AMSC460: Computational Methods

James Zhang*

September 12, 2024

These are my notes for UMD's CMSC460: Computational Methods. These notes are taken live in class ("live-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

*Email: jzhang72@terpmail.umd.edu

§1 Scientific Computing

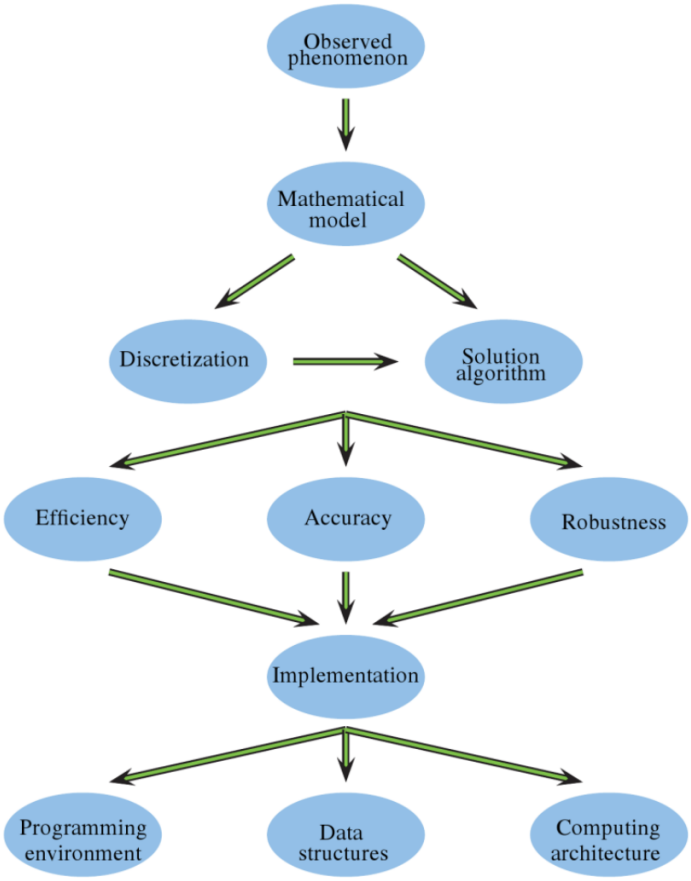


Figure 1.1. *Scientific computing.*

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

```
e = exp(1);
n = 1 : 10;
# Note that the following are vectors of len 10.
S_n = sqrt(2*pi*n).*((n/e).^n);
# Compute absolute and relative err.
fact_n = factorial(n);
abs_err = abs(fact_n - S_n);
rel_err = abs_err./abs(fact_n);
format short g
[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 ξ 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

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

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

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

$$\implies f'(x_0) = \frac{f(x_0+k)-f(x_0)}{h} - \left(\frac{h}{2}f''(x_0)+\cdots\right)$$

$$\implies \left|f'(x_0) - \frac{f(x_0+k)-f(x_0)}{h}\right| = \left|\frac{h}{2}f''(x_0)+\cdots\right|$$

Definition 1.7 (Big-$\mathcal{O}$ and Θ 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 Θ 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

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

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
0.1	4.716676e-2
0.01	4.666196e-3
0.001	4.660799e-4
1.e-4	4.660256e-5
1.e-7	4.619326e-8

h	Absolute error
1.e-8	4.361050e-10
1.e-9	5.594726e-8
1.e-10	1.669696e-7
1.e-11	7.938531e-6
1.e-13	4.250484e-4
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. $x \approx \pm 1$

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

where $d_1, d_2, \cdots$ 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 $x = -(1.10110\cdots) \times 2^1$ which means $x = -1 + 1 \times \frac{1}{2} + 0 \times \frac{1}{2^2} + 1 \times \frac{1}{2^3} + \cdots$ where the $\frac{1}{2^t}$ term, we denote as $\frac{1}{s^t}$

Definition 1.15 (Truncating)**.** Chopping ignores digits $d_t, d_{t+1}, \cdots$ yielding $\widetilde{x} = \pm(1.\{d_1\}\cdots\{d_{t-1}\}) \times 2^e \approx x$ 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 $x \in \mathbb{R}$

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

where e is an integer chosen to make d_0 not 0. The $1.d_1d_2\cdots$ is known as the mantissa.
For example: $x = 2^{12} \implies e = 12, x = (1.00\cdots) \times e^{12}$

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

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

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 (β, t, L, U) where

- β 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 $0\le\tilde{d_i}<\beta-1$ and e is chosen such that $\tilde{d_0}\neq 0$.

Definition 2.7 (Chopping)**.** Ignore $d_t,\cdots$ 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 $x = \frac{8}{3} = 2.66\cdots \in \mathbb{R}$

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

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

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

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

