LECTURE NOTES "APPLIED NUMERICAL METHODS" (151 A)

I typed these notes from handwritten lecture notes of Professor Chris Anderson with some modifications and extensions, and adding resources from WWW. As usual corrections and comments are welcome, preferably by email.

1. Introduction and overview

The question of what is generally understood to be numerical analysis may be answered by how it is used. The aim is to provide a "machine" solution to math problems. There are several aspects and fields involved with providing and studying machine solution of math problems:

- Technology: computational hardware such as desktop and laptop computers, smart phones, GPS devices, smart watches, smart cars, internet of things (IoT) devices, etc.
- Computer science: computer architecture, languages and algorithms
- Mathematics: Numerical analysis (understanding and developing numerical methods for math problems), and applied mathematics (data science, statistics, financial mathematics) by adapting numerical methods for practical purposes

What are examples of the type of math problems which are considered in numerical analysis? The problems that arise when

- *Solving differential equations* which include mathematical models in engineering, physics, chemistry, biology, etc.
- *Signal processing* problems such as remove noise and feature detection.
- *Data processing and mining* such as efficient search, predicting behavior based on history, identification of models e.g. model a successful Go player or a chemist who can predict which atoms will bind together, etc.

If you think about it a bit - when you take courses in applied math or engineering and the sciences, a big part of what you learn about are the math problems that are associated with the topic being studied. Typical homework problem requires that you

- set up an appropriate math problem
- solve the math problem
- interpret math problem solution with respect to the homework question

When learning about the appropriate math problems, you are generally told how to solve them. The solution procedures can be "analytic", e.g. using pencil and paper, or computational, e.g. run this program or software tool to get an answer. This is fine - but - when you are working on problems that are not "homework" problems, in addition to setting up the appropriate math problem you are also responsible for choosing or developing a procedure to solve the problem. To develop analytic procedures, you call upon what you have learned in 1st and 2nd year calculus, and other "theoretical applied math" courses. To develop computational procedures, you use what you have learned in calculus and theoretical applied math courses *and* numerical analysis courses.

What is involved in developing computational procedures?

- Choosing appropriate numerical methods
- Implementing (or finding implementations) of the methods
- Debugging when "things don't work out"

Date: Fall 2019, UCLA.

- When things don't work - there are several possibilities:
  - Is it a programming bug?
  - Did I choose the right method?
  - Is it the problem?
  - ...

In this course, we will learn about some of the fundamental numerical algorithms, that are useful by themselves, or as components in more complex algorithms (such as those covered in Math 151 B). You will also learn the role of theory - how it is not only useful for selecting appropriate methods, but also as an aid in debugging.

We will cover the following topics:

(1) Solving nonlinear equations
(2) Numbers and precision
(3) Interpolation
(4) Numerical differentation
(5) Numerical integration
(6) Solving linear systems of equations

When developing numerical procedures, implementation can take quite a bit of time, and there are software packages that facilitate rapid implementation. We will be using Matlab (or GNU-Octave). Why?

- It will make doing the assignments easier than using C++.
- It is a widely used software tool that every math major should know how to use.
- If you don't already know Matlab, you will be learning to use it in a way that mimics a way to learn about other software or programming packages.

Assignment 1. You are using Matlab to create a plot and then turn in a `pdf` of the plot.

(i) Download and run Matlab script.
(ii) Modify script and create plot of $\sin(x^2)$.
(iii) Create `pdf` of plot, upload modified script and `pdf` to course web site.

Tip: You need to add 3 characters to the script - not 2.

For this assignment, you will need access to Matlab for which you have the following options

- PIC Lab
- Buy student version online
- Download GNU-Octave (Matlab clone)

We start with [2, Chapter 2].

2. Solving nonlinear equations

We recall the definition of a linear function.

Definition 2.1. A function $f\colon \mathbb{R} \to \mathbb{R}$ is *linear* if

(i) $f(x+y) = f(x) + f(y)$ for all $x, y \in \mathbb{R}$,
(ii) $f(cx) = cf(x)$ for all $c \in \mathbb{R}$ and $x \in \mathbb{R}$.

A function $f\colon \mathbb{R}^m \to \mathbb{R}^n$ is *linear* if

(i) $f(\vec{x}+\vec{y}) = f(\vec{x}) + f(\vec{y})$ for all vectors $\vec{x}, \vec{y} \in \mathbb{R}^m$,
(ii) $f(c\vec{x}) = cf(\vec{x})$ for all scalars $c \in \mathbb{R}$ and every vector $\vec{x} \in \mathbb{R}^m$.

A linear function may intuitively be described as:

- $f(x+y) = f(x) + f(y)$ means that the output of the sum of two inputs is equal to the sum of the outputs of each input (this extends to finite sums).
- $f(cx) = cf(x)$ means that output is directly proportional to input.

Remark 2.2. A linear function $f\colon \mathbb{R}^m \to \mathbb{R}^n$ can always be represented by an $m \times n$ matrix A in the sense that $f(\vec{x}) = A\vec{x}$ where $A\vec{x}$ denotes the product of a matrix with a vector. Special case $f\colon \mathbb{R} \to \mathbb{R}$ given by $f(x) = ax$.

A linear function passes always through the origin. More generally, we can define a "shifted" linear function:

Definition 2.3. An *affine* function is the sum of a linear function and a constant:

a) For a scalar function $l\colon \mathbb{R} \to \mathbb{R}$, this means that $l(x) = ax + b$ for some constant scalar $b \in \mathbb{R}$.
b) For a vector-valued function $L\colon \mathbb{R}^m \to \mathbb{R}^n$, this means that $L(\vec{x}) = A\vec{x} + \vec{b}$ for some constant vector $\vec{b} \in \mathbb{R}^n$.

We have the following examples of linear functions.

Examples 2.4. 1.) Scalar multiplication $f(x) = 3x$. Verification:

$$f(x+y) = 3(x+y) = 3x + 3y = f(x) + f(y)$$
$$f(cx) = 3cx = c(3x) = cf(x)$$

2.) Matrix multiplication

$$\vec{F}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 + 2x_2 \\ 3x_1 + 4x_2 \end{bmatrix}$$

We have the following properties of matrix multiplication

$$A(\vec{x} + \vec{y}) = A\vec{x} + A\vec{y}$$
$$A(c\vec{x}) = cA(\vec{x})$$

from which linearity of $\vec{F}$ follows.

What are some real world examples of linear functions:

- sales tax one pays on a purchase
- Hooke's law: Restoring force of a spring $F = -\kappa d$ where κ is the spring constant and d is the displacement (when d is small).

Definition 2.5. A function is said to be *nonlinear* if it is not affine!

There are plenty of examples of nonlinear functions. Here are some simple ones.

Examples 2.6. 1.) $f(x) = \sqrt{x}$ since additivity $\sqrt{x+y} \neq \sqrt{x} + \sqrt{y}$ is violated ($\sqrt{1+3} = 2 \neq 1 + \sqrt{3}$).
2.) $f(x) = \sin(x)$ since $\sin(x+y) \neq \sin(x) + \sin(y)$. Actually, we have the sine addition formula

$$\sin(x+y) = \sin(x)\cos(y) + \sin(y)\cos(x).$$

3.) A vector-valued polynomial

$$\vec{F}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 3x_1^2 \\ 4x_1 - 3x_2^3 \end{bmatrix}$$

since neither x^2 nor x^3 are linear functions.

Some real world examples of nonlinear functions are the following.

Examples 2.7. 1.) The braking distance of a vehicle as a function of velocity:

$$\text{dist} = \frac{v^2}{2\mu g}$$

where μ is the coefficient of friction and g is the gravity of Earth.

2.) The monthly payment on a loan as a function of the interest rate

$$p = \frac{1}{12}\frac{rAe^{rN}}{(e^{rN}-1)}$$

where A is loan amount, N is duration of loan, and r is interest rate (compounded continuously).
3.) Federal tax one pays as a function of income and deductions.
4.) Spread of infectious diseases and population growth, etc.

2.1. **Solving equations.** For scalar functions $f\colon \mathbb{R} \to \mathbb{R}$, given a value $b \in \mathbb{R}$ find $x \in \mathbb{R}$ such that $f(x) = b$. For vector-valued functions $\vec{F}\colon \mathbb{R}^m \to \mathbb{R}^n$, given a vector $\vec{b} \in \mathbb{R}^n$ find $\vec{x} \in \mathbb{R}^n$ such that $F(\vec{x}) = \vec{b}$.

Example 2.8. Find the interest rate so a \$5,000 and 5 year loan will have a monthly payment of \$180:

$$\frac{1}{12}\frac{r5000e^{r5}}{(e^{r5}-1)} = 180$$

This is a problem of the form $f(x) = b$.

Depending on the function f or $\vec{F}$, there are different solution methods:

- If $f, \vec{F}$ is linear or affine, we can use techniques from linear algebra and computer to do the work.
- If $f, \vec{F}$ are nonlinear, we distinguish between computational techniques to solve scalar nonlinear equations (which will be the topic of the following lectures) and nonlinear systems of equations (this is covered in Math 151 B).

For equations that involve linear or affine functions you've learned techniques to solve them (algebra, linear algebra):

$$3x = 6 \quad (x = 2), \qquad \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \vec{x} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \;\to\; \vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

We'll be discussing "getting the computer to do the work" for these problems later.

For equations with nonlinear functions, if you learned solution techniques they were problem specific - a specific sequence of algebraic manipulations - or the application of a formula of limited applicability, e.g. the quadratic formula.

Examples 2.9.

$$\frac{1}{x} + x = 3 \rightsquigarrow 1 + x^2 = 3x \rightsquigarrow x^2 - 3x + 1 = 0 \text{ use quadratic formula}$$

$$\frac{1}{x^7} + x = 3 \rightsquigarrow 1 + x^8 = 3x^7 \rightsquigarrow x^8 - 3x^7 + 1 = 0 \text{ there is no formula}$$

We'll be learning about techniques to solve problems of the form $f(x) = b$ when $f(x)$ is nonlinear and are *generally applicable*, that is they are methods that assume very little about the exact form of $f(x)$.

