

## LECTURE NOTES “APPLIED NUMERICAL METHODS” (151 A)

*These lecture notes are based on lecture notes of Chris Anderson. Many thanks to Chris for this! Corrections and comments are always welcome, preferably send to me directly 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”
- 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 differentiation
- (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)

## 2. SOLVING NONLINEAR EQUATIONS

We start with [2, Chapter 2].

We recall the definition of a linear function.

**Definition 2.1.** A function  $f: \mathbb{R} \rightarrow \mathbb{R}$  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  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$  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  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$  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  $f: \mathbb{R} \rightarrow \mathbb{R}$  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  $l: \mathbb{R} \rightarrow \mathbb{R}$ , this means that  $l(x) = ax + b$  for some constant scalar  $b \in \mathbb{R}$ .
- b) For a vector-valued function  $L: \mathbb{R}^m \rightarrow \mathbb{R}^n$ , 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:

$$\begin{aligned} f(x + y) &= 3(x + y) = 3x + 3y = f(x) + f(y) \\ f(cx) &= 3cx = c(3x) = cf(x) \end{aligned}$$

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

$$\begin{aligned} A(\vec{x} + \vec{y}) &= A\vec{x} + A\vec{y} \\ A(c\vec{x}) &= cA(\vec{x}) \end{aligned}$$

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  $\kappa$  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  $\mu$  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

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

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  $f: \mathbb{R} \rightarrow \mathbb{R}$ , given a value  $b \in \mathbb{R}$  find  $x \in \mathbb{R}$  such that  $f(x) = b$ . For vector-valued functions  $\vec{F}: \mathbb{R}^m \rightarrow \mathbb{R}^n$ , given a vector  $\vec{b} \in \mathbb{R}^n$  find  $\vec{x} \in \mathbb{R}^m$  such that  $\vec{F}(\vec{x}) = \vec{b}$ .

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

$$\frac{1}{12} \frac{r 5000 e^{r5}}{(e^{r5} - 1)} = 180$$

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):

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

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?

Plot  $\rightsquigarrow$  Zoom  $\rightsquigarrow$  Plot  $\rightsquigarrow$  Zoom  $\rightsquigarrow$  ...

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  $f(x) = 0$ .** 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  $x$  and  $y$  (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  $\epsilon$ ' (which you may choose to be  $10^{-10}$ ))

is much easier. To illustrate the situation, you can draw an interval of length  $2\epsilon$  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  $f: [a, b] \rightarrow \mathbb{R}$  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

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

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

*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](#).  $\square$

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

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

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

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

where

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

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  $|R| \leq C|x - x_0|^{n+1}$  for  $x$  sufficiently close to  $x_0$  (in Taylor’s theorem above, you may choose  $C \geq \frac{M_{n+1}}{(n+1)!}$ ). Here is Wikipedia article on [Big-O notation](#), see also the additional material “Asymptotic Methods in Analysis” of de Bruijn 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  $n$ th approximation (starting from  $n = 0$ ) is  $O(|x - x_0|^{n+1})$ . If  $|x - x_0| < 1$ , then  $|x - x_0|^{n+1} \rightarrow 0$  as  $n \rightarrow \infty$ . This implies that we get better approximations as  $n \rightarrow \infty$  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  $f(x) = 0$ .** 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<sup>1</sup>:

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

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

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

<sup>1</sup>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  $e_k = x_k - x^*$

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

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  $f: [a, b] \rightarrow \mathbb{R}$  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  $x_k \rightarrow x^*$ , then  $|x_k - \xi| \rightarrow 0$ , that is  $\xi$  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

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

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  $\lim_{k \rightarrow \infty} x_k = x^*$  or  $x_k \rightarrow x^*$  as  $k \rightarrow \infty$  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  $\lim_{k \rightarrow \infty} x_k = x^*$ . 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  $x_k \rightarrow x^*$ ?
- (b) If a method converges, that is  $x_k \rightarrow x^*$ , then can we quantify how fast  $x_k \rightarrow x^*$ ?
  - (i) Can one determine theoretically how fast  $x_k \rightarrow x^*$ ?
  - (ii) How does one estimate how fast  $x_k \rightarrow x^*$  from experimental results?

Plan:

- (1) Definitions of order of convergence and asymptotic error constant that are needed to quantify how fast  $x_k \rightarrow x^*$ .
- (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  $f: [a, b] \rightarrow \mathbb{R}$  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  $f(a) \cdot f(b) \leq 0$ , 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  $f(a_0) \cdot f(b_0) \leq 0$ . 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  $k$ th step we have

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

*Proof.* The proof goes by induction.

- At step 0: Since  $f(a_0) \cdot f(b_0) \leq 0$ , the intermediate value theorem applies and there exists  $x^* \in [a_0, b_0]$ , and  $x_0 = \frac{a_0 + b_0}{2}$  satisfies  $|x_0 - x^*| \leq |b_0 - a_0|/2$ .
- At step  $k$ : If  $f(a_k) \cdot f(b_k) \leq 0$ , 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  $f(a_{k+1}) \cdot f(b_{k+1}) \leq 0$ . Otherwise,  $f(a_k) \cdot f(b_k) \leq 0$  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  $k \rightarrow k + 1$ .

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  $k \rightarrow \infty$ . □

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  $\alpha$  and  $\lambda$  such that

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

then we say that  $x_k$  converges to  $x^*$  with *convergence order*  $\alpha$  and *convergence rate*  $\lambda$ .

**Remark 4.5.** See [Rates of convergence](#) for additional reading and some examples.

If  $x_k \rightarrow x^*$  with order  $\alpha$  and rate  $\lambda$ , 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,  $\lambda$  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  $\alpha$  and convergence rate  $\lambda$  if there exists a sequence  $(\varepsilon_k)$  such that

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

and the sequence  $(\varepsilon_k)$  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

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

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

**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  $g: \mathbb{R} \rightarrow \mathbb{R}$  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,  $\alpha$  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  $\alpha$  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  $x_{k+1} = g(x_k)$ ,  $k = 0, 1, 2, \dots$

“Fixed point theory” consists of results concerning the behaviour of fixed point iterates. For example, the theorem we will be 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  $\alpha$  times continuously differentiable for all  $x$  near  $x^*$  for some  $\alpha \geq 2$ . Furthermore assume*

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

*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  $\alpha$  and*

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

**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  $f(x) = 0$ ,

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

is a fixed point iteration for the fixed point problem

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

or if you wish

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

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  $\alpha = 1, 2, \dots$  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

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

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  $\alpha = 1, 2, \dots$
- Determine the differentiability required to insure  $\frac{dg^{(\alpha)}}{dx}$  is continuously differentiable for needed amount of  $\alpha$ .
- 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

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

Now let’s use Taylor’s formula about  $x^*$ :

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

which we can rewrite as

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

By applying (4.4), we get

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

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

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

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  $k = 0, 1, 2, \dots$ . 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  $\xi_0$  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  $\gamma$  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

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

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  $\xi$  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  $\xi_k$  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

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

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,

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



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

$$\begin{aligned} |x^* - x_1| &\leq M|x - x_0|^2 \\ M|x^* - x_1| &\leq (M|x^* - x_0|)^2 \end{aligned}$$

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  $x_1, x_2, \dots$  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

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

and inductively,

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

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

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

□

#### 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

□ □□...□ □□...□

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<sup>2</sup>. Matlab uses doubles to represent floating point numbers.


  
 sign  $s$       exponent  $c$  for 2 – 12th bits      mantissa  $f$  for 13 – 64th bits

<sup>2</sup>That is, we can represent the whole real line with  $2^{64}$  different patterns.

$$(-1)^s(1+f)2^{c-1023}$$
$$0 \quad 10000000011 \quad 1011100100010 \dots 0$$
$$c = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + \dots 0 \cdot 2^9 + 1 \cdot 2^{10} = 1027$$
$$f = 1 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^4 + 1 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^8 + 1 \cdot \left(\frac{1}{2}\right)^{12}$$
$$(-1)^2(1+f)2^{c-1023} = (-1)^0 2^{1027-1023} (1 + (\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096})) = 27.56640625$$
$$0 \quad 10000000011 \quad 1011100100001 \dots 1$$
$$0 \quad 10000000011 \quad 1011100100010 \dots 1$$

In double, the smallest normalized positive number representable is

and the largest normalized positive number is

**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.

[illegible]

**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

$$.d_1d_2\cdots d_m \times 10^{e_1e_2\cdots e_q} \quad (5.1)$$

where  $d_i = 0, \dots, 9$ . So we have  $m$  digits in the mantissa and  $q$  digits in the exponent<sup>3</sup>. 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.

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.
- (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  $|x_T - x_A| = 10$ .<sup>4</sup>

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:

$$\begin{aligned} x_T &= .d_1d_2\cdots \times 10^{e_1e_2e_3} \\ x_A &= .d_1d_2\cdots d_m \times 10^{e_1e_2e_3} \end{aligned}$$

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):

$$\begin{aligned} \frac{|x_T - x_A|}{|x_T|} &= \frac{.d_{m+1}d_{m+2}\cdots \times 10^{e_1e_2\cdots e_q-m}}{.d_1d_2\cdots \times 10^{e_1e_2\cdots e_q}} \\ &\leq \frac{10^{(e_1e_2\cdots e_q-m)+1}}{10^{e_1e_2\cdots e_q}} \\ &\leq 10^{-m+1} \end{aligned}$$

So we have

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

<sup>3</sup>Here and in the following, we change to a decimal system (base 10) to make the discussion easier.

<sup>4</sup>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.

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  $\varepsilon$  such that  $1 + \varepsilon = 1$ . This  $\varepsilon$  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).

$\varepsilon$	$1 + \varepsilon$
.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  $\varepsilon = .0000999999 \dots = .0001$ . So the unit round-off will be  $1 \times 10^{-4}$ . Note that this 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  $\varepsilon$ 's that get increasingly smaller, or you can often extract  $\varepsilon$  from the computer. Knowing  $\varepsilon$ , 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 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 this 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.<sup>5</sup>

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 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.

<sup>5</sup>Base 10 results usually overestimate error bound.

Unit round-off=largest positive number  $\varepsilon$  so  $1 + \varepsilon = 1$ . You can estimate it by doing an experiment where you add ever smaller numbers  $\varepsilon_0, \varepsilon_1, \varepsilon_2, \dots, \varepsilon_k$  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  $\varepsilon$ .

Next we want to study consequence (2). Assuming that errors occur when converting floating point numbers to their machine representation - how do these errors influence the result of arithmetic observations? If we assume that 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 numbers  $x_T$  and  $y_T$ , then define

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

We would like understand how  $\text{Rel}(x_A \circ y_A)$  can be determined as a function of  $\text{Rel}(x_A)$  and  $\text{Rel}(y_A)$ :

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

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:

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

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

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

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

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

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

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

**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),

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

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  $x \rightarrow 0$ . 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  $\pm 2$  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  $f: [a, b] \rightarrow \mathbb{R}$  and  $\epsilon > 0$  there exists a polynomial  $p: [a, b] \rightarrow \mathbb{R}$  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  $k$ th divided difference defined recursively by

$$f[x_k] = f(x_k)$$

$$f[x_i, x_{i+1}, \dots, x_{i+m}] = \frac{f[x_{i+1}, \dots, x_{i+m}] - f[x_i, x_{i+1}, \dots, x_{i+m-1}]}{x_{i+m} - x_i}$$

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

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

**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:

$$p(1) = 1$$

$$p(2) = 1 + 1 = 2$$

$$p(3) = 1 + 2 + 2 = 5$$

$$p(4) = 1 + 3 + 6 + 6 = 16.$$

What’s the idea behind Newton’s divided differences? “Incrementally increasing the degree of the polynomial interpolant”:

1 data point:  $p(x) = f(x_0)$

2 data points:  $p(x) = f(x_0) + d_1(x - x_0)$

→ add term that vanishes at  $x_0$ , and find  $d_1$  so  $p(x_1) = f(x_1)$

3 data points:  $p(x) = f(x_0) + d_1(x - x_0) + d_2(x - x_0)(x - x_1)$

→ add term that vanishes at  $x_0$  and  $x_1$ , and find  $d_2$  so  $p(x_2) = f(x_2)$

$\vdots$

$n + 1$  data points:  $p(x) = f(x_0) + \sum_{k=1}^n d_k(x - x_0)(x - x_1) \cdots (x - x_{k-1})$

→  $d_k$ ’s term are determined in increasing order by requiring  $p(x_k) = f(x_k)$

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  $p(x_i) = f(x_i)$ ,  $i = 0, \dots, n$  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  $i = 0, \dots, n$ .

