

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.

John von Neumann

The material we will be studying in numerical analysis assumes a good foundation in calculus. In particular, we will have a need to use basic facts of continuity, differentiability, integration, and power series. In addition, we will present many of the ideas in the form of numerical algorithms, so a good working knowledge of some programming language will be needed. Our suggestions include Matlab, python, or R.

1.1 Frequently Used Theorems from Calculus

Numerical analysis relies on several fundamental theorems or analysis. We will refer to several of these repeatedly and have use of the following 4 in particular.

Theorem 1.1 (Mean Value Theorem) Suppose that (1) f is continuous on the closed finite interval $[a, b]$ and (2) $f'(x)$ exists for every x in the open interval (a, b) . Then there exists a point c such that

$$a < c < b$$

and

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

We will be looking at numerous cases of continuous functions over closed and bounded intervals. The following theorem will prove useful in our analyses.

Theorem 1.2 (Intermediate Value Theorem) Suppose that (1) f is continuous on the closed finite interval $[a, b]$ and (2) $f(a) < c < f(b)$. Then there exists some point $x \in [a, b]$ such that $f(x) = c$.

Remark: One way to interpret the IVT says is that if a continuous function on an interval takes on any 2 values, it takes on every value in between. This will be particularly useful in our analysis of root finding methods.

Similar to the MVT above, there is a variation that applies to integrals. It is well worth noting that in this case, there is an important assumption without which the theorem does not apply, so care must be taken when applying it to certain problems.

Theorem 1.3 (Weighted Mean Value Theorem for Integrals) Suppose that $f \in C[a, b]$, the Riemann integral of g exists on $[a, b]$, and $g(x)$ does not change sign of $[a, b]$. Then there exists a number $c \in (a, b)$ such that

$$\int_a^b f(x)g(x)dx = f(c) \int_a^b g(x)dx$$

1.2 Computer Programming

In terms of programming, we suggest one of 3 possible languages: Matlab, python, or R. Python and R have the advantage of being open-source and most of our examples will be in R and sometimes in python. In addition, both python and R can be easily installed on most computer platforms and both have powerful programming environments similar to Matlab. For python, one can use JupyterLab ([jupyter](#)); for R, one can use Rstudio/Posit ([Posit](#)).

We will also note that in real-world applications, most scientific codes will use other languages such as Fortran or C++. For the purposes of this introductory course, any high-level language will suffice.

1.3 Other Useful References

You should be able to find references to all of the material here in standard introductory courses on calculus. A good online reference for some of the material above can be found at [openstax.org](#).

- [Mean Value Theorem](#)
- [Intermediate Value Theorem](#)
- [Mean Value Theorem for Integrals](#)
- [Taylor polynomials and Taylor's Theorem](#)

Another good set of resources are the one-pagers provided by the UCM Math Center. You can check them all out at: [UCM The Math Center](#) and in particular the Math 23 refresher one-pager might prove useful: [Math 23 refresher](#).

2 Taylor's Theorem

All models are wrong but some models are useful

George E.P. Box

Topics Covered:

- How can we approximate mathematical functions that cannot be evaluated exactly?
- How does the approximation behave near the approximation point?
- Can we quantify the error in the approximation?

2.1 Taylor Polynomials

A fact that is highly underappreciated is that most functions in mathematics cannot be evaluated exactly. Common examples include functions such as $\exp(x)$, $\cos(x)$, $\ln(x)$.

This leads us to consider ways to approximate a function $f(x)$ by something else that is easier to compute. Let's consider the simple case of the exponential:

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

The first approximation we might try is $1 + x$, which is clearly easy to compute, but likely not very accurate. A second attempt might be to use the first three terms $1 + x + \frac{x^2}{2}$. We could proceed in the obvious way, adding new terms until we are satisfied or we get tired.

But there is an underlying question - when should we stop? One would hope that as k increases we should get more accuracy, but it's also clear that the more terms we use the harder is to compute the approximation.

Goal: Find functions that *approximate* f at some point, but with some *guarantee of accuracy*. Hopefully, these approximating functions are also *cheaper* to compute than the original function.

Important

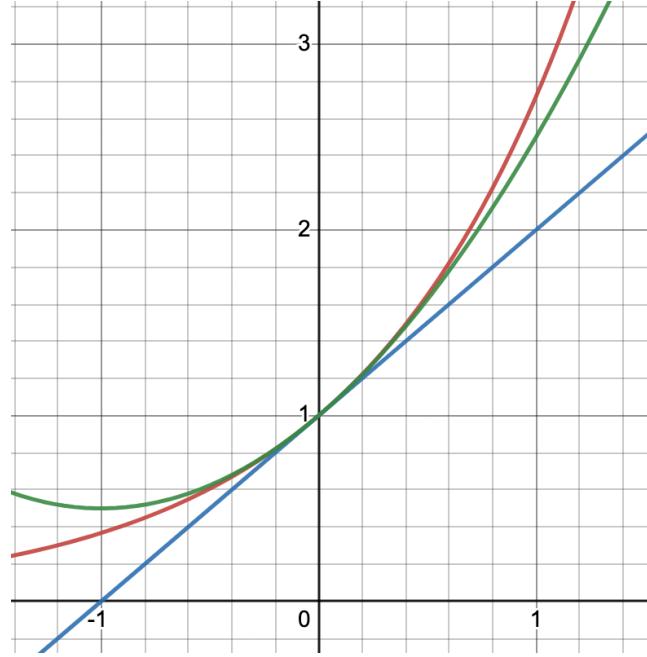
In numerical analysis there is a constant trade off between accuracy and computational work. We seek to balance these two goals for a given problem.

A first natural approach is to use a polynomial. They are 1) easy to understand, 2) easy to compute, and 3) in general easier to analyze. One commonly used approach is that of approximating a function via a Taylor polynomial. In addition, we will see that Taylor's Theorem with remainder can be used to analyze our approximations. You should be familiar with both the concepts and how to apply them in different situations.

Definition 2.1 (Taylor Polynomials) Suppose f is a function with n derivatives at the point $x = x_0$. Then the n th Taylor polynomial for f at x_0 is given by:

$$P_n(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \cdots + \frac{(x - x_0)^n}{n!}f^{(n)}(x_0) \quad (2.1)$$

Let's take a quick look at how some of the approximations work on $\exp(x)$.



First and second order approximations to e^x . Red is $\exp(x)$, green is $1 + x + x^2/2$, blue is $1 + x$

Some observations:

- As n increases, we might expect accuracy to increase
- As we move away from the selected point x_0 , we might expect the accuracy to decrease.

Using Taylor polynomials to approximate a function $f(x)$ leads to several natural questions:

1. How do we know all the derivatives exist?
2. What is the error when approximating $f(x)$ by $P_n(x)$?
3. Given a desired accuracy, how do we choose n ?

The question of the existence of all of the derivatives is a non-trivial one. We will have need of them in order to analyze the approximation, but in practice, one rarely has more than the first derivatives available. For the remainder of the course, we will assume that we have enough derivatives to allow us to proceed with our analysis, but the student should be aware that such assumptions are difficult to establish in real-world problems.

2.2 Taylor's Theorem

In order to answer questions 2-3 above, we will need the following theorem.

Theorem 2.1 (Taylor's Theorem with remainder) Let $f \in C^{n+1}$ on an interval I containing the real number x_0 . Also, let $P_n(x)$ be the n th Taylor polynomial of f at x_0 as defined by [Equation 2.1](#). Then for each x in I , we can write

$$f(x) = P_n(x) + R_n(x),$$

where $R_n(x)$ is given by:

$$R_n(x) = \frac{(x - x_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi),$$

for some ξ between x_0 and x .

Important

It is important to note that the value of ξ will depend on x , (and hence the derivative term $f^{n+1}(\xi)$).

Taylor's theorem will prove to be incredibly useful in our analysis of algorithms. In particular, we will use it to determine the accuracy of various approximations and more importantly it will be essential in the error analysis of algorithms.

Remark

We will have use of other forms of Taylor's Theorem - you should get comfortable recognizing and using them. For example, use the substitution $h = x - x_0$ to rewrite Taylor's Theorem.

2.3 Examples/Exercises

Example 2.1 Consider $f(x) = \exp(x)$. Evaluate the n th Taylor polynomial about the point $x_0 = 0$ and evaluate its remainder at x_0 .

Solution.

First we write down the Taylor Polynomial for $f(x) = \exp(x)$ at x_0 .

$$P_n(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \cdots + \frac{(x - x_0)^n}{n!}f^{(n)}(x_0)$$

Next we note that $f^{(k)}(x) = \exp(x)$, $k = 0, 1, 2, \dots$, so all of the derivatives exist. In particular, for the point $x_0 = 0$, we can also evaluate the function and all of its derivatives, $e^{x_0} = e^0 = 1, \forall k = 0, 1, \dots$

That means we can write the Taylor polynomial and the remainder term as follows:

$$P_n(x) = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$$

$$R_n(x) = \frac{x^{n+1}}{(n+1)!} e^{\xi(x)},$$

Exercise 2.1 (In class exercise) Consider $f(x) = \sqrt[3]{x}$. a) Find the first and second Taylor Polynomials for $f(x)$ at $x_0 = 8$. b) Evaluate both Taylor polynomials at the point $x = 11$

.

Solution.

a. Let's break this down into steps:

Step 1. Writing out the first two polynomials using Taylor's theorem we see that:

$$P_1(x) = f(x_0) + (x - x_0)f'(x_0)$$

$$P_2(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0)$$

Step 2. Now let's substitute for the point $x_0 = 8$:

$$P_1(x) = f(8) + (x - 8)f'(8)$$

$$P_2(x) = f(8) + (x - 8)f'(8) + \frac{(x - 8)^2}{2!}f''(8) \quad (2.2)$$

Step 3. The next step is to calculate the various derivative terms in the polynomial and evaluate them at the point $x_0 = 8$. It is helpful to write out a table that includes all of the necessary terms as shown below.

Taylor Polynomial terms for $\sqrt[3]{x}$

Term	Evaluation at $x_0 = 8$
$f(x) = \sqrt[3]{x}$	$f(8) = \sqrt[3]{8} = 2$
$f'(x) = \frac{1}{3x^{2/3}}$	$f'(8) = \frac{1}{3 \cdot 8^{2/3}} = \frac{1}{12}$
$f''(x) = \frac{-2}{9x^{5/3}}$	$f''(8) = -\frac{2}{9 \cdot 8^{5/3}} = -\frac{1}{144}$

Step 4. Finally, inserting these values into [Equation 2.2](#) yields:

$$P_1(x) = 2 + \frac{(x - 8)}{12}$$

$$P_2(x) = 2 + \frac{(x - 8)}{12} - \frac{(x - 8)^2}{288}$$

b. Estimate $f(11) = \sqrt[3]{11}$

$$P_1(11) = 2 + \frac{(11 - 8)}{12} = 2.25$$

$$P_2(11) = 2 + \frac{(11 - 8)}{12} - \frac{(11 - 8)^2}{288} = 2.21875$$

For comparison, $\sqrt[3]{11} \approx 2.22398009$

2.4 Other forms for the Remainder Term

There are alternate forms for the remainder term that you may see, the most common of which is the *integral form*, which is given by:

$$R_n(x) = \frac{1}{n!} \int_{x_0}^x (x - t)^n f^{(n+1)}(t) dt,$$

It is fairly easy to show that the two forms are equivalent through the use of the Weighted Mean Value Theorem for Integrals ([Theorem 1.3](#)). First note that the first term $(x - t)^n$ does not change sign over the interval $[x_0, x]$. Also by assumption $f^{(n+1)}$ is continuous on the same interval. As a result, we can use the WMVTI as follows.

$$\begin{aligned} \int_{x_0}^x (x - t)^n f^{(n+1)}(t) dt &= f^{(n+1)}(\xi) \int_{x_0}^x (x - t)^n dt, \\ &= f^{(n+1)}(\xi) \cdot \frac{-(x - t)^{n+1}}{n + 1} \Big|_{x_0}^x, \\ &= f^{(n+1)}(\xi) \cdot \frac{(x - x_0)^{n+1}}{n + 1}. \end{aligned}$$

The last step follows since the integrand vanishes at the upper endpoint. If we now multiply both sides by $1/n!$ we get the desired result.

Remark

There are other forms of the remainder term as well. The ones presented here are the most common. Which form you decide to use will depend on the particular situation. But it's nice to have options.

2.5 Choosing $P_n(x)$ to achieve a desired accuracy

This now leaves us with our final question: can we determine what degree polynomial is required for a given accuracy?

Example 2.2 Once again, let's consider $f(x) = \exp(x)$. Evaluate the n th Taylor polynomial about the point $x_0 = 0$. How big should n be to approximate $\exp(1)$ to an accuracy of 10^{-9} .

Solution.

Recall that

$$R_n(x) = \frac{x^{n+1}}{(n + 1)!} \cdot e^\xi$$

Evaluating at $x = 1$ gives us

$$R_n(x) = \frac{1}{(n+1)!} \cdot e^\xi$$

Our overall goal is to bound the remainder (error) such that:

$$l \leq R_n(1) \leq u$$

where l, u are some chosen bounds. In general it will be easier to work with the following form:

$$|R_n(1)| \leq M$$

for some value of M , in this case e.g. we would let $M = 10^{-9}$.

As with many cases in numerical analysis, we will need to make use of any information we may have about the function and the domain we're working with. In particular for this function we know that, $0 < \xi < 1$, by Taylor's Theorem. That means that:

$$1 = e^0 < e^\xi < e^1 = e$$

dividing by $(n+1)!$ we see that:

$$\frac{1}{(n+1)!} < \frac{e^\xi}{(n+1)!} < \frac{e}{(n+1)!}$$

Note that the middle term is just $R_n(1)$. Since we're not supposed to actually know the value of e , let's just assume that we know a rough upper bound, say 3, (but we could just as easily have used any other number) and substitute it into the last term. Then what we're really saying is that we would like to have:

$$|R_n(1)| < \frac{3}{(n+1)!} < 10^{-9}$$

A quick calculation suggests that $n+1 \geq 13$ satisfies this condition, so that a 12th degree Taylor polynomial should give us the desired accuracy.

Exercise 2.2 (In class exercise)

- Write out the n th Taylor Polynomial for $f(x) = \sin(x)$ about $\pi/4$.
- Find the smallest degree Taylor polynomial such that the remainder

$$|R_n(\pi/4)| < 10^{-6}$$

on $[0, \pi/4]$.

2.6 Summary

- We showed that we can approximate a given function through the use of a Taylor Polynomial.
- Taylor's Theorem allows us to write down exactly what the remainder (error) term is.

- Two (equivalent) forms of the remainder term were introduced: Lagrange form and the Integral Form.
- We can estimate the degree of the Taylor Polynomial needed to guarantee a desired accuracy over a given interval.
- Useful bounds will depend on knowledge of the given function and its derivatives.

2.7 References

A good online reference for some of the material above can be found at openstax.org.

- [Taylor polynomials and Taylor's Theorem](#)

Another good set of resources are the one-pagers provided by the UCM Math Center. You can check them all out at: [UCM The Math Center](#) and in particular the Math 23 refresher one-pager might prove useful: [Math 23 refresher](#).

And of course, there's always Wikipedia:[Taylor's Theorem](#)

Revised: Tuesday, Sept. 5, 2023

3 Errors

There will always be a small but steady demand for error analysts to ... expose bad algorithms' big errors and, more importantly, supplant bad algorithms with provably good ones

W. Kahan (1980)

There's no sense in being precise when you don't even know what you're talking about

John von Neumann

Topics Covered:

- What types of errors might one encounter in numerical computing and what are their sources?
- When is a computed solution accurate enough?
- What is the difference between precision and accuracy?

3.1 Sources of Errors

One of the first challenges when working in numerical analysis is the question of how to determine when a computed solution is sufficiently accurate. In order to talk about this we first need to develop some nomenclature and definitions that will allow us to quantify more precisely how good a solution we have.

Definition 3.1 Suppose we have x and an approximation \hat{x} . Then the **absolute error** is given by

$$|x - \hat{x}|.$$

Definition 3.2 Similarly the **relative error** is defined by

$$\frac{|x - \hat{x}|}{|x|}.$$

Both absolute and relative errors are used in our analysis, but the relative error is generally preferred in practice as it is scale independent, and as long as you stay away from $x = 0$.

Precision and Accuracy

Precision and accuracy are often confused. In numerical analysis, they will have a specific meaning.

By **accuracy**, we mean the absolute or relative error of an approximation as defined in ([Definition 3.1](#),[Definition 3.2](#)). **Precision** will refer to the accuracy with which the basic arithmetic operations (+, -, *, /) are performed. More of this to come.

One should also note that accuracy is not limited by the precision of a computer - there are many software packages that can provide extended precision in numerical calculations.([Bailey 1993](#))

3.2 Sources of Errors

The next question that naturally arises is what types of errors might we encounter in solving a problem numerically and where might they arise? As it turns out, there are many different ways to classify errors. We will use the following general categories:

- **model errors/uncertainty:**

- errors in mathematical models used to approximate a real world problem
- errors in the input data, e.g. physical measurements, observations, etc.

- **approximation errors:**

- errors in approximating our problem due to truncation/discretization ,
e.g. Taylor series, interpolation, integration.

- **roundoff errors:**

- errors due to the finite precision inherent of computer arithmetic. This is a basic fact of computational mathematics and there is not much we can do to limit these. However, wise choices of algorithms can circumvent some of the problems and bad choices of algorithms can exacerbate them.

(!) Remarks

1. Model errors and their quantification is a subject area all of its own - we will not cover it.
The interested student can check out the vast literature on topics such as *uncertainty quantification, verification, and validation*.

2. Approximation errors (truncation/discretization) usually dominate other types of errors and their analysis is a major task of numerical analysis

Example 3.1 (Approximation/Discretization) Error)

Suppose we want to compute an approximation to $f'(x)$ for some function $f(x)$ at the point $x = x_0$.

This situation might arise if you have the function but do not know $f'(x)$, or perhaps it is too expensive to compute.

Let's use Taylor's Theorem. Recall

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \cdots + \frac{(x - x_0)^{(n+1)}}{(n+1)!}f^{(n+1)}(c).$$

for some c between x and x_0 . Note, that we need to assume f has $n+1$ derivatives in some interval.

Now let's introduce some new notation. In particular, let

$$x = x_0 + h, \quad h > 0,$$

so that

$$h = x - x_0.$$

Then we can rewrite the Taylor expansion as

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \cdots + \frac{h^{(n+1)}}{(n+1)!}f^{(n+1)}(c).$$

Rearranging the terms to put the derivative on the left hand side of the equation gives us:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \left[\frac{h}{2}f''(x_0) + \cdots + \frac{h^{(n)}}{(n+1)!}f^{(n+1)}(c) \right].$$

Idea

Use the first two terms on the RHS to approximate the derivative at x_0 and “throw away” the rest of the terms.

Taking absolute values, we can see that:

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} \right| = \left| \frac{h}{2}f''(x_0) + \cdots + \frac{h^{(n)}}{(n+1)!}f^{(n+1)}(c) \right|, \quad (3.1)$$

where we denote the RHS of [Equation 3.1](#) by the **discretization error**. If $f''(x_0) \neq 0$, then for small enough h , we can estimate the discretization error by:

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} \right| \approx \left| \frac{h}{2}f''(x_0) \right|.$$

Again, we are discarding all higher order terms, which if h is “small enough” seems like a reasonable idea (of course we will need to prove this rigorously, but that comes later). In the meantime, it will prove helpful to denote quantities like the right hand side by the following notation:

$$\left| \frac{h}{2} f''(x_0) \right| = O(h).$$

Here by the notation, $O(h)$ (read Big O of h), we mean that a quantity is at most proportional to h , for example Ch , for some constant C . One way to think of it is as

$$\lim_{h \rightarrow 0} \frac{O(h)}{h} = C < \infty$$

This can be generalized to higher orders of h as well. (We'll talk more about this later as well.)

The important concept is that we talk about the above *approximation for the derivative as being an $O(h)$ approximation to the true derivative*. For more details (or if you need a refresher) on Big O notation see [Section A.1](#).

Example 3.2 Approximate $f'(x)$ for $f(x) = \cos(x)$ and $h = 10^{-3} - 10^{-15}$, $x = \pi/6$.

h	F.D	aberr	relerr
1e-01	-0.5424323	4.243228e-02	8.486456e-02
1e-02	-0.5043218	4.321758e-03	8.643515e-03
1e-03	-0.5004329	4.329293e-04	8.658587e-04
1e-04	-0.5000433	4.330044e-05	8.660088e-05
1e-05	-0.5000043	4.330117e-06	8.660235e-06
1e-06	-0.5000004	4.330569e-07	8.661137e-07
1e-07	-0.5000000	4.359063e-08	8.718126e-08
1e-08	-0.5000000	8.063495e-09	1.612699e-08
1e-09	-0.5000000	4.137019e-08	8.274037e-08
1e-10	-0.5000000	4.137019e-08	8.274037e-08
1e-11	-0.5000000	4.137019e-08	8.274037e-08
1e-12	-0.5000445	4.445029e-05	8.890058e-05
1e-14	-0.4996004	3.996389e-04	7.992778e-04
1e-15	-0.5551115	5.511151e-02	1.102230e-01

Table 3.1: Absolute and Relative Error in finite difference (F.D) approximation as a function of stepsize h.

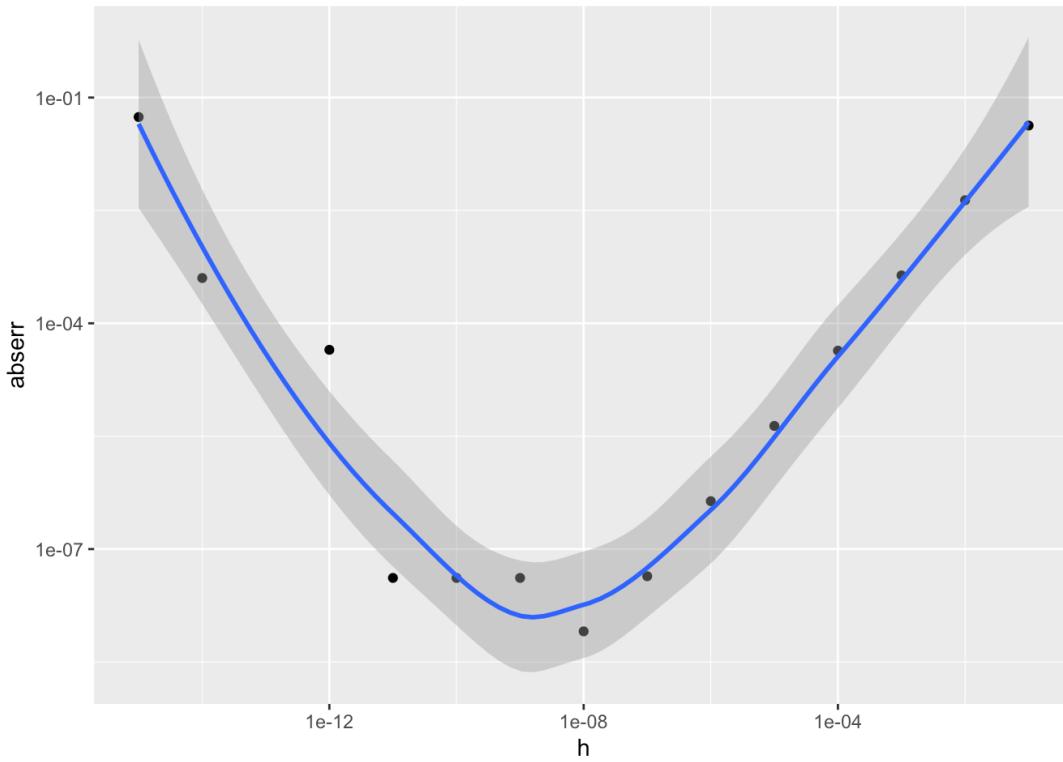


Figure 3.1: Absolute Error in finite difference approximation as a function of stepsize h . Notice the decrease in error as h decreases. Eventually the error increases as roundoff error dominates the truncation error.

⚠ Remark

Notice that for each decrease in the value of h by an order of magnitude, both the absolute and relative error have a corresponding decrease in their values. *This is exactly what the theory predicts.* However, it is important to note that this is true only up to a certain point. We'll discuss this more fully when we get to the sections on computer arithmetic and roundoff error.

Exercise 3.1 Approximate $f'(x)$ for $f(x) = \sin(x)$ at the point $x_0 = 1.2$ and using $h = 0.1, 0.01, 0.001$. Can you estimate the discretization error? Explain.

Solution.

3.3 Summary

This section covered

- the concepts of absolute error and relative error,
- the difference between accuracy and precision,
- presented some of the most common sources of errors when modeling problems through the use of mathematical or computational models,
- the effects of discretization error and,
- provided an example for approximating the first derivative of some known function.

► Code

```
[1] "Revised: March 05 2024"
```

Stability and Condition Numbers

Math 131: Numerical Analysis

J.C. Meza

2/1/24

Section 1

Introduction: Algorithms

Topics Covered:

- What is an algorithm?
- What is a stable algorithm?
- What types of problems are amenable to accurate solutions?
- What is the condition number of a problem?

What is an algorithm?

- Algorithms have a long history dating back over 4000 years. The name itself is said to be derived from the mathematician al-Khwārizmī (c.780-c.850), who wrote a book on the subject.
- One can think of an algorithm as a step by step procedure for solving some problem. A common analogy is that of a recipe in which one is given explicit instructions on how to make something.
- Can also think of an algorithm as a systematic calculation or process for achieving some desired result.
- More recently, the notion of *finiteness* has been added to the concept although it is not essential.

For our purposes we will use the following definition:

An **algorithm** is a well-defined process that takes an input (or inputs) and produces an output in a finite-number of steps or time.

Desired Properties of Algorithms

- What kind of properties should a good numerical algorithm have?
- While one can come up with a long list of desirable characteristics for an algorithm, we will concentrate on three that were briefly introduced on the first day of class:
 - ① Accuracy
 - ② Efficiency
 - ③ Robustness
- Let's take a look at each of these in turn.

Section 2

Accuracy

Accuracy of an algorithm

- Since a numerical algorithm is designed to solve some mathematical problem, one of the key characteristics should be that it provides us with an accurate approximation to the true solution, whatever that might be.
- We've already seen some examples of these and how we can use error analysis to allow us to predict the behavior of an algorithm.
- Here we are free to use any of a number of measures (metrics): absolute error, relative error, residual, etc.

Section 3

Efficiency

Preliminaries: Big O Notation

- Big O Notation is frequently used both in mathematics and computer science.
- While it can be confusing at first, the thing to remember is that it is mostly a means to characterize and understand how an algorithm or an approximation behaves asymptotically.
- In fact, one of the chief advantages to using this notation is that it hides some of the detailed information that isn't usually useful to the analysis.

Uses of Big O Notation

There are two main uses that you will see for Big *O* Notation in this class.

- ① Understand how an approximation to a given function behaves - for example, when we use Taylor's Theorem to approximate a function. In these cases, we usually use h or x for our notation and implicitly assume that h or x are small or tending to zero.
- ② The computational workload of an algorithm as a function of the dimension of the problem, typically denoted by n . In these cases, we usually assume that n is large or tending to infinity.

The second case is more prominent in computer science applications when analyzing the complexity of an algorithm, but we will have use for it as well in numerical analysis.

Motivation for Big O in computational workload

- It is important to understand how the workload will behave as the size of a problem increases. An algorithm that works fine on a problem of dimension 1, 2, or 3, can be prohibitively expensive when applied to a problem with dimension 1,000,000.
- Many algorithms can be quite complicated. The power of Big O notation is that it is a high-level description that allows us to quickly say something about an algorithm or an approximation without getting into all the messy details.
- In addition, counting every single arithmetic operation could be tedious (and with modern computers not as important).
- Big O can be used to compare two algorithms to see which one is more efficient.

One definition for Big O

We say that $g(n)$ is $O(f(n))$ if:

$$g(n) \leq C f(n), \quad \forall \text{ integers } n \geq n_0$$

for some constants $C > 0, n_0 > 0$.

Example:

$$3n^3 + 2n^2 + 5 = O(n^3)$$

because

$$3n^3 + 2n^2 + 5 \leq 4n^3, \quad n \geq 3$$

- It should also be noted that neither one of the constants C, n_0 are (usually) known.
- For more information see [[@knuth1997](#)],

Caution on Big O Notation

- In all cases the statements are understood to be ***in the limit.***
- For specific cases there may be (and usually are) counterexamples.

Caution

An important point to remember is that we are only interested with how the computational workload of an algorithm behaves as the dimension increases, and ***not with the details of the computation.***

Example of Big O

Suppose we wanted to compute the mean of a set of n numbers:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

It is easy to see that the formula entails adding n numbers, which amounts to $n - 1$ additions. We then have to divide by n , so there's 1 division. The total is therefore n floating point operations (*flops*).

Everything else being equal, that's also the computational workload for this simple algorithm.

- If $n = 100$, then we have 100 *flops*;
- if $n = 1,000,000$ then we have 1,000,000 *flops*.

We say that this algorithm is $O(n)$, because the workload increases linearly with the dimension of the problem n .

Example 2 of Big O

Let's consider the problem of evaluating a polynomial given by:

$$p_n(x) = c_0 + c_1x + c_2x^2 + \dots + c_nx^n.$$

- A naive algorithm would require $O(n^2)$ operations or **flops** (floating point operations) as they are called. To see this all you need to do is think about each of the terms in the polynomial in turn and add up all of the operations.
- For example, the last term requires n multiplications: $n - 1$ to compute x^n plus one more to multiply by c_n . Each term of lower order will require one less multiplication.
- In all, there are $n + (n - 1) + (n - 2) + \dots + 1 = O(n^2)$ multiplications. Finally there are the n additions required to sum up all of the individual terms.

Nested form

Consider rewriting the formula in what is known as the ***nested form***:

$$p_n(x) = c_0 + x(c_1 + x(c_2 + \cdots + x(c_{n-1} + x(c_n)) \cdots)),$$

which only requires $O(n)$ operations. (You should work this out for yourself.)

- This may not make much of a difference for small values of n , but one nonetheless needs to be careful as generally speaking we don't know ahead of time how an algorithm will be used.
- For example, this polynomial evaluation could be at the heart of an algorithm that is called millions of times.

A note on Flops

- Historically, the operation count (flops) has been an important measure of efficiency (or at least workload).
- SuperComputing Top500: <https://www.top500.org/> lists top 500 supercomputers in the world
- On today's computers, and especially supercomputers, there are many more things to consider when considering computational efficiency:
 - ▶ memory access,
 - ▶ communication,
 - ▶ vectorization/parallelization,
 - ▶ etc.

Section 4

Robustness

Robustness in plain terms

- Finally, we would like an algorithm that we can “trust”, in the sense that it either gives us an answer or reports back that something went wrong.
- Ideally, an algorithm should also work under most foreseeable circumstances and different values of inputs.
- Another thing to watch out for is the rate of accumulation of errors within an algorithm.

Tip

You should be on the lookout for any calculations that might cause an IEEE floating point exception: overflow, underflow, infinity, NaN.

Section 5

Condition Numbers and Stability

Condition Numbers

- The first concept is that of the problem sensitivity or as it is more commonly referred, ***condition numbers***.
- By this we mean that a problem is ***well-conditioned*** if small changes in the inputs do not lead to large changes in the outputs.
- Otherwise the problem is said to be ***ill-conditioned***.

III-Conditioned Problem

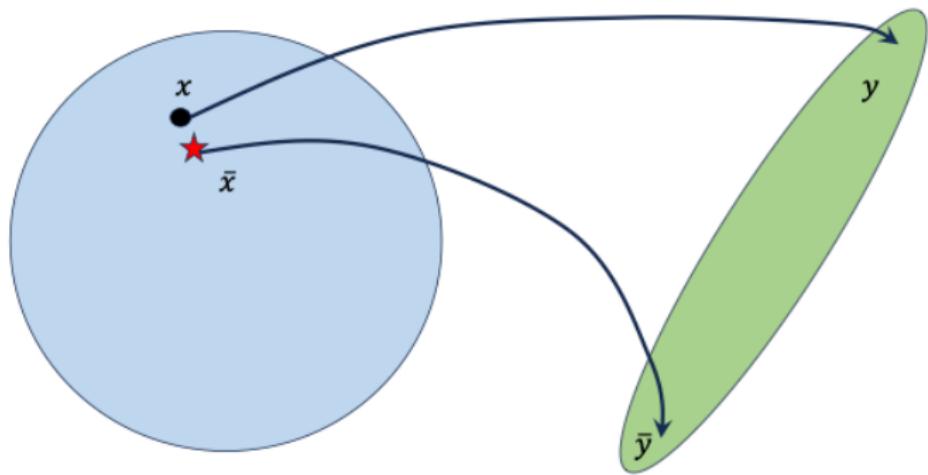


Figure 1: Ill-conditioned problem where small changes in the input can lead to large changes in the output

Example: Well-conditioned problem

Let's take a look at a simple example of a well-conditioned problem first.

Consider the function $f(x) = \sqrt{1+x}$ with $f'(x) = \frac{1}{2\sqrt{1+x}}$.

Let's suppose that we fix $|x| << 1$ and consider $\bar{x} = 0$ as a small perturbation of x , and $y = f(x)$. Then

$$\bar{y} = f(\bar{x}) = \sqrt{1+0} = 1,$$

and the change in the output can be given by

$$y - \bar{y} = \sqrt{1+x} - 1.$$

Example (cont.)

Approximating $\sqrt{1+x}$ by the first 2 terms of a Taylor series about 0,

$$f(x) \approx 1 + \frac{x}{2}.$$

Substituting into the prior equation we can write the change in the output:

$$y - \bar{y} \approx \left(1 + \frac{x}{2}\right) - 1 = \frac{x}{2}.$$

This can then be rewritten in terms of the change in the inputs as:

$$y - \bar{y} \approx \frac{1}{2}(x - \bar{x}).$$

Notice that if $x = 0.001$ then the change in the output is only $y - \bar{y} \approx 0.0005$, one half of what the change in input was.

Condition Number

- We're now ready for a more rigorous definition of condition number.
- First, note that when we introduced the concept of a condition number it was to describe the sensitivity of a problem (function) to change in the input data x .
- As such it seems natural to look at:

$$\frac{\text{Relative change in } f(x)}{\text{Relative change in } x}.$$

Mathematically we can translate this into:

$$\begin{aligned}\frac{\text{Relative change in } f(x)}{\text{Relative change in } x} &= \frac{\frac{f(x) - f(\bar{x})}{f(x)}}{\frac{x - \bar{x}}{x}} \\ &= \frac{f(x) - f(\bar{x})}{f(x)} \cdot \frac{x}{x - \bar{x}}, \\ &= \frac{f(x) - f(\bar{x})}{x - \bar{x}} \cdot \frac{x}{f(x)}, \\ &\approx \frac{f'(x) \cdot x}{f(x)}.\end{aligned}$$

This leads us to the following definition:

The quantity

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

is called the (relative) condition number of the problem.

- If κ is large, we say that the problem is ***ill-conditioned***.
- More on this important topic later!

Looking ahead

- The condition number of a problem is especially useful in numerical linear algebra, where it can be used to estimate the accuracy of the solutions for systems of linear equations, such as $Ax = b$.
- Also important for many partial differential equations, inverse problems, etc.

Section 6

Stability

Stability of algorithms

- The second important concept is that of stability in an algorithm.
- Here we say that an algorithm is ***stable*** if the output generated is the exact result for a slightly perturbed input.
- Another way of phrasing this is that a stable algorithm solves a ***nearby problem***.

Stability (cont.)

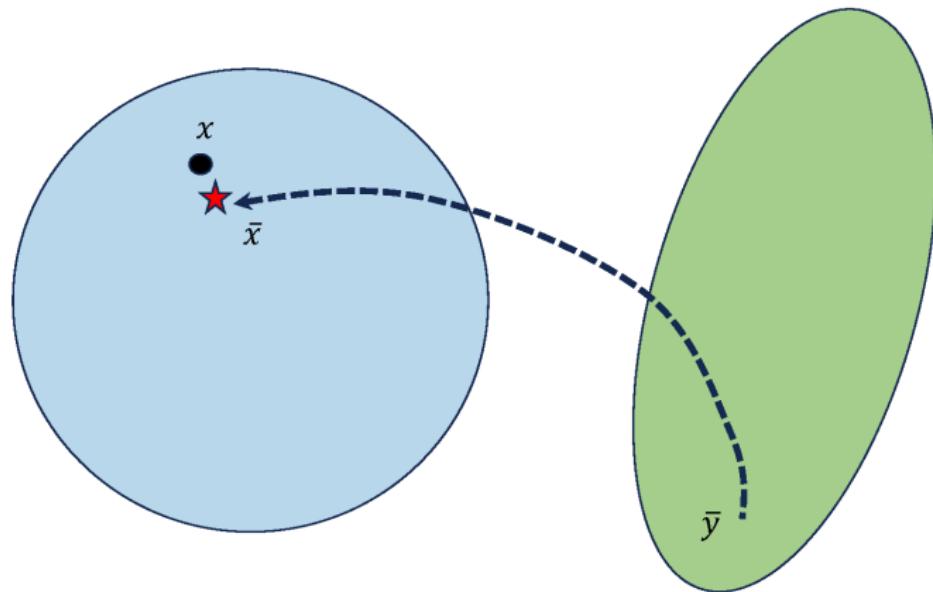


Figure 2: A stable algorithm is one that is the exact solution to a slightly perturbed problem.

Roundoff error and stability

- It is natural to expect that any algorithm will produce some roundoff error. In fact, it is common that accumulation of roundoff error will occur as we progress through a computation.
- If the error grows slowly as the computation proceeds it won't be a problem, but if it grows too fast then we need to be more careful.
- Let E_n be the error at the n th step of some computation. If the error $E_n \approx C \cdot nE_0$, where C is a constant and E_0 is the original error, then the growth of error is said to be *linear*.

Stability (cont.)

- If the error $E_n \approx C^n E_0$, where C is a constant and E_0 is the original error, then the growth of error is said to be ***exponential***.
- An algorithm that has an exponential error growth rate is said to be ***unstable***.
- We should note that an algorithm could be stable for solving one problem, but unstable for solving another problem.
- The stability of algorithms will become important in our study of differential equations (e.g. IVPs)

Tips on designing stable algorithms

While there are no hard and fast rules, there are several things to watch out for in designing an algorithm. The tips below are paraphrased from the excellent discussion given in Higham [[@higham2002](#)].

- Watch out for cancellation errors. Try to avoid subtracting quantities of near equal magnitude, especially if they are contaminated by error.
- Look for different formulations that are mathematically equivalent, but perhaps numerical better.
- It can be useful to write formulas in the form of:

$$x_{\text{new}} = x_{\text{old}} + \Delta x$$

where Δx is a small correction to the previous (old) value. You'll see that many numerical methods take this form.

Tips on designing stable algorithms (cont.)

- Minimize the size of intermediate results relative to the final result.
It's also a good idea to check the intermediate results when first writing a code.
- Avoid ill-conditioned transformations of the problem.
- Take precautions against underflow and overflow.

Summary

This lesson covered the fundamental concepts of condition numbers of numerical problems and stability of algorithms. Key takeaway messages include:

- The condition number of a problem, which speaks to the sensitivity of the outputs to the inputs is related to the ***relative change in the output of a problem to the relative change in the inputs.***
- All algorithms will generate errors as a consequence of doing computer arithmetic. If the error growth rate is reasonable, for example linear with respect to the number of steps in the calculation, then we can expect good results (if the problem is also well-conditioned).
- If the error growth rate is exponential instead, then the algorithm is said to be ***unstable***.
- An algorithm can be stable for one type of problem but unstable for another type of problem.