We start with an important observation. When considering methods to solve nonlinear equations one observes that solving $g(x) = b$ is equivalent to solving $g(x) - b = 0$, which amounts to finding roots x^* of the function $f(x) = g(x) - b$:

$$f(x^*) = 0 \quad \longleftrightarrow \quad g(x^*) - b = 0 \quad \longleftrightarrow \quad g(x^*) = b.$$

When seeking methods to solve $g(x) = b$ one seeks methods *for finding roots* of nonlinear equations. Thus, the methods we'll be considering will all be 'root finding' methods, that is methods to find x^* so that $f(x^*) = 0$. If you are confronted with a problem of the form

$g(x) = b$, you need to apply these methods to the equivalent root finding problem $f(x) = 0$ where $f(x) = g(x) - b$.

Example 2.10. For the problem in Example 2.8, to find r, one would apply a root finding method to $f(r) = 0$ where

$$f(r) = \frac{1}{12}\frac{5000e^{5r}}{e^{5r}-1} - 180$$

3. Root finding methods

What are methods for finding roots? Finding x^* such that $f(x^*) = 0$. If you didn't know numerical analysis, and have a plotting program how would you find a root?

$$\text{Plot} \rightsquigarrow \text{Zoom} \rightsquigarrow \text{Plot} \rightsquigarrow \text{Zoom} \rightsquigarrow \dots$$

This works - but unfortunately involves a human being and lots of functions evaluations to create plots.

Exercise 3.1. You should try this out with some plotting software for different functions.

Can this be automated? Yes. This leads to the 'bisection method'.

3.1. The bisection method idea to find roots of $\mathbf{f(x) = 0}$.

The bisection method works under the following two assumptions:

- There is a root of $f(x)$ in $[a, b]$.
- $f(a) \cdot f(b) < 0$.

The idea that underlies the bisection method can be summarized as:

- Starting with an interval containing a root $[a_0, b_0]$ (by the intermediate value theorem, this amounts to finding $a_0 < b_0$ such that $f(a_0) \cdot f(b_0) < 0$ for a continuous function).
- Iterate then
  - bisect interval; use bisection point as approximate root.
  - determine if a root is in left or right interval by checking sign change.
  - Reset interval.

Let's describe a pseudo-code for the bisection method.

Algorithm.
```
1. Choose an interval $[a_0, b_0]$ such that $f(a_0) \cdot f(b_0) < 0$.
2. Let $x_k = \frac{a_k+b_k}{2}$.
3. Check $f(a_k) \cdot f(x_k) < 0$:
 a.) Yes, a root in $[a_k, x_k]$, so set $k \leftarrow k+1$, $a_k \leftarrow a_k$ and $b_k \leftarrow x_k$.
 b.) No, a root in $[x_k, b_k]$, so set $k \leftarrow k+1$, $a_k \leftarrow x_k$ and $b_k \leftarrow b_k$.
4. Repeat.
```

Remark 3.2. The bisection method works under the assumption that f is continuous on the initial interval $[a_0, b_0]$ as the existence of a root in this interval is a consequence of the intermediate value theorem. However, we don't have to guarantee that there is exactly one root in $[a_0, b_0]$. The method will find exactly one root (even when there are multiple roots in $[a_0, b_0]$).

Demonstration 3.3. Wolfram demonstration of the bisection method for different functions.

Next, we will discuss some implementation and theoretical aspects of the bisection method:

- Idea and implementation
- Root finding terminology: 'error' and 'residual'
- Stopping conditions

Usually, it is not too difficult to figure out how to implement the basic steps - the difficult part is to figure out when to stop. To this end, we need to introduce some standard terminology; but before we do that we need to discuss how one should think about expressions $|x|$ and $|x-y|$. If you have taken Real Analysis 131A, you already think the right way - if not, you may need to change how you think of these expressions. You should *not* think of $|x|$ as the absolute value of x, but rather as the magnitude or size of x. We will be always thinking of $|x|$ as a measure of x's size. You think about $|x|$ as the absolute value when you want to evaluate the value. In higher dimensions, the analogue is the norm (or magnitude) $\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$. Similarly, we don't think of $|x-y|$ as the absolute value of $x-y$ but rather as the distance between $x-y$ (What was again the higher dimensional analogue of the distance?). If you think about $|x-y|$ as the distance between x and y, then solving problems such as

> Find all x such that $|x-a| < \epsilon$ ('the distance between x and a is not more than ϵ'(which you may choose to be 10^{-10}))

is much easier. To illustrate the situation, you can draw an interval of length 2ϵ around a point x. Let us introduce some terminology which allows us to analyze the accuracy of approximate solutions.

Definition 3.4. We consider the problem of determining solutions to

$$f(x) = 0$$

Let x^* be an exact solution, that is $f(x^*) = 0$. Let x_k be an approximate solution. The *error* of the approximation is the value $e_k = x_k - x^*$. The *residual* of the approximation is the value $f(x_k)$. The size of the error $|x_k - x^*|$ measures how far away you are from a *solution of the equation.* The size of the residual $|f(x_k)|$ measures how well the *equations are being satisfied.*

Now that we have introduced measures for the quality of a solution, we can use these to describe stopping conditions for root finding methods. We denote by 'tol' a specified stopping tolerance (which is usually some small positive number such as 10^{-6}).

(1) When the residual is sufficiently small, that is stop when $|f(x_k)| < \text{tol}$.
(2) When a computer error bound or estimate is sufficiently small, that is at step k, determine a value B_k such that $|x_k - x^*| \leq B_k$, and stop when $B_k < \text{tol}$.
(3) When the difference between two iterates is sufficiently small, that is stop when $|x_k - x_{k+1}| < \text{tol}$.
(4) When the number of iterations exceeds a prespecified maximal iteration number.
(5) Combinations of (1)–(4).

Let's apply this to the bisection method. We have an easily computed error bound. At step k, we have that x_k and x^* lie between a_k and b_k (**illustrate**), so that the distance between x_k and x^* is at most $\frac{b_k - a_k}{2}$, that is

$$|x_k - x^*| \leq \frac{b_k - a_k}{2}$$

A stopping condition to use: stop when $\frac{b_k - a_k}{2} < \text{tol}$, since this immediately implies that $|x_k - x^*| < \text{tol}$.

Look at the code `bisect.m`. Using both residual stopping condition and error bound stopping conditions. More precisely,

- Checking $|f(a_0)|$ and $|f(b_0)|$.
- Checking $|f(x_k)|$ and $\frac{|b_k - a_k|}{2}$.
- Checking that number of iterations $<$ iterations maximum (this protects against infinite loops caused by programming errors or user input errors).

Remark 3.5. In assignment 2, you need to modify `bisect.m` (after renaming it to `false.m`) to implement a Regula-Falsi approach. The Regula-Falsi method is almost the same as the bisection method, but the interval isn't bisected, it's split up using the point where the line through $(a_k, f(a_k))$ and $(b_k, f(b_k))$ intersect the x-axis (**illustrate**).

General plan for the next lectures.

- Background on Taylor series.
- We learn two more root finding methods, the Newton and secant methods. We derive these methods, discuss stopping conditions, and the relation between residual and error.
- Convergence theory, fixed point theory, error bound behavior.

3.2. Taylor's theorem.

You may find this also in [2, Theorem 1.14].

Theorem 3.6. *Let $f\colon [a,b] \to \mathbb{R}$ be an $n+1$ times continuously differentiable function. Then for all $x_0, x \in [a,b]$ there is a number $\xi(x) \in [x_0, x]$ such that*

$$\begin{aligned} f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 \\ + \frac{f'''(x_0)}{3!}(x-x_0)^3 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + \frac{f^{(n+1)}(\xi(x))(x-x_0)^{n+1}}{(n+1)!} \end{aligned}$$

where $f^{(i)}$ is the ith derivative for $i = 1, \ldots, n+1$.

Proof. Sketch: Use successively integration by parts, and then apply the extreme value theorem and the intermediate value theorem. A full proof based on this sketch and a list of other ‘simple’ proofs can be found here Proofs of Taylor's formula. □

A useful result that follows immediately from this theorem is:

Corollary 3.7. *If f is $n+1$ times continuously differentiable over $[a,b]$ and*

$$\max_{x\in[a,b]} |f^{(n+1)}(x)| \le M_{n+1}.$$

Then for all $x, x_0 \in [a,b]$, we have

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \frac{f'''(x_0)}{3!}(x-x_0)^3 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + R_{n+1}$$

where

$$|R| \le \frac{M_{n+1}}{(n+1)!}|(x-x_0)^{n+1}|.$$

We can rewrite this last expression compactly as

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + O(|x-x_0|^{n+1})$$

where $O(|x-x_0|^{n+1})$ is an ‘order notation’ which is read as big-O of $|x-x_0|^{n+1}$:

Remark 3.8. ‘R is $O(|x-x_0|^{n+1})$’ means that there exists a constant $C > 0$ such that $|R| \le C|x-x_0|^{n+1}$ for x sufficiently close to $x+0$ (in Taylor's theorem above, you may choose $C \ge \frac{M_{n+1}}{(n+1)!}$). Here is Wikipedia article on Big-O notation, see also the additional material “Asymptotic Methods in Analysis” of de Brujin on CCLE.

Taylor's theorem is useful for us because it gives a way of considering a sequence of approximations to a function:

- $f(x) \simeq f(x_0)$ constant approximation (1 term)
- $f(x) \simeq f(x_0) + f'(x_0)(x-x_0)$ linear approximation (2 term)

- $f(x) \simeq f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$ quadratic approximation (3 terms)
- higher-order approximations ...

The size of the maximal error of the nth approximation (starting from $n = 0$) is $O(|x-x_0|^{n+1})$. If $|x-x_0| < 1$, then $|x-x_0|^{n+1} \to 0$ as $n \to \infty$. This implies that we get better approximations as $n \to \infty$ as long as $\frac{M_{n+1}}{(n+1)!}$ doesn't grow too fast with n.

Demonstration 3.9. See Taylor approximation in one variable, for a demonstration of Taylor approximation of different functions.

3.3. **Newton's and Secant method for finding roots of $\mathbf{f(x) = 0}$.** Idea of Newton's method: given an iterate x_k, the next iterate is the root of the 2nd term linear Taylor series approximation to $f(x)$ about x_k.

Derivation of formula to obtain x_{k+1} from x_k:

- 2 term Taylor approximation: $l(x) = f(x_k) + f'(x_k)(x - x_k)$.
- $x_{k+1} =$ root of $l(x)$ which amounts to finding x^* with $l(x^*) = 0$ and then set $x_{k+1} = x^*$:

$$\begin{aligned} f(x_k) + f'(x_k)(x^* - x_k) &= 0 \\ \rightsquigarrow x^* - x_k &= \frac{-f(x_k)}{f'(x_k)} \\ \rightsquigarrow x^* &= x_k - \frac{f(x_k)}{f'(x_k)} \end{aligned}$$

Algorithm (Newton's method)**.**

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

Demonstration 3.10. See Newton's method, for a demonstration of the Newton's method.