Let’s try this out with our initial example:

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

As for (i), assume  $p(x) = a_0 + a_1x + a_2x^2$  which is a quadratic polynomial so we have 3 “free” coefficients = # of data points. As for the equations in (ii):

$$p(1) = 1 \quad \Rightarrow \quad a_0 + a_1(1)^1 + a_2(1)^2 = 1$$

$$p(2) = 3 \quad \Rightarrow \quad a_0 + a_1(2)^1 + a_2(2)^2 = 3$$

$$p(3) = 1 \quad \Rightarrow \quad a_0 + a_1(3)^1 + a_2(3)^2 = 1$$

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  $(x_i, f(x_i))$ ,  $i = 0, \dots, n$ , 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

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

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  $\{x_i: i = 0, \dots, n\}$  such that  $x_i \neq x_j$  whenever  $i \neq j$  (distinct data points), then the  $k$ th Lagrange interpolating polynomial is given by

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

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  $(x_i, f(x_i))$   $i = 0, \dots, n$ , 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.  $\square$

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.

Suppose we have 3 data points. We know that there is a polynomial of degree  $\leq 2$  that interpolates the data. Here we may distinguish two cases:

- The points do not lie on a line: the unique interpolating polynomial is quadratic.
- The points lie on a line: the unique interpolating polynomial is an affine function, in other words, if we would try to find a quadratic function  $p(x) = a_0 + a_1x + a_2x^2$  to fit the data, we would always discover that  $a_2 = 0$ .

However, one could always find an enormous amount of higher order polynomials that fit the data. For example, for any fourth data point distinct from the initial three, we would find a unique polynomial of degree at most 3 interpolating the data. In particular, this polynomial would interpolate the initial three data points. As such a polynomial exists due to theorem [6.7](#) for any fourth distinct data point, we find uncountably many polynomials of degree  $> 2$  that interpolate the initial three data points.

What other conclusion can be drawn:

- The polynomial created using Newton divided differences is a polynomial of degree  $\leq n$  that interpolates the data. The method of construction makes no assumption about the labeling of the data, and since the polynomial that results from any labeling of the data is equal to the polynomial from any other labeling, we conclude that it doesn't matter how one sets up the divided difference table.
- Let  $x_i \neq x_j$  for all  $i \neq j$ , and  $i = 0, 1, \dots, n$ . Let  $A$  be the Vandermonde matrix

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}$$

We claim that  $\det(A) \neq 0$ . By Theorem [6.7](#), for any family of values  $f(x_i) = f_i$ ,  $i = 0, 1, \dots, n$ , there exists a unique polynomial

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

This implies that the problem

$$A \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix} = \vec{f}$$

has a unique solution for any  $\vec{f}$ . By basic results in linear algebra, this is equivalent to  $\det(A) \neq 0$ .

Given a set of  $n + 1$  distinct data points it therefore makes sense to consider *the* interpolating polynomial (of degree  $\leq n$ ). Next, we would like to answer the question of how accurate is the polynomial interpolant. For a given point  $x$ , can we determine a bound for  $|p(x) - f(x)|$  where

$p(x)$  is the interpolant through  $n + 1$  distinct data points  $\{x_i, f(x_i)\}$ ? So we are interested in finding a bound for the interpolation error.

Plan:

- Present 2 theorems whose result gives a bound which will be presented without proof.
- Discuss the practical use for the bounds for each theorem and conclusions about the error in polynomial interpolation that can be drawn from them.

The first theorem is taken from the [2], and corresponds to Theorem 3.3 there.

**Theorem 6.8.** Suppose  $x_0, x_1, \dots, x_n$  are distinct points in the interval  $[a, b]$  and  $f \in C^\infty[a, b]$ . Then for all  $x \in [a, b]$  there exists a value  $\xi(x) \in [a, b]$  (depending on  $x$ ) such that

$$f(x) = p(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n)$$

where  $p(x)$  is the interpolating polynomial to the data  $(x_i, f(x_i))$ ,  $i = 0, 1, \dots, n$  with  $\deg(p) \leq n$ .

A proof is given in [2]. On CCLE, you find a cleaner proof by Atkinson [1]. What is the practical use of this theorem? If we know a bound for

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

then

$$\max_{x \in [a, b]} |f(x) - p(x)| \leq \frac{M}{(n+1)!} \max_{x \in [a, b]} \left| \prod_{i=0}^n (x - x_i) \right|$$

So if we can bound

$$\max_{x \in [a, b]} \left| \prod_{i=0}^n (x - x_i) \right|$$

we can obtain a uniform bound. Unfortunately, finding a completely general theoretical bound that can be computed for arbitrary sets of points  $\{x_i\}$  is not easy, however, there is a nice bound if the points are equispaced. (**Draw a picture**). Formally, given an interval  $[a, b]$ , then  $n + 1$  equispaced points in  $[a, b]$  are obtained by  $x_k = a + kh$  for  $k = 0, \dots, n$  where  $h = \frac{b-a}{n}$ . From [3], we have

**Theorem 6.9.** Let  $f$  be a function such that  $f^{(n+1)}(x)$  is continuous over  $[a, b]$  and satisfies  $\max_{x \in [a, b]} |f^{(n+1)}(x)| \leq M$ . Let  $p(x)$  be the polynomial of at most degree  $n$  that interpolates  $f$  at  $n + 1$  equispaced nodes in  $[a, b]$  including the endpoints. Then

$$\max_{x \in [a, b]} |f(x) - p(x)| \leq \frac{M}{4(n+1)} h^{n+1} \quad (6.1)$$

where  $h = \frac{b-a}{n}$ .

Before justifying the theorem (a proof by Cheney and Kincaid is posted on CCLE as [ErrorboundProofCheney.pdf](#), let's discuss a simple example.

**Example 6.10.** We consider linear interpolation. In this case, we have two points  $x_0 < x_1$ , and we set  $h = x_1 - x_0$ . If  $\max_{x \in [x_0, x_1]} |f''(x)| \leq M$ , then Theorem 6.8 yields a bound

$$\max_{x \in [x_0, x_1]} |f(x) - p(x)| \leq \frac{M}{2} \max_{x \in [a, b]} |x - x_0| |x - x_1|$$

where  $p$  is the linear interpolant of  $f$ . A geometric argument (**draw it**), shows that

$$\max_{x \in [x_0, x_1]} |x - x_0| |x - x_1| \leq \frac{1}{4} h^2$$

which yields a uniform bound

$$\max_{x \in [x_0, x_1]} |f(x) - p(x)| \leq \frac{Mh^2}{8}$$

Notice that the error bound for linear interpolation is second order in the mesh width  $h$  (= distance between points). For example

$$\begin{aligned} h = .1 &\Rightarrow \text{error bound} \simeq .01 \\ h = .01 &\Rightarrow \text{error bound} \simeq .0001 \end{aligned}$$

**Remark 6.11.** Notice that the error bound (6.8) is not ‘sharp’ as it doesn’t vanish when  $x = x_0$  or  $x = x_1$  as does the error bound in Theorem 6.8.

What is behind the bound (6.1)? Using the bound from Theorem 6.8

$$\max_{x \in [a, b]} |f(x) - p(x)| \leq \frac{M}{(n+1)!} \max_{x \in [a, b]} \prod_{i=0}^n |x - x_i|$$

together with

**Lemma 6.12.** If  $\{x_i\}$  are  $n+1$  equispaced points, i.e.  $x_k = a + kh$ ,  $k = 0, 1, \dots, n$  with  $h = \frac{b-a}{n}$ , then it holds that

$$\max_{x \in [a, b]} \prod_{i=0}^n |x - x_i| \leq \frac{1}{4} h^{n+1} n!$$

We will take advantage of both of the error bounds. Let’s use the second bound (6.1) in an application as follows.

**Example 6.13.** Suppose one seeks a polynomial approximation to  $e^{-x}$  for  $x \in [0, 1]$  using equispaced polynomial interpolation. Using (6.1), what is the fewest number of equispaced points that will ensure

$$\max_{x \in [0, 1]} |f(x) - p(x)| \leq 1 \times 10^{-6}?$$

We have

$$M = \max_{x \in [0, 1]} |f^{n+1}(x)| = \max_{x \in [0, 1]} \left| \frac{d^{n+1}}{dx^{n+1}} (e^{-x}) \right| = \max_{x \in [0, 1]} |e^{-x}| \leq 1$$

So we just need to determine  $n$  so that

$$\frac{1}{4(n+1)} h^{n+1} = \frac{1}{4(n+1)} \left( \frac{1}{n} \right)^{n+1} \leq 1 \times 10^{-6}$$

since  $h = \frac{b-a}{n} = \frac{1}{n}$ .

Let’s find  $n$  so

$$\frac{1}{4(n+1)} \left( \frac{1}{n} \right)^{n+1} = 1 \times 10^{-6}$$

and then round up to the nearest integer. This looks like a root finding problem... One can try Bisection method “manually” as follows. When  $n = 1$  we have  $\frac{1}{8} - 1 \times 10^{-6} > 0$ , and when  $n = 10$ , then we have  $\frac{1}{4(10+1)} (10^{-1})^{11} - 1 \times 10^{-6} < 0$ . So  $n$  must lie between 0 and 10. For  $n = 5$ , we get

$$\frac{1}{4(n+1)} \left( \frac{1}{n} \right)^{n+1} - 1 \times 10^{-6} = 2.6666... \times 10^{-6} - 1 \times 10^{-6} > 0$$

So root  $> 5$ . We are close, so let's just try  $n = 6$ , and compute that

$$\frac{1}{4(6+1)} \left(\frac{1}{6}\right)^{6+1} - 1 \times 10^{-6} < 0$$

So  $n = 6$ , or 7 points!

This means that if one uses a 6th degree polynomial and 7 equispaced points,

$$|f(x) - p(x)| \leq 1 \times 10^{-6} \quad \text{for all } x \in [0, 1].$$

You'll be doing a similar problem in your homework.

Can we always use high order equispaced polynomial interpolation? To answer this question, let's recall the error bound for equispaced polynomial interpolation:

$$\max_{x \in [a, b]} |f(x) - p(x)| \leq \frac{M}{4(n+1)} h^{n+1} \quad (6.2)$$

where

$$M = \max_{x \in [a, b]} |f^{(n+1)}(x)|, \quad h = \frac{b-a}{n}.$$

The error bound (6.2) has two factors. So if

$$M = M_n = \max_{x \in [a, b]} |f^{(n+1)}(x)|$$

is uniformly bounded for all  $n$ , then the error in equispaced polynomial interpolation tends to zero as the number of points increases since we do have control on the size of the second factor

$$\frac{h^{n+1}}{4(n+1)}$$

However, the Runge function

$$f(x) = \frac{1}{1+x^2}$$

shows that the requirement that  $M_n$  be bounded (or at least slowly growing) can fail.

In class, we showed demonstrations in each case equispaced polynomial interpolation fails to produce an accurate approximation. These demonstrations by Prof. Chris Anderson are posted on CCLE. The demonstrations show the plot of the polynomial interpolant with 33 ( $= n+1$ ) equispaced points.

- For  $f(x) = \frac{1}{4}|x|$  failure is because  $f'(x)$  is not continuous over  $[a, b]$
- $f(x)$  = square wave function fails because  $f$  itself is not continuous over  $[a, b]$ .
- The Runge function

$$f(x) = \frac{1}{1+x^2}$$

is  $C^\infty$  (i.e. infinitely differentiable over  $[a, b]$ ), however its  $n$ th derivative grows quickly to infinity as  $n \rightarrow \infty$ . One could use the pointwise error bound

$$f(x) - p(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!} \prod_{i=0}^n (x - x_i)$$

to observe large errors. This is known as [Runge's phenomenon](#), see this Wolfram demonstration [Runge's phenomenon](#) where you can compare equispaced interpolation with Chebyshev interpolation which will be discussed later in this chapter<sup>6</sup>.

---

<sup>6</sup>There is an active field on research on this [Lebesgue constant](#)



**Remark 6.14.** From these examples, we observe that either the bound can’t be applied, or, it doesn’t tend to zero as  $n$  increases. A large error bound doesn’t necessarily mean that the actual error will be large, but it is usually suggestive, and as demonstrated in class, large errors can be observed for functions that aren’t sufficiently differentiable (e.g.  $f(x) = |x|$ ) or for functions that are differentiable, but whose high derivatives are very large, e.g. the Runge function  $1/(1+x^2)$ . From these examples, we conclude that with regard to high order equispaced polynomial interpolating “more” (that is, more nodes) isn’t necessarily better.

In your homework, you’ll be explaining what happens when the function one is interpolating has “noise”.

**Remark 6.15.** Here is a link to the YouTube channel of Prof. Wen Shen (Penn State) with lots of [videos](#) on numerical analysis topics some of which you might find helpful as an additional source of explanation and learning. One of her videos is posted below as an optional learning material for least squares approximation.

If one is just given function values  $f(x_i)$ , one may not know if the function is differentiable, or, if it is differentiable, how large the high derivatives are. So, a question arises, “what can be done to avoid the problems with high order (large  $n$ ) equispaced polynomial interpolation?” How can one create accurate approximations that get better as  $n \rightarrow \infty$  (i.e. you do more, you expect better results). We have a few possibilities:

- (1) Choose ordinates  $x_i$ ’s differently.
- (2) Use piecewise polynomial interpolation.
- (3) Use [least squares approximation](#) and or [orthogonal polynomial](#) expansions.
- (4) Combination of (1), (2), and (3).
- (5) Use functions other than polynomials to approximate a function such as [Fourier interpolants](#) or [kernel ridge regression](#).

