1}$ always satisfies $1 \leq m < \beta$ (why?)
- Why we do normalization?

Properties



- Number x is represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^E$$

$\begin{array}{ccccccc} 1. & . & 0 & 0 & \dots & 0 \\ (\beta-1) & (\beta-1) & \dots & \dots & (\beta-1) \end{array}$

where $0 \leq d_i \leq \beta - 1$, $i = 0, \dots, p - 1$, and $L \leq E \leq U$

- Largest and smallest representable numbers are

▪ Underflow level (smallest positive normalized number): $UFL = \beta^L$

▪ Overflow level (largest floating-point number): $OFL = \beta^{U+1}(1 - \beta^{-p})$ $= (10^8 - 10^{-4}) \times 10^2$

$$9.999 = 10 - 10^{-4}$$

$$\begin{aligned}
 \beta &= 10 \\
 p &= 4 \\
 E &= 2 \\
 \hline
 OFL &= \\
 &5.999 \times 10^2
 \end{aligned}$$

Rounding: Motivating Example

$d_0.d_1d_2 \times 10^{-e}$

- Example

- For decimal base $\beta = 10$ and precision $p = 3$, the number $x = 0.1$ is represented as

$$x = \underbrace{1.00}_{\text{---}} \times 10^{-1} \quad \Delta$$

- For binary base $\beta = 2$ and precision $p = 24$, the number $x = 0.1$ is represented as

$$x = \underbrace{1.10011001100110011001101}_{\text{---}} \times 2^{-4} \quad P = \underbrace{24,000,000,000}_{\text{---}}$$

- Can we represent $x = 0.1$ by finitely many digits in binary system?

$\sqrt{3}, \sqrt{2}, \sqrt{1},$

No

Rounding

- Not all real numbers can be represented exactly by floating-point systems
- The numbers **representable** are called **machine numbers**
- The **numbers not representable** are approximated by “nearby” floating-point numbers
- Two commonly used rounding rules
 - Chop: truncate base- β expansion of x after $(p - 1)$ -th precision digit; also called round toward zero
 - Round to nearest: $\text{fl}(x)$ is nearest floating-point number to x
- Round to nearest is most accurate and is default in IEEE systems

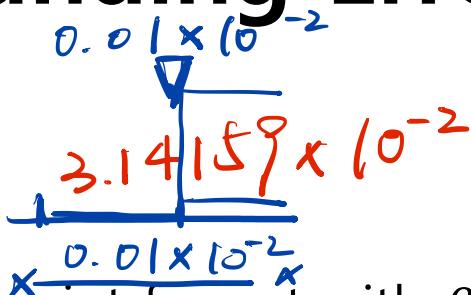
Rounding (cont'd)

- Two commonly used rounding rules
 - Chop: truncate base- β expansion of x after $(p - 1)$ -th precision digit; also called round toward zero *chopping*
 - Round to nearest: $\text{fl}(x)$ is nearest floating-point number to x
- What is the five-digit value of π under chopping and round to nearest?
- The error that results from replacing a number with its floating-point form is called **round-off error** regardless of whether the rounding to nearest or chopping is applied

Measuring Rounding Error

- Example

- Consider a floating-point format with $\beta = 10, p = 3$
- The number $x = 0.0314159$ is rounded to $\tilde{x} = 3.14 \times 10^{-2}$ in this system
- The (absolute) error is $|x - \tilde{x}| = 0.0000159$, or 0.159 units in the last place (ulps)



$$d_0, d_1, d_2 \times 10^{-e}$$

$$d_0 \neq 0$$

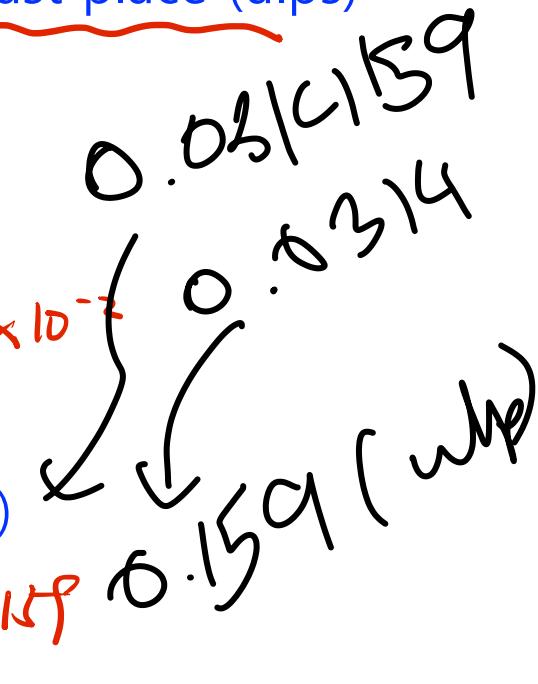
- General setting

- If a real number x is represented by the floating-point system as

$$x \approx \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^e \quad \leftarrow 3.14 \times 10^{-2}$$

- The error is $\left| \frac{x}{\beta^e} - d_0 \cdot d_1 \cdots d_{p-1} \right| \times \beta^{p-1}$ units in the last place (ulps)

$$\left| \frac{0.0314159}{10^{-2}} - 3.14 \right| \times 10^{3-1} = 0.0159$$



Example: Machine Precision

$$a+b+c \neq a+(b+c)$$

$$\begin{aligned} & 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{10^6} \\ & + \frac{1}{10^7} + \frac{2}{10^7} + \frac{3}{10^7} + \dots \end{aligned}$$

- Task: evaluate the following sum

$$\sum_{n=1}^{\infty} \frac{1}{n} = +\infty \quad (\ln x)_{x=1}^{\infty} = \infty$$

- Ideally, what is the result?
- When doing computation...
- What happens?

$$\begin{aligned} 15.4037 &\rightarrow 1.54037 \times 10^1 \\ 4.5 \cdot 10^{-7} &\rightarrow 0.00000045 \times 10^1 \end{aligned}$$

```
In [2]: import numpy as np
In [*]: n = int(0)
          float_type = np.float32
          my_sum = float_type(0)
          while True:
              n += 1
              last_sum = my_sum
              my_sum += float_type(1 / n)
              if n % 200000 == 0:
                  print("1/n = %g, sum0 = %g" % (1.0 / n, my_sum))
```

```
1/n = 5e-06, sum0 = 12.7828
1/n = 2.5e-06, sum0 = 13.4814
1/n = 1.66667e-06, sum0 = 13.8814
1/n = 1.25e-06, sum0 = 14.1666
1/n = 1e-06, sum0 = 14.3574
1/n = 8.33333e-07, sum0 = 14.5481
1/n = 7.14286e-07, sum0 = 14.7388
1/n = 6.25e-07, sum0 = 14.9296
1/n = 5.55556e-07, sum0 = 15.1203
1/n = 5e-07, sum0 = 15.311
1/n = 4.54545e-07, sum0 = 15.4037
1/n = 4.16667e-07, sum0 = 15.4037
1/n = 3.84615e-07, sum0 = 15.4037
1/n = 3.57143e-07, sum0 = 15.4037
1/n = 3.33333e-07, sum0 = 15.4037
1/n = 3.125e-07, sum0 = 15.4037
1/n = 2.94118e-07, sum0 = 15.4037
1/n = 2.77778e-07, sum0 = 15.4037
1/n = 2.63158e-07, sum0 = 15.4037
1/n = 2.5e-07, sum0 = 15.4037
1/n = 2.38095e-07, sum0 = 15.4037
1/n = 2.27273e-07, sum0 = 15.4037
1/n = 2.17391e-07, sum0 = 15.4037
1/n = 2.08333e-07, sum0 = 15.4037
```