Idea of Secant method: Given two iterates x_{k-1} and x_k, the next iterate is the root of the secant approximation to f, that is the linear function passing through $(x_{k-1}, f(x_{k-1}))$ and $(x_k, f(x_k))$ (**illustrate**).

Remark 3.11. It should be called an affine function, however convention tells to call functions of the form $ax + b$ linear.

Derivation of formula:

- Secant approximation[1]:

$$l(x) = f(x_k) + \left[\frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}\right](x - x_k)$$

- Solve $l(x^*) = 0$ for x^* and set $x_{k+1} = x^*$:

$$\begin{aligned} f(x_k) + \left[\frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}\right](x - x_k) &= 0 \\ \rightsquigarrow x^* - x_k &= -\frac{f(x_k)}{\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}} \\ \rightsquigarrow x^* &= x_k - \frac{f(x_k)}{\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}} \\ \rightsquigarrow x_{k+1} &= x_k - \frac{f(x_k)}{\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}} \end{aligned}$$

[1]Work it out, or check that $l(x)$ is (i) linear, (ii) $l(x_{k+1}) = f(x_{k+1})$, and (iii) $l(x_k) = f(x_k)$.

Algorithm (Secant method)**.**

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

Demonstration 3.12. Check Secant method. Play with the parameters and see what happens.

We want to better understand:

- When to stop?
- When do they work?
- If they don't always work, why use them?

With the bisection method, at step k, we know that $x_k = (a_k+b_k)/2$, and one has the following bound for the error $e_k = x_k - x^*$

$$|x_k - x^*| \le \frac{|b_k - a_k|}{2}$$

So stopping when

$$\frac{|b_k - a_k|}{2} < \text{tol}$$

implies that we could bound the error by

$$|x_k - x^*| < \text{tol}$$

In words, the error in the approximate root is at most tol.

With Newton's method or the secant method, we don't have such a bound. Therefore, a common stopping condition is to stop when the residual $|f(x_k)|$ is small enough. This raises the question: Near a root $|f(x_k)|$ will be small, but how does one relate the size of the residual $|f(x_k)|$ to the size of the error $|x_k - x^*|$? To answer this question, we need the mean value theorem:

Theorem 3.13. *If $f\colon [a,b] \to \mathbb{R}$ is continuously differentiable, then for all $z_1, z_2 \in [a,b]$ with $z_1 \neq z_2$ there exists $\xi \in (z_1, z_2) \cup (z_2, z_1)$ such that*

$$f'(\xi) = \frac{f(z_2) - f(z_1)}{z_2 - z_1}$$

or

$$f(z_2) = f(z_1) + f'(\xi)(z_2 - z_1)$$

(which is the 1-term Taylor approximation with remainder).

Now if we choose $z_1 = x^*$ and $z_2 = x_k$ (and assume that $x^* \neq x_k$), then from the previous theorem we find $\xi \in (x^*, x_k) \cup (x_k, x^*)$ such that

$$f(x_k) - f(x^*) = f'(\xi)(x_k - x^*)$$

Since $f(x^*) = 0$, we have

$$f(x_k) = f'(\xi)(x_k - x^*)$$

This implies that

$$|f(x_k)| = |f'(\xi)||x_k - x^*|$$

where the left-hand side is the size of the residual the second term on the right-hand side is the size of the error. We notice that for $\xi \in [x^*, x_k] \cup [x_k, x^*]$, if $x_k \to x^*$, then $|x_k - \xi| \to 0$, that is ξ and x_k are close under convergence. If we now use that $f'(x_k) \simeq f'(\xi)$ (which is a good approximation when x_k is close to x^*), then one can use the estimate

$$|x_k - x^*| \simeq \frac{|f(x_k)|}{|f'(x_k)|}$$

to stop. Indeed, if one stops when

$$\frac{|f(x_k)|}{|f'(x_k)|} < \text{tol}$$

then this will imply that

$$|x_k - x^*| \leq \text{tol}$$

This is now a suitable stopping condition one can use for Newton's method when one seeks an approximation such that $|x_k - x^*| < \text{tol}$.

What makes this stopping condition interesting is that it's easy to obtain, since for Newton's method

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

So if tol is given, stopping when

$$|x_{k+1} - x_k| < \text{tol} \quad \rightsquigarrow \quad \frac{|f(x_k)|}{|f'(x_k)|} < \text{tol} \quad \rightsquigarrow \quad |x_k - x^*| \leq \text{tol}$$

This is the stopping condition used in the pseudo-code given in the book [2, Algorithm 2.3].

Similarly for the secant method, if one wants a stopping condition so that $|x_k - x^*| < \text{tol}$, then one can use the condition $|x_{k+1} - x_k| < \text{tol}$ since

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

$$\rightsquigarrow \quad |x_{k+1} - x_k| = \frac{|f(x_k)|}{\left|\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}\right|}$$

However we know that

$$\left|\frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}\right| \simeq f'(x_k) \simeq f'(\xi)$$

whenever x_{k+1} and x_k are both close to x^*. So

$$|x_{k+1} - x_k| = \frac{|f(x_k)|}{\left|\frac{f(x_{k-1})-f(x_k)}{x_{k-1}-x_k}\right|} \simeq \frac{|f(x_k)|}{|f'(x_k)|} \simeq \frac{|f(x_k)|}{f'(\xi)} \simeq |x_k - x^*|$$

Stopping when $|x_{k+1} - x_k| < \text{tol}$ implies that $|x_k - x^*| \leq \text{tol}$ for the secant method.

If you just want to determine an approximation x_k to x^* such that $|f(x_k)|$ is small, and one isn't concerned with how close x_k is to x^*, then one typically uses a stopping condition of the form

Stop when $|f(x_k)| < \text{tol}$ for some user specified tolerance.

If you want to determine an approximation x_k to x^* with an error less than some specified tolerance tol, one typically uses the stopping condition

Stop when $|x_{k+1} - x_k| < \text{tol}$ for some user specified tolerance.

As discussed above, when x_k is close to x^*, then for both Newton's method and the secant method, stopping when $|x_{k+1} - x_k| < \text{tol}$ implies $|x_k - x^*| \leq \text{tol}$.

Remark 3.14. In practice, there are some problems where you are just interested in x_k with a small residual, while with other problems, you may want an approximation x_k that has small error, that is $|x_k - x^*|$ is small.

4. Convergence theory

Definition 4.1. Let (x_k) be an infinite sequence of real numbers. The sequence converges to x^* if for all $\epsilon > 0$ there exists $N(\epsilon)$ such that if $k > N(\epsilon)$ then $|x_k - x^*| < \epsilon$. The notation $\lim_{k\to\infty} x_k = x^*$ or $x_k \to x^*$ as $k \to \infty$ means that the sequence (x_k) converges to x^*.

In general, in non-numerical analysis course you investigate sequences (x_k) and were primarily concerned with determining whether or not the sequence converges; specifically if there is a value x^* such that $\lim_{k\to\infty} x_k = x^*$. In numerical analysis, when considering sequences of values (x_k) that are generated by some computational process, one is still concerned with convergence; specifically one seeks to determine under what conditions (e.g. starting iterates, function properties, algorithm parameters) the sequence of approximations will converge to the correct value. However, one is also concerned with knowing how 'fast' the sequence converges when it does converge. The question we would like to answer for methods that seek roots x^* such that $f(x^*) = 0$:

(a) Under what conditions can one guarantee that the iterates $x_k \to x^*$?
(b) If a method converges, that is $x_k \to x^*$, then can we quantify how fast $x_k \to x^*$?
  (i) Can one determine theoretically how fast $x_k \to x^*$?
  (ii) How does one estimate how fast $x_k \to x^*$ from experimental results?

Plan:

(1) Definitions of order of convergence and asymptotic error constant that are needed to quantify how fast $x_k \to x^*$.
(2) Discuss how to estimate the order of convergence and the asymptotic rate.
(3) Discussion of theorems concerning convergence results and rates of convergence.

Announcements:

- Have a look at the handout "General theory of one point methods".
- Application of a theorem for fixed point methods that can be used to theoretically predict rates of convergence (apply this to solve [T3] of Assignment 3).

Recall the intermediate value theorem.

Theorem 4.2. *Let $f\colon [a,b] \to \mathbb{R}$ be a continuous function. Then for every value $y \in [f(a), f(b)]$ there is a value x such that $f(x) = y$.*

An application of this is the idea underlying the bisection method. If f is a continuous function over $[a,b]$ and $f(a)\cdot f(b) \le 0$, then there exists a value $x^* \in [a,b]$ such that $f(x^*) = 0$. We start with the convergence result for the bisection method.

Theorem 4.3. *Let f be a continuous function over $[a_0, b_0]$ and $f(a_0)\cdot f(b_0) \le 0$. Then the values $x_k = \frac{a_k+b_k}{2}$ generated by the bisection method converge to a root $x^* \in [a_0, b_0]$. Moreover, at the kth step we have*

- $f(a_k)\cdot f(b_k) \le 0$
- $|b_k - a_k| = (b_0 - a_0)/2^k$
- *there exists a root x^* such that* $|x_k - x^*| \le (b_0 - a_0)/2^{k+1}$

Proof. The proof goes by induction.

- At step 0: Since $f(a_0)\cdot f(b_0) \le 0$, the intermediate value theorem applies and there exists $x^* \in [a_0, b_0]$, and $x_0 = \frac{a_0+b_0}{2}$ satisfies $|x_0 - x^*| \le |b_0 - a_0|/2$.
- At step k: If $f(a_k)\cdot f(b_k) \le 0$, then we can choose $[a_{k+1}, b_{k+1}]$ to be either $[a_k, \frac{a_k+b_k}{2}]$ or $[\frac{a_k+b_k}{2}, b_k]$ such that $f(a_{k+1})\cdot f(b_{k+1}) \le 0$. Otherwise, $f(a_k)\cdot f(b_k) \le 0$ would be contradicted. By the intermediate value theorem, there exists a root $x^* \in [a_{k+1}, b_{k+1}]$, and we have
$$b_{k+1} - a_{k+1} = \frac{b_k - a_k}{2} = \frac{b_0 - a_0}{2^{k+1}}$$

so that

$$|x_{k+1} - x^*| \leq \frac{b_0 - a_0}{2^{k+2}}$$

This implies the induction step $k \to k+1$.