In (3), (4) and (5), for an approximation  $q(x)$ , just seek  $q(x)$  so  $|q(x) - f(x_i)|$  is small, but not necessarily zero. These topics are discussed in 151B. We will discuss (1) and (2) in the following.

- (1) **Choose  $x_i$ ’s differently**<sup>7</sup>.

Recall the pointwise bound

$$|f(x) - p(x)| \leq \frac{M}{(n+1)!} \left| \prod_{i=0}^n (x - x_i) \right|$$

Since we don’t have control over  $M$  for increasing  $n$ , we could try to attempt to minimize the factor

$$\left| \prod_{i=0}^n (x - x_i) \right|$$

uniformly in  $x$ . In other words, does there exist a family  $\{\tilde{x}_i\}$  of  $n+1$  points such that

$$\max_{x \in [a,b]} \left| \prod_{i=0}^n (x - \tilde{x}_i) \right| \leq \max_{x \in [a,b]} \left| \prod_{i=0}^n (x - x_i) \right|$$

for any other choice  $\{x_i\}$  of  $n+1$  points?

If we assume that  $[a, b] = [-1, 1]$ , the answer is given by

$$\tilde{x}_i = \cos \left( \frac{2i+1}{2n+2} \pi \right), \quad i = 0, 1, \dots, n$$

The  $\{\tilde{x}_i\}$  are the roots of the  $(n+1)$ st Chebyshev polynomial.

---

<sup>7</sup>Covered in [2, Chapter 3, Section 8].

These points have the property that

$$\frac{1}{2^n} = \max_{x \in [-1, 1]} \left| \prod_{i=0}^n (x - \tilde{x}_i) \right| \leq \min_{\{x_i\}_{i=0}^n} \left[ \max_{x \in [-1, 1]} \left| \prod_{i=0}^n (x - x_i) \right| \right],$$

that is, they solve a mini-max problem. Any set of  $n + 1$  points other than  $\{\tilde{x}_i\}$  will have a larger maximum. They are the “best” or “optimal” points in the sense that the bound for which

$$\left| \prod_{i=0}^n (x - x_i) \right|$$

is the smallest. With these points, we get

$$|f(x) - p(x)| \leq \frac{M}{(n+1)!2^n}$$

Where does this result come from? It’s based upon the “mini-max” properties of [Chebyshev polynomials](#). A handout with proofs from [2] is posted on CCLE. Here is sketch of steps:

- (i) The Chebyshev polynomials can be defined by the recurrence relation

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad n = 1, 2, \dots$$

Put

$$\tilde{T}_n(x) = \frac{T_n(x)}{2^{n-1}}$$

which is a polynomial of degree  $n$  with leading term  $x^n$ .

- (ii)  $\tilde{T}_n(x)$  has the property that

$$\frac{1}{2^{n-1}} = \max_{x \in [-1, 1]} |\tilde{T}_n(x)| \leq \min_{\deg(Q) \leq n-1} \left[ \max_{x \in [-1, 1]} |x^n + Q(x)| \right]$$

Among all ‘monic’<sup>8</sup> polynomials  $P(x) = x^n + Q(x)$  with  $\deg(Q) \leq n - 1$ , the one with the smallest maximum is  $\tilde{T}_n(x)$ .

- (iii) We have

$$\prod_{i=0}^n (x - x_i) = x^{n+1} + Q'(x) \quad \text{with } \deg(Q') \leq n$$

So by (ii), with  $n$  replaced by  $n + 1$ , one should choose  $x_i$  so that

$$\prod_{i=0}^n (x - x_i) = \tilde{T}_{n+1}(x)$$

but this implies that  $x_i$  are the  $n + 1$  roots of  $T_{n+1}(x)$ , roots that one knows<sup>9</sup> are

$$\tilde{x}_i = \cos \left( \frac{2i+1}{2n+2} \pi \right), \quad i = 0, 1, \dots, n$$

Now what to do about general intervals  $[a, b]$ ? Use points in the same relative location. This can be achieved by mapping the points from  $[-1, 1] \rightarrow [a, b]$  by

$$l(x) = a + \frac{b-a}{2}(x+1)$$

<sup>8</sup>Leading term has coefficient 1.

<sup>9</sup>From the fact that an alternative definition of  $T_n(x)$  is

$$T_n(x) = \cos(n \arccos(x))$$

see [2].

Now use

$$\tilde{z}_i = a + \frac{b-a}{2}(\tilde{x}_i + 1), \quad \tilde{x}_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \quad i = 0, 1, \dots, n$$

(2) **Piecewise polynomial approximation.**

Idea: Instead of using a single high order polynomial over the whole interval  $[a, b]$ , use a collection of lower degree polynomials over subintervals. **Draw a picture** Why is this a reasonable idea? If we used degree  $m$  equispaced polynomial interpolation over each subinterval, then for the  $k$ th subinterval,  $I_k$ , one has

$$h_k = \frac{b-a}{Nm}$$

Then

$$|f(x) - p(x)| \leq \max_{x \in I_k} |f^{(m+1)}(x)| \frac{1}{4(m+1)} \left(\frac{b-a}{Nm}\right)^{m+1}$$

If we fix  $m$  (# points per subinterval) and let  $N \rightarrow \infty$ , then

$$|f(x) - p(x)| \leq \max_{x \in I_k} |f^{(m+1)}(x)| \times \left[ \frac{1}{4(m+1)} \left(\frac{1}{m}\right)^{m+1} \right] \times \left[ \frac{|b-a|}{N} \right]^{m+1}$$

where the first factor on the right-hand side of the previous inequality depends only on the first  $m$  derivatives, the second factor is a constant, and the last factor tends to 0 as  $N \rightarrow \infty$ . The error bound we obtain is

$$O(\tilde{h}^{m+1}), \quad \tilde{h} = \frac{b-a}{N} \text{ (subinterval size)}$$

Notice that as we add points, improvement in accuracy doesn't depend on the size of the derivatives of increasing order.

Also if the function is not sufficiently differentiable at some point, the error bound caused by this is localized to the subinterval that contains the problematic point.

**Draw picture**

Showed in class a demonstration of the difference between use of Chebyshev nodes and equispaced nodes for Runge function, see the file posted on CCLE.

## 7. NUMERICAL DIFFERENTIATION

Plan:

- Derivation of formulas
- Estimating the error using Taylor series (done in [2] - but not emphasized)

**The problem:** Given discrete data values  $(x_i, f(x_i))$  determine an approximation to  $f'(x)$  at a point  $\tilde{x}$ . **Draw a graph with 3 nodes** It is common practice to seek  $f'(x)$  at the location of the data points  $x_i$ , e.g.  $\tilde{x} = x_i$  for some  $i$ .

One method for obtaining approximations

- Construct an interpolating polynomial through  $(x_i, f(x_i))$
- Use the approximation

$$f'(x)|_{x=\tilde{x}} \simeq p'(x)|_{x=\tilde{x}}$$

We are familiar with 2 ways of representing the interpolating polynomial - Newton's divided differences and Lagrange interpolating polynomials. The book [2] uses Lagrange interpolating for deriving formulas - in the lecture, we'll use Newton divided differences.

Linear interpolation. Given two values  $(x_0, f(x_0))$  and  $(x_1, f(x_1))$ , we want to approximate  $f'(x_0)$  and  $f'(x_1)$ . Using a divided difference table, we find that the interpolating polynomial is

$$p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

We get

$$p'(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

and therefore the approximations

$$\begin{aligned} p'(x_0) &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ p'(x_1) &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

When  $x_1 = x_0 + h$ , we can rewrite the previous approximations as

$$\begin{aligned} f'(x_0) &= \frac{f(x_0 + h) - f(x_0)}{h} \rightsquigarrow \text{forward differences} \\ f'(x_1) &= \frac{f(x_1) - f(x_1 - h)}{h} \rightsquigarrow \text{backward differences} \end{aligned}$$

The forward differences coincide with the definition of the derivative, and so as  $h \downarrow 0$  (e.g.  $x_1 \rightarrow x_0$ ), then we expect the approximation to improve.

Quadratic interpolation. Lets assume equispaced data points  $x_0, x_1 = x_0 + h$  and  $x_2 = x_0 + 2h$ . What is an approximation to  $f'(x_1)$ ? Let  $f_0 = f(x_0)$ ,  $f_1 = f(x_1)$  and  $f_2 = f(x_2)$ . Using divided differences (and doing some reductions on the coefficients), we get an interpolant

$$\begin{aligned} p(x) &= f_0 + \left[ \frac{f_1 - f_0}{h} \right] (x - x_0) + \left[ \frac{f_2 - 2f_1 + f_0}{2h^2} \right] (x - x_0)(x - x_1) \\ &= f_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) \end{aligned}$$

where we denoted

$$d_1 = \frac{f_1 - f_0}{h} \quad \text{and} \quad d_2 = \frac{f_2 - 2f_1 + f_0}{2h^2}$$

Differentiating the interpolant, we get

$$p'(x) = d_1 + d_2(x - x_1) + d_2(x - x_0)$$

and evaluating this at  $x = x_1$ , we have

$$p'(x)|_{x=x_1} = d_1 + d_2(x_1 - x_0) = d_1 + d_2h$$

which is

$$\frac{f_1 - f_0}{h} + \frac{f_2 - 2f_1 + f_0}{2h^2}h = \frac{f_2 - f_0}{2h}$$

So we get the approximation

$$f'(x_1) \simeq \frac{f(x_2) - f(x_0)}{2h} \tag{7.1}$$

This approximation is usually specified using the data points

$$x_{-1} = x_0 - h \quad x_0 \quad x_1 = x_0 + h$$

So that (7.1) becomes

$$f'(x_0) \simeq \frac{f(x_0 + h) - f(x_0 - h)}{2h} \tag{7.2}$$

where the latter is known under the name ‘centered’ difference approximation. Why would one use (7.2) instead of the ‘usual’ forward differences. The answer is that (7.2) converges to

the exact values faster than the forward differences as  $h \rightarrow 0$ . This answer follows from the following error estimate of the accuracy of each approximation.

Error bound results. We will verify:

- (1) Assuming  $f$  is 2 times continuously differentiable in a neighborhood of  $x_0$ ,  $x \in [x_0, x_0 + \delta]$  where  $\delta > 0$ , then for all  $h \in (0, \delta]$ ,

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} \right| \leq Mh$$

where

$$M = \frac{\max_{x \in [x_0, x_0 + \delta]} |f''(x)|}{2}$$

In other words, the error is  $O(h)$ , and as  $h \rightarrow 0$ , convergence is 1st order in  $h$ .

- (2) Assuming that  $f$  is 3 times continuously differentiable in a neighborhood of  $x_0$ ,  $x \in [x_0 - \delta, x_0 + \delta]$ , then for all  $h \in (0, \delta]$ , we have

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0 - h)}{2h} \right| \leq Mh^2$$

where

$$M = \frac{\max_{x \in [x_0, x_0 + \delta]} |f'''(x)|}{6}$$

In other words, the error is  $O(h^2)$ , and as  $h \rightarrow 0$ , convergence is 2nd order in  $h$ .

Both results will follow from a derivation of an “asymptotic error expansion” in terms of  $h$ . The derivation relies on Taylor series of the form

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \frac{f'''(x_0)}{3!}h^3 + \dots + \frac{f^{(k+1)}(\xi)}{(k+1)!}h^{k+1}$$

for some  $\xi \in [x_0, x_0 + h]$ . Inserting the previous expansion in forward difference, we get

$$f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} = -\frac{f''(x_0)}{2!}h - \frac{f'''(x_0)}{3!}h^2 - \dots - \frac{f^{(k+1)}(\xi)}{(k+1)!}h^k$$

The right-hand side of the last equation is called *asymptotic error expansion*, where the *leading term* is  $\frac{f''(x_0)}{2!}h$ .

Now if we terminate the Taylor series that was resulted with a remainder term  $\frac{f''(\xi)}{2!}h^2$  for any  $h$  we get that there exists  $\xi \in [x_0, x_0 + h]$  such that

$$f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} = -\frac{f''(\xi)}{2}h$$

so that

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} \right| \leq Mh$$

where

$$M = \frac{\max_{x \in [x_0, x_0 + \delta]} |f''(x)|}{2}$$

What did we do to derive the error estimate:

- (1) Insert a Taylor expansion.
- (2) Cancel terms to obtain an asymptotic error expansion.
- (3) The leading term indicates the order of accuracy. To get an error bound insert Taylor series with an appropriate number of terms that are indicated by the order.