$$\frac{1}{n} = 4.5 \times 10^{-7}$$

$$\underline{15.4037}$$

$\rightarrow -8$

$\rightarrow -8$

★ Machine Precision

- Definition 1: The machine precision or machine epsilon is the smallest number ϵ such that $\text{fl}(1 + \epsilon) > 1$, formally,

$$\epsilon_{\text{mach}} = \operatorname{argmin}_{\epsilon > 0} \{ \text{fl}(1 + \epsilon) \neq 1 \}$$

- Under FP system: $1 + 0.0001 = 1$
- Under FP system: $1 + 0.001 = 1.001 \neq 1$
- Which is larger, the machine epsilon or underflow level?

Machine Precision (cont'd)

- Definition 2: (Equivalently,) maximum relative error in representing real number x within range of floating-point system is

$$\rightarrow \left| \frac{\text{fl}(x) - \hat{x}}{x} \right| \leq \underline{\epsilon_{\text{mach}}}$$

- When a real number is rounded to the nearest floating-point number, the relative error is always bounded by ϵ_{mach} , which is referred to as **machine epsilon**

Machine Precision (cont'd)

- Given a number x represented as

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^{\lfloor \log_{\beta} x \rfloor}$$

- The machine epsilon of such a system is

- With rounding by chopping, $\epsilon_{\text{mach}} = \beta^{1-p}$ precision.
- With rounding to nearest, $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$

- For IEEE floating-point systems

- In single precision: $\epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7}$
- In double precision: $\epsilon_{\text{mach}} = 2^{-53} \approx 10^{-16}$

Machine Precision (cont'd)

- Exercise

Q1a-4⁵. In 1958 the Russians developed a ternary (base-3) computer called *Setun*, after the Setun River that flows near Moscow State University where it was built. In contrast with today's binary computers, this machine used "trits" (ternary bits) whose three possible states can be represented as $\{0, 1, 2\}$. Its floating-point number system was based on 27-trit numbers, with 9 trits reserved for the exponent and 18 for the mantissa. What was the value of machine epsilon ϵ_M for the *Setun*?

- (A) 3^{-19}
- (B) 3^{-18}
- (C) 3^{-9}
- (D) $\frac{1}{3} \cdot 2^{-18}$

Setun – Moscow State University



{ Source: JMS, plus info from <http://homepage.divms.uiowa.edu/~jones/ternary/numbers.shtml> }

Machine Precision (cont'd)

- Machine precision and units in the last place
 - Consider a floating-point format with $\beta = 10, p = 3$
 - The machine epsilon is $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p} = 5\times(10)^{-3} = 0.005$
 - The real number $x = 12.35$ is represented as $\tilde{x} = 1.24\times10^1$
 - The absolute error is $|x - \tilde{x}| = 0.5\text{ulp}$ s, the relative error is $\frac{|x - \tilde{x}|}{|x|} = 0.8\epsilon_{\text{mach}}$
 - Next, the number $8x = 98.8$ is represented as $8\tilde{x} = 9.92\times10^1$
 - The absolute error is $|8x - 8\tilde{x}| = 4.0\text{ulp}$ s, the relative error is $\frac{|8x - 8\tilde{x}|}{|8x|} = 0.8\epsilon_{\text{mach}}$

Machine Precision (cont'd)

- Machine precision and underflow level
 - Machine epsilon is determined by number of digits in mantissa
 - Underflow level is determined by number of digits in exponent field
- In all practical floating-point systems

$$0 < \text{UFL} < \epsilon_{\text{mach}} < \text{OFL}$$

Floating-Point Arithmetic

- Rule: When adding two numbers, the exponents align before mantissa adds

$$\underline{2.414 \times 10^3} + \underline{1.2 \times 10^2} = \underline{2.414} \times 10^3 + \underline{0.12} \times 10^3 = 2.534 \times 10^3$$

- Addition or subtraction: Shifting of mantissa to make exponents match may cause loss of some digits of smaller number, possibly all of them

Some familiar laws of real arithmetic are not necessarily valid in floating-point system, e.g., floating-point addition and multiplication are commutative but not associative

$$\epsilon \approx 0.8 \epsilon_{\text{mach}}$$

- If ϵ is slightly smaller than the machine precision ϵ_{mach} , then $(1 + \epsilon) + \epsilon = 1$, but $1 +$

$$(\epsilon + \epsilon) > 1$$

$$\underline{1.6 \epsilon_{\text{mach}}}$$

Floating-Point Arithmetic (cont'd)

0.1

- Example
- Suppose $x = 1.48234$ and $y = 1.48235$, then $x - y = -0.00001$; but...

Let's make two numbers with very similar magnitude:

```
x = 1.48234
y = 1.48235
```

$x - y$

$f(x - y)$

Now let's compute their difference in double precision:

```
x_dbl = np.float64(x)
y_dbl = np.float64(y)
diff_dbl = x_dbl - y_dbl
print(repr(diff_dbl))
```

-1.000000000065512e-05

-1.0×10^{-5}

Floating-Point Arithmetic (cont'd)

- Given machine epsilon ϵ_{mach} , bound the error incurred when two floating-point numbers x and y add up

$$\kappa_{\text{add}} = \frac{|x+y - \text{fl}(x) - \text{fl}(y)|}{|x+y|} \leq \frac{|x - \text{fl}(x)| + |y - \text{fl}(y)|}{|x+y|}$$

Give your fl.
 $x \rightarrow f(x)$

$$\checkmark \quad \frac{|x - \text{fl}(x)|}{|x|} \leq \epsilon_{\text{mach}}$$

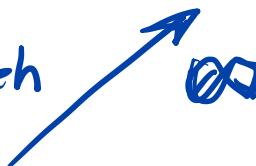
$$\leq \frac{\epsilon_{\text{mach}} \cdot |x| + \epsilon_{\text{mach}} \cdot |y|}{|x+y|}$$

$$\checkmark \quad \frac{|y - \text{fl}(y)|}{|y|} \leq \epsilon_{\text{mach}}$$

$$= \frac{|x| + |y|}{|x+y|} \cdot \epsilon_{\text{mach}}$$

- What happens if $y \approx -x$?