Convergence to a root $x^* \in [a_0, b_0]$ follows from the bound

$$|x_k - x^*| \leq \frac{b_0 - a_0}{2^{k+1}}$$

by letting $k \to \infty$. □

The following corresponds to [2, Definition 2.7].

Definition 4.4. Suppose that (x_k) is a sequence converging to x^* with $x_k \neq x^*$ for all k. If there exist positive constants α and λ such that

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^\alpha} = \lambda,$$

then we say that x_k converges to x^* with *convergence order* α and *convergence rate* λ.

Remark 4.5. See Rates of convergence for additional reading and some examples.

If $x_k \to x^*$ with order α and rate λ, then, by definition, for sufficiently large k, we have

$$|x_{k+1} - x^*| \simeq \lambda |x_k - x^*|^\alpha$$

Or, denoting $e_k = |x_k - x^*|$, we have

$$e_{k+1} \sim \lambda e_k^\alpha$$

where the left-hand side is the error at the next step, λ is a factor, and the right-hand side is the (error at the previous step)$^\alpha$. For $\alpha = 1$, we speak of 'linear' convergence, for $\alpha = 2$ of 'quadratic' convergence, and $\alpha = 3$ of 'cubic' convergence, etc.

Different methods usually converge at different rates. Does this make a difference? The experiment we will do is to apply different numerical methods to

$$f(x) = x^2 - 2 = 0$$

with solutions $x^* = \pm\sqrt{2}$ and observe the value $e_k = |x_k - x^*|$. The methods we will apply are

(a) Newton's
(b) Secant
(c) Regula-Falsi

In-class presentation: Rates of convergence.

Remark 4.6. The above definition is not always applicable to analyze convergence behavior for some methods, although the sequences under consideration converges reasonably fast. Therefore, we sometimes employ the following extended definition of convergence order and convergence rate. We say that a sequence (x_k) converges to x^* with at least order α and convergence rate λ if there exists a sequence (ε_k) such that

$$|x_k - x^*| \leq \varepsilon_k \quad \text{for all } k$$

and the sequence (ε_k) converges to zero with order $\alpha > 0$ and rate $\lambda > 0$ according to the above "simple" definition.

With this remark at hand, we can complete the convergence analysis of the bisection method started in the last lecture. We have already shown that if f is continuous and $f(a_0) \cdot f(b_0) \leq 0$, then the sequence (x_k) of iterates obtained from the bisection method converges to some x^* such that $f(x^*) = 0$. We have the following convergence behaviour for the bisection method.

Proposition 4.7. *The bound for the error of the bisection method iterates is linear convergent (order* 1*) with rate* 1/2*.*

Proof. We have

$$|x_k - x^*| \leq \frac{1}{2^{k+1}}|b_0 - a_0| = B_k$$
$$|x_{k+1} - x^*| \leq \frac{1}{2^{k+2}}|b_0 - a_0| = B_{k+1}$$

Since $|B_{k+1}| = 1/2|B_k|$ for all k, the sequence (B_k) converges to 0 as $k \to \infty$, and B_k is linear convergent with convergence rate 1/2. □

Remark 4.8. Definition 4.4 is in general not applicable for the Bisection method as the following example of Minghao Pan shows. Let $f(x) = x - 2/7$ be defined on the interval $[0, 1]$, and denote by e_{3m+1} the error at the $3m+1$th iterate. Then one can prove that $e_{3m+2} > e_{3m+1}$.

4.1. Theoretical results for Newton's method.

- Present and then apply results for fixed point problems to Newton's method.
- Discuss the general idea behind the fixed point theory results (applied Taylor series).
- Give an alternative and complete proof of convergence for Newton's method under slightly weaker conditions (not based on fixed point theory).

Definition 4.9. A *fixed point problem* is a problem where one seeks solutions to a problem of the form

$$x = g(x)$$

where $g \colon \mathbb{R} \to \mathbb{R}$ is a function. A solution x^* such that $x^* = g(x^*)$ is called a *fixed point* of g.

Remark 4.10. In the Atkinson handout, the solutions x^* are called roots of $x = g(x)$ and unfortunately for us, α is used to denote a solution and p is used to denote the order of convergence. We will continue to use x^* to denote solutions of the equations and α to denote the order of convergence.

Remark 4.11. You can imagine fixed points of g as the intersections of the graph of g with the diagonal in the plane passing through the first and third quadrants.

Definition 4.12. A *fixed point iteration* is an iteration of the form $x_{k+1} = g(x_k)$, $k = 0, 1, 2, \ldots$.

"Fixed point theory" consists of results concerning the behaviour of fixed point iterates. For example, the theorem we will be using using to obtain a convergence result for Newton's method (cf. [1, Theorem 2.8]):

Theorem 4.13. *Assume x^* is a solution of $x = g(x)$ and $g(x)$ is α times continuously differentiable for all x near x^* for some $\alpha \geq 2$. Furthermore assume*

$$g'(x^*) = g''(x^*) = \ldots = g^{(\alpha-1)}(x^*) = 0 \tag{4.1}$$

Then if x_0 is chosen sufficiently close to x^*, the iteration

$$x_{k+1} = g(x_k), \quad k \geq 0$$

will have order of convergence α and

$$\lim_{k\to\infty} \frac{x^* - x_{k+1}}{(x^* - x_k)^\alpha} = (-1)^{\alpha-1}\frac{g^{(\alpha)}(x^*)}{\alpha!}$$

Remark 4.14. If we insert absolute values, then the convergence rate will be

$$\lambda = \left|\frac{g^\alpha(x^*)}{\alpha!}\right|$$

How do we apply the fixed point theory to obtain a result for Newton's method? Observe that Newton's method iteration for finding roots of $f(x) = 0$,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \tag{4.2}$$

is a fixed point iteration for the fixed point problem

$$x = x - \frac{f(x)}{f'(x)}$$

or if you wish

$$g(x) = x \quad \text{where} \quad g(x) = x - \frac{f(x)}{f'(x)} \tag{4.3}$$

If we assume that $f'(x^*) \neq 0$, then any solution x^* to $x^* = g(x^*)$ will be a solution to $f(x^*) = 0$. Convergence results for Newton's method (4.2) will follow directly from convergence results for the equivalent fixed point problem (4.3).

What are the convergence results for $x = g(x)$ where $g(x) = x - f(x)/f'(x)$? In order to apply Theorem 4.13, we need two things

(1) Determine restrictions on f that insure g is sufficiently differentiable.
(2) Determine how many of the conditions $\frac{dg^{(\alpha)}}{dx}(x^*) = 0$ for $\alpha = 1, 2, \ldots$ we need.

Let's work out (2), and then determine (1). We have

$$\frac{dg}{dx} = \frac{d}{dx}\left(x - \frac{f(x)}{f'(x)}\right) = \frac{f(x)f''(x)}{[f'(x)]^2}$$

by the quotient rule. By another application of the quotient rule, we have the following second derivative

$$\frac{d^2g}{dx^2} = \frac{d^2}{dx^2}\left(x - \frac{f(x)}{f'(x)}\right) = \frac{f''(x)}{f'(x)} + \frac{f(x)f'''(x)}{[f'(x)]^2} - \frac{2f(x)[f''(x)]^2}{[f'(x)]^3}.$$

So at x^*, we have $f(x^*) = 0$. We check the other assumptions of the fixed point theorem.

- If $f'(x^*) \neq 0$, and $f'(x)$ and $f''(x)$ exist and are continuous in a neighborhood of x^*, then g is continuously differentiable and $\frac{dg}{dx}(x^*) = 0$.
- If $f'(x^*) \neq 0$, $f'(x)$,$f''(x)$, and $f'''(x)$ exist and are continuous in a neighborhood of x^*, then g is twice continuously differentiable and

$$\frac{d^2g}{dx^2}(x) = \frac{f''(x)}{f'(x)}.$$

We have the following convergence result for Newton's method based on fixed point theory.

Theorem 4.15. *Assume that $f(x)$ is three times continuously differentiable for all x sufficiently close to x^* where x^* is a solution to $f(x^*) = 0$ and $f'(x^*) \neq 0$. Then there exists a $\delta > 0$ such that for all initial values $x_0 \in [x^* - \delta, x^* + \delta]$, the Newton iterates converge to x^* and*

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \left|\frac{f''(x^*)}{2f'(x^*)}\right|$$

That is, Newton's method converges quadratically ($\alpha = 2$) with convergence rate $\lambda = \left|\frac{f''(x)}{2f'(x)}\right|$.

Remark 4.16. If x_0 is outside of $(x^* - \delta, x^* + \delta)$, we have no information, and it may or may not converge to x^*. If x_0 is in this neighborhood, x_k is also in this neighborhood, and the convergence is quadratic.

Unlike the Bisection method (or Regula Falsi), one can only guarantee convergence of Newton's method if x_0 is sufficiently close to x^*. If one has a conceptual understanding of how Newton's method works, then the requirement x_0 is sufficiently close to x^* is completely understandable.

Let summarize what we did:

- Express iteration as a fixed point iteration $x_{k+1} = g(x_k)$ (identify g).
- Determine expressions for $\frac{dg^{(\alpha)}}{dx}$ for $\alpha = 1, 2, \ldots$.
- Determine the differentiability required to insure $\frac{dg^{(\alpha)}}{dx}$ is continuously differentiable for needed amount of α.
- Apply Theorem 4.13, or if $\frac{dg}{dx}(x^*) \neq 0$, verify that $\left|\frac{dg}{dx}(x^*)\right| < 1$, and apply [1, Theorem 2.7].

