

# AMSC460: Computational Methods

JAMES ZHANG<sup>\*</sup>

September 10, 2024

These are my notes for UMD's CMSC460: Computational Methods. These notes are taken live in class ("live- $\text{\TeX}$ -ed). This course is taught by Professor Haizhao Yang. The textbook for the course is *A First Course in Numerical Methods* by OU Ascher and Chen Greif.

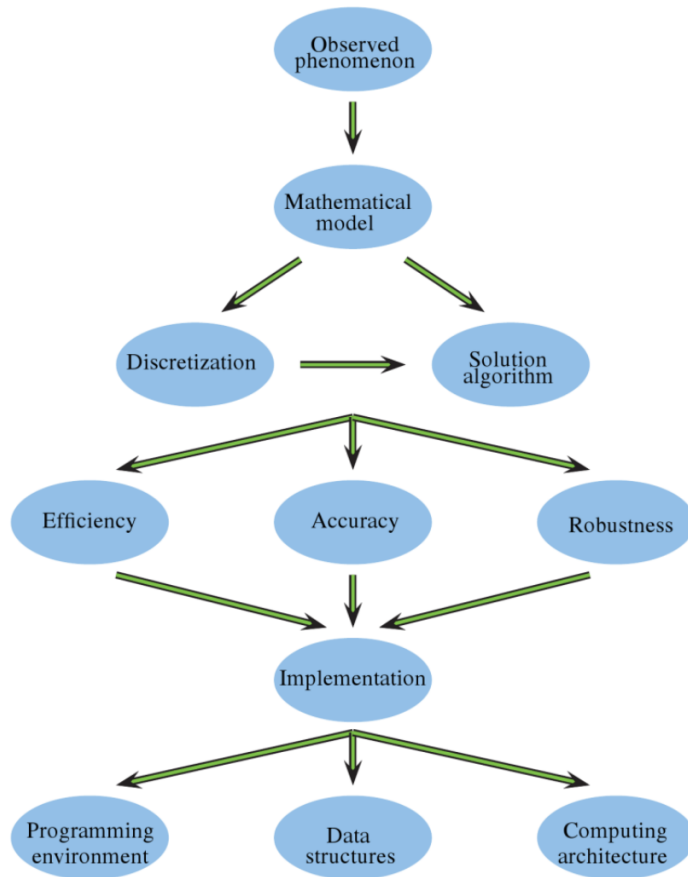
## Contents

<b>1</b>	<b>Scientific Computing</b>	<b>2</b>
1.1	Numerical Algorithms and Errors . . . . .	3
1.2	Algorithm Properties . . . . .	5
1.3	Binary Representation, Rounding Errors, Truncation Errors . . . . .	5
<b>2</b>	<b>Roundoff Errors</b>	<b>5</b>
2.1	Floating Point Systems . . . . .	6
2.2	Errors in Computation . . . . .	7
<b>3</b>	<b>Nonlinear Equations in One Variable</b>	<b>8</b>
3.1	Iterative Methods to Find Roots . . . . .	8
3.2	Bisection Method . . . . .	9
3.3	Fixed Point Iteration . . . . .	9

---

<sup>\*</sup>Email: [jzhang72@terpmail.umd.edu](mailto:jzhang72@terpmail.umd.edu)

## §1 Scientific Computing



**Figure 1.1.** *Scientific computing.*

**Definition 1.1** (Relative and Absolute Errors). Let the target value be  $u \in \mathbb{R}$  and let the numerical solution be  $v \in \mathbb{R}$ , then the **absolute error** is  $|u - v|$  and the **relative error** is  $\frac{|u-v|}{|u|}$ .

**Note 1.2.** If  $|u|$  is large, then relative error is used and if  $|u|$  is very small, then relative error is not a good measurement.

**Example 1.3** (The Stirling Approximation)

The formula  $u = S_n = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$  is used to approximate  $n!$ .

```

1  e = exp(1);
2  n = 1 : 10;
3  # Note that the following are vectors of len 10.
4  S_n = sqrt(2*pi*n).*((n/e).^n);
5  # Compute absolute and relative err.
6  fact_n = factorial(n);
7  abs_err = abs(fact_n - S_n);
8  rel_err = abs_err./abs(fact_n);
9  format short g
10 [n; fact_g; abs_err; real_err;]'
```

**§1.1 Numerical Algorithms and Errors**

**Definition 1.4** (Error Types).

1. Error in the problem to be solved. These may be errors in the mathematical model or errors in the input data.
2. Approximation errors, which can consist of discretization errors (errors in interpolation, differentiation, integration) or convergence errors, which can also arrive in iterative methods
3. Roundoff errors