Let's do the same analysis for centered differences.

$$\begin{aligned}
 f'(x_0) &= \frac{f(x_0 + h) - f(x_0 - h)}{2h} \\
 &= f'(x_0) - \frac{f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \frac{f'''(x_0)}{3!}h^3 + \dots + \frac{f^{(k+1)}(\xi_1)}{(k+1)!}h^{k+1}}{2h} \\
 &\quad - \frac{f(x_0) - f'(x_0)h + \frac{f''(x_0)}{2!}h^2 - \frac{f'''(x_0)}{3!}h^3 + \dots + (-1)^{k+1} \frac{f^{(k+1)}(\xi_2)}{(k+1)!}h^{k+1}}{2h} \\
 &= -\frac{f'''(x_0)}{3!}h^2 - \frac{f^{(5)}(x_0)}{5!}h^4 - \frac{f^{(7)}(x_0)}{7!}h^6 - \dots - \frac{f^{(k+1)}(\xi_1)}{(k+1)!}h^k + (-1)^{k+1} \frac{f^{(k+1)}(\xi_2)}{(k+1)!}h^k
 \end{aligned}$$

We observe that the leading term  $\frac{f'''(x_0)}{3!}h^2$  is  $O(h^2)$ . If we had inserted a Taylor series with remainder  $\frac{f'''(x)}{3!}h^3$ , then we would obtain that there exist  $\xi_1 \in [x_0, x_0 + \delta]$  and  $\xi_2 \in [x_0 - \delta, x_0]$  such that

$$f'(x_0) - \frac{f(x_0 + h) - f(x_0 - h)}{2h} = -\left(\frac{f'''(\xi_1)}{12}h^2 + \frac{f'''(\xi_2)}{12}h^2\right)$$

If  $0 < h < \delta$ , then

$$\left|f'(x_0) - \frac{f(x_0 + h) - f(x_0 - h)}{2h}\right| \leq Mh^2 = O(h^2)$$

where

$$M = \frac{\max_{x \in [x_0 - \delta, x_0 + \delta]} |f'''(x)|}{6}$$

Since the exponent of  $h$  in the error bound is 2, and the error bound holds for all  $0 < h < \delta$  as  $h \rightarrow 0$ , one says “the order of the method is 2nd order” (or, it is often said, “the method is of 2nd order accuracy”).<sup>10</sup>

**Remark 7.1.** Note that the order of the leading term of the asymptotic error estimate determines the order of the method and knowing the form of the leading term enables one to determine the order of the Taylor series with remainder that's used to derive the error bound.

Let's summarize what we did for forward difference.

(1) Leading term is  $O(h) \rightsquigarrow$  first order method. Leading term is in terms of  $f'' \rightsquigarrow$  use

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

to obtain an error bound: If there exists  $\delta > 0$  such that  $f$  is twice continuously differentiable over  $[x_0, x_0 + \delta]$ , then for all  $h < \delta$ , we have

$$\left|f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h}\right| \leq \frac{Mh}{2!} \leq M'h = O(h), \quad M' = \frac{M}{2}, M = \max_{x \in [x_0, x_0 + \delta]} |f''(x)|$$

The approximation

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

has first order accuracy.

<sup>10</sup>In your homework - one is just asked to find the leading term of the asymptotic expansion of the error for the formulas one derives.

**7.1. Error estimation for numerical differentiation.** Next we would like to estimate the order accuracy based upon computational experiments. Why?

- Check interpolation
- Detect non-differentiability of function whose derivative is being approximated

Let  $D_h f$  denote an approximation of  $f'(x_0)$  obtained using a numerical differentiation formula with data point spacing  $h$ . We assume we know exact value of  $f'(x_0)$ . We also assume an asymptotic error expansion of the form

$$e_h = f'(x_0) - D_h f = c_p h^p + c_{p+1} h^{p+1} + \dots$$

where  $c_p$  are constants, possibly depending on  $x_0$ , but not on  $h$ . To determine the value of the order of the leading term  $p$ , one computes  $D_h f$  for two spacings, say  $\tilde{h}$  and  $\frac{\tilde{h}}{2}$ , to obtain  $D_{\tilde{h}} f$  and  $D_{\frac{\tilde{h}}{2}} f$  and the associated errors  $e_{\tilde{h}}$  and  $e_{\frac{\tilde{h}}{2}}$ . If  $\tilde{h}$  is sufficiently small, we get

$$\begin{aligned} \frac{e_{\tilde{h}}}{e_{\frac{\tilde{h}}{2}}} &= \frac{f'(x_0) - D_{\tilde{h}} f}{f'(x_0) - D_{\frac{\tilde{h}}{2}} f} = \frac{c_p \tilde{h}^p + c_{p+1} \tilde{h}^{p+1} + \dots}{c_p \left(\frac{\tilde{h}}{2}\right)^p + c_{p+1} \left(\frac{\tilde{h}}{2}\right)^{p+1} + \dots} \\ &\simeq \frac{c_p \tilde{h}^p}{c_p \left(\frac{\tilde{h}}{2}\right)^p} \\ &= 2^p \end{aligned}$$

Then we obtain

$$p = \log_2 \left( \frac{e_{\tilde{h}}}{e_{\frac{\tilde{h}}{2}}} \right)$$

To avoid problem with logarithm, one often uses

$$\log_2 \left( \frac{|e_{\tilde{h}}|}{|e_{\frac{\tilde{h}}{2}}|} \right)$$

We can expect the following behavior of the estimate:

- For large  $\tilde{h}$ , this estimate of  $p$  may not be accurate (additional terms in error expansion should not be neglected).
- For very small  $\tilde{h}$ , the computation of  $e_{\tilde{h}}$  and  $e_{\frac{\tilde{h}}{2}}$  involve the subtraction of nearly equal numbers ( $f'(x_0) - D_h f$ ) which could lead to a loss of accuracy due to a catastrophic cancellation, in which case, the estimate may not be accurate for very small  $\tilde{h}$ .

To estimate the error, one typically creates a list of errors  $e_{\tilde{h}}$ ,  $e_{\tilde{h}/2}$ ,  $e_{\tilde{h}/4}$ ,  $e_{\tilde{h}/8}$ , ... etc. and determines an estimate for all pairs

$$p \simeq \log_2 \left( \frac{e_{\tilde{h}}}{e_{\tilde{h}/2}} \right), \quad p \simeq \log_2 \left( \frac{e_{\tilde{h}/2}}{e_{\tilde{h}/4}} \right), \quad p \simeq \log_2 \left( \frac{e_{\tilde{h}/4}}{e_{\tilde{h}/8}} \right), \dots$$

This creates a list of estimations which we can observe and interpret appropriately.

From the table of estimated values for centered differences by C. Andersen put on CCLE (see `errorestimate differentiation.pdf`), we observe different types of behavior and accuracy of the error and the order approximation in different ranges of  $h$ . This is because in addition to problems that finite precision causes for estimating order of convergence, finite precision can also significantly effect the accuracy of numerical approximation of derivatives.

Why would one expect that the inaccuracy of the computation of  $e_h$  is not the complete explanation? The expectation arises from an intuitive understanding of when catastrophic

cancellation will be a problem. Catastrophic cancellation occurs when  $f'(x_0)$  and  $D_h f$  agree to most of the digits in the mantissa. For example, if we had

$$f'(x_0) = .1234512345 \times 10^1, \quad D_h f = .1234512311 \times 10^1$$

subtracting them leads to a number

$$.34xxxxxxxx \times 10^{-8}$$

which is small but imprecise since  $xxxxxxxx$  cannot be determined. So in our case, since we have 14 digits in the mantissa, we expect problems to occur when  $e_h$  is  $10^{-12}$ ,  $10^{-13}$ ,  $10^{-14}$ . In the table, we can observe that after  $f'(x_0)$  and  $D_h f$  agree to nearly all the printed digits the estimate of  $p$  deteriorates. But, if we look at smaller  $h$ , we see that  $f'(x_0)$  and  $D_h f$  no longer share the same digits, the inaccuracy of the evaluation of  $e_h$  isn't the source of the problem. So which is the problem? The inaccuracy of the evaluating

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h}$$

when  $h$  is small. We'll do a preliminary analysis, we limit ourselves to the errors introduced by using inexact machine representations of  $f(x_0 + h)$  and  $f(x_0 - h)$ . We assume there are  $\epsilon_1$  and  $\epsilon_2$  such that  $\epsilon_i \simeq 10^{-m+1}$  for  $i = 1, 2$  where  $m$  is the number of decimal digits in the mantissa. The relative errors occurring when representing  $f(x_0 + h)$  and  $f(x_0 - h)$  are

$$\begin{aligned} \frac{f_A(x_0 + h) - f_E(x_0 + h)}{|f_E(x_0 + h)|} &= \epsilon_1 \quad \rightsquigarrow \quad f_A(x_0 + h) = f_E(x_0 + h) + \epsilon'_1 \\ \frac{f_A(x_0 - h) - f_E(x_0 - h)}{|f_E(x_0 - h)|} &= \epsilon_2 \quad \rightsquigarrow \quad f_A(x_0 - h) = f_E(x_0 - h) + \epsilon'_2 \end{aligned}$$

Then the difference between the exact value  $f'(x_0)$  and the computed value is

$$\begin{aligned} f'(x_0) - \frac{f_A(x_0 + h) - f_A(x_0 - h)}{2h} &= f'(x_0) - \frac{[f_E(x_0 + h) + \epsilon'_1] - [f_E(x_0 - h) + \epsilon'_2]}{2h} \\ &= f'(x_0) - \frac{f_E(x_0 + h) - f_E(x_0 - h)}{2h} + \frac{-\epsilon'_1 + \epsilon'_2}{2h} \end{aligned}$$

If  $f(x_0)$  is  $O(1)$  so that  $\epsilon_1 \simeq \epsilon'_1$  and  $\epsilon_2 \simeq \epsilon'_2$ , and both are  $O(10^{-m+1})$ , then for  $h$  sufficiently small we have

$$\text{Error} = \left| f'(x_0) - \frac{f_A(x_0 + h) - f_A(x_0 - h)}{2h} \right| \leq Mh^2 + \frac{\epsilon}{h}$$

**Draw a picture**

If this preliminary analysis is correct, we should be able to predict (approximately) the value  $h^*$  where the error is minimized. Let

$$g(h) = Mh^2 + \frac{\epsilon}{h}$$



We have

$$g'(h) = 2Mh - \frac{\epsilon}{h^2}$$

Thus

$$g'(h) = 0 \quad \rightsquigarrow \quad 2Mh = \frac{\epsilon}{h^2} \quad \rightsquigarrow \quad h^3 = \frac{\epsilon}{2M}$$

Thus  $h^*$  is  $O(\epsilon^{1/3})$ . So if  $\epsilon$  is  $10^{-14}$  or  $10^{-15}$ , then  $h^* \simeq 10^{-5}$  which is about where the most accurate solutions occurs. This is actually not a small value of  $h$ !

From this analysis, we may conclude that given a  $p$ th order method, then one expects the optimal  $h$ ,  $h^*$ , to be  $\epsilon^{1/(p+1)}$ . For example, for a first order method,  $h^*$  is  $O(\sqrt{\epsilon})$  when  $\epsilon$  is the machine precision. This may not be a small value. In single precision representation (32 bits), if  $\epsilon$  is  $10^{-8}$ , then  $\sqrt{\epsilon}$  is  $10^{-4}$ .

## 8. NUMERICAL INTEGRATION, LECTURES 20 AND 21

Plan:

- Newton-Cotes integration formula
- Composite integration formulas
- Error Analysis
- Aiken estimation  $\rightsquigarrow$  Richardson extrapolation and Romberg integration formula

The basic problem is to numerically approximate

$$\int_a^b f(x) dx$$

**8.1. Newton-Cotes formula.** The idea behind the Newton-Cotes integration formulas is to use a polynomial interpolant:

$$\int_a^b f(x) dx \simeq \int_{x_0=a}^{x_n=b} \{\text{equispaced polynomial interpolant through } (x_0, f(x_0)), \dots, (x_n, f(x_n))\} dx$$

with  $x_i = a + ih$ ,  $i = 0, \dots, n$  and  $h = (b - a)/n$ . **Make an illustration**

Let's work through some lower degree interpolants.