What's in the Atkinson notes leading up to Theorem 4.13? A collection of results concerning convergence results and their proofs for fixed point iterations $x_{k+1} = g(x_k)$. These results are of interest by themselves and they are combined to lead to a proof of Theorem 4.13. We are not going to go over these results in detail, instead, we sketch the 'idea' behind the results, so that if you do take the time to go over the results, they will "make sense". We are concerned with how the sequence of iterates (x_k) generated by a fixed point iteration, $x_{k+1} = g(x_k)$, behave. Assume that x^* is a solution of the fixed point problem so that $x^* = g(x^*)$. Now if we subtract the equation $x_{k+1} = g(x_k)$ from the equation $x^* = g(x^*)$, what we get is

$$x_{k+1} - x^* = g(x_k) - g(x^*) \tag{4.4}$$

Now let's use Taylor's formula about x^*:

$$g(x_k) = g(x^*) + g'(x^*)(x_k - x^*) + \frac{g''(x^*)}{2}(x_k - x^*)^2 + \ldots + R$$

which we can rewrite as

$$g(x_k) - g(x^*) = g'(x^*)(x_k - x^*) + \frac{g''(x^*)}{2}(x_k - x^*)^2 + \ldots + R$$

By applying (4.4), we get

$$x_{k+1} - x^* = g'(x^*)(x_k - x^*) + \frac{g''(x^*)}{2}(x_k - x^*)^2 + \ldots + R$$

Denoting the error $\tilde{e}_k = x_k - x^*$, we have

$$\tilde{e}_{k+1} = g'(x^*)\tilde{e}_k + \frac{g''(x^*)}{2}(\tilde{e}_k)^2 + \frac{g'''(x^*)}{3!}(\tilde{e}_k)^3 + \ldots + R$$

What does one expect?

(i) If $g'(x^*) \neq 0$, $|g'(x^*)| < 1$, and x_0 is sufficiently close to x^*, then

$$\tilde{e}_1 \simeq g'(x^*)\tilde{e}_0$$

so that $|g'(x^*)| < 1$ implies that $|\tilde{e}_1| < |\tilde{e}_0|$, and similarly $\tilde{e}_2 \simeq g'(x^*)\tilde{e}_1$ so that $|g'(x^*)| < 1$ implies that $|\tilde{e}_2| < |\tilde{e}_1|$, and so on, such that $\tilde{e}_{k+1} \simeq g'(x^*)\tilde{e}_k$ so that $|g'(x^*)| < 1$ implies that $|\tilde{e}_{k+1}| < |\tilde{e}_k|$ for all $k = 0, 1, 2, \ldots$. In this case, we expect *linear* convergence with convergence rate $\lambda = |g'(x^*)|$ (see [1, Theorem 2.7] for a proof that what's expected does occur).

(ii) If $g'(x^*) = 0$, and x_0 is sufficiently close to x^*, we have

$$\tilde{e}_1 \simeq \frac{g''(\xi_0)}{2}\tilde{e}_0^2$$

for some ξ_0 between x^* and x_0 (this comes from the remainder term in Taylor's formula), implying $\tilde{e}_1 \simeq \left(\frac{g''(\xi_0)}{2}\tilde{e}_0\right)\tilde{e}_0$, and so if we choose x_0 sufficiently close to x^* such that $|\frac{g''(\xi_0)}{2}| < 1$, then $|\tilde{e}_1| < |\tilde{e}_0|^2$. If we continue the iteration, then we arrive at $\tilde{e}_{k+1} \simeq \frac{g''(\xi_k)}{2}\tilde{e}_k^2$ for all $k = 0, 1, 2, \dots$. So that we expect a *quadratic* convergence with $\lambda = |g''(x^*)/2|$. The proof of [1, Theorem 2.8] tells you that what's expected does occur.

The use of this way of analyzing fixed point iteration may be answered by the following sample question: Consider the iteration $x_{k+1} = x_k - \gamma f(x_k)$ for some $\gamma \in \mathbb{R}$ fixed. Suppose that x^* is a root of f and $f'(x^*) \neq 0$.

(a) If (x_k) converges, what is the expected order of convergence?
(b) What are the restrictions on γ that need to be satisfied to insure convergence?

We have

$$\begin{aligned} x_{k+1} &= x_k - \gamma f(x_k) \\ x^* &= x^* - \gamma f(x^*) \end{aligned}$$

This results in

$$\begin{aligned} x_{k+1} - x^* &= (x_k - x^*) - \gamma[f(x_k) - f(x^*)] \\ &= (x_k - x^*) - \gamma[f'(x^*)(x_k - x^*) + \frac{f''(x^*)}{2}(x_k - x^*)^2 + \dots + R] \end{aligned}$$

From which if x_0 is close enough to x^*, we can expect to have

$$\tilde{e}_{k+1} \simeq [1 - \gamma f'(x^*)]\tilde{e}_k$$

So the expected order of convergence is 1, and requiring $|1 - \gamma f'(x^*)| < 1$ we could verify convergence.

We provide another proof of convergence for Newton's iterates under slightly weaker assumptions on f than we needed in order to use fixed point theory to establish convergence order and convergence rate. For Newton's method we need an initial estimate x_0 for the root x^* of a function f which hopefully leads to convergence of the iterates (x_k) obtained from the recursion

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \dots \tag{4.5}$$

The approach is again based on a Taylor approximation of order 2:

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

for some ξ between x and x_k. Let $x = x^*$, then $f(x^*) = 0$, and let's solve the last equation for x^*:

$$x^* = x_k - \frac{f(x_k)}{f'(x_k)} - \frac{(x^* - x_k)^2}{2}\frac{f''(\xi_k)}{f'(x_k)}$$

where ξ_k is between x_k and x^*. We can drop the error term (the last term in the last equation), and recognize that the remaining is x_{k+1} from (4.5), so that

$$x^* - x_{k+1} = -(x^* - x_k)^2\frac{f''(\xi_k)}{2f'(x_k)}, \quad k \geq 0. \tag{4.6}$$

We have the following convergence result.

Theorem 4.17. *Assume that $f(x)$, $f'(x)$, $f''(x)$ are continuous for all x in some neighborhood of x^*, and assume $f(x^*) = 0$, $f'(x^*) \neq 0$. Then if x_0 is chosen sufficiently close to x^*, the iterates x_k, $k \geq 0$, of (4.5) converge to x^*. Moreover,*

$$\lim_{k\to\infty} \frac{x^* - x_{k+1}}{(x^* - x_k)^2} = -\frac{f''(x^*)}{2f'(x^*)}$$

proving that the iterates have a convergence order $\alpha = 2$ and convergence rate $\lambda = \left|\frac{f''(x^*)}{2f'(x^*)}\right|$.

Proof. Pick sufficiently small interval

$$I = [x^* - \epsilon, x^* + \epsilon]$$

on which $f'(x) \neq 0$ (this is possible by continuity of f'). By the extreme value theorem, let

$$M = \frac{\max_{x\in I} |f''(x)|}{2\min_{x\in I} |f'(x)|}$$

From (4.6) we have

$$|x^* - x_1| \leq M|x - x_0|^2$$
$$M|x^* - x_1| \leq (M|x^* - x_0|)^2$$

Pick x_0 such that $|x^* - x_0| \leq \epsilon$ and $M|x^* - x_0| < 1$. Then $M|x^* - x_1| < 1$, and

$$M|x^* - x_1| \leq M|x^* - x_0|$$

which implies

$$|x^* - x_1| \leq \epsilon.$$

We can apply the same argument to $x_1, x_2, \ldots$ inductively, showing that $|x^* - x_k| \leq \epsilon$ and $M|x^* - x_k| < 1$ for all $k \geq 1$. To show convergence, use (4.6) to give

$$|x^* - x_{k+1}| \leq M|x^* - x_k|^2$$
$$M|x^* - x_{k+1}| \leq (M|x^* - x_k|)^2$$

and inductively,

$$M|x^* - x_k| \leq (M|x^* - x_0|)^{2^k}$$
$$|x^* - x_k| \leq \frac{1}{M}(M|x^* - x_0|)^{2^k}$$

Since $M|x^* - x_0| < 1$, this shows that $x_k \to x^*$ as $k \to \infty$. In (4.6), the unknown ξ_k was chosen to be between x_k and x^* implying that $\xi_k \to x^*$ as well as $k \to \infty$. Thus

$$\lim_{k\to\infty} \frac{x^* - x_{k+1}}{(x^* - x_k)^2} = -\lim \frac{f''(\xi_k)}{2f'(x_k)} = -\frac{f''(x^*)}{2f'(x^*)}.$$

□

4.2. Summary on root-finding algorithms.

- **Bisection**
  - Starting points: a_0, b_0 such that $f(a_0) \cdot f(b_0) \leq 0$
  - what it takes to continue: evaluate f
  - Order of convergence: 1
  - Guaranteed convergence: always.
- **Regula Falsi**
  - Starting points: a_0, b_0 such that $f(a_0) \cdot f(b_0) \leq 0$
  - what it takes to continue: evaluate f
  - Order of convergence: 1
  - Guaranteed convergence: always.
- **Newton**
  - Starting points: x_0
  - what it takes to continue: evaluate f, f'
  - Order of convergence: 2
  - Guaranteed convergence: x_0 sufficiently close to x^*.
- **Secant**

- Starting points: x_0, x_1
- what it takes to continue: evaluate f
- Order of convergence: $\frac{1+\sqrt{5}}{2}$
- Guaranteed convergence: x_0 sufficiently close to x^*.

Remark 4.18. The choice of method depends on the problem being solved - there is no universal "best" root finding method. Often methods are combined; e.g. do bisection, then Newton. There are also many other root finding methods (see root finding method) - some for general, some for specific problems (e.g. finding roots of polynomials). When you have to work with root finding methods - you shouldn't hesitate to do a little exploring of these other methods.

5. Representation of numbers and precision

In the following lectures, we discuss how computers represent numbers and which consequences this has on calculations.

Motivating questions:

(1) When using root finding methods one specifies a stopping tolerance. Why use values such as $1 \cdot 10^{-6}$, $1 \cdot 10^{-8}$, and not $1 \cdot 10^{-99}$?
(2) We've observed that estimating convergence order when the iterates are very close to the root is inaccurate. Why? (Or alternatively, what does it mean "it's due to finite precision"?)

To answer these questions requires an understanding of how computers represent numbers and the effect that machine representation has on calculations.

Facts of life:

(a) Computers use a finite number of bits to store numbers.
(b) The number of bits a computer uses to store numbers is both hardware and software dependent.

Remark 5.1. *Bits* are the smallest unit of storage ('atoms') of information. Each bit stores either a number 0 or 1. From Wikipedia: "A bit can be stored by a digital device or other physical system that exists in either of two possible distinct states. These may be the two stable states of a flip-flop, two positions of an electrical switch, two distinct voltage or current levels allowed by a circuit, two distinct levels of light intensity, two directions of magnetization or polarization, the orientation of reversible double stranded DNA, etc." One *byte* corresponds to 8 bits. For example, 01010010 is a byte representing an 8-string of 0 and 1s. So n bits can store 2^n different patterns, e.g. 1 byte can store 256 patterns.