References

- ① Math 131 Lecture Notes
- ② Big O notation, https://en.wikipedia.org/wiki/Big_O_notation
- ③ The Art of Computer Programming, [@knuth1997]

Nonlinear Equations: Bisection

Math 131: Numerical Analysis

J.C. Meza

2/6/24

Nonlinear equations

Suppose that we have a scalar, nonlinear equation $f : \mathbb{R}^1 \rightarrow \mathbb{R}^1$ on an interval $[a, b]$

If $x^* \in \mathbb{R}$ is such that

$$f(x^*) = 0, \tag{1}$$

then we will call x^* a *zero* or *root* of f .

- Standard assumption is that $f(x)$ is continuous on the closed interval $[a, b]$.

Some Examples

- ① $f(x) = 6x^2 - 7x + 2 = (2x - 1)(3x - 2)$
- ② $f(x) = 2^{x^2} - 10x + 1$
- ③ $f(x) = \cosh(\sqrt{x^2 + 1}) - e^x) + \log |\sin(x)|$
- ④ $f(x) = \blacksquare \leftarrow \text{some black box}$

Relation to optimization

Many of the methods that we study in this section are also applicable to the problem of optimization. For example:

$$\min f(x)$$

- A minimum will occur at points where $f'(x) = 0$.
- Special care must be taken that the method approaches a minimum and not a maximum or an inflection point, but this is usually not hard to handle.

Notes on general setting

- Except for some special cases, nonlinear equations will not have a closed form solution, so we are naturally led to developing alternative means for solving Equation 1.
- We could have just as easily defined the problem in higher dimensions, i.e. $f : \mathbb{R}^n \rightarrow \mathbb{R}^p$, for $n > 1, p > 1$, and n not necessarily equal to p .
- This would most likely be the case for a real-world problem, but while many of the ideas (and algorithms) can be generalized to higher dimensions, the concepts are easier to understand in one dimension to begin with.
- We note in passing that if $n = p$, then this is an example of solving a ***nonlinear system of equations***. If $n \neq p$, then this is an example of a ***nonlinear least squares*** problem.

General Iterative Framework

Most nonlinear equations cannot be solved analytically and as a result most approaches involve some form of iteration, which can be described at a high-level as:

- ① First guess a solution,
- ② Check to see how accurate it is, and if not satisfied,
- ③ Update the guess and try again, i.e. go back to step 1

This is the essence of an ***iterative method***.

Challenges

- Using an iterative technique to find roots of equations can be tricky.
- The initial guess can sometimes be important, and if not chosen properly can lead to slow convergence or even non-convergence.
- How one updates the guess is also important. Without some theory behind the updating scheme, an iterative method is nothing more than trial and error.
- Finally, there is always the question as to when to terminate an iterative method, in other words when is an estimate of the root “good enough”?

Note

For all of these reasons, having an algorithm based on sound theoretical foundations is of fundamental importance.

Overview

In the next few sections, we discuss some of the more popular iterative methods for finding the roots of a nonlinear equation and provide some general guidance for when to use them. In particular, we will study:

- ① bisection method,
- ② Newton's method,
- ③ secant method and
- ④ fixed-point iterations

Bisection (Quick Summary)

- One of the simplest methods for solving a nonlinear equation is known as the bisection method.
- The main advantage is the robustness of the method - if the method is applicable to the problem, it is guaranteed to find a solution.
- On the other hand, the method can often take far more iterations than some of the other methods we will discuss.

Let's first describe the general method visually

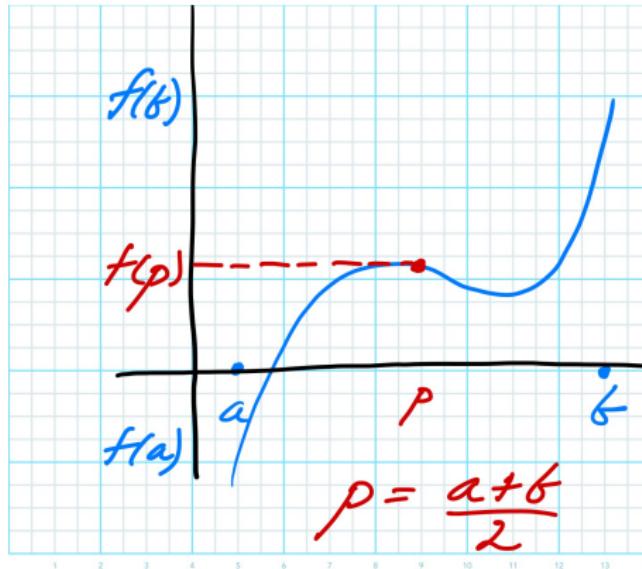


Figure 1: Bisection Method

Idea

- Suppose that we happen to know two points where the function is of opposite sign as in Figure 1.
- Using the Intermediate Value Theorem (IVT) we also know that if the function is continuous, then the function must take every value in between the two values:

Important

In particular, ***it must be equal to 0 somewhere in the interval.***

Mathematically

If $f(x) \in C[a, b]$ and $f(a) \cdot f(b) < 0$, then there exists an $x^* \in [a, b]$ such that $f(x^*) = 0$. This leads us to the following general procedure:

- ① Start with an interval $[a, b]$ such that $f(a) \cdot f(b) < 0$.
- ② Cut the interval in half (bisect).
- ③ Evaluate the function f at the midpoint of the interval.
- ④ Choose whichever sub-interval contains a sign change.
- ⑤ Repeat as necessary.

Start by writing this in pseudocode:

Given a_0, b_0 s.t. $f(a_0)f(b_0) < 0$

for $k=0, \dots, \text{maxiter}$ **do**

$mid \leftarrow (a_k + b_k)/2$

if $f(a_k)f(mid) < 0$ **then**

$a_{k+1} \leftarrow a_k; b_{k+1} \leftarrow mid$

else

$a_{k+1} \leftarrow mid; b_{k+1} \leftarrow b_k$

end if

end for

Figure 2: Bisection Algorithm

Example

Consider $f(x) = x^3 + 4x^2 - 10 = 0$ on $[1, 2]$ where $x^* = 1.365$. We should first check that the function has opposite signs on the given interval:

$$f(1) = 1 + 4 - 10 = -5$$

$$f(2) = 8 + 16 - 10 = 14$$

$$f(1) \cdot f(2) < 0 \quad YES!$$

- ① Compute $p = \frac{a+b}{2} = \frac{1+2}{2} = 1.5$
- ② Evaluate $f(1.5) = 3.375 + 9 - 10 = 2.375$
- ③ $f(a) \cdot f(p) = f(1) \cdot f(1.5) < 0$ so we choose left interval and set $b = p$
- ④ Repeat

After 13 iterations $p = 1.365112305$, $f(p) = -0.00194$, and $|b_{14} - a_{14}| = 0.000122070$.

Let's take a look at how this pseudocode could be implemented in python.

```
import numpy as np
def bisect(a0, b0, ftol, maxiter=30):
    ak = a0
    bk = b0
    iters = 0
    for k in range(1, maxiter):
        mid = (ak + bk)/2
        fmid = fx(mid)
        if (fx(ak)*fmid < 0):
            bk = mid
        else:
            ak = mid
        if (np.abs(fmid) < ftol):
            iters = k
            break
```

Let's define a function and call the bisection algorithm.
For this example, let's use the function:

$$f(x) = (x - 1.5)^3 - x + 2$$

on the interval $I = [a, b] = [0, 2.5]$.

```
# Example function
```

```
def fx(x):
    fvalue = (x-1.5)**3 - x + 2
    return fvalue
```

Call the bisection algorithm.

For initial values we'll use an initial starting guess of $a_0 = 0.0$, $b_0 = 2.5$ and a function tolerance of $ftol = 10^{-6}$

```
#Initialize
a = 0.0
b = 2.5
ftol = 1.e-6

# Check assumptions
fa = fx(a)
fb = fx(b)
print ("fa = %f, fb = %f" %(fa, fb))

# Call Bisection function
xstar = bisect(a, b, ftol)
```

When should we stop?

- Before proceeding further, we should discuss when and how to terminate an iterative algorithm. This decision is one of great importance in real-world applications because many of the problems are expensive or time consuming.
- Both the expense and time are usually a result of the complexity of the function being evaluated.
- In some cases it could take hours if not days of computer time to yield one function evaluation. In these cases, it is not unusual that a scientist or an engineer will decide to terminate an algorithm based simply on how much computer time they are willing to use.

Some options

- There are numerous possibilities for convergence criteria, each with pros and cons.
- The most obvious would be to check to see how close we are to our desired solution, but in general this would be impossible since we don't know what the solution is ahead of time (except for academic exercises).
- On the other hand, it is not unreasonable to assume that we might have some sort of bound, for example in some cases, the solution might be known to be positive or have a minimum/maximum value.

First option

- Suppose that a general iterative algorithm has produced a sequence of iterates starting with an initial guess:

$$x_0, x_1, x_2, \dots, x_k.$$

- One logical approach is to take a look at the magnitude of $f(x_k)$ and check to see how close we are to zero:

$$|f(x_k)| < atol$$

Second option

- Another frequent approach is to stop an iteration when “sufficient” progress has been made. In other words, an engineer is simply interested in reducing the initial function value by some fraction, i.e.

$$\frac{|f(x_k)|}{|f(x_0)|} < ftol$$

Other approaches

- In general, these are good approaches, but there are cases for which progress towards the solution may be slow and little is to be gained from each new iteration.
- In this case, we may decide to stop if we believe we are not making sufficient progress towards a solution.
- This could be construed, for example, if the difference between successive iterates (step size) becomes small:

$$|s_k| = |x_{k+1} - x_k| < stol$$

Relative error

- Here, we should note that it might also make sense to check that the relative step size is small:

$$\frac{|x_{k+1} - x_k|}{|x_{k+1}|} < rtol$$

- In practice, many algorithms will employ some combination of these (and sometimes others).
- Experience and knowledge of the specific problem are usually needed to ensure that we don't stop too early or waste time iterating for too long.

Convergence of Bisection method

- One of the first questions one should ask about any iterative method is when and under what conditions do we expect that it might converge to a solution.
- In the case of the bisection method, it turns out that the only condition we need to have is that the function $f(x)$ is continuous, given that the two initial points yield functions values with opposite signs, i.e. IVT.
- Moreover, if these conditions hold, we can prove that the error bound between the solution and the iterates x_k can be given by:

$$|x^* - x_k| \leq \frac{b-a}{2^k}, \quad k = 0, 1, \dots \quad (2)$$

Proof

Notice that for $k = 0$ all we're really saying is that the root must lie within the given interval $[a_0, b_0]$. At the next step, the interval and hence the error is cut in half so that

$$|x^* - x_1| \leq \frac{b_0 - a_0}{2}, \quad (k = 1)$$

At each iteration, the interval is cut in half by construction, so that at the k th iteration we get our desired result.

As constructed then, ***the bisection method cannot fail***. This type of method is known as a ***robust*** algorithm.

Error Bounds

Additionally, we can use the error bound to estimate the number of iterations required to achieve a certain accuracy, ϵ .

Using Equation 2 we want

$$|x^* - x_k| \leq \frac{b - a}{2^k} \leq \epsilon.$$

Taking logs of both side we have

$$\log(b - a) - \log 2^k \leq \log \epsilon,$$

or rearranging

$$\log(b - a) - \log \epsilon \leq k \log 2.$$

If we would like to have the error be less than ϵ then solving for the number of iterations gives us

$$k \geq \frac{\log[(b - a)/\epsilon]}{\log 2}. \quad (3)$$

Example

Let's set $\epsilon = 0.001$ and $a = 1, b = 2$.

- Then solving for k , we have

$$k \geq \frac{\log[1/0.001]}{\log 2} = \frac{3}{0.301} \approx 9.97,$$

- so $k = 10$ iterations should suffice to achieve an error tolerance of $\epsilon = 0.001$.

Summary

- Bisection is a robust algorithm for finding the zero of a nonlinear function.
- Need to have 2 initial points where function value is of opposite sign.
- It is guaranteed to converge.

Table 1: **Bisection Method Summary**

Advantages	Disadvantages
Always converges, i.e. <i>robust</i> algorithm	Need to provide a specific interval, with 2 points where function is of opposite sign
Error bound easily derived and can be used to estimate number of iterations need to achieve a desired tolerance	Slow convergence - error only decreases by $1/2$ at each iteration
Can be used to start other methods	Not clear how to generalize to higher dimensions

Nonlinear Equations: Newton's Method

Math 131: Numerical Analysis

J.C. Meza

2/8/24

Newton's Method

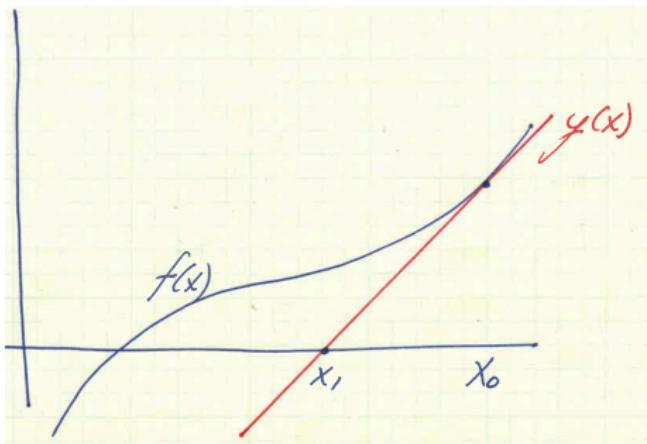
- In the last lecture we saw that the bisection method was robust and would always converge to a solution given the right set of initial values.
- However, it could exhibit slow convergence.
- As a result, most solvers use other types of root-finding algorithms.
- In this section we will study two such methods that can provide much faster convergence to a root.

Newton's Method (Quick Summary)

- Newton's method is likely the most popular and certainly the most powerful method for solving nonlinear equations [@meza2011newton].
- The idea behind Newton's method is to use the slope of the function at the current iterate to compute a new iterate.
- Naturally, this requires that we first assume that the given function $f(x)$ is differentiable.

Visually

- Note that if we take the derivative at the current iterate and use that to set up a linear equation, which we can solve for the new iterate.



Idea

Important

One approach for deriving Newton's method is to think about building a ***linear model of the function*** at the current iterate. Let's consider the linear model $m(x)$:

$$m(x) = f(x_0) + f'(x_0)(x - x_0). \quad (1)$$

Notice that at $x = x_0$ the model agrees with the function $f(x)$, in other words $m(x_0) = f(x_0)$. The idea is to then solve for the root of Equation 1 and use the root as the next guess of our iterative method:

Solving for new iterate

Using this idea let's solve for the root x^* of the linear model, $m(x)$, i.e.

$$\begin{aligned} m(x) &= f'(x_0)(x - x_0) + f(x_0) = 0, \\ \implies x^* &= x_0 - \frac{f(x_0)}{f'(x_0)}. \end{aligned}$$

We can then set $x_1 = x^*$ as the next iterate in our sequence and repeat the process. This gives us the general procedure for Newton's method:

Newton's Method

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

Another derivation

- We will note in passing that another derivation is to use Taylor's theorem to approximate our function $f(x)$ out to the first degree with a remainder term that includes the second derivative.
- We will ignore the second derivative term based on the argument that when we are near the solution the term would be small.
- Solving for our new iterate, we can derive the same equation as before.

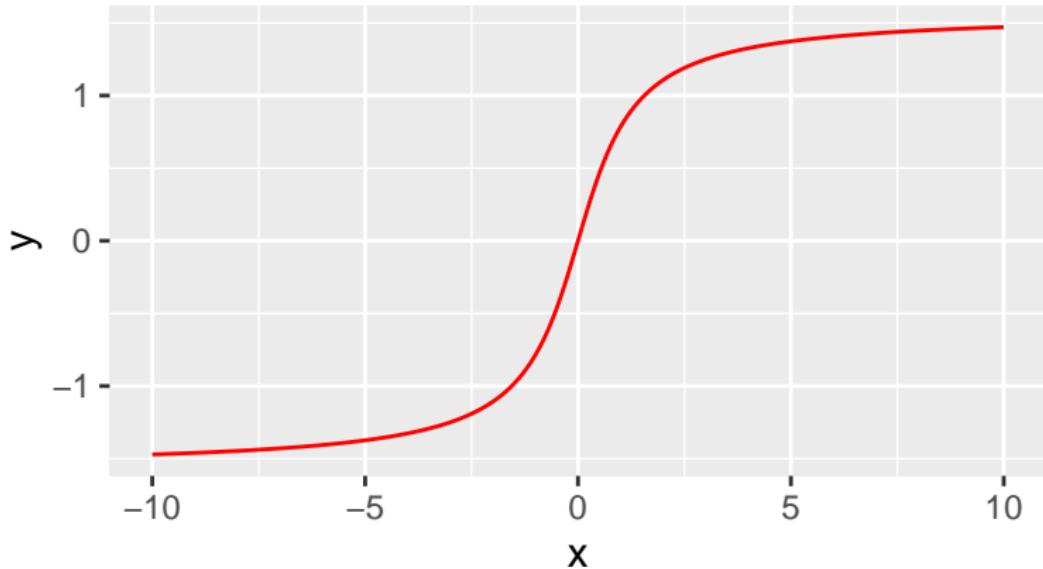
Remark

A natural question to ask is under what conditions does Newton's method converge?

- In fact, it isn't hard to show that if the initial point x_0 is not chosen properly (i.e. close enough to a root), Newton's method will diverge.
- A typical example would be $y = \arctan(x)$, where if x_0 isn't close enough to the root the iterates quickly diverge to infinity.

Arctan(x)

$$y = \arctan(x)$$



Example

Let $f(x) = x^6 - x - 1 = 0$ and let $x_0 = 1.5$. It is easy to verify that one root is given by $x^* = 1.134724$.

To use Newton's method we first need to calculate the derivative -
 $f'(x) = 6x^5 - 1$.

Using Equation 2 allows us to compute the $k + 1$ iteration:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$
$$x_{k+1} = x_k - \frac{x_k^6 - x_k - 1}{6x_k^5 - 1}.$$

Example (cont.)

Proceeding in the natural way from x_0 , we can generate the following sequence of iterates:

k		x_k
0	1.5	1.5
1	$x_1 = x_0 - \frac{x_0^6 - x_0 - 1}{6x_0^5 - 1} = 1.5 - \frac{8.8906}{44.5625}$	1.3005
2	$x_2 = x_1 - \frac{x_1^6 - x_1 - 1}{6x_1^5 - 1} = 1.3005 - \frac{2.5373}{21.3197}$	1.1815
3	$x_3 = x_2 - \frac{x_2^6 - x_2 - 1}{6x_2^5 - 1} = 1.1815 - \frac{0.5387}{12.8140}$	1.1395

Notice that after only 3 iterations, the iterates is already correct to 3 significant digits.

Analysis

Several questions one might consider at this point include:

- Under what conditions might we expect (local) convergence?
- Here by local we mean that the algorithm will converge if we start sufficiently close to a root. We will define this more carefully later.
- If Newton's method converges, how fast can we expect the convergence to be?

Error Analysis for Newton's Method

Let's consider the Taylor expansion about $x = x^*$.

$$0 = f(x_k) + (x^* - x_k)f'(x_k) + \frac{(x^* - x_k)^2}{2}f''(\xi).$$

Dividing by $f'(x_k)$ (we will assume for the time being that it's not equal to zero for any x_k) we get:

$$0 = \frac{f(x_k)}{f'(x_k)} + (x^* - x_k) + \frac{f''(\xi)}{f'(x_k)} \frac{(x^* - x_k)^2}{2}.$$

Using the equation for Newton's method we see that the first term is nothing but $x_k - x_{k+1}$ and substituting into the above equation we get:

$$0 = x_k - x_{k+1} + (x^* - x_k) + \frac{f''(\xi)}{f'(x_k)} \frac{(x^* - x_k)^2}{2}.$$

Error Analysis (cont.)

- We see that the x_k terms cancel out. Rearranging to put the error on the left-hand side of the equation yields:

$$x^* - x_{k+1} = -\frac{f''(\xi)}{2f'(x_k)} (x^* - x_k)^2. \quad (3)$$

- The quantity on the left-hand side of the equation is just the error at the $k + 1$ iteration, while the last term on the right-hand side is the error at the k iteration (squared).

Interpretation

$$\begin{aligned}|e_{k+1}| &= |x^* - x_{k+1}| = \left| \frac{f''(\xi)}{2f'(x_k)} \right| \cdot (x^* - x_k)^2, \\ &= \left| \frac{f''(\xi)}{2f'(x_k)} \right| \cdot |e_{k+1}|^2,\end{aligned}\tag{4}$$

- We can interpret the equation to mean that the error at the $k + 1$ iteration is proportional to the square of the error at the k iteration.

Important

This type of error bound is called ***quadratic convergence***

Remark

- If $f \in C^2[a, b]$ and $f'(x^*) = 0$, then Newton's method still converges but just not as rapidly.
- Consider for example $f(x) = x^4$, which has a root at $x = 0$, but where the first derivative is also equal to 0.

Summary for Newton's Method

Table 2: **Newton's Method Summary**

Advantages	Disadvantages
Doesn't require interval with function sign change	Need to have derivatives
Fast convergence rate – quadratic	May not converge from all starting points
Can generalize to higher dimension	Can be expensive (especially in higher dimensions)

Nonlinear Equations: Secant Method

Math 131: Numerical Analysis

J.C. Meza

2/13/24

Secant Method (Quick Summary)

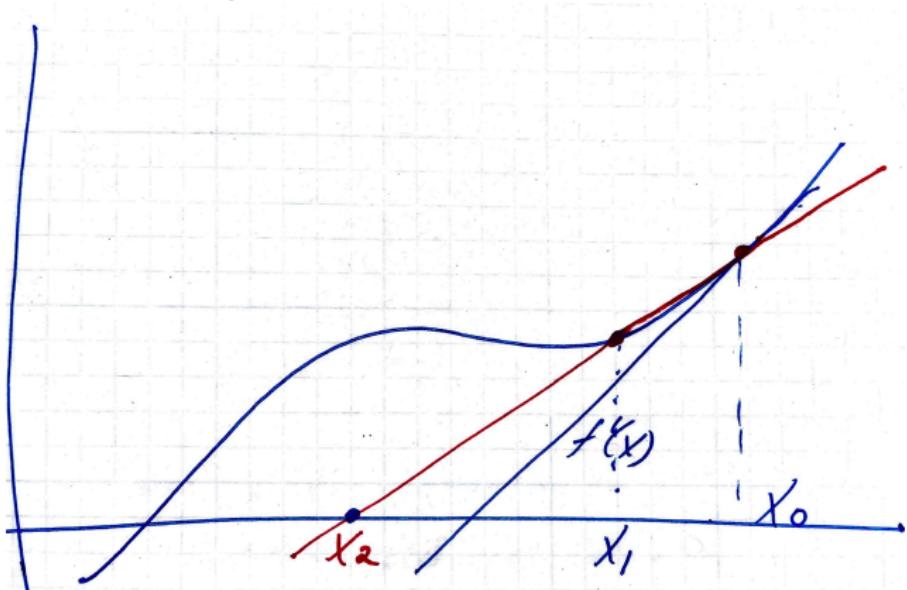
- Recall that Newton's method uses the derivative of the function whose roots we seek. That is both its power and its main disadvantage.
- In many real-world problems, the derivative may be difficult to compute. In other cases, it could be expensive. And in the worst case, it may not even be available.
- The secant method tries to address this disadvantage through an approximation to the derivative $f'(x)$ that uses two points close to each other, i.e. the secant. Using the secant, a new iterate is computed in a fashion similar to Newton's method.

Historical Note

- The secant method is one of the oldest methods for solving nonlinear equations
- Has an interesting history that can be traced back to the Rule of Double False Position described in the 18th-century BCE Egyptian Rhind Papyrus[@papakonstantinou2009].

Visually

Consider a line through two points $(x_0, f(x_0))$ and $(x_1, f(x_1))$. Let x_2 be the x intercept of this line.



Mathematically

Then it follows that

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(x_1) - f(x_2)}{x_1 - x_2}$$

But notice that $f(x_2) = 0$, which leads to

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(x_1) - 0}{x_1 - x_2}$$

General Form

Rearranging and solving for x_2 yields

$$x_2 = x_1 - \left[\frac{(x_1 - x_0)}{f(x_1) - f(x_0)} \right] f(x_1)$$

which is used as the next guess in our sequence.

This then yields the form for the general **secant method**:

Secant Method

$$x_{k+1} = x_k - \left[\frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \right] f(x_k), \quad k = 0, 1, \dots \quad (1)$$

Remark

- Another way to view this is to note that the term in the brackets

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

approximates the derivative of a function (or rather in this case, the inverse).

- Therefore, one could interpret the secant method as just Newton's method with a finite difference approximation to the derivative.

Summary for Secant Method

Table 1: **Secant Method Summary**

Advantages	Disadvantages
Do not need to have derivatives	Need to provide 2 initial points.
Can have fast convergence (although not quadratic)	May not converge from all starting points
Generalizes to higher dimensions	Can be expensive in higher dimensions

Regula Falsi

- Given that both bisection and secant method require two points, it may not be surprising to learn that the two methods can be combined into a new method
- For example, where the updated points in the secant method are chosen in a manner similar to bisection.
- This method goes by several names including the ***method of false position*** and ***regula falsi***.

Root Finding in Higher Dimensions

- Finding roots of nonlinear functions in dimensions higher than one has a long and rich history.
- Of the methods that we have discussed: 1) bisection, 2) Newton's, and 3) Secant, only Newton's method has an obvious path forward.
- This section gives a brief overview on how one proceeds in the case of Newton's method, and also provides a more general iterative procedure that is used in many applications.

Higher Dimensions (cont.)

Recall that Newton's method is based on approximating the next iterate in the sequence of approximations by using the following equation:

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

First, let's rewrite the equation as follows:

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

Higher Dimensions (cont.)

- Consider $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$, where $n > 1$.
- We can still take the derivative of this function, following all the usual rules.
- In this case, it results in a matrix, which is called the **Jacobian** and is given by:

$$J(x_k) = F'(x_k) = \left[\frac{\partial f_i(x_k)}{\partial x_j} \right] \quad i, j = 1, \dots, n.$$

Example

Let $F(x) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $x = (x_1, x_2)$

Consider

$$F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} e^{x_1} - x_2 \\ x_1^2 - 2x_2 \end{bmatrix}$$

and

$$F'(x) = \begin{bmatrix} e^{x_1} & -1 \\ 2x_1 & -2 \end{bmatrix},$$

where $F'(x)$ is a 2×2 matrix.

Newton's Method in higher dimensions

Newton's method can then be written as:

$$x_{k+1} = x_k - J(x_k)^{-1} F(x_k), \quad k = 0, 1, \dots$$

where the inverse is interpreted as *matrix inversion*.

or

$$J(x_k)(x_{k+1} - x_k) = -F(x_k), \quad k = 0, 1, \dots$$

Remark

- It is a fundamental precept in numerical analysis that one rarely computes the inverse of a matrix.
- As such, the usual method for stating Newton's method in higher dimensions is as follows:

At each iteration k compute the step $s_k = (x_{k+1} - x_k)$ by solving the linear equation:

$$J(x_k)s_k = -F(x_k).$$

The new iterate is computed by:

$$x_{k+1} = x_k + s_k$$

Nonlinear Equations: Fixed Point

Math 131: Numerical Analysis

J.C. Meza

2/15/24

Fixed Point

Methods for root finding are hard to analyze, especially if we include derivatives of f . Another approach is to reformulate the problem to allow for easier analysis using what is known as a fixed point formulation. This approach has a long history and is also known as functional iteration, Picard iteration, and successive substitution.

Note:

We will do everything in \mathbb{R}^1 but the ideas hold in higher dimensions with appropriate generalizations.

First let's start with a definition.

Fixed Points

Definition: The point $x^* \in \mathbb{R}$ is said to be a **fixed point** of $g : \mathbb{R} \rightarrow \mathbb{R}$ if $g(x^*) = x^*$.

Using the definition it is easy to see that finding fixed points is equivalent to finding roots of an equation.

For example, let's define a function $g(x)$ such that

$$g(x) = x - f(x)$$

Then x^* is a zero of $f(x)$ if and only if x^* is a fixed point of $g(x)$, i.e.

$$f(x^*) = 0 \iff g(x^*) = x^*$$

Equivalence of zeros and fixed points

By definition:

$$g(x) = x - f(x).$$

If x^* is a zero of $f(x)$ then clearly $g(x^*) = x^*$.

On the other hand, if $g(x^*) = x^*$ then

$$f(x^*) = x^* - g(x^*) = x^* - x^* = 0$$

Note

Once we know how to find fixed points, we can find zeros of a function.

Equivalence of Fixed Points and Zeros

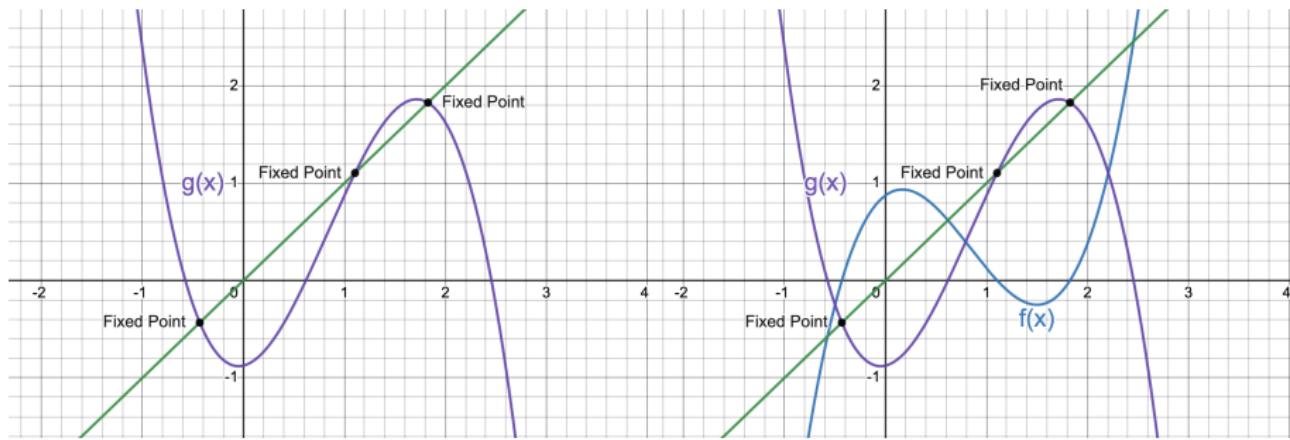


Figure 1: Fixed Point is where $g(x)$ meets $y=x$. Zeros align with fixed points

Idea

- The general idea is that if we have an approximation, x_0 , to the fixed point x^* , then if we take $x_1 = g(x_0)$, our hope is that x_1 will be an even better approximation.
- There are many choices we can make for $g(x)$ for any given $f(x)$. And not too surprisingly, the choice of $g(x)$ will be extremely important in determining whether we produce a good algorithm or not.
- In the rest of the section, we will look into some of the details of this approach and what conditions are needed to ensure that the iteration converges.

Algorithm

```
# Example: Fixed point iteration
maxiter = 50
x = x0
for k in range(1,maxiter):
    x = g(x)
    if (abs(g(x)-x) < 1.e-15):
        break
xsol = x
```

Example

Let's suppose that the $\sqrt{}$ key on your calculator is broken and you need to compute $\sqrt{3}$. This is equivalent to finding a solution of the function $f(x) = x^2 - 3 = 0$.

First attempt:

$$g(x) = 3/x, x_0 = 1$$

Using the algorithm above we get the following sequence of iterates:

$$x_0 = 1$$

$$x_1 = 3/x_0 = 3$$

$$x_2 = 3/x_1 = 1$$

$$x_3 = 3/x_2 = 3$$

$$\vdots$$

Apparently, this is not a good choice!

Second attempt:

$$g(x) = \frac{1}{2}(x + 3/x), x_0 = 1$$

This time the sequence of iterates is:

$$x_0 = 1$$

$$x_1 = \frac{1}{2}(x_0 + 3/x_0) = 2$$

$$x_2 = \frac{1}{2}(x_1 + 3/x_1) = 1.75$$

$$x_3 = \frac{1}{2}(x_2 + 3/x_2) = 1.73214$$

$$x_4 = \frac{1}{2}(x_3 + 3/x_3) = 1.7320508$$

Note that after only 4 iterations, our solution appears to be a great approximation to $\sqrt{3}$. Later on, we'll find out why this choice of $g(x)$ is a good choice.

In class exercise:

Compute $\sqrt[3]{5}$ using the same procedure as above with
 $g(x) = \frac{1}{3}(2x + 5/x^2)$, $x_0 = 1$.

$$x_0 = 1$$

$$x_1 = \frac{1}{3}(2x_0 + 5/x_0^2) =$$

$$x_2 =$$

$$x_3 =$$

$$x_4 =$$

⋮

Existence and Uniqueness of Fixed Points

At this point, there are several questions that arise naturally:

- ① Is there a fixed point x^* in the interval $[a, b]$?
- ② If yes, is it unique?
- ③ Does a given fixed point iteration generate a sequence of iterates $\{x_k\} \rightarrow x^*$?
- ④ And if yes, how fast will the iterates converge to the fixed point?

We'll take each of these points in turn starting with (1) and (2) using the following theorem.

Theorem: Fixed Point Existence/Uniqueness

If $g \in C[a, b]$ and $g(x) \in [a, b] \quad \forall x \in [a, b]$, then there exists a fixed point $x^* \in [a, b]$.

If in addition, $g'(x)$ **exists** on (a, b) and there exists a positive constant $k < 1$ with $|g'(x)| \leq k \quad \forall x \in (a, b)$ then there exists **exactly 1** fixed point in $[a, b]$.

The first part of the theorem states the condition for the existence of a fixed point. The second part states the conditions necessary for uniqueness. Let's take them one at a time.

Existence

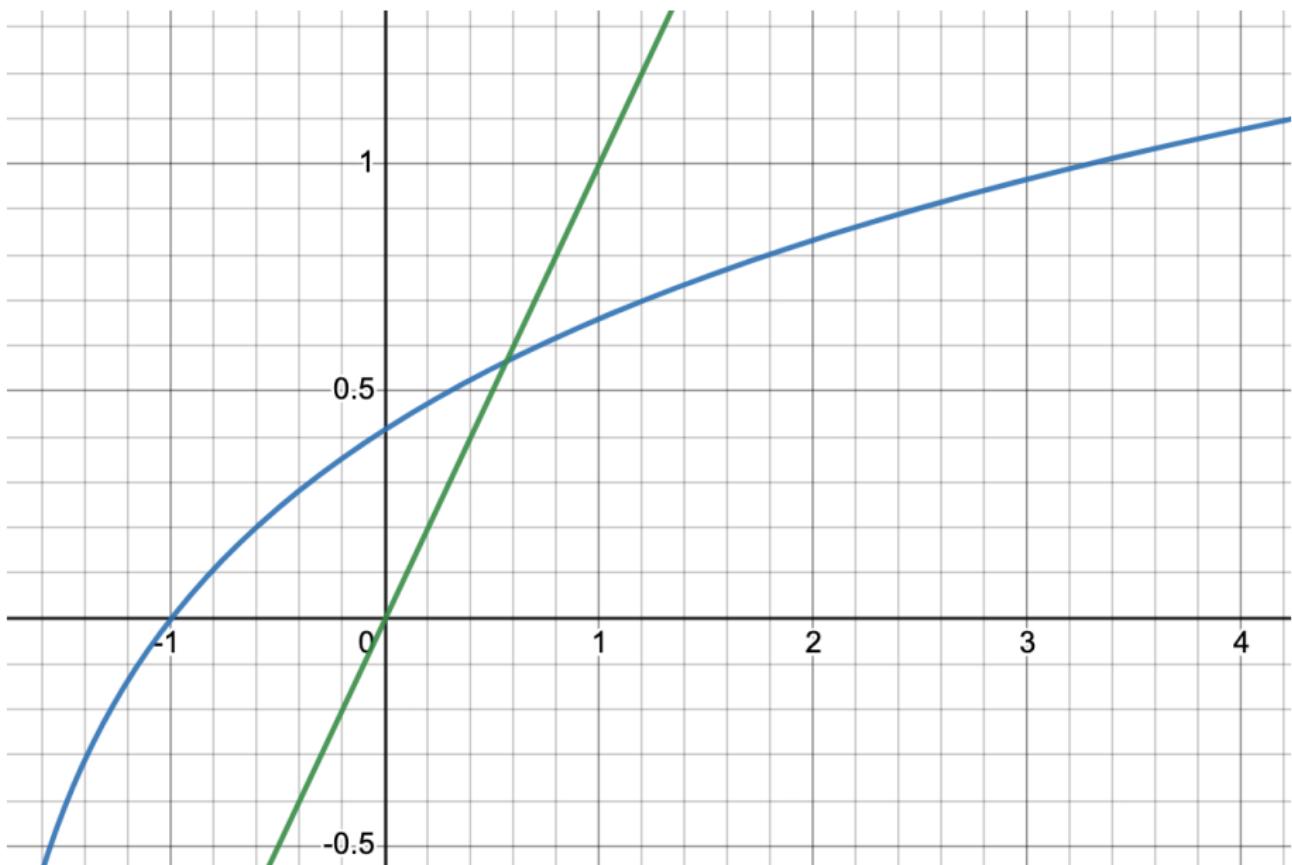
There are two cases to consider. The first is if the fixed point is one of the endpoints of the interval $[a, b]$. The second case, is if the fixed point is in the interior.

If $g(a) = a$ or $g(b) = b$, then clearly the fixed point $x^* = a$ or b , so we're done.

If not, then if the fixed point exists it must lie in the interior, i.e. $g(a) \neq a$ and $g(b) \neq b$. By assumption $g(x)$ maps the interval $[a, b]$ onto itself, so we can deduce that

$$g(a) > a \quad \text{and} \quad g(b) < b.$$

Visually



Existence Proof (cont.)

Let's now consider the function

$$h(x) = g(x) - x$$

First note that $h(x)$ is continuous on $[a, b]$ since $g(x)$ is continuous. Next, it is easy to see that it also satisfies the conditions:

$$h(a) = g(a) - a > 0 \quad \text{and} \quad h(b) = g(b) - b < 0$$

By the Intermediate Value Theorem, there exists a point $x^* \in (a, b)$ such that $h(x^*) = 0$. But using the definition of $h(x)$ that means that

$$\begin{aligned} h(x^*) &= 0 = g(x^*) - x^* \\ \implies g(x^*) &= x^* \end{aligned}$$

So x^* must be a fixed point in $[a, b]$.

Uniqueness

To show that a unique fixed point exists we will have need of the second condition

$$|g'(x)| \leq k < 1$$

The proof will be by contradiction. Let's assume that we have two fixed points x^*, y^* and that $x^* \neq y^*$. Using the Mean Value Theorem we can say that:

$$\frac{g(x^*) - g(y^*)}{x^* - y^*} = g'(\xi) \quad \xi \in [x^*, y^*] \subset [a, b].$$

Uniqueness Proof (cont.)

Let's now consider $|x^* - y^*|$:

$$|x^* - y^*| = |g(x^*) - g(y^*)| = |g'(\xi)| \cdot |x^* - y^*| < k \cdot |x^* - y^*|$$

The first equation is true by the definition of a fixed point, and the second from the equation above derived from the MVT. The final inequality is due to the assumption on the bound of the derivative of g . But since the constant $k < 0$, this reduces to:

$$|x^* - y^*| < |x^* - y^*|$$

which is a contradiction.

Therefore, x^* must be unique. ■

More on convergence of sequences

- The question of convergence is not simply a matter of deciding when one should stop an iterative method.
- When one has a choice of different algorithms to pick from, it would make sense to choose the one that is fastest, where fastest can be loosely defined to be the one that is likely to take the fewest number of iterations.
- Towards that end, let's discuss convergence in a bit more detail.

