Lecture 1

CSC 338: Numerical Methods

Ray Wu

University of Toronto

January 11, 2023

About this course

- Numerical Methods (aka Numerical Analysis, Scientific Computing) has a long history, predating modern digital computers.
- Many of the algorithms we will learn in this class were derived on pen and paper.
- The goal of this course is not to just focus on how the algorithms work, but also to emphasize the requirements on the input, interpreting the program output, etc.

Teaching Team

- **Instructor**: Ray Wu (myself)
  - Office Location: DH-3097D
  - Office hours: Wednesday 13-14, after class
- **Tutorial TAs**:
  - Akira Takaki (Wednesday 10-11, MN 2100)
  - Harshit Gupta (Thursday 11-12, DH 4001)
- **other TA**: Mohan Zhao

The first lecture

- Logistics (syllabus)
- Intro to numerical methods
- Intro to floating point systems

Overview of course

- We will have nine topics.
  - Intro & Floating-point systems
  - One-variable nonlinear equations
  - Systems of linear equations
  - Least squares
  - Eigenvalues
  - Systems of nonlinear equations and optimization
  - Function interpolation
  - Numerical integration
  - Iterative methods for systems of linear equations
- In general, 2 lectures per assignment (except the last assignment).
- Not necessarily one topic per lecture. If we end early, use time for review or more topics (none of which will be tested).

Logistics

- Course Website: `http://www.cs.toronto.edu/~rwu/csc338/`
  - Lecture Slides and Assignments posted here.
- Piazza discussion board: `https://piazza.com/class/lch0r4yvhw76xn`
  - If you have questions about the course content
  - For **private questions**, post a private note on Piazza or e-mail me.
  - When emailing, use [CSC338] in the header to ensure your email gets read.
- Handin assignments using MarkUs. We will ask for both writeup and code.
- In general, we are not marking your code, but we will be emphasizing on the interpretation of your code's output.

Grading

- Five assignments: 50% total, equally weighted.
- Midterm: 10%. On first two assignments.
- Final exam: 40%, cumulative, scheduled by the university.
  - Must receive $\geq$40% on final to pass course.
  - Do not make travel plans before the final exam schedule is released! I can't give you an earlier exam time.
- Everything is to be done individually
- If you miss the midterm, the weight will go to the final exam.
- If you score better on the final, it will replace your midterm grade.

Assignments

- You are free to use either Matlab or Python.
  - The assignments are probably easier to complete in matlab, simply because matlab is designed more for numerical computation.
- A combination of maths (derivations) and programming.
- Start the assignments early. Ask for help on piazza if you get stuck.
- You are not allowed to import any package(s) that will trivialize the assignment question(s).
  - If you are unsure, ask.
  - If you use any such packages, you will probably get a zero.

Asking for help

These are the places you can get help:

- Piazza
- Instructor office-hours
- Tutorials
- Your classmates (make friends with your neighbour)
- Textbooks
- The general web (most topics are covered in many places, e.g. Wikipedia)

Do not pass off work that you claim to be your own, and do not allow yourself to be copied off from. Both are equally serious academic offenses (AO), and UofT takes AOs **very** seriously.

Midterm and Final

- You will likely need a calculator on the final, but not the midterm.
- You are permitted one on both exams anyways (scientific, non-programmable)
- You are allowed one sheet of notes, double-sided, either typed or handwritten is fine.

Prerequisites

- Calculus up to integral calculus (Calc 2).
  - We will use some Calc 3 for derivation of multi-dimensional algorithms, but they will be covered as they come up.
- Introductory linear algebra (you should know what matrices and vectors are, how multiplication between them works, solving linear equations, etc)
- Introductory programming
- Basic algorithm analysis (Big-$\mathcal{O}$).

