Introduction to Computational Mathematics (AMATH 242/CS 371)

University of Waterloo
Winter 2019

General Information

- Instructor: Maryam Ghasemi
  - Background: B.Eng. Telecommunication Engineering
  - Msc and PhD: Applied Mathematics
- Email: m23ghase@uwaterloo.ca
- Office hour: Friday 10:30-1:30 pm, MC 6518
- Graduate TAs: Abdullah Ali Sivas, Giselle Sosa Jones
- Midterm: March 1 from 6:30-8:00

More information in Waterloo LEARN AMATH 242/CS 371- Winter 2019

General Information: References

- Primary Reference: Course notes by Hans De Sterck (available in MC 2018)
- Additional useful references:
  - Numerical Analysis, R.L. Burden and J.D. Faires (any addition)
  - Numerical Methods: Algorithms and Applications, L. Fausett, Prentice Hall, 2003
  - Introduction to Scientific Computing, Van Loan, Prentice Hall, 2000
  - Numerical Computing with MATLAB, C. Moler, SIAM, 2004
  - Numerical Analysis, T. Sauer, Addison Wesley, 2005

Grade Calculation

- Assignments 30% (4 equally weighted assignments)
- Midterm 30%
- Final 40%

Outline

- Floating point (chapter 1)
- Root Finding (chapter 2)
- Numerical Linear Algebra (chapter 3)
- Polynomial Interpolation (chapter 5)
- Integration (chapter 6)
- Discrete Fourier Methods (chapter 4)

What is the goal?

The goal of computational mathematics is defined as

- Finding or developing algorithms that solve mathematical problems computationally (i.e. with a computer).
- Desired properties of our algorithms:
  - **Accuracy:** produce a result that is numerically very close to the actual solution
  - **Efficiency:** quickly solve the problem with reasonable computational resources
  - **Robustness:** algorithm works well for a variety of inputs

Solving a problem

In solving a problem specially nonlinear stiff problem with little regularity

- Consider the problem itself
  - May be very sensitive to small changes in data
  - Can we get a good solution numerically in such cases?
- Find an algorithm
  - Some may work for all data
  - Some will only work well for particular data
  - How/why to choose one over another?

Source of error

Two main sources of error are:

1. Errors in input
   - Measurement error
   - Rounding error: computers have finite number of digits; therefore infinite precision can not be achieved. Let $x = 0.003456978$ be the input for a calculator with 4-digit-precision. After normalization, $x = 0.3456978 \times 10^{-2}$ but in the considered calculator this number is represented as $\hat{x} = 0.3457 \times 10^{-2}$. Absolute error in this process is $Error = |x - \hat{x}| = 0.0000022$.
2. Errors as a result of calculation, approximation and algorithm
   - Truncation error: Taylor series
   - Rounding error in elementary steps of the algorithm: In calculus $(\sqrt{3})^2 = 3$. Do we have precisely $(\sqrt{3})^2 = 3$ with computer arithmetic too? Try addition of $a = 2.0126 \times 10^4$ and $b = 5.6271 \times 10^5$ in a 4-digit-precision computer.

Round-off error, Catastrophic cancellation

Types of error:

- Absolute error$=|x-\hat{x}|$, where $\hat{x}$ is an approximation of x.
- Relative Error$=\dfrac{|x-\hat{x}|}{|x|}$

Example:

• Determine the absolute and relative error in the following cases:
(a) $p = 0.30012 \times 10^1$, $p^* = 0.30200 \times 10^1$
(b) $p = 0.30012 \times 10^{-2}$, $p^* = 0.30200 \times 10^{-2}$

• As another example, evaluate $f(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ at $x = 4.71$ using three-digit arithmetic. Can you introduce an alternative approach to decrease the round-off error?

An algorithm can propagate the error:
• Let $a = 0.1234567$ and $b = 0.1234111$. Show how small rounding error produced in floating point representation with 5 digits, can result in a significant error in $a - b$.

Truncation Error: Taylor series

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number a is the power series

$$f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{k=0}^{\infty} \frac{f^k(a)}{k!}(x-a)^k$$

Considering finite terms of the above infinite series gives an approximation of $f(x)$ which we define as $p_n(x)$

$$\begin{aligned} p_n(x) &= f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n \\ &= \sum_{k=0}^{n} \frac{f^k(a)}{k!}(x-a)^k \end{aligned}$$

and is called the **nth Taylor polynomial**.

Truncation Error: Taylor Expansion

Error due to truncation is defined as

$$R_n(x) = \frac{f^{n+1}(\xi_n(x))}{(n+1)!}(x-a)^{n+1}$$

Example:
1.(a) Determine Taylor series of $f(x) = \cos(x)$ about $a = 0$.

$$p_n(x) = f(0) + f'(0)(x) + \frac{f''(0)}{2!}(x)^2 + \cdots = 1 - \frac{x^2}{2} + \frac{x^4}{4!} + \cdots$$

(b) What is the largest error which might result from using the first three terms of the series to approximate $f(x) = \cos(x)$ if $-\pi \leq x \leq \pi$.