**Definition 1.5** (Taylor Series). Assume that  $f(x)$  has  $k+1$  derivatives in an interval containing the point  $x_0$  and  $x_0 + k$ . Then

$$f(x_0 + k) = f(x_0) + hf'(x_0) + \frac{h^2}{2} + \cdots + \frac{h^k}{k!} f^{(k)}(x_0) + \frac{h^{k+1}}{(k+1)!} f^{(k+1)}(\xi)$$

where  $\xi$  is some point between  $x_0$  and  $x_0 + h$ , and the term  $\frac{h^{k+1}}{(k+1)!} f^{(k+1)}(\xi)$  is the remainder term.

**Note 1.6.** To find  $f'(x_0)$  observe that

$$f'(x_0) = \frac{f(x_0 + k) - f(x_0)}{h}$$

and then if we take the limit of this

$$f'(x_0) = \lim_{h \rightarrow 0} \frac{f(x_0 + k) - f(x_0)}{h}$$

we recover the  $h$ -definition of a derivative, and the discretization error is  $\frac{h}{2} f''(\xi)$  because our model is

$$hf'(x_0) = f(x_0 + k) - f(x_0) - \frac{h^2}{2} f''(x_0) + \cdots$$

$$\begin{aligned} \Rightarrow f'(x_0) &= \frac{f(x_0 + k) - f(x_0)}{h} - \left( \frac{h}{2} f''(x_0) + \dots \right) \\ \Rightarrow \left| f'(x_0) - \frac{f(x_0 + k) - f(x_0)}{h} \right| &= \left| \frac{h}{2} f''(x_0) + \dots \right| \end{aligned}$$

**Definition 1.7** (Big- $\mathcal{O}$  and  $\Theta$  Notation). We define these for error characterization in terms of some parameters.

$$\begin{cases} h \text{ represents a small parameter} \\ n \text{ represents a large parameter} \end{cases}$$

**Note 1.8.** An error  $e$  depending on  $h$  we denote  $e = \mathcal{O}(h^q)$  and if there are two positive constants  $q$  and  $C$  such that

$$|e| \leq Ch^q$$

Similarly, we write  $e = \Theta(h^q)$  if  $\exists C_1, C_2$  and  $q > 0$  such that

$$C_1 h^q \leq |e| \leq C_2 h^q$$

$n$  represents the problem size and then we use big  $\mathcal{O}$  and  $\Theta$  to denote the time or memory complexity of an algorithm.

### Example 1.9

If we say  $T = \Theta(n \log n)$  then we find  $C_1, C_2, x_0$  such that

$$C_1 n \log n \leq T \leq C_2 n \log n \quad \forall x \geq x_0$$

**Note 1.10.** Note that errors go down and then back up as  $h$  changes. A small number divided by another small number is a dangerous, think about exploding and vanishing gradients when training neural networks.

$h$	Absolute error	$h$	Absolute error
		1.e-8	4.361050e-10
0.1	4.716676e-2	1.e-9	5.594726e-8
0.01	4.666196e-3	1.e-10	1.669696e-7
0.001	4.660799e-4	1.e-11	7.938531e-6
1.e-4	4.660256e-5	1.e-13	4.250484e-4
1.e-7	4.619326e-8	1.e-15	8.173146e-2
		1.e-16	3.623578e-1

**Note 1.11.** In practice, error is the sum of **discretization error** + **rounding error**

## §1.2 Algorithm Properties

Some good assessments of the quality of an algorithm are accuracy, efficiency, and robustness

**Definition 1.12** (Accumulated Error). Suppose you're evaluating polynomial with large degree. Your error  $e_1$  from  $p_1$  gets compounded by  $e_2$  from  $p_2$  and so on and so forth, so the total error follows

$$\text{Total error} \leq |e_1| + \cdots + |e_n|$$

## §1.3 Binary Representation, Rounding Errors, Truncation Errors

**Remark 1.13.** Math claim: Any real number  $x \in \mathbb{R}$  is accurately representable by an infinite sequence of digits, eg.  $x \approx \pm 1$

$$x = \pm c(1.\{d_1\} \dots \{d_{t+1}\} \dots) \times 2^e$$

where  $d_1, d_2, \dots$  are integer numbers 0 or 1, and  $e$  is the integer exponent. For each  $e$ , you can find a sequence to represent this real number  $x$ .