**Trapezoidal** Linear interpolant through two points  $(x_0, f_0 = f_0(x_0))$  and  $(x_1, f_1 = f_1(x_1))$   
**method** (assume  $x_0 < x_1$ ) with  $h = \frac{x_1 - x_0}{1} = x_1 - x_0$ . Using divided differences, the linear interpolant is given by

$$p(x) = f_0 + \left[ \frac{f_1 - f_0}{h} \right] (x - x_0)$$

Integrating  $p$ , we get

$$\begin{aligned}
 \int_{x_0}^{x_1} p(x)dx &= \int_{x_0}^{x_1} f_0(x)dx + \int_{x_0}^{x_1} \frac{f_1 - f_0}{h}(x - x_0)dx \\
 &= f_0 h + \left[ \frac{f_1 - f_0}{h} \right] \frac{(x - x_0)^2}{2} \Big|_{x_0}^{x_0+h} \\
 &= f_0 h + \left[ \frac{f_1 - f_0}{h} \right] \frac{h^2}{2} \\
 &= f_0 h + \frac{h}{2}(f_1 - f_0) \\
 &= \frac{h}{2}(f_0 + f_1)
 \end{aligned}$$

So that we have an approximation

$$\int_{x_0=a}^{x_1=b} f(x)dx \simeq \frac{h}{2}(f_0 + f_1)$$

**Make an illustration**

**Simpson’s** Quadratic interpolant through three equispaced points  $(x_0, f_0 = f_0(x_0))$ ,  $(x_1, f_1 = f_1(x_1))$ ,

**Rule** and  $(x_2, f_2 = f_2(x_2))$  (assume  $x_0 < x_1 < x_2$ ) with  $h = \frac{x_2 - x_0}{2}$ . Integrating the quadratic polynomial interpolating these values, one obtains the approximation

$$\int_{x_0=a}^{x_2=b} f(x)dx \simeq \int_a^b \{\text{quadratic polynomial interpolant}\}dx = \frac{h}{3}(f_0 + 4f_1 + f_2)$$

You’ll be deriving Simpson’s Rule in the homework.

**Simpson’s** Cubic interpolant through four equispaced points. We obtain the approximation  
**3/8th Rule**

$$\int_{x_0=a}^{x_3=b} f(x)dx \simeq \int_a^b \{\text{cubic polynomial interpolant}\}dx = \frac{3}{8}h(f_0 + 3f_1 + 3f_2 + f_3)$$

What is the error in these approximations? Just as with polynomial interpolation and numerical differentiation, one can derive error bounds for these methods. [2, Theorem 4.2] (see the handout posted on CCLE) gives the general result, and the pages following it give the specific results (all of which assume  $f$  has a sufficient number of continuous derivatives):

**Trapezoidal:**

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}(f_0 + f_1) - \frac{h^3}{12}f''(\xi) \quad (8.1)$$

for some  $\xi \in [x_0, x_1]$ .

**Simpson’s:**

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) - \frac{h^5}{90}f^{(4)}(\xi)$$

for some  $\xi \in [x_0, x_2]$ . This formula is derived on [2, p. 194-195] (see handout posted on CCLE) which assumes existence of higher continuous derivatives for the sake of the higher-order error term.

**3/8th’s Rule:**

$$\int_{x_0}^{x_3} f(x)dx = \frac{3}{8}h(f_0 + 3f_1 + 3f_2 + f_3) - \frac{3h^5}{80}f^{(4)}(\xi)$$

for some  $\xi \in [x_0, x_3]$ .

Derivation? A simple idea, but somewhat complicated execution. Idea: Start with an error expression for polynomial interpolation and integrate. If  $p(x)$  is the interpolant through  $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ , then

$$f(x) = p(x) + \text{polynomial interpolation error}(x)$$

so that

$$\int_{x_0}^{x_n} f(x)dx = \int_{x_0}^{x_n} p(x)dx + \int_{x_0}^{x_n} \text{polynomial interpolation error}(x)dx$$

For the integral of the interpolating polynomial one applies the Newton-Cotes formulas, and the integral of the error term needs some “real analysis” work to determine  $k$  and  $C$  so there exists  $\xi \in [x_0, x_n]$  such that

$$\int_{x_0}^{x_n} \text{polynomial interpolation error}(x)dx = C_k f^{(k)}(\xi)h^k$$

For the Trapezoidal method, you may find a proof on CCLE from [1].

**Remark 8.1.** You may want to have a look at these Wolfram demonstrations on [Newton-Cotes formula](#).

**Remark 8.2.** Here is an excellent Wolfram article on [Newton-Cotes formulas](#) which covers much of the material of this part of the lecture.

**Remark 8.3.** Wondering where those coefficients of the integral of the interpolating polynomial come from? Here is a list of the sequence of coefficients emerging from Newton-Cotes formulas recorded on the Online Encyclopedia of Integer Sequences [A093735](#) and [A093736](#).

Assuming  $f$  has a sufficient number of continuous derivatives, one can deduce an error bound for the integral approximations which we specify for the Trapezoidal method and Simpson’s Rule:

**Trapezoidal:** For all  $h < \delta$  (here  $\delta$  comes from Taylor’s formula),

$$\left| \int_{x_0}^{x_1} f(x)dx - \frac{h}{2}(f_0 + f_1) \right| \leq Ch^3, \quad C = \frac{\max_{x \in [x_0, x_0+\delta]} f''(x)}{12}$$

As  $h \rightarrow 0$ , the error in the Trapezoidal approximation to the integral converges to 0 with order  $3 \rightsquigarrow$  “3rd order accurate approximation”.

**Simpson’s:** For all  $h < \delta$ ,

$$\left| \int_{x_0}^{x_2} f(x)dx - \frac{h}{3}(f_0 + 4f_1 + f_2) \right| \leq Ch^5, \quad C = \frac{\max_{x \in [x_0, x_0+2\delta]} f^{(4)}(x)}{90}$$

As  $h \rightarrow 0$ , the error in the Simpson’s approximation to the integral converges with order  $5 \rightsquigarrow$  “5th order accurate approximation”.

The bounds show that as  $h \rightarrow 0$ , the approximations to the integrals become more accurate.

**Make an illustration**

A problem remains, and it is that we want an approximation to an integral over a fixed interval  $[a, b]$ —so how can one create approximations that become more accurate as  $h \rightarrow 0$ ? With the methods we know, there are two possibilities (suppose that we divided the interval into  $M$  pieces):

(I) Choosing higher degree polynomials:

$$\int_a^b f(x)dx \simeq \int_a^b (\text{degree } M \text{ polynomial interpolant})dx$$

(II) Choosing lower degree approximations on subintervals:

$$\int_a^b f(x)dx \simeq \sum_{j=0}^M \text{approximation to integral } \int_{x_i}^{x_{i+1}} f(x)dx$$

Using higher-order equispaced interpolation is generally a bad idea (e.g. Runge’s phenomenon), so one chooses possibility (II). Integration methods obtained by adding up a collection of integral approximations are called “composite integration formulas”.

**8.2. Composite integration formulas.** We start with the *composite Trapezoidal method*. When all subintervals are of equal size:  $n$  subintervals (or panels), then we have  $x_i = a + hi$ ,  $i = 0, 1, \dots, n$  with  $h = (b - a)/n$ , and

$$\begin{aligned} \int_a^b f(x)dx &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx \\ &\simeq \sum_{i=0}^{n-1} \frac{f_i + f_{i+1}}{2} h \\ &= \frac{h}{2} f_0 + h \sum_{i=1}^{n-1} f_i + \frac{h}{2} f_n \end{aligned}$$

For the *composite Simpson’s Rule* assume that  $n$  is even (and choose  $x_i$  as above). We have

$$\begin{aligned} \int_a^b f(x)dx &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx \\ &\simeq \sum_{i=0}^{n-2} \frac{h}{3} (f_i + 4f_{i+1} + f_{i+2}) \\ &= \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + \dots + 2f_{n-2} + 4f_{n-1} + f_n) \end{aligned}$$

What is the error in these approximations? [2, Theorems 4.4 and 4.5] answer this. We specify the result for the above two methods and prove it for the first:

**Composite  
Trapezoidal:**

$$\int_a^b f(x)dx = \frac{h}{2} f_0 + h \sum_{i=1}^{n-1} f_i + \frac{h}{2} f_n - \frac{b-a}{12} h^2 f''(\mu)$$

for some  $\mu \in [a, b]$ .

**Composite  
Simpson’s:**

$$\int_a^b f(x)dx = \frac{h}{3} \left[ f_0 + \sum_{j=1}^{n/2} 4f_{2j-1} + \sum_{j=1}^{n/2-1} 2f_{2j} + f_n \right] - \frac{b-a}{180} h^4 f^{(4)}(\mu)$$

for some  $\mu \in [a, b]$ .

We note that the composite Trapezoidal method has a 2nd order accurate approximation, and the composite Simpson's Rule has a 4th order accurate approximation.

**Proposition 8.4.** *Assume  $f$  is 2 times continuously differentiable over  $[a, b]$ , then there exists  $\mu \in [a, b]$  such that*

$$\int_a^b f(x)dx - \frac{h}{2}f_0 - h \sum_{i=1}^{n-1} f_i - \frac{h}{2}f_n = -\frac{b-a}{12}h^2 f''(\mu)$$

where  $h = (b-a)/n$ ,  $f_i = f(x_i)$ , and  $x_i = a + ih$ ,  $i = 0, \dots, n$ .

*Proof.* By an additivity property of the Riemann integral, we have

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

By (9.1), there exists  $\mu_i \in [x_i, x_{i+1}]$  such that

$$\int_{x_i}^{x_{i+1}} f(x)dx = \frac{h}{2}(f_i + f_{i+1}) - \frac{h^3}{12}f''(\mu_i)$$

so

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx = \frac{h}{2} \sum_{i=0}^{n-1} (f_i + f_{i+1}) - \sum_{i=0}^{n-1} \frac{h^3}{12} f''(\mu_i)$$

Hence

$$\sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx - \frac{h}{2}f_0 - h \sum_{i=1}^{n-1} f_i - \frac{h}{2}f_n = - \left( \sum_{i=0}^{n-1} f''(\mu_i) \right) \frac{h^3}{12}$$

Now

$$\min_{x \in [a, b]} f''(x) \leq \frac{1}{n} \sum_{i=0}^{n-1} f''(\mu_i) \leq \max_{x \in [a, b]} f''(x)$$

Since  $f''(x)$  is continuous and takes all values between its maximum and minimum there must be a  $\mu \in [a, b]$  such that  $f''(\mu) = \frac{1}{n} \sum_{i=0}^{n-1} f''(\mu_i)$ . Thus

$$\sum_{i=0}^{n-1} f''(\mu_i) = n \left[ \frac{1}{n} \sum_{i=0}^{n-1} f''(\mu_i) \right] = n f''(\mu) = \frac{b-a}{h} f''(\mu)$$

This proves the claim. □

## 9. NUMERICAL INTEGRATION

We introduced the Newton-Cotes formulas with error terms which specified for linear and quadratic interpolation that are

**Trapezoidal:**

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}(f_0 + f_1) - \frac{h^3}{12}f''(\xi) \quad (9.1)$$

for some  $\xi \in [x_0, x_1]$ .

**Simpson's:**

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) - \frac{h^5}{90}f^{(4)}(\xi)$$

for some  $\xi \in [x_0, x_2]$ .

You may want to check [2, Theorem 4.2]. The idea behind these formulas is to start with an error expression for polynomial interpolant of an integrand. If  $p(x)$  is the interpolant through  $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ , then

$$f(x) = p(x) + \text{polynomial interpolation error}(x)$$

see Theorem 6.8. Integrating gives us

$$\int_{x_0}^{x_n} f(x)dx = \int_{x_0}^{x_n} p(x)dx + \int_{x_0}^{x_n} \text{polynomial interpolation error}(x)dx$$

where the value of the integral  $\int_{x_0}^{x_n} p(x)dx$  is given by the Newton-Cotes formula, and the value of the integral of the error term can be represented as

$$\int_{x_0}^{x_n} \text{polynomial interpolation error}(x)dx = C_k f^{(k)}(\xi) h^{k+1}$$

where  $k$  and  $C$  depend on the number of distinct points defining the interpolant. The error in the above simple Trapezoidal and Simpson's methods becomes small if  $h$  is small which means that the value of the approximation is only accurate over small integration intervals. One way to get an accurate approximation over a fixed integration range (interval) is to use the composite integration formulas. The composite Newton-Cotes rules and their error for linear and quadratic interpolation are given by

**Composite  
Trapezoidal:**

$$\int_a^b f(x)dx = \frac{h}{2}f_0 + h \sum_{i=1}^{n-1} f_i + \frac{h}{2}f_n - \frac{b-a}{12}h^2 f''(\mu)$$

for some  $\mu \in [a, b]$ .

**Composite  
Simpson's:**

$$\int_a^b f(x)dx = \frac{h}{3} \left[ f_0 + \sum_{j=1}^{n/2} 4f_{2j-1} + \sum_{j=1}^{n/2-1} 2f_{2j} + f_n \right] - \frac{b-a}{180}h^4 f^{(4)}(\mu)$$

for some  $\mu \in [a, b]$ .

For the Trapezoidal method, there is another important (and useful) error relation - the Euler-Maclaurin formula. A full and precise statement taken from [1] is posted on CCLE. For our purposes we'll just need the following result.

