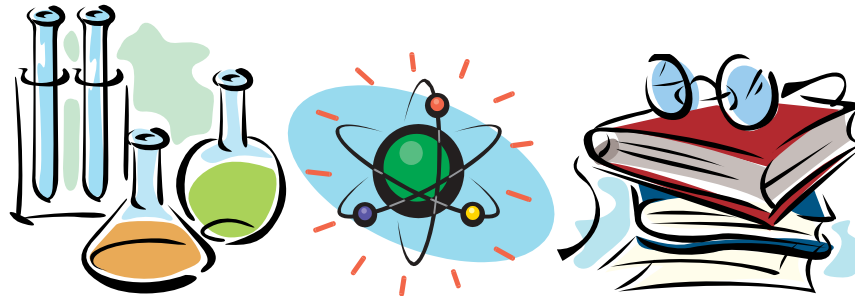


CISC 271

Scientific Computing

Notes by Randy Ellis

Introduction



Topics to be covered:

- A. Representations of Floating Point Numbers, Taylor Series Approximations, and Representational Error Sources. (4 hours)
- B. Root-finding in non-linear functions: Bisection Method, Newton's, Secant and False-Position Methods. Properties of these methods. (4 hours)
- C. Interpolation I: Polynomials, Lagrange's Method, Newton Divided Differences and Finite Differences, and Interpolation Errors. (5 hours)
- D. Linear Systems: Gaussian Elimination, and using Scaling and Pivoting. Error analysis of Gaussian elimination. General linear system computations. (4 hours)
- E. Interpolation II: Piecewise Polynomials. Cubic Splines. (2 hours)
- F. Functional approximations, and least squares approximations. (2 hours)
- G. Quadrature: Newton-Cotes integration, and adaptive integration. Gaussian quadrature. (4 hours)
- H. Ordinary Differential Equations: Euler Method, Higher Order Taylor Methods, and Runge-Kutta Methods. (3 hours)

CISC 271 Class 1

Taylor Series Approximation

Motivation

Suppose we have the following problem:

- We want to compute $\cos(0.1)$.
- We don't have access to a calculator or ancient look-up tables.
- We know the value of the $\cos(x)$ function for a nearby number, say, $x = 0$.
- We stayed awake during our calculus classes, and so know lots of derivatives of $\cos(x)$ at $x = 0$.

Question: Can we use what we know to approximate $\cos(0.1)$? And, if so, what will be the error of our approximation?

Taylor Series

To solve this problem, we introduce the Taylor Series, which will be used extensively in this course.

Taylor Series: If $f^{(k)}(x)$ exist at $x = c$ for $k = 0, 1, 2, \dots$ then

$$\begin{aligned} f(x) &\approx f(c) + f^{(1)}(c)(x - c) + \frac{f^{(2)}(c)}{2}(x - c)^2 + \dots \\ &= \sum_{k=0}^{\infty} \frac{f^{(k)}(c)}{k!}(x - c)^k \end{aligned}$$

In this definition, c is a constant and much is known about $f^{(k)}(c)$, whereas x is a variable near c and the value of $f(x)$ is sought. If $c = 0$, this series is known as the Maclaurin series.

So, suppose that we terminate the series after n terms, what would be the error in our approximation. To find the answer, we again turn to Taylor.

Taylor's Theorem: If $f(x)$ and $(n + 1)$ derivatives of $f(x)$ are continuous on (a, b) , and $c \in (a, b)$, then for any $x \in (a, b)$

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

$$\begin{aligned} & \frac{f^{(n)}(c)}{n!}(x-c)^n + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-c)^{n+1} \\ = & \sum_{k=0}^n \frac{f^{(k)}(c)}{k!}(x-c)^k + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-c)^{n+1}, \end{aligned}$$

where $\xi(x)$ is between c and x .

The last term is known as the truncation error, due to ending the infinite series at the n -th term. The truncation error gives us an idea how good an approximation of the function will be at n terms.

Therefore, to use the Taylor Series Approximation, we do the following:

- Write the formulae for k derivatives of $f(x)$, $f^{(k)}(x)$.
- Choose c , if not already specified.
- Write out the summation and the error term.
- If the error term goes to zero as $n \rightarrow \infty$, then the series *converges*, and the *infinite* Taylor series represents $f(x)$.

If we don't know if the error term goes to zero as $n \rightarrow \infty$, we can still estimate the size of the error term.

EXAMPLES

Since the Taylor series will be so helpful, let's consider some examples.

Example 1

What is the Taylor series for $f(x) = e^x$, $|x| < \infty$?

We know that $f^{(k)}(x) = e^x$, for all k . Next, we choose $c = 0$. Therefore, $f^{(k)}(c) = e^0 = 1$, for all k , such that

$$e^x = \sum_{k=0}^n \frac{x^k}{k!} + \frac{e^{\xi(x)}}{(n+1)!}x^{n+1}$$

Also, the error term goes to zero as $n \rightarrow \infty$ (since $(n+1)!$ always grows faster than x^{n+1} as $n \rightarrow \infty$ for any choice of $x < \infty$), so

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Example 2

What is the Taylor series for $f(x) = \sin(x)$, $|x| < \infty$?

We know that $f^{(k)}(x) = \sin\left(x + \frac{\pi k}{2}\right)$, for all k . Next, we choose $c = 0$. Therefore, we have

$$\sin(x) = \sum_{k=0}^n \frac{\sin\left(\frac{\pi k}{2}\right)}{k!} x^k + \frac{\sin\left(\xi(x) + \frac{\pi(n+1)}{2}\right)}{(n+1)!} x^{n+1}$$

Since the error term goes to zero as $n \rightarrow \infty$, so the upper limit for k is ∞ . Also, note that the even k terms are zero (since $\sin(0) = \sin(\pi) = \sin(2\pi) = \dots = 0$). Suppose, then that we change the summation parameter k to another parameter l where none of the terms of the summation is zero. I.e., let $l = 0, 1, 2, \dots$ where $k \rightarrow 2l + 1$. Therefore, the Taylor series for $\sin(x)$ becomes

$$\begin{aligned} \sin(x) &= \sum_{l=0}^{\infty} \frac{\sin\left(\frac{\pi(2l+1)}{2}\right)}{(2l+1)!} x^{(2l+1)} \\ &= \sum_{l=0}^{\infty} \frac{(-1)^l x^{2l+1}}{(2l+1)!} \\ &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \end{aligned}$$

Example 3

What is the Taylor series approximation for $\cos(0.1)$?

Using the same derivation process as for Example 2, above, we find that the Taylor series for $\cos(x)$ with respect to expansion point $c = 0$ is

$$\begin{aligned} \cos(x) &= \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} \\ &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots \end{aligned}$$

The actual value is $\cos(0.1) = 0.99500416527803\dots$ The Taylor approximations for different orders of n of $\cos(0.1)$ are (for the error, recall that the maximum value for $|\cos(x)|$ is 1):

n	approximation	$ \text{error} \leq$
0	1	$0.01/2!$
1	0.995	$0.0001/3!$
2	0.99500416	$0.000001/4!$
3	0.99500416527778	$0.00000001/5!$
\vdots	\vdots	\vdots

So, after just a few terms, we have an accurate approximation to $\cos(0.1)$.

PROXIMITY OF x TO c

Suppose we have the problem where we want to approximate $\ln(2)$. What we would like to know, is how important is the proximity of the value x to the expansion point c of the Taylor series approximation. So, let's consider two different solutions.

Solution 1: In this solution, we use the Taylor approximation for $\ln(1+x)$ about the expansion point $c = 0$ with $x = 1$. Note that $|x - c| = 1$. Therefore, the Taylor approximation is

$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \frac{1}{8} + \cdots$$

Solution 2: In this solution, we use the Taylor approximation for $\ln\left(\frac{1+x}{1-x}\right)$ about the expansion point $c = 0$ with $x = \frac{1}{3}$. Note that $|x - c| = \frac{1}{3}$. Therefore, the Taylor approximation is

$$\ln 2 = 2 \left(3^{-1} + 3^{-3} + 3^{-5} + 3^{-7} + \cdots \right)$$

So, how good are these approximations?

- Solution 1, with 8 ($n = 7$) terms: 0.63452
- Solution 2, with 4 ($n = 3$) terms: 0.69313

The actual value, rounded, is 0.69315. Therefore, we can see that the choice of c and x and their relative proximity is important to the accuracy of a Taylor approximation. The smaller $|x - c|$, in general, the more accurate the approximation given a fixed number of terms.

NOTE ON $\xi(x)$

Consider the following special case of the Taylor series expansion.

Mean value theorem (Taylor, $n = 0$): If $f(x)$ is at least once differentiable on $[a, b]$, then the $n = 0$ Taylor expansion of $f(x)$ about expansion point $c = a$ and evaluated at $x = b$ is

$$f(b) = f(a) + (b - a)f'(\xi), \quad \xi \in (a, b)$$

or

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

It is easy to see that ξ in this case always exists.

CISC 271 Class 2

Number Representation

BASE REPRESENTATION

Base 10

We, humans, often represent numbers in the decimal form. That is, we use base 10.

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 4 \times 1$$

\rightarrow # of 1's
 \rightarrow # of 10's
 \rightarrow # of 100's
 \rightarrow # of 1000's

We are used to numbers in this form, and know instantly that the number represented is one thousand,

In general, base 10 numbers represent

$$a_n a_{n-1} \dots a_0 = \sum_{k=0}^n a_k 10^k$$

Base 16

Equally, we could use other forms to represent number. For example, the hexadecimal form, i.e., base 16.

$$(1\ 2\ 3\ 4)_{16} = 1 \times 16^3 + 2 \times 16^2 + 3 \times 16 + 4 \times 1 = (4660)_{10}$$

\downarrow # of 1's
 \downarrow # of 16's
 \downarrow # of 16^2 's
 \downarrow # of 16^3 's

This number in hexadecimal form has a different meaning than a number in decimal form. To distinguish it, we use the $()_{16}$ subscript. Note that the digits of a hexadecimal number can range from $0 \rightarrow F$, where F represents number 15. Hence, 16 possible digits.

In general, base 16 numbers represent

$$a_n a_{n-1} \dots a_0 = \sum_{k=0}^n a_k 16^k$$

Base 2

Computers often use base two to store numbers. Base 2 just has an '0' and '1', or for computers: an "off" and "on" for a "bit" which holds our binary digit. Base 2 numbers are called binary numbers.

$$(1\ 0\ 1\ 1)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 \times 1 = (11)_{10}$$

Of course, numbers could be represented in base β , where β is any natural number, but these above, including base 8 or octal numbers, are the most commonly used in numerical analysis.

BASE CONVERSIONS

To convert a binary number to a decimal form is easy. You simply follow the above rules. For example,

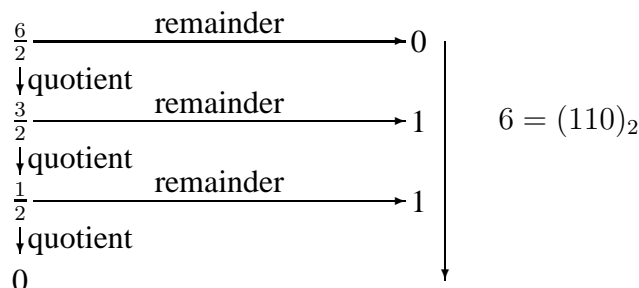
$$(110)_2 = 1 \times 4 + 1 \times 2 + 0 = (6)_{10}.$$

Now let us try to do it the other way - convert a decimal number to a binary number. First, consider an example from base 10 to base 10, say, 321 into ones (1), tens (2), and hundreds (3):

$\frac{321}{10}$	remainder	→ 1	$\left. \begin{array}{c} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{array} \right\} 321 = (321)_{10}$	
\downarrow quotient	$\frac{32}{10}$	remainder		→ 2
\downarrow quotient	$\frac{3}{10}$	remainder		→ 3
\downarrow quotient	0			

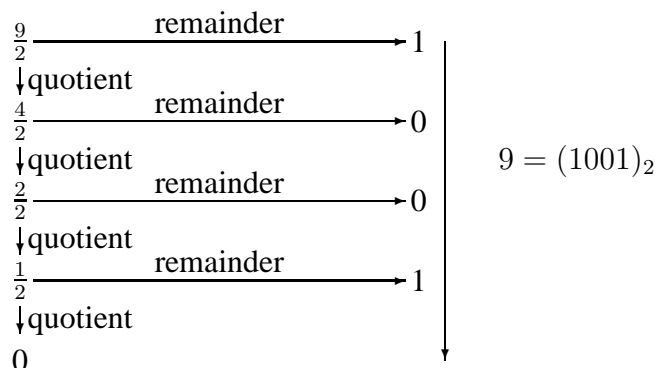
Now consider the analogy of the above conversion to the following conversion of a number from base 10 to base 2:

$$6 = (?)_2$$



Or

$$9 = (?)_2$$



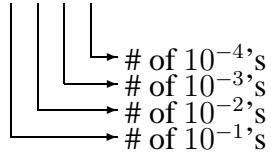
Now, let's convert the latter binary number back to base 10 to check: $(1001)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 = (9)_{10}$.

Note that the binary numbers are generated from the smallest digit (i.e., number of 2^0 's) to the largest digit (e.g., the number of 2^3 's in the latter example).

FRACTIONS

Up to now, all the numbers were integers. What if the numbers represented were fractions?

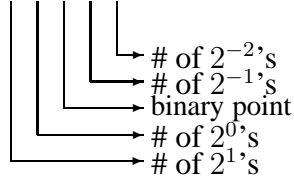
First, a base 10 example:

$$(0.1\ 2\ 3\ 4)_{10} = 1 \times 10^{-1} + 2 \times 10^{-2} + 3 \times 10^{-3} + 4 \times 10^{-4}$$


Recall the general representation for real numbers in base 10:

$$a_n \dots a_0 . b_1 b_2 \dots = \sum_{k=0}^n a_k 10^k + \sum_{k=1}^{\infty} b_k 10^{-k}$$

Real numbers in bases other than 10 are represented the same way. For example, in base 2:

$$(1\ 1\ 0\ .\ 0\ 1)_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (6.25)_{10}$$


The general representation for real numbers in base 2 is

$$(a_n \dots a_0 . b_1 b_2 \dots)_2 = \sum_{k=0}^n a_k 2^k + \sum_{k=1}^{\infty} b_k 2^{-k}$$

Or the general representation for real numbers in a base β :

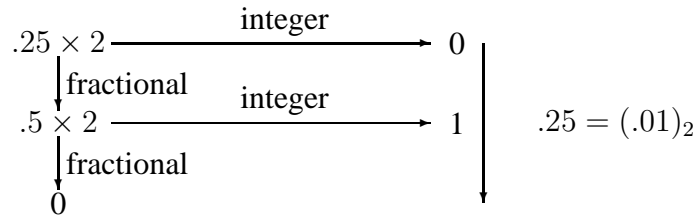
$$(a_n \dots a_0 . b_1 b_2 \dots)_{\beta} = \sum_{k=0}^n a_k \beta^k + \sum_{k=1}^{\infty} b_k \beta^{-k}$$

Fractional base conversions

So, how do we represent a decimal form fraction in binary form? Consider

$$6.25 = (?)_2.$$

First, put break the number into its integer and its purely factional parts: $6.25 = 6 + .25$. Its integer part of $(6)_{10} = (110)_2$, as before. Now for the fractional part:

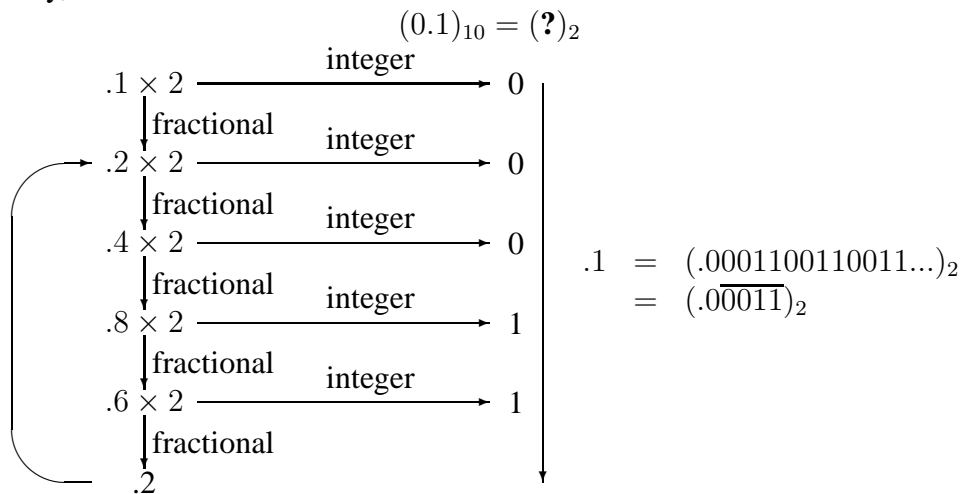


Therefore $(6.25)_{10} = (110.01)_2$.

Note that we could use a shortcut to generate base 2 numbers from base 10 numbers. We do this by first converting the base 10 number to base 8, then from base 8 to base 2. The former conversion produces about one octal digit for every step of the hand conversion of the decimal digit of an integer. And the latter conversion is trivial with each octal digit resulting in three binary digits (bits). For example:

$$(361.8125)_{10} = (551.64)_8 = (101\ 101\ 001.110\ 100)_2$$

Finally, let us consider



Notice that 0.1 cannot be represented by a finite number of digits in base 2. Also, recall that some numbers cannot be represented by a finite number of digits in base 10, for example,

$$\frac{1}{3} = (0.33333\dots)_{10} = (0.1)_3.$$

Indeed, there are some numbers that are irrational in all rational base representations: e.g., π , e , $\sqrt{2}$. But note that needing an infinite number of digits in a

base, as with our $\frac{1}{3}$ example above, does not mean that the number is necessarily irrational.

FLOATING-POINT NUMBERS

If a number is not an integer, we often write it in “scientific notation.” E.g.,

$$\left. \begin{array}{l} 18.25 = 0.1825 \times 10^2 \\ -0.056 = -0.56 \times 10^{-1} \end{array} \right\} \text{base 10}$$

and

$$\left. \begin{array}{l} (110.01)_2 = 0.11001 \times 2^3 \\ (27.25)_{10} = (11011.01)_2 \\ = 0.1101101 \times 2^5 \end{array} \right\} \text{base 2}$$

General Form

$$\begin{array}{c} \downarrow \text{sign} \\ \boxed{\pm} 0.\overline{a_1 a_2 \dots a_s} \times \boxed{\beta}^{\boxed{e}} \leftarrow \text{exponent (or characteristic)} \\ \text{radix point } \uparrow \quad \quad \quad \uparrow \text{base (radix)} \\ \quad \quad \quad \swarrow \\ \quad \quad \quad \text{mantissa or fraction} \end{array}$$

In the above, β is the base that the number is represented in. Therefore, if $\beta = 10$, then the number is in decimal form; if $\beta = 2$, then the number is in binary form. Also, $0.a_1 a_2 \dots a_s$ is the mantissa and s is the number of significant digits in the mantissa. And e is the exponent of the number.

$s, a_1, a_2, \dots, a_s, \beta, e$ are all integers.

$$\text{each } a_i < \beta, \text{ therefore } \begin{cases} \beta = 10 \Rightarrow a_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \\ \beta = 2 \Rightarrow a_i = 0, 1 \end{cases}$$

Some examples of numbers in different bases in scientific notation are:

$$0.1295 \times 10^5, \quad -0.1276 \times 8^2, \quad -0.10101 \times 2^3$$

However, a number may have many such representations:

$$18.25 = 0.1825 \times 10^2$$

$$\begin{aligned}
&= 0.01825 \times 10^3 \\
&= 0.001825 \times 10^4
\end{aligned}$$

Normalised form

For a unique representation - the first digit after the radix point must be greater than 0.

$$\text{i.e., } 1 \leq a_1 < \beta, \quad 0 \leq a_i < \beta, i = 2, \dots, s$$

This is called the *normalised form* of the number – each number can be uniquely represented in the normalised form. Zero? Zero is represented by possibly $0.0000\dots 0 \times \beta^0$ or $0.0000\dots 0 \times \beta^L$, where L is the smallest possible exponent (this will be introduced later).

What does this notation used for the general form represent?

$$\begin{aligned}
0.a_1a_2\dots a_s \times \beta^e &= \left(\frac{a_1}{\beta^1} + \frac{a_2}{\beta^2} + \dots + \frac{a_s}{\beta^s} \right) \times \beta^e \\
&= a_1\beta^{e-1} + a_2\beta^{e-2} + \dots + a_s\beta^{e-s}
\end{aligned}$$

For example, $0.101 \times 2^2 = 1 \times 2^{2-1} + 0 \times 2^{2-2} + 1 \times 2^{2-3} = 2 + \frac{1}{2} = (2.5)_{10}$.

FLOATING-POINT SYSTEM

If we collect all the numbers of the above general form, we get a set of numbers. This set of numbers is denoted by:

$$F(\beta, t, L, U); \quad \beta, t, L, U \text{ are given integers.}$$

This is a collection of all floating-point numbers in base β , with a mantissa that has exactly t digits (i.e., $s = t$). Also, the exponent is in between the integers L and U , or $L \leq e \leq U$. Therefore, $F(\beta, t, L, U)$ contains all floating-point numbers of the form

$$\pm 0.a_1a_2\dots a_t \times \beta^e, \quad \text{with } L \leq e \leq U,$$

and one additional number - zero.

The exponent is called the *characteristic* when a constant, usually L , is added to the exponent such that the actual stored integer is positive. Although extra work, this increases the exponent range by one value (consider a simple example with

three bits, one bit representing the sign of the exponent and two bits representing the range of the exponent, versus all three representing the range of the characteristic).

Examples of systems:

$$\begin{aligned}
 F(10, 1, 0, 1) &= \{\pm 0.a_1 \times 10^e | e = 0, 1; a_1 = 1, 2, \dots, 9\} \cup \{0\} \\
 &= \{\pm 0.1 \times 10^0, \pm 0.2 \times 10^0, \dots, \pm 0.9 \times 10^0, \pm 0.1 \times 10^1, \dots, \pm 0.9 \times 10^1\} \cup \{0\} \\
 &= \{\pm 0.1, \pm 0.2, \dots, \pm 0.9, \pm 1, \pm 2, \dots, \pm 9\} \cup \{0\} \\
 \text{Total} &: 37 \text{ numbers}
 \end{aligned}$$

$$\begin{aligned}
 F(2, 1, -1, 1) &= \{\pm 0.a_1 \times 2^e | e = -1, 0, 1; a_1 = 1\} \cup \{0\} \\
 &= \{\pm 0.1 \times 2^{-1}, \pm 0.1 \times 2^0, \pm 0.1 \times 2^1\} \cup \{0\} \\
 &= \{\pm 0.05, \pm 0.1, \pm 0.2\} \cup \{0\} \text{ (in base 10)} \\
 \text{Total} &: 7 \text{ numbers}
 \end{aligned}$$

Properties of $F(\beta, t, L, U)$

How many numbers in $F(\beta, t, L, U)$? Count all the possible choices for a_i and e .

$$\begin{aligned}
 \text{representation:} & \quad \pm 0.a_1 a_2 \dots a_t \times \beta^e \\
 \text{\# of choices:} & \quad 2(\beta - 1)\beta^{t-1}(U - L + 1) \\
 \text{Total} &= 2(\beta - 1)\beta^{t-1}(U - L + 1) + 1
 \end{aligned}$$

Largest number, Ω , in $F(\beta, t, L, U)$? Take the largest possible for each of a_i and e .

$$\begin{aligned}
 \Omega &= . \underbrace{(\beta - 1)(\beta - 1) \dots (\beta - 1)}_t \times \beta^U \\
 &= \left(\frac{\beta - 1}{\beta^1} + \frac{\beta - 1}{\beta^2} + \dots + \frac{\beta - 1}{\beta^t} \right) \times \beta^U \\
 &= \beta^U - \beta^{U-t} = \beta^U (1 - \beta^{-t})
 \end{aligned}$$

Smallest positive number, ω , in $F(\beta, t, L, U)$? Take the smallest possible value for the mantissa and the exponent, e .

$$\omega = 0.100\dots 0 \times \beta^L = \frac{1}{\beta} \times \beta^L = \beta^{L-1}.$$

EXAMPLE

IBM 360/370 Mainframes

Some IBM mainframes are $F(16, 6, -64, 63)$ - single precision.

Register	\pm	exp	a_1	a_2	a_3	a_4	a_5	a_6
32 bits	1	7	4	4	4	4	4	4

sign	1 bit	\pm
exp	7 bits	- 128 numbers (-64, 63)
a_i	4 bits	- 16 numbers (0,1,...,E,F)

Example, some properties of $F(16, 6, -64, 63)$:

$$\begin{aligned} \text{Total \# of fl-p numbers: } & 2 \times 15 \times 16^5 \times 128 + 1 = 4,026,531,841 \\ \Omega = & 16^{63} - 16^{57} \approx 7 \times 10^{75} \\ \omega = & 16^{-65} \approx 5 \times 10^{-79} \end{aligned}$$

For double precision $F(16, 14, -64, 63)$ - 64 bits - 32 more bits are added to the mantissa. Total number of floating-point numbers = 34,359,738,369. Ω and ω are almost the same as with single precision.

Extra Notes

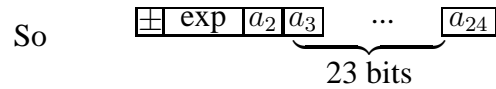
IEEE Standard

The IEEE is approximately $F(2, 24, -126, 127)$, but is more complicated.

Single precision	\pm	exp	mantissa	
32 bits	1	8	23	

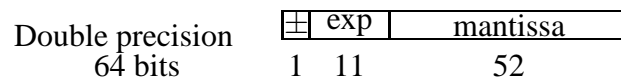
Only 23 bits are assigned to the mantissa. Why is $t = 24$?

In the representation of $F(2, 24, \dots) \rightarrow \pm.a_1a_2\dots a_{24}$, each $a_i = 0$ or 1 , but $a_1 > 0$, so $a_1 = 1$. Since a_1 is always 1, there is no need to store it, so this is a hidden bit. This unused bit in the mantissa is then given to the exponent, doubling its range with 8 bits compared to 7 for the IBM 360/370 mainframe.



Note, the IEEE standard is **not** implemented exactly as $F(2, 24, -126, 127)$. For example, zero is represented in a special form.

Double precision $\approx F(2, 53, -1022, 1023)$.



Also, an added degree of precision has been added with extended precision numbers $\approx F(2, 64, -16382, 16382)$. In this case, 15 bits are used in the exponent, and there is no hidden bit in the mantissa. There is a total of 80 bits in this representation.

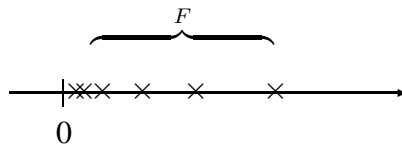
End of Extra Notes

CISC 271 Class 3

Error sources in Number Representations

REPRESENTATION OF A REAL NUMBER IN A FLOATING-POINT SYSTEM

Any $F(\beta, t, L, U)$ contains only a finite number of numbers. So, since the set of real numbers is infinite, many real numbers are not contained in F .



E.g., $F(10, 4, -3, 3)$

$$\begin{aligned}
 12.34 &= 0.1234 \times 10^2 \in F \\
 12.345 &= 0.12345 \times 10^2 \notin F, \quad 5 \text{ significant digits} \\
 1234 &= 0.1234 \times 10^4 \notin F, \quad e = 4 > 3 \\
 \frac{1}{3} &= 0.333\dots \notin F, \quad \infty \text{ number of significant digits}
 \end{aligned}$$

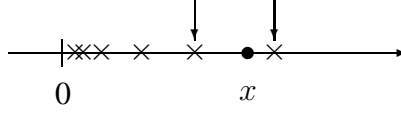
Consider the floating-point system $F(\beta, t, L, U)$. A real number x is in F only if

- a) the sum terminates before/at t terms *and*
- b) $L \leq e \leq U$.

If x does not satisfy both of the above, it is not in F . If a computer uses F to represent real numbers, then x can not be stored exactly.

Rounding and Chopping

How can we do computation with x ? We need to approximate x by a nearby floating-point number in $F(\beta, t, L, U)$.



Such a nearby floating-point number is called a floating-point representation of x denoted by $fl(x)$.

i.e., approximate x by $fl(x) \in F$.

Suppose x is a real number, we can always write it in the form

$$x = \left[\frac{a_1}{\beta^1} + \frac{a_2}{\beta^2} + \cdots \right] \beta^e = (0.a_1a_2\dots) \times \beta^e,$$

but the sum may have more than t terms, or may not terminate. E.g., $0.35 = 0.01011\overline{0} \times 2^0$.

So, how to find $fl(x)$? Consider the t -th and $t+1$ -th terms of the above summation.

$$\begin{aligned} x &= \left(\frac{a_1}{\beta^1} + \frac{a_2}{\beta^2} + \cdots + \frac{a_t}{\beta^t} + \frac{a_{t+1}}{\beta^{t+1}} + \cdots \right) \times \beta^e \\ &= 0.a_1a_2\dots a_t a_{t+1} \dots \times \beta^e, \text{ assume } L \leq e \leq U \end{aligned}$$

If $x \notin F(\beta, t, L, U)$, we can find $fl(x)$ in two ways:

1. Chopping, where we ignore the tail ($a_{t+1}a_{t+2}\dots$)

$$fl(x) = 0.a_1a_2\dots a_t \times \beta^e$$

2. Rounding up

$$fl(x) = \begin{cases} 0.a_1a_2\dots(a_t + 1) \times \beta^e; & \text{if } a_{t+1} \geq \frac{\beta}{2} \\ 0.a_1a_2\dots a_t \times \beta^e; & \text{if } a_{t+1} < \frac{\beta}{2} \end{cases}$$

Chopping is easier to implement using computer hardware, whereas rounding is more accurate, as we will see.

E.g., $F(10, 4, -3, 3)$

$$x = 12.345 = 0.12345 \times 10^2 \notin F.$$

$$\begin{array}{ll} \text{chopping} & fl(x) = 0.1234 \times 10^2 = 12.34 \\ \text{rounding} & fl(x) = 0.1235 \times 10^2 = 12.35 \end{array}$$

$$\begin{array}{ll} x = \frac{1}{3} = 0.3333... & \Rightarrow fl(x) = 0.3333 \\ x = 0.1234 \times 10^2 \in F & \Rightarrow fl(x) = x = 0.1234 \times 10^2 \\ x = 0.33333, y = 0.33334 & \Rightarrow fl(x) = fl(y) = 0.3333 \end{array}$$

On computers, everything is done within the floating-point system. If x is a real number, it is first represented by $fl(x)$ on the computer, and then all computations are carried out by using $fl(x)$.

E.g., in $F(10, 4, -3, 3)$, any computations involving $x = 0.33333$ are the same as those involving $y = 0.33334$.

Aside: You can think of your two hands as two digits of base 6 (i.e., you can store 0,1,2,3,4,5 on either hand). How could you store 0.153×6^2 ?

Overflow, Underflow

Consider the following real number, $x = (0.a_1a_2...a_ta_{t+1}...) \times \beta^e$.

If $L \leq e \leq U$, then we can find a $fl(x) \in F$ representation of the number using chopping or rounding. I.e., x has a representation ($fl(x)$) in F .

However, if

$$e < L \text{ or } e > U \rightarrow \text{No Representation!}$$

$$e < L \rightarrow \text{underflow}, \quad e > U \rightarrow \text{overflow}$$

E.g., $F(10, 4, -3, 3)$

$$\begin{array}{lll} x = 12345 = 0.12345 \times 10^5 & e = 5 > 3 & \text{overflow} \\ x = 0.00001 = 0.1 \times 10^{-4} & e = -4 < -3 & \text{underflow} \end{array}$$

MACHINE EPSILON

If x is a real number, and if $x > 0$, then we always have

$$1 + x > 1$$

no matter how small x is. However, in a floating-point system, this is *not* true.

E.g. $F(\beta, t, L, U) = F(10, 4, -3, 3)$. Take $x \in F$ to be

$$x = 0.1 \times 10^{-3} = 0.0001 > 0.$$

Take sum

$$1 + x = 1 + 0.1 \times 10^{-3} = 1.0001 = 0.10001 \times 10^1.$$

Note that

$$1 + x = 0.10001 \times 10^1 \notin F(10, 4, -3, 3).$$

We must represent it by

$$fl(1 + x) = fl(0.10001 \times 10^1) = 0.1000 \times 10^1 = 1.$$

So in a floating-point system, it is possible that when a positive number is added to 1, the result is still 1.

Define the *machine epsilon* (also called machine unit), μ , to be the smallest positive floating-point number in $F(\beta, t, L, U)$ such that

$$fl(1 + \mu) > 1.$$

Note that μ is different that ω , the smallest positive number in F , and is nearly always larger.

E.g., $F(10, 4, -3, 3)$:

$$1. \text{ Chopping } \begin{cases} fl(1 + 0.0009) = fl(1.0009) = 1 \\ fl(1 + 0.0010) = fl(1.001) = 1.001 > 1. \end{cases}$$

$$\text{so } \mu_c = 0.001 = 10^{-3}.$$

$$2. \text{ Rounding } \begin{cases} fl(1 + 0.0004) = fl(1.0004) = 1 \\ fl(1 + 0.0005) = fl(1.0005) = 1.001 > 1 \end{cases}$$

$$\mu_r = 0.0005 = 0.5 \times 10^{-3} = \frac{1}{2}10^{-3} = \frac{1}{2}\mu_c.$$

Position in Mantissa

$$\begin{array}{rcccccccc|ccc}
 & & & 1 & 2 & 3 & & t-1 & t & t+1 & & \\
 + & 1 & . & 0 & 0 & 0 & \dots & 0 & 0 & 0 & & \\
 & 0 & . & 0 & 0 & 0 & \dots & 1 & 0 & 0 & & \\
 \hline
 & 1 & . & 0 & 0 & 0 & \dots & 1 & 0 & 0 & \leftarrow \text{(Result of addition)} & \\
 \hline
 & 1 & . & 0 & 0 & 0 & \dots & 1 & 0 & 0 & \leftarrow \text{(Converted into} \\
 & & & & & & & & & & \text{floating-point form)} &
 \end{array}
 = \frac{1}{\beta^{t-1}} = \beta^{-(t-1)}$$

Figure 3.1: Value of μ when chopping is used.