$$|x+y| \approx 0$$

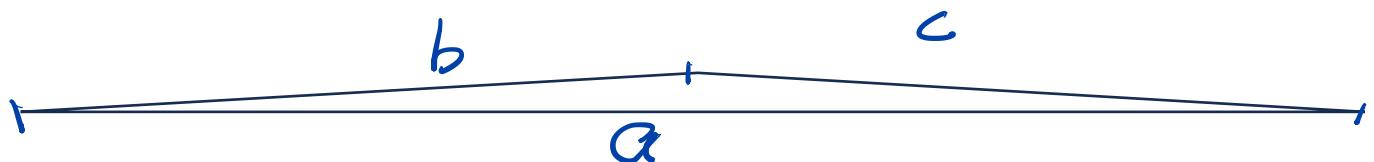


Floating-Point Arithmetic (cont'd)

- Cancellation often implies serious loss of information
- Operands are often uncertain due to rounding or other previous errors, so relative uncertainty in difference may be large
- It is generally bad idea to compute any small quantity as difference of large quantities, since rounding error is likely to dominate result

Floating-Point Arithmetic (cont'd)

- Example
 - The area of can be expressed directly in terms of its sides a , b , and c
- $$A = \sqrt{s(s - a)(s - b)(s - c)} \quad \text{where } s = \frac{a+b+c}{2} \approx a$$
- Suppose the triangle is very flat, i.e., $\underline{a \approx b + c}$. For instance, $a = 9.0$ and $b = c = 4.53$. Then, the correct value of s is $s = 9.03$ and $A = 2.34$
 - Even if we made a small rounding error in s by $s \approx 9.05$ (an error of 2 ulps), the computed value is $A = 3.04$, an error of 60 ulps
- How to improve it?



Floating-Point Arithmetic (cont'd)

- Example
 - The area of can be calculated in terms of its sides a , b , and c

$$A = \frac{\sqrt{(1a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}}{4} \quad \text{where } \underbrace{a \geq b \geq c}$$

- The calculation becomes much more stable

Rounding Error in Multiplication

- This operation, in general, is safe...
- Given machine epsilon ϵ_{mach} , bound the error incurred when two floating-point numbers x and y multiply

$$\text{fl}(xy) \approx xy(1 + \epsilon)^3$$

$$\kappa_{mul} = \frac{|xy - \text{fl}(xy)|}{|xy|} \approx |1 - (1 + \epsilon)^3| \leq 3\epsilon_{\text{mach}}$$

CS450: Numerical Analysis

Howard Hao Yang

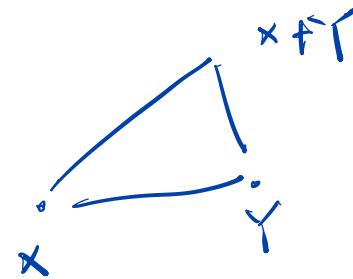


Assistant Professor, ZJU-UIUC Institute

11/10/2023

Vector Norms

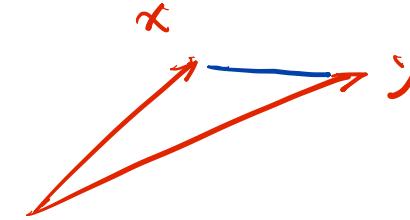
- What is a (vector) norm?
- A metric to measure the “length” of a vector, or “distance” between two vectors
 - An operator that $\|\cdot\|: \mathbb{R}^d \rightarrow \mathbb{R}_+$, and satisfies
 - $\|x\| \geq 0$ and $\|x\| = 0$ if and only if $x = \mathbf{0}$
 - $\|a \cdot x\| = |a| \cdot \|x\|$ for any $a \in \mathbb{R}$
 - $\|x + y\| \leq \|x\| + \|y\|$



Vector Norms (cont'd)

- Typical norms for vectors
 - The L-p norms
 - L-p norm: $\|x\|_p = \left(\sum_{j=1}^d x_j^p\right)^{1/p}$, $p \geq 1$
 - $p = 1$, $\|x\|_1 = \sum_j |x_j|$
 - $p = 2$, $\|x\|_2 = \sqrt{\sum_j |x_j|^2}$
 - $p = \infty$, $\|x\|_\infty = \max_j |x_j|$

Vector Norms (cont'd)



- Distance and convergence

- If x and y are vectors in \mathbb{R}^n , the distance between these two vectors is $\|x - y\|$
- Particularly, under L-2 norm and L ∞ norm, the distance is

$$\|x - y\|_2 = \sqrt{\sum_j |x_j - y_j|^2} \text{ and } \|x - y\|_\infty = \max_j |x_j - y_j|$$

- Definition (convergence): a sequence $\{x_k\}_{k=1}^\infty$ is said to converge to x in norm $\|\cdot\|$ if

$$\lim_{k \rightarrow \infty} \|x_k - \underline{x}\| = 0$$

$$\|x_k - x\|$$

$$x_k = \frac{1}{k} \rightarrow x = 0$$


Vector Norms (cont'd)

- Exercise
 - Let's try the following:

$$\begin{aligned}
 & \text{show } \|\mathbf{x}_k - \mathbf{x}\|_\infty \\
 &= \|(0, \frac{1}{k}, \frac{3}{k^2}, e^{-k} \sin k)\|_\infty \\
 &= \max\{|0|, |\frac{1}{k}|, |\frac{3}{k^2}|, |e^{-k} \sin k|\} \rightarrow 0 \\
 & k \rightarrow \infty \quad \underbrace{|0| + |\frac{1}{k}|}_{\xrightarrow{k \rightarrow \infty} 0} + \underbrace{|\frac{3}{k^2}| + |e^{-k}|}_{\xrightarrow{k \rightarrow \infty} 0} \rightarrow 0 \\
 & \|\mathbf{x}_k - \mathbf{x}\|_1 \rightarrow 0
 \end{aligned}$$

Show that

$$\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, x_4^{(k)})^t = \left(1, 2 + \frac{1}{k}, \frac{3}{k^2}, e^{-k} \sin k\right)^t$$

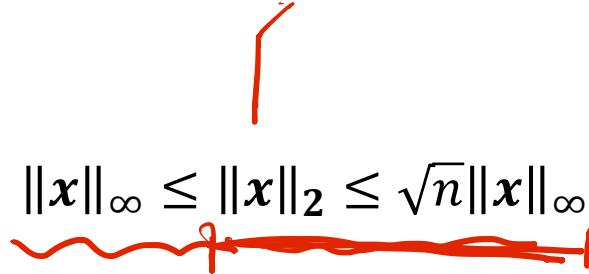
converges to $\mathbf{x} = \underline{(1, 2, 0, 0)^t}$ with respect to the l_∞ norm.