Floating point numbers are stored by storing the information of the number in 'normalized' scientific notation. Given a fixed number of bits

$$\square \quad \square\square\ldots\square \quad \square\square\ldots\square$$

the first box (representing a bit) from the left is used to specify the *sign* (i.e. $\pm$), the next block of boxes is specified for the *exponent or characteristic* which determines the scale, and the last block is reserved to store the *significand or mantissa* which tells the precision (number of digits in the significand).

In the following, I will describe a double float which is based on a storage size of 64 bits for each number[2]. Matlab uses doubles to represent floating point numbers.

$$\underbrace{\square}_{\text{sign } s} \qquad \underbrace{\square\square\ldots\square}_{\text{exponent } c \text{ for } 2-12\text{th bits}} \qquad \underbrace{\square\square\ldots\square}_{\text{mantissa } f \text{ for } 13-64\text{th bits}}$$

[2]That is, we can represent the whole real line with 2^{64} different patterns.

Due to binary type of storage, numbers are represented in base 2. So the 52 binary digits of the mantissa correspond to approximately $16-17$ decimal digits. Exponent of 11 binary digits gives a range from 0 to $2^{11}-1 = 2047$ where half of them are used for positive exponents (1024) and the other half negative exponents (1023). This leads to the following normalized scientific notation

$$(-1)^s(1+f)2^{c-1023}$$

Consider

$$0 \quad 10000000011 \quad 1011100100010\dots 0$$

Thus $s = 0$ which means the number is positive. The exponential part gives

$$c = 1\cdot 2^0 + 1\cdot 2^1 + 0\cdot 2^2 + \dots 0\cdot 2^9 + 1\cdot 2^{10} = 1027$$

The mantissa reads as

$$f = 1\cdot\left(\frac{1}{2}\right)^1 + 1\cdot\left(\frac{1}{2}\right)^3 + 1\cdot\left(\frac{1}{2}\right)^4 + 1\cdot\left(\frac{1}{2}\right)^5 + 1\cdot\left(\frac{1}{2}\right)^8 + 1\cdot\left(\frac{1}{2}\right)^{12}$$

Putting everything together, we have

$$(-1)^2(1+f)2^{c-1023} = (-1)^0 2^{1027-1023}(1+(\frac{1}{2}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\frac{1}{256}+\frac{1}{4096})) = 27.56640625$$

Now the next smallest number is

$$0 \quad 10000000011 \quad 1011100100001\dots 1$$

and the next largest number is

$$0 \quad 10000000011 \quad 1011100100010\dots 1$$

This means the the machine number 27.56640625 not only represents itself but possibly also half of the numbers between itself and the next largest machine number and half of the numbers between itself and the next smallest machine number, that is a whole interval of real numbers!

In double, the smallest normalized positive number representable is

$$\begin{aligned}0 \quad 00000000001 \quad 0000000000000000000000000000000000000000000000000000 \\ = 2^{-1022}(1+0) \simeq 0.22251\times 10^{-307}\end{aligned}$$

and the largest normalized positive number is

$$\begin{aligned}0 \quad 11111111110 \quad 1111111111111111111111111111111111111111111111111111 \\ = 2^{1023}(2-2^{-52}) \simeq 0.17977\times 10^{309}\end{aligned}$$

Remark 5.2. Numbers smaller than the smallest representable number lead to an *underflow* and are automatically set 0, and numbers larger than the largest representable number lead to an *overflow* and usually the computation stops.

We have also

$$\begin{array}{lll}
0 & 000000000000000000000000000000000000000000000000000000000000000 & := +0 \\
1 & 000000000000000000000000000000000000000000000000000000000000000 & := -0 \\
0 & 111111111110000000000000000000000000000000000000000000000000000 & := +\infty \\
1 & 111111111110000000000000000000000000000000000000000000000000000 & := -\infty
\end{array}$$

Remark 5.3. The output NaN ('not a number') results from having invalid operations such as $0/0$, $\infty\times 0$, or $\sqrt{-1}$. Usually, one of the machine numbers is also reserved for NaN.

Remark 5.4. Here is an excellent online overview of IEEE Standard 754 Floating Point Numbers (the system which was introduced above) and is considered the international standard for storing numbers on computers

One way to think about how numbers are stored is

$$.d_1 d_2 \cdots d_m \times 10^{e_1 e_2 \cdots e_q} \tag{5.1}$$

where $d_i = 0, \ldots, 9$. So we have m digits in the mantissa and q digits in the exponent[3]. So if you consider a number line that shows all the representable numbers of the form (5.1) we would see a line with 'gaps' of increasing size with increasing scale (i.e. higher exponents) because numbers of the form (5.1) don't fill up the real number line completely. **Make a picture**

What are the consequences:

(1) For a given real number x_T, there may be (and usually is) an error made when converting x_T to a machine representable number. **Make a picture**
(2) Errors in the evaluation of arithmetic operators $(+, -, \times, \div)$ due to inexact representation of operands and/or inexact implementation of arithmetic operators.
(3) Errors in higher level operations caused by inexact representations.

In discussing errors, let us remind us about the difference between absolute and relative errors.

Definition 5.5. Let x_T be the "true" or "exact" value, and x_A the approximate value. Then the *absolute error* is

$$|x_T - x_A|$$

and *relative error* is

$$\frac{|x_T - x_A|}{|x_T|}.$$

Generally, when doing calculations one is interested in relative errors. For example, if $x_T = 8000$ and $x_A = 8010$, then one would consider 8010 a good approximation of 8000 since

$$\frac{|x_T - x_A|}{|x_T|} = \frac{10}{8000} = .00125$$

The relative error is less than .2%. One is not overly concerned with $|x_T - x_A| = 10$.[4]

Let start by answering question (1) above. To quantify the size of the error due to inexact representation, we derive an error bound for the relative error

$$\frac{|x_T - x_A|}{|x_T|}$$

Assume m digits in the mantissa, and assume 'chopping' is used when converting a number to its machine representation:

$$\begin{aligned} x_T &= .d_1 d_2 \cdots \times 10^{e_1 e_2 e_3} \\ x_A &= .d_1 d_2 \cdots d_m \times 10^{e_1 e_2 e_3} \end{aligned}$$

So we obtain x_A by just truncating the mantissa to m digits. The exponent remains the same.

Now let us compute the relative error the representation (5.1):

$$\begin{aligned} \frac{|x_T - x_A|}{|x_T|} &= \frac{.d_{m+1} d_{m+2} \cdots \times 10^{e_1 e_2 \cdots e_q - m}}{.d_1 d_2 \cdots \times 10^{e_1 e_2 \cdots e_q}} \\ &\leq \frac{10^{(e_1 e_2 \cdots e_q - m)+1}}{10^{e_1 e_2 \cdots e_q}} \\ &\leq 10^{-m+1} \end{aligned}$$

[3]Here and in the following, we change to a decimal system (base 10) to make the discussion easier.

[4]For example, if you would like to buy a house which costs half a million euros, and the seller tells you to pay half a million and 10 euros, then since 10 is such a small fraction of 500000 it doesn't really matter.

So we have

$$\frac{|x_T - x_A|}{|x_T|} \le 10^{-m+1}$$

where m is the number of decimal digits of the mantissa used in the machine representation. If one rounds, then the error is $1/2 \times 10^{-m+1}$.

Conclusion: Due to the finite number of bits used to store numbers,

(a) One expects inexact representation of numbers.
(b) A bound for the relative error of the machine representation is 10^{-m+1} (chopping) or $(1/2) \times 10^{-m+1}$ (rounding) when storing numbers with m decimal digits in the mantissa.

Remark 5.6. In Matlab, we use a double float number system, which results as we use binary digits in a representation error of approximately 10^{-16}.

Question: Can one do an experiment to determine the number of decimal digits in the mantissa? Sure: Experimentally estimate ε such that $1+\varepsilon = 1$. This ε is called the *unit round-off* or *machine epsilon*. It is the largest positive number so that $1+\varepsilon = 1$.

Example 5.7. Let $m = 5$ in (5.1).

ε	$1+\varepsilon$
.1	$.11000 \times 10^1$
.01	$.10100 \times 10^1$
.001	$.10010 \times 10^1$
.0001	$.10001 \times 10^1$
.00001	$.10000 \times 10^1 = 1$

Estimate of $\varepsilon = 10^{-5}$. Moreover, if you think about it, the largest value such that $1+\varepsilon = 1$ will be $\varepsilon = .00009999999\cdots = .0001$. So the unit round-off will be 1×10^{-4}. Note that unit round-off= $10^{-m+1} \equiv$ relative error in machine representation. So we have $m = -\log_{10}(\varepsilon)+1$. Given a desired precision, we can determine the length of the mantissa m.

Conclusion: You can experimentally estimate the unit round-off by looping over ε's that get increasingly smaller, or you can often extract ε from the computer. Knowing ε, you can determine the number of significant digits (either decimal or binary).

Two consequences of working with a finite number of bits are

(1) limitations on the accuracy of the machine representation of floating point numbers
(2) introduction of errors in the results of arithmetic operations $(+, -, \cdot, \div)$

A note about (1):

Remark 5.8. Generally, machines use base 2 representation so what we do, or derive, are limitations in the base 10 that correspond to the base 2 limitations. The actual representation and corresponding limitations are in base 2, we interpret those limitations in base 10; be aware that correspondence isn't always exact! In particular, deriving a bound for the error in representation due to machine representation based on number of digits in the mantissa base 10 will give different values than using a number of digits in the mantissa base 2.[5]

Recall that we denoted by x_T the exact or true value of a real number and by x_A its machine representation or approximate value which results from a round-off operations (e.g. truncating (or chopping) or rounding). A bound for the relative error in representing a given

[5]Base 10 results usually overestimate error bound.

floating point number:

$$\frac{|x_T - x_A|}{|x_T|} \leq 10^{-m+1} \quad m = \# \text{ digits in the mantissa (base 10)}$$
$$\frac{|x_T - x_A|}{|x_T|} \leq 2^{-m'+1} \quad m' = \# \text{ digits in the mantissa (base 2)}$$

These bounds won't necessarily be the same.