When rounding is used, the machine epsilon is smaller. Also, note that in comparison, $\omega = 10^{-4}$.

In general, for $F(\beta, t, L, U)$, we can show:

$$\mu = \begin{cases} \beta^{-t+1} & \text{chopping} \\ \frac{1}{2}\beta^{-t+1} & \text{rounding} \end{cases}$$

For examples consider Figures 3.1 and 3.2, where the different values of μ are shown relative to 1. The larger t is, the smaller μ is, and more precision the system has. I.e., double precision is better than single precision.

An aside: μ can be calculated on a binary base machine using chopping by using following Pascal code:

```

var
  mu : real;

  mu := 1.0;
  while ((1.0 + mu) > 1.0) do begin
    mu := mu/2;
  end;
  return (2*mu);

```

Consider why.

Position in Mantissa

$$\begin{array}{rcccccccc|c|c|c}
 & & & 1 & 2 & 3 & & t-1 & t & t+1 & & \\
 + & 1 & . & 0 & 0 & 0 & \dots & 0 & 0 & 0 & & \\
 & 0 & . & 0 & 0 & 0 & \dots & 0 & \frac{\beta}{2} & 0 & & \\
 \hline
 & 1 & . & 0 & 0 & 0 & \dots & 0 & \frac{\beta}{2} & 0 & \leftarrow \text{(Result of addition)} & \\
 \hline
 & 1 & . & 0 & 0 & 0 & \dots & 1 & 0 & 0 & \leftarrow \text{(Converted into floating-point form)} & \\
 \hline
 \end{array}
 \quad = \frac{\beta}{2} \frac{1}{\beta^t} = \frac{\beta^{-(t-1)}}{2}$$

Figure 3.2: Value of μ when rounding is used.

On a more general β base machine, $F(\beta, t, L, U)$, we would replace $\mu/2$ by μ/β , and the returned value is $\beta\mu$.

ROUND-OFF ERROR

If x is real and if $x \notin F(\beta, t, L, U)$, we represent x by $fl(x)$. There is an error in this presentation – round-off error: which is the error in representing a real number by a floating-point number (using rounding or chopping).

Let's consider the round-off error associated with using a floating point system using chopping

$$\begin{aligned}
 x &= \left(\frac{a_1}{\beta^1} + \dots + \frac{a_t}{\beta^t} + \frac{a_{t+1}}{\beta^{t+1}} + \dots \right) \times \beta^e \\
 fl(x) &= \left(\frac{a_1}{\beta^1} + \dots + \frac{a_t}{\beta^t} \right) \times \beta^e
 \end{aligned}$$

The error is then:

$$x - fl(x) = \left(\frac{a_{t+1}}{\beta^{t+1}} + \dots \right) \times \beta^e$$

So the absolute error in the representation is:

$$\text{Absolute Error} = |x - fl(x)| \leq \left| \frac{(\beta - 1)}{\beta^{t+1}} + \dots \right| \times \beta^e = \beta^{e-t}$$

Equivalently, when using rounding the absolute error in the representation $f(x)$ is:

$$|x - fl(x)| \leq \frac{1}{2} \beta^{e-t}.$$

RELATIVE ERROR

The errors above are absolute errors ($|x - fl(x)|$) — the differences between the true value and its floating-point approximation. But absolute error does not usually tell the true story. For example, the error might be one inch in sending a rocket to the moon, or one inch in making a table. Obviously, the latter is of more relevance.

We will define *relative error* – error in reference to the true value:

$$R.E. = \frac{|\text{Absolute Error}|}{|\text{True Value}|}$$

Therefore, for chopping,

$$R.E. = \frac{|x - fl(x)|}{|x|} \leq \frac{\beta^{e-t}}{(0.a_1a_2\dots a_t\dots) \times \beta^e} \leq \frac{\beta^{e-t}}{(0.1000\dots) \times \beta^e} = \frac{\beta^{e-t}}{\left(\frac{1}{\beta}\right) \times \beta^e} = \beta^{1-t}$$

For rounding,

$$R.E. = \frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\beta^{1-t}$$

Thus in general,

$$\frac{|x - fl(x)|}{|x|} \leq \mu \quad \text{-- machine epsilon.}$$

$$\left. \begin{array}{l} \mu - \text{small} \Rightarrow \\ \uparrow \\ t - \text{large} \Rightarrow \end{array} \right\} \text{round-off error is small.}$$

Thus the range of values that are represented by a floating-point representation is

$$\boxed{fl(x) = x(1 + \varepsilon), \quad |\varepsilon| \leq \mu.}$$

EXAMPLES USING A FLOATING POINT SYSTEM

Consider the following floating point system: $F(10, 4, -20, 20)$ using rounding. First, determine Ω , ω , and μ .

$$\begin{aligned} \Omega &= 10^{20}(1 - 10^{-4}) = 9.999 \times 10^{19} \\ \omega &= 10^{-20-1} = 1.0 \times 10^{-21} \\ \mu &= \frac{10^{-4+1}}{2} = 5.0 \times 10^{-4} \end{aligned}$$

Note that the machine epsilon μ is much larger than the smallest representable number, ω , in $F(10, 4, -20, 20)$.

Suppose we have the following real numbers, $a = 0.68335 \times 10^8$, $b = 0.98\bar{6} \times 10^{-7}$, and $c = 0.25 \times 10^2$. What are the absolute errors and relative errors of the following operations done in $F(10, 4, -20, 20)$, using rounding? i) $a + b + c$?, ii) $\frac{a}{0.3} - \frac{c}{b}$?, and iii) $a * b * c$? First note that $fl(a) = 0.6834 \times 10^8$, $fl(b) = 0.9867 \times 10^{-7}$, and $fl(c) = 0.2500 \times 10^2$.

i) The exact solution is $0.68340025000000009867 \times 10^8$. In $F(10, 4, -20, 20)$ we have

$$\begin{aligned}
 a + b + c &= fl(fl(a + b) + c) \\
 &= fl(fl(fl(a) + fl(b)) + fl(c)) \\
 &= fl(fl(0.68340000000000009867 \times 10^8) + fl(c)) \\
 &= fl(0.6834 \times 10^8 + fl(c)) \\
 &= fl(0.68340025 \times 10^8) \\
 &= 0.6834 \times 10^8
 \end{aligned}$$

$$\begin{aligned}
 \text{Absolute Error:} &= |\text{Exact Soln} - \text{Floating Point Soln}| \\
 &= |0.68340025000000009867 \times 10^8 - 0.6834 \times 10^8| \\
 &= |0.25000000009867 \times 10^2| \\
 &= 0.25000000009867 \times 10^2 \\
 &\approx 0.2500 \times 10^2 \\
 \text{Relative Error:} &= \frac{|\text{Exact Soln} - \text{Floating Point Soln}|}{|\text{Exact Soln}|} \\
 &= \frac{0.25000000009867 \times 10^2}{|0.68340025000000009867 \times 10^8|} \\
 &\approx 0.3658 \times 10^{-6} \\
 &\approx 0.3658 \times 10^{-4}\% \text{ of Exact Soln}
 \end{aligned}$$

ii) The exact solution is $-0.2556981859 \times 10^8$. In $F(10, 4, -20, 20)$ we have

$$\begin{aligned}
 \frac{a}{3} - \frac{c}{b} &= fl\left(fl\left(\frac{a}{0.3}\right) - fl\left(\frac{c}{b}\right)\right) \\
 &= fl\left(fl\left(\frac{fl(a)}{0.3}\right) - fl\left(\frac{fl(c)}{fl(b)}\right)\right)
 \end{aligned}$$

$$\begin{aligned}
&= fl(fl(0.2278 \times 10^9) - fl(0.2533698186 \times 10^9)) \\
&= fl(0.2278 \times 10^9 - 0.2534 \times 10^9) \\
&= fl(-0.2560 \times 10^8) \\
&= -0.2560 \times 10^8
\end{aligned}$$

$$\begin{aligned}
\text{Absolute Error:} &= |\text{Exact Soln} - \text{Floating Point Soln}| \\
&= |-0.2556981859 \times 10^8 - (-0.2560 \times 10^8)| \\
&= 0.3018141 \times 10^5 \\
&\approx 0.3018 \times 10^5 \\
\text{Relative Error:} &= \frac{|\text{Exact Soln} - \text{Floating Point Soln}|}{|\text{Exact Soln}|} \\
&= \frac{0.3018141 \times 10^5}{|-0.2556981859 \times 10^8|} \\
&\approx 0.1180 \times 10^{-2} \\
&\approx 0.118\% \text{ of Exact Soln}
\end{aligned}$$

iii) The exact solution is 0.168577695×10^3 . In $F(10, 4, -20, 20)$ we have

$$\begin{aligned}
a * b * c &= fl(fl(a * b) * c) \\
&= fl(fl(fl(a) * fl(b)) * fl(c)) \\
&= fl(fl(0.67431078 \times 10^1) * fl(c)) \\
&= fl(0.6743 \times 10^1 * fl(c)) \\
&= fl(0.168575 \times 10^3) \\
&= 0.1686 \times 10^3
\end{aligned}$$

$$\begin{aligned}
\text{Absolute Error:} &= |0.168577695 \times 10^3 - 0.1686 \times 10^3| \\
&= |-0.22305 \times 10^{-1}| \\
&\approx 0.2231 \times 10^{-1} \\
\text{Relative Error:} &= \frac{0.22305 \times 10^{-1}}{|0.168577695 \times 10^3|} \\
&\approx 0.1323 \times 10^{-3} \\
&\approx 0.1323 \times 10^{-1}\% \text{ of Exact Soln}
\end{aligned}$$

CISC 271 Class 4

Error Propagation and Avoidance

SUBTRACTIVE CANCELLATION

Consider the following calculation.

$F(10, 2, -10, 10)$ where $x = -.119$ and $y = .12$, such that $-fl(x) = fl(y) = .12$. The relative error is

$$R.E. = \left| \frac{(x + y) - fl(fl(x) + fl(y))}{(x + y)} \right| = \left| \frac{.001 - 0}{.001} \right| = 1 \quad \text{Large!}$$

Let us analyse why this relative error can be large. When subtracting two numbers of the same sign, with similar magnitude, significant digits may be lost. This results in a large relative error – called subtractive cancellation, or cancellation error.

The reason for this is as follows. Consider $x, y \notin F$.

$$\begin{aligned} fl(x) &= .a_1 \cdots a_{p-1} a_p \cdots \tilde{a}_t \times \beta^e \\ fl(y) &= .a_1 \cdots a_{p-1} b_p \cdots \tilde{b}_t \times \beta^e \\ fl(x) - fl(y) &= \underbrace{.0 \cdots 0}_{p-1} c_p \cdots \tilde{c}_t \times \beta^e \\ &= .c_p \cdots \tilde{c}_t \times \beta^{e-p} \end{aligned}$$

where \tilde{a}_t and \tilde{c}_t denote inaccurate values. Originally, the inaccurate digit is in the t -th place (with a maximum relative error of the floating point representation on order β^{-t}). After subtraction, the inaccurate digit is in the $(t - p)$ - th place (with a relative error on order of $\beta^{-(t-p)}$, larger by a factor of β^p). I.e., the most significant digits have been cancelled out. In an extreme case, all the digits except the last are the same, whereupon they are cancelled out after subtraction with only the largest digit left, but it is in error in the first place.

AVOIDING CANCELLATION ERROR

Essentially, *avoid* subtracting two nearly equal numbers.

E.g., if $x \approx 0$, then $\cos(x) \approx 1$. Thus $\frac{1 - \cos x}{\sin^2 x}$ may result in cancellation. So, rewrite the equation to avoid the problem:

$$\frac{1 - \cos x}{\sin^2 x} = \frac{(1 - \cos x)(1 + \cos x)}{\sin^2 x(1 + \cos x)} = \frac{1 - \cos^2 x}{\sin^2 x(1 + \cos x)} = \frac{1}{(1 + \cos x)}$$

There is no cancellation error in $1 + \cos x$, when $x \approx 0$.

Example: above with $F(10, 4, -10, 10)$ with chopping with $x = 0.05$ radians.

exact solution: $\frac{1 - \cos x}{\sin^2 x} \simeq 0.5003126$

with $F(10, 4, -10, 10)$:

$$\begin{aligned}\cos x &= .9987 \times 10^0 \\ 1 - \cos x &= .1300 \times 10^{-2} \\ \sin x &= .4997 \times 10^{-1} \\ (\sin x)^2 &= .2497 \times 10^{-2} \\ \frac{1 - \cos x}{\sin^2 x} &= .5206 \times 10^0 \\ \frac{1}{1 + \cos x} &= .5005 \times 10^0\end{aligned}$$

$$\begin{aligned}\text{Relative Error} &= \left| \frac{.5003126 - .5206}{.5003126} \right| \\ &\cong 0.04 \\ &= 4\%\end{aligned}$$

Another solution would be to use a Taylor expansion of $\cos x$ about 0. So for $\cos x$,

$$\cos x \simeq 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

ignoring the smaller terms, since $x \approx 0$.

Therefore,

$$\frac{1 - \cos x}{\sin^2 x} = \frac{\frac{x^2}{2!} - \frac{x^4}{4!}}{\sin^2 x}$$

which does not suffer as much from cancellation error.

Another example. Suppose we want to compute $e^x - e^{-2x}$ at $x \approx 0$? Use the Taylor series for e^x twice and add common terms (details left as an exercise).

AVOIDING ROUND-OFF ERROR GROWTH

The procedure used for avoiding round-off error growing quickly very much depends on the problem at hand. Some useful strategies are:

- Double or extended precision
Sometimes this is the only solution to deal with cancellation error.
- Taylor expansion
- Changing definition of variables
- Rewriting the equation to avoid cancellation error

- Grouping equivalent terms

Here, we try to avoid summation of numbers of different orders of magnitude. One approach is to sum from the smallest number to the largest. For example, consider the following summation in $F(10, 3, -10, 10)$ with rounding:

$$1 + .002 + .002 + .002$$

Rather than summing as

$$fl(fl(fl(1 + .002) + .002) + .002) = 1$$

we sum as

$$fl(1 + fl(.002 + fl(.002 + .002))) = 1.01$$

SUMMARY OF WAYS ERRORS CAN BE INTRODUCED

In summary, ways that errors may be generated:

- Mathematical modelling
- Programming errors
- Uncertainty in Physical Data
- Machine errors (finite representation - ex., round-off error, cancellation error, propagated error, changing base representation)
- Truncation error (ex., Taylor series)
- Algorithmic design errors (ex., the function may be well-posed but the algorithm used is not)

ARITHMETIC RULES

With real numbers, arithmetic rules hold. For example, the Associative rule: $x, y, z \text{ real} \Rightarrow (x+y)+z = x+(y+z)$. *But*, in floating-point operations, the usual arithmetic rules do not hold in general. I.e., $fl(fl(x+y)+z) \neq fl(x+fl(y+z))$. E.g.,

$$fl(fl(1 + \frac{\mu}{2}) + \frac{\mu}{2}) = fl(1 + \frac{\mu}{2}) = 1$$

$$fl(1 + fl(\frac{\mu}{2} + \frac{\mu}{2})) = fl(1 + \mu) > 1$$

CISC 271 Class 5

Rootfinding: Bisection

Consider the following example. We want to compute $\sqrt{3}$. I.e., find x where $x = \sqrt{3}$.

$$x = \sqrt{3} \Rightarrow x^2 = 3 \text{ or } x^2 - 3 = 0.$$

So, the problem reduces to finding a *root* of $x^2 - 3 = 0$.

In many applications, we need to find a root of

$$f(x) = 0, \quad f(x) \text{ is a continuous function, } f : R \rightarrow R, \text{ on } [a, b].$$

See Figure 5.1 for a generic function example.

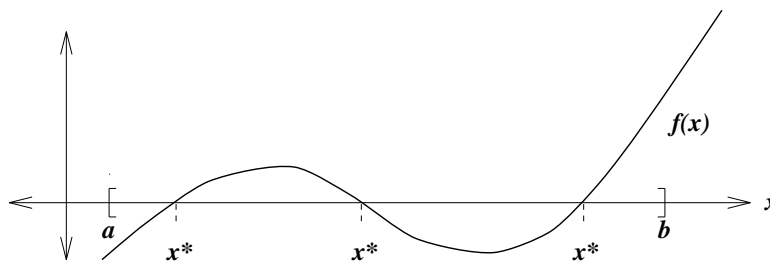


Figure 5.1: A generic function example.

In the above example, $f(x) = x^2 - 3$.

Definition: root: A root of $f(x)$ is a $x^* \in [a, b]$ such that $f(x^*) = 0$.

A root of $f(x) = 0$ is also called a *zero* of function $f(x)$.

So, we may restate our problem as:

$$\text{find an } x^* \text{ such that } f(x^*) = 0 \text{ for a given } f : R \rightarrow R.$$

But, since we are dealing with floating point representations, this problem usually restated as:

$$\text{find an } \tilde{x}^* \text{ such that } \tilde{f}(\tilde{x}^*) < \epsilon \text{ for a given } \tilde{f} \text{ and } \epsilon,$$

where $\tilde{x} = fl(x)$. The ϵ is the tolerance of error in our solution.

Generally stated, the approach to finding a root of a function is to, given one or more guesses, iteratively generate a better guess. In general, there are two kinds

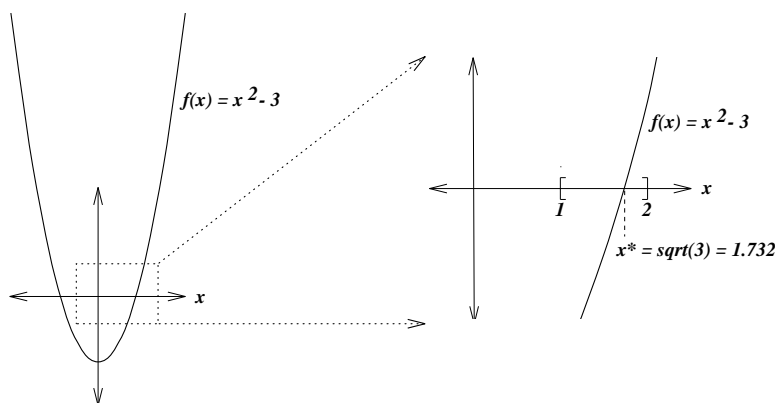


Figure 5.2: Picture of root for $x^2 - 3 = 0$.

of methods, global and local. A global method will always converge, whereas a local method will converge if the guess is “close enough.” We will consider both types of “root-finding” methods. An ideal code will start with a [slower] global method, and then switch to a [faster] local method later to get a more precise answer quickly.

Global Algorithms: Bisection, false position
 Local Algorithms: Secant, Newton’s method

BISECTION

Let us first consider an intuitive approach to finding a root of $x^2 - 3 = 0$.

See Figure 5.2: Picture of the function $f(x) = x^2 - 3 = 0$.

Since $1^2 < 3$ and $2^2 > 3$, then we know that $x^* \in [1, 2]$. So, as a first approximation, we take $x_1 = 1.5$

$$x_1^2 = 2.25 < 3 \Rightarrow x^* \in [1.5, 2]$$

A second approximation is $x_2 = 1.75$

$$x_2^2 = 3.06 > 3 \Rightarrow x^* \in [1.5, 1.75]$$

After many iterations, we have $[a_n, b_n]$, and $x^* \in [a_n, b_n]$. If $b_n - a_n$ is small enough (to meet a certain accuracy requirement, or tolerance), then we can use any point $x_n \in [a_n, b_n]$ to approximate $x^* = \sqrt{3}$.

More formally:

Suppose $f(x)$ is continuous on $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs, i.e.,

$$f(a)f(b) < 0,$$

then there exists a root $x^* \in [a, b]$.

To find an approximation of x^* , we proceed as follows. First, assume, for demonstration purposes, that $f(a) < 0$, and $f(b) > 0$.

See Figure 5.3: Picture of the starting point for the Bisection method.

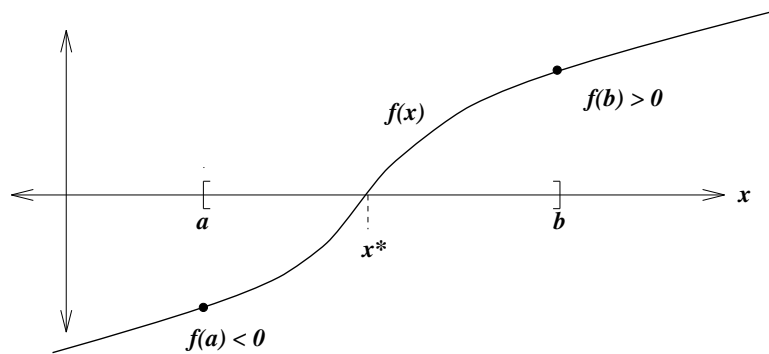
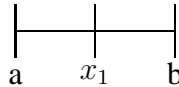


Figure 5.3: Picture of starting point for Bisection method.

Let $x_1 = \frac{a+b}{2}$, (mid-point).



Sketch of the the Bisection Algorithm

if $f(x_1) = 0 \Rightarrow$ I've found the root $x_1 = x^*$, and stop
otherwise $\begin{cases} f(x_1) > 0, & \text{then } x^* \in [a, x_1] \Rightarrow x_2 = \frac{a+x_1}{2} \\ f(x_1) < 0, & \text{then } x^* \in [x_1, b] \Rightarrow x_2 = \frac{x_1+b}{2} \end{cases}$
if $f(x_2) = 0 \Rightarrow$ I've found the root $x_2 = x^*$, and stop,
otherwise find the sign of $f(x_2)$ and determine x_3 , etc.

We continue this to get x_1, x_2, x_3, \dots

Are we guaranteed to find a root?

In general, none of $x_1, x_2, \dots, x_n, \dots$ will be x^* , even if we continue forever. So why do this? Even though we may never find x^* this way, we hope that x_n will get closer to x^* as n gets large.

$$\begin{aligned} x_1 : & \quad x^* \in [a, x_1] \text{ or } [x_1, b] \text{ so } |x^* - x_1| \leq \frac{b-a}{2} \\ x_2 : & \quad |x^* - x_2| \leq \frac{1}{2} \frac{b-a}{2} \\ & \quad \vdots \\ x_n : & \quad |x^* - x_n| \leq \frac{b-a}{2^n} \end{aligned}$$

We may never find x^* , but we can find x_n which can be as close to x^* as we please within the constraints of the floating point system. Note that $f(x)$ need not be continuous.

STOPPING CRITERIA

This brings us to the question of an appropriate stopping criterion. We may consider using

$$|\tilde{f}(x_n)| < \epsilon.$$

There are two possible problems with this choice. First, is that there may not exist any such x_n . Consider a function which is nearly vertical as it crosses the x -axis. Then it may be that $|x_n - x_{n-1}| < \mu$, where μ is the machine epsilon, but $|\tilde{f}(x_n)| > \epsilon$. Since we can't subdivide the interval any further, the method fails. Also, since the Bisection method does not require that $f'(x)$ be continuous, then it is possible that the function be vertical across the axis. Second, if the function is very flat as it crosses the axis, $|\tilde{f}(x_n)| < \epsilon$, but x_n still be very far from the actual root.

Rather than the above criterion, we use a stopping criterion of the form

$$\left| \frac{(\tilde{b} - \tilde{a})}{(\tilde{a} + \tilde{b})/2} \right| < \epsilon,$$

which is a variation of $|\tilde{b} - \tilde{a}| < \epsilon$. This can be considered as the relative tolerance for the accuracy of the root. This is tricky to use if the root is near zero.

In pseudocode, the Bisection algorithm looks like:

```
Input:      a, b, f(), tol;
Algorithm:  set epsilon = mu + tol;
```

```

m := (a + b)/2;
fa := f(a); fb := f(b); fm := f(m);
while (fm not= 0 and (|b-a|/max(1,|m|)) > epsilon) do begin
    s := sign(fa);
    if (s*fm < 0) then
        b := m; fb := fm;
    else
        a := m; fa := fm;
    end if;
    m := (a+b)/2; fm := f(m);
end while;
return (m);

```

Analysis: Note the special setting for `epsilon`. This is because if the given relative error tolerance is less than the machine unit, the algorithm may never terminate. Also, the usual condition is $f(x) * f(m)$, but as $m \rightarrow x^*$, this can cause underflow as $f(x) \rightarrow 0$. If x^* is huge, may need to normalize $|b - a|$, or change the variable used ($x \rightarrow \frac{1}{y}$). If x^* is tiny, need to avoid round-off and underflow.

To use the algorithm, one needs to find a, b such that $f(a)f(b) < 0$. Chosen as such, $[a, b]$ always contains a root.

Convergence: logarithmic (1 binary bit/iteration).

Example. We want to find $\sqrt{3}$ by the bisection method, starting with $[1, 2]$. If we want the solution to be correct up to 5 digits, how many steps do we need?

$$|\sqrt{3} - x_n| \leq \frac{b-a}{2^n} = \frac{1}{2^n}.$$

For accuracy of 5 digits, we let $\frac{1}{2^n} < 10^{-6}$, and solve for n to find out how many steps.

$$2^n > 10^6 \quad n > 6 \left(\frac{\ln 10}{\ln 2} \right) \approx 20.$$

I.e., after 20 steps, we are guaranteed to have 5 correct digits. We know this in advance.

Another example. Consider $[a, b] = [0, 1]$ where $x^* = 0.1 \times 2^{-3}$ (in base 2). Therefore $x_1 = \frac{0.1 \times 2^1}{2} = 0.1 \times 2^0$. Then $x_2 = \frac{0.1 \times 2^0}{2} = 0.1 \times 2^{-1}$, etc., gaining one bit of accuracy with each iteration.

There is a possible problem with the Bisection method in that the method may catch a singularity (where $f(x^* - \delta) \rightarrow +\infty$ and $f(x^* + \delta) \rightarrow -\infty$ as $\delta \rightarrow 0$) as if it were a root. So, the algorithm used should check if $|f(x_n) - f(x_{n-1})| \rightarrow 0$.

Newton's Method, Secant Method

Although the Bisection method is easy to compute, it is slow. Now, we will consider more interesting methods of root-finding. The first two methods are local methods, and the last, which will be discussed in next class, is a global method. All the methods are based on using lines to get better iterative approximations for the root of a function.

NEWTON'S METHOD

One of the most widely used methods is Newton's method. Originally, Newton used a similar method to solve a cubic equation. It has since been extended to differential equations. Over a very small interval, most functions can be locally approximated by a line. This idea is the basis of Newton's method.

The idea is to start with an initial guess for the root, x_1 . Then draw a line tangent to the function at the point $(x_1, f(x_1))$. The tangent line's intersection with the x -axis is defined to be x_2 . We repeat this process to get x_1, x_2, x_3, \dots

Figure 6.1 - Example of Newton's Method.

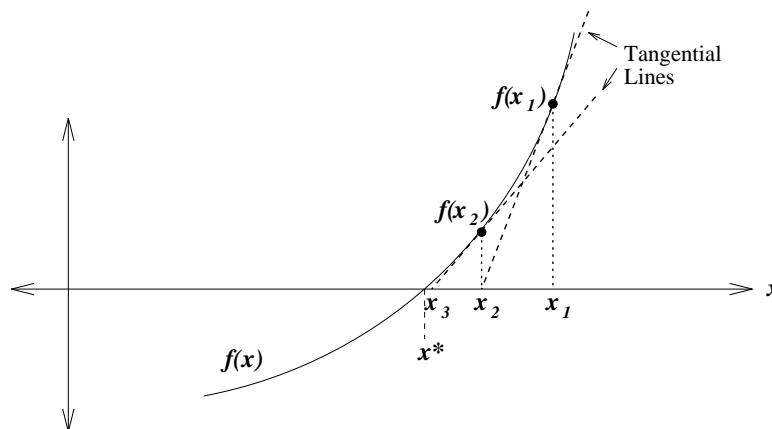


Figure 6.1: Example of Newton's Method.

Why a tangent line? If the function $f(x)$ is a linear function, i.e.,

$$f(x) = ax + b,$$

then $y = f(x)$ is a line. If we start off with any guess x_1 , the tangent line at $(x_1, f(x_1))$ agrees with $y = f(x)$. Therefore, $x_2 = x^*$. I.e., for linear functions, Newton's method yields an exact solution after one iteration.

Now, if $f(x)$ is any function, we may approximate it by a linear function. At the point $(x_1, f(x_1))$,

$$\text{Taylor expansion: } f(x) \approx f(x_1) + f'(x_1)(x - x_1) + \dots$$

Figure 6.2 - Picture of the Taylor approx at a point x_1 .

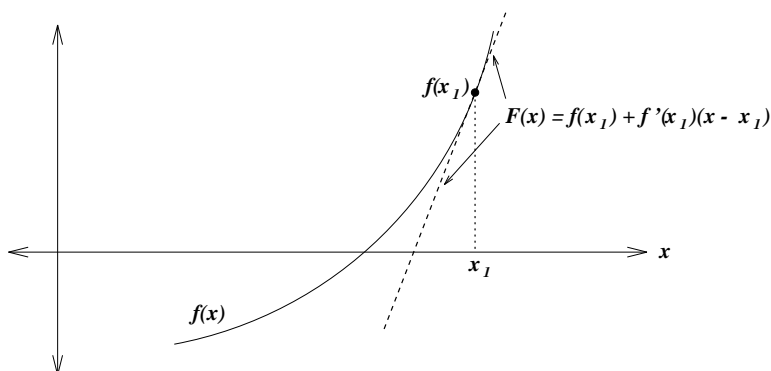


Figure 6.2: Picture of the Taylor approx at a point x_1 .

Let $f(x) \approx F(x) = f(x_1) + f'(x_1)(x - x_1)$ which is linear. Instead of looking for the root of $f(x) = 0$, look for a root of $F(x) = 0$.

$$\begin{aligned} \text{i.e., } f(x_1) + f'(x_1)(x - x_1) &= 0 \\ \Rightarrow x &= x_1 - \frac{f(x_1)}{f'(x_1)} \leftarrow \text{root of } F(x) = 0. \end{aligned}$$

Regard it as a good approximation to x^* . So, let $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$. Repeating the process, we have

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

We hope $x_n \rightarrow x^*$ as $n \rightarrow +\infty$.

The algorithm would have the form:

```
guess x;
for n = 1, 2, ... do
  x := x - f(x)/f'(x);
end;
```

Example. Want to evaluate $\sqrt{3}$. $f(x) = x^2 - 3$ and $f'(x) = 2x$.

$$x_{n+1} = x_n - \frac{x_n^2 - 3}{2x_n} = \frac{1}{2}x_n + \frac{3}{2x_n}$$

$x_1 = 1, x_2 = \frac{1}{2} + \frac{3}{2} = 2, x_3 = \frac{2}{2} + \frac{3}{4} = 1.75, x_4 = \frac{1}{2} \times 1.75 + \frac{3}{2 \times 1.75} = 1.7321$.
The exact solution is 1.73205....

Second Example. We want to now evaluate $\frac{1}{7}$. So, we want $f(x) = \frac{1}{x} - 7 = 0$,
with $f'(x) = -\frac{1}{x^2}$.

$$x_{n+1} = x_n - \left(\frac{1}{x_n} - 7 \right) \left(-x_n^2 \right) = 2x_n - 7x_n^2$$

So, by using only the $+$, $-$, and $*$ functions, we can calculate an $/$ function.

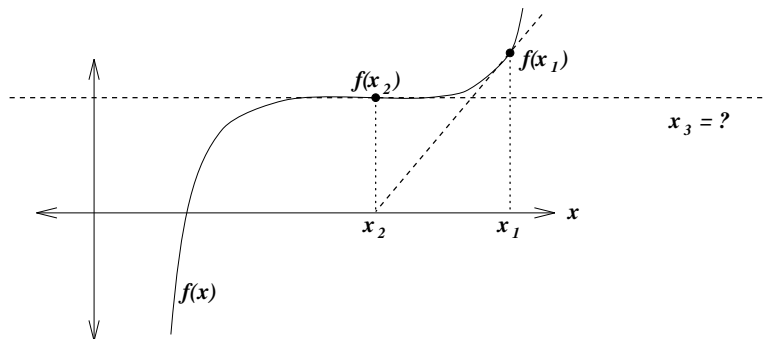


Figure 6.3: Picture of a potential problem function for Newton's Method.

Notes regarding Newton's Method:

- Need only one initial guess, whereas bisection needs a and b .
- Need to compute the derivative $f'(x)$.
- Requires that $f'(x) \neq 0$ in the neighbourhood of x^* . Otherwise, denominator blows up.
See Figure 6.3 - picture of a potential problem function.
- At each iteration evaluate $f(x)$ and $f'(x)$ (two function evaluations).

So why use this rather than bisection? — Fast.

SECANT METHOD

In Newton's method, $f'(x)$ is needed. But

- $f'(x)$ may be difficult to compute.
- may not ever know $f'(x)$; e.g., if $f(x)$ is provided by a subroutine.