Definition: Rate of Convergence

We say that the **rate of convergence** of x_k to x^* is of **order** $p \geq 1$ if

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} < \infty$$

Remark

If $p = 2$, we say the rate of convergence is **quadratic**. All other things being equal, the larger the value of p , the faster an algorithm will converge to a solution.

Rate of convergence (cont.)

For the special case of $p = 1$ we ask instead that

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} < 1$$

This is known as a **linear** rate of convergence.

In addition, if for some sequence $\{c_k\} \rightarrow 0$ we have

$$|x_{k+1} - x^*| \leq c_k |x_k - x^*|$$

then the sequence $\{x_k\}$ is said to converge **superlinearly** to x^* .

Summary

A quadratic rate of convergence is better than a superlinear rate of convergence, which is better than linear.

Examples.

① $x_k = a^k, \quad 0 < a < 1$

② $x_k = a^{2^k}, \quad 0 < a < 1$

③ $x_k = \left(\frac{1}{k}\right)^k$

Solutions:

(1) linear, (2) quadratic, (3) superlinear.

Nonlinear Equations: Fixed Point Iteration Convergence

Math 131: Numerical Analysis

J.C. Meza

2/20/24

Last lecture

- ① Is there a fixed point x^* in the interval $[a, b]$?
- ② If yes, is it unique?
- ③ Does a given fixed point iteration generate a sequence of iterates $\{x_k\} \rightarrow x^*$?
- ④ And if yes, how fast will the iterates converge to the fixed point?

Fixed Point Iteration Convergence (Part 1)

Now that we've answered questions (1) and (2) on the existence and uniqueness of fixed points, we will turn our attention to the last two questions dealing with the convergence of the fixed point iteration itself.

- ③ Does a given fixed point iteration generate a sequence of iterates $\{x_k\} \rightarrow x^*$?
- ④ And if yes, how fast will the iterates converge to the fixed point?

Recall: Fixed Point Iteration

```
# Example: Fixed point iteration
maxiter = 50
x = x0
for k in range(1,maxiter):
    x = g(x)
    if (abs(g(x)-x) < 1.e-15):
        break
xsol = x
```

In order to better understand the possible cases, let's take a look at a couple of pictures.

Visually - Convergence

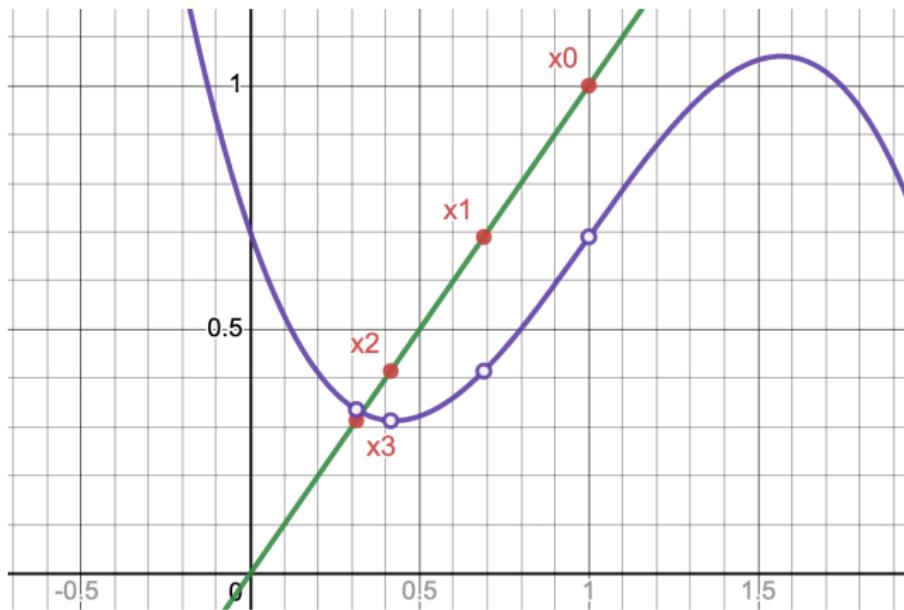


Figure 1: Fixed Point Iteration Converges

Visually - Divergence

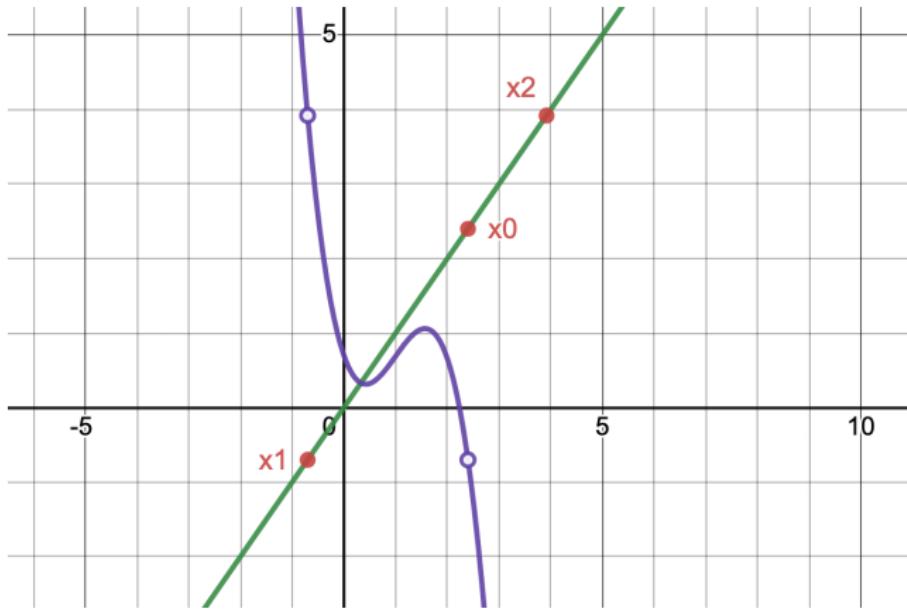


Figure 2: Fixed Point Iteration can diverge

Discussion

- If the slope of the function $g(x)$ is too large, then the fixed point iteration might not converge.
- Another way to think about this is that the derivative must be bounded in some way for the fixed point iteration to converge.
- Your textbook proves convergence by making an assumption on the derivative of $g(x)$ and specifically that there is a constant $k < 1$ such that:

$$|g'(x)| \leq k \quad \forall x \in [a, b]$$

We will use two other concepts that will prove useful in later lectures. It also has the additional benefit of not requiring the assumption that the derivative of $g(x)$ exists.

Lipschitz Continuity

Definition. A function $g : \mathbb{R} \rightarrow \mathbb{R}$ is said to be **Lipschitz continuous** if

$$|g(x) - g(y)| \leq L |x - y| \quad \forall x \in [a, b].$$

L is called the **Lipschitz constant**.

Important

A function that is Lipschitz continuous is bounded in how much it can change. However, the definition only requires the function be bounded **on an interval** and not over the entire domain of the function.

Relation to derivative

As it turns out there is a close relation of the Lipschitz constant to the derivative. Recall that by the MVT

$$g(x) - g(y) = g'(\xi)(x - y) \quad \xi \in (x, y) \subset [a, b].$$

Taking absolute values of both sides we get:

$$|g(x) - g(y)| = |g'(\xi)| \cdot |x - y| \quad \xi \in (x, y) \subset [a, b].$$

Proof (cont.)

Now, if we let

$$L = \max_{\xi \in [a,b]} |g'(\xi)|,$$

it follows that:

$$|g(x) - g(y)| \leq L \cdot |x - y| \quad x \in [a, b].$$

Intuitively this makes sense as bounding the derivative is similar to bounding the change in the function values.

Contraction Mapping

Definition. We say that $g : \mathbb{R} \rightarrow \mathbb{R}$ is a **contraction mapping** if

$$|g(x) - g(y)| \leq L |x - y| \quad \forall x, y \in [a, b].$$

with $L < 1$.

Note

Notice the close similarity to Lipschitz continuity, but with the important distinction that $L < 1$.

Example: Contraction Mapping

Consider

$$\begin{aligned}f(x) &= e^x - 2x - 1 \quad \text{on } [1, 2] \\g(x) &= \ln(2x + 1)\end{aligned}$$

Show that $g(x)$ is a contraction mapping. Note: $x^* \approx 1.256$

Solution Outline

- ① Compute g'
- ② Note that g' is monotonically decreasing on given interval
- ③ Show bound on g' , and in particular that the bound is < 1 .

We're now ready to show our first convergence theorem for fixed point iterations.

Fixed Point Convergence Theorem

Theorem. Suppose $g : \mathbb{R} \rightarrow \mathbb{R}$, with $g(x) \in [a, b] \quad \forall x \in [a, b]$ be a continuous contraction mapping. Then there exists a unique fixed point x^* with $g(x^*) = x^*$ and the fixed point iteration converges to x^* for any starting point $x_0 \in [a, b]$.

Proof:

Existence and uniqueness of a fixed point was proven earlier.

To prove convergence of the fixed point iteration we need to show that $\{x_k\} \rightarrow x^*$ for the fixed point iteration:

$$x_{k+1} = g(x_k)$$

Proof (cont.)

Let's consider: $x_k - x^* = g(x_{k-1}) - g(x^*)$

Taking absolute values and using the fact that g is a contraction mapping we can write

$$|x_k - x^*| = |g(x_{k-1}) - g(x^*)| \leq L |x_{k-1} - x^*|.$$

Now we apply this inductively:

$$\begin{aligned}|x_k - x^*| &\leq L^2 |x_{k-2} - x^*|. \\&\leq \dots \\&\leq L^k |x_0 - x^*|\end{aligned}$$

Proof (cont.)

But since $L < 1$ we know that $L^k \rightarrow 0, k \rightarrow \infty$ and therefore

$$\|x_k - x^*\| \rightarrow 0, k \rightarrow \infty$$

as we set out to show. ■

Remarks

- As it turns out, while a contraction mapping is a useful tool, it is often hard to verify in a real-world application.
- Instead what is usually assumed is that either the function is Lipschitz continuous in a neighborhood of the fixed point x^* , or that the function has a bounded derivative at the root.
- Another thing to keep in mind is that it might be unrealistic to assume a contraction mapping property for all $x \in [a, b]$. An alternative is to assume that we have some property that holds at or near the solution. In this case, we have a different type of convergence.

Local Convergence

Definition. The iterative process $x_{k+1} = g(x_k)$ is said to be **locally convergent** to the fixed point x^* if there exists $\delta > 0$ such that $x_k \rightarrow x^*$ for any x_0 satisfying $|x^* - x_0| \leq \delta$.

Tip

The simplest interpretation of this definition is to say, if we start **close enough** to a fixed point, then the method will converge to it.

What defines close enough, will depend not only on the initial point, but also on the function we use.

With this background, we can state the following theorem.

Local Convergence Theorem

Theorem. Suppose $g : \mathbb{R} \rightarrow \mathbb{R}$ is continuously differentiable and:

- ① $x^* = g(x^*)$
- ② $|g'(x^*)| < 1$

Then the iterative process $x_{k+1} = g(x_k)$ is **locally convergent** to x^* .

Remark: Note the difference here that we are only asking for a bound on g' at a single point, namely the fixed point, x^* .

The proof follows much the same as before, but now we have to also show that $|g'(x)| < 1$ for all x in a neighborhood of x^* . Since we are assuming that g is continuously differentiable, this is fairly easy to show.

Fixed Point Iteration Convergence (Part 2)

Theorem. Suppose $g : \mathbb{R} \rightarrow \mathbb{R}$ is twice continuously differentiable and the iterative process $x_{k+1} = g(x_k)$ is locally convergent.

Then the convergence rate is **linear** if

$$g'(x^*) \neq 0$$

and the convergence rate is at least **quadratic** if

$$g'(x^*) = 0.$$

Proof - Linear case

We've already shown that the fixed point iteration converges.

To show that we have linear convergence, we can either

- ① show that we can bound the derivative in a neighborhood of the fixed point or
- ② we can use the Lipschitz continuity condition, to show that the error at any iteration is bounded by $L < 1$, which gives us a linear rate of convergence.

Proof - Quadratic case

To show that the convergence rate is at least quadratic, let's first expand $g(x)$ in a Taylor polynomial about the point x^* .

$$g(x) = g(x^*) + g'(x^*)(x - x^*) + \frac{g''(\xi)}{2}(x - x^*)^2. \quad (1)$$

By assumption we know that:

$$g(x^*) = x^* \text{ and } g'(x^*) = 0.$$

Proof (cont.)

Substituting into (Equation 1), we get:

$$g(x) = x^* + 0 + \frac{g''(\xi)}{2}(x - x^*)^2.$$

Consider $x = x_k$:

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

with ξ_k between x_k and x^* .

Proof (cont.)

Using the definition of the fixed point iteration: $x_{k+1} = g(x_k)$, we have

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

Rearranging we have:

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

Proof (cont.)

Since the fixed point iteration is locally convergent this means that

$$\{x_k\} \rightarrow x^* \implies \{\xi_k\} \rightarrow x^*$$

since ξ_k is between x_k and x^* .

Therefore

$$\frac{x_{k+1} - x^*}{(x_k - x^*)^2} = \frac{g''(x^*)}{2} = C < \infty,$$

which proves that the iterates are converging q-quadratically.

Designing good $g(x)$ functions

Important

This theorem shows us that if we want quadratic convergence, we should look for fixed point methods with the property that $g'(x^*) = 0$.

Let's consider the general form

$$g(x) = x - \phi(x)f(x) \quad (2)$$

where $\phi(x)$ is differentiable and to be chosen later.

Taking derivatives we have:

$$g'(x) = 1 - \phi'(x)f(x) - f'(x)\phi(x).$$

Letting $x = x^*$, such that $f(x^*) = 0$, we get:

$$g'(x^*) = 1 - f'(x^*)\phi(x^*).$$

Conditions for Quadratic Convergence

By our theorem, if we want quadratic convergence we need to have $g'(x^*) = 0$, hence

$$0 = 1 - f'(x^*)\phi(x^*),$$

and solving for ϕ we get the result:

$$\phi(x^*) = \frac{1}{f'(x^*)}.$$

Substituting back into our general form (Equation 2) we have:

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

But that's just Newton's method!

As a result, an immediate corollary is that Newton's method is quadratically convergent.

Example

Recall our earlier example where we wanted to find the root of $x^2 - 3$, that led us to try different fixed point iterations.

① $g(x) = 3/x, \Rightarrow g'(x) = -3/x^2.$

$$g'(x^*) = g'(\sqrt{3}) = -1 \quad \Rightarrow \quad \text{No convergence}$$

② $g(x) = \frac{1}{2}(x + 3/x), \Rightarrow g'(x) = \frac{1}{2}(1 - 3/x^2).$

$$g'(x^*) = g'(\sqrt{3}) = 0 \quad \Rightarrow \quad \text{q-quadratic convergence}$$

③ $g(x) = x - \frac{x^2 - 3}{7}, \Rightarrow g'(x) = 1 - \frac{1}{7}(2x).$

$$g'(x^*) = g'(\sqrt{3}) = 0.505 \quad \Rightarrow \quad \text{Linear convergence}$$

Estimating number of iterations

- Sometimes it is useful to estimate the number of iterations required to achieve a certain reduction in the error.
- We were able to do this in the case of the bisection method as each iteration reduced the interval in half, so therefore it was easy to compute the number of iterations needed to reduce the error by a certain factor.
- We can do something similar for other methods, although it can be a bit trickier.

Convergence (cont.)

Suppose that we know that we have q-linear convergence with a constant λ , i.e.

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

One way to approach this is to consider our proof of convergence for the fixed point iteration. We were able to show that

$$|x_k - x^*| \leq L^k |x_0 - x^*|.$$

Iterations to Convergence

Suppose we would like to reduce the error by a factor of 10 after k iterations. Setting $\lambda = L$, this means that we want to have:

$$\lambda^k = 10^{-1}$$

Taking logs of both sides we have:

$$k \log_{10} \lambda = -1$$

or equivalently

$$k = -\frac{1}{\log_{10} \lambda}$$

The value of k can be interpreted to mean that it approximates the number of iterations required to reduce the error by a factor of 10.

Example

Suppose that we knew that a certain fixed point iteration g had a Lipschitz constant of $L = 2/3$. Then setting $\lambda = L$.

$$k = -\frac{1}{\log_{10}(2/3)} \approx \frac{1}{0.176} = 5.68$$

which we can interpret to mean that it would take approximately $k = 6$ iterations to reduce the error by a factor of 10.

Summary Comparison of nonlinear equation methods

Method	Assumptions	Advantages	Disadvantages
Bisection	f is continuous; 2 starting points where function has opposite sign	Robust; easy to implement	Linear convergence
Newton	f is continuously differentiable	Quadratic convergence	Need derivatives
Secant	f is continuous	No derivatives; superlinear convergence	Cancellation error possible
Fixed Point	Need to have a good $g(x)$	Easy to implement	Linear convergence

Polynomial Interpolation

Math 131: Numerical Analysis

J.C. Meza

2/22/24

Section 1

Introduction

Interpolation and Extrapolation

- Interpolation is usually not an end product in numerical analysis, but it is used in many other techniques that we will study.
- Having a good understanding of the concepts in approximating functions or interpolating some given data will prove useful later.
- Let's consider approximation first. There are two basic problems:
 - ① data fitting, and
 - ② approximating other functions.

Data fitting

For ease, we will only consider the case of \mathbb{R}^1 here, although (as before) many of the ideas translate into higher dimensions. Suppose we are given a set of data points

$$\{(x_i, y_i)\}_{i=0}^n.$$

The points x_i are sometimes called the ***node points*** or simply just ***nodes***.

Goal

Find a function $v(x)$ that “fits” the data in some yet to be determined way.

Interpolation

If the data is believed to be accurate, we could also insist that $v(x)$ match the data exactly, i.e. that

$$v(x_i) = y_i \quad i = 0, 1, \dots n.$$

If this is the case, then we say that $v(x)$ **interpolates** the data.

This still leaves open the question of what we mean by *fit*, and also what might constitute a good function, i.e. what properties do we want in a function that fits the data.

Approximating a function

- The second area we will study is that of approximating another function.
- This situation might arise if we had a complicated or computationally expensive function.

Goal

Find a simpler or cheaper function that we could use to approximate our expensive function. The techniques will be similar to the earlier case, but with the difference that here, we might be able to choose the node points ourselves.

Uses of approximation

- One typical use for ***approximating*** functions widely used in practice is to predict the value of the function at points other than those given (or chosen).
- If the point at which we want to predict the function value is inside the interval set by the data points, then we call it ***interpolation***.
- If the point is chosen outside the interval, then we say it is ***extrapolation***.

Other uses

Another use for approximating functions arises in the context of other numerical methods. The most common example is in helping us to take derivatives or compute integrals of other functions.

General Representation

We will first consider the general case of a **linear form**, in which we will write the approximating (interpolating) function as:

$$v(x) = \sum_{j=0}^n c_j \phi_j(x)$$

Here we call $\{c_j\}_{j=0}^n$ the unknown coefficients, and $\{\phi_j\}_{j=0}^n$ the **basis functions**.

We will further assume that $\{\phi_j\}_{j=0}^n$ are linearly independent.

Linear form

When we say that we will take the **linear form**, we mean that $v(x)$ is written as a linear combination of the basis functions and not that $v(x)$ is a linear function of x .

Types of Basis Functions

Example

- ① Polynomial (monomial): $\phi_j(x) = x^j \quad j = 0, 1, \dots n$
- ② Trigonometric: $\phi_j(x) = \cos(jx) \quad j = 0, 1, \dots n$
- ③ Gaussian functions: $\phi_j(x) = e^{-x^2} \quad j = 0, 1, \dots n$
- ④ Many, many more

There are many other forms that can be chosen, most of which are used for specific applications or problems with a special structure.

For now we will restrict ourselves to polynomial interpolants.

Polynomial Interpolants

Some of the reasons that polynomials are a good first choice include:

- Easy to both construct and evaluate the interpolants.
- Easy to sum and multiply polynomials
- Easy to differentiate and integrate
- And despite their simple appearance, they can fit many different types of data.

Applications of Polynomial Interpolation

- Polynomials of one form or another are ubiquitous in numerical analysis.
- We will see that they can be used to approximate data, they are used to approximate derivatives in other contexts, and even to help us evaluate integrals numerically.
- Some of the uses of polynomials include approximating commonly used functions and in computer graphics to smooth curves and surfaces of geometric objects.

Stages for using interpolants

Before we go to the next section, we should also say that there are two main stages when using (polynomial) interpolation.

- ① ***Constructing an interpolant.*** This usually entails computing the unknown coefficients given a particular set of data points. This can be expensive, but it is done only once for a given data set.
- ② ***Evaluating an interpolant.*** Once the polynomial is constructed, we are then able to evaluate the polynomial at some point or some set of points. This is typically much less expensive per evaluation, but we may need to do it many times.

Recall

We can write a polynomial (monomial) as:

$$\begin{aligned} p(x) &= \sum_{j=0}^n c_j x^j, \\ &= c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n, \end{aligned} \tag{1}$$

for a given set of $n + 1$ data points:

$$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n).$$

Reminder

A polynomial of degree n has $n + 1$ coefficients.

Shifted Power Form

Tip

While the form given by Equation 1 is the most convenient for analysis, it can lead to numerical errors and we will often see instead the *shifted power form*:

$$p(x) = c_0 + c_1(x - a) + c_2(x - a)^2 + \cdots + c_n(x - a)^n,$$

Interpolating Problem

Goal

Given a set of distinct points $\{(x_i, y_i)\}_{i=0}^n$, find the $n + 1$ coefficients c_0, c_1, \dots, c_n , such that we satisfy the interpolating conditions:

$$p(x_i) = y_i, \quad i = 0, 1, \dots, n.$$

Note: It is very important that the data points be distinct, i.e.

$$x_i \neq x_j \quad \forall \quad i \neq j.$$

Example: Constructing an Interpolating Polynomial

Example

Suppose we are given the data $(x_0, y_0) = (1, 1)$, $(x_1, y_1) = (2, 3)$ and we wish to interpolate the two data points using a first degree polynomial $n = 1$. In other words we want to construct the interpolating linear polynomial

$$p_1(x) = c_0 + c_1 x. \quad (2)$$

Solution

The interpolating conditions give us:

$$\begin{aligned} p_1(x_0) &= c_0 + 1 \cdot c_1 = 1 \quad (= y_0), \\ p_1(x_1) &= c_0 + 2 \cdot c_1 = 3 \quad (= y_1). \end{aligned}$$

This is a set of two equations in two unknowns, which you can easily verify has the solution:

$$c_0 = -1 \quad c_1 = 2.$$

(cont.)

These coefficients can then be substituted into our polynomial form given by Equation 2 to yield the desired polynomial interpolant:

$$p_1(x) = 2x - 1.$$

Now that we have constructed the polynomial, we are free to evaluate it at any number of other points.

Remark: An alternate derivation of this form can be found in the supplemental section at the end of this section.

Exercise

Construct interpolating polynomial of degree $n = 2$, using an additional third point given below:

Table 1: Data

	$i = 0$	$i = 1$	$i = 2$
x	1	2	4
y	1	3	3

Solution:

$$c_0 = -7/3 \quad c_1 = 4 \quad c_2 = -2/3$$

$$p_2(x) = \frac{1}{3}(-2x^2 + 12x - 7). \quad (3)$$

Vandermonde Matrices

In solving the exercise problem above you will have noticed that you had to set up a set of equations that looked like:

$$\begin{aligned} p_2(x_0) &= c_0 + c_1x_0 + c_2x_0^2 &= y_0 \\ p_2(x_1) &= c_0 + c_1x_1 + c_2x_1^2 &= y_1 \\ p_2(x_2) &= c_0 + c_1x_2 + c_2x_2^2 &= y_2. \end{aligned}$$

These can be written more concisely in matrix notation as:

$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}. \quad (4)$$

or simply as

$$Xc = y$$

Vandermonde matrices

- This system of linear equations can now be solved for the coefficients c_0, c_1, c_2 , which determine our interpolating polynomial.
- The matrix X in Equation 4 is an example of a **Vandermonde** matrix.
- The general form of the Vandermonde matrix for $n + 1$ data points is given by:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \cdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_n \end{bmatrix}. \quad (5)$$

Vandermonde (cont.)

Given

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \cdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_n \end{bmatrix}.$$

- It can be shown that the $\det X \neq 0$, ***as long as the given data points are distinct.***
- This implies that X is nonsingular and hence the linear system has a unique solution.
- This observation leads to the following theorem.

Polynomial Interpolant Uniqueness

Theorem

For any real data points $\{(x_i, y_i)\}_{i=0}^n$ such that x_i are distinct there exists a unique polynomial $p(x)$ of degree at most n that satisfies the interpolating conditions $p(x_i) = y_1$ for $i = 0, 1, \dots, n$.

Interpretation

There are many ways to construct an interpolating polynomial - in the end they all yield the same polynomial.

Summary - Monomials as basis functions for interpolation.

- Introduced approximation and interpolation - fitting a given set of points with a function, with interpolation being a special case of approximation.
- Considered the general linear form and looked at our first case using polynomials and specifically monomials, i.e. $\phi_j(x) = x^j$.
- Led to the construction of the Vandermonde matrix, which is extremely easy to set up.
- A big disadvantage is that the Vandermonde matrices are notoriously ill-conditioned. For $n = 10$, $\kappa(A) \approx 10^{10}$. As a result, using this approach is highly discouraged. (Notwithstanding this comment, for small n , this approach might be useful.)
- The resulting coefficients don't have a clear relation to the y_i values, which is sometimes desired. This will come up when we study numerical differentiation and integration.

Section 2

Supplemental Materials

Supplemental Materials Weierstrass Approximation

A natural question to ask is if or when a general function $f(x)$ can be approximated by a polynomial. This was answered by Weierstrass (1885) in a theorem bearing his name.

Theorem (Weierstrass Approximation)

Suppose $f(x)$ is defined and continuous on a closed interval $[a, b]$, and $\epsilon > 0$ is given. Then there exists a polynomial $p(x)$ on $[a, b]$ such that

$$|f(x) - p(x)| < \epsilon \quad \forall x \in [a, b].$$

The most common way of stating this is that any continuous function on a closed interval can be approximated with arbitrary precision by a polynomial.

Alternate derivation of linear interpolation using monomial basis functions

Let's begin with some simple cases - in particular the linear case. Suppose we are given two points (x_0, y_0) and (x_1, y_1) , and $x_0 \neq x_1$. Then the straight line passing through these two points is given by the linear polynomial:

$$p_1(x) = \frac{(x_1 - x)y_0 + (x - x_0)y_1}{x_1 - x_0}. \quad (6)$$

Let x_0, x_1, \dots, x_n be $n + 1$ distinct points on some interval $[a, b]$.

(cont.)

We say that $p_1(x)$ **interpolates** the value y_i at the points $x_i, i = 0, 1$ if

$$p_1(x_i) = y_i \quad i = 0, 1.$$

To see that Equation 6 satisfies this property note that:

$$\begin{aligned} p_1(x_1) &= \frac{(x_1 - x_1)y_0 + (x_1 - x_0)y_1}{x_1 - x_0} \\ &= \frac{(0)y_0 + (x_1 - x_0)y_1}{x_1 - x_0} \\ &= \frac{(x_1 - x_0)}{x_1 - x_0}y_1 = y_1 \end{aligned}$$

You can use a similar argument to show that $p_1(x_0) = y_0$.

Section 3

Lagrange Polynomials

Lagrange Polynomials

- As a result of the advantages and disadvantages listed above, other approaches have been developed that attempt to address the disadvantages.
- One such approach for constructing an interpolating polynomial consists of using what are known as the Lagrange polynomials for the basis functions.

Construction

Proceeding as before, let's try to construct a polynomial using the general linear form using the following equation:

$$\begin{aligned} p(x) = p_n(x) &= \sum_{j=0}^n y_j L_j(x), \\ &= y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x) + \cdots + y_n L_n(x), \end{aligned} \tag{7}$$

where $L_j(x)$ are called **Lagrange polynomials** with the following properties:

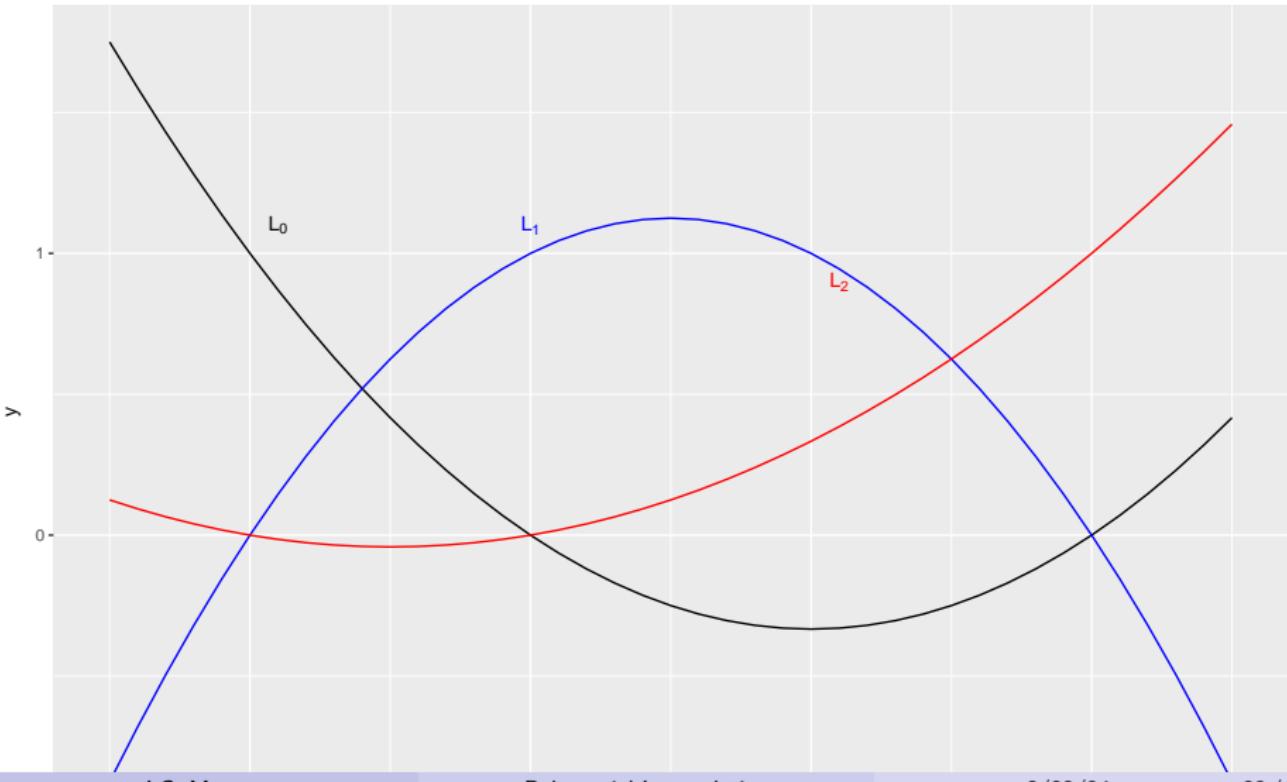
$$L_j(x_i) = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \quad j = 0, 1, \dots, n. \tag{8}$$

Using this definition, it isn't hard to show that

$$p_n(x_i) = y_i, \quad i = 0, 1, 2, \dots, n.$$

Visually

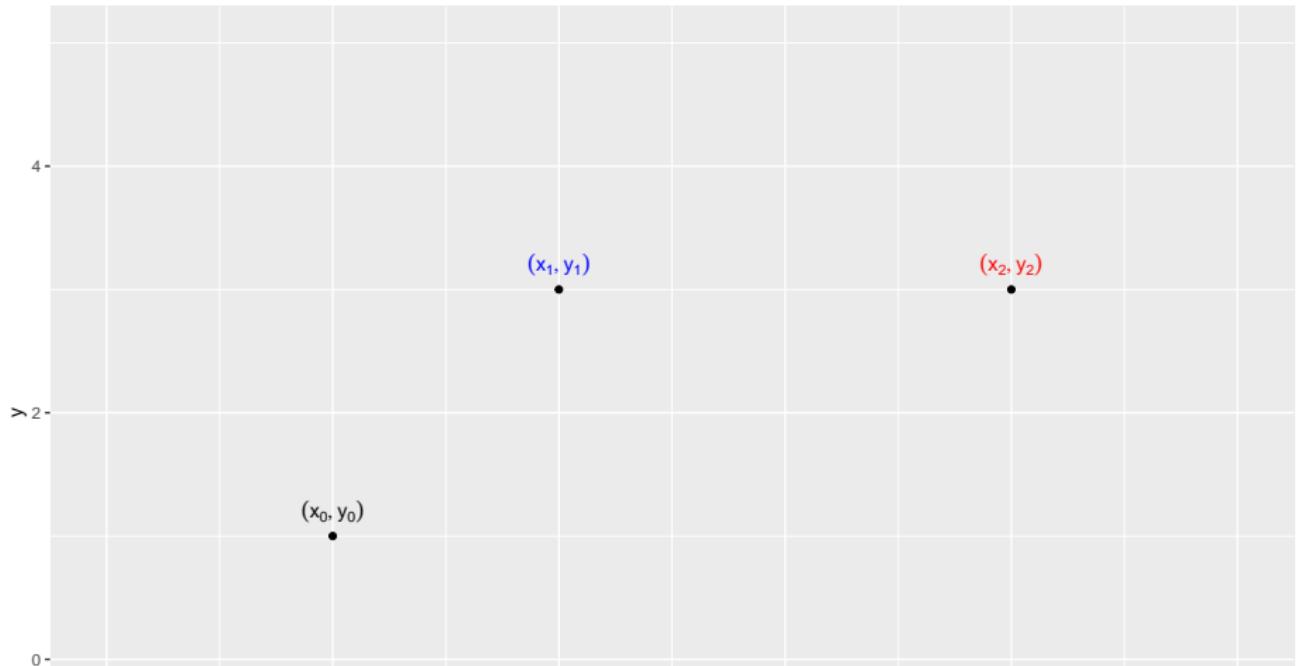
Lagrange Polynomials



Example

Construct $p_2(x)$ for data points $(1, 1), (2, 3), (4, 3)$ using the Lagrange polynomial formulation.

Fit a quadratic polynomial to 3 data points



Solution

The first step is to compute the individual $L_j(x)$ for the given data points $(1, 1), (2, 3), (4, 3)$.

Let's start with the first Lagrange polynomial $L_0(x)$. By definition (Equation 8), we know that the polynomial must satisfy the following conditions for $j = 0$.

$$L_0(x_i) = \begin{cases} 0, & i \neq 0 \\ 1, & i = 0 \end{cases}$$

(cont.)

We also know that L_0 must be a quadratic polynomial (**Why?**) As a result, we know that the general form for L_0 must be:

$$L_0(x) = a \cdot (x - r_1)(x - r_2),$$

where r_1, r_2 are the two roots.

To satisfy condition $L_0(x) = 0$ at the two nodes other than x_0 , means that the two other nodes, $x_1 = 2, x_2 = 4$ must be the two roots of the quadratic.

$$L_0(x) = a \cdot (x - 2)(x - 4).$$

(cont.)

We can now use the remaining condition $L_0(x_0) = 1$ to see that:

$$\begin{aligned}L_0(x_0) &= 1 = a \cdot (x_0 - 2)(x_0 - 4), \\&= a \cdot (1 - 2)(1 - 4) \quad \text{plugging in } x_0 \\1 &= 3a\end{aligned}$$

Therefore $a = 1/3$, giving us the solution:

$$L_0(x) = \frac{1}{3}(x - 2)(x - 4)$$

(cont.)

Using the same procedure, we can compute:

$$L_1(x) = -\frac{1}{2}(x-1)(x-4).$$

and finally:

$$L_2(x) = \frac{1}{6}(x-1)(x-2).$$

I leave the computation of these two for you to practice on.

Putting it all together

Using (Equation 7), we can now combine the 3 Lagrange polynomials and multiply by the corresponding y_j values to give us the desired interpolating polynomial :

$$p_2(x) = y_0 \cdot \frac{(x-2)(x-4)}{3} - y_1 \cdot \frac{(x-1)(x-4)}{2} + y_2 \cdot \frac{(x-1)(x-2)}{6}$$

Substituting for the values of y we get:

$$p_2(x) = \frac{1}{3}(x-2)(x-4) - \frac{3}{2}(x-1)(x-4) + \frac{3}{6}(x-1)(x-2) \quad (9)$$

Visually

Figure 2 shows all three of the Lagrange polynomials in addition to the final interpolating polynomial $p_2(x)$ of degree 2 for the 3 given data points.

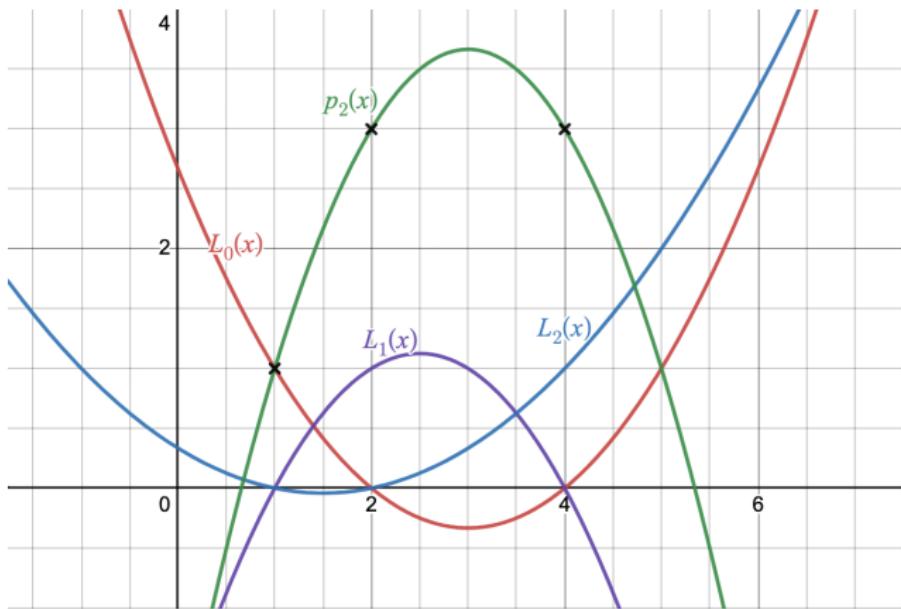


Figure 2: Lagrange Polynomials and resulting 2nd degree polynomial

Uniqueness (again)

Important

While the two approaches appear different, they yield the same second degree interpolating polynomial.

Proof. Suppose there were 2 polynomials $p_2(x), q_2(x)$ that interpolate the given points. Define $r(x) = p_2(x) - q_2(x)$.

First note that $r(x)$ is also of degree ≤ 2 . But

$$r(x_i) = p_2(x_i) - q_2(x_i) = y_i - y_i = 0, \quad i = 0, 1, 2$$

This means that $r(x)$ has three distinct roots and has degree ≤ 2 . That isn't possible unless $r(x) \equiv 0$ by the Fundamental Theorem of Algebra. Therefore $p_2(x) = q_2(x)$.

Exercise

Show that the monomial polynomial approach yields the same polynomial as the Lagrange polynomial approach for our example, i.e. Equation 3 = Equation 9 .

$$\begin{aligned} p_2(x) &= \frac{1}{3}(x - 2)(x - 4) - \frac{3}{2}(x - 1)(x - 4) + \frac{3}{6}(x - 1)(x - 2) \\ &= \frac{1}{3}(-2x^2 + 12x - 7). \end{aligned}$$

General Form of Lagrange Polynomials

To generalize this idea to a higher number of data points, first notice that we can write the Lagrange polynomial as:

$$\begin{aligned} L_j(x) &= \frac{(x - x_0)(x - x_1) \cdots (\textcolor{blue}{x - x_{j-1}})(\textcolor{blue}{x - x_{j+1}}) \cdots (x - x_n)}{(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)} \\ &= \prod_{i=0, i \neq j}^n \frac{x - x_i}{x_j - x_i} \quad j = 0, 1, \dots n. \end{aligned} \tag{10}$$

and

$$p_n(x) = \sum_{j=0}^n y_j L_j(x).$$

Does it satisfy our desired properties?