Truncation Error: *Taylor Expansion*

$$|R_3(x)| = |\frac{1}{4!}x^4 f^4(\xi(x))| = |\frac{1}{4!}x^4 \cos(\xi(x))| \qquad -\pi \le \xi \le x \le \pi$$

$$|\cos(\xi(x))| \le 1 \Longrightarrow |R_3(x)| \le |\frac{1}{4!}x^4| \le \frac{\pi^4}{4!}$$

2. Find the smallest value of n for which the n^{th} degree Taylor series for $f(x) = e^{2x}$ at $a = 0$ approximates e^{2x} on the interval $0 \le x \le 1$ with an error no greater than 10^{-6}

$$|R_n(x)| = |\frac{2^{n+1}}{(n+1)!}e^{2\xi(x)}x^{n+1}| = \frac{2^{n+1}}{(n+1)!}e^{2\xi(x)}|x^{n+1}| \quad \text{for} \quad 0 \le \xi \le x \le 1$$

$$e^{2\xi(x)} \le e^2, \quad |x^{n+1}| \le 1 \Longrightarrow |R_n(x)| \le \frac{2^{n+1}}{(n+1)!}e^2$$

$$\Longrightarrow \frac{2^{n+1}}{(n+1)!} \le 10^{-6}$$

using trial-and-error gives $n = 14$.

An algorithm can propagate the error

Catastrophic cancellation+Truncation error:
Let $f(x) = e^x$. Find $z = f(-5.5)$ by a computer with 5 significant digits. Assume that there is no initial rounding error.

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \Longrightarrow e^x = \sum_{k=0}^{n} \frac{x^k}{k!} + R_n$$

Taylor polynomial for $n = 24$ yields $\hat{z} = 0.0057563$ while the actual value is approximately 0.0040868. Relative error is significant (-41%), why?
How can we solve the issue?
Lets change the algorithm and find $f(-5.5)$ using the following algorithm:

$$e^{-x} = (\sum_{k=0}^{\infty} \frac{x^k}{k!})^{-1}$$

Then $e^{-5.5}$ using 24-th order Taylor polynomial is 0.0040865.
What do you learn?

Condition Number

Definition

A problem is well conditioned with respect to the absolute error if small changes in the input result in small changes in the output.
A problem is ill conditioned with respect to the absolute error if small changes in the input result in large changes in the output.

- Condition number with respect to the absolute error is defined as: $\kappa_A = \dfrac{\| \Delta z \|}{\| \Delta x \|}$, where Δx is change in the input and Δz is change in the output.
- Condition number with respect to the relative error is defined as: $\kappa_R = \dfrac{\frac{\|\Delta z\|}{\|z\|}}{\frac{\|\Delta x\|}{\|x\|}}$.
- For $0.1 \leq \kappa_A, \kappa_R < 10$, problem is well conditioned and for $\kappa_A, \kappa_R \to \infty$ problem is ill conditioned.

Condition Number

- Consider a problem $y = \dfrac{x}{1-x}$. Is this a well conditioned problem?

Stability of a Numerical Algorithm

If an algorithm propagates the error and produces large errors, it is called an **unstable algorithm**.
If $E_0 > 0$ denotes an error introduced at some steps in the calculations and E_n represents the magnitude of error after n subsequent operations:

- If $E_n \approx CnE_0$ (C is a constant) then the growth of error is linear.
- If $E_n \approx C^n E_0$ for $C > 1$, then the growth of error is called exponential.

Therefore, algorithm with linear growth of error is stable whereas an algorithm exhibiting exponential error growth is unstable.

Stability, Representing numbers on a computer

Example:

For any constants c_1 and c_2, $p_n = c_1(\frac{1}{3})^n + c_2 3^n$ is a solution to the recursive equation $p_n = \frac{10}{3}p_{n-1} - p_{n-2}, \quad n = 2, 3, \cdots$ Investigate the stability of this procedure if $p_0 = 1$ and $p_1 = \frac{1}{3}$.