History of Numerical Methods

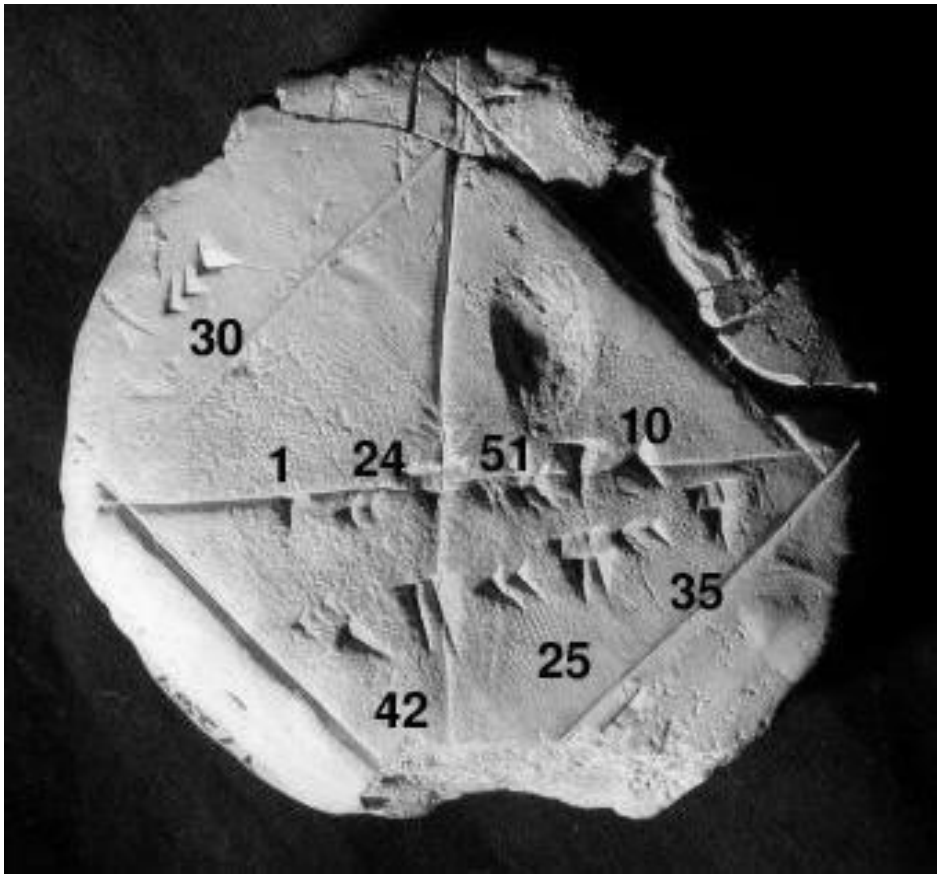

Figure 1: Babylonian clay tablet c. 1800-1600 BCE. approximating $\sqrt{2}$ to 6 decimal digits, the greatest known computational accuracy in the ancient world.

Numerical Methods

In applied mathematics, we frequently use continuous variables. For example, the temperature in this room or a person's height can be measured to arbitrary precision, hence, a continuous variable.

- This causes difficulties in computation, because we have continuous models and a discrete computing environment.
  - errors are inevitable.
- Approach the study of numerical methods with the study of errors.
  - When you know the error, you know a lot about the computed solution.

Absolute vs Relative Error

- If a scalar quantity u' approximates u, then
- The absolute error is given by

$$|u - u'| \tag{1}$$

- The relative error is given by

$$\frac{|u - u'|}{|u|} \tag{2}$$

  If $|u| = 0$, then the relative error is not defined.
- Generalizable to anything that has norms – we'll get into norms later in the course.

Types of error

- Model error
- Truncation error
- Discretization error
- Roundoff error
  - Catastrophic cancellation

Example: Model error

What is the volume of the earth? Model the earth as a sphere:

$$V = \frac{4}{3}\pi r^3 \tag{3}$$

What is the (obvious) problem that arises?

- Earth is not a perfect sphere – it is wider around the equator due to centrifugal forces. (assumption doesn't hold).
- Not the emphasis/scope of the course, **but important for industry/practical applications**

Even if you compute the model perfectly, it will be limited by any simplifying assumptions that you have made.

Impacts of model error

- Prior to 2008, financial derivatives (contracts for buying/selling assets such as stocks, at predetermined prices) did not take into account default risk – that is, the risk that either the buyer or seller would default (not honor the contract).
- This contributed in part to excessive lending, which in part ultimately led to the financial crisis of 2008.
- Subsequently, mathematical models took credit risk into account.

Truncation Error

Computing the constant $e \approx 2.718...$ with the infinite series

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} \tag{4}$$

- Cannot sum the series forever.
- Have to end somewhere.

First n terms	Error
4	5.16e-02
5	9.95e-03
6	1.62e-03
7	2.26e-04
8	2.79e-05

Table 1: Table of errors for $n = 4, 5, 6, 7, 8$ terms in the series.