Idea: do not compute $f'(x_n)$ explicitly. Instead, approximate $f'(x_n)$ as follows:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Newton	Secant
$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$	$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}} = \frac{x_{n-1}f(x_n) - x_nf(x_{n-1})}{f(x_n) - f(x_{n-1})}$

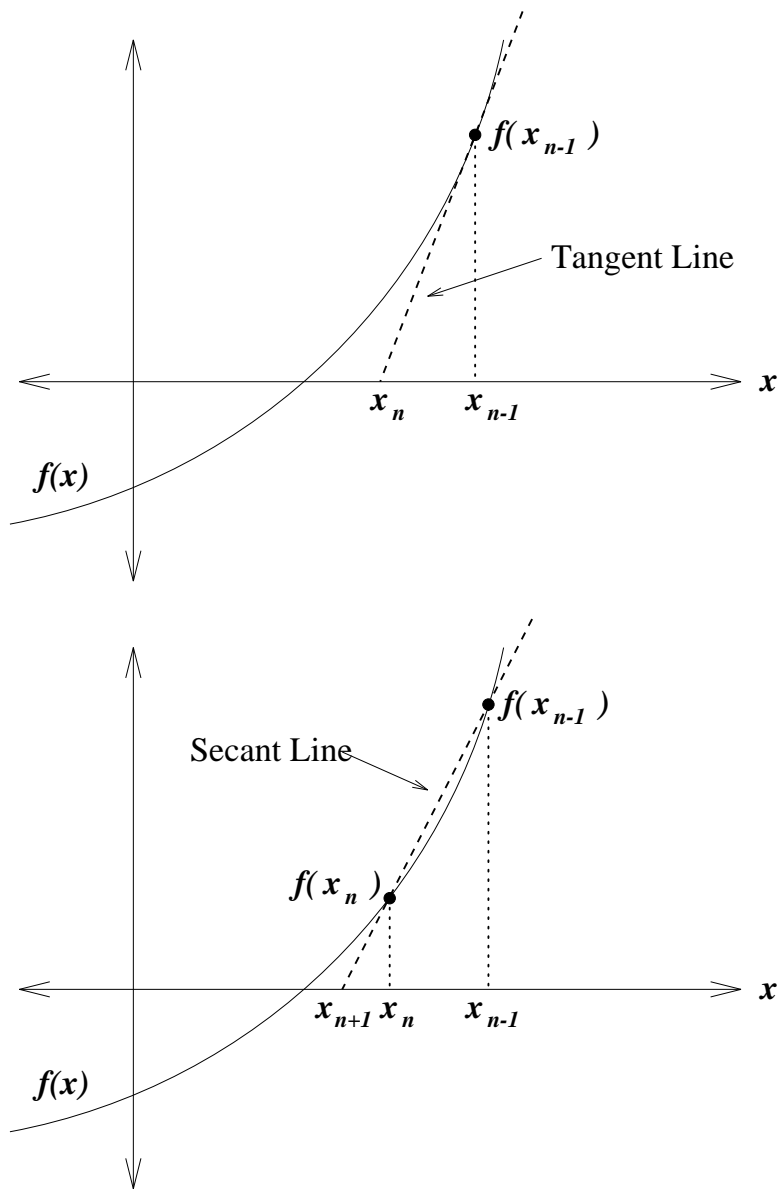


Figure 6.4: Picture comparing Newton and Secant lines on a function.

See Figure 6.4 comparing Newton and Secant lines for a function.

Observation: $\lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0} = f'(x)$. So, in the limit (which is also at a root of the function), the Secant method becomes Newton's method.

The Secant algorithm would have the form:

```
guess x_1, x_2;
for n = 1, 2, ... do
  new := (x_1*f(x_2) - x_2*f(x_1)) / (f(x_2) - f(x_1));
  x_1 := x_2; x_2 := new;
end;
```

This is a fast, single-point local iterative method. Good termination criteria are

$$|f(x_n)| \leq \epsilon_1 \quad \text{or} \quad |x_{n-1} - x_n| \leq \epsilon_2 |x_n|$$

Some comments:

- Two initial guesses are needed, but does not require $f(x_1)f(x_2) < 0$, unlike bisection. But there might be problems if the root lies outside the convergence range.
- Must have $f(x_n) \neq f(x_{n-1}) \forall n$ (similar to $f'(x_n) \neq 0$ in Newton's method). I.e., a very flat function.
- Another problem case might occur is a generated guess is the same as a previous guess, resulting in the possibility of an infinite loop that never reaches the root.
- Again, two function evaluations per iteration.

See Figure 6.5 - Picture of possible problem case for secant method.

The Secant Method is a local method and requires a pair of guesses that are reasonably close to the desired root. If they aren't, the new value can be *further* from the root (and there is no way of telling). However, once the root is bracketed, the Secant method is no worse than the False Position method, which we will discuss next.

Calculation note. For the Secant method, the basic update looks like

$$x_{n+1} = \frac{x_{n-1}f(x_n) - x_nf(x_{n-1})}{f(x_n) - f(x_{n-1})}$$

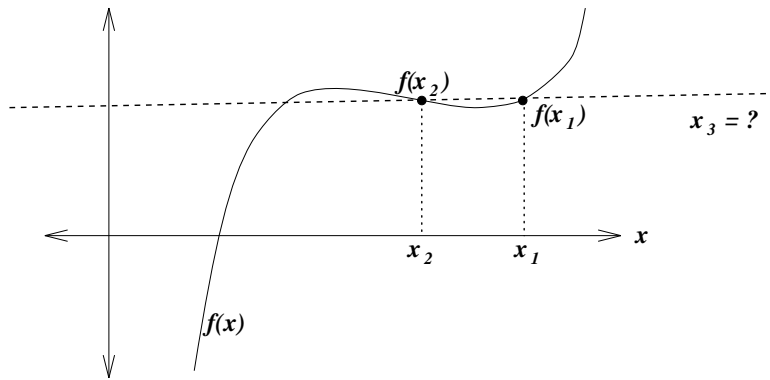


Figure 6.5: Picture of possible problem case for secant method.

Because $f(x_n)$ and $f(x_{n-1})$ may have the same sign, $f(x_n) - f(x_{n-1})$ may go to zero, due to subtractive cancellation. However, although it does not cure the problem, if it is rewritten as

$$x_{n+1} = x_n - \left\{ \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \right\} \quad \leftarrow \text{corrective term}$$

it is far more stable.

False Position Method**FALSE POSITION**

This method is also called *Regula Falsi* (method of false position). This method is similar to the Bisection method, since the root is bracketed by an interval. It is a globally convergent form of the Secant method.

The idea is to use information at two endpoints a and b to choose a better iterative value. We start with $[a, b]$ where $f(a)f(b) < 0$. Join points $(a, f(a))$, $(b, f(b))$ by a line, and let x_1 be the intersection of the line with the x -axis. Then check the sign of $f(x_1)$. If

$$\begin{aligned} f(x_1)f(a) < 0 &\Rightarrow \text{Take } [a, x_1] \text{ as the new interval} \\ f(x_1)f(a) > 0 &\Rightarrow \text{Take } [x_1, b] \text{ as the new interval} \end{aligned}$$

and repeat.

Figure 7.1 of the False Position Method at starting position.

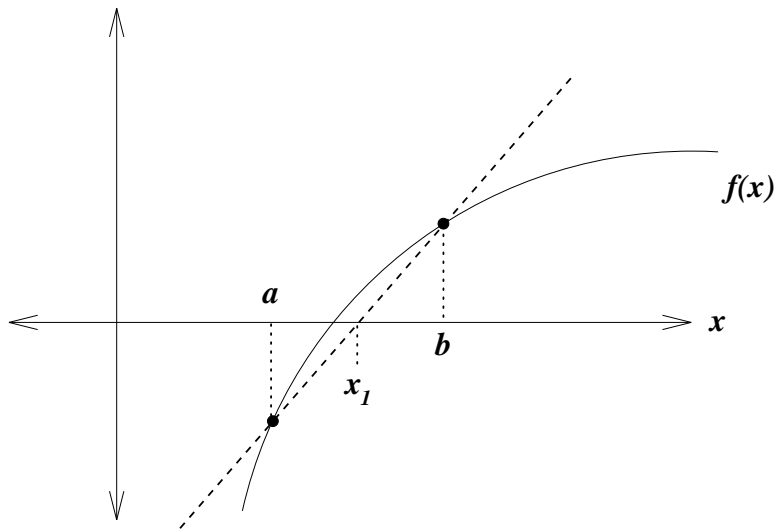


Figure 7.1: False Position Method at starting position.

Again, we have a sequence of intervals $[a_1, b_1], [a_2, b_2], \dots$. Each contains a root, however $|b_{n+1} - a_{n+1}|$ is not necessarily equal to $\frac{|b_n - a_n|}{2}$.

Line joining $(a, f(a)), (b, f(b))$:

$$y = f(a) + \frac{x-a}{b-a} (f(b) - f(a))$$

$$y = 0 \Rightarrow x_1 = a - \frac{b-a}{f(b)-f(a)} f(a) = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Bisection	False Position
$x_1 = \frac{1}{2}b + \frac{1}{2}a$	$x_1 = \frac{af(b)-bf(a)}{f(b)-f(a)} = w_1a + w_2b$ $w_1 = \frac{f(b)}{f(b)-f(a)}, w_2 = -\frac{f(a)}{f(b)-f(a)}$
average of a and b	weighted average of a, b
does not use information about $f(x)$	uses information about $f(x)$ which may give some idea of where root is located.

Algorithm (False Position)

```

find a, b, f(a)*f(b) < 0;
for n = 1,2... do
    new := (a*f(b) - b*f(a))/(f(b) - f(a));
    if sign(f(a))*f(new) > 0 then
        a := new;
    else
        b := new;
    end;
    if (convergence condition satisfied) exit for loop;
end;
return (new);

```

What is the convergence criterion? $|f(x)| \leq \epsilon_1$, but this may be impossible.
 $|(x_n - x_{n-1})/x_n| \leq \epsilon_2$, but this may take a *long* time.

How fast does it converge? In general, False position may be better, but there are exceptions. For locally convex functions, could be slower than the bisection method.

See Figure 7.2 - picture of locally convex function.

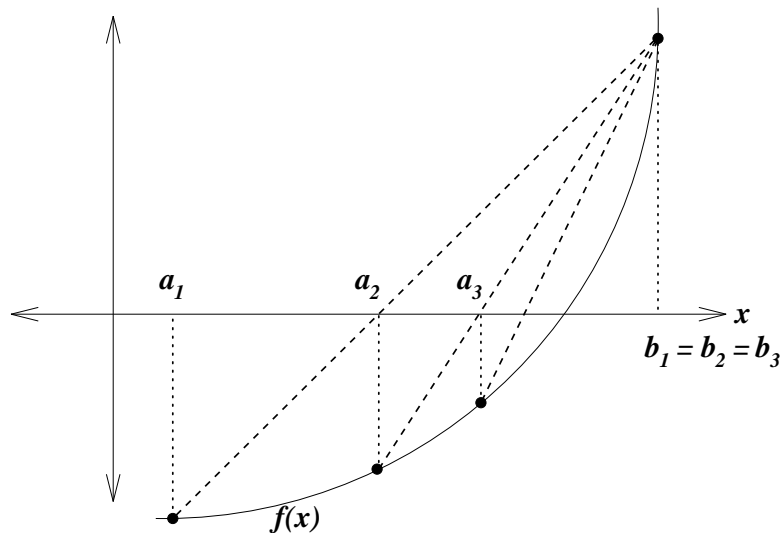


Figure 7.2: picture of locally convex function.

Extra Notes

We can consider a speed-up for the False Position method. The slow convergence of the False Position method is because the update looks like

$$[a, b] \rightarrow [a, x_1] \rightarrow [a, x_2] \rightarrow [a, x_3] \rightarrow \cdots \text{ [stationary]}$$

and a faster one is

$$[a, b] \rightarrow [x_1, b] \rightarrow [x_1, x_2] \rightarrow [x_3, x_2] \rightarrow \cdots \text{ [non-stationary]}$$

ILLINOIS METHOD

The Illinois method (or modified position method) is to use $\frac{1}{2^{i-1}}f(c)$ instead of $f(c)$ if c is a stagnant end point has been repeated twice or more, where i is the number of times the end point has been repeated.

See Figure 7.3 - A picture of the Illinois method.

This modification markedly improves the rate of convergence of the method in general.

End of Extra Notes

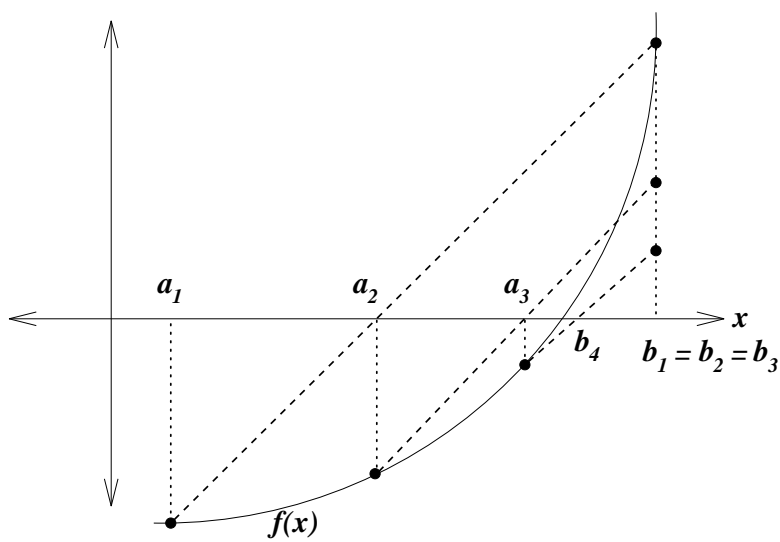


Figure 7.3: A picture of the Illinois method.

CISC 271 Class 8

Root-Finding Methods - Properties

CONVERGENCE

In all the above methods, we've found a sequence of

$$x_1, x_2, x_3, \dots$$

We would like to answer the following questions:

1. Does the sequence converge to x^* , i.e., is x_n close to x^* if n is large?

$$|x_n - x^*| \xrightarrow{?} 0 \text{ as } n \rightarrow +\infty$$

2. If the sequence is convergent, how fast does it approach to x^* ?

fast \Rightarrow few iterations needed for a given accuracy

CONDITIONS FOR CONVERGENCE

Bisection Method $- f(a)f(b) < 0$

Newton's Method $- f'(x_1) \neq 0, x_1 \text{ close to } x^*$

Secant Method $- f(x_2) - f(x_1) \neq 0, x_1, x_2 \text{ close to } x^*$

Regula Falsi/Illinois Method $- f(a)f(b) < 0$

CONVERGENCE RATE

Suppose $|x_n - x^*| \rightarrow 0$, i.e., we have a convergent method. A sequence converges, within an error constant $c_p > 0$, as

$$\lim_{n \rightarrow +\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^p} = c_p,$$

which for large n approximates to

$$|x_{n+1} - x^*| \approx c_p |x_n - x^*|^p,$$

or, if $e_n = x_n - x^*$,

$$|e_{n+1}| \approx c_p |e_n|^p.$$

Then we say that x_n converges to x^* with order p .

p – order of convergence, c_p – convergence rate

$p = 1$ – linear, $1 < p \leq 2$ – superlinear.

To see better what's happening, take logarithms – if k_n is the number of correct decimal digits in x_n (i.e., $|x_n - x^*| = 10^{-k_n} |x^*|$), then

$$\begin{aligned} 10^{-k_{n+1}} |x^*| &\approx c_p 10^{-pk_n} |x^*|^p \\ \Rightarrow -k_{n+1} + \log_{10} |x^*| &\approx -pk_n + \log_{10} c_p + p \log_{10} |x^*| \\ \Rightarrow k_{n+1} &\approx pk_n - \log_{10} c_p \end{aligned}$$

So, p is related to the number of digits we will gain after each iteration.

Example

Let's consider the order of convergence of Newton's Method.

Newton's Method is a special case of what are called Fixed-Point Methods. For the Fixed-Point Method, the error at step $n + 1$ is

$$|e_{n+1}| < K^{n-1} |e_1|$$

where $K = \max_{1 \leq i \leq n} \{|g'(\xi_i)|\} < 1$. Therefore, $p = 1$, and the Fixed-Point method is linear.

With Newton's Method, we have

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = g(x_n)$$

Since this is a special form of the Fixed-Point method, this converges if $|g'(x)| < 1$, or

$$|g'(x)| = \left| 1 - \frac{f'(x)}{f'(x)} + \frac{f(x)f''(x)}{[f'(x)]^2} \right| = \left| \frac{f(x)f''(x)}{[f'(x)]^2} \right| < 1$$

If this holds on an interval I_δ , then the method will converge. Since $x_{n+1} = g(x_n)$, then

$$x_{n+1} - x^* = g(x_n) - g(x^*).$$

Use a special form of the Taylor expansion for $g(x_n)$, expanding around x^* ,

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

where the last term is the remainder term with $\xi_n \in [x^*, x_n]$.

Now, $g'(x^*) = \frac{f(x^*)f''(x^*)}{[f'(x^*)]^2} = 0$, since $f(x^*) = 0$. So,

$$g(x_n) - g(x^*) = \frac{g''(\xi_n)}{2}(x_n - x^*)^2,$$

or, recalling that $x_{n+1} - x^* = g(x_n) - g(x^*)$,

$$x_{n+1} - x^* = e_{n+1} = \frac{g''(\xi_n)}{2}(e_n)^2.$$

Therefore,

$$|e_{n+1}| = \left| \frac{g''(\xi_n)}{2} \right| |e_n|^2.$$

So, we can see that $p = 2$ for Newton's method. Also, note that as $x_n \rightarrow x^*$, so does $\xi_n \rightarrow x^*$.

Comparison of methods:

Bisection Method:	$p = 1$
Fixed-Point Method:	$p = 1$
Regula Falsi:	$p = 1$
Illinois Method:	$p \approx 1.442$
Secant Method:	$p \approx 1.618$
Newton's Method:	$p = 2 \quad !!$

Hence, we can say that Newton's method has a quadratic order of convergence.

Note that the convergence rate is not enough, we need to account for the "cost" of each iteration of a method, ex., by counting the number of function evaluations. But, the quicker the convergence, the fewer the number of iterations needed to converge close to the root, and the less round-off error that is introduced into the solution.

STOPPING CRITERIA

Ideally, we stop when $|x_n - x^*| \leq \epsilon$. This is impossible, since we don't know x^* .

We several alternate approaches:

- (1.) We can stop if $f(x) = 0$ or nearly so (problem is solved). I.e., given a tolerance, $\epsilon_f = \text{small}$,

$$\text{stop if } |f(x_n)| \leq \epsilon_f$$

- (2.) We can stop if x_n has “converged” or nearly so. In this case, no further improvement can be made. I.e., given a tolerance ϵ_x ,

$$\text{stop if } |x_n - x_{n-1}| \leq \epsilon_x$$

- (3.) We can stop if the method is not working, i.e., x_n fails to converge (method fails). If after a large number of iterations, none of 1. or 2. is satisfied, then we should stop. I.e., given a large number M .

$$\text{stop if } n \geq M$$

If the values of either the root or the function are unknown, then use relative forms of the above stopping criteria, i.e., $|x_n - x_{n-1}| \leq \epsilon_x |x_n|$. This is especially true for methods where the root is not bracketed, i.e., Newton’s and the Secant methods.

Extra Notes

ABSOLUTE ERROR

As mentioned, the methods we have used generate a sequence of approximations $\{x_n\}$ to a true root x^* , i.e., $f(x^*) = 0$. According to the Mean Value Theorem, if $f(x)$ is continuous and differentiable,

$$f(x_n) - f(x^*) = f(x_n) - 0 = f'(\xi)(x_n - x^*) \quad \text{for some } \xi \in [x_n, x^*].$$

Thus, the absolute error $e_n = x^* - x_n$ is such that

$$|e_n| = \frac{|f(x_n)|}{|f'(\xi)|}.$$

Assuming that $f'(x^*) \neq 0$, there is a small region I_δ around x^* where

$$I_\delta = [x^* - \delta, x^* + \delta] \quad \text{and} \quad f'(x) \neq 0 \quad \forall x \in I_\delta.$$

By the extreme (minimum) value theorem, there is a minimum value $m_\delta > 0$ such that

$$|f'(x)| \geq m_\delta \quad \forall x \in I_\delta.$$

As $n \rightarrow \infty$, x_n will eventually fall into I_δ , and so will ξ . Thus

$$|e_n| \leq \frac{|f(x_n)|}{m_\delta}.$$

This m_δ is a bound on $f'(x)$ near x^* , and so we have a *method-independent* measure of the absolute error.

If m_δ is large, the problem is *well-conditioned*, and we can get very near the root. If m_δ is small, the problem is *ill-conditioned* and all methods will behave poorly near the root.

ROUND-OFF ERROR

Newton's iteration looks like $x_{n+1} = g(x_n)$, so

$$\tilde{x}_{n+1} = g(\tilde{x}_n) + \delta_n$$

where δ_n is the round-off error from true $x_{n+1} = g(x_n)$. The difference from the real root is

$$x^* - \tilde{x}_{n+1} = g(x^*) - g(\tilde{x}_n) - \delta_n$$

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

The mean value theorem gives an absolute bound of

$$x^* - \tilde{x}_{n+1} = g'(\xi_n)(x^* - \tilde{x}_n) - \delta_n$$

where $\xi_n \in [x^*, x_n]$. Subtract $g'(\xi_n)(x^* - \tilde{x}_{n+1})$ from both sides,

$$[1 - g'(\xi_n)](x^* - \tilde{x}_{n+1}) = g'(\xi_n)(\tilde{x}_{n+1} - \tilde{x}_n) - \delta_n$$

As $n \rightarrow +\infty$, the method in question converges to x^* , so $|g'(\xi_n)| < 1$.

If we can find the maximum M so that $|g'(\xi_n)| \leq M < 1$, we can get a bound on the round-off error as

$$|x^* - \tilde{x}_{n+1}| \leq \frac{M}{1 - M} |\tilde{x}_{n+1} - \tilde{x}_n| + \frac{|\delta_n|}{1 - M}.$$

If we know the real root, we can access the method.

End of Extra Notes

CISC 271 Class 9

Polynomial Interpolation

POLYNOMIALS

Examples of polynomials

$$\begin{array}{ll} x^2 - x + 1 & \text{-- degree 2 (polynomial in } x) \\ ax^4 + bx^2 - c & \text{-- degree 4} \\ at^3 + bt^2 + ct & \text{-- degree 3 (polynomial in } t) \end{array}$$

A polynomial of degree n :

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n, \quad a_n \neq 0.$$

$a_0, a_1, a_2, \dots, a_n$ are constants, and x is a variable.

A n -degree polynomial is uniquely determined by its $n + 1$ coefficients $a_0 \dots a_n$.

$$P_n \Leftrightarrow a_0 \dots a_n$$

EVALUATION OF POLYNOMIALS

Compute $P_3(x)$ at $x = 2$, where

$$P_3(x) = 5x^3 + 2x^2 - 3x + 2.$$

$$\begin{aligned} P_3(x) &= 5(2)^3 + 2(2)^2 - 3(2) + 2 = 5 \times 8 + 2 \times 4 - 3 \times 2 + 2 \\ &= 40 + 8 - 6 + 2 \\ &= 44 \end{aligned}$$

Alternative.

$$\begin{aligned} P_3(x) &= ((5x + 2)x - 3)x + 2 = ((5 \times 2 + 2) \times 2 - 3) \times 2 + 2 \\ &= (12 \times 2 - 3) \times 2 + 2 \\ &= 21 \times 2 + 2 \\ &= 44 \end{aligned}$$

$$\begin{aligned} P_n(x) &= a_0 + a_1x + \cdots + a_nx^n \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + x(\cdots (a_{n-2} + x(a_{n-1} + a_nx)) \cdots)))) \end{aligned}$$

For example,

$$P_3(x) = a_0 + x(a_1 + x(a_2 + xa_3))$$

Horner's Rule for computing $P_n(x) = a_0 + a_1x + \dots + a_nx^n$.

We can write an algorithm to evaluate a polynomial via Horner's Method, where the coefficients defining the polynomial are kept in a array $a[0..n]$.

```

y := a[n];
for i = n-1 down to 0 do
  y := a[i] + x*y;
end;
return (y);

```

At the end of the program, y has the value $P_n(x)$ for the given x .

Sometimes, a polynomial can be written as

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \dots + a_n(x - x_0)^n.$$

E.g., $P_3(x) = 2 + 3(x - 1) - 5(x - 1)^2 + 3(x - 1)^3$. In this case, Horner's rule is the same as above with $x \rightarrow (x - x_0)$.

Aside: Given a polynomial, the polynomial has a finite number of non-zero derivatives. Therefore, the Taylor series of the polynomial leads to the original polynomial since the error is zero.

POLYNOMIAL INTERPOLATION

Given $n + 1$ points

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

find a polynomial $P_n(x)$ which passes through these points. Thus we could estimate the values in between the given values. This is called the interpolation of these given points.

Figure 9.1: General example of an interpolation.

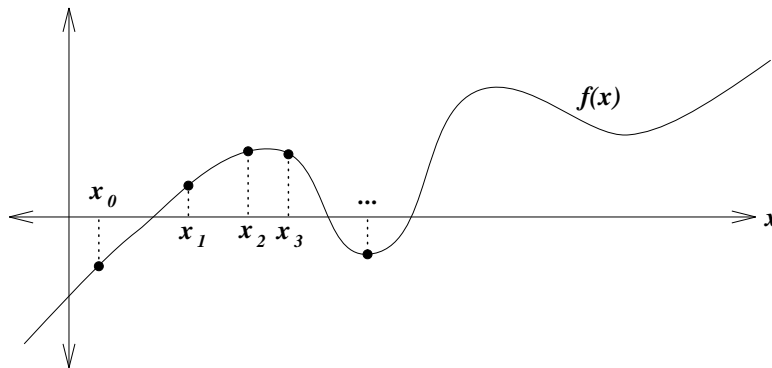


Figure 9.1: General Example

Examples.

Suppose we were given two points $\{x_0, x_1\}$ and the values at those points? We would draw a line.

Figure 9.2: Linear example.

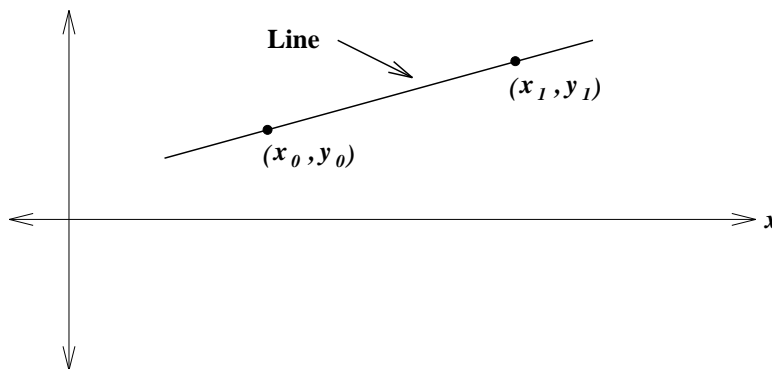


Figure 9.2: Linear example

Suppose we were given three points $\{x_0, x_1, x_2\}$? We would draw a parabola.

Figure 9.3: Parabolic interpolation example.

So, for $n = 1$, we have a line and for $n = 2$, we have a parabola. How about $n + 1$ points, $\{x_0, x_1, \dots, x_n\}$? We would then draw a n polynomial, $P_n(x)$.

Let $P_n(x) = a_0 + a_1x + \dots + a_nx^n$. The *problem* is to find a_0, a_1, \dots, a_n such that $P_n(x)$ passes through $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$.

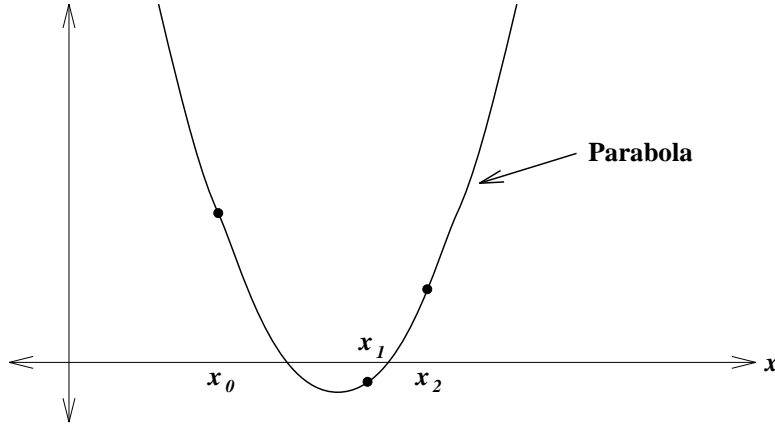


Figure 9.3: Parabolic interpolation example.

Conditions:

$$\left. \begin{array}{l} (x_0, y_0), \quad P_n(x_0) = y_0 \\ (x_1, y_1), \quad P_n(x_1) = y_1 \\ \vdots \\ (x_n, y_n), \quad P_n(x_n) = y_n \end{array} \right\} \begin{array}{l} a_0 + a_1x_0 + \cdots + a_nx_0^n = y_0 \\ a_0 + a_1x_1 + \cdots + a_nx_1^n = y_1 \\ \vdots \\ a_0 + a_1x_n + \cdots + a_nx_n^n = y_n \end{array}$$

In the above, a_0, a_1, \dots, a_n are unknowns, and $\{x_i\}$ and $\{y_i\}$ are known values. We can find the polynomial, if we solve the above for a_0, a_1, \dots, a_n .

Example. Find $P_1(x)$ passing through (x_0, y_0) and (x_1, y_1) . $P_1(x)$ has the form $P_1(x) = a_0 + a_1x$.

$$\left. \begin{array}{l} P_1(x_0) = a_0 + a_1x_0 = y_0 \\ P_1(x_1) = a_0 + a_1x_1 = y_1 \end{array} \right\} a_1(x_0 - x_1) = y_0 - y_1$$

$$\Rightarrow a_1 = \frac{y_0 - y_1}{x_0 - x_1} \quad \text{if } x_0 \neq x_1$$

$$a_0 = y_0 - a_1x_0 = y_0 - \frac{y_0 - y_1}{x_0 - x_1}x_0 = \frac{x_0y_1 - x_1y_0}{x_0 - x_1}.$$

$$P_1(x) = \frac{x_0y_1 - x_1y_0}{x_0 - x_1} + \frac{y_0 - y_1}{x_0 - x_1}x$$

If $x_0 = y_0 = 0, x_1 = y_1 = 1$, $P_1(x) = x$. I.e., the polynomial $P_1(x) = x$ passes through $(0, 0)$ and $(1, 1)$. Is this the only possible solution? Yes. Why?

Fact. For any given $n + 1$ points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, if x_0, x_1, \dots, x_n are distinct, i.e., $x_i \neq x_j$ if $i \neq j$, then there exists a unique interpolating polynomial $P_n(x)$ of degree n ; i.e., there is a unique $P_n(x)$ which passes through $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$. This can be proved by constructing a linear system of n -th order equations.

Example. If x_0, x_1, \dots, x_n are distinct. Suppose we have a polynomial $P_n(x)$ of degree n , so that

$$P_n(x_i) = 0 \text{ for } i = 0, 1, \dots, n.$$

What is $P_n(x)$? $P_n(x) = 0$, i.e., $a_0 = a_1 = \dots = a_n = 0$. Why? $P_n(x) = 0$ interpolates $(x_0, 0), (x_1, 0), \dots, (x_n, 0)$, and this is a unique interpolation of the points.

CISC 271 Class 10

Lagrange's Method

LAGRANGE POLYNOMIALS

Given distinct x_0, x_1, \dots, x_n , there is a unique polynomial of degree n passing through

$$(x_0, 1), (x_1, 0), (x_2, 0), \dots, (x_n, 0) \Rightarrow l_0^n(x).$$

See Figure 10.1: Picture of $l_0^n(x)$.

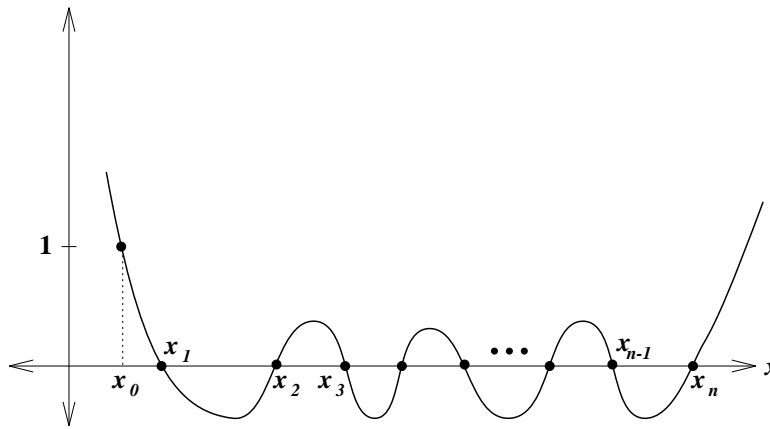


Figure 10.1: Picture of $l_0^n(x)$.