What about $p_n = 2p_{n-1} - p_{n-2}$ with $p_n = c_1 + c_2 n$ as the solution, is this a stable procedure?($p_0 = 1$ and $p_1 = \frac{1}{3}$)

Representing integers on a computer:

- Integers-infinite range, positive and negative
- On a computer:
    - Only finite number of digits can be stored
    - A base must be chosen
    - On a computer, commonly base 2, and we call the binary digits bits.
    - There is a smallest possible integer and a largest possible integer
    - Integers in this range are stored exactly
    - Integer computations in this range are exact

Represent real numbers on a computer

- Real numbers- infinite range, positive and negative, with infinite precision (number of digits)
- Between any two real values, there are an infinite number of values
- On a computer
  - Only a finite number of digits can be stored before and after the decimal
  - Choices: fixed point and floating point representation

Fixed point representation

- A fixed point number system is characterized by 3 values:
  - b, base
  - I, number of digits for integer part
  - F, number of digits for fractional part
- Numbers are of the form: $\pm i_1 i_2 \cdots i_I . f_1 f_2 \cdots f_F$

Floating point representation

Floating point numbers are the standard tool for approximating real numbers on a computer.

- Unlike the real numbers, they provide finite precision
- Allow the decimal point to "float"

Definition

A floating point number system is defined by three components:

1. base: base of the number system, b_f
2. The mantissa: contains the normalized value of the number, m_f
3. exponent: which defines the offset from normalization, e_f.

$$F[b = b_f, m = m_f, e_f] \equiv \pm \underbrace{0.x_1 x_2 \cdots x_m}_{\text{mantissa}} \times b^{\overbrace{y_1 y_2 \cdots y_e}^{\text{exponent}}} \quad \text{where,}$$

$$1 \leq x_1 \leq b-1 \text{ and } 0 \leq x_i \leq b-1, \quad i = 2, 3, \cdots m$$

Comparing fixed point and floating point

Fixed point

- Values are evenly spaced
- Really small or really large values cannot be represented
- To represent a real number, choose the "closest" computer value (round or chop)

Floating point

- Values are not evenly spaced- smaller values are closer together
- Greater range of values can be represented (large and small)
- Again, rounding or chopping is required when choosing a representation of a value

Examples:

• Consider a fictitious computer with a floating point number system with parameters $b = 3$, $m = 4$, and $e = 2$ i.e. $F[3, 4, 2]$, represent $x = 0.0011220212$ in this system.

$$x = 0.0011220212 \xrightarrow{\text{normalization}} x = 0.11220212 \times 3^{-2}$$

$$fl(x) = \hat{x} = 0.1122 \times 3^{-02}$$

• Consider a binary computer with a floating number system $F[b = 2, m = 5, e = 3]$, represent $x = 11010.101$ in this system.

$$x = 11010.101 \xrightarrow{\text{normalization}} x = 0.11010101 \times 2^{5}$$

$$fl(x) = \hat{x} = 0.11010 \times 2^{5} \xrightarrow{\text{adjust 5}} 0.11010 \times 2^{101}$$

Standard floating point system

Single precision numbers:
This is one of the standard floating point number system, where numbers are represented on a computer in a 32-bit memory (4 bytes).

s_m	$b_1 b_2 \dots b_{23}$	s_e	$e_1 e_2 \dots e_7$

where s_m and s_e are sign bits (0 for positive, 1 for negative).
Remark(IEEE standard). Usually, signed integers are stored as two's complement because storing signed integers does not support the binary arithmetic.
So, the exponent is stored as an unsigned value and when being interpreted it is converted into an exponent within a signed range by subtracting the bias.

Therefore, 8 bits is allocated to the exponent and it can be stored in the range of $1, 2, \cdots, 254$. Exponent is interpreted by subtracting the bias (127) from an 8-bit exponent to get $-126, \cdots, +127$. So, what is represented is $E = e + 127$.
The largest number that can be represented (not in IEEE standard) is represented as :

$$
\begin{aligned}
&\hat{x} \approx |0|11111111111111111111111|0|1111111| \\
&= \frac{1}{2}\left(\frac{1 - (\frac{1}{2})^{23}}{1 - \frac{1}{2}}\right) = (1 - 2^{-23}) \times 2^{127} \\
&\approx (1 - 2^{-23}) \times 2^{127} \approx 2^{127} \approx 1.7 \times 10^{38}
\end{aligned}
$$

Standard floating point system: double precision

The smallest number is

$$
\begin{aligned}
\hat{x} &\approx |0|00000000000000000000001|1|111111| \\
&= 2^{-23} \times 2^{-127} \\
&= 2^{-150} \approx 7 \times 10^{-46}
\end{aligned}
$$

Double precision numbers:

s_m	$b_1 b_2 \dots b_{52}$	s_e	$e_1 e_2 \dots e_{10}$

The largest number is

$$\hat{x} = \frac{1}{2}\left(\frac{1-(\frac{1}{2})^{52}}{1-\frac{1}{2}}\right) = (1-2^{-52}) \times 2^{1023} \approx 9 \times 10^{308}$$

The smallest number is

$$\hat{x} = 2^{-52} \times 2^{-1023} = 2^{-1075} \approx 2 \times 10^{-324}$$

Example:

• Consider the floating point number system $F[10, 3, 2]$, corresponding to base 10. Nonzero numbers have the form $\pm 0.d_1 d_2 d_3 \times 10^{\pm e_1 e_2}$ where $d_1 \neq 0$.

a) find the largest and smallest positive normalized value that can be stored in F?

b) how many different nonzero numbers (positive and negative) can be stored in F?