Recall we want the Lagrange polynomials to satisfy:

$$L_j(x_i) = \delta_{ij}, \quad i, j = 0, 1, \dots, n$$

Note that the term in the numerator is missing the j th term:

$$(x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)$$

Hence for any $x = x_i$, other than x_j , (i.e. $\forall x_i \neq x_j$)

$$(x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n) = 0$$

Does it satisfy our desired properties?

In addition, if $x = x_j$, then the numerator and the denominator are equal,

$$L_j(x_j) = \frac{(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)}{(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)} = 1$$

hence $L_j(x)$ satisfies the conditions for a Lagrange polynomial.

Given this formulation, the construction and evaluation of the interpolating polynomials proceeds as before.

Construction

① **Construction of $L_j(x)$.**

To simplify notation, we define the following terms for the denominators of the Lagrange polynomials:

$$\begin{aligned}\rho_j &= \prod_{i \neq j}^n x_j - x_i \quad j = 0, 1, \dots, n, \\ w_j &= \frac{1}{\rho_j} \quad j = 0, 1, \dots, n,\end{aligned}\tag{11}$$

where the w_j are called the **barycentric weights**. ([@berrut2004])

Remark: Having computed these weights that's it for construction!

Notice also that this computation can be done in $O(n^2)$ flops.

Evaluation

② **Evaluation of** $p_n(x)$

To evaluate the polynomial we just need to bring the different parts together. In order to simplify this process, first notice that if we define

$$\begin{aligned}\Psi(x) &= (x - x_0)(x - x_1) \cdots (x - x_n) \\ &= \prod_{i=0}^n (x - x_i)\end{aligned}\tag{12}$$

then we can write the numerators of the Lagrange polynomials as:

$$\frac{\Psi(x)}{(x - x_j)}$$

Here all we've done is notice that the numerator of Equation 10 is nothing more than Equation 12 divided by the missing term $(x - x_j)$.

Evaluation (cont.)

This leads to the following form for the interpolating polynomial.

$$p_n(x) = \Psi(x) \sum_{j=0}^n \frac{w_j \cdot y_j}{(x - x_j)}. \quad (13)$$

- Notice that since $\Psi(x)$ doesn't depend on j it can be pulled out of the summation sign.
- This form also has the advantage that we just need to compute the $\Psi(x)$ term once!
- The rest of the evaluation can be done in $O(n)$ flops.

Example

Let's do one quick example using the data given below:

Table 2: Data

	$i = 0$	$i = 1$	$i = 2$	$i = 3$
x	-1.1	1.1	2.2	0.0
y	0.0	6.75	0.0	0.0

Example Solution: Construction

First we construct the **barycentric weights**, $w_j = 1/\rho_j$, $j = 0, 1, 2, 3$.

- $j = 0$

$$\begin{aligned}\rho_0 &= \prod_{i \neq j}^n (x_j - x_i), \quad j = 0 \\ &= (x_0 - x_1)(x_0 - x_2)(x_0 - x_3) \\ &= (-1.1 - 1.1)(-1.1 - 2.2)(-1.1 - 0) \\ &= (-2.2)(-3.3)(-1.1) \\ &= -7.986\end{aligned}$$

Example Solution: Construction (cont.)

- Similarly for $j = 1$

$$\begin{aligned}\rho_1 &= \prod_{i \neq j}^n (x_j - x_i), \quad j = 1 \\ &= (x_1 - x_0)(x_1 - x_2)(x_1 - x_3) \\ &= (1.1 - (-1.1))(1.1 - 2.2)(1.1 - 0) \\ &= (2.2)(-1.1)(1.1) \\ &= -2.662\end{aligned}$$

Example Solution: Construction (cont.)

- I'll leave the computation of the other two as an exercise.

$$\begin{aligned}\rho_2 &= (x_2 - x_0)(x_2 - x_1)(x_2 - x_3) & j = 2 \\ &= 7.986\end{aligned}$$

$$\begin{aligned}\rho_3 &= (x_3 - x_0)(x_3 - x_1)(x_3 - x_2) & j = 3 \\ &= 2.662\end{aligned}$$

Having computed the ρ values, the barycentric weights, w_j , can now be easily computed.

An interesting feature of this formulation is that the construction of the interpolating polynomial **does not depend on the values of y** - we only need the data nodes x_i .

Computational Tip

Tip

Once we have computed the weights for a given set of x values, we can use them for any function whatsoever (as long as we know values at those nodes). This also means that since we never assumed any order of the nodes, the weights are independent of the order that the nodes are in. Based on this, it is not hard to show that Equation 13 can be re-written in the more elegant form of:

$$p_n(x) = \frac{\sum_{j=0}^n \frac{w_j \cdot y_j}{(x-x_j)}}{\sum_{j=0}^n \frac{w_j}{(x-x_j)}}. \quad (14)$$

where the term $\Psi(x)$ has been eliminated.

Summary - Lagrange basis functions as an interpolating function

- Introduced the concept of a Lagrange polynomial that can be used as a basis for an interpolating function
- We also showed how to construct an interpolating polynomial using a special set of weights called barycentric weights.
- The construction of this polynomial can be shown to be $O(n^2)$, and does not depend on the values of y or $f(x)$.
- Once the construction has been completed, the interpolating polynomial can be used for any number of different functions, each evaluation costing $O(n)$ flops.
- It can also be shown that if we choose the data point properly, then the algorithm for constructing and evaluating this polynomial is stable. ([@higham2004])

References

- ① Interpolation and Approximation. [@davis1975]
- ② Barycentric Lagrange Interpolation. [@berrut2004]
- ③ The numerical stability of barycentric Lagrange interpolation.
[@higham2004]

Divided Differences and Newton Interpolating Polynomials

Math 131: Numerical Analysis

J.C. Meza

2/27/24

Section 1

Divided Differences

Divided Differences

- Although an interpolating polynomial for a given set of points is unique, there are several algebraic representations we can use. We have one more form for interpolating polynomials that is frequently used.
- It has the advantage that the coefficients can be easily computed as new data becomes available.
- This could prove especially useful in an experimental setup where one may not know how much data will be available at the beginning of the experiment.

Notation

We'll need some new notation first in order to write down this new form for an interpolating polynomial.

- Let's first recall the shifted form for expressing a polynomial:

$$p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \dots \\ + c_n(x - x_0)(x - x_1) \dots (x - x_{n-1})$$

Note that we could determine the coefficients through the following procedure:

$$p_n(x_0) = c_0 = f(x_0)$$

$$p_n(x_1) = c_0 + c_1(x_1 - x_0) = f(x_1) \implies c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

...

Divided Differences

Definition

The **zeroth divided difference** of a function f with respect to x_i is defined as $f[x_i] = f(x_i)$. Using the procedure above as our guide, the **first divided difference** is defined as:

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

Higher order differences

Higher order divided differences can likewise be defined **recursively** in terms of lower order divided differences. For example the **second divided difference** can be expressed as:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i},$$

so specifically for $i = 0$ we would have:

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

Higher order differences (general form)

In general, we can express the divided difference as:

$$f[x_0, x_1, x_2, \dots, x_j] = \frac{f[x_1, x_2, \dots, x_j] - f[x_0, x_1, \dots, x_{j-1}]}{x_j - x_0} \quad (1)$$

Equation 1 is also known as the **Newton divided difference** form.

Useful Properties

Divided differences have some useful properties that we will use later on.

The first important one is that the ***order of the data points doesn't matter.***

In other words:

$$f[x_{i_0}, x_{i_1}, x_{i_2}, \dots, x_{i_n}] = f[x_0, x_1, x_2, \dots, x_n]$$

where $(i_0, i_1, i_2, \dots, i_n)$ is a permutation of $0, 1, 2, \dots, n$.

One can easily show that, for example:

$$f[x_1, x_0] = f[x_0, x_1].$$

Example

Compute the first and second divided differences for $f(x) = \cos(x)$, using $x_0 = 0.2, x_1 = 0.3, x_2 = 0.4$.

$$\begin{aligned}f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} \\&= \frac{\cos(0.3) - \cos(0.2)}{0.3 - 0.2} \\&= \frac{0.955336489 - 0.980066578}{0.1} \\&= -0.24730089\end{aligned}$$

Example (cont.)

Likewise

$$\begin{aligned}f[x_1, x_2] &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} \\&= \frac{\cos(0.4) - \cos(0.3)}{0.1} \\&= \frac{0.921060994 - 0.955336489}{0.1} \\&= -0.34275495\end{aligned}$$

Example (cont.)

The second divided difference is therefore given by:

$$\begin{aligned}f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\&= \frac{(-0.34275495) - (-0.24730089)}{0.4 - 0.2} \\&= -0.477727030\end{aligned}$$

Section 2

Newton Interpolating Polynomials

Newton Interpolating Polynomials

- While divided differences have many applications, the one we will use here is as a means to write down an interpolating polynomial with an important computational property.
- Suppose as before that we have an interpolating polynomial of degree n such that $p_n(x) = f(x_i)$, $i = 0, 1, \dots, n$ with distinct data points.
- Then an interpolating polynomial can be written using Newton divided differences as follows:

$$p_1(x) = f(x_0) + f[x_0, x_1](x - x_0)$$

$$p_2(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$

⋮

$$\begin{aligned} p_n(x) &= f(x_0) + f[x_0, x_1](x - x_0) + \dots \\ &\quad + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}) \end{aligned}$$

Advantages of Newton Polynomial

Note that for all interpolating polynomial of degree $n > 1$, they can all be constructed by using the previous interpolating polynomial of degree $n - 1$, by simply adding one additional term. For example:

$$p_1(x) = f(x_0) + f[x_0, x_1](x - x_0)$$

$$p_2(x) = p_1(x) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$

⋮

$$p_n(x) = \color{blue}{p_{n-1}(x)} + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

Adding new points

In general we can write:

$$p_{k+1}(x) = \textcolor{blue}{p_k(x)} + f[x_0, x_1, \dots, x_k](x - x_0)(x - x_1) \cdots (x - x_k)$$

The beauty of this form is that we can easily go from a polynomial of degree k to degree $k + 1$, by adding one (k -th order) divided difference.

Remark

As before, it is not difficult to show that the Newton divided difference interpolating polynomial is exactly the same as the one we derived earlier (if perhaps in a slightly disguised form).

Exercise: Divided Difference Table

Table 1: Divided Difference Table

i	x_i	$f[\cdot]$	$f[\cdot, \cdot]$	$f[\cdot, \cdot, \cdot]$
0	1	1		
1	2	3		
2	4	3		

Compute the second degree Newton polynomial, i.e. $p_2(x)$.

Theorem

The following theorem will prove useful in future analysis.

Theorem

Let $n \geq 1$ and assume that $f(x) \in C^n[a, b]$. Let x_0, x_1, \dots, x_n be $n + 1$ distinct points in $[a, b]$. Then

$$f[x_0, x_1, x_2, \dots, x_n] = \frac{1}{n!} f^n(\xi),$$

for some ξ between the minimum and maximum of x_0, x_1, \dots, x_n .

Remark: This might look familiar as it looks similar to the remainder term in the Taylor polynomial. As a result, it won't come as a surprise that it will be useful in our error analysis

Summary

We've now seen three different approaches for computing interpolating polynomials. Let's briefly summarize some of the most important features of each of them.

Table 2: Interpolating Polynomials Summary

Basis	$\phi_j(x)$	Constr.	Eval.	Feature
Monomial	x^j	$O(n^3)$	$O(n)$	Simple, easy
Lagrange	$L_j(x)$	$O(n^2)$	$O(n)$	$c_j = y_j$, most stable
Newton	$\prod_{i=0}^{j-1} (x - \frac{x_i}{x})$	$O(n^2)$	$O(n)$	Adaptive (can add new points)

Section 3

Error Analysis

Error in Polynomial Interpolation

If we have an interpolating polynomial for a given function at a set of points, we know that by definition, it must match the function exactly at the nodes. However, this also brings up a related question - How does $p_n(x)$ behave at points other than the nodes $\{x_i\}$?

In order to consider this question we will make a few assumptions to help us with the analysis, specifically:

- ① $f(x)$ is defined on $[a, b]$.
- ② f has all needed derivatives and they are bounded.

Error Function

Let's first define the error function for the $n-th$ degree interpolating polynomial:

$$e_n(x) = f(x) - p_n(x), \quad x \in [a, b].$$

First step

The question we would like to ask is what does the error look like for a point in the interval $[a, b]$ that is not one of the node points.

Here we will use the simple observation that any x that is not a node can be treated as a new interpolation point. As such, we can use the Newton Interpolation formula for adding a new point, i.e.

$$f(x) = p_{n+1}(x) = \textcolor{blue}{p_n(x)} + f[x_0, x_1, \dots, x_n, x] \Psi_n(x),$$

where

$$\Psi_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n).$$

Substituting this into the error function we get:

$$e_n(x) = f(x) - p_n(x) = f[x_0, x_1, \dots, x_n, x] \Psi_n(x).$$

Alternate form for divided difference

Using previous theorem we can replace the divided difference with the derivative:

$$f[x_0, x_1, x_2, \dots, x_n, x] = \frac{f^{n+1}(\xi)}{(n+1)!},$$

to give us the following formula for the error function:

$$e_n(x) = f(x) - p_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!} \Psi_n(x).$$

This leads us directly to the following theorem.

Polynomial Interpolation Error

Theorem (Polynomial Interpolation Error)

Let $n \geq 0$ and $f \in C^n[a, b]$. Suppose we are given x_0, x_1, \dots, x_n distinct points in $[a, b]$. Then

$$f(x) - p_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!} \Psi_n(x), \quad (2)$$

with $\Psi_n(x) = \prod_{i=0}^n (x - x_i)$, for $a \leq x \leq b$, where $\xi(x)$ is an unknown between the min and max of x_0, x_1, \dots, x_n and x .

Furthermore

$$\max |f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \max_{a \leq t \leq b} |f^{(n+1)}(t)| \cdot \max_{a \leq s \leq b} \prod_{i=0}^n |s - x_i|$$

Interpretation

- This theorem is nice, but in order to proceed much further, we would need to know more about both f and $\Psi_n(x)$.
- Nonetheless, it does suggest that in order to minimize the error, it would be good to keep the new point x close to one of the interpolating points.
- A final note is in order. We used the Newton form to obtain this error bound, but the bound will apply to other polynomial forms since we know we have a unique polynomial, and the various forms we have used are merely disguised versions of each other.

Practical Tips

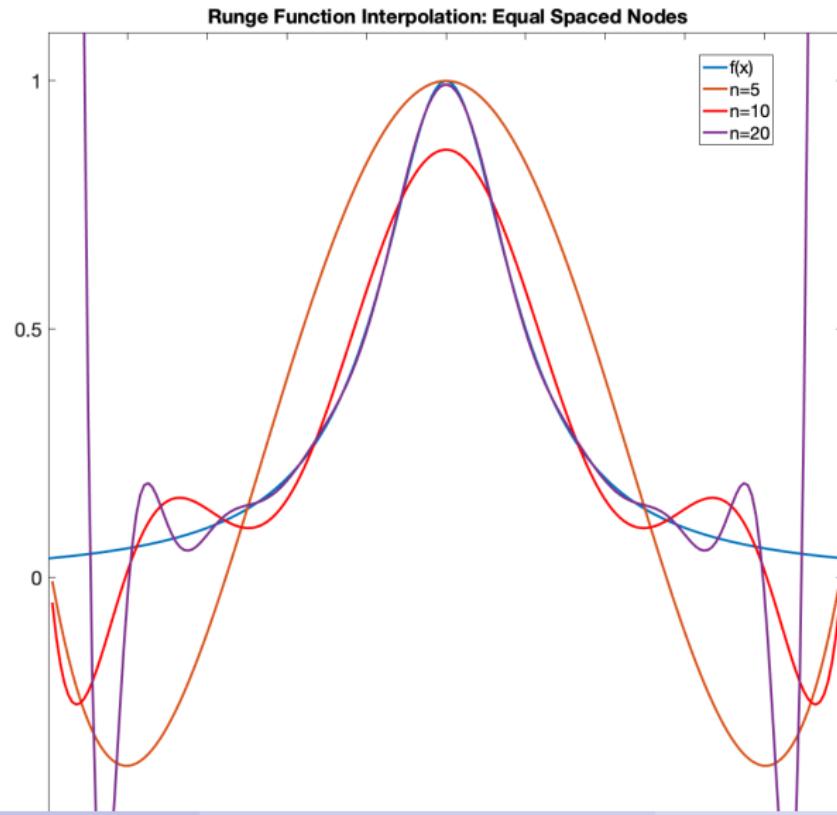
- ① In understanding the error we might encounter when using interpolation, we can use Equation 2, to give us a sense of how the error behaves throughout the interval. Clearly, at the node points themselves, the error will be zero, but what about the rest of the points in $[a, b]$?
- ② When considering the error bounds above, the interpolation error is likely to be smaller when evaluated at points close to the middle of the domain.
- ③ In practice, high degree polynomials with equally spaced nodes are not suitable for interpolation because of this oscillatory behavior.
- ④ However, if a set of suitably chosen data points that are not equally spaced may be useful in obtaining polynomial approximations of some functions.

Example Runge Function

- An excellent example of the type of behavior that can occur when using equally spaced nodes was described by Runge in 1901.
- If one uses equally spaced nodes, for example $x_i = \frac{2i}{n} - 1$ on the interval $[-1, 1]$, then it can be shown that the interpolation error grows without bound at the ends of the interval.
- For example consider the seemingly innocuous function:

$$f(x) = 1/(1 + x^2), \quad -5 \leq x \leq 5.$$

Example (cont.)



High-Degree Polynomials - DON'T!

Important

In general, one should be wary of using a high-degree polynomial as they can oscillate drastically and care must be taken whenever used. Other approaches will prove to be more useful in situations where more accuracy is required or more data points are given.

Chebyshev Points

The Chebyshev points are defined on the interval $[-1, 1]$ by:

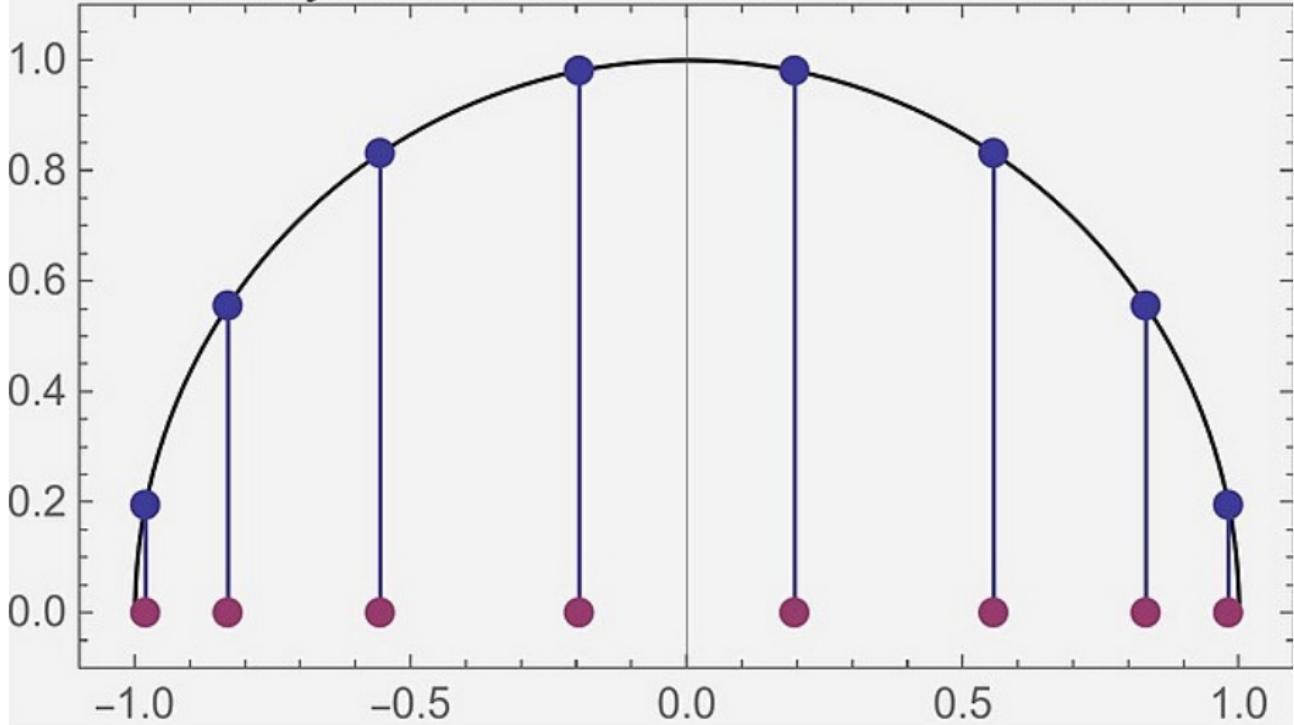
$$x_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right), \quad i = 0, \dots n. \quad (3)$$

To generate the desired points for a general interval $[a, b]$, one then uses the transformation on Equation 3:

$$\tilde{x}_i = a + \frac{b-a}{2}(x_i + 1), \quad i = 0, \dots n. \quad (4)$$

Chebyshev Points

Chebyshev Nodes of the First Kind for $n = 8$



Advantages

These new points have the feature that they are clustered near the end points of the interval rather than being uniformly spaced across the interval.

If one uses, for example, the Lagrange polynomial form with the Chebyshev points defined by Equation 4 then it can be shown that the interpolation error is greatly reduced.

Note

Chebyshev points and interpolation have many interesting properties and many different applications. Unfortunately, we will not have time to go over most of these advanced topics.

Piecewise Polynomials and Cubic Splines

Math 131: Numerical Analysis

J.C. Meza

2024-03-11

Section 1

Review

Polynomial Interpolation Error

Theorem (Polynomial Interpolation Error)

Let $n \geq 0$ and $f \in C^n[a, b]$. Suppose we are given x_0, x_1, \dots, x_n distinct points in $[a, b]$. Then

$$f(x) - p_n(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!} \Psi_n(x), \quad (1)$$

with

$$\Psi_n(x) = \prod_{i=0}^n (x - x_i),$$

for $a \leq x \leq b$, where $\xi(x)$ is an unknown between the min and max of x_0, x_1, \dots, x_n and x .

Practical Tips

- ① We can use Equation 1 to give us a sense of how the error behaves throughout the interval. Clearly, at the node points themselves, the error will be zero, but what about the rest of the points in $[a, b]$?
- ② When considering the error bounds, the interpolation error is likely to be smaller when evaluated at points close to the middle of the domain or near a node.
- ③ High degree polynomials with equally spaced nodes are not suitable for interpolation because of oscillatory behavior and dependence on higher order derivatives.
- ④ Sometimes low-order polynomials with a set of suitably chosen data points may be useful in obtaining approximations of some functions.

Section 2

Piecewise Polynomials

Piecewise Polynomials Motivation

- ① Theorem showed that the error bound depended on both the size of the interval as well as the higher derivatives, which could be large.
- ② Higher order polynomials oscillate (wiggle) a lot. When fitting data that doesn't have a lot of oscillations the polynomial will not approximate the function well in certain areas of the interval (usually near the ends of the intervals).
- ③ Data are often only piecewise smooth, but polynomials are infinitely differentiable. Asking a polynomial to fit data that isn't as smooth as itself may not be fair.
- ④ Changing a single data point could drastically alter the entire interpolant.

What to do?

As we discussed in the practical tips section, it is best to think of using:

- ① low-order polynomials,
- ② within small intervals,
- ③ and only think of them as local approximations.

This leads us to think about using an alternative approach, which can be briefly described as:

Idea

Instead of finding one single polynomial to fit all the data find a set of polynomials for different regions within the given interval.

Mathematically

In more detail, this approach can be described as:

- ➊ Divide the interval $[a, b]$ into a set of smaller subintervals (elements)

$$a = t_0 < t_1 < \dots < t_r = b.$$

The t_i are often referred to as break points, or sometimes just ***knots***.

- ➋ Fit a ***low-degree polynomial*** $s_i(x)$ in each of the subintervals $[t_i, t_{i+1}]$, $i = 0, \dots, r - 1$
- ➌ Patch (glue) the polynomials together so that

$$v(x) = s_i(x), \quad t_i \leq x \leq t_{i+1} \quad i = 0, \dots, r - 1.$$

Piecewise Linears Example

Suppose we were given the following data points

i	0	1	2	3	4
x	1	2	4	5	6
y	1	1.8	2	1.8	0.5

Consider using the Newton form for a linear polynomial **within** each of the sub-intervals.

$$s_i(x) = f(x_i) + f[x_i, x_{i+1}](x - x_i),$$

where

$$t_i \leq x \leq t_{i+1}, \quad 0 \leq i \leq 4.$$

Here note that for now, $t_i = x_i$

Example (cont.)

For example, for $i = 0$, we would have:

$$\begin{aligned}s_0(x) &= f(x_0) + f[x_0, x_1](x - x_0), \\&= 1 + \frac{1.8 - 1}{2 - 1}(x - 1), \\&= 1 + 0.8(x - 1).\end{aligned}$$

Likewise, we would then compute s_1, s_2, s_3, s_4 .

Exercise: Compute s_1 .

Section 3

Error Analysis

Error Bounds

It turns out to be fairly easy to compute an error bound for this case (we've done most of the heavy lifting in the previous sections already).

First let's provide some notation to help us in this new situation.

Let

$$n = r \quad \text{number of subintervals}$$

$$t_i = x_i \quad \text{knots = data points}$$

$$h = \max_{1 \leq i \leq n} (t_i - t_{i-1}) \quad \text{maximum subinterval size}$$

Error Bounds (cont.)

Theorem

For any $x \in [a, b]$.

$$|f(x) - v(x)| \leq \frac{h^2}{8} \max_{a \leq \xi \leq b} f''(\xi),$$

Proof:

First, let's note that for any $x \in [a, b]$, it must lie in some interval, say i , therefore $t_{i-1} \leq x \leq t_i$. We can now apply our Polynomial Interpolation Error theorem, and since we are using linear interpolation, $n = 1$, the bound states that:

$$f(x) - v(x) = \frac{f''(\xi)}{2!}(x - t_{i-1})(x - t_i). \quad (2)$$

Proof (cont.)

Next we note that the maximum of the quantity $(x - t_{i-1})(x - t_i)$ occurs at the point

$$x = \frac{t_{i-1} + t_i}{2}.$$

(Why?)

Therefore we can say that:

$$\begin{aligned}|(x - t_{i-1})(x - t_i)| &\leq \left(\frac{t_i - t_{i-1}}{2}\right)^2, \\ &\leq \frac{h^2}{4}.\end{aligned}$$

The result now follows by substituting this back into Equation 2.

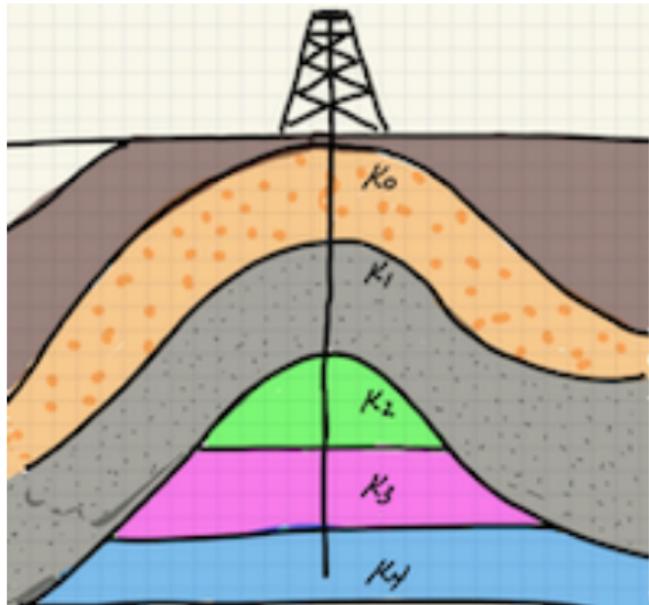
Section 4

Some Special Cases

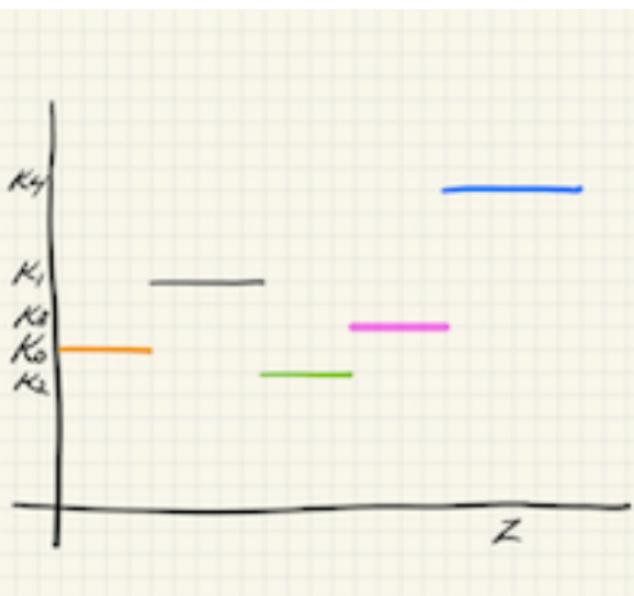
Piecewise Constants

- Before moving to higher-order piecewise interpolation, it might be good to note that sometimes it is useful to consider piecewise constants.
- This approach could be used, for example in applications where the data is known to have discontinuities.
- Consider the case of modeling the subsurface of the earth in oil reservoir and geophysical exploration models.

Oil reservoir modeling



(a) Reservoir Model with different layers



(a) Piecewise constant data for layers

Figure 2: Oil Reservoir Model with piecewise constant data

Section 5

Cubic Splines

Smoothness

Piecewise linear polynomials appear to be a good compromise but they do have one clear disadvantage – the final interpolant will likely have corners at the knots.



Smoothness (cont.)

- What if we want to have a *smoother* interpolant?
- This could be quite important if we are trying to approximate a function that is known to have certain smoothness properties, or
- if we are modeling some physical or engineering problem that we wish to have smoothness, such as an airplane wing or a car body.

Cubic Splines

The most popular approach for creating a smooth piecewise interpolant is known as **cubic splines**.

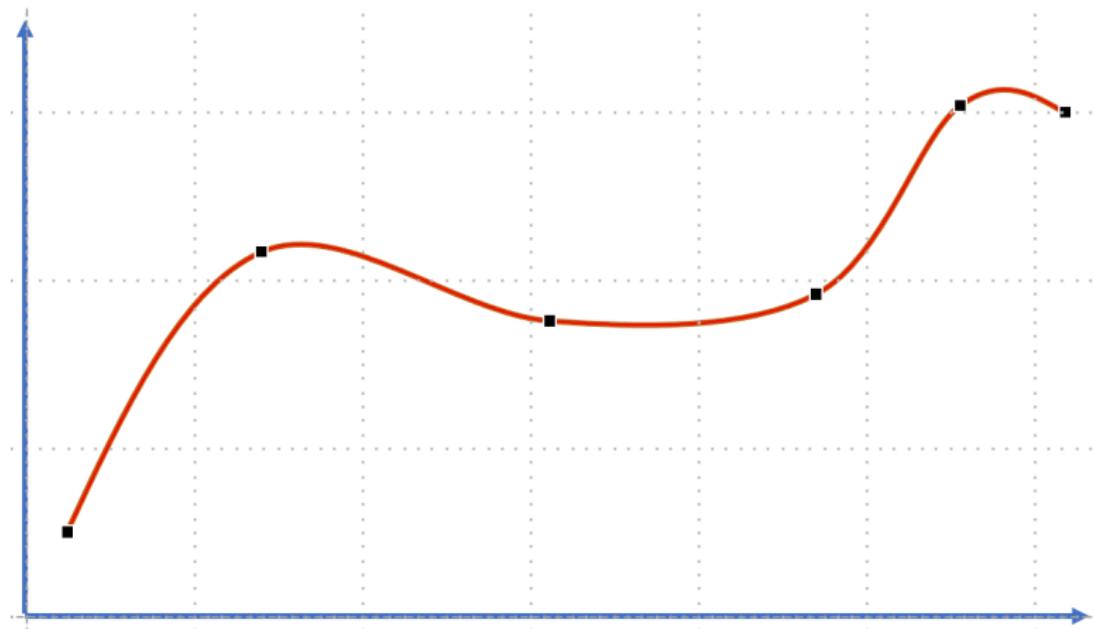


Figure 3: Cubic Spline with 6 data points

Mathematically

Let's consider a cubic interpolant for the i th interval, which we can write as:

$$v(x) = s_i(x) = a_i + b_i(x - t_i) + c_i(x - t_i)^2 + d_i(x - t_i)^3.$$

where

$$t_i \leq x \leq t_{i+1} \quad i = 0, 1, \dots, n - 1$$

Note that there are 4 unknowns a_i, b_i, c_i, d_i per sub-interval. If we had n intervals, then there are $4n$ unknowns in total.

That means if we want to have a unique solution, we need to also have $4n$ equations (conditions) specified.

Approach

The usual approach is to generate these equations through a combination of:

- ① interpolation conditions, and
- ② continuity conditions

Let's first note that in the linear case we had:

$$\begin{aligned}s_i(t_i) &= f(t_i), \quad i = 0, 1, \dots, n - 1, \\ s_i(t_{i+1}) &= f(t_{i+1}), \quad i = 0, 1, \dots, n - 1.\end{aligned}\tag{3}$$

This gave us $2n$ conditions for the $2n$ unknowns. In addition, continuity was implied because

$$s_i(t_{i+1}) = f(t_{i+1}) = s_{i+1}(t_{i+1}).$$

Approach (cont.)

- With cubic splines, we have $4n$ unknowns. **Why?**
- We can use the interpolating conditions (Equation 3) to give us $2n$ conditions.
- The question before us now is how to choose the additional $2n$ conditions required to give us a unique solution.

Approach (cont.)

Idea

Use remaining $2n$ conditions so as to satisfy $v(x) \in C^2[a, b]$.

- In other words, ensure that each of the splines is twice-continuously differentiable:
 - ▶ $s_i(x)$ is continuous at the knots
 - ▶ $s'_i(x)$ is continuous at the knots
 - ▶ $s''_i(x)$ is continuous at the knots

Mathematically

This idea translates into:

$$\begin{aligned} s_i(t_i) &= f(t_i), & i = 0, 1, \dots, n-1, & n \text{ conditions} \\ s_i(t_{i+1}) &= f(t_{i+1}), & i = 0, 1, \dots, n-1, & n \text{ conditions} \\ s'_i(t_{i+1}) &= s'_{i+1}(t_{i+1}), & i = 0, 1, \dots, n-2, & n-1 \text{ conditions} \\ s''_i(t_{i+1}) &= s''_{i+1}(t_{i+1}), & i = 0, 1, \dots, n-2, & n-1 \text{ conditions} \end{aligned} \tag{4}$$

- It is important to note that the last two conditions only hold at the internal knots since that is where two splines meet and need to be aligned to maintain continuity of the derivatives.
- Counting up the conditions therefore leaves us with only $4n - 2$ conditions.

Approaches

There are two popular approaches to resolving this problem:

- ① **Free boundary** (natural spline):

$$v''(t_0) = v''(t_n) = 0 \quad (5)$$

- ② **Clamped boundary**:

$$\begin{aligned} v''(t_0) &= f''(t_0), \\ v''(t_n) &= f''(t_n). \end{aligned} \quad (6)$$

Free vs. Clamped B.C. Comparison

Free Boundary

- Easiest to implement and apply.
- Rather arbitrary and there is no *a priori* reason to expect it to be true.

Clamped Boundary

- More realistic,
- Disadvantage of requiring the second derivative of the function.
- If the second derivative is known (or can be approximated) however, this approach would be preferred.

Section 6

Cubic Splines Example

Cubic Splines Example

Suppose we were given the following data points

i	0	1	2
x	0	1	2
y	1.1	0.9	2.0

Compute the cubic spline that interpolates the above data

Recall:

$$v(x) = s_i(x) = a_i + b_i(x - t_i) + c_i(x - t_i)^2 + d_i(x - t_i)^3, \quad (7)$$

where we have chosen to use $t_i = x_i$, $i = 0, 1$ for this particular case.

Solution

First, let's set up a roadmap of what we need to do:

Given the form for the cubic spline (Equation 7), we need to determine each of 4 coefficients a_i, b_i, c_i, d_i for each of the individual splines, $s_i(x), i = 0, 1$ such that they:

- ① Satisfy the interpolation conditions
- ② Satisfy the continuity conditions
- ③ Satisfy the first derivative conditions
- ④ Satisfy the second derivative conditions

In other words we need to satisfy Equation 3 plus either one of the free or clamped boundary conditions.

Solution (cont.) Interpolation conditions

First note that there are only 3 points, hence we have $n = 2$ intervals. As a result we know that we have $4n = 8$ coefficients that need to be determined and hence we will need 8 equations.

First let's write down what the general form of each of the s_i functions

$$v(x) = \begin{cases} s_0(x) = a_0 + b_0(x - t_0) + c_0(x - t_0)^2 + d_0(x - t_0)^3 & t_0 \leq x < t \\ s_1(x) = a_1 + b_1(x - t_1) + c_1(x - t_1)^2 + d_1(x - t_1)^3 & t_1 \leq x \leq t \end{cases}$$

And plugging in for the actual values of t_0, t_1 , we have

$$\begin{aligned} s_0(x) &= a_0 + b_0x + c_0x^2 + d_0x^3 & 0 \leq x < 1 \\ s_1(x) &= a_1 + b_1(x - 1) + c_1(x - 1)^2 + d_1(x - 1)^3 & 1 \leq x \leq 2 \end{aligned} \quad (8)$$

Solution (cont.) Interpolation conditions

We can set up the first 2 equations by using the interpolation conditions:
 $s_i(t_i) = f(t_i)$.

Setting $x = t_i, i = 0, 1$, into Equation 8 we get

$$s_0(0) = a_0 = 1.1$$

$$s_1(1) = a_1 = 0.9$$

Here we could also have noticed that in each case all the coefficients, except the first one, drop out of the equations for the given value of $x = t_i$.

Solution (cont.) Continuity conditions

Repeating the same procedure and substituting the right-end points of each interval into Equation 8 we have:

$$s_0(1) = a_0 + b_0 + c_0 + d_0 = 0.9$$

$$s_1(2) = a_1 + b_1 + c_1 + d_1 = 2.0$$

or substituting the now known values for a_0, a_1 we have:

$$s_0(1) = 1.1 + b_0 + c_0 + d_0 = 0.9$$

$$s_1(2) = 0.9 + b_1 + c_1 + d_1 = 2.0$$

(9)

At this point continuity is satisfied and we can now look at satisfying the first derivative conditions.

Solution (cont.) First derivative conditions

First let's notice that the only place to apply the derivative conditions is at the one internal knot that we have, namely $t_1 = 1$.

Starting with Equation 8 and taking the derivative we have:

$$\begin{aligned}s'_0(x) &= b_0 + 2c_0x + 3d_0x^2 & 0 \leq x < 1 \\ s'_1(x) &= b_1 + 2c_1(x - 1) + 3d_1(x - 1)^2 & 1 \leq x \leq 2\end{aligned}$$

for which the two equations must be equal at $x = t_1 = 1$, namely

$$s'_0(1) = b_0 + 2c_0 + 3d_0 = s'_1(1) = b_1 + 0 + 0$$

or simplifying

$$b_0 + 2c_0 + 3d_0 = b_1 \tag{10}$$

Solution (cont.) Second derivative conditions

The last step is to enforce continuity of the second derivative at $t_1 = 1$.