### Example 1.14

Let  $x = -(1.10110\dots) \times 2^1$  which means  $x = -1 + 1 \times \frac{1}{2} + 0 \times \frac{1}{2^2} + 1 \times \frac{1}{2^3} + \dots$  where the  $\frac{1}{2^t}$  term, we denote as  $\frac{1}{s^t}$

**Definition 1.15** (Truncating). Chopping ignores digits  $d_t, d_{t+1}, \dots$  yielding  $\tilde{x} = \pm(1.\{d_1\} \dots \{d_{t-1}\}) \times 2^e \approx x$  and so the error is  $\mathcal{O}(\frac{1}{2^{te}}) = \mathcal{O}(\frac{1}{s^t})$

## §2 Roundoff Errors

**Definition 2.1** (Binary Floating Point Representations). Infinite sequence representing  $x \in \mathbb{R}$

$$x = \pm C(1.d_1d_2d_3\dots) \times 2^e$$

where  $e$  is an integer chosen to make  $d_0$  not 0. The  $1.d_1d_2\dots$  is known as the **mantissa**.

For example:  $x = 2^{12} \implies e = 12, x = (1.00\dots) \times e^{12}$

**Note 2.2.** Note that  $d_0 = 1$  always in binary representations because we have restrictions that  $d_0 \neq 1$ , which we will see later.

**Note 2.3** (Geometric Understanding). A real number  $\implies$  infinite sequence. For binary, if  $x \in (1, 1.5)$  then choose  $d_1 = 0$ . If  $x \in (1.25, 1.5)$ , choose  $d_1 = 1$ . For all  $x \in \mathbb{R}$ , you can always find  $2^e$  such that  $\frac{x}{2^e} \in [1, 2]$ .

For example:  $x = 1 + 2^{12} \implies 2^e = 2^{12} \implies \frac{1+2^{12}}{2^{12}} \in [1, 2]$  and  $x = 2^{-5} + 2^{-4} \implies 2^e = 2^{-4} \implies \frac{2^{-5}+2^{-4}}{2^{-4}} \in [1, 2]$

**Note 2.4.** To get a real number from the infinite sequence, use the equation

$$(1.d_1d_2\cdots) \times 2^e = \left(1 + \frac{d_1}{2} + \frac{d_2}{4} + \cdots\right) \times 2^e$$

**Note 2.5.** A general floating point representation is denoted as

$$fl(x) = \text{sign}(x) \times (1.\hat{d}_1\hat{d}_2\cdots) \times 2^e$$

where  $\hat{d}_i$  is determined by  $d_i$ . Because of truncation, we have errors and this is called rounding errors or machine accuracy.

## §2.1 Floating Point Systems

**Definition 2.6.** A floating point system can be characterized by a 4-tuple  $(\beta, t, L, U)$  where

- $\beta$  is the base of the number system
- $t$  = precision, or the number of digits
- $L$  is the lower bound on the exponent  $e$
- $U$  is the upper bound on the exponent  $e$

This definition leads to a generalization of floating point representations

$$fl(x) = \text{sign } x \left( \frac{\tilde{d}_0}{\beta_0} + \frac{\tilde{d}_1}{\beta_1} + \cdots \right) \times \beta^e$$

where  $0 \leq \tilde{d}_i < \beta - 1$  and  $e$  is chosen such that  $\tilde{d}_0 \neq 0$ .

**Definition 2.7** (Chopping). Ignore  $d_t, \dots$  yielding  $\tilde{d}_i = d_i$  and  $fl(x) = \pm(d_0.d_1d_2\cdots d_{t-1}) \times \beta^e$

**Definition 2.8** (Rounding). Consult  $d_t$  to determine the approximation

$$fl(x) = \begin{cases} \pm(d_0.d_1d_2d_3\cdots d_{t-1}) \times \beta^1 & \text{if } d_t < \frac{\beta}{2} \\ \pm(d_0.d_1d_2d_3\cdots d_{t-1} + \beta^{1-t}) \times \beta^e & \text{otherwise} \end{cases}$$

Essentially, we consider what we want to keep and discard. Note that the next term would be  $\frac{d_t}{\beta_t} \times \beta^e$

**Example 2.9**

Consider  $x = \frac{8}{3} = 2.66\cdots \in \mathbb{R}$

$$x = \left( \frac{2}{10^0} + \frac{6}{10^1} + \cdots \right) \times 10^0$$

$\beta = 10, e = 0$  so that  $d_0 \neq 0$

- We can chop this so that  $t = 4$  in which case

$$fl(x) = \left( \frac{2}{10^0} + \frac{6}{10^1} + \frac{2}{10^2} + \frac{6}{10^3} \right) \times 10^0 = 2.666$$