Decimal Machine Numbers:

Binary digits do not show the computational difficulties that occure when a finite collection of machine numbers is used to represent the real numbers. To show this problem, we use decimal numbers which are more familiar.

Decimal machine numbers

Normalized decimal floating-point form is :

$$\pm 0.d_1 d_2 \cdots d_k \times 10^n, \qquad 1 \leq d_1 \leq 9 \quad \text{and} \quad 0 \leq d_i \leq 9 \qquad i = 2, 3, \cdots, k$$

Numbers of this form are called k-digit decimal machine numbers. Any positive real-number can be normalized to the form

$$y = 0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^n$$

The floating-point form of y, $fl(y)$ or y^*, is obtained by terminating the mantissa of y at k decimal digits. There are two ways of termination:

- chopping, just simply chop off the digits $d_k d_{k+1} \cdots$
- rounding

Decimal machine numbers

We usually can't find the exact value for the error. Instead, we find a bound for the error (mostly relative error). Relative error in converting a real number x to a floating point number $fl(x)$ is defined as $\delta_x = \dfrac{x - fl(x)}{x}$. To find the upper bound of δ_x, we introduce machine epsilon.

The machine epsilon, ϵ_{mach}, is the smallest number $\epsilon > 0$ such that $fl(1 + \epsilon) > 1$.

Machine epsilon

Proposition

The machine epsilon is given by:
(a) $\epsilon_{mach} = b^{1-m}$ if chopping is used.
(b) $\epsilon_{mach} = \frac{1}{2}b^{1-m}$ if rounding is used

By this definition we have the following theorem:

Theorem

For any floating point system F, under chopping

$$|\delta_x| = |\frac{x - fl(x)}{x}| \leq \epsilon_{mach}$$

Therefore,
under single precision: $|\delta_x| \leq 0.24 \times 10^{-6}$
under double precision: $|\delta_x| \leq 0.44 \times 10^{-15}$

Floating point operation

$\oplus$ denotes floating point addition:

$$a \oplus b = fl(fl(a) + fl(b))$$

As $\delta_x = \dfrac{x - fl(x)}{x}$, then $fl(x) = x(1 - \delta_x)$. Since $|\delta_x| \leq \epsilon_{mach}$ we have $fl(x) = x(1 + \eta), \quad |\eta| \leq \epsilon_{mach}$.

$$a \oplus b = fl(fl(a) + fl(b)) = (fl(a) + fl(b))(1 + \eta), \qquad |\eta| \leq \epsilon_{mach}$$

or

$$a \oplus b = (a(1 + \eta_1) + b(1 + \eta_2))(1 + \eta), \qquad |\eta_1|, |\eta_2|, |\eta| \leq \epsilon_{mach}$$

Note that we can define other operations in a same way.

Floating point operation

Example:

• Suppose that $x = \dfrac{5}{7}$ and $y = \dfrac{1}{3}$. Use five-digit chopping for calculating $x + y$, $x - y$, $x \times y$ and $x \div y$.

Vector Norms:

Definition

Suppose V is a vector space over $\mathbb{R}^n$. Then $\| \cdot \|$ is a vector norm on V if and only if $\| v \| \geq 0$, and

- $\| \vec{v} \| = 0$ if and only if $v = 0$.
- $\| \lambda \vec{v} \| = |\lambda| \, \| \vec{v} \| \quad \forall \vec{v} \in V$ and $\lambda \in \mathbb{R}$
- $\| \vec{u} + \vec{v} \| \leq \| \vec{u} \| + \| \vec{v} \|$(triangle inequality)

Vector norms

There are three standard vector norms known as the 2-norm, the 1-norm and the 1-norm.

Definition: The 2-norm over $\mathbb{R}^n$ is defined as

$$\| \vec{x} \|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Definition: The ∞-norm over $\mathbb{R}^n$ is defined as

$$\| \vec{x} \|_\infty = max(x_i) \qquad 1 \leq i \leq n$$

Definition: The 1-norm over $\mathbb{R}^n$ is defined as

$$\| \vec{x} \|_1 = \sum_{i=1}^{n} |x_i|$$

Theorem. Cauchy-Schwartz Inequality. Let $\| \cdot \|$ be a vector norm over a vector space V induced by an inner product. Then

$$|\vec{x} \cdot \vec{y}| \leq \| \vec{x} \| \| \vec{y} \|$$