Taking the second derivatives:

$$s_0''(x) = 2c_0 + 6d_0 x \quad 0 \leq x < 1$$

$$s_1''(x) = 2c_1 + 6d_1(x - 1) \quad 1 \leq x \leq 2$$

for which the two equations must be equal at $x = t_1 = 1$.

Following the same procedure as before leads us to the following equation:

$$2c_0 + 6d_0 = 2c_1 \tag{11}$$

That gives us a total of 4 equations. As discussed in class, we now have the freedom to choose either one of the free or clamped boundary conditions to complete the cubic spline.

Solution (cont.) Boundary conditions

To complete the example, let's suppose we choose to use the free boundary conditions.

That means that the second derivatives must equal 0 at both of the end knots, $t_0 = 0, t_2 = 2$

$$s_0''(0) = 0 = 2c_0$$

$$s_1''(2) = 0 = 2c_1 + 6d_1$$

That gives us 6 equations with 6 unknowns, so we can solve for all of the remaining coefficients. (Yes, c_0 is actually known in this case, but it won't be in general.)

Complete Solution

If you'd like to complete the example, the final solution is:

$$v(x) = \begin{cases} s_0(x) = 1.1 - 0.525x + 0.325x^3 \\ s_1(x) = 0.9 + 0.45(x - 1) + 0.975(x - 1)^2 \end{cases}$$

Section 7

Summary

Summary

- **Idea** - use piecewise interpolating polynomials within subintervals of the domain as opposed to using one single polynomial over the entire domain.
- Approach has several advantages over using one polynomial including the ability to take into account more of the structure of the problem
- Produces smoother interpolants over the entire region.
- Widely used in practice

Approximation: Linear Least Squares

Math 131: Numerical Analysis

J.C. Meza

March 19, 2024

Section 1

Introduction

Recall - Data Fitting

Suppose we are given a set of data points

$$\{(x_i, y_i)\}_{i=0}^n.$$

The points x_i are sometimes called the **node points** or simply just **nodes**.

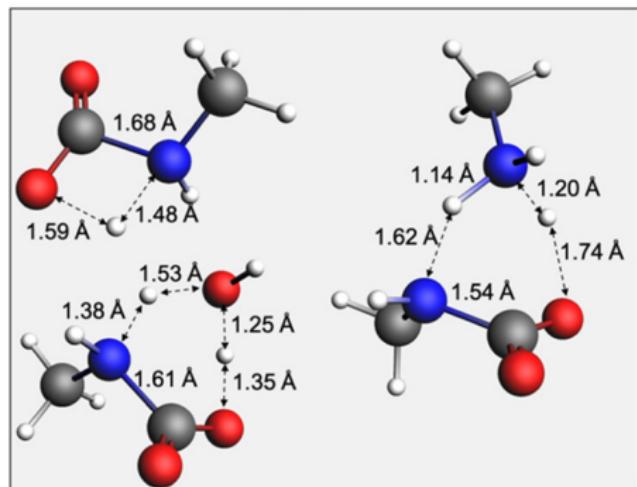
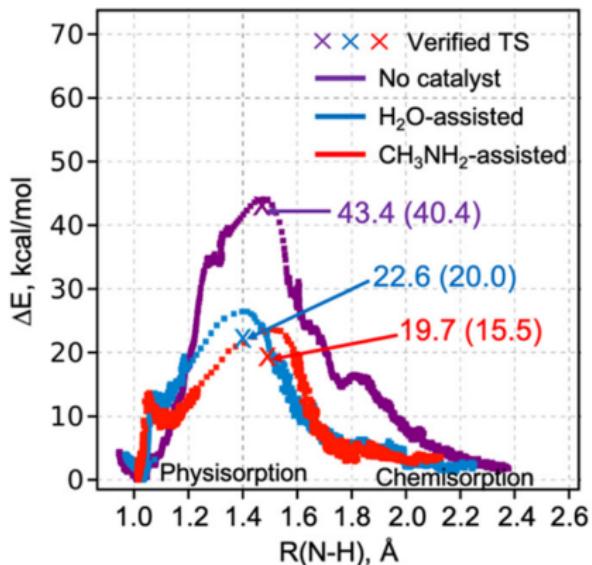
Goal

Find a function $v(x)$ that “fits” the data in some yet to be determined way.

Before:

- Considered the case where the approximation interpolated the data.
- What about situations where the data is known to have some error (experimental, observational, etc.)

Real Data



<https://pubs.acs.org/doi/full/10.1021/acs.jpcc.3c07183>

Real Data

Fig 2. Medium Concentrations (1–128)

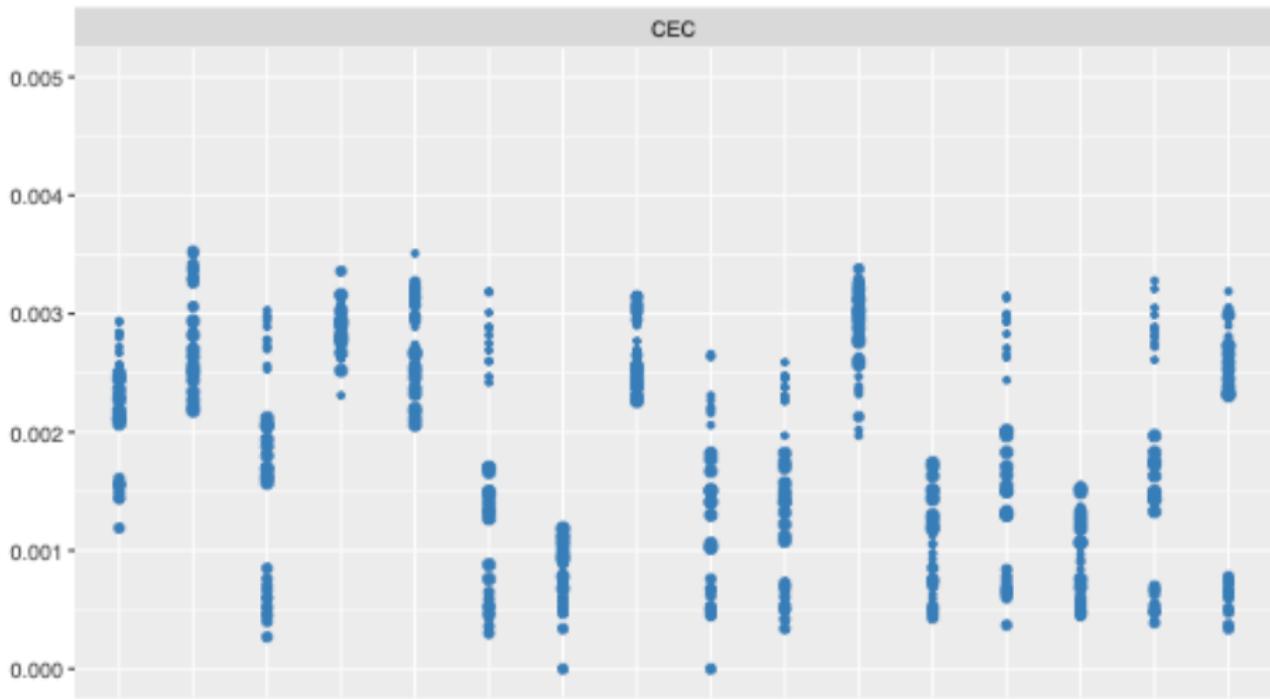


Figure 1: Growth Rate vs. Antibiotics

Section 2

Linear Algebra Review: Norms

Review (Vector Norms)

Consider $x \in \mathbb{R}^n$

By the l_2 -norm we mean the standard Euclidean length:

$$\|x\|_2 = \sqrt{x^T x}$$

By the l_∞ -norm we mean:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

By the l_1 -norm we mean:

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

Example

Suppose we want to compute the error between 2 vectors, x and y

$$x = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix} \quad y = \begin{bmatrix} 1.1 \\ 2.2 \\ 10.0 \end{bmatrix} \quad z = y - x = \begin{bmatrix} 0.1 \\ 0.2 \\ 7.0 \end{bmatrix}$$

$$\|x\|_1 = 0.1 + 0.2 + 7.0 = 7.3$$

$$\|x\|_2 = \sqrt{0.1^2 + 0.2^2 + 97^2} = 7.0036$$

$$\|x\|_\infty = 7$$

Note

All norms are the same order of magnitude in finite-dimensional spaces.

Review (Matrix Norms)

Consider $A \in \mathbb{R}^{m \times n}$. We can define an ***induced norm*** associated with each of the vector norms: $p = 1, 2, \infty$.

$$\begin{aligned}\|A\|_p &= \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \\ &= \max_{\|x\|_p=1} \|Ax\|_p\end{aligned}$$

The matrix norms can be shown to satisfy the following 4 properties:

- ① $\|A\| \geq 0; \|A\| = 0 \iff A = 0$;
- ② $\|\alpha A\| = |\alpha| \|A\| \quad \forall \alpha \in \mathbb{R}$;
- ③ $\|A + B\| \leq \|A\| + \|B\| \quad \forall A, B \in \mathbb{R}^{m \times n}$; ***triangle inequality***
- ④ $\|A \cdot B\| \leq \|A\| \cdot \|B\| \quad \forall A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times k}$; ***consistency***

Matrix Norms (cont.)

This leads to similar formulas for the case of a matrix.

By the *infinity-norm* of a matrix we mean:

$$\|A\|_{\infty} = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|$$

By the *1-norm* of a matrix we mean:

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|$$

The *2-norm* of a matrix is more complicated:

$$\|A\|_2 = \sqrt{\rho(A^T A)} = \text{largest singular value of } A$$

Section 3

Least Squares

Back to the Least Squares problem

Idea

Minimize the errors between the data and the model using one of the above norms, for example the l_2 -norm.

- ① First construct a model as before, e.g.

$$f(x) = c_0\phi_0(x) + c_1\phi_1(x) + \dots + c_n\phi_n(x)$$

- ② Minimize the difference between the function and the data points, e.g.

$$\min_{c_0, \dots, c_n} F(c_0, \dots, c_n) = \sum_{i=0}^m (f(x_i) - y_i)^2$$

Note

This is usually referred to as minimizing the ***sum of squares***

Choice of basis functions

Basis function vary widely and depend on the problem to be solved.

Examples:

① $\phi_j(x) = x^j$

② $\phi_j(x) = e^{\beta_j x}$, β_j are known

③ $\phi_j(x) = \sin(\beta_j x)$, β_j are known

④ Piecewise-polynomials or even mixed basis functions

$$\phi_0(x) = x^3, \quad \phi_1(x) = 1/x, \quad \phi_2(x) = \ln(x)$$

$$f(x) = c_0 x^3 + c_1/x + c_2 \ln(x)$$

Remark

The important thing to note is that the coefficients appear **linearly** in the equation.

General Case

This leads us to set up the minimization problem as:

$$\begin{aligned}\min_{c_0, \dots, c_n} F(c_0, \dots, c_n) &= \sum_{k=0}^m (f(x_k) - y_k)^2 \\ &= \sum_{k=0}^m \left(c_0 \phi_0(x_k) + \dots + c_n \phi_n(x_k) - y_k \right)^2\end{aligned}$$

If we notice that this is really just a quadratic function, then it makes sense to take the derivative and set it equal to zero. But now because we're not in 1D, we need to take all of the partial derivatives, i.e.

$$\frac{\partial}{\partial c_j} F(c_0, \dots, c_n) = 0, \quad j = 0, \dots, n.$$

General Case (cont.)

Let's first consider one of the partial derivatives. Note that

$$\frac{\partial}{\partial c_j} F(c_0, \dots, c_n) = 2 \sum_{k=0}^m \left(c_0 \phi_0(x_k) + \dots + c_n \phi_n(x_k) - y_k \right) \cdot \phi_j(x_k)$$

For $j = 0$ this looks like

$$\sum_{k=0}^m c_0 \phi_0(x_k) \phi_0(x_k) + \dots + c_n \phi_n(x_k) \phi_0(x_k) = \sum_{k=0}^m y_k \phi_0(x_k)$$

For $j = n$ this looks like

$$\sum_{k=0}^m c_0 \phi_0(x_k) \phi_n(x_k) + \dots + c_n \phi_n(x_k) \phi_n(x_k) = \sum_{k=0}^m y_k \phi_n(x_k)$$

Matrix notation

In matrix notation we can write this as:

$$\begin{bmatrix} \sum \phi_0(x_k)^2 & \dots & \sum \phi_n(x_k)\phi_0(x_k) \\ \vdots & \vdots & \vdots \\ \sum \phi_0(x_k)\phi_n(x_k) & \dots & \sum \phi_n(x_k)^2 \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \sum y_i \phi_0(x_k) \\ \vdots \\ \sum y_i \phi_n(x_k) \end{bmatrix} \quad (1)$$

Note: all summations run from $k = 0, \dots, m$

or more compactly as:

$$Bc = y$$

General Term

In fact, we can write down the general term for any of the entries in the matrix as:

$$B_{ij} = \sum_{k=0}^m \phi_i(x_k) \phi_j(x_k)$$

Properties of matrix

There are several important properties of the B matrix: 1) symmetric, 2) positive definite, 3) good chance for ill-conditioning when m, n are large.

Example for n=2

Suppose we have only 2 basis functions and we choose them as
 $\phi_0 = 1, \phi_1 = x$

Then using Equation 1 we can show that the equations we need to solve are:

$$\begin{bmatrix} \sum 1 & \sum x_k \\ \sum x_k & \sum x_k^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} \sum y_k \\ \sum y_k x_k \end{bmatrix}$$

Note: all summations run from $k = 0, \dots, m$

Section 4

Linear Least Squares - Normal Equations

Problem Statement

We can state the problem of fitting data by the following:

$$\min_x \frac{1}{2} \|b - Ax\|^2$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $m \geq n$.

The quantity $r = b - Ax$ is called the **residual**.

Minimization problem

Using the definition of the residual we can restate the minimization problem:

$$\min_x \frac{1}{2} \|b - Ax\|^2 = \min_x \Psi(x)$$

where

$$\psi(x) = \frac{1}{2} \|b - Ax\|^2 = \frac{1}{2} \|r\|^2$$

Geometrically

It's much easier to see the conditions for a minimum in a figure. Note that the residual is orthogonal to the column space of A .

Normal equations

From our picture we have:

$$A^T r = 0$$

Using the definition of r and rearranging the equation:

$$A^T r = A^T(b - Ax) = A^T b - A^T A x = 0$$

Leading to what are called the:

Normal equations

$$A^T A x = A^T b$$

Theory

When do we have a solution?

- Does a minimizer exist, i.e. a solution to the normal equations?
- If so, is the solution a global minimizer?
- Is the problem well-conditioned? Ill-conditioned?

LSQ uniqueness

Unique solution to normal equations

If A has full column rank then $A^T A$ is ***symmetric positive definite*** (s.p.d) and the least squares problem

$$\min_x \frac{1}{2} \|b - Ax\|^2$$

has a unique solution that satisfies the normal equations.

Solution to normal equations

First write down the normal equations

$$A^T A x = A^T b$$

Can write the solution to the normal equations as $x = (A^T A)^{-1} A^T b$

Pseudo-inverse

The term

$$A^\dagger = (A^T A)^{-1} A^T$$

is called the **pseudo-inverse**.

Think of the pseudo-inverse as a generalization of the standard matrix inverse for the case where the matrix A is not square

Algorithm: Solving the Normal Equations

- ① Form $B = A^T A$ and $y = A^T b$
- ② Compute the Cholesky factors of B ($B = GG^T$).
- ③ Solve the lower triangular system $Gz = y$ for z
- ④ Solve the upper triangular system $G^T x = z$ for x

Section 5

Example

Data Fitting Example

- Let's consider fitting a set of m data points, (t_i, b_i) by a straight line:

$$v(t) = x_1 + x_2 t$$

- Unlike interpolation, this time we will only require an approximate fit:

$$v(t_i) \approx b_i, \quad i = 1, \dots, m.$$

where we want to:

$$\min ||b - Ax||^2$$

$$A = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix},$$

Note: x is the unknown and b corresponds to the function values.

Exercise

Construct least squares approximation to the following data:

	$i = 1$	$i = 1$	$i = 3$
t_i	0.0	1.0	2.0
b_i	0.1	0.9	2.0

Solution

Step 1. Setup

$$A = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ 1 & t_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1.0 \\ 1 & 2.0 \end{bmatrix}, \quad A^T = \begin{bmatrix} 1 & 1 & 1 \\ t_1 & t_2 & t_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1.0 & 2.0 \end{bmatrix}$$

$$B = A^T A = \begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix} \quad y = A^T b = \begin{bmatrix} 3 \\ 4.9 \end{bmatrix}$$

Solution (cont.)

Step 2. Solve for x using $Bx = y$:

$$\begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix} x = \begin{bmatrix} 3 \\ 4.9 \end{bmatrix}$$

$$x = \begin{bmatrix} 0.5 \\ 0.95 \end{bmatrix}$$

Tip

Note: One can use any standard linear solver. In practice, you should use a linear solver specific to symmetric matrices (Cholesky decomposition).

Notes on solution

- Unlike interpolation, the linear approximation only gets close to the data points
- In fact, it's easy to see that the residual is nonzero

$$r = b - Ax = \begin{bmatrix} 0.05 \\ -0.10 \\ 0.05 \end{bmatrix}$$

and

$$\|r\|_2 = 0.1225$$

Practical Tips

- This approach is easy to set up and solve
- The normal equations generate a s.p.d matrix
- This implies that B is nonsingular and hence the linear system has a unique solution.
- Good linear solvers available for solving s.p.d. matrix
- Unfortunately, this problem may be ill-conditioned, so care must be taken. Can show that the condition number for the $A^T A$ matrix is the square of the condition number of A .

Using other norms

- We've restricted ourselves to the l_2 norm, which is common
- Other norms can be used, for example l_1 , which are popular for certain problems.
- One other norm you might see is the ***infinity*** norm, l_∞ .

Section 6

Summary

Summary -

- Introduced the general linear least squares problem
- The normal equations can be used to find a solution to the least squares problem.
- Applications of least squares abound in all areas of science and engineering.

Section 7

Supplemental Materials (Optional)

Alternate derivation of normal equations

We can state the problem of fitting data by the following:

$$\min_x \frac{1}{2} \|b - Ax\|^2$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $m \geq n$.

The quantity $r = b - Ax$ is called the **residual**.

Minimization problem

Using the definition of the residual we can restate the minimization problem:

$$\min_x \frac{1}{2} \|b - Ax\|^2 = \min_x \Psi(x)$$

where

$$\psi(x) = \frac{1}{2} \|b - Ax\|^2 = \frac{1}{2} \|r\|^2$$

Necessary conditions

- As in the 1D case, our approach is to take the derivative and set it equal to 0.
- As before, because x is a vector, we need to take the derivative of $\psi(x)$ with respect to each of the components of x .

$$\frac{\partial}{\partial x_k} \psi(x) = 0, \quad k = 1, \dots, n.$$

Taking derivatives

If

$$\psi(x) = \frac{1}{2} \sum_{i=1}^m (b_i - Ax_i)^2$$

Then (as before):

$$\frac{\partial}{\partial x_k} \psi(x) = \sum_{i=1}^m \left[(b_i - \sum_{j=1}^n a_{ij}x_j)(-a_{ik}) \right] = 0, \quad k = 1, \dots, n$$

where the last part can be rewritten as:

$$\sum_{i=1}^m a_{ik} \sum_{j=1}^n a_{ij}x_j = \sum_{i=1}^m a_{ik}b_i, \quad k = 1, \dots, n$$

or

$$A^T A x = A^T b.$$

Finite Differences

Math 131: Numerical Analysis

J.C. Meza

April 2, 2024

Section 1

Finite Differences

Motivation

Computing derivatives numerically arises in many situations in numerical analysis as well as scientific computing. Some of the most common examples include:

- Numerically solving ordinary differential equations (ODEs), partial differential equations (PDEs), nonlinear equations and optimization (e.g. Newton's method).
- Computing derivatives of complicated functions.
- Situations where f is not known explicitly or only as a black box.

The basic tools used for numerical derivatives include one tool that we've been using extensively (Taylor series), and one more recently (polynomial interpolation).

Many Other Examples in Science/Engineering

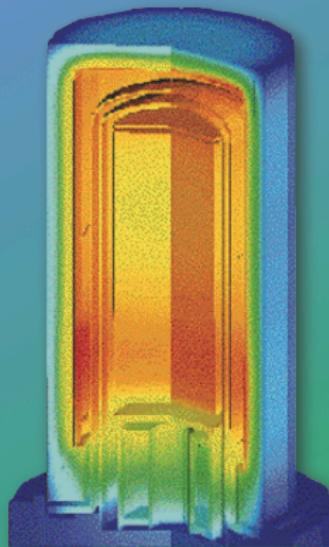
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f \quad \text{Poisson's Equation}$$

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad \text{Heat Equation}$$

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \quad \text{Wave Equation}$$

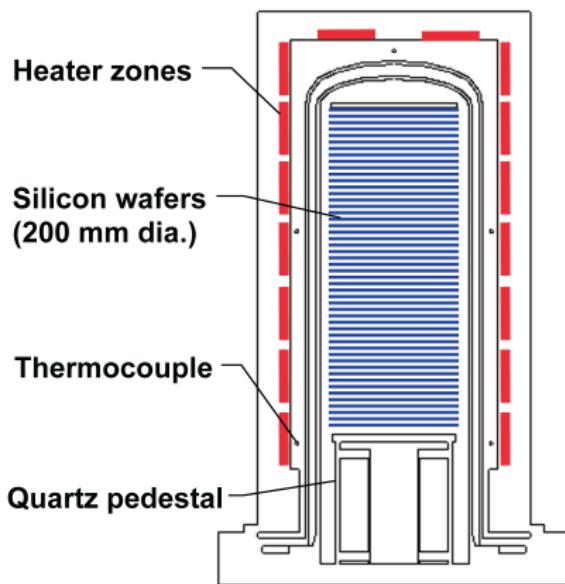
$$\frac{\partial u}{\partial t} = c \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} \quad \text{Convection-Diffusion Equation}$$

Chemical Vapor Deposition Control



Small-batch LPCVD

The design of a small-batch fast-ramp LPCVD furnace can be posed as an optimization problem



- Temperature uniformity across the wafer stack is critical
- Independently controlled heater zones regulate temperature
- Wafers are radiatively heated
- Design parameters:
 - Number of heater zones
 - Size / position of heater zones
 - Pedestal configuration
 - Wafer pitch
 - Insulation thickness
 - Baseplate cooling



Governing Equations

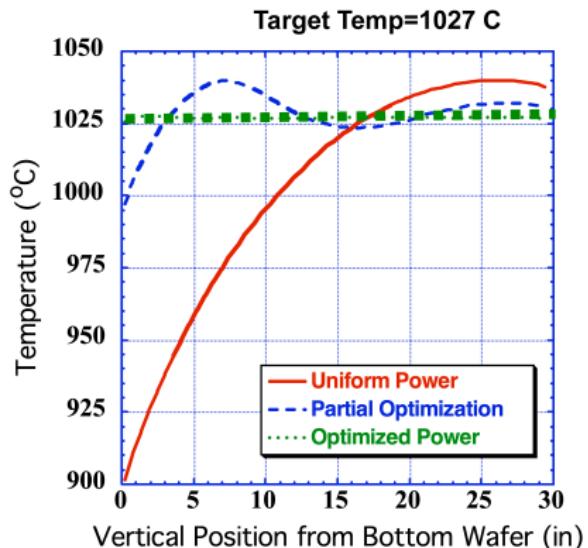
Can cast this problem as heat conduction with radiation losses described by the following equation:

$$\frac{\rho c_P}{k} \frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} - \frac{2\sigma\epsilon}{kR}(T^4 - T_{amb}^4)$$

where ρ, c_P, k, ϵ are material constants, R is radius of silicon wafer, σ is the Stefan-Boltzmann constant, T_{amb} is fixed ambient temperature.

Optimized power distribution

Optimized power distribution enhances wafer temperature uniformity for steady-state operation



*Simulation of Equipment Design Optimization in Microelectronics Manufacturing, J.C. Meza,
C.H. Tong, C.D. Moen, Proc. 30th Annual Simulation Symposium, Atlanta, GA, April 7-9, 1997.*



CVD - Moral of the story

- In theory you have derivatives, but in practice you don't
- Lack of derivatives reduces the choice of numerical methods one can use for optimization
- Accurately computing derivatives trickier than one might think at first glance
- Solution time reduced from a week long job into a 30 minute computation with a solution that was an order of magnitude better

Section 2

First-order Approximations

Taylor Series Approach

Let $f : R^1 \rightarrow R^1$ with as many derivatives as we need.

Recall the *Taylor series expansion* for a function

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi), \quad \xi \in [x, x + h]$$

Rearranging we can write:

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

Forward Difference

This leads to the ***forward difference*** approximation, which can be written as:

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

where we will denote the *truncation error* by

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

Notice that the truncation error can be expressed in Big O notation as $O(h)$ and we therefore say that forward differences are an $O(h)$ approximation or that it is ***first order accurate***.

Reminder

! Truncation Error is NOT Roundoff Error

The truncation error should not be confused with roundoff error!

Backward Difference

Similarly the ***backward difference*** approximation can be written as

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

with a similar error term. Clearly, backward differences are also first order accurate.

Which one should I use?

- There is no major difference between the two; it mostly comes down to a matter of convenience or preference.
- There are however some situations, where it makes more sense to use one over the other.
 - ▶ One recent example is when we had to impose a condition on the cubic spline interpolant and we needed to have the derivative of the first and last spline match the derivative of the function.
 - ▶ In that case, we should use the forward difference approximation for the first spline and the backward difference approximation for the last spline.

Example: Forward Difference

Computation of forward difference for $\exp(x)$, $x = 1$

Table 1: Forward Difference for $\exp(x)$, $x = 1$

h	fprime	F.D	err (F.D)
0.100000	2.718282	2.858842	0.1405601
0.050000	2.718282	2.787386	0.0691040
0.025000	2.718282	2.752545	0.0342635
0.012500	2.718282	2.735342	0.0170603
0.006250	2.718282	2.726794	0.0085124
0.003125	2.718282	2.722534	0.0042517

Exercise

Compute the forward difference of

$$f(x) = \tan(x), x = \pi/4$$

for

$$h = 0.1, 0.05, 0.025, 0.125, 0.00625, 0.003125$$

Solution

Table 2: Forward/Backward Difference for $\tan(x)$, $x = \pi/4$

h	fprime	F.D	B.D	err (F.D)	err (B.D)
0.100000	2	2.230489	1.823712	0.2304888	0.1762881
0.050000	2	2.107112	1.906275	0.1071118	0.0937249
0.025000	2	2.051721	1.951616	0.0517205	0.0483838
0.012500	2	2.025423	1.975410	0.0254233	0.0245897
0.006250	2	2.012605	1.987603	0.0126050	0.0123966
0.003125	2	2.006276	1.993776	0.0062761	0.0062241

Section 3

Central Differences

Central Differences

Consider the Taylor series at $x + h$ and $x - h$

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1(x)) \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2(x)) \end{aligned} \tag{1}$$

for some $\xi_1 \in [x, x+h]$, $\xi_2 \in [x-h, x]$.

(cont.)

Subtracting the second equation from the first gives us:

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{6}[f'''(\xi_1(x)) + f'''(\xi_2(x))]$$

and solving for f' we get:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} \cdot \frac{1}{2}[f'''(\xi_1(x)) + f'''(\xi_2(x))] \quad (2)$$

(cont.)

The last term in Equation 2 that includes the third derivative at two different points can be replaced by using the Intermediate Value Theorem to yield:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(\xi(x)) \quad (3)$$

(see the supplementary materials for details).

Equation 3 is called the central (or centered) difference approximation.

Example: Central vs. F.D.

Compare central vs. forward differences for $f(x) = \tan(x)$, $x = \pi/4$

Table 3: Forward v. Central Difference for $\tan(x)$, $x = \pi/4$

h	F.D	err (F.D)	C.D	err (C.D.)
0.100000	2.230489	0.2304888	2.027100	0.0271004
0.050000	2.107112	0.1071118	2.006693	0.0066934
0.025000	2.051721	0.0517205	2.001668	0.0016683
0.012500	2.025423	0.0254233	2.000417	0.0004168
0.006250	2.012605	0.0126050	2.000104	0.0001042
0.003125	2.006276	0.0062761	2.000026	0.0000260

Section 4

Higher Order

Practical Tips

- It is not too difficult to generate other (higher-order) formulas using similar techniques.
- One must remember that we need to balance accuracy with the additional work needed and to be aware of any special conditions or structure available
- For example the case of specifying derivative conditions for cubic splines at the end points, discussed earlier in this section.

Section 5

Second Derivatives

Second Derivative Formulas

Using similar techniques as for the centered difference formulas we can develop approximations for the second derivative of a function.

Consider once again the Taylor series for a function about a point:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_1(x))$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_2(x))$$

(cont.)

As we want to have a formula for the second derivative, our goal is to get rid of the other terms, and in particular the first derivative. Hence, let's add the two equations:

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{24} \left[f^{(4)}(\xi_1(x)) + f^{(4)}(\xi_2(x)) \right]$$

Solving for f'' we get:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{24} \left[f^{(4)}(\xi_1(x)) + f^{(4)}(\xi_2(x)) \right]$$

(cont.)

Using the same trick we used before by appealing to the IVT, we have the following formula for numerically computing the second derivative.

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{12} f^{(4)}(\xi(x)). \quad (4)$$

This approximation is clearly $O(h^2)$ and we say that it is a ***second order approximation.***

Example: Second derivatives

Second Derivatives using finite differences

Table 4: Second Derivate for $\exp(x)$, $x = 1$

h	$f(x)$	$f_{2\text{prime}}$	err
0.10000	2.718282	2.720548	0.0022660
0.05000	2.718282	2.718848	0.0005664
0.02500	2.718282	2.718423	0.0001416
0.01250	2.718282	2.718317	0.0000354
0.00625	2.718282	2.718291	0.0000088

Section 6

Summary

Summary

- Introduced concept of numerical differentiation
- Presented 3 formulas for first derivatives: two of which are first order approximation and one that is second order
- Presented a formula for second derivatives

Section 7

Supplementary Materials

Review - Big O Notation

Remember that we denote a quantity x as $O(h)$ if it is at most proportional to h , for example Ch for some constant C . One way to think of it as

$$\lim_{h \rightarrow 0} \frac{O(h)}{h} = C < \infty$$

Notice that according to our definition, both forward and backward difference formulas have truncation error that is $O(h)$.

Question

- What is the order of

$$\frac{h}{2}f''(x) + \frac{h^2}{3}f'''(x)?$$

- Just need to take a look at the lowest power of h . If the power of h is 1, we say that it is *first order*. If the power of h is 2, we say that it is *second order*, and so on.

Tip

All other things being equal, we want as high a power of* h for our error term as we can achieve!

Proof for Central Differences Formula

The last term in `?@eq-fwddiff2` that includes the third derivative at two different points can be replaced by using the Intermediate Value Theorem:

Intermediate Value Theorem

Suppose that (i) f is continuous on the closed finite interval $[a, b]$ and (ii) $f(a) < c < f(b)$. Then there exists some point $x \in [a, b]$ such that $f(x) = c$.

(cont.)

Notice that, since the average value must lie in between the value at the two end points a straightforward application of the IVT says:

$$\frac{1}{2}[f'''(\xi_1(x)) + f'''(\xi_2(x))] = f'''(\xi), \quad \xi \in [x - h, x + h]$$

Substituting for the f''' terms yields:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(\xi(x))$$

This is called the central (or centered) difference approximation.

Numerical Differentiation: Part 2

Math 131: Numerical Analysis

J.C. Meza

April 4, 2024

Outline

- Higher-order finite difference formulas
- Stability of numerical differentiation
- Richardson Extrapolation

Section 1

Higher-order Finite Difference formulas

Motivation

- First order numerical differentiation formulas good, but sometimes we desire more accuracy
- Higher order required in certain computing applications
- Already seen one example of higher order in the last lecture - *centered differences*
- Another strategy is to use the Lagrange interpolating polynomials.

($n + 1$)-point Formulas

Idea

- Replace the function $f(x)$ by a Lagrange interpolating polynomial.
- Use the derivative of the Lagrange polynomial in place of the derivative of $f(x)$.

Remark: Appropriate error estimates can also be produced.

General $(n + 1)$ -point formula

The $(n + 1)$ -point formula (Equation 4.2, p. 176, textbook) is given by:

$$f'(x_j) = \sum_{k=0}^n f(x_k) L'_k(x_j) + \frac{f^{(n+1)}(\xi(x_j))}{(n+1)!} \prod_{k=0, k \neq j}^n (x_j - x_k)$$

where $[x_0, x_1, \dots, x_n]$ are $n + 1$ distinct points on some interval.

Observations

- ① In general, more points leads to higher accuracy.
- ② This comes at the expense of more function evaluations and greater roundoff error.
- ③ Formulas most useful when we choose equally spaced points

3-point formulas

The **3-point midpoint (centered)** approximation for the first derivative

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi) \quad (1)$$

The **3-point endpoint** approximation for the first derivative

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + \frac{h^2}{3} f^{(3)}(\xi) \quad (2)$$

Comparison

Note the difference in the number of function evaluations and the constant in the error term.

5-point formulas

The **5-point midpoint (centered)** approximation for the first derivative

$$f'(x) = \frac{f(x - 2h) - 8f(x - h) + 8f(x + h) - f(x + 2h)}{12h} + \frac{h^4}{30} f^{(5)}(\xi) \quad (3)$$

The **5-point endpoint** approximation for the first derivative

$$f'(x) = \frac{1}{12h} \left[\frac{-25f(x) + 48f(x + h) - 36f(x + 2h)}{2h} + 16f(x + 3h) - 3f(x + 4h) \right] + \frac{h^4}{5} f^{(5)}(\xi) \quad (4)$$

Practical Tips

- More points yield more accurate approximations.
- More points also means more function evaluations.
- Roundoff error will also increase with higher-order.
- In general midpoint formulas have smaller errors for same number of points.
- In practice, most commonly used formulas are the 3 and 5 point formulas.

Section 2

Stability

Stability

- Sometimes an algorithm will fail to yield a good result due to stability.
- Computing derivatives by using finite differences is a prime example of such a possibility. This is due to the properties of computer arithmetic.
- In practice there is a delicate balance between truncation error and roundoff error. As a result one needs to be careful when choosing the **step size, h** .

We briefly looked at an example earlier where we looked at the accuracy of the forward difference approximation as a function of h .

Forward Difference Example (Recap of earlier lecture)

Example

Approximate $f'(x)$ for $f(x) = \cos(x)$ and $h = 10^{-3} - 10^{-15}$, $x = \pi/6$.

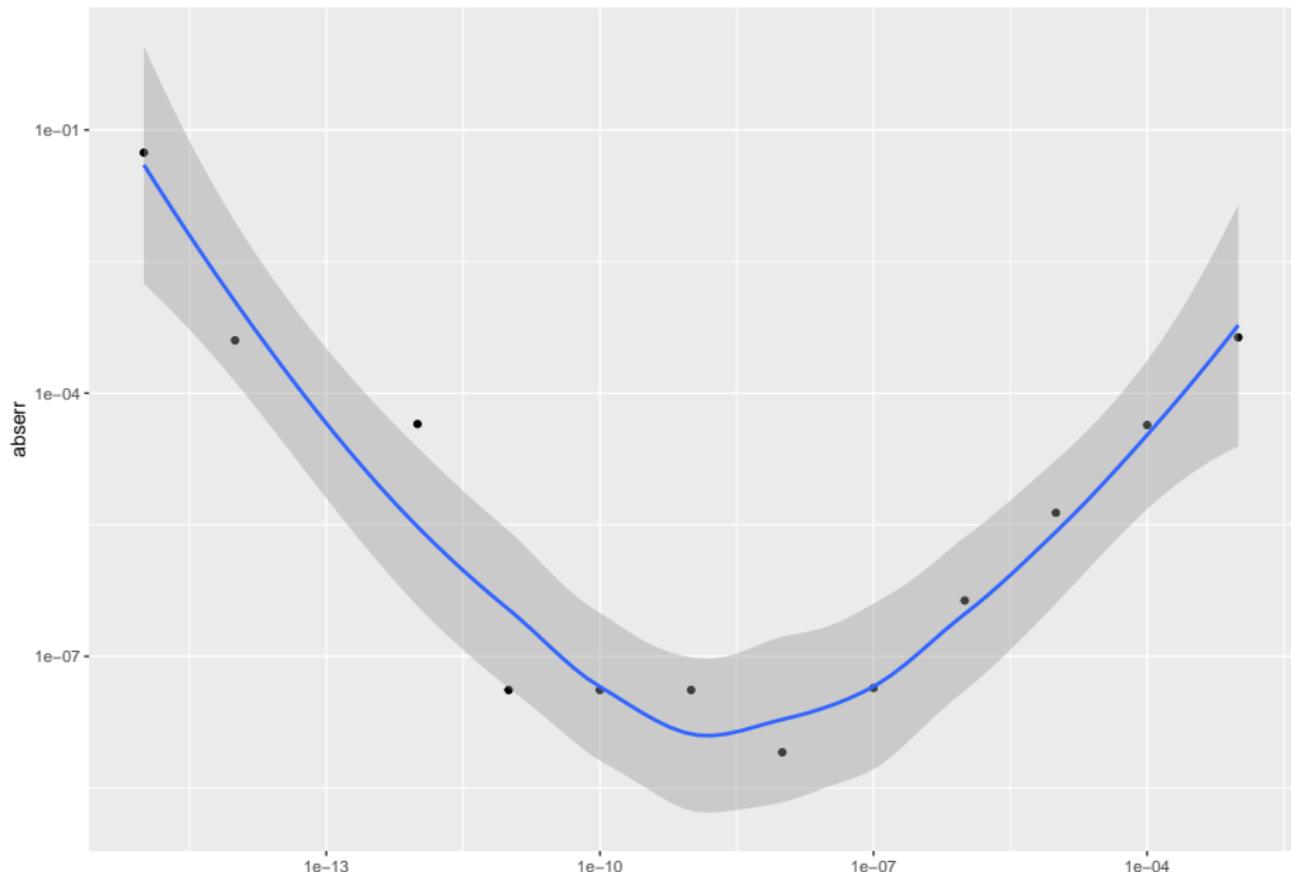
$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

Forward Difference Example Solution

Table 1: Absolute and Relative Error in finite difference (F.D) approximation as a function of stepsize h.

h	F.D	abserr	relerr
1e-03	-0.5004329	4.329293e-04	8.658587e-04
1e-04	-0.5000433	4.330044e-05	8.660088e-05
1e-05	-0.5000043	4.330117e-06	8.660235e-06
1e-06	-0.5000004	4.330569e-07	8.661137e-07
1e-07	-0.5000000	4.359063e-08	8.718126e-08
1e-08	-0.5000000	8.063495e-09	1.612699e-08
1e-09	-0.5000000	4.137019e-08	8.274037e-08
1e-10	-0.5000000	4.137019e-08	8.274037e-08
1e-11	-0.5000000	4.137019e-08	8.274037e-08
1e-12	-0.5000445	4.445029e-05	8.890058e-05
1e-14	-0.4996004	3.996389e-04	7.992778e-04
1e-15	-0.5551115	5.511151e-02	1.102230e-01

Forward Difference Example Solution



Remarks

Important

- Notice that for each decrease in the value of h by an order of magnitude, both the absolute and relative error have a corresponding decrease in their values.
- *This is exactly what the theory predicts.*
- However, it is important to note that this is true only up to a certain point. We'll discuss this more fully when we get to the sections on computer arithmetic and roundoff error.

