

# CS 450: Numerical Analysis<sup>1</sup>

## Introduction to Scientific Computing

University of Illinois at Urbana-Champaign

---

<sup>1</sup> *These slides have been drafted by Edgar Solomonik as lecture templates and supplementary material for the book “Scientific Computing: An Introductory Survey” by Michael T. Heath ([slides](#)).*

# Scientific Computing Applications

- ▶ **Mathematical modelling for computational science** *Typical scientific computing problems are numerical solutions to PDEs*
  - ▶ *Newtonian dynamics: simulating particle systems in time*
  - ▶ *Fluid and air flow models for engineering*
  - ▶ *PDE-constrained numerical optimization: finding optimal configurations (used in engineering of control systems)*
  - ▶ *Quantum chemistry (electronic structure calculations): many-electron Schrödinger equation*
- ▶ **Numerical algorithms: linear algebra and optimization**
  - ▶ *Linear algebra and numerical optimization are building blocks for machine learning and data science*
  - ▶ *Computer architecture, compilers, and parallel computing use numerical algorithms (matrix multiplication, Gaussian elimination) as benchmarks*

# Course Structure

- ▶ **Complex numerical problems are generally reduced to simpler problems**
  - ▶ *Discretization* corresponds to representing a continuous function/model by a discrete set of points Map continuous space to discrete space.
  - ▶ *Nonlinear problems are mapped to linear problems*
  - ▶ *Complicated functions are mapped to polynomials*
  - ▶ *Differential equations are mapped to algebraic equations*
- ▶ **The course topics will follow this hierarchical structure**
  - ▶ *Error, conditioning, and floating point are the starting point for representation and evaluation of algorithms for any numerical problem*
  - ▶ *Linear systems provide the simplest and most important building block for solving linear algebra problems*
  - ▶ *Least squares and eigenvalue problems provide basic technology for matrices*
  - ▶ *Nonlinear equations and optimization make use of matrix algebra to solve more general modelling problems*
  - ▶ *Numerical interpolation, differentiation, and quadrature provide the building blocks to reduce numerical PDE problems to matrix algebra*

# Numerical Analysis

## ► Numerical Problems involving Continuous Phenomena:

Given input  $x \in \mathbb{R}^n$ , approximate output  $y = f(x)$

- Problem is *well-posed* if  $f$  is a smoothly varying function,  $f(\hat{x}) \rightarrow f(x)$  as  $\hat{x} \rightarrow x$ .
- Otherwise, problem is *ill-posed*

## ► Error Analysis:

Quality of approximation is quantified by distance to the solution

- If solution  $y = f(x)$  is a scalar, distance from computed solution  $\hat{y}$  to correct answer is the *absolute error*

$$\Delta y = \hat{y} - y,$$

while the normalized distance is the *relative error*

$$\Delta y / y = \frac{\hat{y} - y}{|y|}$$

- More generally, we are interested in the error

$$\Delta \mathbf{y} = \hat{\mathbf{y}} - \mathbf{y}$$

the magnitude of which is measured by a given *vector norm*

# Sources of Error

## ► Representation of Numbers:

- *We cannot represent arbitrary real numbers in a finite amount of space, e.g. a computer cannot exactly represent  $\pi$*
- *Moreover, hardware architectures are only well-fit to work with fixed-length (32-bit or 64-bit) representations*
- *As we will see, the best we can do is represent a wide range of numbers with a relatively uniform **precision**, which corresponds to **scientific notation***
- *With scientific notation, we seek to store the most significant digits of each number, so that the magnitude of the relative error in the floating point representation  $fl(x)$  for most real numbers  $x$  will be  $|fl(x) - x|/|x| \leq \epsilon$*

## ► Propagated Data Error: *error due approximations in the input, $f(\hat{x}) - f(x)$*

## ► Computational Error = $\hat{f}(x) - f(x)$ = **Truncation Error** + **Rounding Error**

- **Truncation error** *is the error made due to approximations made by the algorithm (simplified models used in our approximation)*
- **Rounding error** *is the error made due to **inexact representation** of quantities computed by the algorithm*