In fact, we can construct a whole set of these polynomials, each passing through 1 for a different x_i value.

$$\begin{array}{lll} l_0^n(x) & (x_0, 1), (x_1, 0), (x_2, 0), \dots, (x_n, 0) & l_0^n(x_0) = 1, l_0^n(x_1) = 0, l_0^n(x_2) = 0, \dots, l_0^n(x_n) = 0 \\ l_1^n(x) & (x_0, 0), (x_1, 1), (x_2, 0), \dots, (x_n, 0) & l_1^n(x_0) = 0, l_1^n(x_1) = 1, l_1^n(x_2) = 0, \dots, l_1^n(x_n) = 0 \\ l_2^n(x) & (x_0, 0), (x_1, 0), (x_2, 1), \dots, (x_n, 0) & l_2^n(x_0) = 0, l_2^n(x_1) = 0, l_2^n(x_2) = 1, \dots, l_2^n(x_n) = 0 \\ \vdots & & \\ l_n^n(x) & (x_0, 0), (x_1, 0), (x_2, 0), \dots, (x_n, 1) & l_n^n(x_0) = 0, l_n^n(x_1) = 0, l_n^n(x_2) = 0, \dots, l_n^n(x_n) = 1 \end{array}$$

A general short form for these polynomials is

$$l_i^n(x)$$

where n is the degree and i is the place in the set $\{x_j\}$ where it has value 1.

$$l_i^n(x_j) = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

Example. $n = 1$. we have x_0, x_1 such that

$$\begin{aligned} l_0^1(x_0) &= 1, & l_0^1(x_1) &= 0 \\ l_1^1(x_0) &= 0, & l_1^1(x_1) &= 1 \end{aligned}$$

See Figure 10.2: A picture of $l_0^1(x)$ and $l_1^1(x)$.

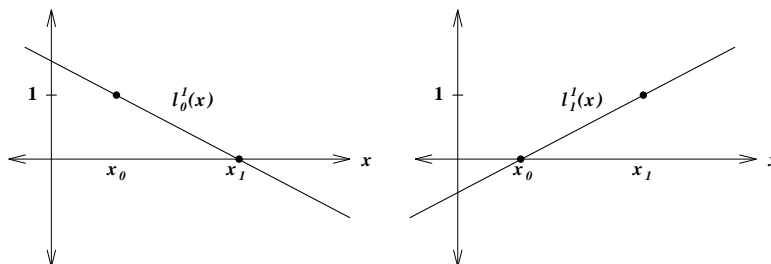


Figure 10.2: A picture of $l_0^1(x)$ and $l_1^1(x)$.

How to find $l_j^n(x)$?

$$l_0^n(x) \text{ --- degree } n = \begin{cases} 0 & \text{at } x_1, x_2, \dots, x_n \\ 1 & \text{at } x_0 \end{cases}$$

Consider the following polynomial of degree n

$$\begin{aligned} q_n(x) &= (x - x_1)(x - x_2) \cdots (x - x_n) \\ &= 0 \text{ at } x_1, x_2, \dots, x_n \end{aligned}$$

$q_n(x)$ is almost $l_0^n(x)$, but $q_n(x_0) = (x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n) \neq 1$ in general. But

$$\frac{q_n(x)}{q_n(x_0)} = \frac{(x - x_1)(x - x_2) \cdots (x - x_n)}{(x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n)} = \begin{cases} 0 & \text{at } x_1, x_2, \dots, x_n \\ 1 & \text{at } x_0 \end{cases}$$

and is a degree n polynomial.

$$\boxed{l_0^n(x) = \frac{q_n(x)}{q_n(x_0)} = \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)}}$$

This polynomial interpolates $(x_0, 1), (x_1, 0), \dots, (x_n, 0)$. Similarly

$$l_i^n(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} = \frac{\prod_{\substack{j=0 \\ j \neq i}}^n (x-x_j)}{\prod_{\substack{j=0 \\ j \neq i}}^n (x_i-x_j)}$$

interpolates $(x_0, 0), (x_1, 0), \dots, (x_i, 1), \dots, (x_n, 0)$.

Why Lagrange polynomials?

For a given $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, consider

$$P_n(x) = y_0 l_0^n(x) + y_1 l_1^n(x) + \cdots + y_n l_n^n(x)$$

where

1. $P_n(x)$ has degree n .
2. $P_n(x_i) = y_i$.

In other words, $P_n(x)$ is the interpolating polynomial for $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$.

LAGRANGE FORMULA

The interpolating polynomial for $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ is given by

$$\boxed{P_n(x) = y_0 l_0^n(x) + y_1 l_1^n(x) + \cdots + y_n l_n^n(x) = \sum_{i=0}^n y_i l_i^n(x)}$$

provided $x_i \neq x_j, i \neq j$.

What does this interpolating formula look like? Consider $n = 1$.

$$P_1(x) = y_0 \frac{(x-x_1)}{(x_0-x_1)} + y_1 \frac{(x-x_0)}{(x_1-x_0)}$$

INTERPOLATING FUNCTIONS BY POLYNOMIALS.

If we have a complicated function $f(x)$, we may want to approximate it by a polynomial of degree n , $P_n(x)$.

See Figure 10.3: A picture of general example.

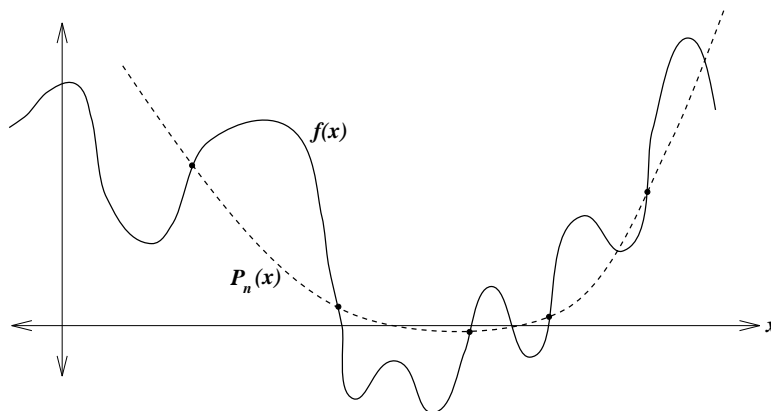


Figure 10.3: A picture of general example.

How to approximate this function, $f(x)$?

We require $P_n(x)$ and $f(x)$ to have the same values at some given set of $\{x_i\}$, x_0, x_1, \dots, x_n . I.e., $P_n(x_i) = f(x_i)$, $i = 0, 1, 2, \dots, n$.

Therefore, $P_n(x)$ must interpolate $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$.

Use the Lagrange formula,

$$P_n(x) = \sum_{i=0}^n f(x_i) l_i^n(x)$$

This is a polynomial of degree n which interpolates $f(x)$ at x_0, x_1, \dots, x_n .

Example. Suppose a function $f(x)$ is given by the following table

i	0	1	2	3
x_i	0	1	3	4
$f(x_i)$	3	2	1	0

Find the interpolating polynomial and use it to approximate the value of $f(2.5)$.

1. Find the Lagrange polynomials.

$$l_0^3(x) = \frac{(x-1)(x-3)(x-4)}{(-1)(-3)(-4)}, \quad l_1^3(x) = \frac{(x-0)(x-3)(x-4)}{(1-0)(1-3)(1-4)}$$

$$l_2^3(x) = \frac{(x-0)(x-1)(x-4)}{(3-0)(3-1)(3-4)}, \quad l_3^3(x) = \frac{(x-0)(x-1)(x-3)}{(4-0)(4-1)(4-3)}$$

2. Find interpolating polynomial.

$$\begin{aligned}P_3(x) &= 3l_0^3(x) + 2l_1^3(x) + 1l_2^3(x) + 0l_3^3(x), \\&= 3\frac{(x^3-8x^2+19x-12)}{-12} + 2\frac{(x^3-7x^2+12x)}{6} + \frac{(x^3-5x^2+4x)}{-6} \\&= \frac{(-x^3+6x^2-17x+36)}{12}.\end{aligned}$$

3. Use $P_3(2.5)$ to estimate $f(2.5)$,

$$P_3(2.5) = \frac{-(2.5)^3 + 6(2.5)^2 - 17(2.5) + 36}{12} = 1.28125$$

Therefore, $f(2.5) \approx 1.28125$.

CISC 271 Class 11

Newton Divided Differences

There are two problems with Lagrange's form for the unique interpolating formula:

1. It is expensive computationally.
2. If we have $P_n(x)$, we can't use it to find $P_{n+1}(x)$.

The Lagrange formulation

$$P_n(x) = \sum_{i=0}^n f(x_i) l_i^n(x)$$

is simple in form, but it is difficult to compute the coefficients. So, we will look for another form for $P_n(x)$. Note that we are not looking for another polynomial, since there is only one unique interpolating polynomial. What we are looking for is another form to express the same polynomial, that is easier to compute.

We write the interpolating polynomial in the following form:

$$P_n(x) = A_0 + A_1(x-x_0) + A_2(x-x_0)(x-x_1) + \cdots + A_n(x-x_0)(x-x_1) \cdots (x-x_{n-1}).$$

And try to determine the coefficients A_0, A_1, \dots, A_n .

$$\begin{aligned} (x_0, f(x_0)) \quad P_n(x_0) = f(x_0) &\Rightarrow A_0 = f(x_0) \\ (x_1, f(x_1)) \quad P_n(x_1) = f(x_1) &\Rightarrow \begin{cases} f(x_1) = f(x_0) + A_1(x_1 - x_0) \\ \Rightarrow A_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{cases} \\ (x_2, f(x_2)) \quad P_n(x_2) = f(x_2) &\Rightarrow A_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} \\ &A_3, A_4 \text{ — too complicated} \end{aligned}$$

NEW NOTATION

We can note in the above expressions for A_1 and A_2 a relationship in the forms of the expressions, which leads us to the following new notation.

$$\begin{aligned}
f[x_0] &= f(x_0) & \Rightarrow & A_0 = f[x_0] \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} & \Rightarrow & A_1 = f[x_0, x_1] \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} & \Rightarrow & A_2 = f[x_0, x_1, x_2] \\
&\text{etc.}
\end{aligned}$$

We call $f[x_1, x_2] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$ the *divided difference* at $[x_1, x_2]$, etc.

Thus, the polynomial which interpolates

$$(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$$

can be written as

$$\begin{aligned}
P_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\
&\quad + \dots + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \dots (x - x_{n-1}) \\
&= P_{n-1}(x) + f[x_0, x_1, \dots, x_n] \prod_{i=0}^{n-1} (x - x_i)
\end{aligned}$$

$$\left. \begin{aligned}
f[x_0] &= f(x_0) \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\
f[x_0, x_1, \dots, x_i] &= \frac{f[x_1, x_2, \dots, x_i] - f[x_0, x_1, \dots, x_{i-1}]}{x_i - x_0} \\
f[x_0, x_1, \dots, x_n] &= \frac{f[x_1, x_2, \dots, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{x_n - x_0}
\end{aligned} \right\} \begin{array}{l} \text{Newton's} \\ \text{Divided} \\ \text{Difference} \end{array}$$

We can build a divided difference table very easily:

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4}]$
x_0	$f[x_0]$				
x_1	$f[x_1]$	$f[x_0, x_1]$			
x_2	$f[x_2]$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$		
x_3	$f[x_3]$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	$f[x_0, x_1, x_2, x_3]$	
x_4	$f[x_4]$	$f[x_3, x_4]$	$f[x_2, x_3, x_4]$	$f[x_1, x_2, x_3, x_4]$	$f[x_0, x_1, x_2, x_3, x_4]$

Example. Find the interpolating function for the following table

i	0	1	2	3
x_i	0	1	3	4
$f(x_i)$	3	2	1	0

1. Find Newton's divided difference.

x_i	$f[x_i]$			
0	3			
1	2	-1	$\frac{1}{6}$	$-\frac{1}{12}$
3	1	$-\frac{1}{2}$	$-\frac{1}{6}$	
4	0	-1		

2. Find the interpolating function.

$$P_3(x) = 3 + (-1)(x-0) + \frac{1}{6}(x-0)(x-1) + \left(-\frac{1}{12}\right)(x-0)(x-1)(x-3)$$

COMPUTING THE DIVIDED DIFFERENCE

Given $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$, how can we compute $f[x_0], f[x_0, x_1], \dots, f[x_0, x_1, \dots, x_n]$?

We use two vectors \vec{x}, \vec{y} . Their initial values are

$$\begin{aligned}\vec{x} &= (x_0, x_1, \dots, x_n) \\ \vec{y} &= (f(x_0), f(x_1), \dots, f(x_n))\end{aligned}$$

I.e., $y_0 = f(x_0), \dots, y_n = f(x_n)$

$$\begin{array}{ccccccccc} & & & & & \xrightarrow{j} & & & & \\ & & & & & & & & & \\ i \uparrow & x_0 & y_0 & & y_1 & & y_2 & & y_3 & \\ & x_1 & y_1 & & y_2 & & y_3 & & & \\ & x_2 & y_2 & & y_3 & & & & & \\ & x_3 & y_3 & & & & & & & \\ & & & & & & & & & \\ & j = 1 & j = 2 & j = 3 & & & & & & \\ \hline & x_i - x_{i-1} & x_i - x_{i-2} & x_i - x_{i-3} & & & & & & \end{array}$$

After the first column is completed: $y_0 = f[x_0], y_1 = f[x_0, x_1]$

After the second column is completed: $y_0 = f[x_0], y_1 = f[x_0, x_1], y_2 = f[x_0, x_1, x_2]$

After the third column is completed: $y_0 = f[x_0], y_1 = f[x_0, x_1], \dots, y_3 = f[x_0, x_1, x_2, x_3]$

and then we have all Newton's Divided Differences.

Algorithm

```
let y[0] := f(x[0]), y[1] := f(x[1]), ..., y[n] := f(x[n]);
for j = 1, 2, ..., n do
  for i = n, n-1, ..., j do
    y[i] = (y[i] - y[i-1]) / (x[i] - x[i-j]);
  end;
end;
```

The result of this algorithm is that \vec{y} contains the divided difference

$$y_i = f[x_0, x_1, \dots, x_i]$$

Recall Horner's rule,

$$\begin{aligned} P_n(x) &= y_0 + y_1(x - x_0) + y_2(x - x_0)(x - x_1) + \cdots + y_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}) \\ &= y_0 + (x - x_0)(y_1 + (x - x_1)(y_2 + \cdots)) \end{aligned}$$

This requires only half as many multiplications as the original.

SUMMARY

The polynomial of degree n which interpolates

$$(x_0, f(x_0), (x_1, f(x_1)), \dots, (x_n, f(x_n)))$$

is given by

$$P_n(x) = f(x_0)l_0^n(x) + \cdots + f(x_n)l_n^n(x) = \sum_{i=0}^n f(x_i)l_i^n(x) \text{ --Lagrange formula}$$

||

$$\begin{aligned} P_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}) \\ &= \sum_{i=0}^n f[x_0, x_1, \dots, x_i](x - x_0)(x - x_1) \cdots (x - x_{i-1}) \text{ --Newton's Divided Difference} \end{aligned}$$

CISC 271 Class 12

Finite Difference, Errors

Consider if the points $\{x_i\}$ are evenly spaced. Let h be the fixed distance between the points. Then we can define

$$\begin{aligned}\Delta f(x_i) &= f(x_i + h) - f(x_i) \\ &= f(x_{i+1}) - f(x_i) \\ \text{or } \Delta f_i &= f_{i+1} - f_i, \quad f_i = f(x_i)\end{aligned}$$

This quantity is called the forward difference of $f(x)$ at x_i . Since the points are evenly spaced, $x_i = x_0 + ih$, $i = 0, 1, 2, \dots, n$.

For $r \geq 0$, we can further define

$$\Delta^{r+1} f_i = \Delta^r f_{i+1} - \Delta^r f_i,$$

with $\Delta^0 f_i = f_i$. For example,

$$\Delta^2 f_i = \Delta(\Delta f_i) = \Delta(f_{i+1} - f_i) = \Delta f_{i+1} - \Delta f_i = (f_{i+2} - f_{i+1}) - (f_{i+1} - f_i) = f_{i+2} - 2f_{i+1} + f_i$$

Now, let us consider the form of the Newton Divided Difference with evenly spaced points.

$$\begin{aligned}f[x_0, x_1] &= \frac{f_1 - f_0}{x_1 - x_0} \\ &= \frac{1}{h} \Delta f_0 \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= \frac{1}{2h} \left(\frac{1}{h} \Delta f_1 - \frac{1}{h} \Delta f_0 \right) \\ &= \frac{1}{2h^2} \Delta^2 f_0 \\ &\vdots\end{aligned}$$

In general, and this can be easily proved via proof by induction,

$$f[x_0, x_1, \dots, x_k] = \frac{1}{k! h^k} \Delta^k f_0$$

We can now modify the Newton interpolation formula to an interpolation formula based on forward differences. Since the polynomial is defined continuously, rather than with respect to the discretely spaced points, we will define for the value x at which the polynomial is defined,

$$\mu = \frac{x - x_0}{h},$$

where μ is a continuous parameter.

Therefore,

$$x - x_i = x_0 + \mu h - x_0 - ih = (\mu - i)h$$

which leads to the following form for the interpolating formula

$$P_n(x) = \sum_{i=0}^n \binom{\mu}{i} \Delta^i f_0,$$

where we have used the *binomial coefficients*

$$\binom{\mu}{i} = \frac{\mu(\mu-1)\cdots(\mu-i+1)}{i!}, \quad i > 0$$

and $\binom{\mu}{0} = 1$.

For example, $n = 1$.

$$P_1(x) = f_0 + \mu \Delta f_0$$

As with Newton divided differences, we can easily construct tables to evaluate the forward differences.

x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$	\cdots
x_0	f_0				
x_1	f_1	Δf_0			
x_2	f_2	Δf_1	$\Delta^2 f_0$	$\Delta^3 f_0$	
x_3	f_3	Δf_2	$\Delta^2 f_1$	$\Delta^3 f_1$	\vdots
x_4	f_4	Δf_3	$\Delta^2 f_2$		
\vdots	\vdots				

Example. Find the interpolating function for the following table

i	0	1	2	3
x_i	0	1	2	3
$f(x_i)$	3	2	0	-1

1. Find the forward differences.

x_i	$f[x_i]$	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$
0	3			
1	2	-1		
2	0	-2	-1	
3	-1	-1	1	2

2. Find the interpolating function.

$$P_3(x) = 3 + (-1)(\mu) + (-1)\frac{(\mu)(\mu-1)}{2} + (2)\frac{(\mu)(\mu-1)(\mu-2)}{6}$$

Note: forward differences of order greater than three are almost entirely the result of differencing the rounding errors in the table entries; therefore, interpolation in this table should be limited to polynomials of degree less than four. (See example in Atkinson text, p. 151; G & W, p. 232).

As you can see, there is nothing particularly special about *forward* differences. We can equally define *backward* difference interpolating functions based on

$$\nabla f_i = f_i - f_{i-1}.$$

Extra Notes

ERRORS IN DATA AND FORWARD DIFFERENCES

One use of the finite differences is the detection of noise in data, when the noise is large with respect to the rounding errors or uncertainty errors of physical measurement.

First, let us consider a property of the forward differences, which derives from the fact that they are linear:

$$\Delta^r(\alpha f(x) + \beta g(x)) = \alpha \Delta^r f(x) + \beta \Delta^r g(x)$$

This can easily be proved using proof by induction.

Let $\{\tilde{f}_i\}$ be our experimental results, and $\{e_i = e(x_i)\}$ be an error larger than rounding error, and $\{f_i\}$ be our desired function. Therefore,

$$\tilde{f}_i = f_i - e_i$$

with \tilde{f}_i being our table value that we used to construct our difference table. Then

$$\begin{aligned}\Delta^r \tilde{f}_i &= \Delta^r f_i - \Delta^r e_i \\ &= h^r r! f[x_i, \dots, x_{i+r}] - \Delta^r e_i \\ &= h^r r! \frac{f^{(r)}(\xi_i)}{r!} - \Delta^r e_i \\ &= h^r f^{(r)}(\xi_i) - \Delta^r e_i\end{aligned}$$

where $\xi_i \in (x_i, \dots, x_{i+r})$.

So the first term becomes smaller as r increases, as we saw in an earlier forward difference table. But what is the behaviour of the error? Consider an error of the following form:

$$e_i = \begin{cases} 0 & i \neq k \\ \epsilon & i = k \end{cases}$$

The forward difference table for this function is:

x_i	e_i	Δe_i	$\Delta^2 e_i$	$\Delta^3 e_i$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	
x_{k-4}	0				
x_{k-3}	0	0	0	0	
x_{k-2}	0	0	0	ϵ	
x_{k-1}	0	ϵ	ϵ	-3ϵ	\dots
x_k	ϵ	$-\epsilon$	-2ϵ	3ϵ	
x_{k+1}	0	0	ϵ	$-\epsilon$	
x_{k+2}	0	0	0	0	
x_{k+3}	0	0	0		
x_{k+4}	0				
\vdots	\vdots	\vdots	\vdots	\vdots	\dots

Therefore, the effect of the single rounding error will propagate and increase in value as the larger order differences are calculated. But rounding errors are a general error function composed of a sum of the above error function at each x_i . As their differences grow in size, the higher order $\Delta^r \tilde{f}_i$ become dominated by the rounding errors (especially when they start growing in size).

NOISE IN DATA

Suppose our data has an isolated error that dominates the rounding errors. We would then look for a pattern like that above for a single e_i . Consider the following example:

\tilde{f}_i	$\Delta \tilde{f}_i$	$\Delta^2 \tilde{f}_i$	$\Delta^3 \tilde{f}_i$	Error Guess	Guess $\Delta^3 \tilde{f}_i$
.10396	.01700				
.12096	.01686	-.00014			
.13782	.01669	-.00017	-.00003	0	-.00003
.15451	.01650	-.00019	-.00002	0	-.00002
.17101	.01637	-.00013	.00006	ϵ	-.00002
.18738	.01599	-.00038	-.00025	-3ϵ	-.00002
.20337	.01582	-.00017	.00021	3ϵ	-.00002
.21919	.01555	-.00027	-.00010	$-\epsilon$	-.00002
.23474					

Using $r = 3$ and one of the errors is chosen randomly, say the first, we arrive at a guess of $\epsilon = -.00008$. We could have guessed $\epsilon = -.00007$. Therefore, we can correct one of the entries:

$$f_i = \tilde{f}_i + e_i = .18738 + (-.00008) = .18730$$

If there are more errors, their results might overlap.

End of Extra Notes

CISC 271 Class 14

Interpolation Errors

In all the above we have been using a polynomial $P_n(x)$ to interpolate and approximate a function $f(x)$. Why should we use a polynomial? Because

Weierstrass Approximation Theorem. If $f(x)$ is continuous on a finite interval $[a, b]$, \exists a polynomial $P_n(x)$ of degree n such that

$$|f(x) - P_n(x)| < \epsilon,$$

throughout the interval $[a, b]$, for any given $\epsilon > 0$. (The degree required of $P_n(x)$ is a function of ϵ).

So, we can get uniform approximation using polynomials.

We have discussed the order of the polynomial being n for $n + 1$ given points. But why is the polynomial unique? Consider the following argument by contradiction:

Suppose there are two different polynomials of degree n interpolating the same $n + 1$ points. Call these polynomials $P_n(x)$ and $Q_n(x)$. Their difference is a polynomial of at most degree n :

$$D(x) = P_n(x) - Q_n(x).$$

Of course, $D(x)$ is zero at the given $n + 1$ points, such that $D(x)$ is a polynomial of at degree at most n with $n + 1$ distinct zeros. But this is not possible unless $D(x)$ is identically zero. Hence $P_n(x)$ and $Q_n(x)$ are the same polynomial.

One consequence of this result is that if we have a unique function $f(x)$ and a unique interpolating polynomial $P_n(x)$, then we also have a unique error function:

$$E(x) = f(x) - P_n(x).$$

PROBLEMS IN POLYNOMIAL INTERPOLATION

We know

of interpolating point \sim degree of polynomial n

So with

more points \Rightarrow more places where $P_n(x) = f(x)$

But is it true that

as $n \rightarrow +\infty \stackrel{?}{\Rightarrow} P_n(x)$ agrees with $f(x)$ everywhere on $[a, b]$?

I.e., $|P_n(x) - f(x)| \stackrel{?}{\longrightarrow} 0$ as $n \rightarrow +\infty$

This depends on

1. the function $f(x)$ and
2. the interpolating points x_0, x_1, \dots, x_n .

But, as a rule,

$$|P_n(x) - f(x)| \not\rightarrow 0$$

Reason: A polynomial of degree n has $n - 1$ turning points (i.e., the number of connecting segments define unique directions). If the degree is very high, the polynomial turns up and down very quickly. High degree polynomials tend to have “wiggles” or oscillations. We should try to avoid using high degree polynomials.

Therefore, we end up with two main problems:

1. The error is ill-behaved and
2. The polynomial is not what you might think.

Classical Example

A bell shaped function (Runge):

$$f(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1]$$

If x_0, x_1, \dots, x_n are evenly spaced (i.e., sampled uniformly), then

$$\max_{x \in [-1, 1]} |P_n(x) - f(x)| \rightarrow +\infty, \quad \text{as } n \rightarrow +\infty$$

The behaviour between the sampled points grows without limit.

See Figure 14.1: A picture of this function, and an interpolating polynomial.

This leads us to two unusual results:

1. If $f'(x)$ is bounded on $[-1, 1]$, then the sampling sequence

$$S_n = \{x_i | x_i = -\cos\left(\frac{i}{n}\pi\right), i = 0, \dots, n\}$$

gives an approximating polynomial $P_n(x)$, using the same interpolation method, that uniformly converges on $[-1, 1]$. These are called *Chebyshev* points.

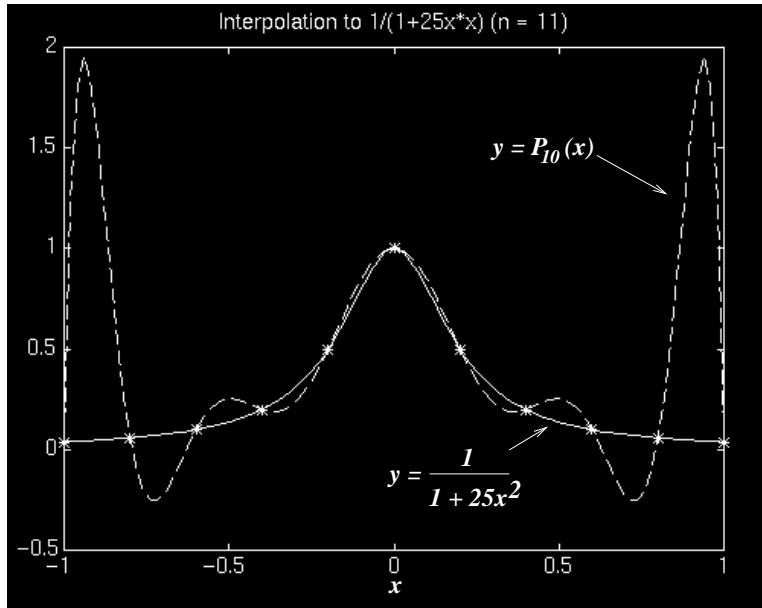


Figure 14.1: A picture of this function, and an interpolating polynomial.

2. For any sampling sequence S_n on $[a, b]$ where $i \neq j \Rightarrow x_i \neq x_j$, there exists a continuous function $f(x)$ on $[a, b]$ that cannot be approximated, i.e., $\lim_{n \rightarrow \infty} (P_n(x) - f(x)) = \infty$.

ROUND-OFF ERRORS

The value at $f(x_i)$ can only be approximated, within a round-off error of ϵ_i ,

$$\tilde{f}_i = f(x_i) + \epsilon_i.$$

This round-off then propagates through the differencing, because

$$\begin{aligned} \tilde{f}[x_0] &= \tilde{f}_0 \\ \tilde{f}[x_0, x_1] &= \frac{\tilde{f}_1 - \tilde{f}_0}{\tilde{x}_1 - \tilde{x}_0} \end{aligned}$$

and so on, so our ideal polynomial, in divided difference form,

$$P_n(x) = \sum_{k=0}^n \left(f[x_0, x_1, \dots, x_k] \prod_{i=0}^{k-1} (x - x_i) \right)$$

is deviated.

If we supposed that the points are equally spaced by h , and let ϵ be the largest ϵ_i , we can derive

$$\max_{a \leq x \leq b} |P_n(x) - \tilde{P}_n(x)| \leq \frac{\epsilon}{2}(2^n + 1)$$

and finally,

$$|f(x) - \tilde{P}_n(x)| \leq \frac{h^{n+1}M_{n+1}}{4(n+1)!} + \frac{\epsilon(2^n + 1)}{2}$$

where M_{n+1} is the discretization error term.

Comment: errors can be decreased if the x value to be approximated is centered on the points used for the interpolation.

CISC 271 Class 15

Piecewise Polynomials

In general, what we observed before was that the polynomial approximation error decreases as we add points, and then begins to worsen. To apply this form of an approximation to a large number of points, we *locally* interpolate. E.g., if quadratic interpolation has the best error, then find an interval $[x_i, x_{i+2}]$ containing x_{i+1} and use those three points.

Fact: degree of interpolating polyn'l = # of interpolating points - 1

Want:	degree = small (avoid oscillating)	# of points = large (good approximation)
-------	---------------------------------------	---

So, this is the idea. We have $n + 1$ points, $x_0, \dots, x_n \in [a, b]$. If we use one polynomial to interpolate all the points, we have a polynomial of degree n . But suppose that we break up $[a, b]$ into m pieces:

$$[a_1, b_1], [a_2, b_2], \dots, [a_m, b_m],$$

with $n_1 + 1$ points in $[a_1, b_1]$, $n_2 + 1$ points in $[a_2, b_2]$, ..., $n_m + 1$ points in $[a_m, b_m]$. Each n_i is much smaller than n . Also, $b_i = a_{i+1}$.

See Figure 15.1 for a picture of the subdivision of the original interval.

Use a polynomial $P_{n_1}(x)$ to interpolate all points in $[a_1, b_1]$	$\Rightarrow P_{n_1}(x)$, degree = n_1
Use a polynomial $P_{n_2}(x)$ to interpolate all points in $[a_2, b_2]$	$\Rightarrow P_{n_2}(x)$, degree = n_2
\vdots	\vdots
Use a polynomial $P_{n_m}(x)$ to interpolate all points in $[a_m, b_m]$	$\Rightarrow P_{n_m}(x)$, degree = n_m

Thus the polynomials $P_{n_1}(x), P_{n_2}(x), \dots, P_{n_m}(x)$ are low degree polynomials, but none of $\{P_{n_i}(x)\}$ interpolates all of the points x_0, x_1, \dots, x_n .

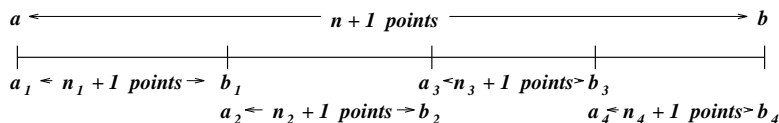


Figure 15.1: A picture of the subdivision of the original interval.

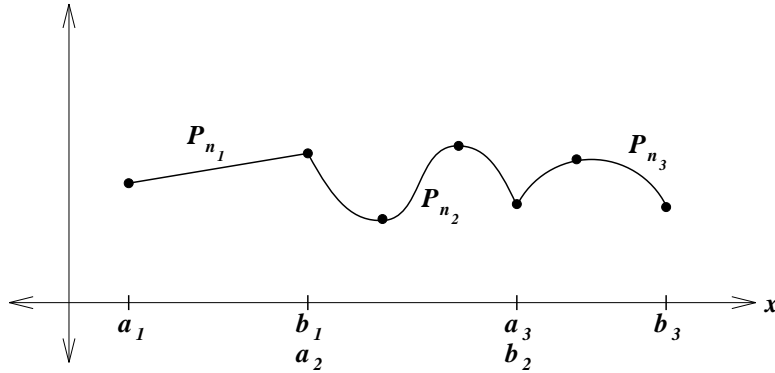


Figure 15.2: Picture of the possible subintervals and interpolating polynomials.

Define a new function $P(x)$

$$P(x) = \begin{cases} P_{n_1}(x) & x \in [a_1, b_1] \\ P_{n_2}(x) & x \in [a_2, b_2] \\ \vdots & \vdots \\ P_{n_m}(x) & x \in [a_m, b_m] \end{cases}$$

1. $P(x)$ is a polynomial on each subinterval $[a_i, b_i]$, with possibly different degrees in each subinterval, but not necessarily a polynomial on the total interval $[a, b]$.

See Figure 15.2 for a picture of the possible subintervals and corresponding interpolating polynomials.

2. $P(x)$ interpolates all points x_0, x_1, \dots, x_n .

Definition: A function $P(x)$ is said to be a piecewise polynomial on $[a, b]$ if there are points $a = z_0 < z_1 < \dots < z_m = b$ such that $P(x)$ is a polynomial on $[z_i, z_{i+1}]$ for each i .

Suppose on the interval $[z_i, z_{i+1}]$,

$$P(x) = P_{n_i}(x), \quad \text{degree} = n_i,$$

how to compute $P(\tilde{x})$ for $\tilde{x} \in [a, b]$?

- (a) locate the interval $[z_i, z_{i+1}]$ which contains \tilde{x} . I.e.,

$$\text{find } i \text{ such that } z_i \leq \tilde{x} \leq z_{i+1}$$

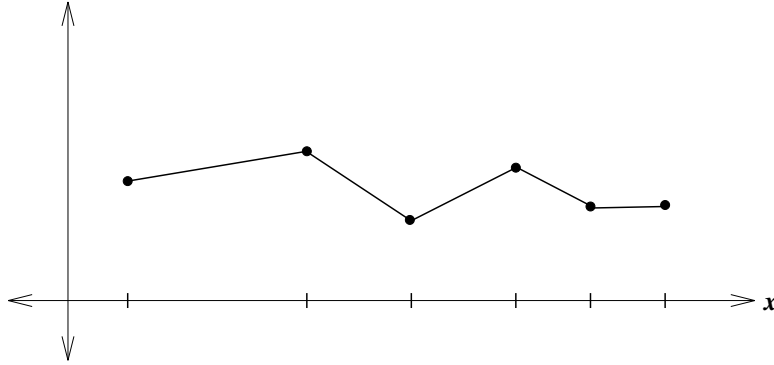


Figure 15.3: Picture of linear interpolation.