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

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

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 $x >> y$
2. If $|y| << 1$, 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 too 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 $[a, b] \implies$ 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

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

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 $|f(X_n)| <$ ftol
  2. If $|X_n - X_{n-1}| <$ atol ≈ 0
  3. If $X_n - X_{n-1}| <$ rtol ≈ 0
  4. If $n >$ 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

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

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

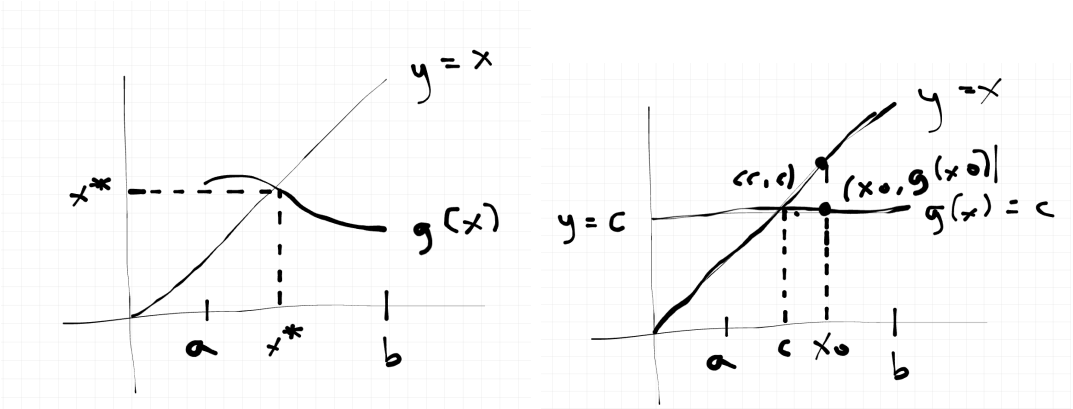
$$g(x^*) = x^*$$

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 geoemtric 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 $\underset{k\to\infty}{x_k} \to x^*$ 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 $k = 0, 1, 2, \cdots$ 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. □

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 $g(x) = c \longleftrightarrow g'(x) = 0$, $x_1 = g(x_0)$ gives a fixed point $x^* = x_1$ with only one step.

   If $g(x)$ changes a lot (ie. $|g'(x)|$) large, then fixed point iteration may be wrong. So, we conejcture that if $|g'(x)|$ is small, we can prove that $x_k \to x^*$

3. Does $x_k \to x^*$?

   Assume that $|g'(x) \leq \rho$ on $[a, b]$. Then we can prove uniqueness and convergence

   Proof. Uniqueness: suppose we have two fixed points x_1^*, x_2^* on $[a, b]$. The following first step is by definition of fixed points, then apply Taylor Series First Order Approximation

   $$|x_1^* - x_2^*| = |g(x_1^*) - g(x_2^*)| = |g'(\xi)(x_1^* - x_2^*)| \leq \rho|x_1^* - x_2^*|$$

   Note that we have shown that $|x_1^* - x_2^*| \leq \rho|x_1^* - x_2^*|$. If $\rho > 1$, then this doesn't show us anything useful. If $\rho < 1$, then this is only satisfied if $|x_1^* - x_2^*| = 0 \implies x_1^* = x_2^*$.

Convergence:

$$x_{k+1} - x^*| = |g(x_k) - g(x^*)| = |g'(\xi)(x_k - x^*) \leq \rho(x_k - x^*)|$$

and so

$$|x_{k+1} - x^*| \leq \rho|x_k - x^*| \leq \rho^2|x_{k-1} - x^*|| \leq \cdots \leq \rho^{k+1}|x_0 - x^*||$$

$$\lim_{k\to\infty} |x_{k+1} - x^*| = 0$$

□

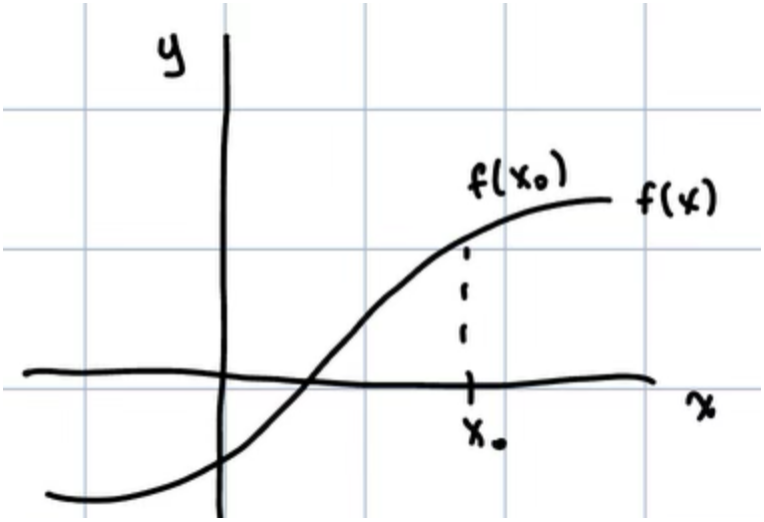
4. If yes, how fast does $x_k \to x^*$?

   This is a contraction by the faster ρ, the smaller ρ is, the faster the convergence

$$\text{rate} = -\log_{10}(\rho)$$

§3.4 Newton's Method

Note 3.15.



Definition 3.16 (Analytic Understanding of Newton's Method)**.** Consider the First Order Taylor Series expansion of $f(x)$

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(\xi(x))}{2}(x - x_k)^2$$

where ξ is between x and x_k. $f'(x_k) = \frac{f(x_k)}{x_k - x_{k+1}}$ is a linear equation in x_{k+1}

$$\Leftrightarrow (x_k - x_{k+1})f'(x_k) = f(x_k)$$

$$\Leftrightarrow f(x_k) + (x_{k+1} - x_k)f'(x_k) = 0$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