- Or we can round this so that  $t = 4$  in which case since  $d_t = 6 > \frac{\beta}{2} = 5$  we have

$$fl(x) = \left( \frac{2}{10^0} + \frac{6}{10^1} + \frac{2}{10^2} + \frac{6}{10^3} + 10^{1-4} \right) \times 10^0 = 2.667$$

**Theorem 2.10** (Floating Point Representation Error)

Absolute error

$$|x - fl(x)| \leq \begin{cases} \beta^{1-t} \cdot \beta^1 & \text{for chopping} \\ \frac{1}{2}\beta^{1-t} \cdot \beta^1 & \text{for rounding} \end{cases}$$

and relative error

$$\frac{|x - fl(x)|}{|x|} \leq \begin{cases} \beta^{1-t} & \text{for chopping} \\ \frac{1}{2}\beta^{1-t} & \text{for rounding} \end{cases}$$

**§2.2 Errors in Computation**

**Note 2.11.**

1.  $x + y$  may have large absolute error if  $x \gg y$
2. If  $|y| \ll 1$ , then  $\frac{x}{y}$  may have large relative and absolute error
3.  $xy$  is dangerous if  $y$  is large
4. If  $x \approx y$  then  $x - y$  has large relative error

$$\text{relative error} = \frac{|v - (x - y)|}{|x - y|}$$

due to the denominator being small

**Definition 2.12** (Overflow and underflow). An **overflow** means a number is too large to fit into the floating point system  $\implies e > U$ . an **underflow** means a number has  $e < L$ , where  $L$  is a large negative number.

Example:  $L = -14, e = -15, \beta = 10, \beta^e = 10^{-15}$  is what you want, but  $\beta^L = 10^{-14}$  is the smallest case the computer can compute. So, the computer will round it at zero.

## §3 Nonlinear Equations in One Variable

**Definition 3.1** (Question 1 in this Section). Given a nonlinear function  $f(x)$  and an interval  $[a, b]$ , find a root denoted as  $x^*$  if it exists on  $[a, b] \implies \text{find } f(x^*) = 0$ . On Matlab, you can use the **fzero** function

**Note 3.2.** Discontinuous functions may not have a root. Similarly,  $e^x > 0 \forall x$ , even though this function is continuous.

**Note 3.3** (Intermediate Value Theorem). If  $f(u)$  and  $f(v)$  change signs, then by the Intermediate Value Theorem, we have least one root on  $[a, b]$  if  $f(x)$  is continuous.

**Definition 3.4** (Question 2 in this Section). Do we have a unique root?

Example: Linear equations in most cases have one root, whereas nonlinear equations may have multiple roots. If we have multiple roots, how can we design an algorithm to find the desired root?

### §3.1 Iterative Methods to Find Roots

**Definition 3.5.** We have a sequential decision-making process to produce

$$X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow \cdots \rightarrow X_n \cdots$$

until we have found a sufficiently good  $X_n$ . Some things we need to think about.

- How do we initialize  $X_0$ ?

Changing signs to find  $[a, b]$ , if  $a = b$  then any point in  $[a, b]$  is good? You can also draw the function to get some good  $x_0$

- How to move from  $X_n$  to  $X_{n+1}$
- When do we stop?

1. If  $|f(X_n)| < \text{ftol}$
2. If  $|X_n - X_{n-1}| < \text{atol} \approx 0$
3. If  $|X_n - X_{n-1}| < \text{rtol} \approx 0$
4. If  $n > \text{max iterations}$  then quit.

**Note 3.6** (Desired Properties of an Algorithm).

1. Efficiency (small number of iterations, cheap computation per iteration, cheap memory per iteration)



2. Robustness
3. Minimum requirement of  $f(x)$
4. Generalizable to other questions

### §3.2 Bisection Method

**Definition 3.7** (Bisection Method). Suppose  $f(x)$  changes sign on an interval  $[a, b]$ . By the Intermediate Value Theorem, we know there exists a root  $x^* \in [a, b]$ . Now evaluate  $p = \frac{a+b}{2}$ , or the midpoint. and check the sign of  $f(a) \cdot f(p)$ . If this product is negative, then the root is in  $[a, p]$  and so set  $b \leftarrow p$ . Else, the root is in  $[p, b]$  and so set  $a \leftarrow p$ . Repeat.

**Note 3.8.** If we repeat this computation, we will have  $[a_0, b_0] \supset [a_1, b_1] \supset \cdots \supset [a_n, b_n] \supset \cdots$  then

$$\lim_{n \rightarrow \infty} a_n = \lim_{n \rightarrow \infty} b_n = x^*$$

the root of  $f(x)$ .