Analysis

Let's try to analyze what is happening here.

Consider the central difference formula for $f'(x)$ at some point x_0 .

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi(x))$$

When evaluating a function on a computer, we know that we do not have infinite precision, hence suppose:

$$f(x_0 + h) = \hat{f}(x_0 + h) + e(x_0 + h)$$

$$f(x_0 - h) = \hat{f}(x_0 - h) + e(x_0 - h)$$

where $e(x)$ is the roundoff error as a result of computing $f(x)$ and $\hat{f}(x)$ is the computed value of the function.

(cont.)

The error in the finite difference approximation can then be written as:

$$\begin{aligned} f'(x_0) - \left(\frac{\hat{f}(x_0 + h) - \hat{f}(x_0 - h)}{2h} \right) \\ = \frac{e(x_0 + h) - e(x_0 - h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi(x)) \end{aligned}$$

Now assume that:

$$\begin{aligned} e(x_0 \pm h) < \tau, \quad \tau > 0, \\ |f^{(3)}(\xi)| < M \quad \xi \in [x_0 - h, x_0 + h]. \end{aligned}$$

The error in the finite difference approximation can be bounded by

$$\text{error} \leq \frac{\tau}{h} + \frac{h^2}{6} M$$

Proof

$$\begin{aligned} & \left| f'(x_0) - \left(\frac{\hat{f}(x_0 + h) - \hat{f}(x_0 - h)}{2h} \right) \right| \\ &= \left| \frac{e(x_0 + h) - e(x_0 - h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi(x)) \right| \\ &\leq \left| \frac{e(x_0 + h) - e(x_0 - h)}{2h} \right| + \left| \frac{h^2}{6} f^{(3)}(\xi(x)) \right| \\ &\leq \frac{1}{2h} \left(|e(x_0 + h)| + |e(x_0 - h)| \right) + \left| \frac{h^2}{6} M \right| \\ &\leq \frac{1}{2h} (2\tau) + \frac{h^2}{6} M \\ &\leq \frac{\tau}{h} + \frac{h^2}{6} M \end{aligned}$$

The proof is a simple exercise in using triangle inequality and the bounds we assumed on the roundoff error and the third derivative.

Observations

As $h \rightarrow 0$ ***the second term goes to 0, but the first term blows up.***

We need to find the “sweet spot” to minimize the error in the approximation.

Can show that the optimal h^* is given by:

$$h^* = \sqrt[3]{\frac{3\tau}{M}}$$

Unfortunately, one rarely knows either ϵ or M .

Rule of thumb

However a good rule of thumb is to choose h so that you only perturb half the digits of x . That suggests

$$h^* = \sqrt{\epsilon}$$

where ϵ is machine precision. On most modern computers $\epsilon = 10^{-16}$, so this translates into

$$h \approx 10^{-8}$$

! Caution

Finite Difference approximations are an excellent example of unstable algorithms.

Section 3

Richardson Extrapolation

Richardson Extrapolation

- The methods we've studied so far for computing numerical derivatives are good and generally lead to good approximations.
- But what if we need to have greater accuracy?
- This section covers one approach to generating higher-order (more accurate) approximations to derivatives using an idea dating back to 1927

Idea

- In spirit, it is not unlike what we did when we derived the central difference approximation
- By noticing that if we took the two $O(h)$ approximations for the forward and backward difference and combined them in such a way as to cancel out one of the error terms we could get a higher-order approximation $O(h^2)$.

Idea

- Combine 2 approximations with similar error terms to obtain a more accurate approximation.
- Can be used in many different contexts including
 - ▶ interpolation (Aitken),
 - ▶ quadrature (Romberg, adaptive methods),
 - ▶ IVP,
 - ▶ acceleration of convergence of sequences.

Example: Second derivative

Let's consider how we arrived at the 3-point formula for the second derivative:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{12} f^{(4)}(\xi).$$

In that derivation we started with the Taylor series about $x+h$ and $x-h$.

(cont.)

This time let's also consider one additional term, which would give us:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{12}f^{(4)}(x) - \frac{h^4}{360}f^{(6)}(x) + O(h^5) \quad (5)$$

Now replace h by $2h$.

$$\begin{aligned} f''(x) &= \frac{f(x+2h) - 2f(x) + f(x-2h)}{4h^2} - \frac{4h^2}{12}f^{(4)}(x) \\ &\quad - \frac{16h^4}{360}f^{(6)}(x) + O(h^5). \end{aligned} \quad (6)$$

Notice that the second formula has the same term for the fourth derivative except it is multiplied by 4.

(cont.)

- This leads to the natural idea of multiplying the first equation by 4 and subtracting the second equation from it, which will cancel that part of the error term.
- If we do this, we will (after a bit of simple algebra) get to the following formula:

$$\begin{aligned}f''(x) = \frac{1}{12h^2} & \left[-f(x+2h) + 16f(x+h) - 30f(x)\right. \\& \left. + 16f(x-h) - f(x-2h) \right] + \frac{h^4}{90} f^{(6)}(x) + O(h^5).\end{aligned}$$

This formula for the second derivative is now **fourth order** accurate.

Another approach

- Notice that using this approach wasn't specific to the second derivative formula.
- All we needed was an approximation to some quantity where we knew the error term.
- Therefore we could use this same approach anytime.

Richardson Extrapolation

Suppose we can write an approximation to some quantity as:

$$M = N(h) + K_1 h^2 + K_2 h^4 + K_3 h^6 + \dots$$

For example let's define:

$$M = f''(x) \quad \text{and} \quad N(h) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Using these we can re-write Equation 5 as:

$$\begin{aligned} M &= N(h) - \frac{h^2}{12} f^{(4)}(x) - \frac{h^4}{360} f^{(6)}(x) + O(h^5). \\ &= N(h) + K_1 h^2 + K_2 h^4 + \dots \end{aligned} \tag{7}$$

(cont.)

Now let's write the formula using the step size $h/2$.

$$M = N(h/2) + K_1 \frac{h^2}{4} + K_2 \frac{h^4}{16} + \dots \quad (8)$$

Let's multiply Equation 8 by 4 and subtract Equation 7:

$$\begin{aligned} 4M &= 4N(h/2) + K_1 h^2 + K_2 \frac{h^4}{4} + \dots \\ - M &= N(h) + K_1 h^2 + K_2 h^4 + \dots \\ \hline 3M &= \left[4N(h/2) - N(h) \right] + K_2 \left(\frac{h^4}{4} - h^4 \right) + \dots \end{aligned}$$

Solving for M , leads to:

$$M = \frac{1}{3} [4N(h/2) - N(h)] + O(h^4)$$

(cont.)

One final adjustment is usually made - we will split

$$4N(h/2) = 3N(h/2) + N(h/2)$$

to give us the form of the formula that is commonly used.

$$M = N(h/2) - \frac{1}{3}[N(h/2) - N(h)] + O(h^4).$$

General procedure

The procedure to obtain a fourth order accurate approximation using the two second order formulas is then easily accomplished through the following procedure:

- ① compute $N(h)$, for some given h
- ② compute $N(h/2)$
- ③ Combine the two using the formula

$$M = N(h/2) - \frac{1}{3}[N(h/2) - N(h)]$$

Practical Tips

The trick is then to find the right combination to cancel out the leading error term, when evaluating the equation at two different points, e.g. h and $h/2$.

There are advantages and disadvantages to Richardson extrapolation as for all numerical methods.

Advantages & Disadvantages

Advantages:

- it is simple and general so we can apply the technique to many different problems.
- applications in interpolation, numerical integration, and even differential equations.
- It also leads to formulas for higher-order approximations for derivatives, which are useful in certain applications.

Disadvantages

- the technique requires more points at which to evaluate the function and
- it makes an assumption that the higher-order derivatives are nicely bounded, which may or may not hold true.

Section 4

Summary

Summary

Some of the key takeaways

- Taylor series can be used to generate finite difference approximations to first derivatives, second derivatives, etc.
- We can also use Lagrange polynomials to generate similar formulas. (covered in book as well as in the optional lecture in Section 5).
- Richardson extrapolation is a simple and general approach that combines lower-order formulas to generate higher-order approximations (at additional function evaluation costs).
- ***Numerical differentiation is inherently unstable*** - but it's also (essentially) the only game in town. Care must be taken in choosing good step sizes.

Numerical Integration Basics

Math 131: Numerical Analysis

J.C. Meza

April 9, 2024

Section 1

Introduction

Motivation

What is the first thing you think of when you see:

$$\int_a^b f(x)dx \tag{1}$$

This leads us to the **general strategy**:

Approximate

$$I(f) = \int_a^b f(x)dx \approx \sum_{i=0}^n a_i f(x_i)$$

for some yet to be determined coefficients a_i .

We will call this ***numerical quadrature***.

Approach

To do this we will follow the same strategy we used for numerical differentiation, i.e. we will replace the function whose integral we seek with one whose integral can be more easily evaluated – an *interpolating polynomial*.

Our overall goal is to approximate the integral in Equation 1 by computing $\sum_{i=0}^n a_i f(x_i)$ through the following 3 steps:

- ① Approximate $f(x)$ by an interpolating polynomial
- ② Integrate the polynomial
- ③ Understand/analyze the truncation error

Intuition

Before we start, let's first develop some intuition on what we're doing. Consider Figure 1 in which we've plotted a generic function. A natural idea is to use the well-known trapezoidal rule to approximate the area under the curve.

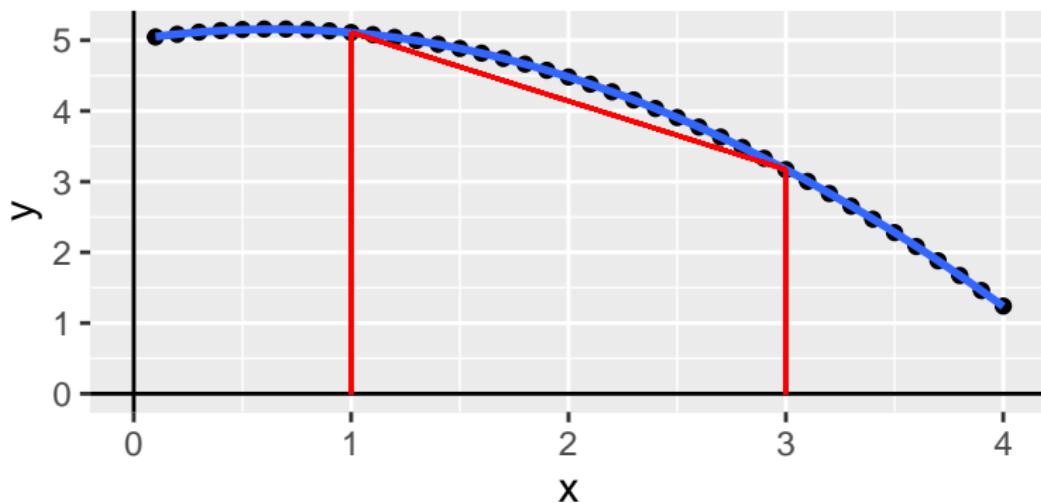


Figure 1: Trapezoidal Rule.

(cont.)

If we do this, it would make sense to approximate the integral as:

$$\int_1^3 f(x)dx \approx \frac{h}{2}[f(x_0) + f(x_1)], \quad x_0 = 1, x_1 = 3,$$

where h is defined as the interval width, i.e. $h = b - a = 3 - 1$.

Notice also, that in Figure 1 all that we did was to approximate the function by using a linear approximation using the two endpoints.

It is natural to conjecture for what type of functions would the trapezoidal rule be exact for? Can you guess?

Remarks

- In order to get a more accurate approximation, we could subdivide the total region into smaller trapezoids and sum over all of them.
- To make this more rigorous we will need to develop our framework and compute error estimates for our approximations.
- We will return to this idea in our lectures on **Composite Integration**

Section 2

Interpolating Polynomials

Interpolating Polynomials

Step 1. Write down our function as a Lagrange interpolating polynomial along with its truncation error.

Let

$$f(x) = \sum_{i=0}^n f(x_i)L_i(x) + \prod_{i=0}^n (x - x_i) \frac{f^{(n+1)}(\xi(x))}{(n+1)!}. \quad (2)$$

For convenience, let's denote

$$\Psi_n(x) = \prod_{i=0}^n (x - x_i).$$

We can then write Equation 2 as:

$$f(x) = \sum_{i=0}^n f(x_i)L_i(x) + \Psi_n(x) \frac{f^{(n+1)}(\xi(x))}{(n+1)!}.$$

Interpolating Polynomials

Step 2. Integrate the interpolating polynomial:

$$\int_a^b f(x) = \int_a^b \sum_{i=0}^n f(x_i)L_i(x) + \int_a^b \Psi_n(x) \frac{f^{(n+1)}(\xi(x))}{(n+1)!}.$$

Notice we can move the integral under the sum for the first term. Now (again for convenience) let's denote

$$a_i = \int_a^b L_i(x)dx, \quad i = 0, \dots, n. \tag{3}$$

(cont.)

Rearranging we get:

$$\int_a^b f(x) = \sum_{i=0}^n a_i f(x_i) + E(f),$$

where we denote the truncation error $E(f)$ by:

$$E(f) = \frac{1}{(n+1)!} \int_a^b \Psi_n(x) f^{(n+1)}(\xi(x)) dx. \quad (4)$$

Notice we are partway to our goal of having written down the integral in the form we wanted:

$$I(f) = \int_a^b f(x) dx \approx \sum_{i=0}^n a_i f(x_i).$$

Example: Linear Interpolating Polynomial

Specific Case: $n = 1$ (**Linear Interpolating Polynomial**)

Let's take the easiest case, $n = 1$, a linear Lagrange interpolating polynomial. To be consistent with our earlier notation we'll also let $a = x_0$ and $b = x_n = x_1$ for this case.

Recall, the first degree Lagrange polynomial takes the form:

$$P_1(x) = \frac{(x - x_1)}{x_0 - x_1} f(x_0) + \frac{(x - x_0)}{x_1 - x_0} f(x_1), \quad (5)$$

and the truncation error Equation 4 reduces to

$$E(f) = \frac{1}{2} \int_{x_0}^{x_1} (x - x_0)(x - x_1) f''(\xi(x)) dx. \quad (6)$$

Remaining Steps

We're now left with only two steps:

- ② Compute a_i for a specific interpolating polynomial, and
- ③ Understand/analyze the error function $E(f)$

Let's take these one at a time

Section 3

Computation of the Integrals

Compute the integrals

Let's consider the second step where we need to compute the integrals of Equation 5, i.e.

$$\begin{aligned}\int_{x_0}^{x_1} P_1(x) dx &= \int_{x_0}^{x_1} \frac{(x - x_1)}{x_0 - x_1} f(x_0) + \frac{(x - x_0)}{x_1 - x_0} f(x_1), \\ &= \left[\frac{(x - x_1)^2}{2(x_0 - x_1)} f(x_0) + \frac{(x - x_0)^2}{2(x_1 - x_0)} f(x_1) \right]_{x_0}^{x_1}.\end{aligned}$$

Notice that for the upper value x_1 the first term in the sum drops out, and likewise for the lower value x_0 the second term drops out, leaving only 2 terms.

(cont.)

Evaluating these two terms we get:

$$\begin{aligned}\int_{x_0}^{x_1} P_1(x) &= \left[\frac{(x_1 - x_0)^2}{2(x_1 - x_0)} f(x_1) - \frac{(x_0 - x_1)^2}{2(x_0 - x_1)} f(x_0) \right], \\ &= \left[\frac{(x_1 - x_0)}{2} f(x_1) - \frac{(x_0 - x_1)}{2} f(x_0) \right], \\ &= \left[\frac{(x_1 - x_0)}{2} f(x_1) + \frac{(x_1 - x_0)}{2} f(x_0) \right], \\ &= \left(\frac{x_1 - x_0}{2} \right) [f(x_1) + f(x_0)], \\ &= \frac{h}{2} [f(x_1) + f(x_0)],\end{aligned}$$

where the last line is due to the fact that $x_1 = x_0 + h$.

Trapezoidal Rule

This leads us to the well-known Trapezoidal Rule:

Trapezoidal Rule

$$\int_a^b f(x)dx \approx \frac{h}{2} [f(x_0) + f(x_1)].$$

Section 4

Error Analysis

Step 3 - Understand/Analyze the Truncation Error

Recall: the truncation error (Equation 6) was given by:

$$E(f) = \frac{1}{2} \int_{x_0}^{x_1} (x - x_0)(x - x_1) f''(\xi(x)) dx.$$

- It would be nice if we could take $f''(\xi(x))$ term outside the integral to simplify the integral.
- Let's first define $g(x) = (x - x_0)(x - x_1)$, and notice that $g(x)$ doesn't change sign on $[x_0, x_1]$.
- That means we can apply the Weighted Mean Value Theorem (WMVT) for integrals and pull the $f''(\xi(x))$ term outside of the integral.

Recall WMVTI

Weighted Mean Value Theorem for Integrals:

Suppose that $f \in C[a, b]$, the Riemann integral of g exists on $[a, b]$, and $g(x)$ does not change sign on $[a, b]$. Then there exists a number $c \in (a, b)$ such that

$$\int_a^b f(x)g(x)dx = f(c) \int_a^b g(x)dx$$

Using WMVTI

That simplifies $E(f)$ so that

$$E(f) = \frac{1}{2} f''(\xi) \int_{x_0}^{x_1} (x - x_0)(x - x_1) dx,$$

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

Integrating the quadratic

That just leaves us with integrating the quadratic inside the integral, which reduces to:

$$E(f) = \frac{1}{2} f''(\xi) \left[\frac{x^3}{3} - \frac{(x_1 + x_0)}{2} x^2 + x_0 x_1 x \right]_{x_0}^{x_1}. \quad (7)$$

To simplify the calculations, let's first do a change of variable, $x' = x - x_0$. Also recall that $x_1 = x_0 + h$.

With the change of variable, the limits reduce to $x_0 \rightarrow 0, x_1 \rightarrow h$, and

$$E(f) = \frac{1}{2} f''(\xi) \left[\frac{x^3}{3} - \frac{(x_1 + x_0)}{2} x^2 + x_0 x_1 x \right]_0^h. \quad (8)$$

(cont.)

As a result, the term in brackets in Equation 8 evaluates to:

$$\left[\frac{h^3}{3} - \frac{(h)h^2}{2} + 0 \right] = \frac{-h^3}{6}.$$

This gives us the following form for $E(f)$

$$E(f) = -\frac{h^3 f''(\xi)}{12}.$$

That takes care of Step 3 - Understand/Analyze the Truncation Error $E(f)$!

Trapezoidal Rule

Pulling it all together, this leads us to our desired result:

Trapezoidal Rule with Error Term

$$\int_a^b f(x)dx = \frac{h}{2} [f(x_0) + f(x_1)] - \frac{h^3}{12} f''(\xi),$$

Since the truncation error is given by $\frac{h^3}{12} f''(\xi)$ we expect that for **any function whose second derivative is identical to zero that the Trapezoidal Rule will be exact**, and in particular for any polynomial of degree 1 or less.

Section 5

Simpson's Rule

Note

Note

The derivation for Simpson's Rule follows the one for the Trapezoidal Rule, with some minor modifications. The following is included for completeness (and for practice), but you may also want to skip down to the final formula (Equation 13) and discussion of the major properties of Simpson's Rule.

Simpson's Rule

In a similar fashion to our approach for deriving the Trapezoid Rule, if we integrate the second-degree Lagrange polynomial, we can derive Simpson's Rule.

Conjecture

- ① For what degree polynomials will Simpson's rule be exact?
- ② What do you think the order of the truncation error will be for Simpson's method?

Setting up the integral of the interpolating polynomial.

As before, we will first approximate the integrand by an interpolating polynomial. In this case we will take 3 equally spaced points x_0, x_1, x_2 such that $x_0 = a, x_1 = x_0 + h, x_2 = b$, where $h = (b - a)/2$.

We assumed that $f(x)$ had as many derivatives as we needed. This time let's write the Taylor expansion for $f(x)$ about x_1 going out to the 4th derivative term. The reason for going out to the 4th derivative will become clear in a minute.

$$\begin{aligned} f(x) &= f(x_1) + f'(x_1)(x - x_1) + \frac{1}{2}f''(x_1)(x - x_1)^2 \\ &\quad + \frac{1}{6}f'''(x_1)(x - x_1)^3 + \frac{1}{24}f^{(4)}(\xi), \quad \xi \in [x_0, x_2] \end{aligned}$$

(cont.)

Now let's integrate this equation:

$$\begin{aligned}\int_{x_0}^{x_2} f(x) dx &= \left[f(x_1)(x - x_1) + \frac{1}{2} f'(x_1)(x - x_1)^2 \right. \\ &\quad \left. + \frac{1}{6} f''(x_1)(x - x_1)^3 + \frac{1}{24} f'''(x_1)(x - x_1)^4 \right]_{x_0}^{x_2} \\ &\quad + \frac{1}{24} \int_{x_0}^{x_2} f^{(4)}(\xi(x))(x - x_1)^4 dx.\end{aligned}\tag{9}$$

(cont.)

Let's consider the error term first and notice that it would be nice to take the $f^{(4)}(\xi(x))$ in the last term outside of the integral.

Using the previous trick, we note that $(x - x_1)^4$ doesn't change sign in the interval $[x_0, x_1]$, so we can again use the Weighted Mean Value Theorem for Integrals to pull the $f^{(4)}$ term out from inside the integral.

$$\begin{aligned} E(f) &= \frac{1}{24} \int_{x_0}^{x_2} f^{(4)}(\xi(x))(x - x_1)^4 dx, \\ &= \frac{f^{(4)}(\xi_1)}{24} \int_{x_0}^{x_2} (x - x_1)^4 dx, \\ &= \frac{f^{(4)}(\xi_1)}{120} (x - x_1)^5 \Big|_{x_0}^{x_2}. \end{aligned}$$

for some $\xi_1 \in [x_0, x_2]$.

(cont.)

Now we can use the fact that $h = x_2 - x_1 = x_1 - x_0$ to reduce the equation to:

$$E(f) = \frac{h^5 f^{(4)}(\xi_1)}{60}. \quad (10)$$

Evaluating the integral of the interpolating polynomial.

Our only remaining task is to evaluate the first term in Equation 9 to produce the approximation for $I(f)$. :

$$\left[f(x_1)(x - x_1) + \frac{1}{2}f'(x_1)(x - x_1)^2 + \frac{1}{6}f''(x_1)(x - x_1)^3 + \frac{1}{24}f'''(x_1)(x - x_1)^4 \right]_{x_0}^{x_2}$$

(cont.)

Recall that $x_2 - x_1 = x_1 - x_0 = h$ – which reduces our formula to:

$$\begin{aligned} & \left[f(x_1)h + \frac{1}{2}f'(x_1)h^2 + \frac{1}{6}f''(x_1)h^3 + \frac{1}{24}f'''(x_1)h^4 \right] - \\ & \left[f(x_1)(-h) + \frac{1}{2}f'(x_1)h^2 + \frac{1}{6}f''(x_1)(-h)^3 + \frac{1}{24}f'''(x_1)h^4 \right]. \end{aligned}$$

Nicely, the h^2 and h^4 terms cancel out, leaving us with

$$2hf(x_1) + \frac{h^3}{3}f''(x_1). \quad (11)$$

Tip

We could just as easily have noticed that (under the assumption of equally spaced nodes), $(x_2 - x_1)^k - (x_0 - x_1)^k = 0$ for even k and that $(x_2 - x_1)^k - (x_0 - x_1)^k = 2h^k$ for odd k .

(cont.)

Now remember our goal was to write our approximation in terms of only $f(x_i)$, meaning we should look for a way to replace the second derivative term in Equation 11.

For this task, we can take advantage of one last substitution, which is to use our finite difference approximation for the second derivative (derived in our previous lecture), i.e.

$$f''(x_1) = \frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2} - \frac{h^2}{12} f^{(4)}(\xi_2)$$

and not forgetting to include its own error term.

(cont.)

When we substitute the second derivative approximation into Equation 11 we get:

$$2hf(x_1) + \frac{h^3}{3} \left[\frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2} - \frac{h^2}{12} f^{(4)}(\xi_2) \right] \quad (12)$$

Combining Equation 12 with Equation 10 and simplifying terms we arrive at our final result, which is called Simpson's Rule:

Simpson's Rule

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90} f^{(4)}(\xi). \quad (13)$$

Exercise: combining the two derivative terms

I leave as an exercise how to combine the two $f^{(4)}$ terms from Equation 10 and Equation 12 into one, i.e. show that

$$-\frac{h^5}{60}f^{(4)}(\xi_1) - \frac{h^5}{36}f^{(4)}(\xi_2) = -\frac{h^5}{90}f^{(4)}(\xi), \quad \xi \in (a, b)$$

Remarks

Note

Other formulas use $\frac{b-a}{6}$ instead of $\frac{h}{3}$ in the definition of Simpson's Rule. Just remember that here, we defined $h = x_2 - x_1 = x_1 - x_0$, so $h = \frac{b-a}{2}$, as a result the two definitions are equivalent.

An important result is that instead of the expected $O(h^4)$ error term, we might have expected from going from a linear interpolant to a quadratic interpolant we have instead ***gained an additional order of accuracy*** in the error term!

Precision

Definition

Definition: The ***precision*** (also degree of accuracy) of a quadrature formula is defined as the largest positive integer n such that the quadrature formula is exact for x^k , for $k = 0, 1, \dots, n$.

- In the case of Simpson's rule, it is exact for any polynomial of degree 3 or less, hence the precision is 3.
- Similarly, the precision for the Trapezoid rule is 1. The easiest way to remember this is to take a look at the derivative in the error term and subtract one order.
- Please do not confuse this with the order of accuracy, which can be seen from the power in the h term!

Section 6

Summary

Summary

- Introduced the concepts of numerical integration
- Derived the Trapezoidal and Simpson's rule using interpolating polynomials
- Introduced the notion of precision of quadrature rules:
 - ▶ Trapezoid has precision 1
 - ▶ Simpson's rule has precision 3

Newton-Cotes

Math 131: Numerical Analysis

J.C. Meza

April 11, 2024

Section 1

Introduction

Recall

Approximate

$$I(f) = \int_a^b f(x)dx \approx \sum_{i=0}^n a_i f(x_i)$$

General approach was to approximate the integral by:

- ① Approximate $f(x)$ by an interpolating polynomial
- ② Integrate the polynomial

Recall (cont.)

Trapezoidal Rule

$$\int_a^b f(x)dx \approx \frac{h}{2} [f(x_0) + f(x_1)].$$

where $h = b - a, x_0 = a, x_1 = b$.

Simpson's rule

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]. \quad (1)$$

where $h = (b - a)/2, x_0 = 1, x_1 = (a + b)/2, x_2 = b$

Section 2

Exercise

Exercise

Exercise

Compute the value of

$$\int_0^1 e^x dx$$

using the Trapezoid and Simpson's Rule for:

- ① $a = 0, b = 1$
- ② $a = 0.9, b = 1$

What is the error in each case?

Trapezoid Rule

Trapezoid Rule:

$$\int_a^b f(x)dx = \frac{(b-a)}{2} \left[f(x_0) + f(x_1) \right],$$

Simpson's Rule:

$$\int_a^b f(x)dx = \frac{(b-a)}{6}[f(x_0) + 4f(x_1) + f(x_2)]$$

Remarks

- Notice that both formulas have a rather large error when we compute the integral over a “large” interval,
- Whereas when we considered a smaller interval, the error was in fact quite small.
- How can we get better estimates?

Section 3

Higher-Order Methods

Newton-Cotes

- The basic quadrature rules derived so far are generally good, but what if we wanted to have formulas with greater accuracy.
- The general approach we used still holds and leads to a family of quadrature formulas known as ***Newton-Cotes*** formulas.
- These are classified under either open or closed depending on whether the formulas include the end points or not.
 - ▶ ***Closed Newton-Cotes*** include the endpoints of closed interval $[a, b]$ as nodes.
 - ▶ ***Open Newton-Cotes*** do not include the endpoints.

In particular

To be specific, for a ***closed*** Newton-Cotes quadrature formula we would choose the node points x_i through the formula:

$$x_i = a + i \frac{b - a}{n - 1}, \quad i = 0, 1, \dots, n - 1. \quad (2)$$

For an ***open*** Newton-Cotes quadrature formula we would use the formula:

$$x_i = a + (i + 1) \frac{b - a}{n + 1}, \quad i = 0, 1, \dots, n - 1. \quad (3)$$

Example

Suppose, we choose $n = 5$ on the interval $[a,b] = [0,1]$.

Then Equation 2 (closed) would generate the points:

$$\begin{aligned}x_i &= a + i \cdot \frac{b - a}{n - 1}, \\&= 0 + i \frac{1}{4}, \\&= \frac{i}{4}, \quad i = 0, 1, \dots, 4,\end{aligned}$$

thereby yielding the set of nodes: $\{x\} = \{0, .25, .5, .75, 1.0\}$.

Example

Similarly Equation 3 (open) would generate the points:

$$\begin{aligned}x_i &= a + (i + 1) \cdot \frac{b - a}{n + 1}, \\&= 0 + (i + 1) \frac{1}{6}, \\&= \frac{i + 1}{6}, \quad i = 0, 1, \dots, 4,\end{aligned}$$

which generates the set of nodes:

$$\{x\} = \{ 1/6, 2/6, 3/6, 4/6, 5/6 \}.$$

Some previous examples

- One example of an Open Newton-Cotes is the midpoint rule

$$\int_a^b f(x)dx = 2hf(x_0) + \frac{h^3}{3}f''(\xi) \quad \xi \in (a, b)]$$

where x_0 is the midpoint between a and b .

- Likewise, both Trapezoidal and Simpson's rules can be categorized as Closed Newton-Cotes.

Other formulas

- There are many different formulas of both the Closed and Open variety all with corresponding error terms.
- All of them can be derived by the methods we've used for Trapezoid and Simpson's rule, so there is little to be gained by re-deriving them.
- Instead we will present them here because an interesting pattern arises that is worth knowing about:

Closed Newton-Cotes formulas:

$n = 2$ (Trapezoid)

$$I(f) = \frac{b-a}{2}[f(x_0) + f(x_1)] \quad (4)$$

$n = 3$ (Simpson's)

$$I(f) = \frac{b-a}{6}[f(x_0) + 4f(x_1) + f(x_2)] \quad (5)$$

$n = 4$ (Simpson's 3/8)

$$I(f) = \frac{b-a}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \quad (6)$$

$n = 5$ (Boole's rule)

$$I(f) = \frac{b-a}{90}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)] \quad (7)$$

Trip-Hazard - Notation

- The formulas here are written using $b - a$ versus h to make them easier to compare.
- However, you will see these formulas written in terms of h in many other places.
- You should be careful in understanding exactly what h represents as it often is taken to mean $h = (b - a)/(n - 1)$, $n \geq 1$, which is related to the number of node points used in the quadrature formula.

Higher-order formulas

- In theory, we could go as high as we wanted (and people have) in generating higher-order quadrature formulas, and of course with additional computational work.
- However, for large n the formulas can be shown to become numerically unstable ($n \geq 11$.) One can actually prove that formulas do not converge for all integrands that are analytic.
- In practice, we tend to only use low-order formulas since they can still give us good accuracy (especially over small intervals (see Exercise 2.1 below)).

Open Newton-Cotes formulas:

$n = 1$ (Midpoint)

$$I(f) = (b - a)f(x_0) \quad (8)$$

$n = 2$

$$I(f) = \frac{b - a}{2}[f(x_0) + f(x_1)] \quad (9)$$

$n = 3$

$$I(f) = \frac{b - a}{3}[2f(x_0) - f(x_1) + 2f(x_2)] \quad (10)$$

Similarly to the closed Newton-Cotes formulas, we could continue and derive higher-order formulas - with the same consequences.

Section 4

Error Estimates

Error Estimates

In both the closed and open Newton-Cotes cases, the formulas have error terms, which we have summarized in the table below, along with the precision of each:

Table 1: Summary of Error Terms for Newton-Cotes quadrature formulas

Name	N (npts)	Error	Precision
Trapezoid	2	$-\frac{(b-a)^3}{12} f^{(2)}(\xi)$	1
Simpson's	3	$-\frac{(b-a)^5}{2880} f^{(4)}(\xi)$	3
Simpson's 3/8	4	$-\frac{(b-a)^5}{6480} f^{(4)}(\xi)$	3
Boole	5	$-\frac{(b-a)^7}{1935360} f^{(6)}(\xi)$	5
Midpoint	1	$\frac{(b-a)^3}{24} f^{(2)}(\xi)$	1
	2	$\frac{(b-a)^3}{36} f^{(2)}(\xi)$	1
	3	$\frac{(b-a)^5}{23040} f^{(4)}(\xi)$	3

Remarks

Important

- An interesting feature of the quadrature formulas is that whenever N is odd then the precision of the formula = N .
- But when N is even then the precision is only $N - 1$.
- We lose one order in the precision whenever N is even! Or we could also say that we gain one order of precision for N odd.

Section 5

Summary

Summary

- A simple approach towards deriving basic quadrature rules is to replace the integrand with an interpolating polynomial on a chosen set of points and integrate the polynomial.
- Taylor's theorem yield error terms that provide us with estimates on how well the quadrature formula approximates the integral.
- The precision of a quadrature formula is the highest degree of the polynomial for which the formula is exact. When N is odd, the precision is also N ; but when N is even, the precision is $N - 1$.
- Higher-order formulas yield greater accuracy, but at greater computational work as well as a fundamental assumption that the higher-order derivatives are nicely behaved (i.e. bounded).
- Basic (low-order) formulas can be accurate, but usually require a small interval. This observation will prove useful in the next sections.

Composite Quadrature Rules

Math 131: Numerical Analysis

J.C. Meza

April 16, 2024

Section 1

Composite Integration

Composite Integration

- In practice we can't use Newton-Cotes over large intervals as it would require high degree polynomials, which would be unsuitable due to the highly oscillatory nature.
- Another disadvantage is that we would need to have equally spaced intervals, which are not suitable for many physical applications.

Idea

Instead of trying to approximate the integral accurately over the entire interval with one polynomial, break up the domain into smaller regions and use low-order polynomials on each of them.

Multiple Panels

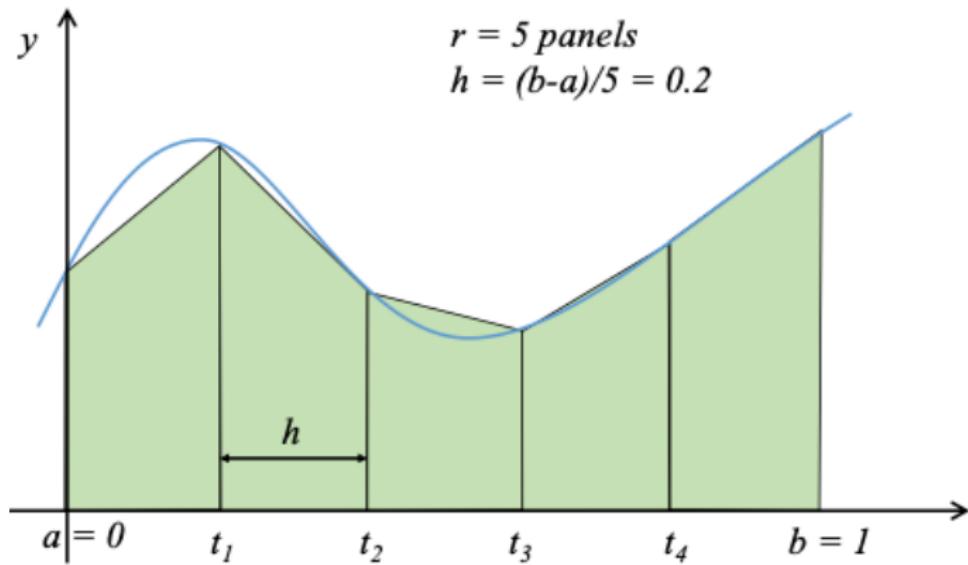


Figure 1: Approximating the integral by subdividing the region into multiple “panels” and using the trapezoid rule will lead to higher accuracy

Strategy

- The strategy is to use something simple like Trapezoid or Simpson's Rule on each of the subregions (often called **panels**), and then sum up the individual contributions to arrive at the solution to the original problem.
- There is nothing in this approach that directs us to use subregions of equal size, but for exposition, we will assume that they are for the time being. We will come back to this point in the next section on adaptive methods.
- In the general case, let's assume that we have sub-divided the interval $[a, b]$ into r subregions, each of equal length $h = \frac{b-a}{r}$. See Figure 1 for the case $r = 5$.

(cont.)

Our composite quadrature rule could then be written as:

$$\int_a^b f(x)dx = \sum_{i=1}^r \int_{t_{i-1}}^{t_i} f(x)dx,$$

where $t_i = a + ih$, $i = 0, \dots, r$, $h = (b - a)/r$.

All we need do now is to apply one of our earlier quadrature formulas to the integrals in each of the subintervals $[t_{i-1}, t_i]$.

Section 2

Example (Trapezoid Rule)

Example

Let's work out an example in the simple case of the Trapezoid Rule:

$$\begin{aligned}\int_a^b f(x)dx &\approx \sum_{i=1}^r \int_{t_{i-1}}^{t_i} f(x)dx, \\ &\approx \sum_{i=1}^r \frac{h}{2} [f(t_{i-1}) + f(t_i)], \\ &= \frac{h}{2} [(f(t_0) + f(t_1)) + (f(t_1) + f(t_2)) + \dots + (f(t_{r-1}) + f(t_r))], \\ &= \frac{h}{2} [f(t_0) + 2f(t_1) + 2f(t_2) + \dots + 2f(t_{r-1}) + f(t_r)], \\ &= \frac{h}{2} [f(a) + 2f(t_1) + 2f(t_2) + \dots + 2f(t_{r-1}) + f(b)].\end{aligned}$$

Composite Trapezoidal Rule

This leads to the Composite Trapezoid Rule:

Composite Trapezoidal Rule

$$\int_a^b f(x)dx = \frac{h}{2} \left[f(a) + 2 \sum_{i=1}^{r-1} f(t_i) + f(b) \right] - \frac{(b-a)}{12} h^2 f''(\mu), \quad (1)$$

where $h = (b-a)/r$; $t_i = a + ih$, $i = 0, 1, \dots, r$ and $\mu \in (a, b)$.

Caution

Notice that the error term loses one order in h . More on this in a minute.

Section 3

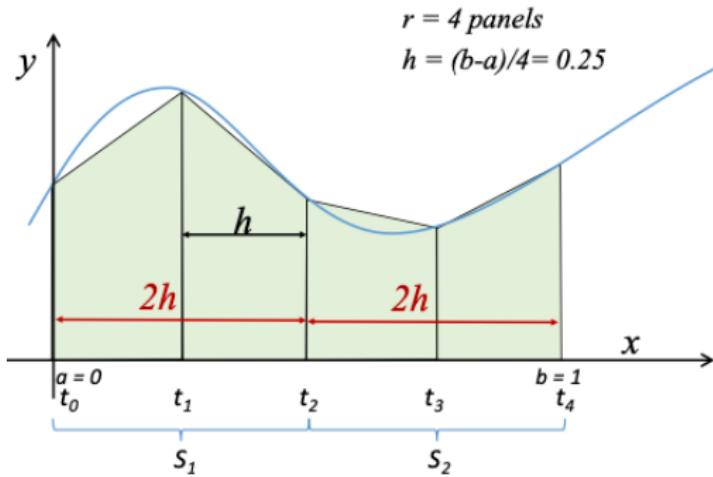
Example (Simpson's Rule)

Composite Simpson's Rule

- The derivation for Composite Simpson's rule isn't difficult, and mostly a matter of getting the right indices.
- One observation that is helpful in deriving the formula is that because of the need for 3 nodes in Simpson's rule we should consider the panels in pairs.
- For this reason, ***the number of panels for Simpson's rule must always be an even number.***

Simpson's rule - 4 panels

- Let's consider the simplest case with 4 panels, which we can group into two pairs, as depicted in the figure below.
- Here you should think of the first integration region spanning the first two panels so as to encompass the area in the interval $[t_0, t_2]$.
- Likewise, the second region will cover the interval $[t_2, t_4]$.