(b) apply Horner's method to compute the polynomial

$$P_{n_i}(\tilde{x})$$

Let us consider an example with linear interpolation, where each interval between points $\{x_i\}$ is interpolated by a straight line.

See Figure 15.3 for a picture of a linear interpolation of a set of points.

For a given pair of points, we define the interpolation over $[x_i, x_{i+1}]$

$$g_2(x) = f(x_i) + f[x_i, x_{i+1}](x - x_i),$$

where $g_2(x)$ is a linear interpolation of two points.

From our previous theorems, we have

$$f(x) - g_2(x) = \frac{f''(\xi_i)}{2}(x - x_i)(x - x_{i+1})$$

for some $\xi_i \in (x_i, x_{i+1})$. Suppose M_2 bounds $f''(x)$ on $[x_i, x_{i+1}]$, and let $h = |x_{i+1} - x_i|$. Then the largest possible value for $(x - x_i)(x - x_{i+1})$ is $\frac{h}{2} \cdot \frac{h}{2}$, so

$$|f(x) - g_2(x)| \leq \frac{M_2}{8}h^2.$$

Thus, to increase accuracy, we just need to add more points.

Three observations:

1. For a k th-order piecewise polynomial, the error bound is

$$\max_{a \leq x \leq b} |f(x) - g_k(x)| \leq M_k \frac{h^2 ((1 + 1/2)h)((2 + 1/2)h) \cdots ((k - 1/2)h)}{4(k+1)!} \leq \frac{h^{k+1}}{4(k+1)} M_k$$

So, these become more accurate as the order increases, as long as M_k is well-behaved as k gets larger.

2. At the *knots*, those x_i where we switch polynomials, we may have a completely useless estimate of $f'(x)$.

E.g., linear and quadratic interpolations have *cusps* at [some] points. See for example, figure 15.3.

3. These are essentially local, and information outside each subinterval has no effect.

CISC 271 Class 16

Cubic Splines

Cubic spline interpolates address these latter two problems mentioned at the end of Class 22. The major problem with the previous piecewise interpolates is that they are not smooth everywhere on $[a, b]$.

A cubic spline, $S(x)$, is a piecewise polynomial such that

1. $S(x)$ = polynomial of degree 3 on each subinterval $[x_i, x_{i+1}]$.
2. $S(x), S'(x), S''(x)$ are continuous on (a, b) .

Interpolation by Cubic Splines

Given $x_0 < x_1 < \dots < x_n$, find a cubic spline $S(x)$ which interpolates $f(x)$ at $\{x_i\}$.

Here we take x_i as the end points of the subintervals. I.e.,

$$[a, b] = [x_0, x_1] \cup [x_1, x_2] \cup \dots \cup [x_{n-1}, x_n] \quad n \text{ intervals in total.}$$

So, given $x_0 < x_1 < \dots < x_n, f(x_0), f(x_1), \dots, f(x_n)$, find $S(x)$.

1. $S(x_i) = f(x_i), \quad i = 0, 1, \dots, n$.
2. $S(x)$ = a polynomial of degree 3 on each interval $[x_i, x_{i+1}], i = 0, 1, \dots, n - 1$.
3. $S(x)$ is smooth, in that $S(x), S'(x), S''(x)$ are continuous on (a, b) .

So, on each $[x_i, x_{i+1}]$,

$$S(x) = a_i + b_i x + c_i x^2 + d_i x^3$$

So, $S(x)$ is determined by $a_i, b_i, c_i, d_i, i = 0, 1, \dots, n - 1$. Therefore, since we have 4 unknown coefficients in each of the n subintervals, we have $4n$ unknowns in total.

Equations

1. $S(x)$ interpolates $(x_0, f(x_0)), \dots, (x_n, f(x_n))$.

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

This gives $n + 1$ equations:

$$\begin{aligned} a_0 + b_0x_0 + c_0x_0^2 + d_0x_0^3 &= f(x_0) \\ a_i + b_ix_{i+1} + c_ix_{i+1}^2 + d_ix_{i+1}^3 &= f(x_{i+1}), \quad i = 0, \dots, n-1 \end{aligned}$$

2. Continuity in $S(x)$. I.e., for $i = 0, \dots, n-2$, $S(x_i)$ has the same value whether $[x_i, x_{i+1}]$ or $[x_{i+1}, x_{i+2}]$ is used. This gives $n-1$ more equations:

$$a_i + b_ix_{i+1} + c_ix_{i+1}^2 + d_ix_{i+1}^3 = a_{i+1} + b_{i+1}x_{i+1} + c_{i+1}x_{i+1}^2 + d_{i+1}x_{i+1}^3$$

3. Continuity in $S'(x)$. I.e., for $i = 0, \dots, n-2$, $S'(x_i)$ has the same value whether $[x_i, x_{i+1}]$ or $[x_{i+1}, x_{i+2}]$ is used. This gives $n-1$ more equations:

$$b_i + 2c_ix_{i+1} + 3d_ix_{i+1}^2 = b_{i+1} + 2c_{i+1}x_{i+1} + 3d_{i+1}x_{i+1}^2$$

4. Continuity in $S''(x)$. I.e., for $i = 0, \dots, n-2$, $S''(x_i)$ has the same value whether $[x_i, x_{i+1}]$ or $[x_{i+1}, x_{i+2}]$ is used. This gives $n-1$ more equations:

$$2c_i + 6d_ix_{i+1} = 2c_{i+1} + 6d_{i+1}x_{i+1}$$

So the total number of equations is $4n-2$, whereas the total number of unknowns is $4n$. Note that we could not have added another derivative (S'''), or level of smoothness, to the definition of the cubic spline since then the problem would be over constrained.

Conclusion: Since $4n-2 < 4n$, there are more than one cubic spline interpolating $(x_0, f(x_0)), \dots, (x_n, f(x_n))$.

To find a unique cubic spline, we need to impose two more constraints. There are three commonly used conditions. Each one gives a unique spline.

1. Complete Cubic Spline.

If $f'(x_0)$ and $f'(x_n)$ are known, we require that

$$S'(x_0) = f'(x_0), \quad S'(x_n) = f'(x_n)$$

With these two additional constraints, there is a unique cubic spline interpolation for $(x_0, f(x_0)), \dots, (x_n, f(x_n))$

Figure 16.1: Complete Cubic Spline.

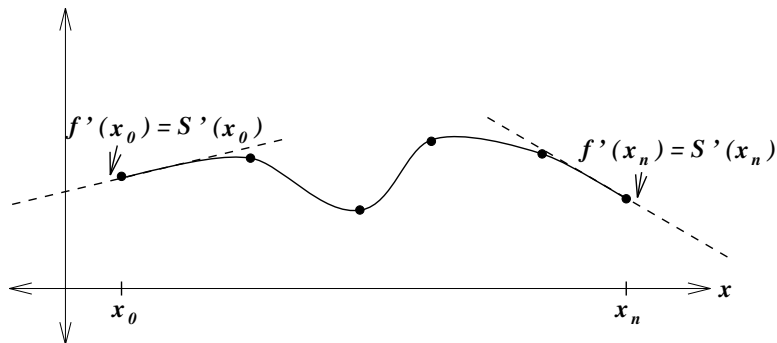


Figure 16.1: Complete Cubic Spline.

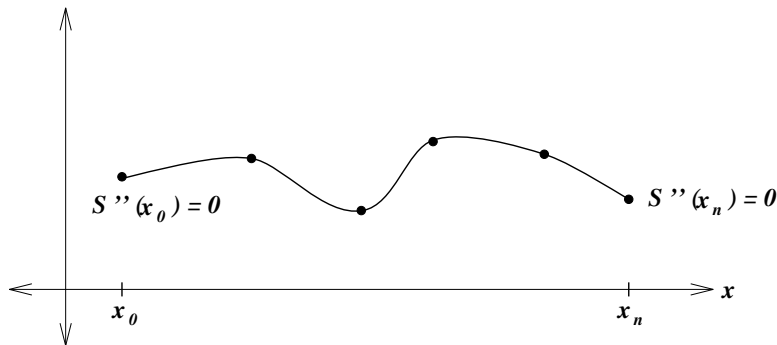


Figure 16.2: Natural Cubic Spline.

2. Natural Cubic Spline.

In this condition, we require that

$$S''(x_0) = 0, S''(x_n) = 0$$

Again, there is a unique natural cubic spline.

Figure 16.2: Natural Cubic Spline.

3. “Not-a-knot” Condition.

In this condition, we require that

$$S'''(x) \text{ is continuous at } x_1 \text{ and } x_{n-1}$$

I.e.,

$$\begin{aligned} S'''(x_1) &= \text{same value when either } [x_0, x_1] \text{ or } [x_1, x_2] \text{ is used} \\ \text{and } S'''(x_{n-1}) &= \text{same value when either } [x_{n-2}, x_{n-1}] \text{ or } [x_{n-1}, x_n] \text{ is used} \end{aligned}$$

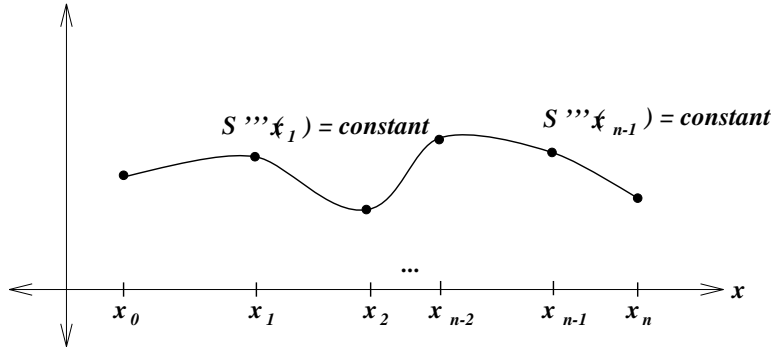


Figure 16.3: “Not-a-knot” Condition.

With these two constraints, there is a unique cubic spline.

Why “not-a-knot”? Since $S(x)$, $S'(x)$, $S''(x)$, and $S'''(x)$ are all continuous at x_1 , $S(x)$ is in fact a polynomial on $[x_0, x_2]$. Similarly, $S(x)$ is a polynomial on $[x_{n-2}, x_n]$. Thus $S(x)$ is a piecewise polynomial on subintervals

$$[x_0, x_2], [x_2, x_3], \dots, [x_{n-3}, x_{n-2}], [x_{n-2}, x_n]$$

So, x_1, x_{n-1} are not end-points of a subinterval so they are not knots. Of course, x_1, x_{n-1} are still interpolating points since $S(x_1) = f(x_1)$, and $S(x_{n-1}) = f(x_{n-1})$.

Figure 16.3: “Not-a-knot” Condition.

HOW TO COMPUTE $S(x)$ AT \tilde{x} .

1. Decide which condition, out of the three outlined above, to use to define which cubic spline to be used.
2. Compute $a_i, b_i, c_i, d_i (i = 0, \dots, n-1)$ by solving $4n$ equations, which we can do by using Gaussian Elimination, or more advanced techniques (e.g., Matlab’s `spline()` function).
3. Locate the interval $[x_i, x_{i+1}]$ such that $\tilde{x} \in [x_i, x_{i+1}]$.
4. Using Horner’s method to compute

$$S(\tilde{x}) = a_{i+1} + b_{i+1}\tilde{x} + c_{i+1}\tilde{x}^2 + d_{i+1}\tilde{x}^3.$$

Note: Usually we write $S(x)$ on $[x_i, x_{i+1}]$ as

$$S(x) = a_{i+1} + b_{i+1}(x - x_i) + c_{i+1}(x - x_i)^2 + d_{i+1}(x - x_i)^3.$$

In this way, a_{i+1} can be easily found since $S(x_i) = f(x_i)$. Thus we have $S(x_i) = a_{i+1} = f(x_i)$. Thus on the interval $[x_i, x_{i+1}]$

$$S(x) = f(x_i) + b_{i+1}(x - x_i) + c_{i+1}(x - x_i)^2 + d_{i+1}(x - x_i)^3.$$

Of course, b_i, c_i, d_i still need to be found by solving $3n$ equations. This can usually be done using library subroutines.

Example

Consider a Natural Cubic Spline fitted to the function $f(x) = e^x \sin x$ at the following points

x_i	$f(x_i)$
0	0
1	2.29
2	6.72
3	2.83

Therefore, $S''(x_0) = S''(x_3) = 0$.

Since $n = 3$, we have 12 equations:

$$\begin{aligned} a_0 + b_0 \cdot 0 + c_0 \cdot 0^2 + d_0 \cdot 0^3 &= 0 \\ a_0 + b_0 \cdot 1 + c_0 \cdot 1^2 + d_0 \cdot 1^3 &= 2.29 \\ a_1 + b_1 \cdot 2 + c_1 \cdot 2^2 + d_1 \cdot 2^3 &= 6.72 \\ a_2 + b_2 \cdot 3 + c_2 \cdot 3^2 + d_2 \cdot 3^3 &= 2.83 \end{aligned}$$

$$\begin{aligned} a_0 + b_0 \cdot 1 + c_0 \cdot 1^2 + d_0 \cdot 1^3 - a_1 - b_1 \cdot 1 - c_1 \cdot 1^2 - d_1 \cdot 1^3 &= 0 \\ a_1 + b_1 \cdot 2 + c_1 \cdot 2^2 + d_1 \cdot 2^3 - a_2 - b_2 \cdot 2 - c_2 \cdot 2^2 - d_2 \cdot 2^3 &= 0 \end{aligned}$$

$$\begin{aligned} b_0 + 2c_0 \cdot 1 + 3d_0 \cdot 1^2 - b_1 - 2c_1 \cdot 1 - 3d_1 \cdot 1^2 &= 0 \\ b_1 + 2c_1 \cdot 2 + 3d_1 \cdot 2^2 - b_2 - 2c_2 \cdot 2 - 3d_2 \cdot 2^2 &= 0 \end{aligned}$$

$$2c_0 + 6d_0 \cdot 1 - 2c_1 - 6d_1 \cdot 1 = 0$$

$$2c_1 + 6d_1 2 - 2c_2 - 6d_2 2 = 0$$

$$2c_0 + 6d_0 0 = 0$$

$$2c_2 + 6d_2 3 = 0$$

which in matrix form is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 9 & 27 & 0 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & -1 & -2 & -4 & -8 & 0 \\ 0 & 1 & 2 & 3 & 0 & -1 & -2 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 12 & 0 & -1 & -4 & -12 & 0 \\ 0 & 0 & 2 & 6 & 0 & 0 & -2 & -6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 12 & 0 & 0 & -2 & -12 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 18 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2.29 \\ 6.72 \\ 2.83 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

which has the solution

$$\begin{aligned} &[a_0, b_0, c_0, d_0, \\ &a_1, b_1, c_1, d_1, \\ &a_2, b_2, c_2, d_2] \approx \\ &[0, 1.16, 0, 1.13, \\ &4.61, -12.67, 13.84, -3.49, \\ &-42.17, 57.50, -21.25, 2.36]. \end{aligned}$$

See figure 16.4 for a plot of the Natural Spline.

If the two additional conditions for the Natural Cubic Spline are changed to that of the Complete Cubic spline, $S'(x_0) = f'(x_0)$, $S'(x_n) = f'(x_n)$, we get the Complete Spline in figure 16.5.

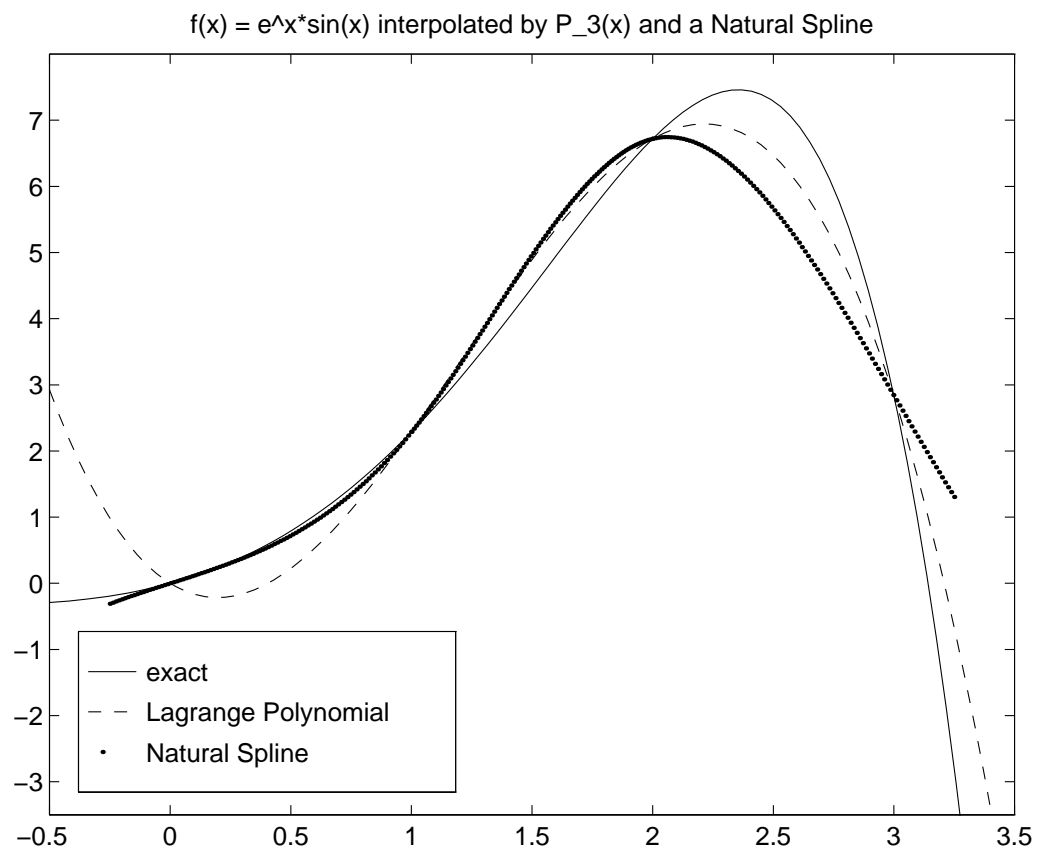


Figure 16.4: A comparison of a Natural Spline and a Lagrange Polynomial interpolating the same function, $e^x \sin x$.

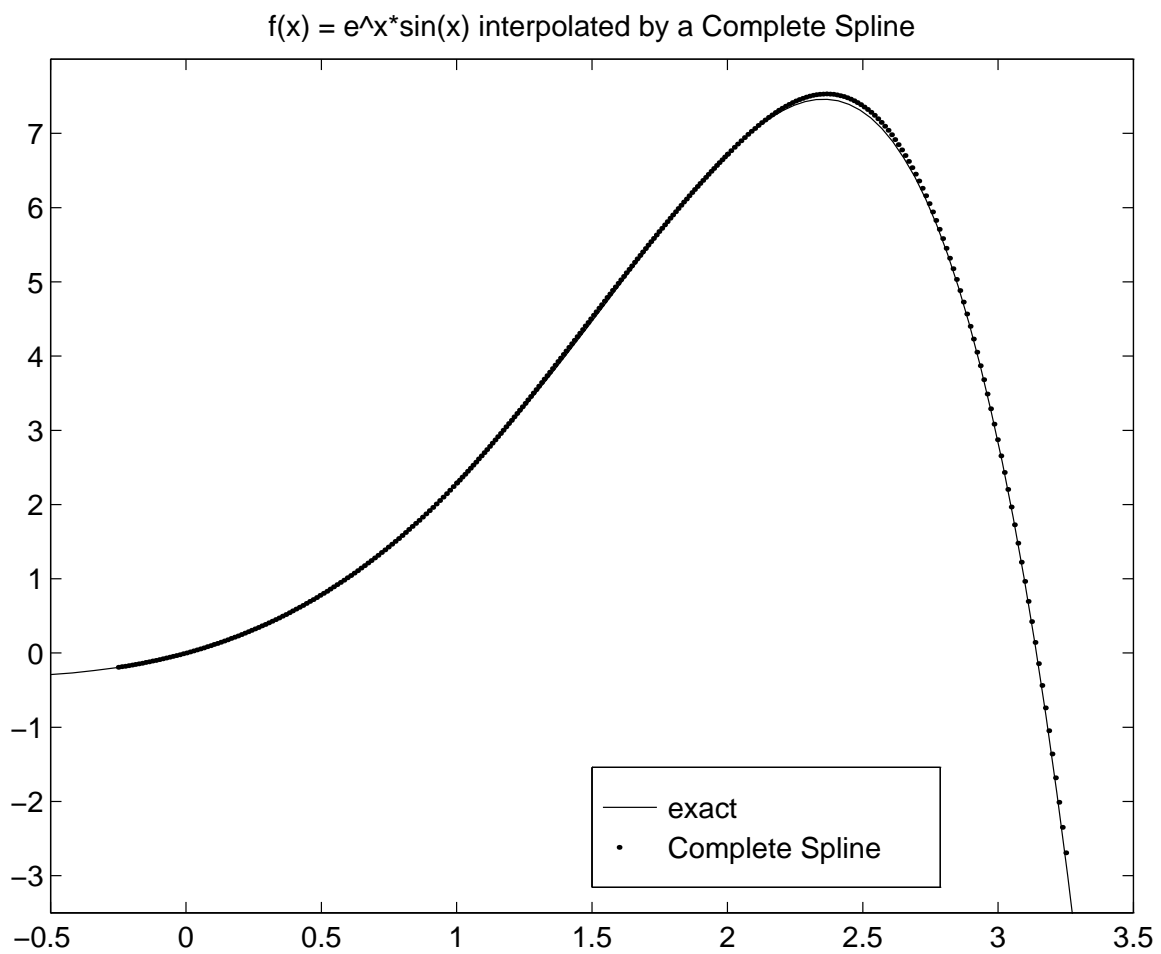


Figure 16.5: A Complete Spline interpolating $e^x \sin x$.

CISC 271 Class 19

Numerical Integration

Quadrature – comes from the process of “squaring”, of finding a square equal in area to a given area, e.g., finding the area of a circle. Now means numerical integration.

The problem is either

- Given $f(x)$ defined on $[a, b]$, find $I = \int_a^b f(x)dx$ **or**
- Given $f(x_i)$ defined on $\{x_i\}$, find $I = \int_{x_0}^{x_n} f(x)dx$

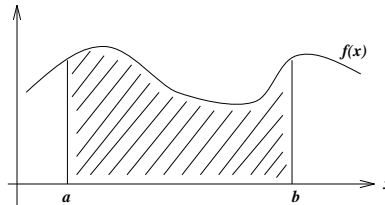


Figure 17.1: Picture of an example of an integration. The area of the shaded region is the result of the integration.

See Figure 17.1 for a generic example.

Easy example: $I = \int_a^b x dx = \frac{1}{2}(b^2 - a^2)$ or more generally: $\int_a^b P_n(x) dx$

Hard example: $I = \int_a^b e^{\cos(x)} dx = ?$

In many applications,

- | | | |
|--------------------|---|---|
| $f(x)$ | – | complicated |
| $\int_a^b f(x) dx$ | – | cannot be calculated analytically |
| | – | must be approximated by a numerical value |

The approach:

1. Locally interpolate $f(x)$ by a simple function $g(x)$, e.g., a polynomial interpolation $P_n(x)$, whose analytical integral is known.
2. Use the integral of the simpler function to approximate $\int_a^b f(x) dx$ locally, summing the local results as we move along.

Our goal is to get as accurate an answer as possible, with as few function evaluations as possible.

Quadrature can be done with *fixed* or variable (*adaptive*) spacing. We will look at two fixed rules: Trapezoid and Simpson's; and one adaptive rule: Trapezoid.

TRAPEZOID RULE

The simplest polynomial approximation to a function is a piecewise linear interpolation. See Figure 17.2.

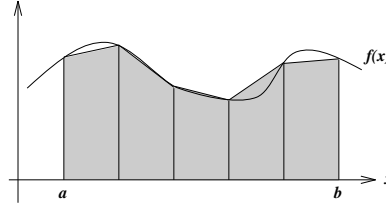


Figure 17.2: Picture of a piecewise linear approximation to the function, and the corresponding resulting integration.

Consider a linear interpolation of $f(x)$ between points x_i and x_{i+1} .

$$\text{Therefore, } \int_{x_i}^{x_{i+1}} f(x) dx \simeq (x_{i+1} - x_i) \frac{f(x_i) + f(x_{i+1})}{2}$$

So, if the stepsize is h , then the area of any trapezoid is

$$\frac{h}{2} \cdot (f_i + f_{i+1})$$

The integral is thus approximately, for $n + 1$ points,

$$\begin{aligned} I(f) \approx T_n(f) &= \sum_{i=0}^{n-1} \frac{h}{2} \cdot (f_i + f_{i+1}) \\ &= \frac{h}{2} \left(\sum_{i=0}^{n-1} f_i + \sum_{i=0}^{n-1} f_{i+1} \right) \\ &= \frac{h}{2} [(f_0 + f_1 + f_2 + \cdots + f_{n-2} + f_{n-1}) + (f_1 + f_2 + \cdots + f_{n-1} + f_n)] \\ &= \frac{h}{2} (f_0 + 2f_1 + 2f_2 + 2f_3 + \cdots + 2f_{n-1} + f_n) \\ &= \frac{h}{2} \left(f_0 + 2 \sum_{i=1}^{n-1} f_i + f_n \right) \end{aligned}$$

where $f_0 = f(a)$ and $f_n = f(b)$.

DISCRETIZATION ERROR

To perform the integration using the Trapezoid Rule, we are approximating $f(x)$ on $[x_0, x_1]$ by a first-order polynomial $P_1(x)$. Thinking of this as a Taylor expansion about x_0 , we know that

$$\begin{aligned} f(x) &= P_1(x) + R_2(x_0) \\ &= P_1(x) + \frac{(x - x_0)^2}{2} f''(x_0) + \frac{(x - x_0)^3}{6} f'''(x_0) + O(h^4) \\ &\approx P_1(x) + \frac{(x - x_0)^2}{2} f''(x_0) + \frac{(x - x_0)^3}{6} f'''(x_0) \end{aligned}$$

and in particular

$$f(x_1) \approx P_1(x_1) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{6} f'''(x_0)$$

The error $E(R)$ in the integral $I(f) - I(P_1)$ is

$$\begin{aligned} E(R) &\approx \int_{x_0}^{x_1} \frac{(x - x_0)^2}{2} f''(x_0) dx \\ &\approx \frac{f''(x_0)}{2} \int_{x_0}^{x_1} (x - x_0)^2 dx \\ &\approx \frac{f''(x_0)}{2} \left[\frac{(x - x_0)^3}{3} \right]_{x_0}^{x_1} \\ &\approx \frac{f''(x_0)}{6} (x_1 - x_0)^3 \end{aligned}$$

Thus, the total error is the Trapezoid Rule minus the integral of $P_1(x)$ minus $E(R)$:

$$\begin{aligned} E &\approx \frac{h}{2} (f(x_0) + f(x_1)) - hf(x_0) - \frac{h^2}{2} f'(x_0) - \frac{h^3}{6} f''(x_0) - E(R) \\ &\approx \frac{h^3}{12} f''(x_0) \\ &\approx \frac{h^3}{12} M_2 \end{aligned}$$

for a bound M_2 on $f''(x)$ over $[x_0, x_1]$.

The total possible quadrature error is the sum of all the errors for each of the panels, $[x_i, x_{i+1}]$,

$$\begin{aligned} T_n(f) - I(f) &\leq \sum_{i=1}^n \left(\frac{M_2 h^3}{12} \right) \\ &= \frac{M_2 h^3 n}{12} \\ &= \frac{M_2 (b-a) h^2}{12} \\ &= O(h^2) \end{aligned}$$

Therefore,

$$|T_n(f) - I(f)| \leq h^2 \frac{M_2(b-a)}{12}$$

so this is a second-order method.

Example

Evaluate $I = \int_0^\pi e^x \cos(x) dx$ by composite trapezoidal rule using 4 subintervals (panels).

Solution: $[a, b] = [0, \pi]$, $f(x) = e^x \cos(x)$

$$n = 4, h = \frac{b-a}{n} = \frac{\pi}{4} \text{ such that } x_0 = 0, x_1 = \frac{\pi}{4}, x_2 = \frac{\pi}{2}, x_3 = \frac{3\pi}{4}, x_4 = \pi.$$

$$\begin{aligned} T_4(f) &= h \left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + f(x_3) + \frac{f(x_4)}{2} \right] \\ &= \frac{\pi}{4} \left[\frac{1}{2} + 1.5509 + 0 + (-7.4605) + \frac{(-23.141)}{2} \right] \\ &= -13.336 \end{aligned}$$

$$\begin{array}{ll} n = 8 & T_8(f) = -12.382 \\ n = 64 & T_{64}(f) = -12.075 \\ n = 512 & T_{512}(f) = -12.070 \\ \text{True Sol'n} & \simeq -12.0703 \end{array}$$

Simpson's Rule and Newton-Cotes Integration

SIMPSON'S RULE

Now, let's locally approximate $f(x)$ by a quadratic polynomial $P_2(x)$. Hereafter, we will *always* assume that n is even (for deep reasons).

See Figure 18.1. The knots for P_2 occur at the even points. The regions between knots are called *panels*. With $n + 1$ points, the number of panels is $n/2$.

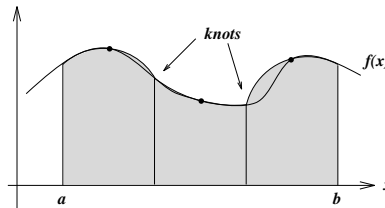


Figure 18.1: Picture of a function approximated by piecewise quadratic polynomial.

We can develop Simpson's Rule by using Lagrangian interpolation to find $P_2(x)$ over $[x_i, x_{i+2}]$ and then integrate it to find $I(P_2)$. See Figure 18.2.

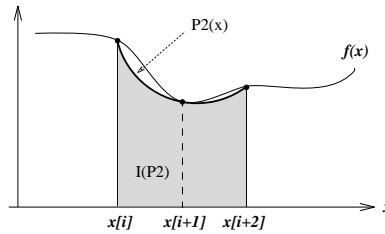


Figure 18.2: Picture of a function locally approximated by a quadratic polynomial, between the points x_i and x_{i+2} .

The interpolation function is

$$P_2(x) = f(x_i)l_0^2(x) + f(x_{i+1})l_1^2(x) + f(x_{i+2})l_2^2(x)$$

where

$$l_0^2(x) = \frac{(x - x_{i+1})(x - x_{i+2})}{(x_i - x_{i+1})(x_i - x_{i+2})}, \quad l_1^2(x) = \frac{(x - x_i)(x - x_{i+2})}{(x_{i+1} - x_i)(x_{i+1} - x_{i+2})},$$

$$l_2^2(x) = \frac{(x - x_i)(x - x_{i+1})}{(x_{i+2} - x_i)(x_{i+2} - x_{i+1})}$$

Then

$$\begin{aligned} I(P_2) &= \int_{x_i}^{x_{i+2}} P_2(x) dx \\ &= f(x_i) \frac{x_{i+2} - x_i}{6} + f(x_{i+1}) \frac{4(x_{i+2} - x_i)}{6} + f(x_{i+2}) \frac{x_{i+2} - x_i}{6} \end{aligned}$$

Therefore,

$$\int_{x_i}^{x_{i+2}} f(x) dx \simeq I(P_2) = \frac{h}{3} (f_i + 4f_{i+1} + f_{i+2}),$$

where $h = x_{i+1} - x_i$.

To get the sum over the entire interval $[a, b]$, we sum over all the panels, noting that the end points of the panels are have even numbered indicies, with $h = (b - a)/n$,

$$\begin{aligned} S_n(f) &= \frac{h}{3} (f_0 + 4f_1 + f_2) + \frac{h}{3} (f_2 + 4f_3 + f_4) + \cdots + \frac{h}{3} (f_{n-2} + 4f_{n-1} + f_n) \\ &= \sum_{\substack{i=0 \\ i=\text{even}}}^{n-2} \frac{h}{3} (f_i + 4f_{i+1} + f_{i+2}) \\ &= \sum_{i=0}^{n/2-1} \frac{h}{3} (f_{2i} + 4f_{2i+1} + f_{2i+2}) \\ &= \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + 4f_5 + \cdots + 4f_{n-3} + 2f_{n-2} + 4f_{n-1} + f_n) \\ &= \frac{h}{3} \left(f_0 + 4 \sum_{i=0}^{n/2-1} f_{2i+1} + 2 \sum_{i=0}^{n/2-2} f_{2i+2} + f_n \right) \end{aligned}$$

By using more advanced techniques, we can show that for even n , and $f(x)$ four times differentiable, the local error per panel (containing three points) is

$$|I(P_2) - I(f)| \leq h^5 \frac{M_4}{90},$$

with M_4 being the bound on $f^{(4)}(x)$. For the composite Simpson's Rule over the entire domain the upper bound on the error is

$$|S_n(f) - I(f)| \leq h^4 \frac{M_4(b-a)}{180} = \frac{1}{n^4} \frac{M_4(b-a)^5}{180},$$

Therefore, Simpson's Rule is *fourth-order*.

Example

Evaluate $I = \int_0^\pi e^x \cos(x) dx$ by composite Simpson's rule using 2 subintervals (panels).

Solution: Again, $[a, b] = [0, \pi]$, $f(x) = e^x \cos(x)$

$$n = 4, h = \frac{b - a}{n} = \frac{\pi}{4} \text{ such that } x_0 = 0, x_1 = \frac{\pi}{4}, x_2 = \frac{\pi}{2}, x_3 = \frac{3\pi}{4}, x_4 = \pi.$$

$$\begin{aligned} C.S.R. &= \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)] \\ &= \frac{\pi}{12} [1 + 4(1.5509) + 2(0) + 4(-7.4605) + (-23.141)] \\ &= -11.985 \end{aligned}$$

The exact solution is -12.0703 . Thus with $n = 4$, the Composite Simpson's Rule has an error of 0.08528 , as compared to the Composite Trapezoid Rule for the same n , which has an error of -1.2657 . With $n = 8$, the result of the CSR has the same magnitude of error as the result using $n = 512$ with the CTR. Since our goal is to have an accurate a result with a few a number of function evaluations as possible, the CSR is a marked improvement for this function.