Vector Norms (cont'd)

$$n \cdot \max |x_i| \geq \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} \geq \max |x_i|$$

- Equivalence of vector norms

- For every x in R^n , it holds that $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$



- Exercise:

In Example 3, we found that the sequence $\{\mathbf{x}^{(k)}\}$, defined by

$$\mathbf{x}^{(k)} = \left(1, 2 + \frac{1}{k}, \frac{3}{k^2}, e^{-k} \sin k \right)^t,$$

converges to $\underline{\mathbf{x}} = (1, 2, 0, 0)^t$ with respect to the $\|\cdot\|_\infty$ norm. Show that this sequence also converges to \mathbf{x} with respect to the $\|\cdot\|_2$ norm.

$$n=4$$

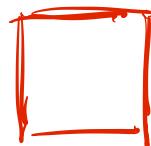
$$\sqrt[4]{4} \cdot \|\mathbf{x}^{(k)} - \mathbf{x}\|_\infty \rightarrow 0$$

$$k \rightarrow \infty$$

$$\mathbb{R}^d \quad \infty$$

X Question: would there exist a sequence $\{\mathbf{x}_k\}_{k=1}^\infty$ that converge in one norm but diverge in another norm?

Matrix Norms



- Given matrix $A \in \mathbb{R}^{n \times n}$

- We want an operator that satisfies

▽

- $\|A\| \geq 0$ and $\|A\| = 0$ if and only if $A = 0$

- $\|a \cdot A\| = |a| \cdot \|A\|$ for any $a \in R$

△

- $\|A + B\| \leq \|A\| + \|B\|$

▷

- $\|AB\| \leq \|A\| \|B\|$

- How to achieve this?

- The Frobenius norm

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

★ Matrix Norms (cont'd)

$$\max \frac{\|Ax\|}{\|x\|} = \max \|A \cdot \frac{x}{\|x\|}\|$$

- Given a vector norm $\|\cdot\|$

- The matrix norm induced by this vector norm is given as

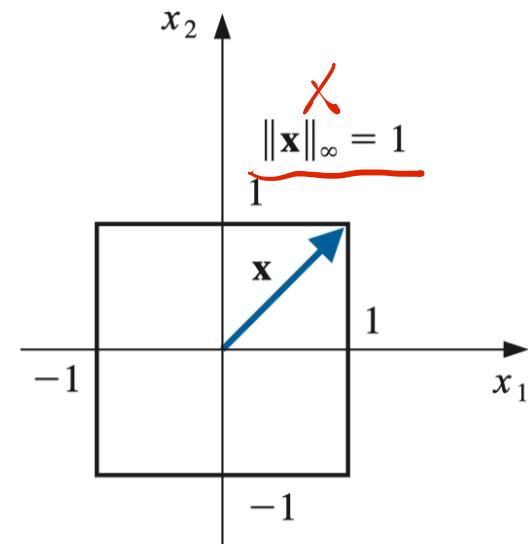
$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

- Different vector norms can induce different matrix norms for the same matrix
- Examples
 - When the norm is L-1 norm, $\|A\|_1 = \max \sum_{i=1}^m |a_{ij}|$: the maximum absolute column sum of the matrix
 - When the norm is L- ∞ norm, $\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$: the maximum absolute row sum of the matrix

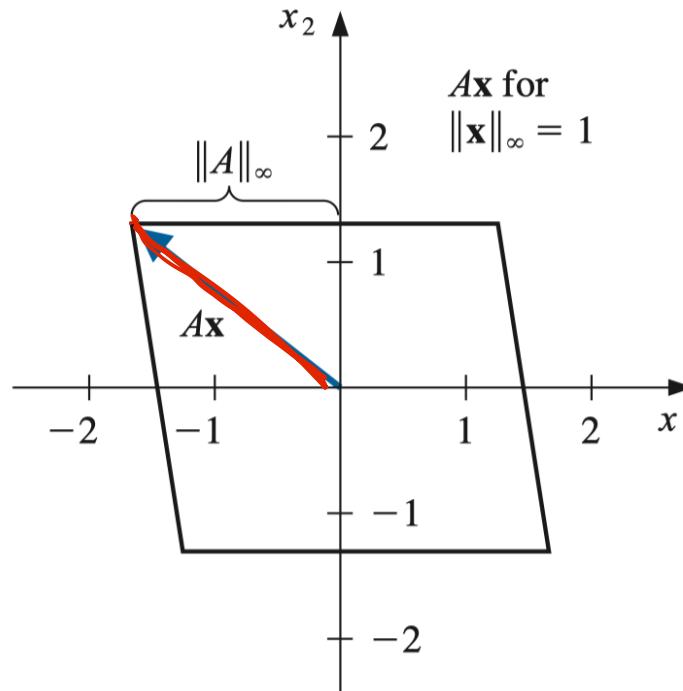
Matrix Norms (cont'd)

- Visualizing

- What is $\|A\|_\infty$ for $A = \begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix}$?



$$\|A\|_\infty = \max \left\{ \frac{0+2=2}{2+0=2} \right\} = 2$$



Matrix Norms (cont'd)

- Exercise

- What are the matrix norms induced by L-1 norm and L- ∞ norm for the following matrix?

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 1 & 0 & 1 \\ 3 & -1 & 4 \end{bmatrix}$$

$$\|A\|_1 = \max\{\|a_1\|_1, \|a_2\|_1, \|a_3\|_1\}$$

$$= \max\{6, 2, 6\} = 6$$

$$\|A\|_\infty = \max\{\|\bar{a}_1\|_1, \|\bar{a}_2\|_1, \|\bar{a}_3\|_1\} = \max\{4, 2, 8\} = 8$$

Matrix Norms (cont'd)

- When the norm is the Euclidean norm $\|\cdot\|_2$
- The matrix norm induced by this vector norm is given as

$$\frac{\|\begin{pmatrix} 1 & 2 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix}\|}{\|\begin{pmatrix} 3 \\ 4 \end{pmatrix}\|}$$

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \underbrace{\max_{\|x\|_2=1} \|Ax\|_2}_{\|Ax\|_2^L}$$

if $A = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}, \|A\|_2 = 10$

if $A = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{bmatrix}, \|A\|_2 = 10$

$$A = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix} \Rightarrow$$

$$\begin{aligned} & \frac{\|Ax\|}{\|x\|} \\ &= \left\| A \cdot \frac{x}{\|x\|} \right\| \\ &= \left\| \begin{pmatrix} 0 & x_1 \\ x_2 \end{pmatrix} \right\| \\ &= \sqrt{x_1^2 + x_2^2} \\ &= \sqrt{\|Ax\|_2^L} \end{aligned}$$

$$\begin{aligned} &= \sqrt{(0x_1)^2 + x_2^2} \\ &= \sqrt{0x_1^2 + x_2^2} \\ &= \sqrt{9x_1^2 + (x_1^2 + x_2^2)} = 1 \\ &= 10 \quad x_1 = 1, x_2 = 0 \end{aligned}$$