**Proposition 9.1** (Composite trapezoidal method asymptotic error expansion). *Let  $h = (b - a)/n$ ,  $x_i = a + ih$ ,  $i = 0, 1, \dots, n$ . Assume  $f$  is  $2m + 2$  times continuously differentiable over  $[a, b]$  for some  $m \geq 0$ . Then there exists constants  $B_k$ ,  $k = 2, 4, 6, \dots, 2m$  depending on  $f^{(k-1)}(x)$  and  $B_{2m+2}$  depending on  $f^{(2m+2)}(x)$  such that*

$$\int_a^b f(x)dx - \left[ \frac{h}{2}f_0 + h \sum_{i=1}^{n-1} f_i + \frac{h}{2}f_n \right] = B_2 h^2 + B_4 h^4 + \dots + B_{2m+2} h^{2m+2}$$

*Proof.* An application of the Euler-Maclaurin formula,

see handout `NumericalIntegrationError_Atkinson.pdf`. □

**Remark 9.2.** The asymptotic error expansion for the Trapezoidal method is analogous to the asymptotic error expansions for numerical differentiation - a power series in  $h$  with coefficients that depend on derivatives of  $f$ .

We will make two uses of asymptotic error expansions.

**9.1. Aitken estimation of the order of a method.** Assume  $I_h$  = an approximation to  $\int_a^b f(x)dx$  depending on a parameter  $h$ . Assume  $I_h$  has an asymptotic error expansion so that

$$e_h = \int_a^b f(x)dx - I_h = c_p h^p + c_{p+1} h^{p+1} + \dots$$

where  $\{c_p\}$ s are constants *not* depending on  $h$ . Then to obtain an estimate of the error *without knowing* the exact value of  $\int_a^b f(x)dx$ , one can use

$$p \simeq \log_2 \left[ \frac{I_{h/2} - I_h}{I_{h/4} - I_{h/2}} \right]$$

That is, the results of 3 separate computations  $h, h/2, h/4$ . Why is this an estimate? Let  $I_E$  = exact value of  $\int_a^b f(x)dx$  so that

$$I_E - I_h = c_p h^p + c_{p+1} h^{p+1} + \dots$$

Then

$$\begin{aligned} \frac{I_{h/2} - I_h}{I_{h/4} - I_{h/2}} &= \frac{-(I_E - I_{h/2}) + (I_E - I_h)}{-(I_E - I_{h/4}) + (I_E - I_{h/2})} \\ &= \frac{-c_p(h/2)^p - c_{p+1}(h/2)^{p+1} - \dots + c_p h^p + c_{p+1} h^{p+1} + \dots}{-c_p(h/4)^p - c_{p+1}(h/4)^{p+1} - \dots + c_p(h/2)^p + c_{p+1}(h/2)^{p+1} + \dots} \end{aligned}$$

Assuming  $h$  is sufficiently small, we get

$$\begin{aligned} &\simeq \frac{-c_p(h/2)^p + c_p h^p}{-c_p(h/4)^p + c_p(h/2)^p} \\ &= \frac{1 - (1/2)^p}{(1/2)^p - (1/4)^p} = \frac{1 - (1/2)^p}{(1/2)^p(1 - (1/2)^p)} = 2^p \end{aligned}$$

Upon taking logarithm,

$$p \simeq \log_2 \left[ \frac{I_{h/2} - I_h}{I_{h/4} - I_{h/2}} \right]$$

**Remark 9.3.** In your homework, you are applying this procedure when  $I_h$  is the composite Trapezoidal method - but one can consider applying it to any numerical process that can assumed to have an asymptotic error expansion. For example, numerical differentiation.

What size to choose  $h$ ? One creates several estimates considering  $h, h/2, h/4, h/8, \dots$  and interpret. Similar to estimates of  $p$  when knowing an exact value, but uses 3 consecutive results instead.

**9.2. Richardson extrapolation.** Another use of asymptotic error expansion is “Richardson extrapolation”. We again assume that a numerical procedure produced an asymptotic error expansion

$$I_E - I_h = c_p h^p + c_{p+1} h^{p+1} + \dots$$

where  $I_E = \int_a^b f(x)dx$  represents the exact value,  $I_h$  is the approximation at panel size  $h$  and  $I_{h/2}$  is the approximation at panel size  $h/2$ . Then one can consider combining  $I_h$  and  $I_{h/2}$  to obtain a more accurate value  $I_E$ :

$$\begin{aligned} \text{(I)} \quad I_E - I_h &= c_p h^p + c_{p+1} h^{p+1} + \dots \\ \text{(II)} \quad I_E - I_{h/2} &= c_p (h/2)^p + c_{p+1} (h/2)^{p+1} + \dots \end{aligned}$$

The idea is to combine these equations so the error term  $c_p h^p$  is canceled. This can be reached by multiplying (I) with  $-(1/2)^p$  and add it to (II), then re-arrange to get

$$I_E - \frac{I_{h/2} - (1/2)^p I_h}{1 - (1/2)^p} = \tilde{c}_{p+1} h^{p+1} + \tilde{c}_{p+2} h^{p+2} + \dots$$

where  $\tilde{c}$  are other constants still independent of  $h$ . The combination of  $I_h$  and  $I_{h/2}$  is an  $h^{p+1}$  approximation to  $I_E$ .

Next, we will apply Richardson extrapolation to the composite Trapezoidal method for which we have the asymptotic error expansion

$$I_E - I_h = B_2 h^2 + B_4 h^4 + \dots$$

The first step Richardson extrapolation using  $p = 2$  gives

$$I_E - (4/3 I_{h/2} - 1/3 I_h) = \tilde{B}_4 h^4 + \tilde{B}_6 h^6 + \dots$$

Denote the extrapolation  $\tilde{I}_h = 4/3 I_{h/2} - 1/3 I_h$ . Then the error  $\tilde{e}_h = I_E - \tilde{I}_h$  has again an asymptotic expansion  $\tilde{B}_4 h^4 + \tilde{B}_6 h^6 + \dots$  to which we can again apply a Richardson extrapolation. As such if one constructs a column of values  $I_h, I_{h/2}, I_{h/4}, I_{h/8}, I_{h/16}, \dots$  one can form a column for the extrapolated values  $\tilde{I}_h, \tilde{I}_{h/2}, \tilde{I}_{h/4}, \tilde{I}_{h/8}, \dots$ . The first column has order 2 accuracy, while the second column represents order 4 accuracy. So we can combine  $\tilde{I}_h$  and  $\tilde{I}_{h/2}$  to get a 6th order approximation (here  $p = 4$ ):

$$I_E - (16/15 \tilde{I}_{h/2} - 1/15 \tilde{I}_h) = \tilde{\tilde{B}}_6 h^6 + \tilde{\tilde{B}}_8 h^8 + \dots$$

This allows us to build again columns of values of approximations of increasing accuracy.

**Illustrate**

Using repeated Richardson extrapolation as far as one can go is called Romberg integration (e.g. 10th order). Implementations of Romberg integration include logic for deciding when to stop..

**Remark 9.4.** Here is a wolfram demonstration of the [Romberg integration method](#).

**9.3. Gauss quadrature: A different approach to constructing numerical integration approximations.** Motivation: The general technique for constructing Newton-Cotes methods. Given nodes  $\{x_i\}_{i=1}^n$  (e.g.  $x_i = a + (i-1)h$ ,  $h = (b-a)/(n-1)$ ) find weights  $\{w_i\}_{i=1}^n$  so that

$$\int_a^b f(x) dx \simeq \sum_{i=1}^n f(x_i) w_i$$

How? Determine weights by simplifying

$$\int_a^b \text{polynomial interpolant through } (x_i, f(x_i)) dx$$

appropriately. Determine  $w_i$  such that  $\int_a^b p(x) dx = \sum w_i f(x_i)$ . In these formulas, the  $\{x_i\}$ s are just “given”.

**Remark 9.5.** Note that in Gauss quadrature  $i = 1, \dots, n$  indexing starts at 1.

There are two ideas underlying Gauss quadrature:



- 1) Simultaneously determine nodes  $\{x_i\}_{i=1}^n$  and weights  $\{w_i\}_{i=1}^n$  so that  $\sum_{i=1}^n f(x_i)w_i$  is as accurate an approximation as possible to  $\int_a^b f(x)dx$ .
- 2) Determine the nodes  $\{x_i\}_{i=1}^n$  and the weights  $\{w_i\}_{i=1}^n$  by requiring that the resulting formula is exact for all polynomials up to as high degree as possible. Concretely, it is required that with those nodes and weights all polynomials of degree  $\leq 2n - 1$  are integrated exactly.

We have the following facts (presented without proof, see [2, Chapter 4.7]):

- The set of  $n$  nodes and  $n$  weights that integrate all polynomials of degree  $\leq 2n - 1$  is uniquely determined (as one might expect for  $2n$  linearly independent equations with  $2n$  unknowns).
- The nodes  $\{x_i\}$  of an  $n$ -point formula for  $[-1, 1]$  are the roots of the  $n$ th Legendre polynomial. The weights are given by

$$w_i = \int_{-1}^1 \prod_{j=1, j \neq i}^n \frac{x - x_j}{x_i - x_j} dx$$

For the simple case  $n = 2$  points, it's not too hard to set up and solve the equations that determine  $x_1, x_2$  and  $w_1, w_2$ . Specifically, get 4 equations by requiring that the formula be exact for the first 4 monomials  $1, x, x^2, x^3$ , the equations are

$$(x_1)^j w_1 + (x_2)^j w_2 = \int_{-1}^1 x^j dx \quad \text{for } j = 0, 1, 2, 3$$

However, this is not the way the nodes and weights are generally determined for  $n > 2$ .

**Remark 9.6.** To obtain the nodes and weights for an integral over  $[a, b]$ , one first obtains the nodes and weights  $\{x_i\}_{i=1}^n$  and  $\{w_i\}_{i=1}^n$  for integrals over  $[-1, 1]$  and then, as you will figure out in Assignment 9, one scales and shifts to get nodes  $\{x_i\}_{i=1}^n$  and weights  $\{w_i\}_{i=1}^n$ .

**Remark 9.7.** Here is a Wolfram demonstration of [Gaussian quadrature](#).

Why does making the formula exact for all polynomials lead to accurate formulas for general functions?

Here is a sketch of an error bound. Assume we have  $\{x_i\}_{i=1}^n$  and  $\{w_i\}_{i=1}^n$  so that for all polynomials  $p(x)$  up to degree  $2n - 1$

$$\int_a^b p(x)dx - \sum_{j=1}^n p(x_j)w_j = 0$$

Given  $f(x)$ , let  $p(x)$  be a polynomial of degree at most  $2n - 1$  approximating  $f(x)$  for  $x \in [a, b]$ . (One can use an interpolating polynomial, or a Taylor series polynomial,...) Then

$$\begin{aligned} \int_a^b f(x)dx - \sum_{j=1}^n f(x_j)w_j &= \int_a^b f(x)dx - \int_a^b p(x)dx + \sum_{j=1}^n p(x_j)w_j - \sum_{j=1}^n f(x_j)w_j \\ &= \int_a^b [f(x) - p(x)]dx - \sum_{j=1}^n [f(x_j) - p(x_j)]w_j \end{aligned}$$

So the error will be small if  $[f(x) - p(x)]$  is small. Since  $p(x)$  is any polynomial of degree  $\leq 2n - 1$ , we could imagine using  $\tilde{p}(x)$  is the “best” approximation (see [Polynomial of best approximation](#)) to  $f(x)$  over  $[a, b]$ . If  $\tilde{M}_n = \max_{x \in [a, b]} |f(x) - \tilde{p}(x)|$ , then

$$|\text{error}| \leq (b - a)\tilde{M}_n + \tilde{M}_n \sum_{j=1}^n w_j = 2\tilde{M}_n(b - a)$$

We have  $\sum_{j=1}^n w_j = b - a$  since the formula integrates  $f(x) \equiv 1$  exactly. As  $n \rightarrow \infty$ , then  $\tilde{p}(x)$  becomes a better approximation to  $f$  as  $\tilde{M}_n \rightarrow 0$ .

However the error bound

$$\left| \int_a^b f(x) dx - \sum_{j=1}^n f(x_j) w_j \right| \leq 2(b-a) \max_{x \in [a,b]} |f(x) - \tilde{p}(x)|$$

where  $\tilde{p}(x)$  is the best approximation to  $f(x)$  of degree  $\leq 2n-1$ , isn't informative unless we have an expression for  $\max_{x \in [a,b]} |f(x) - \tilde{p}(x)|$ . Here is an example form [1], for  $[a, b] = [-1, 1]$ :

$$\tilde{M}_n = \min_{\deg(\tilde{p}) \leq 2n-1} \max_{x \in [-1,1]} |f(x) - \tilde{p}(x)| \leq \frac{\pi}{4^n (2n)!} \max_{x \in [-1,1]} |f^{(2n)}(x)|$$

The factor  $\frac{\pi}{4^n (2n)!}$  goes to zero very quickly.

Why is Gaussian quadrature used in engineering? The idea easily generalizes to formulas for multi-dimensional integrals - a task that is a key ingredient of the finite element method.

$$\int_{\Delta} f(x, y) dx dy \simeq \sum_{i=1}^n f(x_i, y_i) w_i$$

where nodes  $(x_i, y_i)$  and weights  $w_i$  are chosen such that integration of polynomials in  $(x, y)$  are exact.