Assessment

- If the function is smooth, Simpson's Rule is better.
- If the function has abrupt changes, then Trapezoid is better.
- Higher-order methods exist, but are tedious.
- A simple, interesting extension is *spline quadrature*.

NEWTON-COTES INTEGRATION

Consider the following:

1. $a \leq x_0 < x_1 < \dots < x_m \leq b$ where x_0 , and x_m may not be the endpoints.
2. interpolate $f(x)$ at x_0, \dots, x_m by a Lagrange formula.

$$P_m(x) = f(x_0)l_0^m(x) + \dots + f(x_m)l_m^m(x)$$

so that

$$\begin{aligned} \int_a^b P_m(x)dx &= f(x_0) \int_a^b l_0^m(x)dx + \dots + f(x_m) \int_a^b l_m^m(x)dx \\ &= w_0 f(x_0) + \dots + w_m f(x_m) \end{aligned}$$

where $w_i = \int_a^b l_i^m(x)dx$.

3. $\int_a^b f(x)dx \simeq w_0 f(x_0) + \dots + w_m f(x_m)$. This is the *weighted average* of $f(x_0), f(x_1), \dots, f(x_m)$.

Some commonly used Newton-Cotes formulae ($h = \frac{b-a}{m}$). Note that the error is $I(f) - \text{NC}_m(f)$.

m	rule $\simeq \int_a^b P_m(x)dx$	error	Rule
1	$hf(\frac{a+b}{2})$	$\frac{h^3}{24}f^{(2)}(\xi)$	midpoint
1	$\frac{h}{2}[f(a) + f(b)]$	$-\frac{h^3}{12}f^{(2)}(\xi)$	trapezoid
2	$\frac{h}{3}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$	$-\frac{h^5}{90}f^{(4)}(\xi)$	Simpson's
3	$\frac{3h}{8}[f(a) + 3f(a+h) + 3f(b-h) + f(b)]$	$-\frac{3h^5}{80}f^{(4)}(\xi)$	three-eighths rule
4	$\frac{2h}{45}[7f(a) + 32f(a+h) + 12f(\frac{a+b}{2}) + 32f(b-h) + 7f(b)]$	$-\frac{h^7}{945}f^{(6)}(\xi)$	Boole's rule

If h is small, then the error is also small. In practice, $b - a$ is fixed, e.g.,

$$\int_0^1 \quad \text{or} \quad \int_0^{2\pi}$$

So, we divide $[a, b]$ into small intervals (panels).

COMPOSITE RULES

Composite rules are constructed in the following manner:

1. Divide $[a, b]$ into p subintervals (panels) $[a, t_1], [t_1, t_2], \dots, [t_{p-1}, b]$.

2. Apply the basic rule (i.e., one of the Newton-Cotes formulae above) to each of $[t_i, t_{i+1}]$.
3. For convenience, we assume each $[t_i, t_{i+1}]$ has equal length and one basic rule is applied to each interval.

Note that if there are p panels, each using a rule using $m + 1$ points on each panel, then $n = mp$. Let $h = \frac{b-a}{n}$, and $x_i = a + ih$. For example, look at figure 18.3. Note that $t_i = x_{im}$.

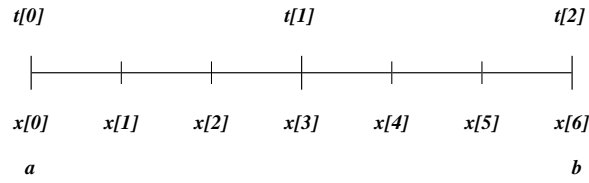


Figure 18.3: Picture showing $p = 2$, $m = 3$, and $n = mp = 6$.

Then

$$\begin{aligned}
 I &= \int_a^b f(x)dx \\
 &= \int_{t_0}^{t_1} f(x)dx + \int_{t_1}^{t_2} f(x)dx + \cdots + \int_{t_{p-1}}^{t_p} f(x)dx
 \end{aligned}$$

Composite Trapezoidal Rule

$$\int_{t_i}^{t_{i+1}} f(x)dx = \int_{x_i}^{x_{i+1}} f(x)dx \simeq \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$\begin{aligned}
 I &= \int_a^b f(x)dx \\
 &= \sum_{i=0}^{p-1} \int_{t_i}^{t_{i+1}} f(x)dx \\
 &\simeq \frac{h}{2} \sum_{i=0}^{n-1} [f(x_i) + f(x_{i+1})] \\
 &= h \left[\frac{f(x_0)}{2} + f(x_1) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2} \right]
 \end{aligned}$$

Composite Simpson's Rule

$$\int_{t_i}^{t_{i+1}} f(x)dx = \int_{x_{2i}}^{x_{2i+2}} f(x)dx \simeq \frac{h}{3}[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})]$$

$$\begin{aligned} I &= \int_a^b f(x)dx \\ &= \sum_{i=0}^{p-1} \int_{t_i}^{t_{i+1}} f(x)dx \\ &= \sum_{i=0}^{\frac{n}{2}-1} \int_{x_{2i}}^{x_{2i+2}} f(x)dx \\ &\simeq \sum_{i=0}^{\frac{n}{2}-1} \frac{h}{3}[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})] \\ &= \frac{h}{3} \left(f_0 + 4 \sum_{i=0}^{n/2-1} f_{2i+1} + 2 \sum_{i=0}^{n/2-2} f_{2i+2} + f_n \right) \end{aligned}$$

Example (again)

Evaluate $I = \int_0^\pi e^x \cos(x)dx$ by composite trapezoidal rule using 4 subintervals (panels).

Solution: $[a, b] = [0, \pi]$, $f(x) = e^x \cos(x)$

$$n = 4, h = \frac{b-a}{n} = \frac{\pi}{4} \text{ such that } x_0 = 0, x_1 = \frac{\pi}{4}, x_2 = \frac{\pi}{2}, x_3 = \frac{3\pi}{4}, x_4 = \pi.$$

$$\begin{aligned} C.T.R. &= h \left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + f(x_3) + \frac{f(x_4)}{2} \right] \\ &= \frac{\pi}{4} \left[\frac{1}{2} + 1.5509 + 0 + (-7.4605) + \frac{(-23.141)}{2} \right] \\ &= -13.336 \end{aligned}$$

$$n = 8 \quad CTR = -12.382$$

$$n = 64 \quad CTR = -12.075$$

$$n = 512 \quad CTR = -12.070$$

$$\text{True Sol'n} \simeq -12.0703$$

So in a composite method, as n gets larger \Rightarrow the error gets smaller. But how do we know which n to take for a given accuracy?

CISC 271 Class 21

Adaptive Integration

LOCAL ERROR ESTIMATES

When we do the quadrature on $[t_i, t_{i+1}]$ we have to pick a stepsize; let's break the interval into 4.

$$\begin{array}{cccccc} x_0 & x_1 & x_2 & x_3 & x_4 \\ & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & & & \\ t_i & & & & & t_{i+1} \end{array}$$

If we do the quadrature with the Composite Simpson's Rule using x_0, x_2 , and x_4 , we get

$$\tilde{S}^i(f) = \frac{2 \cdot h_i}{3} [f_0 + 4f_2 + f_4].$$

How accurate is this?

Try the quadrature on the left interval $[x_0, x_2]$ and then on $[x_2, x_4]$.

$$S_L^i(f) = \frac{h_i}{3} [f_0 + 4f_1 + f_2] \quad S_R^i(f) = \frac{h_i}{3} [f_2 + 4f_3 + f_4]$$

to get a better estimate: $S^i(f) = S_L^i(f) + S_R^i(f)$.

The local errors are (assuming $S^i(f) > I^i(f)$ and $\tilde{S}^i(f) > I^i(f)$)

$$\begin{aligned} I^i(f) - \tilde{S}^i(f) &= -\frac{(2h_i)^5}{90} f^{(4)}(\alpha_i) \\ \text{and } I^i(f) - S^i(f) &= -\frac{h_i^5}{90} [f^{(4)}(\beta_i) + f^{(4)}(\gamma_i)] \end{aligned}$$

Assume that $f^{(4)}(x)$ is nearly constant on $[x_0, x_4]$. We can then set

$$f^{(4)}(\alpha_i) = f^{(4)}(\beta_i) = f^{(4)}(\gamma_i) = M_i.$$

Then,

$$I^i(f) - S^i(f) \approx -\frac{2}{90} h_i^5 M_i \text{ and } I^i(f) - \tilde{S}^i(f) \approx -\frac{2^5}{90} h_i^5 M_i$$

and since

$$S^i(f) - \tilde{S}^i(f) \approx -\frac{2}{90} h_i^5 (2^4 - 1) M_i$$

$$\Rightarrow -\frac{2}{90}h_i^5 M_i \approx \frac{S^i(f) - \tilde{S}^i(f)}{15}$$

and so

$$ERR^i \approx I^i(f) - S^i(f) \approx \frac{S^i(f) - \tilde{S}^i(f)}{15}$$

This leads us to an algorithm, given an error tolerance of epsilon:

```

sum all the ERR[i] over all the panels for a given interval
if (total of ERR[i] < epsilon) then
    use result = sum from i=0 to p-1 of S[i](f);
else
    break each [t[i], t[i+1]] into 2 subintervals (panels) and repeat;
end if

```

This may not terminate due to round-off error or some other difficulty (e.g., $f^{(4)}(x)$ is not really constant). So, it is necessary to subdivide a limited number of times (e.g., 10).

Also, this approach may be inefficient, in that we may be doing some work that is unnecessary.

Example

See Figure 19.1.

Using C.T.R. First calculate I_1, I_2 .

$I_2 - I_1$ is large, so subdivide panels
 $I_4 - I_2$ is large, so subdivide panels
 $I_8 - I_4$ is large, so subdivide panels, etc.

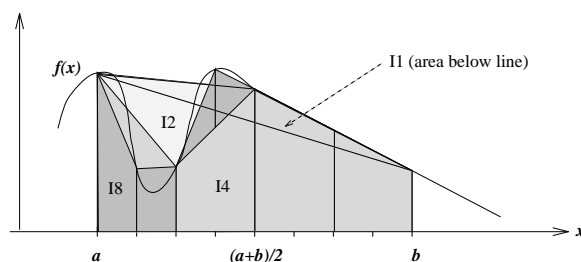


Figure 19.1: Picture of a global adaptive approach to increasing accuracy.

This is not a good way to do this calculation, for the following reasons:

- In computing each I_n , we always work with the whole interval $[a, b]$.
- We did not realize after the first step that the integral in $[\frac{a+b}{2}, b]$ had been obtained exactly.
- The error in I_2 is from $[a, \frac{a+b}{2}]$, *not* from $[\frac{a+b}{2}, b]$.
- Therefore, in the subsequent steps, we should only divide $[a, \frac{a+b}{2}]$ into small intervals. No need to further divide $[\frac{a+b}{2}, b]$.

This leads us to a better method.

ADAPTIVE METHOD

In this method we

- Use more subintervals in places where $f(x)$ is “badly” behaved. Hence, a large number of subintervals are used in the places where $f(x)$ changes rapidly.
- Use fewer subintervals in places where $f(x)$ is “well” behaved. Hence a small number of subintervals are used in the places where $f(x)$ is smooth.

The method is outlined as follows, given a tolerance ϵ :

1. Compute I_1 and I_2
 If $|I_2 - I_1| < \epsilon$
 $I_2 = \text{result and stop}$
 Else
 Note that $I_2 = I_2^1 + I_2^2$
 where $I_2^1 = TR \text{ in } [a, \frac{a+b}{2}]$
 $I_2^2 = TR \text{ in } [\frac{a+b}{2}, b]$
 See Figures 19.2 and 19.3.
2. Now, we compute $I_4 = I_4^1 + I_4^2 + I_4^3 + I_4^4$
 Let $I_4^{1,2} = I_4^1 + I_4^2$ and $I_4^{3,4} = I_4^3 + I_4^4$
 We won't compare $|I_4 - I_2| \stackrel{?}{<} \epsilon$ but rather

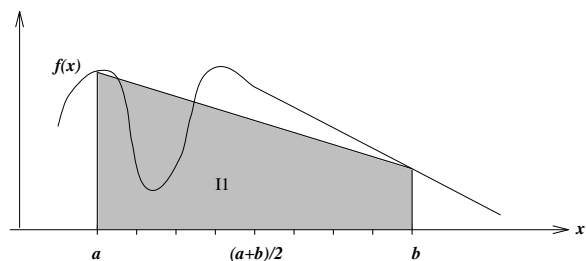


Figure 19.2: Calculation of I_1 .

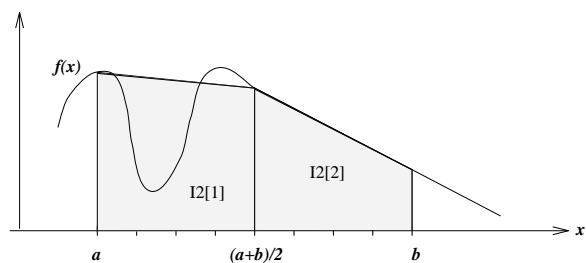


Figure 19.3: Calculation of I_2 .

$$|I_4^{1,2} - I_2^1| \stackrel{?}{<} \frac{\epsilon}{2} \text{ and } |I_4^{3,4} - I_2^2| \stackrel{?}{<} \frac{\epsilon}{2} \text{ If none satisfied}$$

Go to next step.

However, if $|I_4^{3,4} - I_2^2| \leq \frac{\epsilon}{2}$ then

Take $I_4^{3,4}$ as the approximate for $\int f(x)dx$ in the corresponding interval $[\frac{a+b}{2}, b]$, and never come back again.

See Figure 19.4

3. Assume that $|I_4^{3,4} - I_2^2| \leq \frac{\epsilon}{2}$ so that we don't have to consider I_2^2 . Compute

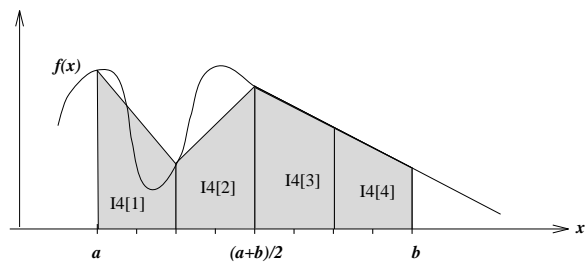


Figure 19.4: Calculation of I_4 .

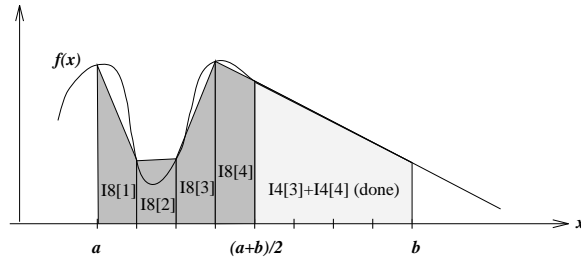


Figure 19.5: Calculation of I_8 on $[a, \frac{a+b}{2}]$.

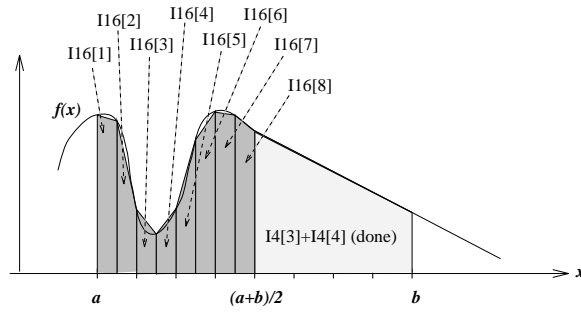


Figure 19.6: Calculation of I_{16} on $[a, \frac{a+b}{2}]$.

I_8 , but only on $[a, \frac{a+b}{2}]$.

$$I_8 = I_8^1 + I_8^2 + I_8^3 + I_8^4$$

Again, let $I_8^{1,2} = I_8^1 + I_8^2$, and $I_8^{3,4} = I_8^3 + I_8^4$
See Figure 19.5.

Compare

$$|I_8^{1,2} - I_4^1| \stackrel{?}{<} \frac{\epsilon}{4} \text{ and } |I_8^{3,4} - I_4^2| \stackrel{?}{<} \frac{\epsilon}{4}$$

Suppose none are satisfied, so we go to the next step.

4. Compute, again on $[a, \frac{a+b}{2}]$,

$$I_{16} = I_{16}^1 + I_{16}^2 + \cdots + I_{16}^7 + I_{16}^8$$

See Figure 19.6.

Compare

$$|I_{16}^{1,2} - I_8^1| \stackrel{?}{<} \frac{\epsilon}{8}, |I_{16}^{3,4} - I_8^2| \stackrel{?}{<} \frac{\epsilon}{8}, |I_{16}^{5,6} - I_8^3| \stackrel{?}{<} \frac{\epsilon}{8}, |I_{16}^{7,8} - I_8^4| \stackrel{?}{<} \frac{\epsilon}{8}.$$

Suppose none are satisfied but $|I_{16}^{5,6} - I_8^3| < \frac{\epsilon}{8}$. Then

$$I_{16}^{5,6} = \text{result in } [a + 2\frac{b-a}{8}, a + 3\frac{b-a}{8}].$$

And we repeat for the other intervals.

CISC 271 Class 22

Gaussian Quadrature

The Newton-Cotes rules and Composite rules:

$$\int_a^b f(x)dx \simeq \sum_{i=0}^n w_i f(x_i)$$

- n is fixed
- x_i are fixed
E.g. In trapezoid: $n = 1$, $x_0 = a$ and $x_1 = b$.
- w_i can be computed when the $\{x_i\}$ are given; i.e., they are determined by x_i
E.g. In trapezoid: $w_0 = \frac{b-a}{2} = w_1$.

Disadvantages: x_i are chosen artificially – how do we know they give us the best result?

Note that we are considering just one panel here.

Another approach

$$I \simeq \sum_{i=0}^n w_i f(x_i)$$

- n is fixed
- w_i, x_i are to be determined, so that

$$\sum_{i=0}^n w_i f(x_i) \text{ gives the “best” result.}$$

“best:” it gives *exact* result for polynomials of highest degree possible.

I.e., we want $I = \sum_{i=0}^n w_i f(x_i)$ if $f(x)$ is a polynomial of some degree, and we want the degree to be as high as possible.

Example

$$n = 1, [a, b] = [-1, 1]$$

$$\int_{-1}^1 f(x)dx \simeq w_0 f(x_0) + w_1 f(x_1)$$

x_0, x_1, w_0, w_1 are to be determined such that

$$\int_{-1}^1 P_m(x)dx = w_0 P_m(x_0) + w_1 P_m(x_1) \quad (\text{Equation A})$$

for m as large as possible.

1. Exact for polynomial of degree 0, i.e., Equation A holds for

$$P_0(x) = 1$$

$$\int_{-1}^1 1dx = w_0 + w_1 \Rightarrow \boxed{w_0 + w_1 = 2}$$

2. Exact for polynomial of degree 1, i.e., Equation A holds for

$$P_0(x) = x$$

$$\int_{-1}^1 xdx = w_0 x_0 + w_1 x_1 \Rightarrow \boxed{w_0 x_0 + w_1 x_1 = 0}$$

3. Exact for polynomial of degree 2, i.e., Equation A holds for

$$P_0(x) = x^2$$

$$\int_{-1}^1 x^2 dx = w_0 x_0^2 + w_1 x_1^2 \Rightarrow \boxed{w_0 x_0^2 + w_1 x_1^2 = \frac{2}{3}}$$

4. Exact for polynomial of degree 3, i.e., Equation A holds for

$$P_0(x) = x^3$$

$$\int_{-1}^1 x^3 dx = w_0 x_0^3 + w_1 x_1^3 \Rightarrow \boxed{w_0 x_0^3 + w_1 x_1^3 = 0}$$

Can we expect the method to be exact for still higher degree polynomials? No.

We have 4 unknowns, x_0, x_1, w_0, w_1 , and if the method is exact for polynomials of degree 3, we already have 4 equations. This is enough to determine the 4 unknowns.

By solving the four equations in boxes above, we find

$$x_0 = -\frac{\sqrt{3}}{3}, x_1 = \frac{\sqrt{3}}{3}, w_0 = 1, w_1 = 1$$

Thus

$$\int_{-1}^1 f(x)dx \simeq f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$$

This is Gaussian Quadrature on $[-1, 1]$ with two nodes.

From above, we know that $f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$ is exact if

$$f = 1, x, x^2, x^3.$$

Is it exact for all polynomials of degree ≤ 3 ?

Yes:

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

$$\begin{aligned} \int_{-1}^1 f(x)dx &= a_0 \int_{-1}^1 dx + a_1 \int_{-1}^1 xdx + a_2 \int_{-1}^1 x^2dx + a_3 \int_{-1}^1 x^3dx \\ &= a_0[1 + 1] + a_1\left[-\frac{\sqrt{3}}{3} + \frac{\sqrt{3}}{3}\right] + a_2\left[\left(-\frac{\sqrt{3}}{3}\right)^2 + \left(\frac{\sqrt{3}}{3}\right)^2\right] + a_3\left[\left(-\frac{\sqrt{3}}{3}\right)^3 + \left(\frac{\sqrt{3}}{3}\right)^3\right] \\ &= \left[a_0 + a_1\left(-\frac{\sqrt{3}}{3}\right) + a_2\left(-\frac{\sqrt{3}}{3}\right)^2 + a_3\left(-\frac{\sqrt{3}}{3}\right)^3\right] + \left[a_0 + a_1\left(\frac{\sqrt{3}}{3}\right) + a_2\left(\frac{\sqrt{3}}{3}\right)^2 + a_3\left(\frac{\sqrt{3}}{3}\right)^3\right] \\ &= f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right) \end{aligned}$$

So, it is exact for all polynomials of degree ≤ 3 .

Example

Evaluate $\int_{-1}^1 3 + 4x + 8x^2 + 2x^3dx$

Via Gaussian Quadrature:

$$\begin{aligned} \int_{-1}^1 3 + 4x + 8x^2 + 2x^3dx &= \left[3 + 4\left(\frac{\sqrt{3}}{3}\right) + 8\left(\frac{\sqrt{3}}{3}\right)^2 + 2\left(\frac{\sqrt{3}}{3}\right)^3\right] \\ &\quad + \left[3 + 4\left(\frac{-\sqrt{3}}{3}\right) + 8\left(\frac{-\sqrt{3}}{3}\right)^2 + 2\left(\frac{-\sqrt{3}}{3}\right)^3\right] \\ &= 2\left[3 + 8\left(\frac{\sqrt{3}}{3}\right)^2\right] \\ &= 2\left[3 + \frac{8}{3}\right] \\ &= \frac{34}{3} \end{aligned}$$

Compare that with straight integration:

$$\begin{aligned}\int_{-1}^1 3 + 4x + 8x^2 + 2x^3 dx &= \left(3x + \frac{4x^2}{2} + \frac{8x^3}{3} + \frac{2x^4}{4} \right) \Big|_{-1}^1 \\ &= 2 \left[3 + \frac{8}{3} \right] \\ &= \frac{34}{3}\end{aligned}$$

Comparison

	Gaussian quadrature	Trapezoidal
Function Evaluations	2	2
Exact for polynomials of degree \leq	3	1

Gaussian Quadrature in General

$$\int_a^b f(x)dx \simeq \sum_{i=0}^n w_i f(x_i)$$

x_i, w_i are chosen so that the method is exact for

$$1, x, x^2, \dots, x^m$$

where m is as large as possible. What is the largest possible m , for a fixed n ?

The number of unknowns are: $2n + 2$ and also $m + 1$ functions $\Rightarrow m + 1$ equations.

$$\text{Unknown} = \text{Eqns} \Rightarrow m + 1 = 2n + 2$$

$$\text{I.e., } m = 2n + 1$$

Conclusion: Gaussian quadrature with $n + 1$ nodes (function evaluations) is exact for a polynomial of degree $\leq 2n + 1$. In comparison, a Newton-Cotes rule of degree n with $n + 1$ nodes is exact for polynomials of degree $\leq n$.

When we have to determine w_i, x_i , we have to solve a non-linear system.

CISC 271 Class 23

Ordinary Differential Equations - Euler Method

ODEs - DEFINITION

First, what are ordinary differential equations (ODEs)? They are equations (obviously!) that

- involve one or more derivatives of $x(t)$, where
- $x(t)$ is unknown and is the desired target

For shorthand, we will use $x = x(t)$, $x' = \frac{dx(t)}{dt}$, $x'' = \frac{d^2x(t)}{dt^2}$, ...

For example,

$$(x'''(t))^{\frac{3}{7}} + 37te^{x^2(t)} \sin \sqrt[4]{x'(t)} - \log \frac{1}{t} = 42$$

Which $x(t)$'s fulfill this behaviour?

Terminology

Ordinary (vs. partial) = *one* independent variable t

Order = highest (composition of) derivatives(s) involved

Linear = derivatives, including zeroth, appear in linear form

Homogeneous = all terms involve some derivative (including zeroth)

Analytical Solutions

Some ODEs are analytically solvable.

$$\begin{aligned}x' - x &= e^t \Rightarrow x(t) = te^t + ce^t \\x'' + 9x &= 0 \Rightarrow x(t) = c_1 \sin 3t + c_2 \cos 3t \\x' + \frac{1}{2x} &= 0 \Rightarrow x(t) = \sqrt{c - t}\end{aligned}$$

In the solutions to the above, c , c_1 , and c_2 are arbitrary constants.

Before we can pin down the exact values of these constants, we need more conditions/information. There are two possible ways to do this:

- Initial Value Problems (IVP)
- Boundary Value Problems (BVP)

FIRST ORDER IVP

The types of problems that we will try to solve are first-order Initial Value Problems (IVPs). Their general form is:

$$x' = f(t, x), \quad x(a) \text{ is given}$$

Note that this equation is non-linear and non-homogeneous in general.

Examples

$$\begin{aligned} x' &= x + 1, \quad x(0) = 0 \Rightarrow x(t) = e^t - 1 \\ x' &= 6t - 1, \quad x(1) = 6 \Rightarrow x(t) = 3t^2 - t + 4 \\ x' &= \frac{t}{x+1}, \quad x(0) = 0 \Rightarrow x(t) = \sqrt{t^2 + 1} - 1 \end{aligned}$$

RHS independent of x

Suppose that the righthand side of a first-order Initial Value Problem is only a function of t , but not x . I.e.,

$$x' = f(t), \quad \text{but } f \neq f(x)$$

For example,

$$x' = 3t^2 - 4t^{-1} + (1 + t^2)^{-1}, \quad x(5) = 17$$

In this case we can perform an indefinite integration of both sides of the differential equation:

$$x(t) = \int \frac{d(x(t))}{dt} dt = \int f(t) dt$$

For the above equation, we then obtain

$$x(t) = t^3 - 4 \ln t + \arctan t + C$$

where $C = 17 - 5^3 + 4 \ln 5 - \arctan 5$.

Need for Numerical Techniques

But usually, an analytical solution is not known. Or even if one is known, perhaps it is very complicated and expensive to compute. Therefore, we need numerical techniques to solve these types of problems. The numerical techniques that we will discuss will

- Generate a table of values for $x(t)$
- Usually equispaced in t with stepsize h

One note of caution: with small h , and seeking a solution far from the initial value, roundoff error can accumulate and kill the solution.

EULER METHOD

In this section, we will develop and demonstrate the Euler Method for first-order IVP's. The problem we want to solve can be stated as follows:

given $x' = f(t, x)$, $x(a)$, we want to find $x(b)$

To find $x(b)$, we will do the following:

- We will use the first two terms of the Taylor series (i.e., $n = 1$) to get from $x(a)$ to $x(a + h)$:

$$\begin{aligned} x(a + h) &= x(a) + hx'(a) + O(h^2) \\ &= x(a) + hf(t, x(a)) + O(h^2) \end{aligned}$$

where $O(h^2)$ is the order of the truncation error. Note that $x'(a)$ was replaced with $f(t, x(a))$.

- Repeat to get from $x(a + h)$ to $x(a + 2h)$, etc.

In the above, we repeat the steps of size h until we arrive at $x(b)$. A total of $n = \frac{b-a}{h}$ steps are needed.

Example

Consider

$$x' = -2t - x(t), \quad x(0) = -1, \quad x(0.5) = ?$$

The analytical solution is

$$x(t) = -3e^{-t} - 2t + 2.$$

By applying Euler's method, with $h = 0.1$, we find

t	$x(t)$	exact	error
0.00000	-1.00000	-1.00000	0.00000
0.10000	-0.90000	-0.91451	0.01451
0.20000	-0.83000	-0.85619	0.02619
0.30000	-0.78700	-0.82245	0.03545
0.40000	-0.76830	-0.81096	0.04266
0.50000	-0.77147	-0.81959	0.04812

See figure 21.1 for a plot of these solutions. The results don't seem to accurate. Why?

Some of the advantages of the Euler Method are the following:

- Accurate early on: $O(h^2)$ for first step.
- Only need to calculate the given function $f(t, x(t))$.
- Only one evaluation of $f(t, x(t))$ needed.

But, as can be seen in figure 21.1, the Euler Method is also

- Pretty inaccurate at $t = b$.
- Cumulative truncation error: $n \times O(h^2) = O(h)$.
- This error does not include the accumulative round-off error.

So, what can we do to minimize/remove these disadvantages?

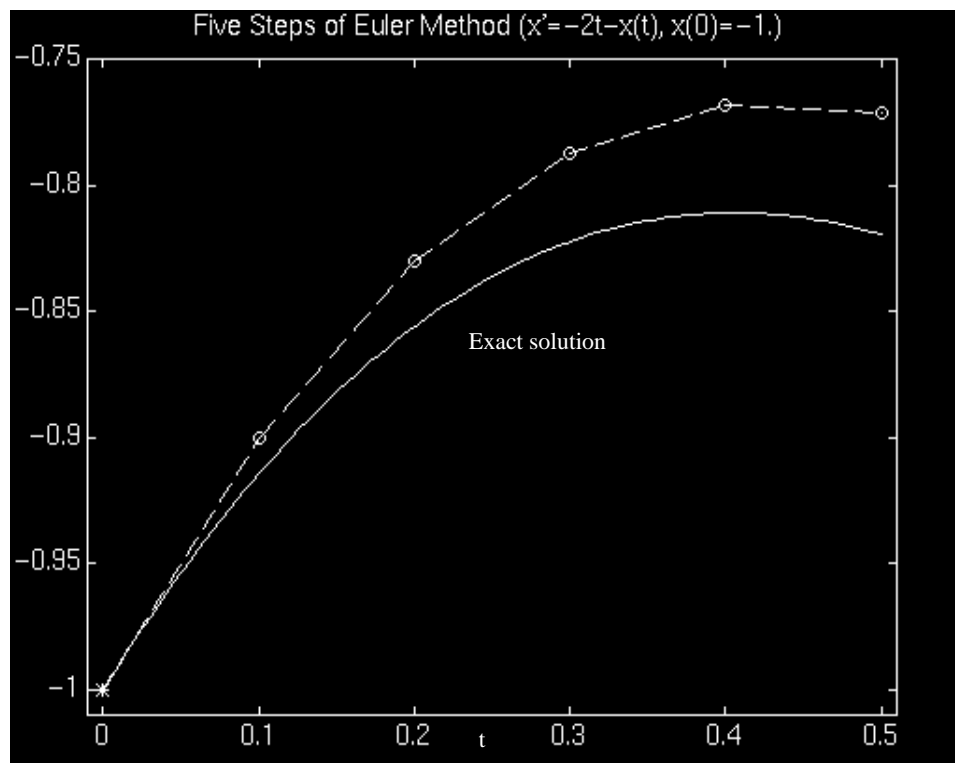


Figure 21.1: A picture of the Euler Method.

CISC 271 Class 24

Higher Order Taylor Methods

In this section, we will try to remove some of the disadvantages of the Euler Method by adding more terms to the Taylor series approximation of $x(a + h)$.

Again, the problem we want to solve can be stated as follows:

$$\text{given } x' = f(t, x), x(a), \text{ we want to find } x(b)$$

Here we will use the first 5 terms of the Taylor series (i.e., $n = 4$ - we could use any number of terms, but 4 is the standard order used) to get from $x(a)$ to $x(a + h)$:

$$x(a + h) = x(a) + hx'(a) + \frac{h^2}{2!}x''(a) + \frac{h^3}{3!}x'''(a) + \frac{h^4}{4!}x^{(iv)}(a) + O(h^5)$$

In this expansion, we will use:

$$\begin{aligned}x'(a) &\Rightarrow f(a, x(a)) \\x''(a) &\Rightarrow f'(a, x(a)) \\x''(a) &\Rightarrow f'(a, x(a)) \\x'''(a) &\Rightarrow f''(a, x(a)) \\x^{(iv)}(a) &\Rightarrow f'''(a, x(a))\end{aligned}$$

Again, as with Euler's Method, we repeat the above to get from $x(a + h)$ to $x(a + 2h)$, etc., until we reach $x(b)$.