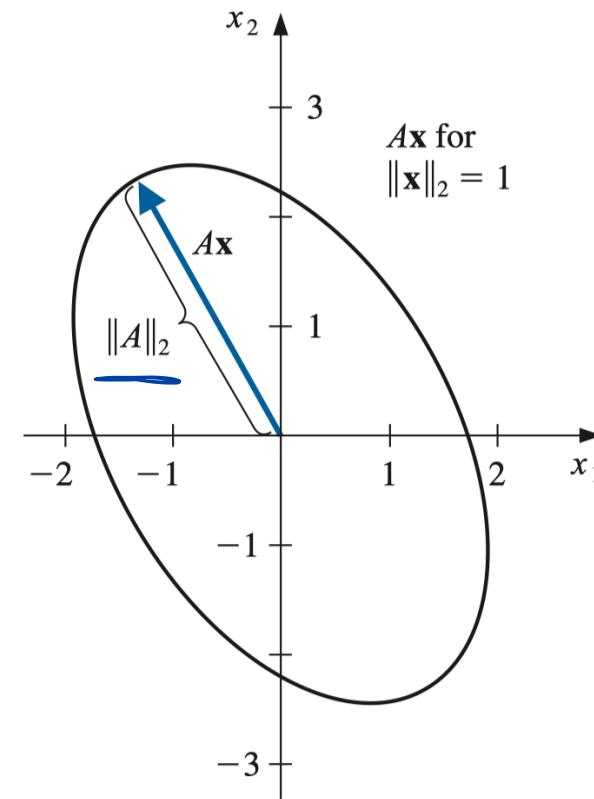
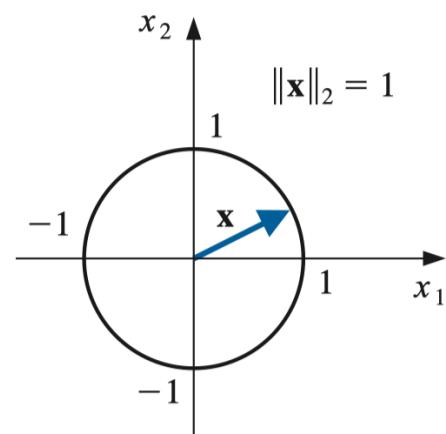
$$A = Q^T \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix} Q X$$

orthogonal matrix

$\max \{ \text{eigen values} \}$

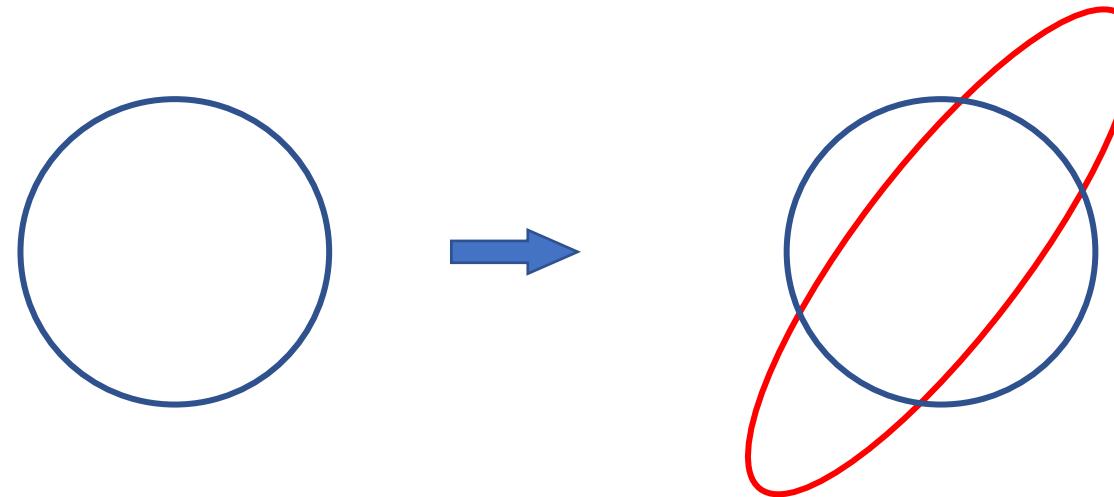
Matrix Norms (cont'd)

- Visualizing
 - What is $\|A\|_2$ for $A = \begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix}$? ≈ 2



Matrix Norm

- What does matrix multiplication do to a linear (sub)space?
 - Rotation & stretching
- What does matrix norm mean?
 - How much a particular linear operator A stretch a space



Matrix Condition Number

$$A \mathbf{x} = \mathbf{b}$$

- How is it related to a linear system?
 - Consider the following system, what is the solution?

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}.$$

A \mathbf{x} b

$$x_1 = x_2 = x_3 = x_4 = 1$$

Matrix Condition Number (cont'd)

- How is it related to a linear system?
 - Consider the following system, what is the solution?

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

\leftarrow

$+0.1$
 -0.1
 $+0.1$
 -0.1

- How about the following system, how much the variation to the above one?

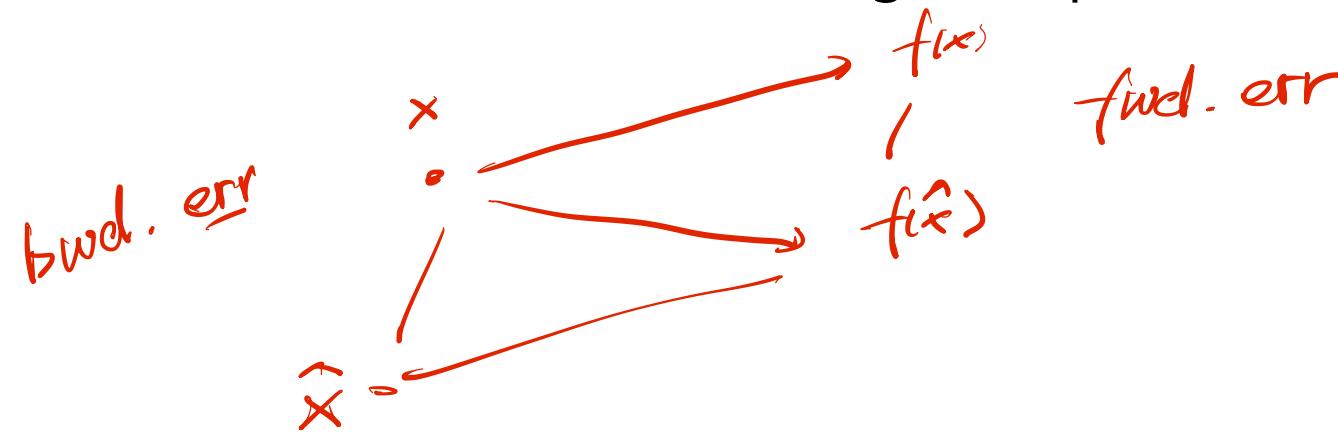
$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 + \Delta x_1 \\ x_2 + \Delta x_2 \\ x_3 + \Delta x_3 \\ x_4 + \Delta x_4 \end{pmatrix} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}$$

9.2
 -12.6
 4.5
 -1.1

$\frac{\|Ab\|}{\|b\|} \approx 0.21$

Conditioning of a Linear System

- Suppose we want to solve the linear system $\underline{Ax = b}$ how much changes in the solution if I perturb the observation a little bit?
- How do we characterize the conditioning of a problem in general?