(cont.)

First recall Simpson's rule:

$$I(f) = \frac{b-a}{6} [f(x_0) + 4f(x_1) + f(x_2)],$$

which we can apply to each one of the two sub-intervals, $[t_0, t_2]$ and $[t_2, t_4]$.

Let's call the integrals over the two sub-intervals, S_1 and S_2 :

$$S_1 = \frac{t_2 - t_0}{6} [f(t_0) + 4f(t_1) + f(t_2)],$$

$$S_2 = \frac{t_4 - t_2}{6} [f(t_2) + 4f(t_3) + f(t_4)].$$

Note: $t_2 - t_0 = t_4 - t_2 = 2h$.

(cont.)

The total integral is then just the sum of these two. Here notice that each sub-interval is of length $2h$, allowing us to simplify the sums.

$$\begin{aligned} I(f) = S_1 + S_2 &= \frac{h}{3}[f(t_0) + 4f(t_1) + f(t_2)] + \\ &\quad \frac{h}{3}[f(t_2) + 4f(t_3) + f(t_4)]. \end{aligned}$$

Now, rearranging the terms we get:

$$\begin{aligned} I_{Simp}(f) &= \frac{h}{3}[f(t_0) + 4f(t_1) + 2f(t_2) + 4f(t_3) + f(t_4)], \\ &= \frac{h}{3}[f(t_0) + 4[f(t_1) + f(t_3)] + 2f(t_2) + f(t_4)]. \end{aligned}$$

Looking closely, one can start to see a general pattern. This leads to the Composite Simpson's Rule formula:

Composite Simpson's

Composite Simpson's Rule

$$\int_a^b f(x)dx = \frac{h}{3}[f(a) + 2 \sum_{i=1}^{r/2-1} f(t_{2i}) + 4 \sum_{i=1}^{r/2} f(t_{2i-1}) + f(b)] - \frac{(b-a)}{180} h^4 f^{(4)}(\mu). \quad (2)$$

where $h = (b-a)/r$; $t_i = a + ih$, $i = 0, 1, \dots, r$ and $\mu \in (a, b)$.

Caution

As with Composite Trapezoid the error term loses one order in h .

Composite Rules' Error Terms

! Important

An important consequence of using composite rules is that **the quadrature formulas lose one order of h in their approximations.**

- Even though each individual panel has the original truncation error, when we add up all the panels, the errors from each of the individual panels add up and we lose one order of magnitude.
- However, since h is smaller, the overall result is a win!

Sketch of proof:

$$\begin{aligned} E &= \sum_{i=1}^r -\frac{h^5}{90} f^{(4)}(\xi_i) = C \sum_{i=1}^r h^5, = C \cdot rh^5, \\ &= C \frac{(b-a)}{h} h^5 = C(b-a)h^4. \end{aligned}$$

Section 4

Stability and Error Analysis

Stability

- While we have a fairly good handle on the error analysis of quadrature formulas and the order of convergence, one question we haven't addressed yet is the stability of quadrature algorithms.
- First, we know that the truncation error is well-behaved and will go to zero at varying degrees depending on the degree of the interpolating polynomial we use.
- What caused problems in numerical differentiation was that roundoff error could increase dramatically leading to unstable algorithms.

Roundoff Error Analysis

- Let's proceed as before, by performing a floating point error analysis similar to the one we used for numerical differentiation.
- Recall, that we first assumed that a computation of a function value always incurs some roundoff error, in other words, we can write:

$$f(t_i) = \hat{f}(t_i) + e_i, \quad i = 0, 1, \dots, n,$$

where \hat{f} is the floating point representation and e_i is the roundoff error incurred when computing the function value.

- As before, we will assume that the errors are uniformly bounded:

$$e_i < \tau, \quad \tau > 0, \quad \forall i$$

(cont.)

If we substitute into the Composite Simpson's rule we can write the error as:

$$\begin{aligned}|e(h)| &= \left| \frac{h}{3} \left(e_0 + 2 \sum_{i=1}^{r/2-1} e_{2i} + 4 \sum_{i=1}^{r/2} e_{2i-1} + e_r \right) \right|, \\ &\leq \frac{h}{3} \left(\tau + 2\left(\frac{r}{2} - 1\right)\tau + 4\left(\frac{r}{2}\right)\tau + \tau \right), \\ &\leq \frac{h}{3} \left(\tau + r\tau - 2\tau + 2r\tau + \tau \right), \\ &\leq \frac{h}{3} (3r\tau) = rh\tau.\end{aligned}$$

(cont.)

Finally, let's remember that $h = (b - a)/r$, which means that the error is bounded by:

$$|e(h)| \leq (b - a) \tau,$$

which is independent of both h and r . Of course, the bigger the interval the bigger the bound.

We can do the same analysis for any of the open or closed Newton-Cotes quadrature formulas.

Important

Unlike numerical differentiation, ***Simpson's rule is stable!***

Section 5

Choosing good values of h

Choosing h to achieve given tolerance

Exercise

Determine values of h that will ensure an approximation error of < 0.00002 when approximating

$$\int_0^{\pi} \sin x dx$$

using

- ① Composite Trapezoidal Rule
- ② Composite Simpson's Rule

Solution

Composite Trapezoid. Our starting point is the formula for the truncation error. For composite Trapezoid we use Equation 1:

$$E(f) = -\frac{b-a}{12} h^2 f''(\mu), \quad \mu \in [a, b]$$

We would like this term to be less than the tolerance $\epsilon = 2 \cdot 10^{-5}$

$$\begin{aligned}|E(f)| &= \left| \frac{\pi}{12} h^2 \sin(\mu) \right| \\&= \frac{\pi}{12} h^2 |\sin(\mu)| \\&\leq \frac{\pi h^2}{12} < 2 \cdot 10^{-5}\end{aligned}$$

with the last inequality because $|\sin(x)| \leq 1$.

(cont.)

Solving for h is then an easy matter:

$$\begin{aligned}\frac{\pi h^2}{12} &< 2 \cdot 10^{-5}, \\ h^2 &< \frac{24 \cdot 10^{-5}}{\pi}, \\ h &< \sqrt{\frac{24 \cdot 10^{-5}}{\pi}}. \\ \implies h &\approx 0.00874.\end{aligned}$$

To achieve the desired accuracy, would then require $r = (b - a)/h$ panels or $r \approx (\pi - 0)/0.00874 \approx 360$ panels.

Solution (cont.)

Composite Simpson. As before, our starting point is the formula for the truncation error. For composite Simpson we use Equation 2:

$$E(f) = -\frac{b-a}{180} h^4 f^{(4)}(\mu).$$

We would like this term to be less than the tolerance $\epsilon = 2 \cdot 10^{-5}$

$$\begin{aligned}|E(f)| &= \left| \frac{\pi}{12} h^4 \sin(\mu) \right|, \\&= \frac{\pi}{12} h^4 |\sin(\mu)|, \\&\leq \frac{\pi h^4}{180} < 2 \cdot 10^{-5},\end{aligned}$$

with the last inequality because $|\sin(x)| \leq 1$.

(cont.)

Solving for h :

$$\begin{aligned}\frac{\pi h^4}{180} &< 2 \cdot 10^{-5}, \\ h^4 &< \frac{360 \cdot 10^{-5}}{\pi}, \\ h &< \sqrt[4]{\frac{360 \cdot 10^{-5}}{\pi}}. \\ \implies h &\approx 0.18399\end{aligned}$$

To achieve the desired accuracy, would then require $r = (b - a)/h$ panels or $r \approx 18$ panels.

For this example, it is clear that Composite Simpson is a much better choice than Composite Trapezoid.

Section 6

Summary

Summary

- There are several choices to be made when considering which quadrature method to use and what size of h to choose.
- Deciding between a lower order method versus a higher order method can also be tricky.
- The error term will depend on both the size of h as well as the magnitude of the highest derivative required.
- Choosing h will require (as in the case of numerical differentiation) a balance between truncation error and roundoff error.

Practical Tips - Composite Quadrature

Choosing a quadrature method and h

- ① The smaller h is, the greater the accuracy for all methods; however that also implies we have more panels and hence a higher computational load.
- ② If the function that is to be integrated is smooth then higher order methods are likely to work well.
- ③ However, if the function is rapidly changing, then the higher derivatives will likely be large and one might want to consider lower order methods, especially if the integral interval is small.

Adaptive Quadrature

Math 131: Numerical Analysis

J.C. Meza

April 18, 2024

Section 1

Introduction

Adaptive Quadrature

- Composite quadrature formulas can be quite effective as we discussed in the last section.
- There is one drawback however - so far we have only used a uniform spacing for the nodes.
- We did this mainly to simplify the analysis, and to highlight the main ideas. However, there was no underlying need to do so.
- In fact, there are many situations where it is clear that a uniform spacing might not be optimal.

Example

Consider for example:

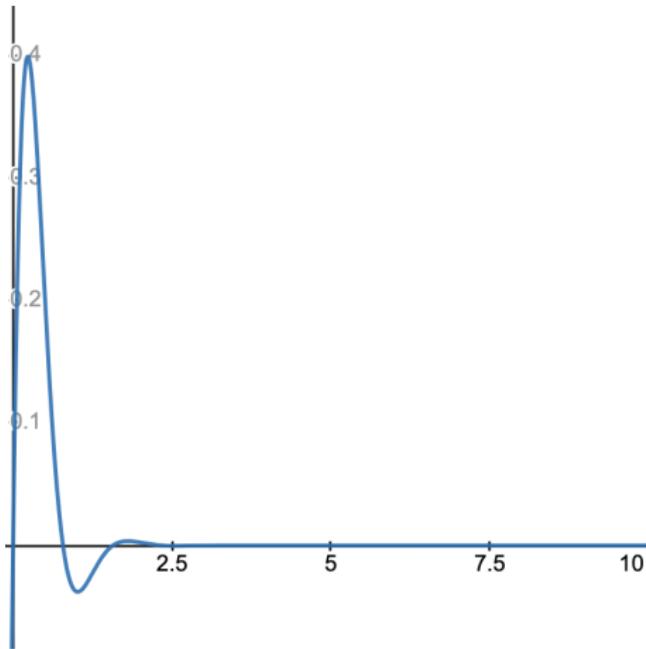


Figure 1: $f(x) = e^{-3x} \sin(4x)$, $x \in [0, 10]$

Uniform spacing

- If we attempt to use a **uniform spacing** that tries to approximate the integral of this function, we will be caught between two competing interests.
- To capture the behavior of the function towards the right end of the interval a spacing of $h = 0.5$ or even $h = 1.0$ would likely be adequate.
- However, if we want to capture the behavior of the function towards the left end of the interval, then it looks likely that we would need to have a spacing of h that is much smaller.
- And if we choose the smaller h then we will be committed to doing additional computational work that is not needed on the right end of the interval.

Uniform spacing in higher dimensions

- This problem can be exacerbated when working in two or three dimensions, where the computational work could increase dramatically, if we have to choose a uniform h in all of the dimensions.
- In the above example, suppose we had to choose one h for the entire region. The table below depicts the size of the problem in terms of the number of “cells” one would have to compute over the interval $[0, 10]$ with a uniform grid, and a nonuniform grid where the majority of the points are chosen in a small subinterval, say $x \in [0, 2]$.
- The difference isn’t particularly noteworthy in one dimension but when you reach a three-dimensional problem, there is an additional factor of 1000 to consider when using a uniform grid.

Uniform spacing in higher dimensions

Type	h	N	N^2	N^3
Uniform	0.1	100	$\approx 10^4$	$\approx 10^6$
	0.01	1000	$\approx 10^6$	$\approx 10^9$
Nonuniform	0.1	≈ 20	≈ 400	≈ 8000
	0.01	≈ 110	$\approx 10^4$	$\approx 10^6$

Grid on $[0, 10]$

Idea

- The solution is obvious, which is to find a value of h that is adapted to what the function is doing over a particular subinterval.
- But the big question is how do we know this without evaluating the function at many different points and
- How do we choose a good value of h that gives us good accuracy without also increasing the computational workload too much.

Idea

If we could predict the variation in the function, then we could choose a smaller h in only those regions that need it to attain the accuracy we want! Our strategy will be to leverage our **error analysis** to help us predict the variation.

Section 2

Romberg Integration

Composite Trapezoid Error Analysis

Let's consider the Composite Trapezoid Rule first. Recall that we can write the quadrature formula as:

$$I(f) = \int_a^b f(x)dx = \frac{h}{2} \left[f(a) + 2 \sum_{i=1}^{r-1} f(t_i) + f(b) \right] - \frac{(b-a)}{12} h^2 f''(\mu),$$

As before with Richardson extrapolation, let's break this approximation down into 2 parts. This approach is also called **Romberg integration**.

(cont.)

Suppose we write the approximation to the integral as:

$$R_1 = \frac{h}{2} \left[f(a) + 2f(a+h) + \dots + 2f(b-h) + f(b) \right] + Kh^2,$$

Now let's suppose we cut h in half and write the Composite Trapezoid rule again:

$$R_2 = \frac{h}{4} \left[f(a) + 2f(a+h/2) + \dots + 2f(b-h/2) + f(b) \right] + K \left(\frac{h}{2} \right)^2,$$

(cont.)

Let's now consider the error in each of these formulas:

$$I(f) - R_1 \approx Kh^2,$$

$$I(f) - R_2 \approx K\left(\frac{h}{2}\right)^2 = \frac{1}{4}Kh^2.$$

Caution

The next step requires us to make an assumption, namely that the terms that constitute the constant K in both of the above terms are approximately equal.

This should be true if the fourth derivative terms in each of the constants are comparable, which seems reasonable since they are from the same function and over similar intervals:

$$f^{(4)}(\mu_1) \approx f^{(4)}(\mu_2).$$

(cont.)

Substituting the first equation into the second we get:

$$I(f) - R_2 \approx \frac{1}{4}[I(f) - R_1]. \quad (1)$$

Let's now consider the error for R_1 :

$$\begin{aligned} I(f) - R_1 &= (I(f) - R_2) + (R_2 - R_1), \\ &\approx \frac{1}{4}[I(f) - R_1] + (R_2 - R_1) \end{aligned} \quad (2)$$

$$\implies I(f) - R_1 \approx \frac{4}{3}(R_2 - R_1).$$

a posteriori error estimates

Finally, we can combine Equation 1 and Equation 2 to get:

$$\begin{aligned} I(f) - R_2 &\approx \frac{1}{4}[I(f) - R_1] \\ &\approx \frac{1}{4}\left[\frac{4}{3}(R_2 - R_1)\right] \end{aligned} \tag{3}$$
$$\implies I(f) - R_2 \approx \frac{1}{3}\left[(R_2 - R_1)\right]$$

The important thing to note is that once everything on the right hand sides (specifically R_1, R_2), have been computed, we can generate an estimate for the error in both quadrature approximations. These types of computations are known as ***a posteriori error*** estimates.

Summarizing Composite Trapezoid *a posteriori* estimates

Composite Trapezoid *a posteriori* error estimates

$$I(f) - R_1 \approx \frac{4}{3}(R_2 - R_1),$$

$$I(f) - R_2 \approx \frac{1}{3}(R_2 - R_1).$$

We can interpret this to mean that the error from the quadrature approximation for R_2 (with $h/2$) should be about $1/3$ of the difference between the two Trapezoid rule approximations at h and $h/2$.

Composite Simpson's *a posteriori* estimates

In a similar manner we can produce *a posteriori* error estimates for composite Simpson's rule and write:

Simpson *a posteriori* error estimates

$$I(f) - S_1 \approx \frac{16}{15} (S_2 - S_1),$$

$$I(f) - S_2 \approx \frac{1}{15} (S_2 - S_1).$$

We can interpret this to mean that the error from the quadrature approximation with $h/2$ (S_2) should be about $1/15$ of the difference between the two Simpson's rule approximations at h and $h/2$.

Strategy for adaptively choosing h

- These observations can lead us to develop a strategy for deciding when to subdivide a panel and when to stop.
- Specifically, suppose we want the error to be less than a certain tolerance, ϵ , for Composite Simpson's Rule.
- Then we can ask whether for a given panel

$$\frac{1}{15} (S_2 - S_1) < \epsilon.$$

If this is true, then we can stop for that given panel. If however, the difference doesn't satisfy the tolerance, that is an indication that we should subdivide the region in half again.

General Algorithm

A general algorithm might look something like:

- ① Initialize by computing Simpson's on $[a, b]$, i.e. S_1
- ② For $i = 1, 2, \dots$
 - ① subdivide the interval into 2 sub-regions and compute S_{i+1} by applying Simpson on each subinterval
 - ② If $|S_{i+1} - S_i| < 15\epsilon$
 - ① converged
 - ② else repeat.

Section 3

Summary

Summary

- This is just a brief introduction into adaptive quadrature.
- The main point to remember is that using our error analysis can help us choose better values of h so that we get better approximations to our integrals.
- There are many other techniques one can use to improve both the accuracy and the efficiency of these methods.
- An interested reader, can find many references under topics such as adaptive mesh refinement.

Euler's Method

Math 131: Numerical Analysis

J.C. Meza

April 18, 2024

Section 1

Introduction

Initial-Value Problems for ODEs

The areas we will cover include:

- ① General statement of Initial Value Problems, systems of ODEs, etc.
- ② Euler's Method including a simple error analysis
- ③ Higher-order methods
- ④ Multi-step methods

Roadmap

Let's first start with a roadmap for the lectures to follow. We will:

- be mostly concerned with introducing methods for the solution of IVPs and providing advantages and disadvantages of them
- not discuss problems that are ill-posed, stiff ODE's, or have other structure
- Our reason for this particular focus is that there is a lot of good software available for these problems, so you may never need to actually implement one of these methods.
- Nevertheless, it will be important to know the differences between the methods, what types of problems they can be used on, and the pros and cons of each method.

Some Applications

- Weather/Climate Modeling (primitive equations)
- Chemical reactions (combustion)
- Molecular dynamics simulations

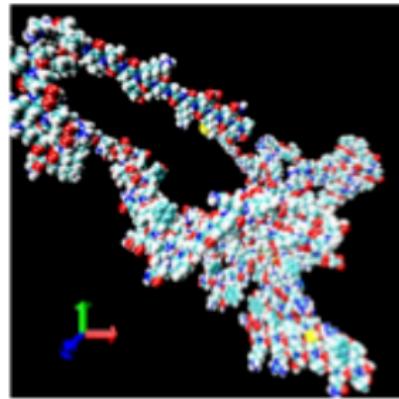
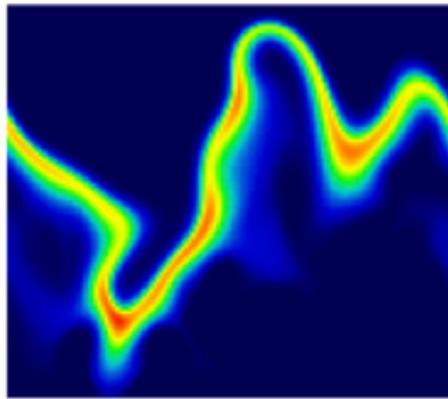


Figure 1a) Jet Flame, b) Combustion simulation, c) protein folding

Figure 2

Euler's Method

Existence and Uniqueness

The material on the existence and uniqueness of the IVP is in the supplemental materials section including:

- ① Concepts of Lipschitz continuity and convex sets
- ② Fundamental Existence and Uniqueness of solutions to the IVP
- ③ Concept of ***well-posed*** problems

Section 2

IVP

Initial Value Problem

The scalar ***initial-value problem*** (IVP) has the form:

$$y' = \frac{dy}{dt} = f(t, y(t)), \quad a \leq t \leq b, \quad y(a) = \alpha. \quad (1)$$

In the general case, we would consider a *system* of ODEs, i.e.

$$y' = \frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha,$$

where

$$y' = \begin{bmatrix} y'_1 \\ \vdots \\ y'_n \end{bmatrix} \quad f(t, y) = \begin{bmatrix} f_1(t, y_1, \dots, y_n) \\ \vdots \\ f_n(t, y_1, \dots, y_n) \end{bmatrix} \quad (2)$$

We will keep it simple for now and assume we have an IVP of the form given by Equation 1.

Higher-order ODEs and Systems of ODEs

- Many science and engineering problems are in the form of higher-order ODES.
- Higher-order ODEs can be reduced to a **system** of ODEs
- It can be shown that a general ***n-th*** order ODE:

$$y^{(n)}(t) = f(t, y(t), y'(t), \dots, y^{(n-1)}(t)).$$

can be written in the form of a system of ODEs.

Example 1: Second order ODE

Consider the simple second order IVP given by:

$$y'' + ay' + by = f$$

If we let:

$$y_1 = y$$

$$y_2 = y'$$

then we can rewrite the IVP as two first order ODEs in the form of
Equation 2:

$$y'_1 = y_2$$

$$y'_2 = f - (ay_2 + by_1).$$

Example 2: Motion of a pendulum

Consider the second order IVP describing the motion of a pendulum:

$$\theta''(t) = -g \sin(\theta(t)),$$

where θ is the angle between the pendulum and the negative vertical axis, g gravity, t is time. If we let:

$$y_1(t) = \theta(t),$$

$$y_2(t) = \theta'(t)$$

then we can rewrite the IVP as a system of two first order ODEs:

$$y'_1 = y_2$$

$$y'_2 = -g \sin(y_1).$$

Autonomous Systems

Autonomous Systems

An IVP where the function f does not depend explicitly on t is said to be in ***autonomous form***, i.e.

$$y' = f(y)$$

- Many software packages for the solution of IVPs assume that the function is given in this form.
- This is generally achieved through the addition of an additional equation of the form $t' = 1$.

Section 3

Euler's Method

Euler's Method

We will now present the simplest method for solving an IVP.

Note

For the remainder of the discussion we will assume that our IVP is well-posed (details to follow).

Recall we are looking for solutions to the IVP:

$$\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha.$$

As in previous lectures, we will take an approach for numerically solving this problem by approximating it on a discrete grid.

Grids and mesh points

In this case, the points are referred to as ***mesh points*** and are typically of the form:

$$t_i = a + ih, \quad i = 0, 1, 2, \dots, N,$$

The difference between two consecutive points $t_{i+1} - t_i$ is called the step size and is given by:

$$t_{i+1} - t_i = \frac{(t_N - t_0)}{N} = h.$$

Terminology

Many other references use Δt in reference to the time evolution of the IVP, in which case the step size is called the ***time step***.

Derivation of Euler's method

In order to derive Euler's method we can either make use of Taylor's Theorem or just use the numerical approximation for the first derivative that we used previously:

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi_i) \quad i = 0, 1, 2, \dots, N-1,$$

where $h = t_{i+1} - t_i$, and $\xi_i \in [t_i, t_{i+1}]$.

Now remember that y' satisfies the IVP. As a result, we can rewrite the above equation as:

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2}y''(\xi_i) \tag{3}$$

(cont.)

Taking the first 2 terms on the right hand side of this equation as our approximation to $y(t_{i+1})$ leads us to propose the following algorithm:

Euler's Method

$$\begin{aligned}y_0 &= \alpha \\y_{i+1} &= y_i + hf(t_i, y_i), \quad i = 0, 1, \dots, N - 1\end{aligned}\tag{4}$$

Here $y_i \approx y(t_i)$ (approximation to the true solution).

This type of equation is known as a ***difference equation***. You can think of it as being derived from a forward difference approximation to the derivative. Another interpretation is that it is the discretization (in time) of the continuous differential equation.

Section 4

Example

Example

Example

Solve the IVP given by:

$$y' = f(t, y) = y - t^2 + 1 \quad y(0) = 0.5 \quad 0 \leq t \leq 2,$$

with $h = 0.5$.

(cont.)

I find it easier before I start, to write down a table with some of the important variables, where I can keep track of the steps. Something like the following is helpful:

Table 1: Euler Computations

i	t_i	y_i
0	$t_0 = a$	$y_0 = y(a)$
1	$t_1 = a + h$	$y_1 = \dots$
2	$t_2 = a + 2h$	
3		
...
N	$t_N = b$	

(cont.)

I then fill in the initial conditions in the first row and as I compute subsequent y_i I fill in the table with those values.

Table 2: Euler Computations

i	t_i	y_i
0	0	0.5
1	0.5	
2	1.0	
3	1.5	
N	2.0	

Example

- Solve the IVP with a step size of $h = 0.5$ using the built-in ODE solver with method chosen to be “euler”.
- Note that this IVP, $f(t, y) = y - t^2 + 1$, has an exact solution given by $(t + 1)^2 - 0.5 \exp(t)$
- We can also compute the true solution and the associated error generated by Euler’s method.

Exact solution versus Euler's method solution

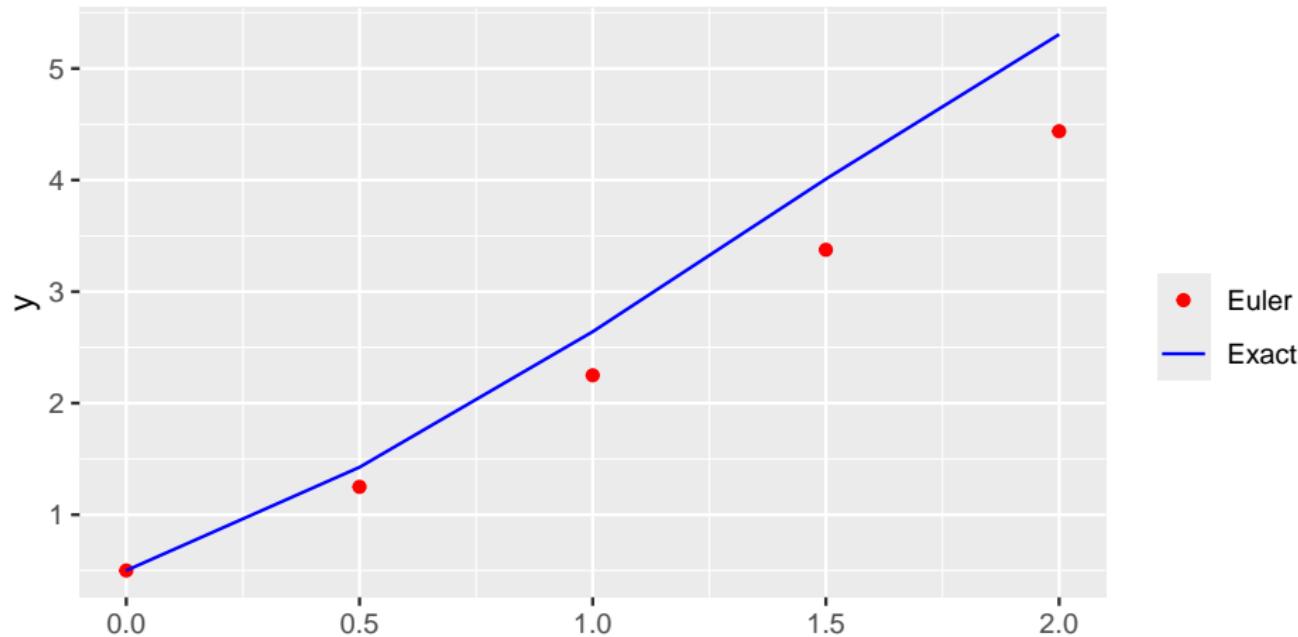
- The table below summarizes the output from the ode solver and compares it to the exact solution.
- What do you notice about the error, especially as time increases?

	time	ysol	yexact	yerr
1	0.0	0.5000	0.500000	0.0000000
2	0.5	1.2500	1.425639	0.1756394
3	1.0	2.2500	2.640859	0.3908591
4	1.5	3.3750	4.009155	0.6341555
5	2.0	4.4375	5.305472	0.8679720

Plots

Let's plot the solution from Euler alongside the exact solution

Euler's Method for $f(t, y) = y - t^2 + 1$



Experimenting with ICs

Let's explore what happens to the solutions by solving the IVP with several different initial conditions (see the plot below).

Euler's Method for $f(t, y) = y - t^2 + 1$

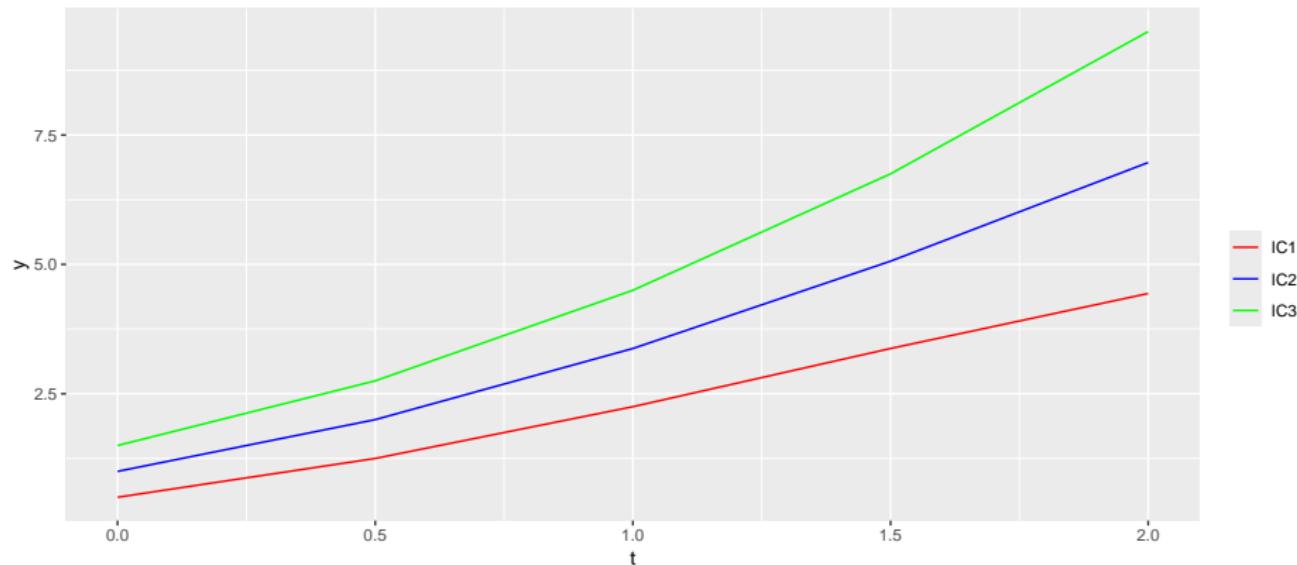


Figure 4: IVP with different ICs

Observations

- One immediate observation is that the IVP generates a family of solutions that can be parameterized by the specific initial condition chosen.
- In this case, also notice that the curves do not converge to a single line.
- What implications would this have on the solutions generated by Euler's method?
- We'll have more to say in later lectures on what is happening and what we can do to help us attain more accurate solutions.

Section 5

Exercise

Exercise

Exercise

Solve the IVP given by:

$$y' = f(t, y) = 1 + (t - y)^2 \quad y(2) = 1.0 \quad 2 \leq t \leq 3$$

with $h = 0.5$ Fill out the table below with your calculations.

Solution

Solution:

Table 3: In class exercise

i	t_i	y_i
0		
1		
2		
3		
...		
N		

Section 6

Backward Euler

Backward Euler

What if we had used a backward difference formula to approximate the derivative of y ? In other words:

$$y'(t_i) = \frac{y(t_i) - y(t_{i-1})}{h} = f(t_i, y(t_i))$$

Following the same procedure as before we would have:

$$y(t_i) = y(t_{i-1}) + h f(t_i, y(t_i))$$

Since we really want to compute the approximation at the next time step, let's shift the index by 1:

$$y(t_{i+1}) = y(t_i) + h f(t_{i+1}, y(t_{i+1}))$$

Backward Euler

Letting the approximation to the true solution be denoted by $y_i \approx y(t_i)$, leads to what is known as **Backward Euler**:

Backward Euler

$$y_0 = \alpha$$
$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}), \quad i = 0, 1, \dots, N - 1$$

Notice that y_{i+1} appears on both the right and left hand sides of this equation. This will therefore require an iterative method to compute the solution at the next time step.

Explicit vs. Implicit Methods

Explicit/Implicit Methods

- (*Forward*) Euler's Method is an example of a type of method called an **explicit** method, because everything we need to compute a quantity at time t_{i+1} is given by known quantities at the previous time step t_i .
- Backward Euler on the other hand is an example of an **implicit** method since we have y_{i+1} on both sides of the equation.
- There are advantages and disadvantages to both approaches.
- In general, one can take longer timesteps with an implicit method. On the other hand, an implicit method will generally require the solution of a system of equations.

Section 7

Summary

Key Points

- Initial Value Problems arise in many scientific and engineering problems
- Euler's method can be used to solve the IVP by using a forward difference approximation to the derivative of y .
- Forward Euler is easy to implement and relatively cheap computationally.
- Using the backward difference approximation yields a similar method, but requires having to solve the difference equation *implicitly*.

Error Analysis for Euler's Method

Math 131: Numerical Analysis

J.C. Meza

April 23, 2024

Section 1

Introduction

Introduction

In order to discuss the error in Euler's method as well as its convergence, we will need to define a few terms.

First let's recall that we are seeking to approximate the solution to the IVP at a set of discrete points in time or ***mesh points*** typically of the form:

$$t_i = a + ih, \quad i = 0, 1, 2, \dots, N.$$

We start with a few definitions.

Definitions

The ***difference method***

$$y_0 = \alpha$$

$$y_{i+1} = y_i + h\phi(t_i, y_i), \quad i = 0, 1, \dots, N - 1$$

has ***local truncation error***

$$d_{i+1} = \frac{y_{i+1} - (y_i + h\phi(t_i, y_i))}{h}, \quad (1)$$

where y_i denotes the solution of the difference equation at t_i , and $\phi(t, y)$ is a given function.

We call d_{i+1} ***local*** because it measures the accuracy of the solution at a specific point (step) in time. It is the error in 1 step if all previous values were exact and there's no roundoff error.

Definitions (cont.)

We say that a method is ***consistent (or accurate) of order q*** if q is the lowest positive integer such that

$$\max_i |d_i| = O(h^q). \quad (2)$$

Finally, the ***global error*** is defined as

$$e_i = y(t_i) - y_i \quad i = 0, 1, \dots N, \quad (3)$$

where $y(t_i)$ is the true solution at time, t_i .

Gloabl error can be $>$ sum of local errors if ODE is unstable, or it could be $<$ sum of local errors if ODE is stable.

Example

Example

Show that Euler's method has local truncation error of $O(h)$.

For Euler's method $\phi(t_i, y_i) = f(t_i, y_i)$. As such we can write the local truncation error as:

$$\begin{aligned} d_{i+1} &= \frac{y_{i+1} - (y_i + hf(t_i, y_i))}{h} = \frac{\frac{h^2}{2}y''(\xi_i)}{h}, \\ &= \frac{h}{2}y''(\xi_i), \quad \xi_i \in (t_i, t_{i+1}), \end{aligned} \tag{4}$$

where the second equation is as a result of Taylor's theorem.

(cont.)

If we assume that the second derivative of y is bounded by some constant M , then we have:

$$|d_{i+1}| \leq \frac{M}{2}h \implies d_{i+1} = O(h)$$

By Equation 2 Euler's method is ***first order accurate***, i.e $q = 1$.

Remarks

- This example shows that the local truncation error for Euler's method is $O(h)$.
- Notice also that the error will depend on 1) the ODE, and 2) the step size.
- By the same argument, it is easy to see that Backward Euler is also first order accurate.

We now state a theorem that provides error bounds on the approximations generated by Euler's method and requirements for convergence.

Section 2

Convergence Theorem

Convergence and Global Error Estimates

Theorem (Euler Method Convergence.)

Suppose $f(t, y)$ is continuous and Lipschitz continuous in y , with constant L on a region $D = \{(t, y) | a \leq t \leq b, -\infty < y < \infty\}$. Let y_1, \dots, y_N be approximations generated by Euler's method for some integer $N > 0$.

Then Euler's method converges and its global error decreases linearly in h .

Furthermore if a constant M exists with $|y''| \leq M$, $\forall t \in [a, b]$, then the global error satisfies

$$|e_i| \leq \frac{hM}{2L} [e^{L(t_i-a)} - 1] \quad \forall i = 0, 1, 2, \dots, N \quad (5)$$

Interpretation

Let's take a quick look to see what the error bound is saying.

Note that:

- The bound is exactly zero for $t_i = a$, which makes sense since $y_0 = y(a)$, the given initial condition.
- The last term depends on the Lipschitz constant, as well as the term $t_i - a$. But $t_i - a$ is bounded by $b - a$, so the entire term is also bounded.
- The bound depends on both the Lipschitz constant as well as the bound, M , on the second derivative of $y(t)$.
- ***the error bound is linear in h .***

Key Points

- The good news is that we can bound the error at each time step.
- Nonetheless it is clear that the error bound will increase at each time step t_i .
- Our hope is that by choosing a small enough $h(\Delta t)$ we can compensate for the other terms and make the error bound small enough to generate an accurate approximation to $y(t)$.

Remark

*Note that the theorem requires a bound on the second derivative. We can sometimes use knowledge of the partial derivatives to obtain an error bound. The important aspect is that the **error bounds are linear in h** . Not surprisingly, as the number of computations grow so will the roundoff error.*

Section 3

Roundoff Error Analysis (Highlights)

Roundoff Error Analysis

As in the numerical differentiation lectures, we can derive a bound on the roundoff error:

$$|y(t_i) - y_i| \leq \frac{1}{L} \left(\frac{hM}{2} + \frac{\delta}{h} \right) [e^{L(t_i-a)} - 1] + |\delta_0| e^{L(t_i-a)},$$

where δ, δ_0 are constants representing the amount of roundoff error incurred at each time step.

Notice that we have the same situation as before with numerical differentiation – one of the terms is going to 0 while the second term blows up as $h \rightarrow 0$.

$$\text{error} \approx \left(\frac{hM}{2} + \frac{\delta}{h} \right).$$

Roundoff Error Analysis (Optimal h)

- A similar type of calculation as in the case of numerical differentiation, yields an optimal h :

$$h = \sqrt{\frac{2\delta}{M}}$$

that will depend on both δ and M .

- If we assume that $\delta \approx \epsilon$, i.e machine epsilon, then depending on the value of M , this implies h should be roughly the square root of machine epsilon.
- For IVPs, the more important question is **stability**, which will depend on choosing an appropriate h .

Stability

- While we can usually get a more accurate solution as h decreases, that means we will also necessarily increase the computational cost.
- If we increase h , however we run against another problem - one of stability
- It can be shown that for forward Euler, there is an upper bound for h determined by a condition known as ***absolute stability***.
- This requirement does not hold for Backward Euler!
- Unfortunately, we don't have time to cover that topic in detail here.

Brief Introduction to Stability

Consider the test equation

$$y' = \lambda y, y(0) = y_0$$

for which the solution is given by $y(t) = e^{\lambda t}y(0)$.