# Error Analysis

## ► Forward Error:

*Forward error* is the computational error of an algorithm

- Absolute:  $\hat{f}(x) - f(x)$
- Relative:  $(\hat{f}(x) - f(x))/|f(x)|$
- Usually, we care about the *magnitude* of the final error, but carrying through signs is important when analyzing error

## ► Backward Error:

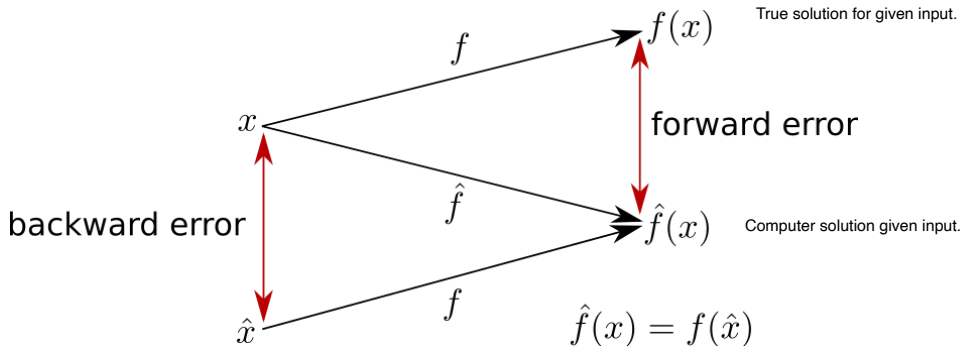
It can be hard to tell what a 'good' forward error is, but *backward error analysis* enables us to measure computational error with respect to data propagation error

- An algorithm is *backward stable* if its a solution to a nearby problem
- If the computed solution  $\hat{f}(x) = f(\hat{x})$  then

$$\text{backward error} = \hat{x} - x$$

- More precisely, we want the nearest  $\hat{x}$  to  $x$  with  $\hat{f}(x) = f(\hat{x})$
- If the backward error is smaller than the propagated data error, the solution computed by the algorithm is as good as possible

# Visualization of Forward and Backward Error



# Conditioning

## ► Absolute Condition Number:

The *absolute condition number* is a property of the problem, which measures its **sensitivity to perturbations** in input

$$\kappa_{abs}(f) = \lim_{\text{size of input perturbation} \rightarrow 0} \max_{\text{inputs}} \max_{\text{perturbations in input}} \left| \frac{\text{perturbation in output}}{\text{perturbation in input}} \right|$$

For problem  $f$  at input  $x$  it is simply the derivative of  $f$  at  $x$ ,

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

When considering a space of inputs  $\mathcal{X}$  it is  $\kappa_{abs} = \max_{x \in \mathcal{X}} \left| \frac{df}{dx}(x) \right|$

## ► (Relative) Condition Number:

$\frac{\text{Relative forward error}}{\text{backward error}} = \text{relative condition number} \times \text{relative backward error}$

The relative condition number considers relative perturbations in input and output, so that

$$\kappa(f) = \kappa_{rel}(f) = \max_{x \in \mathcal{X}} \lim_{\Delta x \rightarrow 0} \left| \frac{(f(x + \Delta x) - f(x))/f(x)}{\Delta x/x} \right| = \frac{\kappa_{abs}(f)|x|}{|f(x)|}$$



# Posedness and Conditioning

- ▶ **What is the condition number of an ill-posed problem?**
  - ▶ *If the condition number is bounded and the solution is unique, the problem is **well-posed***
  - ▶ *An **ill-posed** problem  $f$  either has no unique solution or has a (relative) condition number of  $\kappa(f) = \infty$*
  - ▶ *This condition implies that the solutions to problem  $f$  are continuous and differentiable in the given space of possible inputs to  $f$*
  - ▶ *Sometimes well-posedness is defined to only require continuity*
  - ▶ *Generally,  $\kappa(f)$  can be thought of as the reciprocal of the **distance** (in an appropriate geometric embedding of problem configurations) from  $f$  **to the nearest ill-posed problem***