Example 1

Suppose we want to solve the following first-order IVP:

$$x' = 1 + x^2 + t^3, x(1) = -4, \text{ and we want to find } x(2)$$

The derivatives of $f(t, x)$ are

$$\begin{aligned}x'' &= 2xx' + 3t^2 \\x''' &= 2xx'' + 2(x')^2 + 6t \\x^{(iv)} &= 2xx''' + 6x'x'' + 6\end{aligned}$$

With $n = 100$, we obtain the following solution values for $x(2)$:

- Actual: 4.3712 (5 significant digits)
- Using Euler: 4.2358541
- Using Taylor₄: 4.3712096

Example 2

Consider again

$$x' = -2t - x(t), \quad x(0) = -1, \quad x(0.5) = ?$$

The derivatives of $f(t, x)$ are

$$\begin{aligned} x'' &= -2 - x' \\ x''' &= -x'' \\ x^{(iv)} &= -x''' \end{aligned}$$

By applying Taylor₄ method, with $h = 0.1$, we find

t	$x(t)$	exact	error
0.00000	-1.00000	-1.00000	0.00000000
0.10000	-0.91451	-0.91451	0.00000025
0.20000	-0.85619	-0.85619	0.00000044
0.30000	-0.82246	-0.82245	0.00000060
0.40000	-0.81096	-0.81096	0.00000073
0.50000	-0.81959	-0.81959	0.00000082

These are plotted in figure 22.1, and compared to our results using the Euler method. Note that the single step truncation error of $O(h^5)$ leads to an excellent match. Even if we use a single step size of $5h$, as in figure 22.2, the Taylor₄ method is better than the Euler method:

t	$x(t)$	exact	error
0.00000	-1.00000	-1.00000	0.00000
0.50000	-0.82031	-0.81959	0.00072

So, how does the Taylor₄ method fair overall? As we have seen, the method is very accurate, and the cumulative truncation error of $n \times O(h^5) = O(h^4)$ is relatively low. But the method's disadvantages are the following:

- Need derivatives of $f(t, x(t))$ which might be
 - analytically difficult,
 - numerically expensive, or
 - just plain impossible
- Four evaluations for each step (as compared to just one for Euler)

So, what can we do to avoid these extra derivatives, while maintaining the accuracy of the Taylor_4 method?

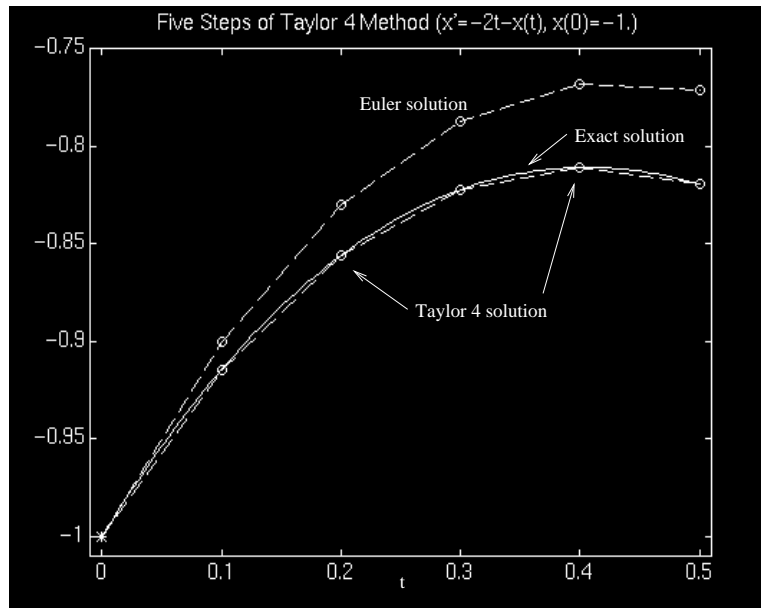


Figure 22.1: A picture of the Taylor₄ Method.

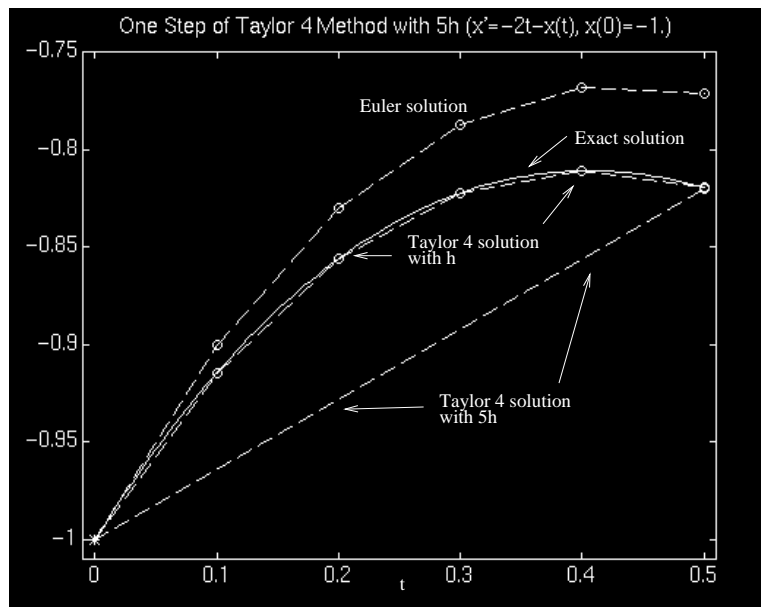


Figure 22.2: A picture of the Taylor₄ method using a single step size of $5h$.

CISC 271 Class 25

Runge-Kutta Methods

MOTIVATION

We would like to develop a method where we avoid calculating the derivatives of $f(t, x(t))$. To do this we adopt a technique similar to that used in the Secant root-finding method where the functional derivative in Newton's method was replaced by an approximation for the derivative.

Derivation approximation

In the Secant method, $f'(x)$ was approximated by

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

which, if $x_n = x_{n-1} + h$, we can modify as

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

Looking at the Taylor expansion,

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi)$$

or

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

we see that the truncation error of this approximation is $O(h)$. This type of approximation will be used below.

RUNGE-KUTTA

The idea behind the Runge-Kutta (RK) of order m is that for each set of size h , we

- Evaluate $f(t, x(t))$ at m interim stages, to
- arrive at an accuracy on the order similar to that of the Taylor method of order m .

We will give sample RK methods at $m = 2, 4$. They have the following characteristics:

- Each $f(t, x(t))$ evaluation is built upon previous function evaluations.
- The weighted average of evaluations produces $x(t + h)$.
- The error for a RK method of order m is $O(h^{m+1})$ for each step of size h .

RK2

In this RK method, $m = 2$. So, let us consider the second order Taylor approximation:

$$x(t + h) = x(t) + hx'(t) + \frac{h^2}{2}x''(t) + O(h^3)$$

or, for our IVP, $x'(t) = f(t, x)$, is

$$x(t + h) = x(t) + hf(t, x) + \frac{h^2}{2}f'(t, x) + O(h^3)$$

To remove the derivative, $f'(t, x)$ we need an approximation that does not reduce the order of accuracy of the $x(t + h)$ approximation. Therefore, use the following as a derivation approximation:

$$f'(x + h) = \frac{f(t + h, x(t + h)) - f(t, x)}{h} + O(h)$$

as inspired by our previous discussion about derivative approximations, while noting that $f(t, x)$ has two *coupled* arguments. Therefore, our $x(t + h)$ approximation becomes

$$\begin{aligned} x(t + h) &\approx x(t) + hf(t, x) + \frac{h^2}{2} \left(\frac{f(t + h, x(t + h)) - f(t, x)}{h} \right) + O(h^3) \\ &= x(t) + \left(\frac{h}{2} \right) f(t, x) + \frac{h}{2} f(t + h, x(t + h)) + O(h^3) \end{aligned}$$

Now, if we approximate $x(t + h)$ by a first order approximation: $x(t + h) = x(t) + hx'(t) + O(h^2) = x(t) + hf(t, x) + O(h^2)$, we have

$$x(t + h) \approx x(t) + \left(\frac{h}{2} \right) f(t, x) + \frac{h}{2} f(t + h, x(t) + hf(t, x)) + O(h^3)$$

All the above manipulations did not change the order of the method, while removing the derivatives of the function.

More formally stated, the RK2 is as follows.

$$x(t+h) = x(t) + \frac{1}{2} (F_1 + F_2)$$

where

$$\begin{aligned} F_1 &= hf(t, x) \\ F_2 &= hf(t+h, x+F_1) \end{aligned}$$

RK4

Although, the derivation is much more involved, the RK4 is perhaps the most commonly used RK method, and is as follows.

$$x(t+h) = x(t) + \frac{1}{6} (F_1 + 2F_2 + 2F_3 + F_4)$$

where

$$\begin{aligned} F_1 &= hf(t, x) \\ F_2 &= hf(t + \frac{1}{2}h, x + \frac{1}{2}F_1) \\ F_3 &= hf(t + \frac{1}{2}h, x + \frac{1}{2}F_2) \\ F_4 &= hf(t + h, x + F_3) \end{aligned}$$

Example

Consider again

$$x' = -2t - x(t), \quad x(0) = -1, \quad x(0.5) = ?$$

By applying RK4 method, with $h = 0.1$, we find

t	$x(t)$	exact	error
0.00000	-1.00000	-1.00000	0.00000000
0.10000	-0.91451	-0.91451	0.00000025
0.20000	-0.85619	-0.85619	0.00000044
0.30000	-0.82246	-0.82245	0.00000060
0.40000	-0.81096	-0.81096	0.00000073
0.50000	-0.81959	-0.81959	0.00000082

Note that these values are essentially identical to that for the Taylor_4 method, but without the need for higher derivatives.

OVERALL SUMMARY

If a IVP is complex and/or complicated enough, one of the above three methods may be required to find approximate solutions. They do so by producing a table of values, at a constant stepsize h . The three methods have different properties:

Euler: simple, but not too accurate.

High-order Taylor: very accurate, but require derivatives of $f(t, x(t))$.

Runge-Kutta: Same order of accuracy as Taylor, but without derivative evaluations.

The main error sources in these methods are:

- Local truncation (of Taylor series approximation)
- Local round-off (due to finite precision)
- Accumulations and combinations of the previous two.

CISC 271 Class 26

Gaussian Elimination

LINEAR SYSTEMS

Conventions

$$\begin{array}{ll}
 \text{Unknowns} - & x_0, x_1, \dots, x_n \quad \vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \\
 \text{Coefficient matrix} - & a_{00}, a_{01}, \dots, a_{mn} \quad A = \begin{bmatrix} a_{00} & \cdots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{m0} & \cdots & a_{mn} \end{bmatrix} \\
 \text{Right-hand side} - & b_0, b_1, \dots, b_m \quad \vec{b} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix}
 \end{array}$$

In the above, there are $m + 1$ rows, and $n + 1$ columns in A , which is an $(m + 1) \times (n + 1)$ array, or $A_{(m+1) \times (n+1)}$. Hence, we have $n + 1$ unknowns with $m + 1$ equations.

$A\vec{x} = \vec{b}$ is the same as

$$\begin{array}{rcl}
 a_{00}x_0 + a_{01}x_1 + \cdots + a_{0n}x_n & = & b_0 \\
 \vdots & & \\
 a_{m0}x_0 + a_{m1}x_1 + \cdots + a_{mn}x_n & = & b_m
 \end{array}$$

A solution of the system is a vector $\hat{\vec{x}}$ which satisfies $A\hat{\vec{x}} = \vec{b}$.

Multiplying two matrices:

$$C = AB \Rightarrow c_{ij} = \sum_{k=0}^n a_{ik}b_{kj}$$

where C is of order $(l + 1) \times (m + 1)$, A is of order $(l + 1) \times (n + 1)$, B is of order $(n + 1) \times (m + 1)$. Multiplication is “row of A by column of B .”

$AB \neq BA$, even for $(n+1) \times (n+1)$ matrices:

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

Properties of general matrices:

- Commutative
 $A + B = B + A$
 $AB \neq BA$
- Associate
 $(A + B) + C = A + (B + C)$
 $(AB)C = A(BC)$
- Distributive
 $A(B + C) = AB + AC$
- Scalar cA scales every entry the same amount.

Note: from here on, we will consider only square matrices. I.e., A is an $(n+1) \times (n+1)$ matrix.

Types of matrices:

- Lower triangular matrix, L
- Upper triangular matrix, U
- Diagonal matrix
- Identity matrix, I

Inverse: The inverse of A is defined to be A^{-1} such that

$$AA^{-1} = A^{-1}A = I$$

for a square matrix.

SOLVING A LINEAR SYSTEM

The problem we are trying to solve: Given A, \vec{b} and \vec{x} , such that $A \in R^{(n+1) \times (n+1)}$ and $\vec{b}, \vec{x} \in R^{n+1}$, where $A\vec{x} = \vec{b}$, find \vec{x} .

One approach: Find A^{-1} . Then

$$A^{-1}(A\vec{x}) = A^{-1}\vec{b} \Rightarrow \vec{x} = A^{-1}\vec{b}.$$

Difficulties:

- What is the algorithm? (i.e, How to find A^{-1} ? Direct solutions are increasingly complicated as n gets large.)
- How good/fast is the algorithm?

ANOTHER APPROACH

One of the properties of a linear system is that

$$A\vec{x} = \vec{b} \iff BA\vec{x} = B\vec{b}$$

if B is one of the following invertible matrices:

- B : permutation matrix
{exchange rows (equations)}
- B : identity matrix with constants along the diagonal
{multiply a row (along with the corresponding value of \vec{b}) by a constant}
- B : arbitrary 0,1 matrix, invertible
{adding row to another}

This is based on formulations based on isolated equations.

SOLVING A SYSTEM BY GAUSSIAN ELIMINATION

(1.) Can we solve

$$(a_{00})x_0 = b_0 ?$$

This is trivial.

(2.) Can we solve

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} ?$$

The problem is that the equations interact.

Solution: convert row 1 to the form in (1.) by adding (row 0) * $\left(\frac{-a_{10}}{a_{00}}\right)$ to (row 1).

$$\begin{bmatrix} a_{00} & a_{01} \\ 0 & c_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

We can now find x_1 , and substitute it back into the first row (first equation).

(3.) Can we solve

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} ?$$

By adding (row 0) $\left(\frac{-a_{20}}{a_{00}}\right)$ to row 2, we can make it

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b'_2 \end{bmatrix}$$

But what next? How do we get rid of c_{21} ?

The insight is that to zero c_{21} , we must *first* zero a_{10} . Otherwise, it effects c_{20} .

So by adding (row 0) $\left(\frac{-a_{10}}{a_{00}}\right)$ to row 1, we form

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & c_{11} & c_{12} \\ 0 & c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b'_1 \\ b'_2 \end{bmatrix}$$

and then by adding (row 1) $\left(\frac{-a_{21}}{c_{11}}\right)$ to row 2, we can finally get (i.e., solve the 2×2 array)

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & c_{11} & c_{12} \\ 0 & 0 & d_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b'_1 \\ b''_2 \end{bmatrix}$$

This procedure leads us to the following algorithm.

GAUSSIAN ELIMINATION ALGORITHM

```

for i = 0 to n-1
  for j = i + 1 to n
    row j = row j - a[j][i]*(row i/a[i][i]);
  end for
end for

```

Note that row i contains b_i . This produces a strictly upper-triangular matrix equivalent to A .

Live example:

$$\begin{bmatrix} 2 & -2 & -4 \\ -1 & 2 & 3 \\ 3 & -1 & -5 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -4 \\ 3 \\ -6 \end{bmatrix}$$

Use Gaussian Elimination on row $j = 1$ (at $i = 0$):

$$\text{first entry in row: } \Rightarrow -1 - \left(\frac{2}{2}\right)(-1) = 0$$

$$\text{second entry in row: } \Rightarrow 2 - \left(\frac{-2}{2}\right)(-1) = 1$$

$$\text{third entry in row: } \Rightarrow 3 - \left(\frac{-4}{2}\right)(-1) = 1$$

$$\text{last entry in row: } \Rightarrow 3 - \left(\frac{-4}{2}\right)(-1) = 1$$

to give

$$\begin{bmatrix} 2 & -2 & -4 \\ 0 & 1 & 1 \\ 3 & -1 & -5 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -4 \\ 1 \\ -6 \end{bmatrix}$$

Use Gaussian Elimination on row $j = 2$ (at $i = 0$):

$$\text{first entry in row: } \Rightarrow 3 - \left(\frac{2}{2}\right)(3) = 0$$

$$\text{second entry in row: } \Rightarrow -1 - \left(\frac{-2}{2}\right)(3) = 2$$

$$\text{third entry in row: } \Rightarrow -5 - \left(\frac{-4}{2}\right)(3) = 1$$

$$\text{last entry in row: } \Rightarrow -6 - \left(\frac{-4}{2}\right)(3) = 0$$

to give

$$\begin{bmatrix} 2 & -2 & -4 \\ 0 & 1 & 1 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix}$$

Finally, use Gaussian Elimination on row $j = 2$ (at $i = 1$):

$$\text{first entry in row: } \Rightarrow 0 - \left(\frac{0}{1}\right)(2) = 0$$

$$\text{second entry in row: } \Rightarrow 2 - \left(\frac{1}{1}\right)(2) = 0$$

$$\text{third entry in row: } \Rightarrow 1 - \left(\frac{1}{1}\right)(2) = -1$$

$$\text{last entry in row: } \Rightarrow 0 - \left(\frac{1}{1}\right)(2) = -2$$

to give

$$\begin{bmatrix} 2 & -2 & -4 \\ 0 & 1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -4 \\ 1 \\ -2 \end{bmatrix}$$

This solves to

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

BACK-SUBSTITUTION

Suppose that A is square and upper-triangular, and we have $A\vec{x} = \vec{b}$. With exact arithmetic, the system can be solved easily:

$$\begin{aligned} x_n &= \frac{b_n}{a_{nn}} \\ x_k &= \frac{b_k - \sum_{j=k+1}^n a_{kj}x_j}{a_{kk}} \end{aligned}$$

E.g.,

$$\begin{aligned} 4x_0 - 3x_1 + x_2 &= 8 & x_0 &= 6\frac{1}{4} \\ x_1 + 2x_2 &= 1 & \Rightarrow x_1 &= 5 \\ -2x_2 &= 4 & x_2 &= -2 \end{aligned}$$

AUGMENTED ARRAYS

We can represent $A\vec{x} = \vec{b}$ as a new $(n+1) \times (n+2)$ matrix $V = [A|\vec{b}]$. Under the set of operations,

- interchange of rows
- multiplication of a row by a non-zero constant
- adding a multiple of a row to another

leads to another matrix $W = [C|\vec{d}]$ where V and W are *equivalent*, i.e., the solution of one is the solution of the other.

CISC 271 Class 27

Gaussian Elimination - Pivoting and Scaling

Previously we exploited the following three properties of systems of equations to solve $A\vec{x} = \vec{b}$, noting that the solution does not change under the following:

- interchange of rows
- multiplication of a row by a non-zero constant
- adding a multiple of a row to another

We generalized the process:

1. Write down the augmented matrix $[A|\vec{b}]$.
2. Using the elementary operations above, reduce $[A|\vec{b}]$ to an upper triangular matrix.
3. Use backsubstitution to solve for \vec{x} .

DIFFICULTIES

Example

$$A = \begin{bmatrix} 7 & 63 & 0 \\ 2 & 18 & 10 \\ 3 & 30 & 0 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 13.3 \\ 3.9 \\ 6.0 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 63 & 0 & 13.3 \\ 2 & 18 & 10 & 3.9 \\ 3 & 30 & 0 & 6.0 \end{bmatrix} \rightarrow \begin{bmatrix} 7 & 63 & 0 & 13.3 \\ 0 & 0 & 10 & 0.1 \\ 0 & 3 & 0 & 0.3 \end{bmatrix}$$

For the next step, we need $(row1)/a_{11} = (row1)/0$, the elimination fails!

To overcome this problem, we need to interchange the last two rows (which does not change the solution), so that $a_{11} \neq 0$.

$$\rightarrow \begin{bmatrix} 7 & 63 & 0 & 13.3 \\ 0 & 3 & 0 & 0.3 \\ 0 & 0 & 10 & 0.1 \end{bmatrix}$$

This process is called *pivoting*. The entry at (i, i) is called the *pivot* or *pivot element*.

A second related type of problem can be seen in the following example.

Example

Solve $A\vec{x} = \vec{b}$ such that

$$A = \begin{bmatrix} 20 & 8 & -4 \\ 10 & 3.9999 & 15 \\ 5 & 6 & 12 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 52 \\ 8.9999 \\ 4 \end{bmatrix}$$

True solution is $\vec{x} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}$

Let's compute the solution by Gaussian Elimination in $F(10, 5, -50, 50)$, with chopping.

$$\begin{aligned} \left[\begin{array}{cccc} 20 & 8 & -4 & 52 \\ 10 & 3.9999 & 15 & 8.9999 \\ 5 & 6 & 12 & 4 \end{array} \right] &\rightarrow \left[\begin{array}{cccc} 20 & 8 & -4 & 52 \\ 0 & -.0001 & 17 & fl(-17.0001) \\ 0 & 4 & 13 & -9 \end{array} \right] \\ &\rightarrow \left[\begin{array}{cccc} 20 & 8 & -4 & 52 \\ 0 & -.0001 & 17 & -17 \\ 0 & 0 & 680010 & -680000 \end{array} \right] \end{aligned}$$

Note that $fl(-17.0001) = -17$.

The computed solution is thus

$$\begin{aligned} x_2 &= \frac{-680000}{680010} = -0.99998 \\ x_1 &= \frac{-17 - 17x_2}{-0.0001} \\ &= \frac{-17 - (-16.999)}{-0.0001} = 10 \\ x_0 &= \frac{52 - (-4)x_2 - 8x_1}{20} \\ &= \frac{52 - (3.9999) - 80}{20} = -1.5999 \end{aligned}$$

$$\text{or } \vec{x} = \begin{bmatrix} -1.5999 \\ 10.0 \\ -.99998 \end{bmatrix}$$

In this example, Gaussian Elimination does not fail, but the computed solution has a very large error.

Why? In the Gaussian Elimination process, all the pivots are non-zero, so that the method does not fail. However, one pivot used is too small, namely at the second step, $a_{11} = -.0001$, so that a_{12}/a_{11} may be very large.

Since some entries in row 1 are divided by a_{11} , a small round-off error in an entry (i.e., when -17.0001 is replaced by -17) will become a large error in the result.

Again, using pivoting may avoid this error.

$$\rightarrow \begin{bmatrix} 20 & 8 & -4 & 52 \\ 10 & 3.9999 & 15 & 8.9999 \\ 5 & 6 & 12 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 20 & 8 & -4 & 52 \\ 0 & -.0001 & 17 & -17.0001 \\ 0 & 4 & 13 & -9 \end{bmatrix}$$

At this point pivoting is used to exchange the last two rows.

$$\begin{aligned} &\rightarrow \begin{bmatrix} 20 & 8 & -4 & 52 \\ 0 & 4 & 13 & -9 \\ 0 & -.0001 & 17 & -17.0001 \end{bmatrix} \\ &\rightarrow \begin{bmatrix} 20 & 8 & -4 & 52 \\ 0 & 4 & 13 & -9 \\ 0 & 0 & 17.0003 & -17 \end{bmatrix} \\ &\rightarrow \begin{bmatrix} 20 & 8 & -4 & 52 \\ 0 & 4 & 13 & -9 \\ 0 & 0 & 17 & -17 \end{bmatrix} \\ &\rightarrow \vec{x} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} \end{aligned}$$

We have an accurate solution this time (it is exact by accident).

PIVOTING

In Gaussian Elimination, as we saw in the previous two examples, we may encounter two difficulties.

1. A pivot is zero, and we cannot continue the process.
2. A pivot is small, and we may have an instability and large error in the solution.

To avoid these difficulties, we use the pivoting technique.

Consider that we are at the k th step in Gaussian Elimination where we have a pivot at a_{kk} . Before dividing row k by a_{kk} , we look for an entry in the remaining part of matrix, a_{ij} , ($i \geq k, j \geq k$), such that

$$|a_{ij}| = \max \forall i, j \geq k.$$

Suppose that a_{IJ} is such an entry, then we interchange

$$\text{row } k \leftrightarrow \text{row } I \quad \text{column } k \leftrightarrow \text{column } J$$

so that a_{IJ} is the new pivot, and proceed as before.

Complete pivoting or total pivoting. It works well but

1. Expensive. Need to locate maximum in the whole matrix.
2. Need to interchange columns – also need to reorder the unknown variables.

e.g., column $k \leftrightarrow$ column J then $x_k \leftrightarrow x_J$

PARTIAL PIVOTING

At the k th step of Gaussian Elimination:

1. Locate the entry which is maximum in magnitude along column k .

$$\left. \begin{array}{c} a_{kk} \\ a_{k+1k} \\ \vdots \\ a_{nk} \end{array} \right\} |a_{Ik}| = \max.$$

2. Interchange row k with row I .
3. Continue the elimination as before.

Advantages.

1. If A is nonsingular, then Gaussian Elimination with partial pivoting always works. I.e., after pivoting, the new pivot must be non-zero. The only way the new pivot could be zero is if $a_{jk}^{(k)} = 0$, for $j = k, \dots, n$, for which A would have to be singular.

If a matrix is *singular* then

- There is no unique solution to the system of linear equations represented by A .
 - After the partial pivoting, there is a zero on the diagonal.
 - The rows of A are *not* linearly independent. At least one row can be expressed as a linear combination of one or more of the other rows.
2. With partial pivoting, Gaussian Elimination is stable for *most problems*. (With total pivoting, Gaussian Elimination is *always* stable).

Algorithm (Gaussian Elimination with Partial Pivoting)

```
Input n, A[n][n], b[n];
for k = 0,1,...,n-1 do
  /* pivoting */
  c = |a[k][k]|;
  p = k;
  /* locate max */
  for i = k+1,k+2,...,n
    if (|(a[i][k]| > c) then
      c = |a[i][k]|;
      p = i;
    end if
  end for

  /* exchange row k and row p */
  for j = k,k+1,...,n
    tmp = a[k][j];
    a[k][j] = a[p][j];
    a[p][j] = tmp;
  end for
  tmp = b[k];
  b[k] = b[p];
  b[p] = tmp;

  /* continue with Gaussian Elimination */
  for i = k+1, k+2, ..., n
    m[i][k] = a[i][k]/a[k][k];
    for j = k+1, k+2, ..., n
      a[i][j] = a[i][j] - m[i][k]*a[k][j];
    end for;
    b[i] = b[i] - m[i][k]*b[k];
  end for;
end for
back_substitution() /* as before */
```

In implementation, there are two ways to do the interchange of rows (and columns in the case of full pivoting):

1. Real: Do the actual interchange, like in our pseudocode above.
2. Virtual: Swap the pointers (indices) instead of values.

CISC 271 Class 28

Error Analysis

Some results to be aware of:

- $(\vec{x} \cdot \vec{y}) = x_0y_0 + x_1y_1 + \cdots + x_ny_n$
- Cauchy-Schwartz inequality:

$$|(\vec{x} \cdot \vec{y})|^2 \leq (\vec{x} \cdot \vec{x})(\vec{y} \cdot \vec{y})$$

- L_2 - norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_{i=0}^n |x_i|^2} = \sqrt{(\vec{x} \cdot \vec{x})}$$

Therefore, Cauchy-Schwartz: $|(\vec{x} \cdot \vec{y})| \leq \|\vec{x}\|_2 \|\vec{y}\|_2$.

- Unitary matrix:

$$UU^* = U^*U = I$$

$$\text{if } U \in R^{n+1 \times n+1} \quad U^* = U^T$$

U^T represents the transpose of the matrix U .

- Eigenvalues(λ) and Eigenvectors(\vec{v}) related by

$$A\vec{v} = \lambda\vec{v}$$

rearranged:

$$A\vec{v} - \lambda\vec{v} = 0$$

$$(A - \lambda I)\vec{v} = 0$$

Since in general $\vec{v} \neq 0$,

$$A - \lambda I = 0$$

this equation is satisfied by the eigenvalues for the matrix A .

- $f_A(\lambda) = \det(A - \lambda I)$
For example, $A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$, $f_A(\lambda) = (\lambda - 1)^2$. We set $f_A(\lambda) = 0$ and solve for λ to find the eigenvalues of A .

Extra Notes

- A is similar to B iff

$$A = P^{-1}BP$$

Therefore, they have the same eigenvalues.

$$A\vec{x} = \lambda\vec{x} \Rightarrow P^{-1}BP\vec{x} = \lambda\vec{x} \Rightarrow B(P\vec{x}) = \lambda(P\vec{x})$$

and if we replace \vec{x} by $P^{-1}\vec{z}$,

$$\Rightarrow B\vec{z} = \lambda\vec{z}$$

The corresponding eigenvector of B is $P\vec{x}$.

- $\text{trace}(A) = \sum_{i=1}^n a_{ii}$. (diagonal sum)
- If $A = P^{-1}BP$ then

$$\Rightarrow \text{trace } A = \text{trace } B$$

$$\Rightarrow \det A = \det B$$

$$\Rightarrow f_A(\lambda) = f_B(\lambda)$$

in $f_A(\lambda)$.

Theorem: If A is of order n (i.e., an $n \times n$ matrix), then $\exists U$, a unitary matrix, such that $T = U^*AU$, where T is an upper triangular matrix.

Corollary: $f_A(\lambda) = f_T(\lambda) = (t_{11} - \lambda)(t_{22} - \lambda) \cdots (t_{nn} - \lambda)$

Proof: $U^* = U^{-1} \Rightarrow A$ is similar to T .

End of Extra Notes

NORMS

Let $\vec{x} \in V$, a vector space. Also, let $N(\vec{x})$ be the norm of \vec{x} . Then it has the following properties:

$$(N1) \quad N(\vec{x}) \geq 0 \quad (0 \text{ iff } \vec{x} = 0)$$

$$(N2) \quad N(\alpha\vec{x}) = |\alpha|N(\vec{x}), \text{ where } \alpha \text{ is a scalar}$$

$$(N3) \quad N(\vec{x} + \vec{y}) \leq N(\vec{x}) + N(\vec{y})$$

Example Consider $V = \mathbb{R}^{n+1}$ The following are possible norms:

- $N(\vec{x}) = \sum_{i=0}^n |x_i| \equiv \|\vec{x}\|_1$
- $N(\vec{x}) = \sqrt{\sum_{i=0}^n |x_i|^2} \equiv \|\vec{x}\|_2$
- $N(\vec{x}) = (\sum_{i=0}^n |x_i|^p)^{\frac{1}{p}} \equiv \|\vec{x}\|_p$
- $N(\vec{x}) = \max_{i=0..n} \{|x_i|\} \equiv \|\vec{x}\|_\infty$

Example

$$\vec{x} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} \quad \|\vec{x}\|_1 = 6, \quad \|\vec{x}\|_2 = \sqrt{14} \approx 3.74, \quad \|\vec{x}\|_\infty = 3$$

MATRIX NORMS

In addition to the above vector norm properties (N1)-(N3), matrix norms satisfy:

$$(N4) \quad \|AB\| \leq \|A\| \|B\|$$

$$(N5) \quad \|A\vec{x}\| \leq \|A\| \|\vec{x}\|_v$$

where \vec{x} is a vector with the vector norm.

Example The Frobenius norm of A:

$$F(A) = \sqrt{\sum_{ij=0}^n |a_{ij}|^2}$$

Usually when given a vector space with a vector norm $\|\cdot\|_v$, e.g., $\|\cdot\|_\infty$ or $\|\cdot\|_2$, the associated matrix norm can be defined by

$$\|A\| = \text{Max}_{\vec{x} \neq 0} \frac{\|A\vec{x}\|_v}{\|\vec{x}\|_v}$$

Several Notes:

1. We can think of A as a mapping (transform)

$$\vec{x} \mapsto A\vec{x}$$

Thus $\|A\|$ indicates how much a vector \vec{x} can be amplified under the transform.

$$\text{E.g. } A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} : \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \mapsto \begin{bmatrix} x_0 \\ 2x_1 \end{bmatrix}$$

Then the norm of a vector can be amplified by a factor of 2, therefore $\|A\|_\infty = 2$.

2. If $A = [a_{ij}]$, then $\|A\|$ depends on the norm defined for the vectors

$$\begin{aligned} \|A\|_\infty &= \max \frac{\|A\vec{x}\|_\infty}{\|\vec{x}\|_\infty} = \max_i \sum_{j=0}^n |a_{ij}| && \text{largest row sum} \\ \|A\|_2 &= \max \frac{\|A\vec{x}\|_2}{\|\vec{x}\|_2} = \max_i |\lambda_i| && \text{largest eigenvalue of } A. \end{aligned}$$

$$\text{Example } A = \begin{bmatrix} -3 & 1 \\ -2 & 3 \end{bmatrix} \text{ with eigenvalues of } A = -\sqrt{7}, \sqrt{7}$$

$$\begin{aligned} \|A\|_\infty &= \max \left\{ \begin{array}{l} |-3| + 1, \\ |-2| + 3 \end{array} \right\} = 5 \\ \|A\|_2 &= \sqrt{7} \end{aligned}$$

$$\text{Example } A = \begin{bmatrix} d_0 & & 0 \\ & \ddots & \\ 0 & & d_n \end{bmatrix} \text{ such that } \|A\|_\infty = \|A\|_2 = \max_i |d_i|.$$

3. If A is nonsingular, then A^{-1} is also a matrix, so we can find $\|A^{-1}\|$.

$$\text{E.g., } A = \begin{bmatrix} d_0 & & 0 \\ & \ddots & \\ 0 & & d_n \end{bmatrix}, d_i \neq 0 \Rightarrow A^{-1} = \begin{bmatrix} d_0^{-1} & & 0 \\ & \ddots & \\ 0 & & d_n^{-1} \end{bmatrix},$$

$$\text{such that } \|A^{-1}\|_\infty = \|A^{-1}\|_2 = \max_i |d_i^{-1}| = \frac{1}{\min_i |d_i|}$$

CONDITION NUMBER

For any nonsingular matrix, define condition number as

$$\text{cond}(A) = K(A) = \|A\| \|A^{-1}\|.$$

If A is diagonal, $K(A) = \frac{\max_i |d_i|}{\min_i |d_i|}$.

