CS371/AM242 Introduction to Computational Mathematics Winter 2014

Module 01

Starting Monday, January 6, 2014

Course Personnel

- Instructor: Lori Case (lori.case@uwaterloo.ca)
- Office Hours: Tuesdays, 2-3pm in DC3103 (starting next week) or by appointment

- Teaching Assistants:
- Arthur Cavalho
- Edward Cheung
- Aditya Tayal
  - Common email: cs371@student.cs.uwaterloo.ca
  - Office hours: tba

- Course Objectives: This course is intended as an introduction to the computational methods and issues encountered when solving realistic examples in scientific computation.

- Related Courses:
  - CS370 – similar topics, more emphasis on the applications (CS majors and non-majors)
  - CS335 – similar topics, focus on financial applications (CS non-majors only)
- Follow-up courses: CS473, 475, 476, and (for CS majors only) CS488

Course Grading

- Assignments 30%
- Midterm 30%
- Final 40%

Note: You must pass the weighted average of the midterm and final in order to pass the course.

Assignments

- Total of 4 assignments throughout the term
- Worth 7.5% each
- ***Must be completed individually***
- Due 4pm on the due date
- Submitted to MC Fourth floor assignment boxes
- Discussion Forum: LEARN
- Late Policy: May be passed in until following Friday noon, to my office, for a 40% penalty.

Midterm

- Wednesday, February 26 in MC2065
- 7:00-8:50 pm
- Time appears on your schedule
- No alternate arrangements (other than with AccessAbility Services)
- You are responsible for resolving any conflicts

MATLAB

- Programming language: MATrix LABoratory
- More like Python than Scheme (loops, etc)
- Will be used in class and on assignments
- Accessible through your undergrad account
- A tutorial session to be arranged.
- Additional information available at: http://ist.uwaterloo.ca/ew/saw/matlab/

Course Delivery and Reference Material

- Lecture material will primarily be presented through board work
- Online resource: Waterloo LEARN
- Supporting slides will be posted after class
- Supporting course notes (by Hans De Sterck) available in MC2018 and on LEARN. Required.
- Additional Reference: *Numerical Analysis* by Burden and Faires (library reserve tba)

Academic Integrity

In order to maintain a culture of academic integrity, members of the University of Waterloo community are expected to promote honesty, trust, fairness, respect and responsibility. Check the Office of Academic Integrity's website, www.uwaterloo.ca/academicintegrity for more information.

Avoiding Academic Offenses

Some students are unaware of the line between acceptable and unacceptable academic behaviour, especially when discussing assignments with classmates and using the work of other students. For information on commonly misunderstood academic offenses and how to avoid them, students should refer to the Faculty of Mathematics Cheating and Student Academic Discipline Policy, http://www.math.uwaterloo.ca/navigation/Current/cheating_policy.shtml

Students with Disabilities

- The AccessAbility Services Office (AS), located in Needles Hall, Room 1132, collaborates with all academic departments to arrange appropriate accommodations for students with disabilities without compromising the academic integrity of the curriculum. If you require academic accommodations to lessen the impact of your disability, please register with the AS at the beginning of each academic term.
- http://uwaterloo.ca/disability-services/

Course Topics*
(*order and details tentative)

- Floating point arithmetic (Chapter 1)
  - How are numbers stored on a computer? How does that affect numerical algorithms and solutions?

- Root finding (Chapter 2)
  - For a function *f*, find *x** satisfying *f(x*) = 0*
  - Example applications: Lots of applications in physics , chemistry, biology, engineering, etc., where the calculation of an unknown is required, e.g. Kepler's laws of planetary motion, ray tracing in graphics

Course Topics

- Linear Systems (Chapter 3)
  - For matrix A, vector c, find x* satisfying Ax*=c
  - Example applications: Many different areas, e.g. image processing, google page rank, circuit analysis
- Polynomial Interpolation (Chapter 5)
  - For a set of points, find a polynomial that "fits" the points
  - Example applications: graphics (splines), CAD, robotics

Course Topics

- Numerical Integration (Chapter 6)
  - Find a numerical approximation to $\int_a^b f(x)dx$
  - Example applications: robotics/GPS data(determining position from acceleration data)

- Discrete Fourier Methods (Chapter 4)
  - Fourier transforms time (or space) to/from frequency. Widely acknowledged as one of the most important numerical algorthms.
  - Example applications: signal processing, image processing, data compression (jpeg/mpeg)

The goal of computational mathematics …

- Find or develop 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

Given a problem …

- Consider the problem itself
  - May be very sensitive to small changes in data
  - Can we can 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?
- Determine how to represent the problem and algorithm numerically
  - Abstraction/model
  - How to represent numbers?