# Stability and Accuracy

## ► Accuracy:

An algorithm is **accurate** if  $\hat{f}(x) = f(x)$  for all inputs  $x$  when  $\hat{f}(x)$  is computed in infinite precision

- In other words, the truncation error is zero (rounding error is ignored)
- More generally, an algorithm is accurate if its truncation error is negligible in the desired context
- Yet more generally, the **accuracy** of an algorithm is expressed in terms of bounds on the magnitude of its truncation error

## ► Stability: Accurate solution = stable algorithm + well-conditioned problem

An algorithm is **stable** if its output in finite precision (floating point arithmetic) is always near its output in exact precision

- **Stability** measures the sensitivity of an algorithm to roundoff and truncation error
- In some cases, such as the approximation of a derivative using a finite difference formula, there is a trade-off between stability and accuracy

# Error and Conditioning

- ▶ Two major sources of error: *roundoff* and *truncation* error.
  - ▶ roundoff error concerns floating point error due to finite precision
  - ▶ truncation error concerns error incurred due to algorithmic approximation, e.g. the representation of a function by a finite Taylor series

$$f(x+h) \approx g(h) = \sum_{i=0}^k \frac{f^{(i)}(x)}{i!} h^i$$

*The absolute truncation error of this approximation is*

$$f(x+h) - g(h) = \sum_{i=k+1}^{\infty} \frac{f^{(i)}(x)}{i!} h^i = O(h^{k+1}) \text{ as } h \rightarrow 0$$

- ▶ To study the propagation of roundoff error in arithmetic we can use the notion of conditioning. *The condition number tells use the worst-case amplification of output error with respect to input error*

$$\kappa(f) = \max_{x \in \mathcal{X}} \lim_{\Delta x \rightarrow 0} \left| \frac{(f(x + \Delta x) - f(x))/f(x)}{\Delta x/x} \right| = \frac{|f'(x)x|}{|f(x)|}$$

# Floating Point Numbers

**Demo:** Picking apart a floating point number

**Demo:** Density of Floating Point Numbers

## ► Scientific Notation

*Floating-point numbers are a computational realization of scientific notation,*

$$4.12165 \times 10^6, 2.145 \times 10^{-3}$$

- *Scientific-notation provides a unique representation of any real number for a given amount of 'precision' (number of significant digits)*
- *Normalized floating-point numbers are just a binary form of scientific notation,*

$$1.01001 \times 2^5, 1.0110 \times 2^{-3}$$

## ► Significand (Mantissa) and Exponent Given $x$ with $s$ leading bits $x_0, \dots, x_{s-1}$

$$\begin{aligned}\text{OFL} &= (1.11)_2 \times 2^1 = (3.5)_{10}, \\ \text{UFL} &= (1.00)_2 \times 2^{-1} = (0.5)_{10}.\end{aligned}$$

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

we wish that

$$fl(x) = \sum_{i=0}^{s-1} x_i 2^{k-i} = \underbrace{x_0.x_1 \dots x_{s-1}}_{\text{significand/mantissa}} \times 2^{\overbrace{k}^{\text{exponent}}}$$

*A floating point number's binary representation has  $s - 1$  significand bits (excluding  $x_0 = 1$ ), some bits to represent the exponent, and a sign bit*

## ► Maximum Relative Representation Error (Machine Epsilon)

- *If we have  $s$  significant digits in scientific notation, our error is bounded to variations of 1 in least significant digit, whose magnitude relative to the number we are trying to represent is  $10^{1-s}$  in decimal and  $2^{1-s}$  in binary*
- *Formally, with  $s$  significant binary digits the **relative representation error** of positive real number  $x$  is (with  $k = \lfloor \log_2(|x|) \rfloor$  and each  $x_i \in \{0, 1\}$ )*

$$x = \sum_{i=0}^{\infty} x_i 2^{k-i} = x_{rem} + \sum_{i=0}^{s-1} x_i 2^{k-i}, \quad \text{where} \quad |x_{rem}/x| \leq 2^{1-s}$$