### §3.3 Fixed Point Iteration

**Definition 3.9** (Fixed Point Iteration). A fixed point denoted as  $x^*$  of a function  $g(x)$  is the point that satisfies

$$g(x^*) = x^*$$

Problem reformulation: Identifying a root of  $f(x)$  is reformulated into a problem of identifying a fixed point of  $g(x)$

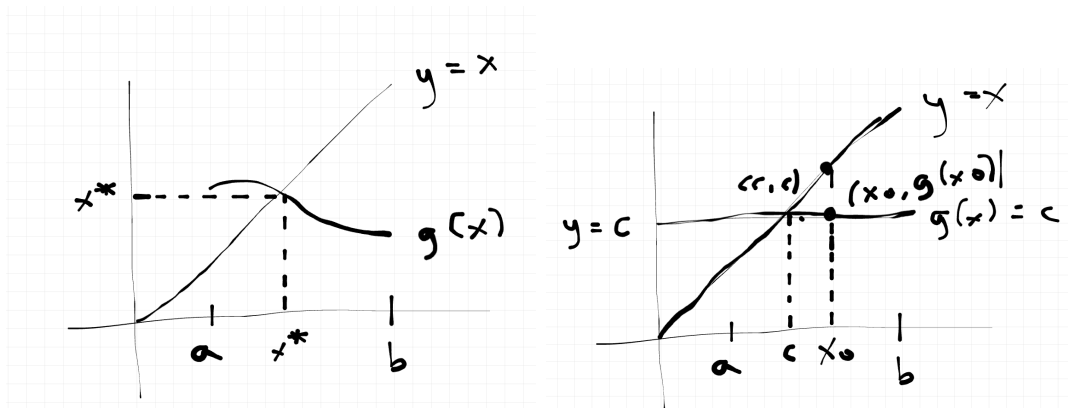
#### Example 3.10

Ex 1:  $g(x) = x - f(x)$  then  $g(x^*) = x^* \implies x^* = x^* - f(x^*) \implies f(x^*) = 0$

Ex 2:  $g(x) = \frac{x-f(x)}{f'(x)} \implies x^* = x^* - \frac{f(x^*)}{f'(x^*)} \implies \frac{f(x^*)}{f'(x^*)} = 0 \implies f(x^*) = 0$

We want to look at our problem and choose  $g$  such that it is easy to find  $x^*$  such that  $f(x^*) = 0$

**Note 3.11** (How do we find a fixed point?). Let's try to obtain a geometric understanding of the problem.



Start at an initial guess  $x_0$ , then evaluate  $g(x_0)$  and set up an improved guess

$$x_1 = g(x_0) = c$$

then surprisingly  $x_1$  is the fixed point  $x^*$ .

$$g(x_1) = g(c) = c = x_1 \implies g(x_1) = x_1$$

and therefore  $x_1$  is a fixed point. So, in the special case,  $g(x)$  is a constant, the iteration rule  $x_{k+1} = g(x_k)$  gives a fixed point after one iteration.

**Note 3.12.** Conjecture: we use  $x_{k+1} = g(x_k)$  as an updating rule to generate a sequence  $\{x_k\}$  for a general  $g(x)$ . We have  $x_k \xrightarrow{k \rightarrow \infty} x^*$  under certain conditions.

**Definition 3.13** (Fixed Point Algorithm). Given  $f(x)$ , select a function  $g(x)$  after an appropriate reformulation.

1. Start from an initial guess  $x_0$
2. For  $k = 0, 1, 2, \dots$  set  $x_{k+1} = g(x_k)$  until  $x_{k+1}$  satisfies a termination criteria.

**Note 3.14** (Questions for Fixed Point Iteration). Some important things to note about fixed point iteration

1. Is there a fixed point  $x^*$  on  $[a, b]$ ?

*Proof.* The proof for this is very similar to the proof of the existence of a root, it follows naturally from the Intermediate Value Theorem.  $\square$

2. If yes, what about convergence?

The fixed point is not necessarily unique ie. imagine a curve that oscillates repeatedly around  $g(x)$ . So  $g'(x)$  is a good property to use to study the fixed point iteration. If  $g(x) = c \iff g'(x) = 0$ ,  $x_1 = g(x_0)$  gives a fixed point  $x^* = x_1$  with only one step.

3. Does  $x_k \rightarrow x^*$ ?
4. If yes, how fast does  $x_k \rightarrow x^*$ ?