**Example 9.8.** Let  $T$  be a triangle in the plane with edges  $\vec{p}_1, \vec{p}_2, \vec{p}_3$ . Then

$$\int_T \int f(x, y) dx dy = \frac{1}{3} \text{Area}(T) \left( f\left(\frac{\vec{p}_1 + \vec{p}_2}{2}\right) + f\left(\frac{\vec{p}_1 + \vec{p}_3}{2}\right) + f\left(\frac{\vec{p}_2 + \vec{p}_3}{2}\right) \right)$$

is exact for all quadratic polynomials.

## 10. ASPECTS OF COMPUTATIONAL LINEAR ALGEBRA

Problem: Given  $n \times n$  matrix, and  $\vec{b}$  ( $n \times 1$  matrix = vector) find  $\vec{x}$  ( $n \times 1$  matrix = vector) such that  $A\vec{x} = \vec{b}$ .

Plan:

- The procedure you learned to solve linear systems of equations (Gaussian elimination) can be implemented on a computer. When implementing the procedure, it is beneficial to switch rows when the diagonal element on the row begin used to remove elements from other rows (the pivot element) is small - not just when it is zero. The resulting procedure is called “Gaussian elimination with partial pivoting”.
- The Gaussian elimination procedure can be encoded in matrix form - this leads to the LU matrix factorization  $A = LU$  (or  $PA = LU$  if pivoting is done).
- It can be useful to compute and store the  $L, U$  ( $P, L, U$ ) factors.

**10.1. Gaussian elimination review.** Form augmented matrix, reduce to upper triangle form, and back substitute:

$$\left[ A \mid \vec{b} \right] \rightsquigarrow \left[ \begin{array}{ccc|c} x & x & x & \vec{b} \\ & x & x & \\ 0 & & x & \end{array} \right] \rightsquigarrow \vec{x} = U^{-1} \vec{b}$$

where by

$$U = \begin{bmatrix} x & x & x \\ & x & x \\ 0 & & x \end{bmatrix}$$

Let’s look at an example.

$$\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

By row reduction,

$$\left[ \begin{array}{ccc|c} 4 & 1 & 1 & 1 \\ 1 & 4 & 1 & 0 \\ 1 & 1 & 3 & 0 \end{array} \right] \rightsquigarrow \left[ \begin{array}{ccc|c} 4 & 1 & 1 & 1 \\ 0 & 15/4 & 3/4 & -1/4 \\ 0 & 3/4 & 11/4 & -1/4 \end{array} \right] \rightsquigarrow \left[ \begin{array}{ccc|c} 4 & 1 & 1 & 1 \\ 0 & 15/4 & 3/4 & -1/4 \\ 0 & 0 & 13/5 & -1/5 \end{array} \right]$$

So we can find  $\vec{x}$  by solving

$$\begin{bmatrix} 4 & 1 & 1 \\ 0 & 15/4 & 3/4 \\ 0 & 0 & 13/5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/4 \\ -1/5 \end{bmatrix}$$

using back-substitution

$$\begin{aligned} (13/5)x_3 &= -1/5 &\Rightarrow x_3 &= -1/13 \\ (15/4)x_2 + (3/4)x_3 &= -1/4 &\Rightarrow (15/4)x_2 + 3(-1/13) &= -1 &\Rightarrow x_2 &= -2/39 \\ 4x_1 + x_2 + x_3 &= 1 &\Rightarrow 4x_1 &= 1 + 2/39 + 1/13 &\Rightarrow x_1 &= 11/39 \end{aligned}$$

Check work using matlab (octave):

```
>> A=[4 1 1; 1 4 1; 1 1 3]; % columns separated by semicolons
>> b=[1 0 0]'; % []' for transpose
>> x=A\b; % \ operator for solving Ax=b
>> x
```

Output

```
x = 0.282051 = 1/39
    -0.051282 = -2/39
    -0.076923 = -1/13
>>
```

Reminder of things that “can go wrong” and what you do about it.

- What happens when you get a zero on the diagonal? You switch rows, then continue. A very simple example is

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

In which case, you swap rows and solve for  $x_1$  and  $x_2$ .

- What happens though if the system is singular ( $\det(A) = 0$ )? You will obtain a zero (or zeros) in the lower right corner. Depending on the right-hand side:
  - (a) There is a solution but it won’t be unique.
  - (b) There may not be a solution.

Let look at two simple examples. For (a), take

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

The upper triangle augmented matrix is

$$\left[ \begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 0 & 0 \end{array} \right]$$

So  $0x_2 = 0$  and we can choose  $x_2$  arbitrarily. On the other hand,  $x_1 + x_2 = 1$  which means that  $x_1 = 1 - x_2$ . The solution set is

$$\vec{x} = \begin{bmatrix} 1 - \alpha \\ \alpha \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \alpha \in \mathbb{R}$$

For (b), take

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

with

$$\left[ \begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 0 & -1 \end{array} \right]$$

The last row represent an inconsistent equation  $0x_2 = -1$ . Thus the system has no solution.

We have the following observations/reflections about Gaussian elimination (done “by hand”).

- (1) To solve  $A\vec{x} = \vec{b}$ , one doesn’t need to construct  $A^{-1}$  and then evaluate  $\vec{x} = A^{-1}\vec{b}$ .
- (2) At the  $k$ th step, if the diagonal element of the  $k$ th row is zero (“the pivot element at step  $k$  is zero”) we switch rows. For example, suppose you get for  $k = 2$

$$\tilde{A}^{(2)} = \left[ \begin{array}{cccc|c} x & x & x & x & x \\ 0 & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{array} \right]$$

The pivot element is  $\tilde{A}_{(2,2)}^{(2)} = 0$  which needs to be switched.

- (a) One has to decide what row to switch with.
- (b) We can switch rows at any time, that is we don’t necessarily have to switch when the element is zero - we might want to do it when it’s just ‘small’.
- (3) If the system is singular, then while  $A^{-1}$  may not exist, it may still be possible to find a solution to  $A\vec{x} = \vec{b}$ . If a solution does exist, it is not unique - so the process obtaining a solution requires extra information.

Facts:

- (1) One can create a program that carries out Gaussian elimination:
  - To avoid numerical problems, row switching is done even when the pivot element is non-zero. If one switches rows only, it’s called “partial pivoting”. One can also switch columns - if one does both row and column switching it’s called “full pivoting”.
  - There are multiple strategies for deciding which rows/or columns to switch, and these are discussed in the text.
- (2) We will see that Gaussian elimination with partial pivoting can be ‘encoded’ in matrix form as the  $LU$  factorization (or decomposition) of  $A$ .
- (3) If the matrix is singular, the computational implementation “breaks down” and may give inaccurate results. This also happens when the matrix is ‘nearly’ singular; good programs give one a warning when the solution is likely to be inaccurate.
- (4) There are multiple solution methods based on factorization, some may require more computational work, but allow one to construct solutions of singular systems, or they have other desirable properties.

Next, practical details (tips) for using Matlab (Octave) to solve linear systems.

- By default, vectors in Matlab are ‘row vectors’ ( $= 1 \times n$  matrices)

```
b(1)=1;
b(2);
```

$$\rightsquigarrow \vec{b} = (2, 3)$$

To get column vectors that are typically wanted when solving linear systems, you either add a second index, 1, when initializing:

```
b(1,1)=2;
b(2,1)=3;
```

$$\rightsquigarrow \vec{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Or, you may initialize the vector with a column of zeros

```
b=zeros(2,1);
b(1)=2;
b(2)=3;
```

$$\rightsquigarrow \vec{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

With Matlab it's easy to make mistakes in the sizes of matrices, since matrices are “size on demand” - so if you get an index wrong, this can lead to an incorrectly sized matrix. A construct that can help is to initialize the matrix with a zero matrix of the appropriate size. If you know the size will be  $m \times n$ , then one uses

```
A=zeros(m,n);
```

Later in the code, if one does `size(A)`, and one does not get the original values for  $m$  and  $n$ , one knows that one has done something one didn't expect to.

- If a matrix has many zeros, it is often useful to use the sparse matrix representation. However, there are special rules for setting the values. Run `help sparse` for more information.
- Solution of the system  $A\vec{x} = \vec{b}$  can be accomplished with the command  $x = A \backslash b$ ;
- If the system is singular, or nearly singular, then an error message will be given

```
warning: matrix singular to machine precision rcond=1.42901 e^{-19}
```

Here `rcond` stands for reciprocal condition number written as  $\frac{1}{\kappa_2}$  ( $\kappa$  is the Greek letter kappa) where

$$\kappa_2 = \left( \frac{\text{largest eigenvalue of } A^*A}{\text{smallest eigenvalue of } A^*A} \right)^{1/2}$$

where  $A^*$  denotes the conjugate transpose of  $A$  ( $= A^T$  the usual transpose if  $A$  is a real matrix). The reason behind this is

**Proposition 10.1.** *If  $A$  is singular, then `rcond` = 0.*

*Proof.* If  $A$  is singular, then there exists  $\vec{v} \in \mathbb{R}^n$ ,  $\vec{v} \neq 0$  such that  $A\vec{v} = 0$ . This implies that the smallest eigenvalue<sup>11</sup> of  $A^*A = 0$ . In this case,  $\kappa_2 = \infty$  and thus `rcond` = 0.  $\square$

- If one gets a warning message about matrix being singular, and it's not expected, check for programming errors in setting up the matrix.

<sup>11</sup>Note that all eigenvalues of  $A^*A \geq 0$ . Indeed, if  $\lambda$  is an eigenvalue of  $A^*A$ , then there exists  $\vec{v} \neq \vec{0}$  such that  $A^*A\vec{v} = \lambda\vec{v}$ . Then

$$\|A\vec{v}\|_2^2 = (A\vec{v})^*(A\vec{v}) = \vec{v}^*A^*A\vec{v} = \lambda\vec{v}^*\vec{v} = \lambda\|\vec{v}\|_2^2$$

This implies that  $\lambda \geq 0$ .

- At the “executive” level, what’s important to know? If  $\mathbf{rcond}$  is very small (near machine epsilon), then the matrix is “ill-conditioned” and solutions to  $A\vec{x} = \vec{b}$  obtained with Gaussian elimination may have large errors. If  $\mathbf{rcond} \simeq 1$ , then the system is well conditioned and accurate solutions can be obtained with Gaussian elimination.

**10.2. Splines.** Task: Given data  $(x_i, f(x_i))$ ,  $i = 0, 1, \dots, n$  determine a spline function  $s(x)$  that approximates  $f(x)$  for  $x \in [x_{\min}, x_{\max}] = [x_0, x_n]$ .

The spline function  $S(x)$  is a piecewise polynomial approximation and the coefficients of the polynomial in the approximation are determined by requiring that

- the approximation and some number of its derivatives be continuous where the piecewise polynomial meet (nodes)
- the approximation interpolates
- additional constraints be satisfied

**Illustrate**

- Linear spline: piecewise linear

$$s_i(x) = a_i + b_i(x - x_{i-1}), \quad x \in [x_{i-1}, x_i], \quad i = 1, \dots, n$$

- Quadratic spline: piecewise quadratic polynomial

$$s_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2, \quad x \in [x_{i-1}, x_i], \quad i = 1, \dots, n$$

- Cubic spline: piecewise cubic polynomial

$$s_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, \quad x \in [x_{i-1}, x_i], \quad i = 1, \dots, n$$

Construction of a spline approximation  $s(x)$ ? For a  $p$ -degree spline there are  $n \times (p + 1)$  free coefficients, therefore one seeks  $n \times (p + 1)$  constraints that lead to an  $[n \times (p + 1)] \times [n \times (p + 1)]$  system of equations. These equations are then solved to obtain values for the coefficients of each of the polynomial in the approximation. The constraints imposed consist of

- requiring continuity of the approximation and some number of derivatives at the ‘interior’ nodes
- requiring the spline approximation interpolates the values of  $f(x)$  at the nodes
- additional constraints as necessary to get a square system of equations.

Here is an example (and background information for Assignment 10).

**Example 10.2.** A quadratic spline approximation using equispaced nodes.  $x \in [x_{\min}, x_{\max}]$ ,  $x_i = x_{\min} + hi$ ,  $h = \frac{x_{\max} - x_{\min}}{n}$  :

$$s_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2, \quad i = 1, 2, \dots, n$$

Each  $s_i(x)$  has 3 coefficients to be determined which leads to specify  $3n$  constraints:

- (i) Continuity of  $s(x)$  at interior nodes

$$s_i(x_i) = s_{i+1}(x_i), \quad i = 1, 2, \dots, n - 1$$

This amounts to  $n - 1$  constraints.

- (ii) Continuity of  $s'(x)$  at interior nodes

$$s'_i(x_i) = s'_{i+1}(x_i), \quad i = 1, 2, \dots, n - 1$$

This amounts to  $n - 1$  constraints.

(iii)  $s(x)$  interpolates  $f(x)$  at  $x_i$

$$s(x_i) = f(x_i), \quad i = 0, 1, \dots, n$$

which amounts to  $n + 1$  constraints.