Consider this computation

```
>> sqrt(10^15) * sqrt(10^15)
ans =
   1.0000e+15

>> sqrt(10^15) * sqrt(10^15) - 10^15
ans =
   -0.1250
```

And this one

```
>> (1/5 + 1/5 + 1/5) * 5
ans =
    3.0000

>> (1/5 + 1/5 + 1/5) * 5 - 3
ans =
   4.4409e-16
```

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
  - What about computations that fall outside this range?

Representing 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

Computer arithmetic is *approximate* arithmetic

- Must store approximations of most real values

→Computer arithmetic is not always exact

- Errors are inherent in any computer solution.
- Questions:
  - How big can the representation errors be?
  - Will the errors propagate through our calculations or will they remain constant throughout?

Definitions: Types of Errors

Consider an exact value, *z*, and an approximation, $\hat{z}$,

- the absolute error is given by

$$\Delta_z = z \; - \hat{z}.$$

- the relative error (assuming *z≠0*) is given by

$$\delta_z = \frac{z-\hat{z}}{z} = \frac{\Delta z}{z}.$$

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: $\pm i_1 i_2 \dots i_I . f_1 f_2 \dots f_F$
- Question: How many values can be represented?

Floating Point Representation

- Allow the decimal point to "float"
- A floating point number system is represented by 3 values:
  - Base, *b*
  - *m,* number of digits in the ***normalized*** fractional part (mantissa)
  - *e*, number of digits in the maximum exponent
- Numbers are of the form in FL(*b,m,e*):

$$0 \qquad \text{or} \qquad \pm 0.x_1x_2\dots x_m \times b^{\pm y_1y_2\dots y_e}$$

where x_1 is 1,…, $b-1$, x_i is 0,1,…, b-1 for *i=2,..m*

Represent x = 12.34567890123 in FL(10, 4, 3)

1. Normalize: x = 0.123456789×10^2
2. Choose 4 digits:
   a) Chopping: 0.1234×10^2
      - Absolute error: $0.000056789 \times 10^2 = 0.56789 \times 10^{-2}$
   b) Rounding: 0.1235×10^3
      - Absolute error: $-0.000043211 \times 10^3 = -0.43211 \times 10^{-2}$

Note: Absolute error in rounding will never exceed chopping error

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

Standard Floating Point Representations

- Base 2 (binary)
- Specified by numbers of binary digits (bits)
- Single precision (4 bytes = 32 bits)

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

- Double precision (8 bytes = 64 bits)

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

- where s_m, s_e are sign bits (0 for positive, 1 for negative)
- What are largest and smallest positive values that can be stored?

Consider the floating point system FL(*b, m, e*), with chopping, where *fl(x)* is the representation of a value x:

- Machine epsilon ε_{mach} is the smallest positive value such that *fl(1+ ε)* > 1.
- *Prove:* $\varepsilon_{mach} = b^{1-m}$ ($\varepsilon_{mach} = \frac{1}{2} b^{1-m}$ *for rounding)*
- What is the smallest relative error in *fl(x)*?
- What is the largest relative error in *fl(x)*?
  - *Prove:* $|\delta_x| \leq \varepsilon_{mach}$
  - We will write $fl(x) = x(1+\eta)$, where $|\eta| \leq \varepsilon_{mach}$
- What accuracy can be we expect in standard single precision? double?

Floating point addition $\oplus$

- *Note: We often write fl(x) = x(1+ η),* where $|\eta| \leq \varepsilon_{mach}$
- Calculating $a \oplus b$
  - fl(a)
  - fl(b)
  - Sum the two values -> s
  - fl(s)
- Consider bound on $a \oplus b$

- Does $(a \oplus b) \oplus c = a \oplus (b \oplus c)$?

Numerical complications with calculations

- Catastrophic cancellation – subtracting two numbers of nearly equal value. The result is a small value, often with a major loss of significant digits.
- Overflow – result of a calculation is too large to be represented in the computer's number system

  (See http://en.wikipedia.org/wiki/Ariane_5)
- Underflow – result of a calculation is too small to be represented as a number distinct from 0
- These errors may propagate through other calculations or may result in a program crash

Aside: 1994 Intel Pentium Chip Debacle

- Used a lookup table to speed up calculations
- Oops: 1066 entries in table, 5 missing
- Result: Loss of accuracy in mathematically intense calculations. Would have little affect for most users.
- Math is fun! Prof working on a prime number problem identified error. Gained media attention.
- Intel's solution: we will replace, only if you can prove it is a problem for you.
- The public did not react well.
- Eventually, Intel offered to replace all, says it cost them $475 million!

Consider the problem: $y = x/(1-x)$

- Suppose $x = 0.93 \rightarrow y = 13\,{}^{2}/_{7}$
- Consider $\widehat{x} = 0.94,\ i.e.\,\Delta_x = 0.01$
- Relative error in x: $1/93 \approx 0.01075\ldots$
- But, $\hat{y} = \hat{x}/(1-\hat{x}) = 15\,{}^{2}/_{3}$
- Relative error in y: $50/279 \approx 0.1792\ldots$
- Small change in x produced a much bigger change in y.
- The issue here is with the problem, not with a numerical solution of this problem.