Discretization Error

Consider computing the derivative of $f(x) = \sin(x)$ at $x = x_0 = 0.4$ with finite differences:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} \tag{5}$$

h	Error
1	3.25e-01
1/2	1.33e-01
1/4	5.80e-02
1/8	2.67e-02
1/16	1.28e-02

Table 2: Table of errors of computing derivative of $\sin(x)$ at $x_0 = 0.4$

- Errors decrease **predictably**: the error is halved when h is halved
- Error is $\mathcal{O}(h)$.

Taylor series

- We will use Taylor series often in this course:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \dots \tag{6}$$

- Used to derive the approximation on the previous slide, by isolating the term $f'(x)$ and dropping higher order terms.
- Generally we take h to be small, so the higher order terms in h are even smaller and can be assumed to be approximately zero.

Order notation

- You may have seen already seen big-O notation: e.g. a sorting algorithm on an array of size n can be $\mathcal{O}(n \log n)$.
- We can think of them as an ordering:

$$\mathcal{O}(n^2) > \mathcal{O}(n \log n) > \mathcal{O}(n) > \mathcal{O}(\log n) \tag{7}$$

- We use big-O notations in two ways in numerical methods: the above way and use to describe the error as a function of a small parameter, typically denoted as h.
- h is generally chosen to be *much* smaller than 1. Hence, the ordering is reversed:

$$\mathcal{O}(h) > \mathcal{O}(h^2) > \mathcal{O}(h^3) > \mathcal{O}(h^4) \tag{8}$$

- In both cases, it's more desirable for the algorithm to be classified closer to the right side (smaller).

Roundoff error

Easiest to illustrate with a code snippet:

```
>> x = 0.1
>> y = 0.2
>> z = 0.3
>> z-y-x
-2.7755575615628914e-17
```

- not zero
- **unpredictable**
- numbers not represented exactly on computer
- focus of next section

Floating point systems

Representation of numbers in scientific notation:

$$-1.2345 \times 10^{-14}$$

- sign: $-$
- mantissa: 1.2345
- base: 10
- exponent: -14

normalized representation: $1 \leq \text{mantissa} < \text{base}$

General floating point representation

Defined by (β, t, L, U)

- β: base
- t: precision; number of digits in mantissa
- L, U: lower and upper bounds on exponent.

Chopping vs Rounding

Suppose precision $t = 5$, then to represent the number 1.23456 in floating point:

- **Chopping**: $\text{fl}(x) = 1.2345$
- **Rounding**: $\text{fl}(x) = 1.2346$ (5 goes up, 4 goes down in decimal)

Error bounds (maximum possible error):

- **Chopping**: $\beta^{1-t}\beta^{e}$
- **Rounding**: $\frac{1}{2}\beta^{1-t}\beta^{e}$.

Relative errors: drop the β^{e}.
Rounding unit:

$$\eta = \frac{1}{2}\beta^{1-t} \tag{9}$$

bound on relative error of floating point calculations

Floating point computations

Addition, subtraction, multiplication, division: Using guard digits (extra digits), exact rounding is achieved. Then if a and b are floating point numbers:

$$\mathrm{fl}(a \pm b) = (a \pm b)(1 + \epsilon_1) \tag{10}$$

$$\mathrm{fl}(a \times b) = (a \times b)(1 + \epsilon_2) \tag{11}$$

$$\mathrm{fl}(a/b) = (a/b)(1 + \epsilon_3) \tag{12}$$

where $|\epsilon_i| \leq \eta$.

Algorithm Properties & Problem conditioning

Algorithm properties:

- Accuracy – what are the bounds on the error
- Efficiency – how fast the algorithm runs
- Robustness – does it handle edge cases well, etc.
- Stability – if we change the input a little bit, how much does it affect the output?

Problem conditioning:

- Ill-conditioned if small change to data gives a large change in the result
- Well-conditioned otherwise.
- Problem conditioning is independent of any algorithm or floating point system.