- for $\lambda > 0$ the exact solution increases, for $\lambda < 0$ the exact solution decreases.
- For Euler's method it would make sense that $|y_{i+1}| \leq |y_i|$
- Given that $y_{i+1} = (1 + h\lambda)y_i$ it can be shown that this requires that:

$$h \leq \frac{2}{|\lambda|}$$

Estimating the order of a method

- Suppose the error term is given by $e(h) \approx Ch^q$, C independent of h .
- As before, we can then estimate the error at $2h$, so that $e(2h) \approx C(2h)^q \approx 2^q e(h)$.
- We can approximate the rate, q , by:

$$\text{rate} = \log_2 \left(\frac{e(2h)}{e(h)} \right)$$

Section 4

Demo

Section 5

Supplementary Materials (Proofs)

Convergence for Euler's Method

We had to bypass the proof of the convergence of Euler's method. It is not a difficult proof and it uses standard techniques. If you're interested this section will provide a brief overview of the proof.

First, we will need a few lemmas that are used in the proof of the convergence of Euler's method. They are included here for completeness.

Lemma 5.7

Lemma 5.7. For all $x \geq 1$ and any positive m we have

$$0 \leq (1 + x)^m \leq e^{mx} \quad (6)$$

Proof. Straightforward application of Taylor's Theorem to $f(x) = e^x$ about $x_0 = 0$.

Lemma 5.8

Lemma 5.8. If

- s, t are positive real numbers
- $\{a_i\}_{i=0}^k$ is a sequence satisfying $a_0 \geq -\frac{t}{s}$
- $a_{i+1} \leq (i+s)a_i + t \quad i = 0, 1, \dots, k-1$

Then $a_{i+1} \leq e^{(i+1)s}(a_0 + \frac{t}{s}) - \frac{t}{s}$.

Proof. Left as an exercise. If you're interested in trying to prove it, the idea is to use a geometric series to show that

$$a_{i+1} \leq (i+s)^{i+1}\left(a_0 + \frac{t}{s}\right) - \frac{t}{s}$$

followed by an application of Equation 6, with $x = s$ to show result.

Convergence for Euler's method Theorem 2.1.

- A method is said to *converge* if the maximum global error tends to 0 as h tends to 0 (assuming that an exact solution exists and is sufficiently smooth).
- For Euler' method, which is $O(h)$, we would then expect that $e_i = y(t_i) - y_i$ should be of the same order.
- Let's consider the local truncation error first:

$$d_i = \frac{y(t_{i+1}) - y(t_i)}{h} - f(t_i, y(t_i)),$$
$$0 = \frac{y_{i+1} - y_i}{h} - f(t_i, y_i),$$

(cont.)

Subtracting the two, gives us a difference formula for the local truncation error:

$$d_i = \frac{e_{i+1} - e_i}{h} - [f(t_i, y(t_i)) - f(t_i, y_i)]$$

Solving for the error at t_{i+1} , we have:

$$e_{i+1} = e_i + h [f(t_i, y(t_i)) - f(t_i, y_i)] + h d_i, \quad i = 0, \dots, N$$

(cont.)

Taking absolute values:

$$|e_{i+1}| \leq |e_i| + h |[f(t_i, y(t_i)) - f(t_i, y_i)]| + |hd|,$$

where d is the maximum of $|d_i|$ over all time steps.

Since f is Lipschitz continuous with constant L , we can simplify the error difference equation to:

$$\begin{aligned} |e_{i+1}| &\leq |e_i| + hL|e_i| + |hd|, \\ &\leq (1 + hL)|e_i| + hd. \end{aligned}$$

(cont.)

Now note that we can repeat this process with e_i , which would give us:

$$\begin{aligned}|e_{i+1}| &\leq (1 + hL)|e_i| + hd, \\&\leq (1 + hL)[(1 + hL)|e_{i-1}| + hd] + hd = (1 + hL)^2|e_{i-1}| + (1 + hL)hd \\&\leq \dots \leq (1 + hL)^{i-1}|e_0| + hd \sum_{j=0}^{i-1} (1 + hL)^j \\&\leq d [e^{L(t_i - a)} - 1] / L\end{aligned}$$

where we've used the Lemmas above to compute the sum and the fact that $e_0 = 0$.

(cont.)

The final step is to note that by definition we have that:

$$d \geq \max_{0 \leq i \leq N-1} |d_i|$$
$$d_i = \frac{h}{2} y''(\xi_i) \quad i = 0, \dots, N$$

So if we can bound the second derivative such that:

$$M = \max_{a \leq t \leq b} |y''(t)| \implies d = \frac{h}{2} M.$$

which gives us our desired error bound:

$$|e_i| \leq \frac{Mh}{2L} [e^{L(t_i-a)} - 1], \quad i = 0, 1, \dots, N.$$

IVP Higher-Order Methods

Math 131: Numerical Analysis

J.C. Meza

April 25, 2024

Section 1

Higher-order Taylor Methods

Higher-order Taylor Methods

- In a similar vein to what we did to derive Euler's method, we can derive higher-order methods by extending the Taylor series approximation to include the higher derivatives.
- This will yield methods that are more accurate, but at a cost of having to compute the higher derivatives.
- This is important to remember because in a real-world problem each function evaluation could cost hours of computer time.

Caution

While higher-order Taylor methods can be generated, they are not often used in practice, as the requirement for higher-order derivatives is rarely met in real-world problems.

Extending Euler's method to higher-order

Idea

If Euler's method used a Taylor series expansion truncated after the first derivative (i.e. $n = 1$), it is natural to try higher values of n .

Taylor's Series

Let's start by writing down the Taylor series expansion of $y(t)$ about the current time step, t_i .

$$\begin{aligned}y(t_{i+1}) &= y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \dots \\&\quad + \frac{h^{(n)}}{n!}y^{(n)}(t_i) + \frac{h^{(n+1)}}{(n+1)!}y^{(n+1)}(\xi_i), \quad \xi_i \in [t_i, t_{i+1}]\end{aligned}$$

Using the statement of the IVP, we know that $y' = f(t, y)$, so we can replace the derivatives of y by the corresponding derivative of $f(t, y)$

(cont.)

Using $y' = f(t, y)$ leads to:

$$\begin{aligned}y(t_{i+1}) &= y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2}f'(t_i, y(t_i)) + \dots \\&\quad + \frac{h^{(n)}}{n!}f^{(n-1)}(t_i, y(t_i)) + \frac{h^{(n+1)}}{(n+1)!}f^{(n)}(t_i, y(t_i)), \quad \xi_i \in [t_i, t_{i+1}]\end{aligned}$$

For convenience let's denote the function $T^{(n)}$ as:

$$T^{(n)} = f(t_i, y(t_i)) + \frac{h}{2}f'(t_i, y(t_i)) + \dots + \frac{h^{(n-1)}}{n!}f^{(n-1)}(t_i, y(t_i)).$$

Example

$$T^{(1)} = f(t_i, y(t_i)),$$

$$T^{(2)} = f(t_i, y(t_i)) + \frac{h}{2} f'(t_i, y(t_i)),$$

$$T^{(3)} = f(t_i, y(t_i)) + \frac{h}{2} f'(t_i, y(t_i)) + \frac{h^2}{3!} f^{(2)}(t_i, y(t_i)),$$

⋮

Higher-order Taylor methods

- In a manner similar to Euler's method, this leads us to propose the following algorithm for higher-order Taylor methods:

-

$$y_0 = \alpha$$

$$y_{i+1} = y_i + hT^{(n)}(t_i, y_i), \quad i = 0, 1, \dots, N - 1.$$

- Clearly Euler's method is the special case of $n = 1$.

Remarks

- By the definition of the **local truncation error**, we can see that the higher-order Taylor method will have a local truncation error of $O(h^n)$, using the same argument that we used for Euler's method.
- For example, if we wanted to have a second order method, we would use $T^{(2)}$ in our difference equation:

$$T^{(2)} = f(t_i, y(t_i)) + \frac{h}{2}f'(t_i, y(t_i))$$

- Main disadvantage is the need for higher derivatives of $f(t, y)$.

Section 2

Runge-Kutta Methods

Runge-Kutta Methods

An alternative approach to higher-order Taylor leads to the well-known class of Runge-Kutta methods.

Idea

The idea for the Runge-Kutta methods is to have the high-order local truncation error of Taylor methods ***without the need to compute and evaluate the derivatives of $f(t, y)$.*** Ideally, we should only need function evaluations.

There are several ways to approach this. Your book shows you one way through matching coefficients to the higher-order Taylor method we just described. We'll take a different approach.

Runge-Kutta Order 2

Let's first rewrite the IVP as an integral:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t)) dt. \quad (1)$$

One simple idea is to now use our methods from numerical integration to evaluate the integral, for example the Trapezoid rule:

$$y_{i+1} = y_i + \frac{h}{2}(f(t_i, y_i) + f(t_{i+1}, y_{i+1})). \quad (2)$$

As with our discussion of Backward Euler, we can see that this is an implicit equation as y_{i+1} is both on the right and left hand sides of the equation and would (in general) require the solution of a nonlinear equation.

Second Idea

But what if we could approximate the value of y_{i+1} on the right hand side of Equation 2 with an explicit method?

Since we only know one explicit method (Euler's method), let's try it out and see what happens.

Let's define:

$$Y = y_i + hf(t_i, y_i),$$

and substitute Y for y_{i+1} into Equation 2:

$$y_{i+1} = y_i + \frac{h}{2}(f(t_i, y_i) + f(t_{i+1}, Y)).$$

These 2 equations comprise the ***explicit trapezoidal method***.

Explicit Trapezoid Method

- The explicit trapezoidal method is also known as a two-stage Runge-Kutta method
- The second stage comes about because we have to evaluate the function at a second point, namely $f(t_{i+1}, Y)$.
- We can summarize this approach by the following steps:

Explicit Trapezoid (2-stage explicit RK)

$$\begin{aligned} Y &= y_i + h f(t_i, y_i) \\ y_{i+1} &= y_i + \frac{h}{2} (f(t_i, y_i) + f(t_{i+1}, Y)) \end{aligned} \tag{3}$$

It can be shown that this method is second order accurate.

Explicit Midpoint

A similar process but using the midpoint rule instead of the trapezoid rule to compute the integral in Equation 1 will also work and yields the following two-stage algorithm, known as the ***explicit midpoint*** method:

Explicit Midpoint

$$\begin{aligned} Y &= y_i + \frac{h}{2} f(t_i, y_i) \\ y_{i+1} &= y_i + h f(t_{i+1/2}, Y) \end{aligned} \tag{4}$$

Programming Tip

Programming Tip

When coding the algorithms above, for example the ***explicit trapezoidal method***, the algorithm is usually stated in terms of the following:

$$k_1 = f(t_i, y_i)$$

$$k_2 = f(t_{i+1}, y_i + hk_1)$$

$$y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2)$$

You should convince yourself that these are equivalent to Equation 3.

Runge Kutta Order 4

We can derive higher-order Runge-Kutta methods using similar techniques. One of the most popular is the 4th order method, which can be written as:

$$k_1 = f(t_i, y_i),$$

$$k_2 = f\left(t_{i+1/2}, y_i + \frac{h}{2}k_1\right),$$

$$k_3 = f\left(t_{i+1/2}, y_i + \frac{h}{2}k_2\right),$$

$$k_4 = f(t_{i+1}, y_i + hk_3),$$

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

As the name implies, this method is $O(h^4)$.

Comparison of Runge Kutta methods

- Higher-order RK methods have higher accuracy but also cost more.
- For RK methods, it can be shown that the following relationship holds between the additional function evaluations and the local discretization error:

Table 1: Function Evals vs. Local Error

Function Evaluations	Local Discretization Error
2	$O(h^2)$
3	$O(h^3)$
4	$O(h^4)$
$5 \leq n \leq 7$	$O(h^{n-1})$
$8 \leq n \leq 9$	$O(h^{n-2})$
$10 \leq n$	$O(h^{n-3})$

Adaptive Steps

- Adaptive methods can be derived that adjust the time step to achieve a desired accuracy
- The methods use an estimate of the truncation error similar to the approach we saw in numerical integration
- One popular method (Runge-Kutta-Fehlberg) uses a combination of 2 RK methods of order 4 and 5 to adjust the time step to keep the discretization error below a desired tolerance

Section 3

Summary

Summary

- Higher-order Taylor methods can be derived, extending the accuracy of Euler's method, although these methods are rarely used in practice.
- Another approach is to use the same idea but replace higher derivatives of $f(t, y)$ with other approximations.
- Leads to multi-stage methods such as explicit Trapezoidal, explicit midpoint methods and the widely used and popular class of methods known as Runge-Kutta methods.
- Choosing a good method will depend on the problem, how smooth the functions are, and the desired accuracy.

Section 4

Supplementary Materials

General form for Runge Kutta methods

The general form of the Runge-Kutta equations with s stages is given by:

$$y_{i+1} = y_i + h \sum_{j=1}^s b_j k_j, \quad (5)$$

where the k_i terms are given by:

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + hc_2, y_i + a_{21}k_1), \\ k_3 &= f(t_i + hc_3, y_i + a_{31}k_1 + a_{32}k_2), \\ &\vdots \\ k_s &= f(t_i + hc_s, y_i + a_{s1}k_1 + a_{s2}k_2 + \dots + a_{s,s-1}k_{s-1}), \end{aligned} \quad (6)$$

The coefficients, a_{ij} , b_i , c_i , are determined by the process we introduced earlier that compares partial derivatives.

(cont.)

In addition, for consistency, we ask that the coefficients satisfy the equations:

$$\sum_{j=1}^{i-1} a_{ij} = c_i, \quad i = 2, \dots, s$$

and

$$\sum_{j=1}^s b_j = 1,$$

An easy way to visualize the coefficients is to use the Butcher tableau:

$$\begin{array}{c|c}\boldsymbol{c} & \boldsymbol{A} \\ \hline & \boldsymbol{b}^T\end{array}$$

where c, b are vectors of coefficients, and A is a matrix of coefficients, as defined in Equation 5 and Equation 6.

(cont.)

Using this tableau, we can, for example, summarize the formula for the Runge-Kutta method of order 2 (RK2) as:

0	0
$\frac{1}{2}$	$\frac{1}{2}$
<hr/>	
0	1

Substituting into Equation 6 yields the formulas:

$$k_1 = f(t_i, y_i),$$

$$k_2 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1\right),$$

$$y_{i+1} = y_i + hk_2.$$

(cont.)

We can also derive a similar set of formulas for a Runge-Kutta method with 4 stages that will yield a method of $O(h^4)$. The corresponding tableau is given by.

0	0
$\frac{1}{2}$	$\frac{1}{2} \ 0$
$\frac{1}{2}$	$0 \ \frac{1}{2} \ 0$
1	$0 \ 0 \ 1 \ 0$
	$\frac{1}{6} \ \frac{1}{3} \ \frac{1}{3} \ \frac{1}{6}$

(cont.)

Again, substituting into Equation 6, yields the Runge-Kutta method of order 4:

$$k_1 = f(t_i, y_i),$$

$$k_2 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1\right),$$

$$k_3 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_2\right),$$

$$k_4 = f(t_i + h, y_i + hk_3),$$

$$y_{i+1} = y_i + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4).$$

Recall that our goal was to replace the need for higher order derivatives while retaining the higher order local discretization error. The price we had to pay was in the form of extra function evaluations.

(cont.)

In the case of RK2, it was an additional 2 function evaluations. For RK4, it was 4 extra function evaluations. It can be shown that the following relationship holds between the additional function evaluations and the local discretization error holds for Runge-Kutta methods:

Table 2: Function Evals vs. Local Error

Function Evaluations	Local Discretization Error
2	$O(h^2)$
3	$O(h^3)$
4	$O(h^4)$
$5 \leq n \leq 7$	$O(h^{n-1})$
$8 \leq n \leq 9$	$O(h^{n-2})$
$10 \leq n$	$O(h^{n-3})$

Summary

- As you can see, when $n > 4$, there is no big advantage to increasing the order in terms of increasing the order of the local discretization error.
- As a consequence, the most popular of the Runge-Kutta methods is the one of order 4.

Multi-Step Methods for IVP

Math 131: Numerical Analysis

J.C. Meza

April 30, 2024

Section 1

Introduction

Summary - Single-Step Methods

- Started with a higher-order Taylor series method
- But we didn't like the requirement of higher-derivatives of f .
- Replaced derivatives with appropriate combinations of function evaluations (Runge-Kutta, etc.)
- Price we paid was that we need more function evaluations at each step (computational expense).

Question

Is it possible to use fewer function evaluations and still retain higher-order accuracy?

Motivation

- All the methods so far belong to a class of methods known as one-step methods, which is to say that all of the information used in the computation of the approximation at the next time step only used information from the immediately prior time step.
- However, you might ask yourself, can we use information from other previous steps to improve our approximation. This would be especially useful in the case when each of the function evaluations are computationally expensive.
- This leads us to proposing a set of methods that attempt to take advantage of all of this additional information already available to us.

Section 2

Multi-step methods

Idea - Multi-step methods

- Maybe we can use past information, and in particular, we could try to use the past values of y_i that we have already computed.
- This leads to the following idea:

Idea

Use some number of past values of y_i (e.g. $y_i, y_{i-1}, y_{i-2}, \dots$) to fit an ***interpolating polynomial to $f(t, y)$*** , which can then be used to derive a higher order method using techniques similar to the ones we used to derive the multi-stage (i.e. Explicit Trapezoid, Runge-Kutta, etc.) methods

Visually

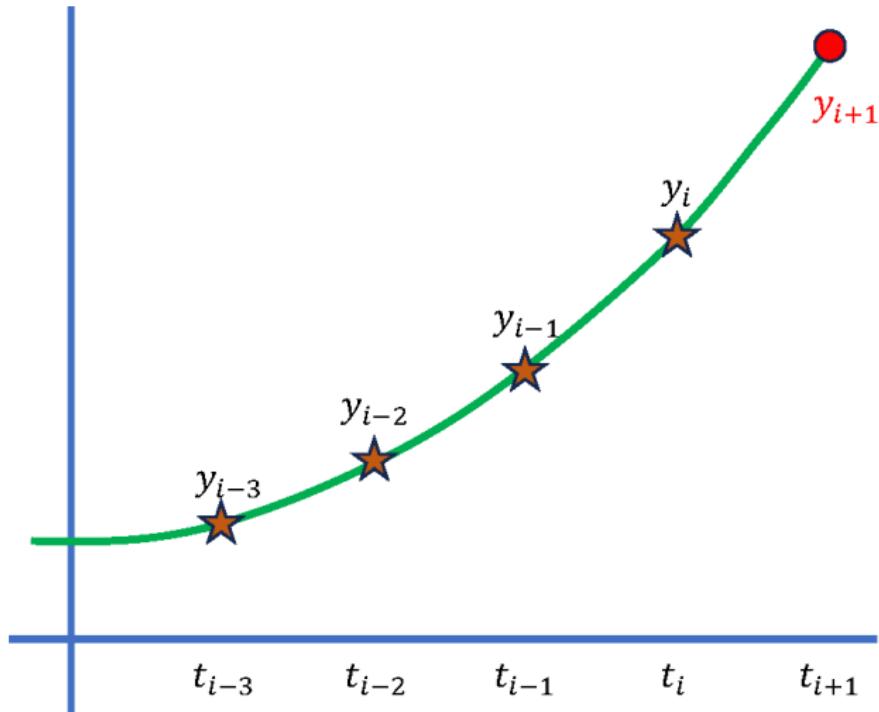


Figure 1: Multi-step method uses past computed values (stars)

Options

- There are many methods one could use to solve the IVP and we will give examples of several of the more popular multi-step methods including the
 - ▶ Adams-Basforth (explicit) and
 - ▶ Adams-Moulton (implicit) methods.
- All methods are derived by using an appropriate interpolating polynomial.
- We will not derive all the methods here, but if you're interested we give a brief derivation for the general case in the supplemental section.

Example: Linear interpolating polynomial

As before let's first rewrite the IVP as an integral:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t)) dt.$$

Using the current and immediate past timesteps as nodal points, we can write a first degree polynomial approximation to f by:

$$P_1(t) = f(t_{i-1}) + \frac{f(t_i) - f(t_{i-1})}{t_i - t_{i-1}}(t - t_{i-1})$$

(cont.)

Similarly to when we looked at quadrature methods, all we need to do is to replace f by $P_1(x)$ in the integral.

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t_{i-1}) + \frac{f(t_i) - f(t_{i-1})}{t_i - t_{i-1}}(t - t_{i-1}) dt.$$

Integrating the polynomial we get:

$$y(t_{i+1}) = y(t_i) + f(t_{i-1})(t_{i+1} - t_i) + \left[\frac{f(t_i) - f(t_{i-1})}{t_i - t_{i-1}} \right] \cdot \frac{(t - t_{i-1})^2}{2} \Big|_{t_i}^{t_{i+1}}.$$

To simplify the equations recall that: $h = (t_{i+1} - t_i)$ and $2h = (t_{i+1} - t_{i-1})$.

(cont.)

$$\begin{aligned}y(t_{i+1}) &= y(t_i) + f(t_{i-1}) \cdot h + \left[\frac{f(t_i) - f(t_{i-1})}{2h} \right] \cdot ((2h)^2 - h^2). \\&= y(t_i) + hf(t_{i-1}) + \frac{3h}{2} [f(t_i) - f(t_{i-1})] \quad \text{combine } h \text{ in 2nd term} \\&= y(t_i) + h \left[f(t_{i-1}) + \frac{3}{2}(f(t_i) - f(t_{i-1})) \right] \quad \text{factor out } h \\&= y(t_i) + h \left[\frac{3}{2}f(t_i) - \frac{1}{2}f(t_{i-1}) \right].\end{aligned}$$

Note: This method can be shown to be $O(h^2)$ accurate.

General Form

Can show that the general formula has form:

$$y(t_{i+1}) = y(t_i) + h \sum_{j=0}^n \beta_{nj} f(t_{i-j})$$

Example: Linear Interpolation, $n = 1$:

$$\beta_{n0} = 3/2, \beta_{n1} = -1/2.$$

Adams-Basforth

Example: Choosing $n = 3$:

$$\beta_{n0} = 55/24; \quad \beta_{n1} = -59/24; \quad \beta_{n2} = 37/24; \quad \beta_{n3} = -9/24.$$

Adams-Basforth fourth-order

$$\begin{aligned} y_0 &= \alpha_0, \quad y_1 = \alpha_1, \quad y_2 = \alpha_2, \quad y_3 = \alpha_3, \\ y_{i+1} &= y_i + \frac{h}{24} \left[55f(t_i, y_i) - 59f(t_{i-1}, y_{i-1}) + \right. \\ &\quad \left. 37f(t_{i-2}, y_{i-2}) - 9f(t_{i-3}, y_{i-3}) \right] \end{aligned} \tag{1}$$

Some Key Points

- Notice that Adams-Bashforth methods are explicit since we only use either the current or past (known) function values.
- Adams-Bashforth methods need to supply additional initial values. For example, in the case of an Adams-Bashforth fourth-order method, we need to have 4 initial values in total.
- These are usually computed through another explicit method, for example a Runge-Kutta method, which only require the one initial value.

Adams-Moulton

- We can easily derive implicit methods, by choosing the point at t_{i+1} as one of the interpolation points. The general philosophy still holds!
- This will of course require additional work to solve a nonlinear equation using some iterative method (fixed-point, Newton, etc.)
- As with other implicit methods, the advantage is that we may be able to take longer time steps, thereby reducing the computational cost.

Adams-Moulton

Adams-Moulton fourth-order

$$\begin{aligned}y_0 &= \alpha_0, \quad y_1 = \alpha_1, \quad y_2 = \alpha_2, \\y_{i+1} &= y_i + \frac{h}{24} \left[9f(t_{i+1}, y_{i+1}) + 19f(t_i, y_i) - \right. \\&\quad \left. 5f(t_{i-1}, y_{i-1}) + f(t_{i-2}, y_{i-2}) \right]\end{aligned}\tag{2}$$

Note that $f(t_{i+1}, y_{i+1})$ appears on both sides of the equation, which signals that this is an implicit method.

Predictor-Corrector Methods

- The fact that we have an option to use either an explicit method or an implicit method suggests the idea of using the two in combination to get the best of both worlds.
- General idea is to use an (explicit) Adams-Bashforth method to “predict” a value for $f(t_{i+1}, y_{i+1})$.
- Then use an Adams-Moulton method to “correct” this value.
- These types of methods are known as ***Predictor-Corrector methods.***

Example Algorithm

① **Predict** using one of the Adams-Basforth formulas:

- $y_{i+1}^{(0)} = y_i + \frac{h}{24} [55f(t_i, y_i) - 59f(t_{i-1}, y_{i-1}) + 37f(t_{i-2}, y_{i-2}) - 9f(t_{i-3}, y_{i-3})]$

② **Evaluate** $f = f(t_{i+1}, y_{i+1}^{(0)})$

③ **Correct** using one of the Adams-Moulton formulas, $k = 0, 1, \dots$:

- $y_{i+1}^{(k+1)} = y_i + \frac{h}{24} [9f(t_{i+1}, y_{i+1}^{(k)}) + 19f(t_i, y_i) - 5f(t_{i-1}, y_{i-1}) + f(t_{i-2}, y_{i-2})]$

④ **If** $|y_{i+1}^{(k+1)} - y_{i+1}^{(k)}| < \tau$ **then** converged

⑤ **Evaluate** $f = f(t_{i+1}, y_{i+1}^{(k)})$

Summary

- Both Adams-Basforth and Adams-Moulton methods are higher-order methods for IVP that do not require additional derivatives. Only 2 function evaluations are required at each step.
- Error estimates are easy to generate.
- Step size and order of the method can be changed to reduce computational cost
- We have to store past values.
- Both methods require additional initial values. These are usually computed through another explicit method, for example a Runge-Kutta method.
- Some of the formulas may be numerically unstable (polynomial interpolation).

Summary of Methods Studied

Table 1: Comparison of Different Solution Methods for IVP

Method	Local Truncation		No. f evals
	Error	Explicit/Implicit	
Euler	$O(h)$	E	1
Backward Euler	$O(h)$	I	1
Higher Order	$O(h^n)$	E	depends
Taylor			
Midpoint	$O(h^2)$	E	2
Runge-Kutta 2	$O(h^2)$	E	2
Runge-Kutta 4	$O(h^4)$	E	4
Adams-Bashforth	$O(h^4)$	E	1
Adams-Moulton	$O(h^4)$	I	1
Predictor-Corrector	$O(h^n)$	E/I	2+

Section 3

Demo

Demo: Basic SIR Model

- We demonstrate the use of a simple ODE/IVP solver by solving a problem of predicting the breakout of an epidemic using data from Merced County COVID cases taken from: USA Facts Merced County, California coronavirus cases and deaths
- To model an epidemic of an infectious disease, the usual approach is to use what is known as the SIR Model.
- This is one of the most basic (and hence easiest) models we can use. Most practical models take this as a starting point and enhance the model with additional equations.

SIR equations

The SIR equations are given by:

$$\frac{dS}{dt} = -\alpha SI$$

$$\frac{dI}{dt} = \alpha SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

$$N = S + I + R$$

where S is the number of **susceptible** (healthy) individuals, I represents the number of **infected** individuals, R is the number of people who have **recovered** from the disease, and N is the total population.

SIR Parameters

- The demo we presented had 4 parameters you can play with:
 - ▶ initial population (N),
 - ▶ the number of days to run the simulation for, and
 - ▶ the 2 parameters that represent the rates between susceptible and infected (α) and between infected and recovered (γ).
- The solver used comes from the deSolve package in R called `ode`. The default solver is “`lsoda`” (Petzold & Hindmarsh), but other choices are available.
- Calling the `ode` solver requires the initial conditions (`init`), the times at which to compute the solution (`times`, the function to evaluate the `ode` (`sir` in this case), and a list of parameters that the `ode` solver passes along to the `ode` function.

SIR Data

The original data taken from the site gave us the following plot:

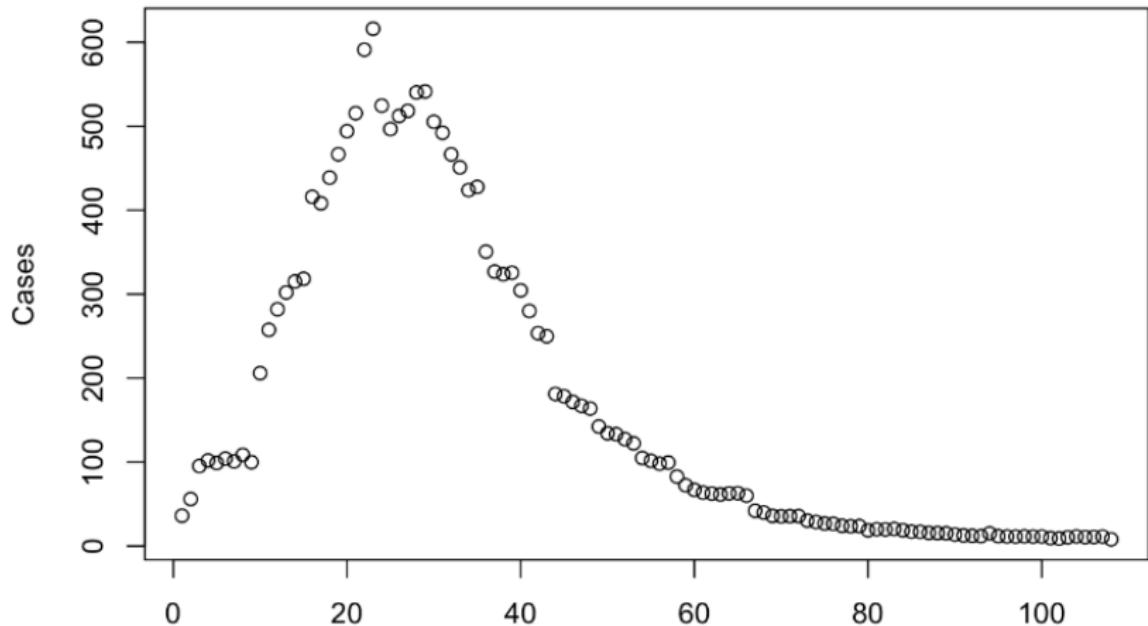


Figure 2: Merced County Covid Cases (weekly average)

Numerical Experiments

- In the demo, we played around with the parameters for the SIR model to match the data as best we could.
- According to the model, the basic Reproduction number $R_0 = [3.5 - 8.5]$.
- For comparison, for measles one of the more contagious diseases, $R_0 = 12 - 18$ while the normal flu has $R_0 \approx 1.28$.

Solution

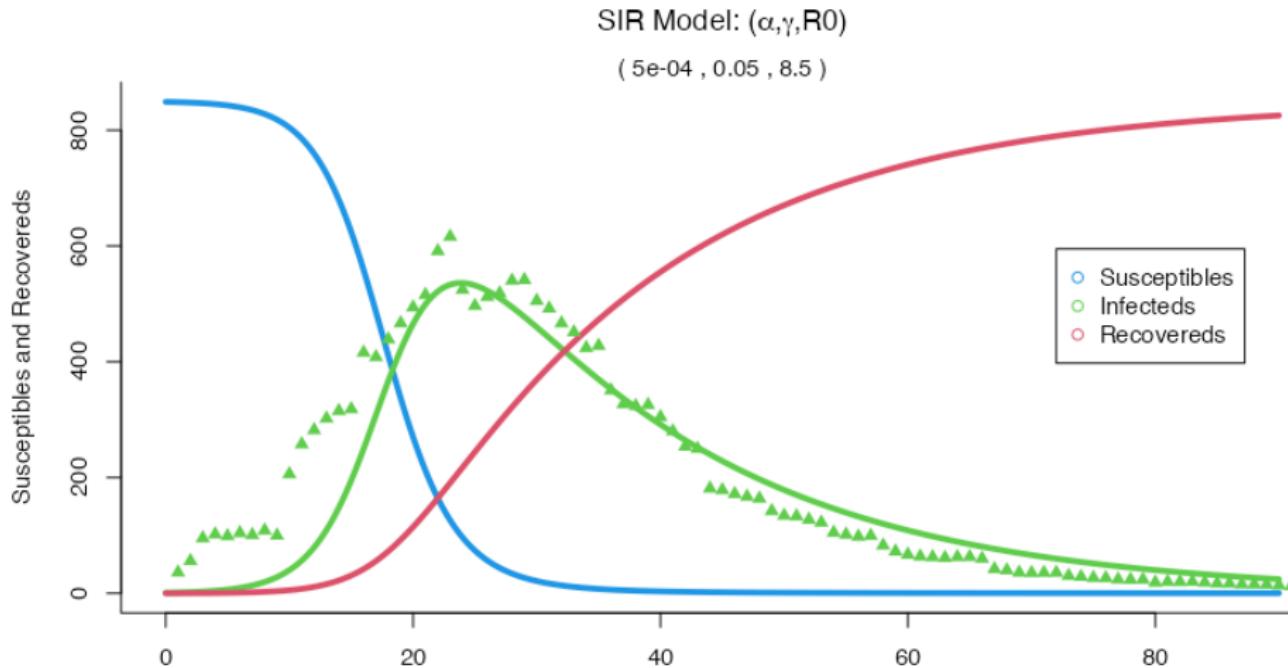


Figure 3: SIR demo using Merced County data

Section 4

Supplemental Materials

Definition

Let's first start with some notation and a definition. As before, let's assume that we have equally spaced time steps such that $t_i = a + ih, i = 0, 1, N$.

Definition: an s -step linear multistep method for solving the IVP has a difference equation of the form:

$$\sum_{j=0}^s \alpha_j y_{i+1-j} = h \sum_{j=0}^s \beta_j f_{i+1-j}, \quad (3)$$

where we let $f_{i+1-j} = f(t_{i+1-j}, y_{i+1-j})$.

Note - without loss of generality, we can assume that $\alpha_0 = 1$ since we can rescale all of the equations.

Explicit methods

- It will also be useful to distinguish between cases that need the function value $f(t_{i+1}, y_{i+1})$ at the next time step to compute y_{i+1} .
- In particular, if $b_0 = 0$ the method is called an **explicit (open) method** and we can write Equation 3 as:

$$y_{i+1} = - \sum_{j=1}^s \alpha_j y_{i+1-j} + h \sum_{j=1}^s \beta_j f_{i+1-j}$$

- Notice that we can recover Euler's method from the explicit form of this equation by setting $\beta_0 = 0, s = 1, \alpha_1 = -1, \beta_1 = 1$.

Implicit methods

The second case is if we let $b_0 \neq 0$ and the method is then called **implicit (closed)** as y_{i+1} appears on both sides of Equation 3 so it is only implicitly defined.

$$y_{i+1} - h\beta_0 f_{i+1} = - \sum_{j=1}^s \alpha_j y_{i+1-j} + h \sum_{j=1}^s \beta_j f_{i+1-j}$$

Note

In general implicit methods are more accurate than explicit methods and we can get by with larger time steps. The disadvantage is that we need to solve a system of linear (or nonlinear) equations at each time step. Additionally, the solution may not be unique (or even exist).

Derivation of multi-step methods

Advanced: This derivation closely follows the proof in Burden and Faires, pages 304-305. There are other similar proofs.

Our first step is to write:

$$\begin{aligned}y(t_{i+1}) - y(t_i) &= \int_{t_i}^{t_{i+1}} y'(t) dt \\&= \int_{t_i}^{t_{i+1}} f(t, y(t)) dt\end{aligned}$$

(cont.)

Let $y_i \approx y(t_i)$, and rearrange the equation to give us an expression for the approximation at the next time step t_{i+1}

$$y(t_{i+1}) \approx y_i + \int_{t_i}^{t_{i+1}} f(t, y(t)) dt \quad (4)$$

This should remind us of a similar problem we studied earlier, namely the numerical approximation for an integral, i.e. quadrature.

In the earlier case, we replaced the function by a polynomial, for which it will be easier to compute the integral. In that case, we used a Lagrange interpolating polynomial.

Here, it will be more convenient to use a ***Newton backward-difference polynomial*** because we can more easily incorporate previously calculated values.

(cont.)

As reminder we can write the $m - 1$ degree interpolating polynomial as
(ref: equation 3.13, p. 130 textbook):

$$P_{m-1}(t) = \sum_{k=0}^{m-1} (-1)^k \binom{-s}{k} \nabla^k f(t). \quad (5)$$

We can then use this polynomial (along with the remainder term) as an approximation to $f(t, y)$

$$f(t, y) = P_{m-1}(t) + \frac{1}{m!} f^{(m)}(\xi_i, y(\xi_i))(t - t_i)(t - t_{i-1}) \dots (t - t_{i+1-m}) \quad (6)$$

where $\xi_i \in (t_{i+1-m}, t_i)$.

(cont.)

Substituting Equation 5 and Equation 6 into Equation 4, and taking the integral of both sides, yields:

$$\begin{aligned} \int_{t_i}^{t_{i+1}} f(t, y(t)) dt &= \int_{t_i}^{t_{i+1}} \sum_{k=0}^{m-1} (-1)^k \binom{-s}{k} \nabla^k f(t_i, y(t_i)) dt \\ &\quad + \int_{t_i}^{t_{i+1}} \frac{1}{m!} f^{(m)}(\xi_i, y(\xi_i))(t - t_i)(t - t_{i-1}) \dots (t - t_{i+1-m}) dt. \end{aligned} \tag{7}$$

The integral is easier to solve by using the variable substitution:

$$\begin{aligned} t &= t_i + sh \\ dt &= hds \end{aligned}$$

in Equation 7

(cont.)

Substituting yields:

$$\begin{aligned} \int_{t_i}^{t_{i+1}} f(t, y(t)) dt &= h \left[\sum_{k=0}^{m-1} \nabla^k f(t_i, y(t_i)) (-1)^k \int_0^1 \binom{-s}{k} ds \right] \\ &\quad + \frac{h^{m+1}}{m!} \int_0^1 (s)(s+1) \dots (s+m-1) f^{(m)}(\xi_i, y(\xi_i)) ds \end{aligned}$$

The integrals involving the binomial function are easily computed (see Table 5.12 (textbook)). **Note:** The header in Table 5.12 incorrectly states the second column. It should read $(-1)^k \int_0^1 \binom{-s}{k} ds$

Also, recall that $\nabla p_n = p_n - p_{n-1}$, $n \geq 1$ and
 $\nabla^k p_n = \nabla(\nabla^{k-1} p_n)$, $k \geq 2$ (see p. 130, textbook)

(cont.)

Substituting the computed values of the integrals simplifies the equation to:

$$\begin{aligned} \int_{t_i}^{t_{i+1}} f(t, y(t)) dt &= h \left[f(t_i, y_i) + \frac{1}{2} \nabla f(t_i, y_i) \right. \\ &\quad \left. + \frac{5}{12} \nabla^2 f(t_i, y_i) + \frac{3}{8} \nabla^3 f(t_i, y_i) + \dots \right] \\ &\quad + \frac{h^{m+1}}{m!} \int_0^1 (s)(s+1) \dots (s+m-1) f^{(m)}(\xi_i, y(\xi_i)) ds \end{aligned}$$

(cont.)

The last step is to recognize that:

$$\nabla^0 f(t_i, y_i) = f(t_i, y_i)$$

$$\nabla^1 f(t_i, y_i) = f(t_i, y_i) - f(t_{i-1}, y_{i-1})$$

$$\nabla^2 f(t_i, y_i) = \nabla(\nabla^1 f(t_i, y_i))$$

$$\nabla^3 f(t_i, y_i) = \nabla(\nabla^2 f(t_i, y_i))$$

to expand the backward difference terms in the integral, followed by collecting like terms to arrive at the Adams-Basforth Four-Step ($m = 4$) Method:

$$y_0 = \alpha_0, \quad y_1 = \alpha_1, \quad y_2 = \alpha_2, \quad y_3 = \alpha_3,$$

$$\begin{aligned} y_{i+1} = y_i + \frac{h}{24} & \left[55f(t_i, y_i) - 59f(t_{i-1}, y_{i-1}) \right. \\ & \left. + 37f(t_{i-2}, y_{i-2}) - 9f(t_{i-3}, y_{i-3}) \right] \end{aligned}$$