Unit round-off=largest positive number ε so $1+\varepsilon = 1$. You can estimate it by doing an experiment where you add ever smaller numbers $\varepsilon_0, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k$ and stop when $1 + \varepsilon_k = 1$. Note that if we let $x_T = 1 + \varepsilon$ and $x_A = 1$, then since

$$\frac{|x_T - x_A|}{|x_T|} = \frac{\varepsilon}{1+\varepsilon}$$

so a bound for the relative error in machine representation must be greater than $\frac{\varepsilon}{1+\varepsilon} \sim \varepsilon$. Usually, one just assume the bound for the relative error is ε.

Next we want study consequence (2). Assuming that errors are when converting floating point numbers to their machine representation - how do these errors influence the result of arithmetic observations? If we assume that the arithmetic is done exactly (which is not an unreasonable assumption) we would like to determine how the accuracy of the output of the operation depends on the accuracy of the input. Given two number x_T and y_T, then define

$$\text{Rel}(x_A) = \frac{x_T - x_A}{x_T} \quad \text{the relative error in representing } x_T \text{ by } x_A$$
$$\text{Rel}(y_A) = \frac{y_T - y_A}{y_T} \quad \text{the relative error in representing } y_T \text{ by } y_A$$

We would like understand how $\text{Rel}(x_A \circ y_A)$ can determined as a function of $\text{Rel}(x_A)$ and $\text{Rel}(y_A)$. That is,

$$\frac{(x_T \circ y_T) - (x_A \circ y_A)}{(x_T \circ y_T)} = \text{ what function of relative error of inputs, Rel}(x_A) \text{ and Rel}(y_A)$$

where $\circ \in \{+, -, \cdot, \div\}$. In the following derivation, we will be employing the useful relation

$$x_A = x_T - x_T \text{Rel}(x_A)$$

- Derivation for multiplication:

$$\begin{aligned}
\text{Rel}(x_A y_A) &= \frac{x_T y_T - x_A y_A}{x_T y_T} \\
&= \frac{x_T y_T - (x_T - x_T \text{Rel}(x_A))(y_T - y_T \text{Rel}(y_A))}{x_T y_T} \\
&= \frac{x_T y_T \text{Rel}(y_A) + x_T y_T \text{Rel}(x_A) - x_T y_T \text{Rel}(x_A)\text{Rel}(y_A)}{x_T y_T} \\
&= \text{Rel}(x_A) + \text{Rel}(y_A) - \text{Rel}(x_A)\text{Rel}(y_A)
\end{aligned}$$

Usually, $|\text{Rel}(x_A)| \ll 1$ and $|\text{Rel}(y_A)| \ll 1$, so that $\text{Rel}(x_A)\text{Rel}(y_A)$ is negligible and we have the result

$$\text{Rel}(x_A y_A) \sim \text{Rel}(x_A) + \text{Rel}(x_B)$$

- Derivation for division:

$$\begin{aligned}
\mathrm{Rel}(x_A/y_A) &= \frac{x_T/y_T - x_A/y_A}{x_T/y_T} \\
&= \frac{x_T/y_T - (x_T - X_T\mathrm{Rel}(x_A))/(y_T - y_T\mathrm{Rel}(y_A))}{x_T/y_T} \\
&= 1 - (x_T - X_T\mathrm{Rel}(x_A))/(y_T - y_T\mathrm{Rel}(y_A))x_T/y_T \\
&= 1 - (1 - \mathrm{Rel}(x_A))/(1 - \mathrm{Rel}(y_A)) \\
&= \frac{\mathrm{Rel}(x_A) - \mathrm{Rel}(y_A)}{(1 - \mathrm{Rel}(y_A))}
\end{aligned}$$

Usually, $|\mathrm{Rel}(y_A)| \ll 1$ so $(1 - \mathrm{Rel}(y_A)) \sim 1$ and we have the result

$$\mathrm{Rel}(x_A/y_A) \sim \mathrm{Rel}(x_A) - \mathrm{Rel}(y_A)$$

- Derivation for addition: Assume x_T and y_T both positive or negative.

$$\begin{aligned}
\mathrm{Rel}(x_A + y_A) &= \frac{x_T + y_T - (x_A + y_A)}{x_T + y_T} \\
&= \frac{(x_T - x_A) + (y_T - y_A)}{x_T + y_T} \\
&= \mathrm{Rel}(x_A)\left(\frac{x_T}{x_T + y_T}\right) + \mathrm{Rel}(y_A)\left(\frac{y_T}{x_T + y_T}\right)
\end{aligned}$$

- Derivation for subtraction: Assume x_T and y_T both positive or negative.

$$\begin{aligned}
\mathrm{Rel}(x_A - y_A) &= \frac{x_T - y_T - (x_A - y_A)}{x_T - y_T} \\
&= \frac{(x_T - x_A) - (y_T - y_A)}{x_T - y_T} \\
&= \mathrm{Rel}(x_A)\left(\frac{x_T}{x_T - y_T}\right) + \mathrm{Rel}(y_A)\left(\frac{y_T}{x_T - y_T}\right)
\end{aligned}$$

Conclusion:

For $+, \cdot, \div$, the relative error in the result of operation due to relative errors in the input and sum of the input errors. There isn't a significant amplification of the errors. However, for subtraction (with x_T and y_T of same sign),

$$\mathrm{Rel}(x_A - y_A) \sim \mathrm{Rel}(x_A)\left(\frac{x_T}{x_T - y_T}\right) + \mathrm{Rel}(y_A)\left(\frac{y_T}{x_T - y_T}\right)$$

the relative error can be large if $x_T - y_T$ is small, e.g. subtraction of nearly equal numbers (also referred to as *catastrophic cancellation*).

What can we do about it? Try to create alternate formulas that allow one to determine the same value, but without requiring the subtraction of nearly equal numbers.

Examples 5.9. • For example, the evaluation of $1 - e^x$ when x is near 0 may result in a significant inaccuracy. A solution is to use a sufficiently large Taylor series when x is small:

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots$$

so that

$$1 - e^x = x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots$$

- Express polynomials in nested form, see [2, pages 23-25].

In your homework, you have to come up for an alternative to deal with

- $1-\cos(x)$: you can try to find an appropriate trigonometric identity
- Quadratic formula: you can have a look at [2, pages 22-23] or the Wikipedia article Loss of significance.

We will analyze another example of computing

$$\frac{x^2}{1-e^{-x^2}}$$

as $x \to 0$. This limit is by l'Hôpital's rule 1. For $1-e^{-x^2}$ an identity is unlikely. How about using Taylor's formula. To keep things simple; use an expansion of e^s and evaluate it at $s=-x^2$:

$$e^{-x^2} = 1 + (-x^2) + \frac{(-x^2)^2}{2!} + \frac{(-x^2)^3}{3!} + \dots$$

When x is small, then the series converges very quickly. So

$$1 - e^{-x^2} = (-x^2) + \frac{(-x^2)^2}{2!} + \frac{(-x^2)^3}{3!} + \dots$$

doesn't have subtraction of nearly equal numbers. How many terms should we evaluate? Remainder for the expansion with p terms is

$$R = e^{\xi}\frac{(-x^2)^p}{p!}, \quad \xi \in [-x^2, 0]$$

When x is small $1-e^{-x^2} \sim x^2$, so choose p so that the relative error size $= |R|/x^2 < \varepsilon$, or

$$\frac{\frac{(-x^2)^p}{p!}}{x^2} < \varepsilon, \quad \text{or} \quad \frac{(-x^2)^p}{p!} < \varepsilon$$

Say $x \in [0, .01]$, and $\varepsilon = 10^{-16}$, then we need to choose p such that $(10^{-4})^{p-1} < \varepsilon p!$ (should give results $< \varepsilon$ for all $x \in [0, .01]$. We find that $p = 5$ is good. So we can replace the original problem of evaluating

$$\frac{x^2}{1-e^{-x^2}}$$

for x small, by the corrected power series evaluation of

$$\frac{x^2}{x^2 - \frac{x^4}{2!} + \frac{x^6}{3!} - \frac{x^8}{4!} + \frac{x^{10}}{5!}}.$$

Finally, the primary consequence of using a finite number of bits to store integers is that their maximal size is limited! Check the following BBC article from 2015, for some real ‘numerical disasters’ occurring from this limitation. The following are some situations where this consequence can be felt in scientific computing:

(1) loop counters: loop bounds are usually integers, there are limits to the range of the index for a storage in 4 bytes signed we have a limitation of ± 2 billion.
(2) sizes of arrays; memory addresses are stored as integers so a 4 byte unsigned integers can only address 4 GB of memory
(3) evaluation of factorial
(4) representation of money or time, for example 4 byte signed, storing 1/100 of a cent leads to a maximal positive amount of $214,748.3647$
(5) pseudo-random numbers are typically generated and stored using integers which leads to a finite number of pseudo-random numbers

You may google “numerical + disasters” to find articles on some significant consequences due to computers using integers with a finite size.

6. Polynomial interpolation

Plan:

- Identify the task/problem
- Show how to find polynomial interpolants (by hand) using a divided difference table
- Discuss theory, other ways of obtaining interpolants, use of interpolants, etc.

Problem: Given $n+1$ distinct data points $(x_i, f(x_i))$, $i = 0, \dots, n$, determine a polynomial $p(x)$ such that $p(x_i) = f(x_i)$. **Draw a picture**

Remark 6.1. Assuming that the data comes from the evaluation of a ‘hidden’ (continuous) function, the Weierstraß approximation theorem provides a theoretical framework: For any continuous function $f\colon [a, b] \to \mathbb{R}$ and $\epsilon > 0$ there exists a polynomial $p\colon [a, b] \to \mathbb{R}$ such that

$$\sup_{x\in[a,b]} |f(x) - p(x)| < \epsilon.$$

A constructive proof of this result can be obtained by Bernstein polynomials (check the proof in this Wikipedia article, it uses a cute probabilistic argument).

We give a construction by hand. Let’s start with an example, then we give a formula generalizing this example, then a more general formula, and then discuss one more example, before we start with more theoretic investigations.

Remark 6.2. In the text book, we start with [2, Section 3.3].

Example 6.3. We are given the data in the table

x_i	$f(x_i)$
1	1
2	3
3	1

We want to construct a polynomial $p(x)$ that interpolates these points. Solution is obtained by using coefficients determined by a divided difference table:

x_i	$f(x_i)$		
1	**1**		
2	3	$\frac{3-1}{2-1} = \mathbf{2}$	
3	1	$\frac{1-3}{3-2} = -2$	$\frac{-2-2}{3-1} = \mathbf{-2}$

An interpolating polynomial is then given by

$$p(x) = \mathbf{1} + \mathbf{2}(x-1) - \mathbf{2}(x-1)(x-2)$$

Check: $p(1) = 1$, $p(2) = 1 + 2 = 3$ and $p(3) = 1 + 4 - 4 = 1$. Notice that $p(x)$ is a 2nd degree polynomial that interpolates the data.