Recap: Conditioning

- Absolute condition number
 - The absolute condition number is a **property of the problem**, measuring its sensitivity to perturbations in input
 - Absolute condition number, defined by the ratio of absolute errors at output and input

$$\kappa_{abs}(f) = \lim_{\delta \rightarrow 0} \max_{\|\Delta x\| < \delta} \frac{\|\Delta f\|}{\|\Delta x\|}$$

- Relative condition number, defined by the ratio of relative errors at output and input

$$\kappa_{rel}(f) = \frac{|x \cdot f'(x)|}{|f(x)|}$$

Conditioning of a Linear System (cont'd)

- Suppose we want to solve the linear system $\underline{Ax = b}$, if there is a small perturbation changing x to $x + \Delta x$, making \underline{b} to be $\underline{b + \Delta b}$

- What is the relationship between $\frac{\|\Delta x\|}{\|x\|}$ and $\frac{\|\Delta b\|}{\|b\|}$?

$$\|A - B\| \leq \|A\| \cdot \|B\|$$

$$\|A\| \cdot \|x\| \geq \|Ax\|$$

$$\begin{aligned} Ax &= b \\ A(x + \Delta x) &= b + \Delta b \end{aligned} \quad \left. \begin{array}{l} \Rightarrow A\Delta x = \Delta b \\ \Delta x = A^{-1}\Delta b \end{array} \right.$$

$$\frac{\|\Delta x\|}{\|x\|} = \frac{\|A^{-1} \cdot \Delta b\|}{\|x\|} \leq \frac{\|A^{-1}\| \cdot \|\Delta b\|}{\|x\|} = \frac{\|A^{-1}\| \cdot \|A\| \cdot \|\Delta b\|}{\|A\| \cdot \|x\|}$$

bwd-err & cond. fwd-err

$$\leq \frac{\|A^{-1}\| \cdot \|A\| \cdot \|\Delta b\|}{\|Ax\|} = \cancel{\|A^{-1}\| \cdot \|A\|} \cdot \underbrace{\frac{\|\Delta b\|}{\|b\|}}_{\text{cond. # of } A}$$

cond. # of A

Matrix Condition Number

$$\|A\| \cdot \|B\| \geq \|AB\|$$

- Given a matrix A , the condition number is

$$\begin{aligned}\kappa(A) &= \underbrace{\|A\|} \times \underbrace{\|A^{-1}\|} = \kappa(A^{-1}) \\ &\geq \|A \cdot A^{-1}\| = \|I\| = 1\end{aligned}$$

- What is $\kappa(A^{-1})$?
- Would it hold?: $\kappa(A) \geq 1$?
- The usefulness of the condition number is in assessing the accuracy of solutions to linear systems

Matrix Condition Number (cont'd)

$$f(x) \rightarrow f^{-1}(x) \text{ max (di)} \quad \text{Note: } f^{-1}(x) = \frac{x}{\max_i d_i}$$

- Given a matrix A , the condition number is

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|$$

- If A is a diagonal matrix, what is $\kappa(A^{-1})$?

- Large $\kappa(A)$ implies the matrix is nearly singular

- Is small determinant a good indicator for singularity?

$$A = \begin{bmatrix} \frac{1}{2^n} & \\ & \frac{1}{2^{2n}} \end{bmatrix} \rightarrow A^{-1} = \begin{bmatrix} 2^n & \\ & 2^{2n} \end{bmatrix}$$

$n \nearrow$

$$A = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} \frac{1}{10} & \\ & 1 \end{bmatrix}$$

$$\|A\| = 10$$

$$\|A^{-1}\| = 1, \kappa(A^{-1}) = 10$$

$$A^{-1}x$$

$$\det(A) = 0$$

$$\det(A) \approx 0$$



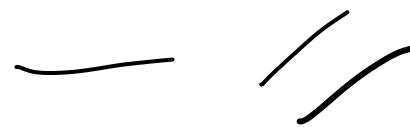
CS450: Numerical Analysis

Howard Hao Yang

Assistant Professor, ZJU-UIUC Institute

08/10/2023

Linear Systems



- We will concentrate on the following problems in the next few weeks
 - Linear system problem: $\mathbf{A}\mathbf{x} = \mathbf{b}$, find \mathbf{x}
 - Eigen vector problem: $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$, find \mathbf{x} and λ
 - Singular value decomposition: $\mathbf{A}\mathbf{v} = \sigma\mathbf{u}$, find \mathbf{v} , \mathbf{u} and σ
 - Factor the matrix as $\mathbf{A} = \mathbf{C}\mathbf{R}$
- These are central problems in linear algebra as well as data science
- Let's begin with a quick review of linear algebra
 - Main focus: column space and ranks

Matrix-Vector Multiplication

- Example: Multiply A times x

By rows $\begin{bmatrix} 2 & 3 \\ 2 & 4 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2x_1 + 3x_2 \\ 2x_1 + 4x_2 \\ 3x_1 + 7x_2 \end{bmatrix} = \text{inner products of the rows with } x = (x_1, x_2)$

By columns $\begin{bmatrix} 2 & 3 \\ 2 & 4 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 \\ 4 \\ 7 \end{bmatrix} = \text{combination of the columns } a_1 \text{ and } a_2$

- Perspective #1: produce three **inner/dot products** by each row, useful for computing, but not for understanding
- Perspective #2: linear combination of a_1 and a_2 .
- In essence, Ax is a linear combination of columns of A

Column Space

- Definition (column space): The combination of the columns fill out the column space of A . In other words, the possible outcome of Ax when x goes through all possible values: $\{Ax | x \in \mathbf{R}^d\}$.

- Example: The possible subspaces of \mathbf{R}^3

The **zero vector** $(0, 0, 0)$ by itself

A **line** of all vectors $x_1 \mathbf{a}_1$

A **plane** of all vectors $x_1 \mathbf{a}_1 + x_2 \mathbf{a}_2$

The **whole \mathbf{R}^3** with all vectors $x_1 \mathbf{a}_1 + x_2 \mathbf{a}_2 + x_3 \mathbf{a}_3$

- Exercise: What are the column spaces of $A_2 = \begin{bmatrix} 2 & 3 & 5 \\ 2 & 4 & 6 \\ 3 & 7 & 10 \end{bmatrix}$ and $A_3 = \begin{bmatrix} 2 & 3 & 1 \\ 2 & 4 & 1 \\ 3 & 7 & 1 \end{bmatrix}$?

$$\begin{bmatrix} 2 & 3 & 5 \\ 2 & 4 & 6 \\ 3 & 7 & 10 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & 1 \\ 2 & 4 & 1 \\ 3 & 7 & 1 \end{bmatrix}$$

Independent Columns and Rank



- Goal: Given matrix A , construct a matrix C where the columns come directly from A , but not to include any column that is a combination of the previous ones, following is the approach:
 - If column 1 of A is not all zero, put it into the matrix C ;
 - If column 2 of A is not a multiple of column 1, put it into C ;
 - If column 3 of A is not a combination of columns 1 and 2, put it into C . Continue.
 - At the end, C will have r columns, where $r \leq n$.
 - They will form a **basis** for the column space of A .
 - All vectors in the space are combinations of the basis vectors.