Condition Number (κ)

- The condition number (κ) is a quantification of the problem conditioning.
- The condition number is always positive and measures how much the output changes for a small change in the input.
- In general, the condition number is measured on a logarithmic scale (i.e. an increase from 1 to 10 is the same significance as an increase from 10 to 100).
- If the condition number κ is 10^k, you may lose up to k digits of accuracy in your computations.
- Next slide we will derive condition number for one-variable functions

Condition Number of a function

- Consider the problem of evaluating a function $f(x)$.
- A small change in the input is Δx
- The relative change in the input is

$$\frac{\Delta x}{x} \tag{13}$$

- The relative change in the output is

$$\frac{f(x+\Delta x)}{f(x)} \tag{14}$$

- The condition number is defined by

$$\frac{\text{relative change in output}}{\text{relative change in input}} \tag{15}$$

Condition Number of a function, part II

- Substituting in the relative change in output and input, we get

$$\kappa(f) = \frac{\frac{f(x+\Delta x)}{f(x)}}{\frac{\Delta x}{x}} = \frac{f(x+\Delta x)}{f(x)} \cdot \frac{x}{\Delta x} \tag{16}$$

- Rearrange the fraction:

$$\kappa(f) = \frac{f(x+\Delta x)}{\Delta x} \cdot \frac{x}{f(x)} \tag{17}$$

- Take the limit as $\Delta x \to 0$, and take absolute value to ensure positivity:

$$\kappa(f) = f'(x) \cdot \frac{x}{f(x)} = \left| \frac{xf'(x)}{f(x)} \right| \tag{18}$$

Overflow, underflow, NaN

- overflow: when exponent is too large, value stored as $\pm\infty$.
  - fatal: computation cannot meaningfully continue.
- underflow: when exponent is too small, value stored as zero.
  - non-fatal: if added to another number, computation can continue.
- not-a-number (NaN): 0/0.
  - also fatal

IEEE standard (double precision)

- The standard that almost everyone uses in computing.
- binary notation
- 64 bits split between:
  - 1 sign bit
  - 11 exponent bits
  - 53 fraction bits.
- Extra bit (1+11+53): first digit stored implicitly, since it must be one in normalized scientific notation.
- machine epsilon: $\frac{1}{2}2^{1-53} = 2^{-53} \approx 1.11 \times 10^{-16}$
- You should remember that machine epsilon is around 10^{-16} in double precision.

IEEE single precision

- Rarely used these days; artefact from early days of computing.
- binary notation
- 32 bits split between:
  - 1 sign bit
  - 8 exponent bits
  - 24 fraction bits.
- implicit bit present also
- machine epsilon: $\frac{1}{2}2^{1-24} = 2^{-24} \approx 5.96 \times 10^{-8}$
- We may use it for instructional purposes on an assignment.

Floating point pitfalls – example 1

Consider computing the following expression, for large x:

$$y = \sqrt{x+1} - \sqrt{x} \tag{19}$$

```
>> x = 1e16;
>> y = sqrt(x+1) - sqrt(x)
0.0
```

- computation clearly not correct
- subtracting two large numbers of similar size causes large relative error

Floating point pitfalls – example 1

Instead compute the following equivalent expression:

$$y = \left(\frac{\sqrt{x+1}+\sqrt{x}}{\sqrt{x+1}+\sqrt{x}}\right)(\sqrt{x+1}-\sqrt{x}) = \frac{1}{\sqrt{x+1}+\sqrt{x}} \tag{20}$$

```
>> y = 1/(sqrt(x+1) + sqrt(x))
5e-09
```

- This is known as the conjugation trick
- correct computational result.

Floating point pitfalls – example 2

The *softplus* function is defined as

$$\ln(1 + \exp(x)) \tag{21}$$

It is intended to be an approximation to the ReLU function $\max(x, 0)$.

```
>> log(1.0 + exp(700))
700.0
>> log(1.0 + exp(750))
Inf
```

What happened?

Floating point pitfalls – example 2

Computing exp(750) caused overflow.
Alternative formula:

$$\begin{aligned}\ln(1+\exp(x)) &= \ln\left(\frac{1+\exp(x)}{\exp(x)}\exp(x)\right)\\ &= \ln(\exp(x)) + \ln\left(\frac{1+\exp(x)}{\exp(x)}\right)\\ &= x + \ln(1+\exp(-x))\end{aligned}$$

Computation:

```
>> 750 + log(1 + exp(-750))
750.0
```

Error accumulation

- In general, we cannot avoid linear roundoff error accumulation:

$$E_n \approx c_0 n E_0 \tag{22}$$

- An exponential error growth is usually problematic:

$$E_n \approx c_1^n E_0 \tag{23}$$

for some $c_1 > 1$.