- *The maximum such error,  $2^{1-s}$ , is called **machine epsilon**,*

$$\epsilon = \operatorname{argmin}_{\epsilon > 0} (fl(1 + \epsilon) - 1)$$

# Rounding Error in Operations (I)

**Demo:** Catastrophic Cancellation

**Activity:** Cancellation in Standard Deviation Computation

## ► Addition and Subtraction

- Subtraction is just negation of a sign bit followed by addition
- *Catastrophic cancellation* occurs when the magnitude of the result is much smaller than the magnitude of both operands
- Cancellation corresponds to losing significant digits, e.g.

$$3.1423 \times 10^5 - 3.1403 \times 10^5 = 2.0 \times 10^2$$

- Generally, we can bound the error incurred during addition of two real numbers  $x, y$  in floating point (ignoring final rounding, which has relative error  $\epsilon$ ) as

$$\frac{|(x + y) - (fl(x) + fl(y))|}{|x + y|} \leq \frac{\epsilon(|x| + |y|)}{|x + y|}$$

by this we can also observe that the condition number of addition of  $x, y$  i.e.  $f(x, y) = x + y$ , is  $\kappa(f(x, y)) = (|x| + |y|)/|x + y|$

- Consequently, when  $x + y = 0$  and  $x, y \neq 0$  addition is *ill-posed*

# Rounding Error in Operations (II)

## ► Multiplication and Division

- *Multiplication is a lot safer than addition in floating point*
- *To analyze its error, we use a 2-term **Taylor series approximation** typical in relative error analysis*

$$f(\epsilon) = (1 + n\epsilon)^k \approx f(0) + \frac{df}{d\epsilon}(0)\epsilon = 1 + kn\epsilon$$

*since  $\epsilon$  is small, this linear approximation is accurate (to within  $O(\epsilon^2)$ )*

- *Aside from final rounding, we can bound the error in multiplication as*

$$\frac{|xy - fl(fl(x)fl(y))|}{|xy|} \leq \frac{|xy - (x(1 + \epsilon)y(1 + \epsilon))(1 + \epsilon)|}{|xy|} \approx 3\epsilon$$

- *Consequently, multiplication  $f(x, y) = xy$  is always **well-conditioned**,  $\kappa(f) \approx 3$*
- *Division is multiplication by the reciprocal, and reciprocation is also well-conditioned*

# Exceptional and Subnormal Numbers

## ► Exceptional Numbers

*We had mentioned that the leading bit in normalized floating point numbers is assumed to be 1, but how do represent 0?*

- *Exceptional* floating point numbers are  $0, -0, \infty, -\infty$ , and  $\text{NaN} = 0/0 = \infty - \infty$

## ► Subnormal (Denormal) Number Range

- *The range of magnitudes of normalized floating point numbers with an exponent range  $[-e, e]$  is  $[2^{-e}, 2^{e+1}(1 - \epsilon/2)]$*
- *For numbers of magnitude  $< 2^{-e}$ , the relative representation error is unbounded*
- *Subnormal numbers* are evenly spaced in  $[-2^{-e}, 2^{-e}]$  with gaps of  $\epsilon 2^{-e}$
- *Consequently, the absolute representation error in  $[-2^{-e}, 2^{-e}]$  is at most  $\epsilon 2^{-e}$*

## ► Gradual Underflow: Avoiding underflow in addition

*The main benefit of subnormal numbers is that for any machine numbers (floating-point numbers)  $x$  and  $y$ ,  $fl(x - y) = 0$  if and only if  $x = y$ , since the gap between any two representable numbers is  $|x - y| \geq \epsilon 2^{-e}$*



## Floating Point Number Line