Independent Columns and Rank (cont'd)

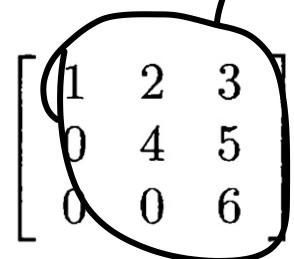
- Examples:

- If $A = \begin{bmatrix} 1 & 3 & 8 \\ 1 & 2 & 6 \\ 0 & 1 & 2 \end{bmatrix}$ then $C = \begin{bmatrix} 1 & 3 \\ 1 & 2 \\ 0 & 1 \end{bmatrix}$

R and S are linearly independent

$n = 3$ columns in A
 $r = 2$ columns in C

- If $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$ then $C = A$.



$n = 3$ columns in A
 $r = 3$ columns in C

- If $A = \begin{bmatrix} 1 & 2 & 5 \\ 1 & 2 & 5 \\ 1 & 2 & 5 \end{bmatrix}$ then $C = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

$n = 3$ columns in A
 $r = 1$ column in C

Independent Columns and Rank (cont'd)

- The number of columns r in matrix C is the “rank” of matrix A .
- The rank counts independent columns, i.e., the rank of a matrix is the dimension of its column space.
- We can construct different basis, but always the same number of vectors.
- The matrix C connects matrix A with another matrix, as $A = CR$; a factorization operation:

$$A = \begin{bmatrix} 1 & 3 & 8 \\ 1 & 2 & 6 \\ 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{bmatrix} = CR$$

Matrix-Matrix Multiplication

- Example: Multiply A times B , as $AB = C$
 - Perspective #1: inner product approach, use row of A multiply column of B , facilitates computations

**row 2 of A
column 3 of B
give c_{23} in C**

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & b_{13} \\ \cdot & \cdot & b_{23} \\ \cdot & \cdot & b_{33} \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & c_{23} \\ \cdot & \cdot & \cdot \end{bmatrix}$$

- Perspective #2: use column of A multiply with row of B , facilitates understanding

Matrix-Matrix Multiplication (cont'd)

- Outer product: one column matrix \mathbf{u} times \mathbf{B} , as $\mathbf{AB} = \mathbf{C}$
 - A column matrix \mathbf{u} times a row matrix \mathbf{v}^T produces a matrix

“Outer product” $\mathbf{uv}^T = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 3 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 6 & 8 & 12 \\ 6 & 8 & 12 \\ 3 & 4 & 6 \end{bmatrix} = \text{“rank one matrix”}$

- The column space of \mathbf{uv}^T is one-dimensional: the line in the same direction of \mathbf{u}
- All nonzero matrices \mathbf{uv}^T have rank one – they are the perfect building blocks for every matrix

Matrix-Matrix Multiplication (cont'd)

- Write the product AB as a sum of rank-one matrices
 - Column-row multiplication of matrices

$$AB = \begin{bmatrix} & & & \\ | & & | & \\ a_1 & \dots & a_n & \\ | & & | & \end{bmatrix} \begin{bmatrix} - b_1^* - \\ \vdots \\ - b_n^* - \end{bmatrix} = \underbrace{a_1 b_1^* + a_2 b_2^* + \dots + a_n b_n^*}_{\text{sum of rank 1 matrices}}$$

Rank 1 matrix

- Example

$$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} [2 \ 4] + \begin{bmatrix} 0 \\ 1 \end{bmatrix} [0 \ 5] = \begin{bmatrix} 2 & 4 \\ 6 & 12 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 17 \end{bmatrix}$$

- Why writing it as this form is important?

Ranks of AB and $A + B$

- Let's think about the following relationships
 - When we multiply matrices, can we increase the rank?
 - In other words, would it hold?: $\text{rank}(AB) > \text{rank}(A)$

- .

- When we sum up matrices, can we increase the rank?
- In other words, would it hold?: $\text{rank}(A + B) > \text{rank}(A)$.

Ranks of AB and $A + B$ (cont'd)

- Important inequalities for ranks
 - When we multiply matrices, we cannot increase the rank:
 - $\text{rank}(AB) \leq \text{rank}(A)$
 - $\text{rank}(AB) \leq \text{rank}(B)$
 - Rank of summations
 - $\text{rank}(A + B) \leq \text{rank}(A) + \text{rank}(B)$
 - Given matrix A , the rank of $A^T A$ satisfies
 - $\text{rank}(A^T A) = \text{rank}(AA^T) = \text{rank}(A)$

Four Fundamental Subspaces

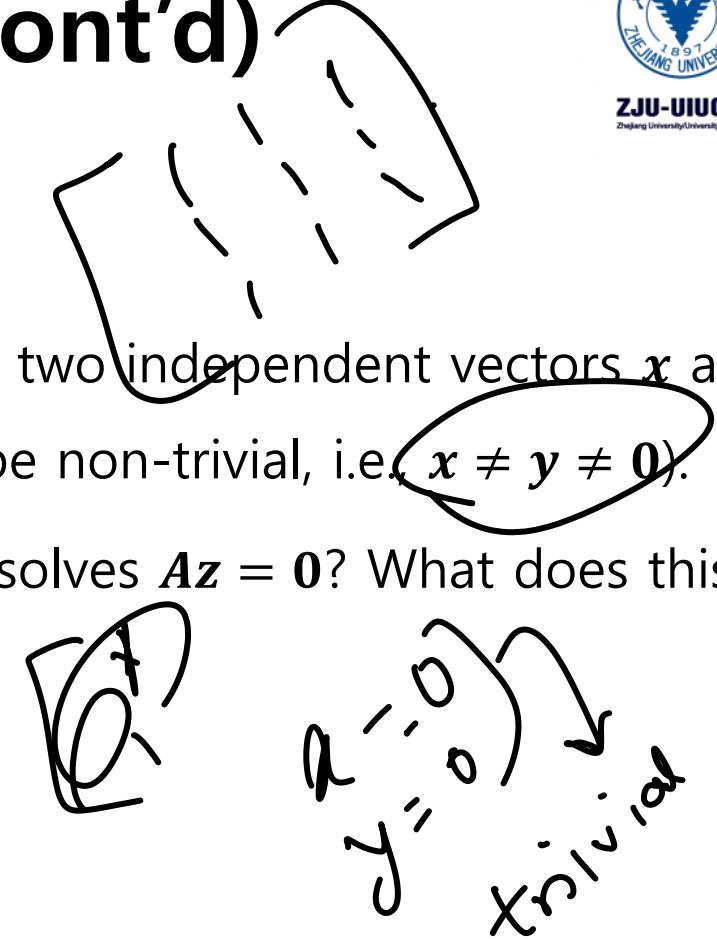
- The following subspaces are essential in characterizing A
 - The column space $\mathbf{C}(A)$ contains all combinations of the columns of A
 - The row space $\mathbf{C}(A^T)$ contains all combinations of the columns of A^T
 - The nullspace $\mathbf{N}(A)$ contains all solutions x to $Ax = \mathbf{0}$
 - The left nullspace $\mathbf{N}(A^T)$ contains all solutions y to $A^T y = \mathbf{0}$
 - Example: The null space