Let’s see what happened. Denoting $1 = f(x_0)$, $2 = f[x_0, x_1]$ and $-2 = f[x_0, x_1, x_2]$, we have

$$p(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$

where

x_0	$f(x_0)$		
x_1	$f(x_1)$	$f[x_0, x_1] := \frac{f(x_1)-f(x_0)}{x_1-x_0}$	
x_2	$f(x_2)$	$f[x_1, x_2] := \frac{f(x_2)-f(x_1)}{x_2-x_1}$	$f[x_0, x_1, x_2] := \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$

The values in the 3rd column are called *1st divided differences*, and the ones in the 4th column *2nd divided differences.* Such a table is called a *Newton divided differences table.* Let's derive a general formula for $(n+1)$ data points.

The general formula: Given $(n+1)$ distinct data values $(x_i, f(x_i))$, a polynomial interpolant for the data is given by

$$p(x) = f(x_0) + \sum_{k=1}^{n} f[x_0, x_1, \dots, x_k](x - x_0)(x - x_1)\dots(x - x_{k-1})$$

where $f[x_0, \dots, x_k]$ is the kth divided difference defined recursively by

$$\begin{aligned} f[x_k] &= f(x_k) \\ f[x_i, x_{i+1}, \dots, x_{i+m}] &= \frac{f[x_{i+1}, \dots, x_{i+m}] - f[x_i, x_{i+1}, \dots, x_{i+n-1}]}{x_{i+m} - x_i} \end{aligned}$$

which can be organized in a divided difference table as follows:

x_0	$f(x_0)$	1st divided differences	2nd divided differences	$\dots$	
x_1	$f(x_1)$	$f[x_0, x_1]$			
x_2	$f(x_2)$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$		
x_3	$f(x_3)$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Remark 6.4. You can remember this by “difference of adjacent differences divided by difference of ‘base’ ”. Or, you can observe that each divided difference results from a triangle where the numerator is the difference between last two values to the left of the right edge of the triangle, and the base is the difference of the edges of the left side of that triangle.

Example 6.5. We compute another example, where we don't write down all the expressions.

x_i	$f(x_i)$	1st divided differences	2nd divided differences	3rd divided differences
1	1			
2	2	1		
3	5	3	1	
4	16	11	4	1

Which leads to the interpolating polynomial

$$p(x) = 1 + (x-1) + (x-1)(x-2) + (x-1)(x-2)(x-3)$$

Let's check:

$$\begin{aligned} p(1) &= 1 \\ p(2) &= 1 + 1 = 2 \\ p(3) &= 1 + 2 + 2 = 5 \\ p(4) &= 1 + 3 + 6 + 6 = 16. \end{aligned}$$

What's the idea behind Newton's divided differences? "Incrementally increasing the degree of the polynomial interpolant":

$$\begin{aligned}
\text{1 data point: } & p(x) = f(x_0) \\
\text{2 data points: } & p(x) = f(x_0) + d_1(x - x_0) \\
& \quad \to \text{add term that vanishes at } x_0\text{, and find } d_1 \text{ so } p(x_1) = f(x_1) \\
\text{3 data points: } & p(x) = f(x_0) + d_1(x - x_0) + d_2(x - x_0)(x - x_1) \\
& \quad \to \text{add term that vanishes at } x_0 \text{ and } x_1\text{, and find } d_2 \text{ so } p(x_2) = f(x_2) \\
& \quad \vdots \\
n+1 \text{ data points: } & p(x) = f(x_0) + \sum_{k=1}^{n} d_k(x - x_0)(x - x_1)\cdots(x - x_{k-1}) \\
& \quad \to d_k\text{'s term are determined in increasing order by requiring } p(x_k) = f(x_k)
\end{aligned}$$

So it's a clever idea to incrementally add terms that don't change what's been created for the previous data points. From this simple idea, one can work out that the d_k's should be the divided differences and the recursion relation that determines the divided differences. You'll have to search through other numerical analysis texts to find a proof (see e.g. the handout "Divided differences.pdf" on CCLE which is from [3, pages 113-115]).

In the remainder of this part on polynomial interpolation, we want to address the following questions:

(1) Are there other methods of constructing the interpolating polynomial?
(2) Does there always exist a polynomial that will interpolate the data? If so, will it be unique?
(3) What are the properties of the interpolating polynomial?
   (a) If the values $f_i = f(x_i)$ are those of a function f, how good an approximation is $p(x)$ to $f(x)$ for $x \neq x_i$? That is, how accurate is $p(x)$ for values in between the data points.
   (b) How does the accuracy depend on f?
   (c) How does the accuracy depend on the location of the x_i's?

We start with question (1), since knowing alternate methods of construction are useful for answering theoretical questions. We'll learn two more methods of constructing an interpolating polynomial:

Method #2: The method of *undetermined coefficients.*
Method #3: *Lagrange interpolating polynomials.*

Method #2 Undetermined coefficients: Given $(n + 1)$ data points $(x_i, f(x_i))$, to find a polynomial $p(x)$ such that $p(x_i) = f(x_i)$, $i = 0, \ldots, n$ do the following:

(i) Assume a representation of the polynomial with at least $n + 1$ "free" coefficients;
(ii) Set up and solve equations for the coefficients that ensure $p(x_i) = f(x_i)$ for $i = 0, \ldots, n$.

Let's try this out with our initial example:

x_i	$f(x_i)$
1	1
2	3
3	1

As for (i), assume $p(x) = a_0 + a_1 x + a_2 x^2$ which is a quadratic polynomial so we have 3 "free" coefficients = # of date points. As for the equations in (ii):

$$\begin{aligned} p(1) = 1 \quad &\Rightarrow \quad a_0 + a_1(1)^1 + a_2(1)^2 = 1 \\ p(2) = 3 \quad &\Rightarrow \quad a_0 + a_1(2)^1 + a_2(2)^2 = 3 \\ p(3) = 1 \quad &\Rightarrow \quad a_0 + a_1(3)^1 + a_2(3)^2 = 1 \end{aligned}$$

This leads to the following linear equation

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

Solve a_0, a_1, a_2 with $a_0 = -5$, $a_1 = 8$ and $a_2 = -2$ so

$$p(x) = -5 + 8x - 2x^2$$

Now in general: Given $n+1$ data points $(x_i, f(x_i))$, $i = 0, \ldots, n$, if one assumes

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

then the equations take the form:

$$A\vec{a} = \vec{f}$$

with

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \ldots & \vdots & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}$$

By basic results in linear algebra, we know that there will be an interpolating polynomial if $\det(A) \neq 0$. The matrix A is called a *Vandermonde matrix.* (For more see the related Wikipedia article on Vandermonde matrix). Later we'll prove that if $x_i \neq x_j$ whenever $i \neq j$ (distinct data), then $\det(A) \neq 0$.

Remark 6.6. Tip for homework: Using the method of undetermined coefficients is a way of transforming questions about existence of an interpolant into questions about the solvability of linear equations.

Method #3 Lagrange interpolating polynomials: Given a set of data points $\{x_i : i = 0, \ldots, n\}$ such that $x_i \neq x_j$ whenever $i \neq j$ (distinct data points), then the kth Lagrange interpolating polynomial is given by

$$l_k(x) = \frac{\prod_{i=0, i\neq k}^{n} (x - x_i)}{\prod_{i=0, i\neq k}^{n} (x_k - x_i)}, \quad k = 0, 1, \ldots, n.$$

These polynomials have the following properties:

(i) $\deg(l_k(x)) = n$ for all k
(ii) l_k vanishes at all data points x_i except x_k:

$$l_k(x_i) = \begin{cases} 0 & i \neq k \\ 1 & i = k \end{cases}$$

Now given $n+1$ distinct data points $(x_i, f(x_i))$ $i = 1, \dots, n$, a polynomial of degree at most n that interpolates the data is given by

$$p(x) = \sum_{k=0}^{n} f(x_k) l_k(x).$$

Inspection:

- $\deg(p) \leq$ maximal degree of $l_k(x) = n$
- $p(x_i) = \sum_{k=0}^{n} f(x_k) l_k(x_i) = f(x_i) l_i(x_i) = f(x_i)$ (only non-zero term occurs when $k = i$ which follows from property (ii) above).

Now that we have seen 3 methods how to construct an interpolating polynomial, we want to answer question (2) about existence and uniqueness:

Theorem 6.7. *If $(x_i, f(x_i))$, $i = 0, \dots, n$, are $n+1$ distinct data points, then there exists a unique polynomial of at most degree n that interpolates the data.*

Proof. For existence, use the Lagrange interpolant

$$p(x) = \sum_{k=0}^{n} f(x_k) l_k(x)$$

which is a polynomial of at most degree n such that $p(x_i) = f(x_i)$ for all $i = 0, \dots, n$.

To prove uniqueness, we employ a contradiction argument. Indeed, suppose there exist $q(x) \neq p(x)$ such that $q(x_i) = f(x_i)$ for all $i = 0, \dots, n$ and $\deg(q) \leq n$. Then the polynomial $r(x) = p(x) - q(x)$ is a polynomial of degree $\leq n$ and $r(x_i) = p(x_i) - q(x_i) = 0$ for all $i = 0, 1, \dots, n$. So $r(x)$ has $n+1$ many roots. However, this contradicts the Fundamental Theorem of Algebra which states that a non-trivial real polynomial of degree k has at most k many roots. □

Conclusions:

- There will always be an interpolating polynomial of degree $\leq n$ (given distinct data points).
- If one restricts to polynomials of degree $\leq n$, then the interpolating polynomial is unique.

References

[1] K.E. Atkinson. *An introduction to numerical analysis*, Wiley, 1978.
[2] R. L. Burden, D. J. Faires and A. M. Burden. *Numerical Analysis*, Cengage Learning, 10th edition, 2015.
[3] W. Cheney and D. Kincaid. *Numerical Mathematics And Computing*, Cengage Learning, 7th edition, 2012.

Department of Mathematics, UCLA, Los Angeles CA 90095-1555, USA.
E-mail address: jasgar@math.ucla.edu