and this is the updating rule to get x_{k+1} which is closer to x^* than x_k. This is the Newton's Method to generate a sequence of $\{x_k\}$.

Definition 3.17 (Newton's Method as an Algorithm)**.**

1. Start from an initial guess x_0.
2. FOr $k = 0, 1, 2, \ldots,$ set
$$x_{k+1} = x_k - \frac{x_k}{f'(x_k)}$$
until x_{k+1} satisfies the termination criteria.

Note 3.18 (Local Convergence)**.** If you don't start with a sufficiently good x_0, you may not have $\lim_{k\to\infty} x_k = x^*$

Note 3.19 (Global Convergence)**.** To find x_0 good enough to ensure convergence, plot the graph and start in a neighborhood near-ish a root. (what?)

Note 3.20 (How Fast is Convergence?)**.**

1. Recall that for fixed point iteration, convergence is dependent on ρ
$$|x_{k+1} - x^*| \leq p|x_k - x^*|$$
this is called **linear convergence** because the right hand side is a linear function of the error term $|x_k - x^*|$.
2. On the other hand, Newton's Method has **quadratic convergence**
$$|x_{k+1} - x^*| \leq M|x_k - x^*|^2$$
Typically, we do'n't go above this (ie. to cubic convergence) due to the increase of computing resources
3. **Superlinear convergence**: if $\exists\ \{\rho_k\}, \rho_k \to 0$ such that
$$|x_{k+1} - x^*| \leq \rho_k|x_k - x^*| \leq \rho|x_k - x^*|$$
Note that superlinear convergence implies linear convergence, and it may or may not be quadratic convergence, but quadratic convergence is always superlinear convergence.

§3.5 Secant Method - Variant of Newton's Method

Definition 3.21. Consider a First Order Taylor Series Expansion once more

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \implies f'(x_0) \approx \frac{f(x) - f(x_0)}{x - x_0}$$

$$\implies f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

$$\implies x_{k+1} = x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

Note 3.22 (Secant Method as an Algorithm)**.**

1. Start from an educated guess x_0
2. For $k = 0, 1, 2, \cdots$ set

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

until x_{k+1} satisfies the termination criteria. The Secant Method has the same convergence properties as Newton's Method.

§3.6 Minimizing a Function in One Variable

Note 3.23. Note that the minimizer x^* is a root of $f'(x)$, so we identify all the possible roots of $f'(x)$ using the root finding algorithms. To determine which root is the minimizer that we want

- If $f''(x^*) > 0$, then x^* is a local minimizer
- If $f''(x^*) < 0$, then x^* is a local maximizer
- If $f''(x^*) = 0$, then undetermined.

To determine which one is the global minimizer among all local minimizers, compare f at all minimizers to determine the smallest one.

§3.7 Linear Algebra Background

Omitted.

§3.8 Linear Systems: Direct Methods

§3.9 Gaussian Elimination

Definition 3.24. Given $Ax = b$,

$$\left[A \quad | \quad b\right] \overset{\text{Gaussian Elimination}}{\Longrightarrow} \left[U \quad | \quad \widetilde{b}\right]$$

where U is an upper triangular matrix. Use backward substitution to sovle $Ux = \widetilde{b}$ very efficiently with operation count $\mathcal{O}(N^2)$ compared to $\mathcal{O}(N^3)$ to compute A^{-1}.

Note 3.25. This leads to the LU Decomposition which we will learn about next class.