$$Bx = \begin{bmatrix} 1 & -2 & -2 \\ 3 & -6 & -6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ has solutions } x_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \text{ and } x_2 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

Four Fundamental Subspaces (cont'd)

- Exercise

- Suppose matrix A is a 3-by-3 matrix of all ones, find two independent vectors x and y that solves $Ax = \mathbf{0}$ and $Ay = \mathbf{0}$ (note: x and y shall be non-trivial, i.e. $x \neq y \neq \mathbf{0}$).
- Why don't I ask for a third independent vector that solves $Az = \mathbf{0}$? What does this imply?



Vector Norms

- What is a (vector) norm?
- A metric to measure the “length” of a vector, or “distance” between two vectors
 - An operator that $\|\cdot\|: \mathbb{R}^d \rightarrow \mathbb{R}_+$, and satisfies
 - $\|x\| \geq 0$ and $\|x\| = 0$ if and only if $x = \mathbf{0}$
 - $\|a \cdot x\| = |a| \cdot \|x\|$ for any $a \in \mathbb{R}$
 - $\|x + y\| \leq \|x\| + \|y\|$

Vector Norms (cont'd)

- Typical norms for vectors
 - The L-p norms
 - L-p norm: $\|\beta\|_p = \left(\sum_{j=1}^d \beta_j^p\right)^{1/p}$, $p \geq 1$
 - $p = 1$, $\|\beta\|_1 = \sum_j |\beta_j|$
 - $p = 2$, $\|\beta\|_2 = \sqrt{\sum_j |\beta_j|^2}$
 - $p = \infty$, $\|\beta\|_\infty = \max_j |\beta_j|$
 - The L-0 norm: Counts the number of non-zero entries, e.g., if $\beta = (10, 0, 2, 0.01, 0, 1)^T$, then $\|\beta\|_0 = 4$

Vector Norms (cont'd)

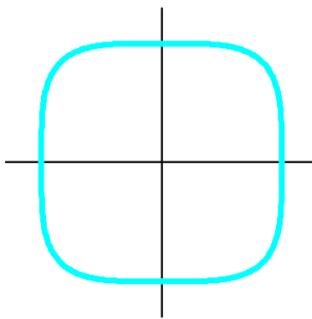
- Exercise
 - For vector $x = [-1.6, 1.2]^T$, calculate the L-1 norm, L-2 norm, and L- ∞
 - In general, for any vector x , does it hold that $\|x\|_1 \geq \|x\|_2 \geq \|x\|_\infty$?

Side-Track: Unit Ball

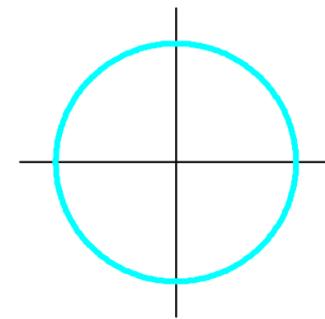
- Unit ball under different norms

- L- p norm: $\|\beta\|_p = \left(\sum_{j=1}^d \beta_j^p\right)^{1/p}$

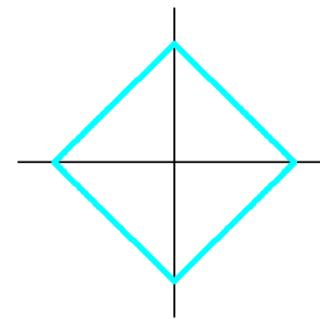
- Unit ball under L- p norm: $\{\beta \in R^d : \|\beta\|_p = 1\}$



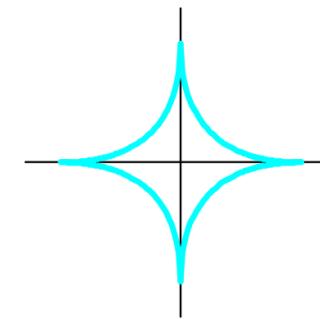
$$p = 4$$



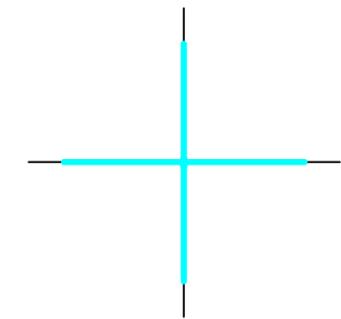
$$p = 2$$



$$p = 1$$



$$p = 0.5$$



$$p = 0.1$$

Matrix Norms

- Given matrix \mathbf{A}
 - We want an operator that satisfies
 - $\|\mathbf{A}\| \geq 0$ and $\|\mathbf{A}\| = 0$ if and only if $\mathbf{A} = \mathbf{0}$
 - $\|a \cdot \mathbf{A}\| = |a| \cdot \|\mathbf{A}\|$ for any $a \in R$
 - $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$
 - How to achieve this?
 - The Frobenius norm

Matrix Norms (cont'd)

- Given a vector norm $\|\cdot\|$
 - The matrix norm induced by this vector norm is given as

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

- Different vector norms can induce different matrix norms for the same matrix
- Examples
 - When the norm is L-1 norm, $\|A\|_1 = \max_j \sum_{i=1}^m |a_{ij}|$: the maximum absolute **column sum** of the matrix
 - When the norm is L- ∞ norm, $\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$: the maximum absolute **row sum** of the matrix

Matrix Norms (cont'd)

- Exercise
 - What are the matrix norms induced by L-1 norm and L- ∞ norm for the following matrix?

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 1 & 0 & 1 \\ 3 & -1 & 4 \end{bmatrix}$$

Matrix Norms (cont'd)

- When the norm is the Euclidean norm $\|\cdot\|_2$
 - The matrix norm induced by this vector norm is given as

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2$$

- This is called the spectral norm of matrix A

Matrix Norm

- What does matrix multiplication do to a linear (sub)space?
 - Rotation & stretching
- What does matrix norm mean?
 - How much a particular linear operator A stretch a space