So the total number of constraints so far  $n - 1 + n - 1 + n + 1 = 3n - 1$ . We need 1 additional constraint. There are many possibilities, we'll use one that is often mentioned in texts

(iv)  $s'(x_0) = \beta$ ,  $\beta$  = a prescribed value

Now we have  $3n$  constraints which leads to  $3n$  equations. Here is a derivation of the equations (version 1).

(i) continuity of  $s$  at interior points yields

$$a_i + b_i(x_i - x_{i-1}) + c_i(x_i - x_{i-1})^2 = a_{i+1} + b_{i+1}(x_i - x_i) + c_{i+1}(x_i - x_i)^2$$

So

$$a_i + b_i h + c_i h^2 - a_{i+1} = 0, \quad i = 1, 2, \dots, n - 1$$

(ii) continuity of  $s'$  at interior points yields

$$b_i + 2c_i(x_i - x_{i-1}) = b_{i+1} + 2c_{i+1}(x_i - x_i)$$

So

$$b_i h + 2c_i h - b_{i+1} = 0, \quad i = 1, 2, \dots, n - 1$$

(iii) interpolating

$$a_i + b_i(x_{i-1} - x_{i-1}) + c_i(x_{i-1} - x_{i-1})^2 = f(x_{i-1})$$

So

$$a_i = f(x_{i-1}), \quad i = 1, 2, \dots, n + 1$$

(iv) additional constraint

$$b_1 + 2c_1(x_0 - x_0) = \beta$$

So

$$b_1 = \beta$$

So you can set up and solve a  $3n \times 3n$  system of equations

$$A \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ \vdots \\ a_n \\ b_n \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{3n} \end{bmatrix}$$

where  $\vec{y}$  depends on  $f(x_i)$ 's and  $\beta = s'_1(x_0)$ . The coefficients determined in this way determine the quadratic spline approximation  $s(x)$ .

While not ‘hard’ it is somewhat tedious to set up these equations (complicated indexing) so there is another way. The other way leads to equations that are easier to implement. You will be using these equations in Assignment 10.

Derivation of equations (version 2). Overview:

- Introduce values  $f_i = f(x_{i-1})$ ,  $i = 1, 2, \dots, n + 1$  and values  $q_i = s'(x_{i-1})$ ,  $i = 1, 2, \dots, n + 1$
- Express the coefficients  $a_i, b_i, c_i$ ,  $i = 1, \dots, n$  in terms of the  $\{f_i\}$ s and  $\{q_i\}$ s
- Determine equation for the  $q_i$  values  $i = 1, \dots, n + 1$

The construction of a spline then consists of setting up and solving the equations for  $q_i$ , and the determining the values of  $a_i, b_i, c_i, i = 1, \dots, n$  from the values of  $f_i$  and  $q_i$ . We'll be using the notation  $f_i = f(x_{i-1})$  and  $q_i = s'(x_{i-1})$ , and

$$s_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2$$

Now

- At  $x_{i-1}$  for  $i = 1, 2, \dots, n$  we require  $s(x_{i-1}) = f(x_{i-1}) = f_i$  since

$$s_i(x_{i-1}) = a_i + b_i(x_{i-1} - x_{i-1}) + c_i(x_{i-1} - x_{i-1})^2 = a_i$$

we have

$$a_i = f_i, \quad i = 1, 2, \dots, n$$

- At  $x_{i-1}$  for  $i = 1, 2, \dots, n$  we require  $s'(x_{i-1}) = q_i$  since

$$s'_i(x_{i-1}) = b_i + 2c_i(x_{i-1} - x_{i-1}) = b_i$$

we have

$$b_i = q_i, \quad i = 1, 2, \dots, n$$

- Over each interval  $[x_{i-1}, x_i]$ ,  $s'_i(x)$  is a linear function and therefore equals its linear interpolant. Therefore

$$s'_i(x) = s'_i(x_{i-1}) + \left[ \frac{s'(x_i) - s'(x_{i-1})}{h} \right] (x - x_{i-1})$$

so

$$s'_i(x) = q_i + \left[ \frac{q_{i+1} - q_i}{h} \right] (x - x_{i-1})$$

Since  $s'_i(x_i) = b_i + 2c_i(x_i - x_{i-1})$  and  $b_i = q_i$  we have

$$2c_i = \frac{q_{i+1} - q_i}{h} \rightsquigarrow c_i = \frac{q_{i+1} - q_i}{2h}, \quad i = 1, 2, \dots, n$$

- There are  $n + 1$  values of  $q_i$  to be determined. To determine  $n$  equations one utilizes the relationship

$$\int_{x_{i-1}}^{x_i} s'_i(x) dx = s_i(x_i) - s_i(x_{i-1})$$

and the 3 facts

$$\begin{aligned} s'_i(x) &= q_i + \left[ \frac{q_{i+1} - q_i}{h} \right] (x - x_{i-1}) \\ s_i(x_{i-1}) &= f(x_{i-1}) = f_i \\ s_i(x_i) &= f(x_i) = f_{i+1} \end{aligned}$$

imply

$$\int_{x_{i-1}}^{x_i} q_i + \left[ \frac{q_{i+1} - q_i}{h} \right] (x - x_{i-1}) = f_{i+1} - f_i$$

so

$$q_i h + \left[ \frac{q_{i+1} - q_i}{h} \right] \frac{h^2}{2} = f_{i+1} - f_i$$

and thus

$$q_i + q_{i+1} = \frac{2}{h} (f_{i+1} - f_i), \quad i = 1, 2, \dots, n$$



There are  $n + 1$   $q_i$  values to be determined - so we need one more equation. There are several possibilities. One possibility, is to specify  $q_1$  by setting for example  $q_1 = s'(x_0) = \beta$ , or set  $q_1 = f'(x_0)$  if this is known, or  $q_1 = 0$ , or  $q_1 =$  one-sided difference approximation to  $f'(x_0)$  (in Assignment 10, you will be investigating these choices). There are others as well.

To get a nice matrix structure (a banded matrix) it is useful to specify the equation  $q_1 = \beta$  as the first equation and then the  $n$  other equations. With this indexing, the equations become

$$\begin{aligned} q_1 &= \beta \\ q_{i-1} + q_i &= \frac{2}{h}(f_i - f_{i-1}), \quad i = 2, 3, \dots, n + 1 \end{aligned}$$

**10.3. LU factorization.** Recall that Gaussian elimination is a row reduction (and possibly pivoting) to transform a matrix problem into an equivalent matrix problem involving an upper triangle from. Gaussian elimination can be encoded in matrix form, this encoding is the “LU factorization” (this corresponds to [2, Chapter 6.5]). To demonstrate this, some preliminaries.

A Gauss transformation matrix is an  $n \times n$  lower triangular matrix  $\tilde{L}^{(k)}$  all diagonal entries of which are 1, and except for diagonal and entries below diagonal in the  $k$ th column, all other values are 0. When one multiplies a matrix  $A$  by  $\tilde{L}^{(k)}$  on the left, then the result is a matrix where the  $k$ th row of  $A$  is multiplied by  $l_{j,k}$  and added to the  $j$ th row for  $j = k + 1, k + 2, \dots, n$ . With properly chosen values  $l_{j,k}$ , a Gauss transform matrix can encode the elimination of elements below the diagonal of  $A$ .

**Example 10.3.** Let

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix}$$

First step in reduction:  $[(-1) \cdot \text{row } 1] + \text{row } 2$ ,  $[(-2) \cdot \text{row } 1] + \text{row } 3$ . Let

$$\tilde{L}^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix}$$

so

$$\tilde{L}^{(1)}A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & -2 & -5 \end{bmatrix}$$

2nd step in reduction:

$$\tilde{L}^{(2)}(\tilde{L}^{(1)}A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & -2 & -5 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -1 \end{bmatrix}$$

The steps of transforming  $A$  to upper triangular form using Gaussian elimination can be “encoded” as

$$\tilde{L}^{(2)}\tilde{L}^{(1)}A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -1 \end{bmatrix}$$

In order to determine the  $L$  part of  $LU$  factorization, we need the following 2 facts about Gauss transformations (which you can check).

- (I) The inverse  $[\tilde{L}^{(k)}]^{-1}$  is the same as  $\tilde{L}^{(k)}$  except that  $l_{k+j,k}^{-1} = -l_{k+j,k}$  for all  $j = 1, \dots, n - k$  (just negate elements below the diagonal).
- (II) If  $p > m$ , then  $\tilde{L}^{(m)}\tilde{L}^{(p)}$  has except for the diagonal two columns with values below the diagonal, namely the  $m$ th and  $p$ th column, that are not zero, and contain the values of the respective columns in  $\tilde{L}^{(m)}$  and  $\tilde{L}^{(p)}$  (just superimpose  $\tilde{L}^{(m)}$  onto  $\tilde{L}^{(p)}$ ).

Using these facts once can encode Gaussian elimination (without pivoting) as

$$\tilde{L}^{(n-1)} \tilde{L}^{(n-2)} \dots \tilde{L}^{(2)} \tilde{L}^{(1)} A = U$$

where  $U$  is an upper triangular matrix. The entries of  $\tilde{L}^{(k)}$  are chosen to zero out elements of  $A$  below the  $k$ th diagonal. Now by fact (I),

$$A = [\tilde{L}^{(1)}]^{-1} [\tilde{L}^{(2)}]^{-1} \dots [\tilde{L}^{(n-2)}]^{-1} [\tilde{L}^{(n-1)}]^{-1} U$$

By fact (II),

$$A = LU$$

where

$$L = [\tilde{L}^{(1)}]^{-1} [\tilde{L}^{(2)}]^{-1} \dots [\tilde{L}^{(n-2)}]^{-1} [\tilde{L}^{(n-1)}]^{-1}$$

is a lower triangular matrix containing the negative values of each  $\tilde{L}^{(k)}$  below its diagonal respectively and the diagonal is filled with ones. So Gaussian elimination without pivoting is equivalent to the construction of a factorization of a matrix into a product of a lower triangular matrix  $L$  and an upper triangular matrix  $U$ .

In other words,  $L$  is unit lower triangle matrix with factors below the  $k$ th diagonal element that are the opposite of the multiplicative factors used to eliminate the entries below the diagonal in the  $k$ th step of Gaussian elimination, and  $U$  is the upper triangle matrix that results when one transforms  $A$  to upper triangle form using Gaussian elimination.

In the above example,

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -1 \end{bmatrix}$$

What about pivoting? Switching rows is encoded using a permutation matrix  $P$ . Switching two rows of  $A$  is the same as multiplying  $A$  on the left by the identity with the corresponding row switches.

**Example 10.4.** Suppose we want to change the 2nd with the 3rd row:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} \rightsquigarrow \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Then

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and we get

$$PA = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

During the Gaussian elimination process you can keep track of all pivots and express them as a single permutation matrix  $P$ . Then  $PA$  is a matrix whose  $LU$  factorization can be done without pivoting, so one obtains.

$$PA = LU$$

where  $P$  is a permutation matrix,  $L$  lower triangle, and  $U$  upper triangle.

What is the use of  $LU$  factorization? First another fact - solving a system using standard Gaussian elimination (as you do manually) requires  $O(n^3)$  operations. Observe that if we are given  $A$  and we want to solve  $A\vec{x} = \vec{b}$ , we can

- (1) Determine its  $LU$  factorization

$$A = LU$$

so that

$$A\vec{x} = \vec{b} \quad \Leftrightarrow \quad LU\vec{x} = \vec{b}$$

Needs  $O(n^3)$  number of operations.

- (2) Solve  $L\vec{z} = \vec{b}$  for  $\vec{z}$  which just requires back substitution and takes  $O(n^2)$  operations.
- (3) Solve  $U\vec{x} = \vec{z}$  for  $\vec{x}$  which just requires back substitution and takes  $O(n^2)$  operations.

Why bother? If  $A$  is  $n \times n$ , then (1) requires  $O(n^3)$  operations but (2) only requires  $O(n^2)$  operations. If we must solve many equations with the same  $A$ , say  $A\vec{x} = \vec{b}_1$ ,  $A\vec{x} = \vec{b}_2, \dots$  We need only do (1) once and then do (2) for each  $\vec{b}_i$ . Since (2) is  $O(n^2)$  this can be much more efficient than solving  $A\vec{x} = \vec{b}_i$  for each  $i$  just Gaussian elimination directly.

Observe:

$$\det(A) = \det(L) \det(U) = 1 \cdot \prod_{i=1}^n u_{i,i}$$

So to evaluate  $\det(A)$  requires  $O(n^3)$  work.

#### 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, 10<sup>th</sup> edition, 2015.
- [3] W. Cheney and D. Kincaid. *Numerical Mathematics And Computing*, Cengage Learning, 7<sup>th</sup> edition, 2012.

DEPARTMENT OF MATHEMATICS, UCLA, LOS ANGELES CA 90095-1555, USA.

E-mail address: [jasgar@math.ucla.edu](mailto:jasgar@math.ucla.edu)