Condition of a mathematical problem P

- Suppose problem P with input $\tilde{x}$ requires computation of an output z = $f_P(\tilde{x})$.
- P is ***well-conditioned*** if small changes $\Delta_{\vec{x}}$ in $\vec{x}$ result in small changes $\Delta_{\vec{z}}$ in $\vec{z}$.
- P is ***ill-conditioned*** if small changes $\Delta_{\vec{x}}$ in $\vec{x}$ result in large changes $\Delta_{\vec{z}}$ in $\vec{z}$.
- Condition number of a problem P, with respect to absolute error, is $\kappa_A = \|\Delta_{\vec{z}}\| / \|\Delta_{\vec{x}}\|$
- Condition number of a problem P, with respect to relative error, is $\kappa_R = \frac{\|\Delta_{\vec{z}}\| / \|\vec{z}\|}{\|\Delta_{\vec{x}}\| / \|\vec{x}\|}$

(Brief) Background: Vector Norms

- Vector space over $\mathfrak{R}^n$
- The 2-norm is defined as

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

- The 1-norm is defined as

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

- Both satisfy the Cauchy-Schwartz Inequality:

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

Consider solving: $x^2+62.1x+1=0$

Calculate the roots using 4 digits of accuracy

- $x_1 = \frac{-62.1+\sqrt{(62.1)^2-4}}{2} = -0.02000$
- $x_2 = \frac{-62.1-\sqrt{(62.1)^2-4}}{2} = -62.08$

Alternate formulas:

- $x_1 = \frac{-2}{62.1+\sqrt{(62.1)^2-4}} = -0.01610$
- $x_2 = \frac{2}{-62.1+\sqrt{(62.1)^2-42}} = -50$

Which solution is correct? Which algorithm is better?

Stability of an algorithm

- Suppose problem P with input $\tilde{x}$ requires computation of an output z = $f_P(\tilde{x})$ is well-conditioned.
- An algorithm A to solve P is unstable if
  - small changes in x produce large changes in the solution, or if
  - Representation errors or round-off errors at one step causes much greater error as the algorithm progresses
- If an algorithm does not have this property, it is stable.
- Note: in the absence of computational errors, an unstable algorithm may solve problem P.
- Quadratic formula (in either form) was unstable for previous problem

Problem: Approximate e^x

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

Use this formula (Algorithm A) to approximate $e^{-5.5}$.

The approximate value is 0.0040868…

Using 5 significant digits in the calculation.

i	Ith term in series	Ith truncated term
0	1.0000000000000	1.0000000000000
1	-5.5000000000000	-4.5000000000000
2	15.1250000000000	10.6250000000000
3	-27.7300000000000	-17.1050000000000
…	…	…
20	0.0002637100000	0.0058115000000
21	-0.0000690670000	0.0057424000000
22	0.0000172670000	0.0057597000000
23	-0.0000041289000	0.0057556000000
24	0.0000009462300	0.0057565000000
25	-0.0000002081700	0.0057563000000
26	0.0000000440350	0.0057563000000

Stop the calculations – sum did not change after this term was added

Using 5-digit calculations, our answer was 0.0057563, which has no correct digits.

Relative error ~ -0.41

Why?

- Repeated substractions of values of similar magnitude resulted in a loss of significant digits, which increased with the calculations.
- Note: the values were close – but not ***that*** close
- If exact arithmetic was used in the summation, the relative error would be ~ 8.9e-6
- The problem was not the process, but the computations

Alternate approach (algorithm B)

- Try a different approach which adds rather than subtracts (still keep to the first 25 terms)

$$e^{-x} \sim \left(\sum_{i=0}^{24} \frac{(-x)^i}{i!}\right)^{-1} = \frac{1}{1+x+\frac{x^2}{2!}+\cdots+\frac{x^{24}}{24!}}$$

This yields 0.040865 using 5 significant digits.

Relative error is 6.6e-5.

More on: $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$

- The equality depends on an infinite sum
- The summation was truncated – another source of error
- In this case, however, the catastrophic cancellation was a greater source of error, and the errors "blew up" as the calculations progressed.

Stable/Unstable Algorithms

- Algorithm A was unstable for $x<0$, and algorithm B was stable.
- However, if calculating for $x>0$, Algorithm A is stable and algorithm B is unstable.
- Algorithm stability may depend on the data.

Possible sources of errors

- Errors in constructing the mathematical models
- Representation errors
- Round-off errors in calculations
- Algorithm truncation errors
- Ill-conditioned problem
- Unstable algorithm
- Human errors (e.g. measurement errors, coding errors, etc.)