If the $\|\cdot\|_2$ - norm is used then

$$K(A) = \max |\lambda_i| \max |\lambda_i^{-1}| = \frac{\max |\lambda_i|}{\min |\lambda_i|}, \lambda_i - \text{eigenvalue}$$

There is a limit on the condition number: $K(A) \geq 1$. The identity matrix has $K(I) = 1$.

ERROR ANALYSIS AND MATRIX NORMS

Consider $A\vec{x} = \vec{b}$ which has \vec{x}^* as its true solution. Using Gaussian Elimination in a floating-point system, we obtain a solution $\hat{\vec{x}}$. Therefore, the error is

$$\vec{e} = \vec{x}^* - \hat{\vec{x}} = \begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{bmatrix}.$$

We want to know how big \vec{e} is.

Norm of \vec{e} ?

$$\begin{aligned} \|\vec{e}\|_\infty &= \max_{0 \leq i \leq n} |e_i| \\ \|\vec{e}\|_2 &= \sqrt{e_0^2 + e_1^2 + \cdots + e_n^2} \end{aligned}$$

But \vec{e} is unknown, since \vec{x}^* is unknown. Instead, we may have another indication of how good a solution is.

$$\begin{aligned} \vec{x}^* - \text{solution} &\Rightarrow A\vec{x}^* = \vec{b} \Rightarrow \vec{b} - A\vec{x}^* = 0 \\ \hat{\vec{x}} - \text{computed solution} &\Rightarrow \vec{b} - A\hat{\vec{x}} \text{ if } \vec{b} - A\hat{\vec{x}} = 0 \Rightarrow \hat{\vec{x}} = \vec{x}^* \end{aligned}$$

Intuitively, if $\vec{b} - A\hat{\vec{x}}$ is small $\Rightarrow \hat{\vec{x}}$ - good solution.

Define:

$$\vec{r} = \vec{b} - A\hat{\vec{x}} \text{ - residual vector}$$

If $\vec{r} = 0 \Rightarrow \hat{\vec{x}} = \vec{x}^*$ - exact solution. But

$$\vec{r} = \text{small} \stackrel{?}{\Rightarrow} \text{error } \vec{e} = \vec{x}^* - \hat{\vec{x}} \text{ is small}$$

RELATION BETWEEN \vec{r} AND \vec{e}

$$\vec{e} = \vec{x}^* - \hat{\vec{x}}$$

$$A\vec{e} = A\vec{x}^* - A\hat{\vec{x}} = \vec{b} - A\hat{\vec{x}} = \vec{r}$$

I.e.,

$$\vec{r} = A\vec{e}$$

Question: \vec{r} small $\stackrel{?}{\Leftrightarrow}$ \vec{e} small.

Answer: Depends on A .

Example

Suppose we have the linear system of equations:

$$A\vec{x} = \begin{bmatrix} 2.0001 & -1 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 7.0003 \\ -7 \end{bmatrix}$$

Suppose two methods (method a and method b) give the answers

$$\begin{array}{lll} \hat{\vec{x}}_a = \begin{bmatrix} 2.91 \\ -1.01 \end{bmatrix} & \text{with} & \vec{r}_a = \begin{bmatrix} .170009 \\ -.17 \end{bmatrix} & \text{with} & \vec{e}_a = \begin{bmatrix} 0.09 \\ 0.01 \end{bmatrix} \\ \hat{\vec{x}}_b = \begin{bmatrix} 2 \\ -3 \end{bmatrix} & \text{with} & \vec{r}_b = \begin{bmatrix} .0001 \\ 0 \end{bmatrix} & \text{with} & \vec{e}_b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{array}$$

but the true answer is

$$\vec{x}^* = \begin{bmatrix} 3 \\ -1 \end{bmatrix} !$$

So, in general,

$$\begin{array}{l} \text{a small } \vec{r} \not\Rightarrow \text{a small } \vec{e} \\ \text{and a small } \vec{e} \not\Rightarrow \text{a small } \vec{r} \end{array}$$

Observations

It seems that if A is a diagonal matrix

$$A = \begin{bmatrix} d_0 & & 0 \\ & \ddots & \\ 0 & & d_n \end{bmatrix}$$

then \vec{r} and \vec{e} have similar size *only* when d_0, \dots, d_n all have similar size. In other words,

$$K(A) = \frac{\text{largest } d_i}{\text{smallest } d_i} \approx 1$$

On the other hand, if $K(A) = \frac{\text{largest } d_i}{\text{smallest } d_i} = \text{large}$, then one cannot draw any conclusion about the size of \vec{e} from the size of \vec{r} and vice versa.

BOUNDS ON RELATIVE ERROR

Let us consider an upper bound on

$$\frac{\|\vec{e}\|}{\|\vec{x}^*\|} = \frac{\|\vec{x}^* - \hat{\vec{x}}\|}{\|\vec{x}^*\|}.$$

To make things easy, we could use the L_1 – norm:

$$\|\vec{x}\|_1 = \sum_{i=0}^n |x_i|, \quad \|A_{(m+1) \times (n+1)}\|_1 = \max_{i=0}^m \sum_{j=0}^n |a_{ij}|$$

The properties of this norm, as with all the norms, that are useful is that

$$\|A\vec{x}\|_1 \leq \|A\|_1 \|\vec{x}\|_1 \text{ and } \|\vec{y} + \vec{z}\| \leq \|\vec{y}\| + \|\vec{z}\|$$

This entails that

$$\begin{aligned} \|A^{-1}\vec{r}\| &\leq \|A^{-1}\| \|\vec{r}\| \\ \Rightarrow \|\vec{x}^* - \hat{\vec{x}}\| &\leq \|A^{-1}\| \|\vec{r}\| \end{aligned}$$

since $A^{-1}\vec{r} = A^{-1}A\vec{e} = \vec{e}$.

Since $\|\vec{b}\| = \|A\vec{x}^*\|$ and $\|A\vec{x}^*\| \leq \|A\| \|\vec{x}^*\|$, we know that

$$\frac{\|\vec{b}\|}{\|A\|} \leq \|\vec{x}^*\| \text{ so that } \frac{\|A\|}{\|\vec{b}\|} \geq \|\vec{x}^*\|^{-1}$$

Multiplying the inequalities (all positive values), we get

$$\frac{\|\vec{e}\|}{\|\vec{x}^*\|} \leq \|A\| \|A^{-1}\| \frac{\|\vec{r}\|}{\|\vec{b}\|} = K(A) \frac{\|\vec{r}\|}{\|\vec{b}\|}$$

The condition number, $K(A)$ represents the amount that the relative residual is magnified.

Indeed, we can show that

$$\frac{1}{K(A)} \frac{\|\vec{r}\|}{\|\vec{b}\|} \leq \frac{\|\vec{e}\|}{\|\vec{x}^*\|} \leq K(A) \frac{\|\vec{r}\|}{\|\vec{b}\|}$$

The first and last terms are the relative residuals, and the central term is the relative error.

If $K(A) = 1$, as would be expected for the identity matrix, then

$$\frac{\|\vec{r}\|}{\|\vec{b}\|} = \frac{\|\vec{e}\|}{\|\vec{x}^*\|}$$

such that the relative residual is equal to the relative error.

The relative residual and relative error have roughly the same size if $K(A) \sim$ small.

If $K(A)$ is large, the size of the relative residual and relative error may be quite different. One may be very large while the other is small.

ERROR IN GAUSSIAN ELIMINATION

It can be shown that in Gaussian Elimination,

$$\text{relative error: } \frac{\|\vec{e}\|}{\|\vec{x}^*\|} \approx K(A) \cdot \mu,$$

Consider an $\hat{\vec{x}}$ which is the computed solution of $A\vec{x} = \vec{b}$ using Gaussian Elimination. The exact solution is \vec{x}^* . Then we can show that

$$\frac{\|\vec{x}^* - \hat{\vec{x}}\|}{\|\vec{x}^*\|} \approx K(A) \cdot \mu,$$

where μ is the machine epsilon.

Therefore, if a system is *ill-conditioned*, i.e.,

$$K(A) \gg 1,$$

then the relative error in the computed solution may be very large, and there is almost certainly a lot of round-off error.

For example, consider the Hilbert matrix, H_{n+1} , whose entries are $\{a_{ij}\}$ where $a_{ij} = \frac{1}{i+j+1}$.

n	$K(H_{n+1})$
2	5×10^2
5	1×10^7
6	4×10^8
9	1×10^{13}

Thus the matrix is ill-conditioned.

CISC 271 Class 29

Linear-System Computations

COMPUTING INVERSES

A method for computing the inverse is to consider A^{-1} in $AA^{-1} = I$ as an unknown matrix,

$$A^{-1} = \{b_{ij}\}$$

such that

$$A \begin{bmatrix} b_{00} \\ b_{10} \\ \vdots \\ b_{n0} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad A \begin{bmatrix} b_{01} \\ b_{11} \\ \vdots \\ b_{n1} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{etc.}$$

Solve all these systems to find $\{b_{ij}\}$. In place of the original A , the U generated via the Gaussian Elimination can be used.

Example.

Find the inverse of

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 0 \\ 2 & 1 & 4 \end{bmatrix}$$

Form an augmented matrix, and use Gaussian Elimination

$$\left[\begin{array}{ccc|ccc} 2 & 1 & 3 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 & 1 & 0 \\ 2 & 1 & 4 & 0 & 0 & 1 \end{array} \right]$$
$$\left[\begin{array}{ccc|ccc} 2 & 1 & 3 & 1 & 0 & 0 \\ 0 & 3/2 & -3/2 & -1/2 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{array} \right]$$

Then solve for the b_{ij} 's.

$$b_{20} = -1$$

$$\begin{aligned}
b_{10} &= \frac{-1/2 - (-3/2)(-1)}{3/2} = -4/3 \\
b_{00} &= \frac{1 - (3)(-1) - (1)(-4/3)}{2} \\
&= \frac{3/3 + 9/3 + 4/3}{2} = 8/3 \\
b_{21} &= 0 \\
b_{11} &= \frac{1 - (-3/2)(0)}{3/2} = 2/3 \\
b_{01} &= \frac{0 - (3)(0) - (1)(2/3)}{2} = -1/3 \\
&\text{etc.}
\end{aligned}$$

Such that

$$A^{-1} = \begin{bmatrix} 8/3 & -1/3 & -2 \\ -4/3 & 2/3 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

LU FACTORIZATION

Gaussian elimination is not the only *direct* method for solving a linear system. Another method is called LU decomposition. Consider the following matrix product: $A = LU$ where L is a lower triangular matrix, and U is an upper triangular matrix. Actually, the LU pair can take an infinite number of forms and by convention we use (to define a unique decomposition)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{10} & 1 & 0 & 0 \\ l_{20} & l_{21} & 1 & 0 \\ l_{30} & l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{00} & u_{01} & u_{02} & u_{03} \\ 0 & u_{11} & u_{12} & u_{13} \\ 0 & 0 & u_{22} & u_{23} \\ 0 & 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

We have actually already calculated U : it is the resultant upper diagonal matrix that we get using Gaussian Elimination. And if we define the following multipliers

$$m_{ik} = \frac{a_{ik}}{a_{kk}}, \quad i = k + 1, \dots, n$$

where k is the k th row from Gaussian Elimination at the k -th step. Then L is

$$L = \begin{bmatrix} 1 & & & 0 \\ m_{10} & 1 & & \vdots \\ \vdots & & \ddots & \ddots \\ m_{n0} & m_{n1} & \cdots & 1 \end{bmatrix}$$

From our previous example, where we found the inverse of

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 0 \\ 2 & 1 & 4 \end{bmatrix},$$

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

To solve $A\vec{x} = \vec{b}$ by LU :

$$LU\vec{x} = \vec{b} \equiv \left. \begin{array}{l} L\vec{y} = \vec{b} \\ U\vec{x} = \vec{y} \end{array} \right\} \text{ Forward and Backward Substitution}$$

Forward Substitution: $L\vec{y} = \vec{b}$

$$\begin{bmatrix} 1 & & & 0 \\ l_{10} & 1 & & \\ \vdots & \ddots & \ddots & \\ l_{n0} & l_{n1} & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Therefore, $y_0 = b_0, y_i = b_i - \sum_{k=0}^{i-1} l_{ik}y_k, i = 1, \dots, n$.

Backward Substitution: $U\vec{x} = \vec{y}$

$$\begin{bmatrix} u_{00} & u_{01} & \cdots & u_{0n} \\ & u_{11} & & \vdots \\ & & \ddots & \\ 0 & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Therefore, $x_n = \frac{y_n}{u_{nn}}, x_i = \frac{(y_i - \sum_{k=i+1}^n u_{ik}x_k)}{u_{ii}}, i = n-1, \dots, 0$.

Actually, only one array is needed to store both L and U , since the m_{ij} values can be stored in the place of the 0's in the lower half of U . Further, if there are a number of \vec{b} 's to be solved for, they can be stored in consecutive columns in the augmented matrix.

So, to find the solution of $A\vec{x} = \vec{b}$:

1. Do LU factorization using either Gaussian Elimination or the explicit formulation. This step is independent of the right hand side (\vec{b} 's), and no extra storage is needed since L is stored in the lower half of A , and U in the upper half.
2. Forward/Backward Substitution.
3. If there are multiple right hand sides, one need only LU factorize A once. Then the backward/forward substitution is performed for each \vec{b} .
4. If pivoting is used during the LU factorization, an additional array must be used to save the information on which rows have been exchanged. This information is needed for the subsequent backward/forward substitution.

There are also iterative methods for solving $A\vec{x} = \vec{b}$, but we won't cover those. We have covered four direct methods here.

Extra Notes

Second Method for finding LU .

There is another manner in which to find the LU decomposition, directly.

0th row: $u_{10} = a_{10}, u_{11} = a_{11}, \dots, u_{1n} = a_{1n}$.

0th column: $l_{10}u_{00} = a_{10} \Rightarrow l_{10} = \frac{a_{10}}{a_{00}}$
 $l_{i0}u_{00} = a_{i0} \Rightarrow l_{i0} = \frac{a_{i0}}{a_{00}} \quad \forall i = 1, \dots, n$

For $i \geq 0$:

$$\begin{aligned} \text{row } i : \quad u_{ij} &= a_{ij} - \sum_{k=0}^{i-1} l_{ik}u_{kj} \quad j = i, \dots, n \\ \text{column } i : \quad l_{ji} &= \frac{(a_{ji} - \sum_{k=0}^{i-1} l_{jk}u_{ki})}{u_{ii}} \quad j = i+1, \dots, n \end{aligned}$$

This process of decomposition can be completed if $U_{ii} \neq 0 \forall i$. If A is nonsingular, then $U_{ii} \neq 0$ if pivoting is used.

End of Extra Notes

CISC 271 Class 30

Functional Approximation, Minimax

APPROXIMATION

The problem we want to solve: Given a function $f(x)$ on $[a, b]$, approximate it by a polynomial $P_n(x)$.

How does this differ from Polynomial Interpolation?

- *Interpolation* – find polynomial to interpolate $f(x)$ at some points on $[a, b]$.
 - Main concern: $P_n(x)$ and $f(x)$ have same value at some points.
 - We *hope* $P_n(x)$ is close to $f(x)$ at the other points, but we don't really care what the error is at the other points.
- *Approximation*
 - Main concern: $P_n(x)$ must be close to $f(x)$ for all values of x in $[a, b]$.
 - We don't care whether $P_n(x)$ has the same value as $f(x)$, or not. I.e., $P_n(x)$ need not reproduce the values of $f(x)$ exactly.

When to use interpolation or approximation?

- Use interpolation if the exact values of $f(x)$ are important. I.e., when you need to reproduce the values of $f(x)$ exactly.
- Use approximation if the overall behaviour of $f(x)$ is important. I.e., you want the error to be small everywhere.

Criteria

If we approximate $f(x)$ by $P_n(x)$, then the error at x is given by

$$e(x) = P_n(x) - f(x).$$

To find a good approximation, we want $e(x)$ to be small at every point in $[a, b]$. It is not enough if $e(x)$ is small only at a few points.

We need to have a way to say $e(x)$ is *small on* $[a, b]$. I.e., we need to invent a “size” or “length” for $e(x)$.

A size of a function is usually called a *norm*.

1. Maximum norm

$$\|f\|_{\infty} = \max_{x \in [a, b]} |f(x)|$$

Also called L_{∞} - norm.

See Figure 28.1 for an example of this maximum norm.

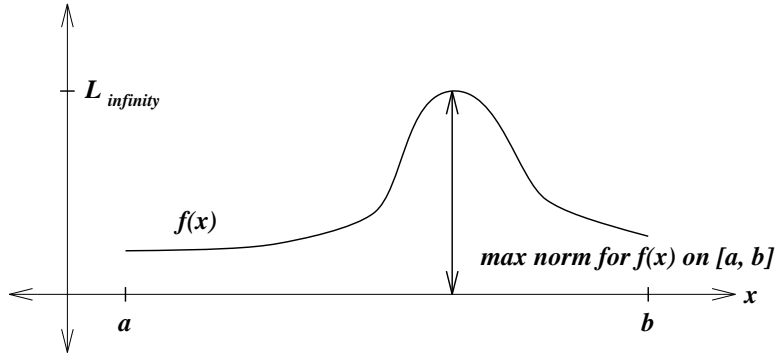


Figure 28.1: An example of this L_{∞} norm.

2. L_2 - norm

$$\|f\|_2 = \sqrt{\int_a^b f(x)^2 dx}$$

Using this method, for each function $f(x)$ on $[a, b]$, we can define a norm size.

Example

$$f(x) = x, \quad [a, b] = [0, 1]$$

$$\|f\|_{\infty} = \max_{x \in [0, 1]} |x| = 1$$

$$\|f\|_2 = \sqrt{\int_0^1 x^2 dx} = \sqrt{\frac{1}{3}}$$

Example

$$f(x) = \sin x, \quad [a, b] = [0, 2\pi]$$

$$\|f\|_{\infty} = \max_{x \in [0, 2\pi]} |\sin x| = 1$$

$$\|f\|_2 = \sqrt{\int_0^{2\pi} \sin^2 x dx} = \sqrt{\pi}$$

Remarks

1. There are many ways to define a norm of a function.
2. If $\|f\|_\infty$ is small, then $f(x)$ is small at every point in $[a, b]$ since $|f(x)| \leq \|f\|_\infty$.
3. If $\|f\|_2$ is small, then $f(x)$ is small on average, but $f(x)$ may be large at some points.

See Figure 28.2 for a picture of a function with small $\|f\|_2$.

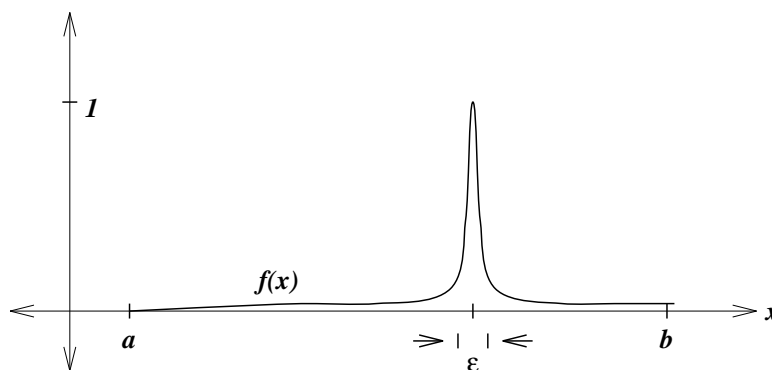


Figure 28.2: Picture of a function with small $\|f\|_2$.

From above, e.g., $\|f\|_2$ is small, but $f(x)$ is 1 at x_0 .

APPROXIMATION PROBLEM

Let's state the problem that we wish to solve. Given $f(x)$ on $[a, b]$, find a polynomial of degree n , $P_n(x)$ which best approximates $f(x)$ in a given norm. I.e., we need to find a polynomial $P_n(x)$ such that

$$\begin{aligned} & \|P_n - f\| = \min. \text{ among all polynomials of degree } n. \\ \text{or } & \|P_n - f\| \leq \|q_n - f\| \text{ for all polynomials } q_n(x) \text{ of degree } n. \\ \text{or } & \|P_n - f\| = \min_{\text{over all } q_n} \|q_n - f\| \text{ for all polynomials } q_n(x) \text{ of degree } n. \end{aligned}$$

Note in the above, the norm is not specified.

If the L_∞ norm is used, then the best approximation $P_n(x)$ is said to be a *minimax* approximation – the polynomial whose maximum error is minimized.

$$\|P_n - f\|_\infty = \min_{q_n} \|q_n - f\|_\infty = \min_{q_n} \max |P_n - f|$$

Fact. If $f(x)$ is continuous on $[a, b]$, then there exists a unique minimax approximation $P_n(x)$ of degree n .

E.g. $n = 1$. Equioscillation property. $f(x)$ cannot be on one side of $P_n(x)$.

See Figure 28.3 for a picture of a linear minimax approximation to a general function.

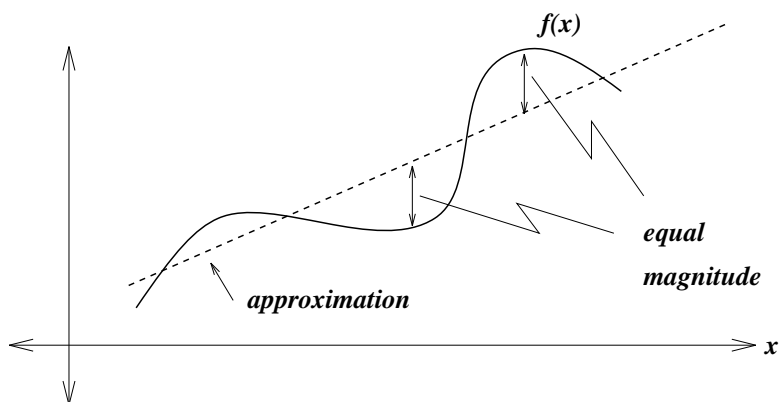


Figure 28.3: Picture of a linear minimax approximation to a general function.

In general, it is difficult to find the minimax polynomial.

So, let's try the L_2 - norm. In this case, we want to find $P_n(x)$ such that

$$\|P_n - f\|_2 = \min.$$

$$\text{i.e., } \sqrt{\int_a^b (P_n(x) - f(x))^2 dx} = \min.$$

Such a $P_n(x)$ is said to be a least squares polynomial, or *least-squares approximation*. This is the continuous version of the least-squares approximation, since we know $f(x)$, which we are trying to approximate, everywhere on x . In the next class, we will look at the discrete least-squares approximation, where we are approximating a given finite set of points.

CISC 271 Class 31

Discrete Least-squares Approximation

Suppose that we were given a set of data points, $\{x_i, Y_i\}, i \in [0, m]$, and we think that they come from a function $f(x)$ where we have a model of $f(x)$, i.e., we know its general form but not know the values of the coefficients, $\{a_0, a_1, \dots, a_n\} = \vec{a}$.

How do we find a good approximation for a_i ? We can address this by trying to minimize residuals,

$$r(x_i) = f(x_i) - Y_i, \quad i \in [0, m]$$

Considering the residuals forming an m -dimensional vector, we can try to minimize any of the following norms

$$\begin{array}{llll} E_1(\vec{a}) & = & L_1(\vec{r}) & = \sum_{i=0}^m |r_i| & \text{(sum of difference)} \\ E_2(\vec{a}) & = & [L_2(\vec{r})]^2 & = \sum_{i=0}^m |r_i|^2 & \text{(squared sum of distance)} \\ E_p(\vec{a}) & = & L_p(\vec{r}) & = (\sum_{i=0}^m |r_i|^p)^{1/p} & \text{(generalization)} \\ E_\infty(\vec{a}) & = & L_\infty(\vec{r}) & = \max_{i=0}^m |r_i| & \text{(largest difference)} \end{array}$$

Note that the norm $L_2()$ is given in its discrete form.

For reasons having to do with subsequent statistical analysis (variance) (discussed in text, G & W, pp. 261-262) we use the L_2 norm. Since minimizing $\sqrt{E_2(\vec{a})}$ is equivalent to minimizing $E_2(\vec{a})$, (any minimum is at the same values of \vec{a} , we minimize $E_2 = \sum r_i^2$, that is, try to find the least squares fit.

Example. Hooke's Law. The form of the relation is

$$f(x) = -kx,$$

and we would like to find \hat{k} from experimental data which is our least-squares fit.

See Figure 29.1 for a graph of example data and approximation setup for Hooke's Law.

The residuals look like $r_i = f_i - Y_i = -kx_i - Y_i$ so $E_2(k)$ is

$$E_2(k) = \sum_{i=0}^m r_i^2 = (-kx_0 - Y_0)^2 + (-kx_1 - Y_1)^2 + \dots + (-kx_m - Y_m)^2$$

This is minimized when

$$\frac{dE_2}{dk}(\hat{k}) = 0.$$

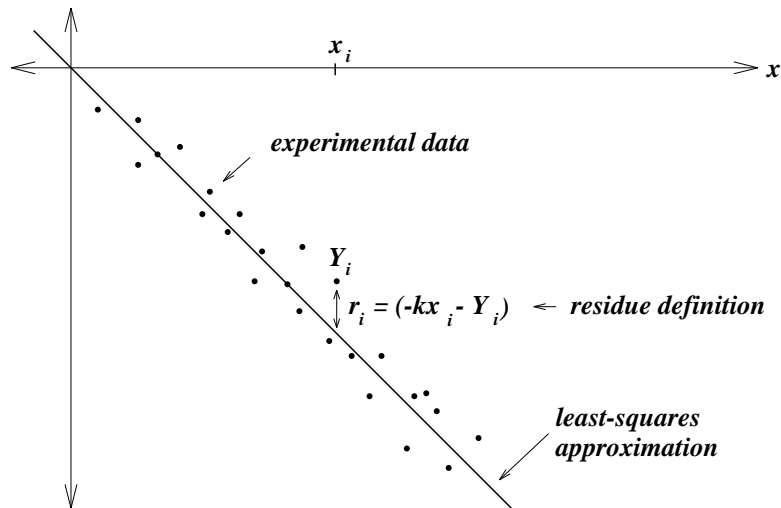


Figure 29.1: Graph of example data and approximation setup.

So, since

$$\begin{aligned}
 \frac{dE_2}{dk} &= \sum_{i=0}^m \frac{d}{dk} (-kx_i - Y_i)^2 \\
 &= \sum_{i=0}^m 2(-kx_i - Y_i)(-x_i) \\
 &= \sum_{i=0}^m (2kx_i^2 + 2Y_i x_i) \\
 &= \sum_{i=0}^m 2kx_i^2 + \sum_{i=0}^m 2Y_i x_i
 \end{aligned}$$

We can use the minimum to find k by

$$\begin{aligned}
 \frac{dE_2}{dk} = 0 &= 2k \sum_{i=0}^m x_i^2 + 2 \sum_{i=0}^m Y_i x_i \\
 \Rightarrow \hat{k} &= \frac{-\sum_{i=0}^m Y_i x_i}{\sum_{i=0}^m x_i^2}
 \end{aligned}$$

Remember that the Y_i 's have opposite sign to their corresponding x_i 's in general, so \hat{k} is a positive value.

LINEAR REGRESSION

Linear regression is a special case of least-squares fitting, where we try to fit a line – not necessarily through the origin – to the data. Thus we have

$$P_1(x) = a_0 + a_1x$$

and we want to find \hat{a}_0 and \hat{a}_1 that minimize $E_2(a_0, a_1)$.

The same principle holds from before, i.e.,

$$\frac{\partial E_2}{\partial a_0}(\hat{a}_0, \hat{a}_1) = \frac{\partial E_2}{\partial a_1}(\hat{a}_0, \hat{a}_1) = 0.$$

$$\begin{aligned}\frac{\partial E_2}{\partial a_0}(\hat{a}_0, \hat{a}_1) &= 2 \sum_{i=0}^m (\hat{a}_0 + \hat{a}_1 x_i - Y_i) \\ \frac{\partial E_2}{\partial a_1}(\hat{a}_0, \hat{a}_1) &= 2 \sum_{i=0}^m (\hat{a}_0 + \hat{a}_1 x_i - Y_i) x_i\end{aligned}$$

So at the zero, noting that $\sum_{i=0}^m (x_i)^0 = m + 1$,

$$\begin{aligned}(m+1)\hat{a}_0 + \left(\sum_{i=0}^m x_i\right)\hat{a}_1 &= \sum_{i=0}^m Y_i \\ \left(\sum_{i=0}^m x_i\right)\hat{a}_0 + \left(\sum_{i=0}^m x_i^2\right)\hat{a}_1 &= \sum_{i=0}^m x_i Y_i\end{aligned}$$

In matrix form, this becomes

$$\begin{pmatrix} (m+1) & \sum_{i=0}^m x_i \\ \sum_{i=0}^m x_i & \sum_{i=0}^m x_i^2 \end{pmatrix} \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^m Y_i \\ \sum_{i=0}^m x_i Y_i \end{pmatrix}$$

which is a linear system, which we know how to solve.

HIGHER DEGREE POLYNOMIALS

We can use this approach for higher-degree polynomials, too. Suppose we want to fit an n -th degree polynomial to the data. Let

$$P_n(x) = \sum_{j=0}^n a_j x^j$$

Still, our given points are $\{(x_i, Y_i)\}, i = 0, \dots, m$.

Thus the norm to be minimized is

$$E_2(\vec{a}) = \sum_{i=0}^m \left[\left(\sum_{j=0}^n (a_j (x_i)^j) - Y_i \right)^2 \right].$$

So that

$$\frac{\partial}{\partial a_k} (E_2(\vec{a})) = 2 \sum_{i=0}^m \left[\left(\sum_{j=0}^n (a_j (x_i)^j) - Y_i \right) \cdot (x_i)^k \right].$$

Therefore,

$$\frac{\partial E_2}{\partial a_k}(\hat{\vec{a}}) = 0$$

yields a set of $n + 1$ linear equations in unknowns $\hat{\vec{a}}$. Each equation (k th shown) has the form

$$\sum_{j=0}^n \left[\hat{a}_j \cdot \sum_{i=0}^m (x_i)^{j+k} \right] = \sum_{i=0}^m Y_i (x_i)^k$$

If we let

$$g_{jk} = \sum_{i=0}^m (x_i)^{j+k} \quad \text{and} \quad \rho_k = \sum_{i=0}^m Y_i (x_i)^k$$

then our system is

$$\begin{pmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0n} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1n} \\ \vdots & & & \ddots & \vdots \\ g_{n0} & g_{n1} & g_{n2} & \cdots & g_{nn} \end{pmatrix} \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_n \end{pmatrix} = \begin{pmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_n \end{pmatrix}$$

which by definition is symmetric (since $g_{ij} = g_{ji}$). The k -th row of the matrix corresponds to the partial derivative with respect to a_k set to zero. Recall that $g_{00} = \sum_{i=0}^m (x_i)^{0+0} = (m + 1)$.

It can be shown that if $m \geq n$, the solution is unique. Thus, we can find any degree of polynomial least-squares approximation that we think is appropriate to the given problem.

Example

Consider the following data

x_i	0.05	0.11	0.15	0.31	0.46	0.52	0.70	0.74	0.82	0.98	1.17
Y_i	0.956	0.890	0.832	0.717	0.571	0.539	0.378	0.370	0.306	0.242	0.104

This data is a perturbation of the relationship $y = 1 - x + 0.2x^2$. Let's fit the data to $P_2(x) = a_0 + a_1x + a_2x^2$. Therefore

$$\begin{aligned}
 g_{10} = g_{01} = \sum x_i &= 6.01 & g_{00} = m + 1 &= 11 \\
 g_{20} = g_{11} = g_{01} = \sum x_i^2 &= 4.6545 & \rho_0 = \sum Y_i &= 5.905 \\
 g_{21} = g_{12} = \sum x_i^3 &= 4.1150 & \rho_1 = \sum x_i Y_i &= 2.1839 \\
 g_{22} = \sum x_i^4 &= 3.9161 & \rho_2 = \sum x_i^2 Y_i &= 1.3357
 \end{aligned}$$

such that

$$\begin{pmatrix} g_{00} & g_{01} & g_{02} \\ g_{10} & g_{11} & g_{12} \\ g_{20} & g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \hat{a}_2 \end{pmatrix} = \begin{pmatrix} \rho_0 \\ \rho_1 \\ \rho_2 \end{pmatrix}$$

which equals

$$\begin{pmatrix} 11 & 6.01 & 4.6545 \\ 6.01 & 4.6545 & 4.1150 \\ 4.6545 & 4.1150 & 3.9161 \end{pmatrix} \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \hat{a}_2 \end{pmatrix} = \begin{pmatrix} 5.905 \\ 2.1839 \\ 1.3357 \end{pmatrix}$$

which has the solution

$$\begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \hat{a}_2 \end{pmatrix} = \begin{pmatrix} 0.998 \\ -1.018 \\ 0.225 \end{pmatrix}$$

such that the polynomial least-squares approximation is

$$P_2(x) = 0.998 - 1.018x + 0.225x^2.$$

CISC 271 Class 32

The Eigenvalue Problem

Eigenvalues and eigenvectors are useful in describing and analyzing many systems, both physical and mathematical. For example, the simple spring equation we looked at previously (to develop discrete least-squares polynomials) was Hooke's Law for a mass hanging from a spring:

$$\begin{aligned} f(x) &= kx \\ \text{or} \quad F &= kx \end{aligned}$$

If we have a mass on a (frictionless) flat surface, and there are two or more springs attached to the mass, then the XY vector of forces is related to the XY vector of position by $K_{2 \times 2}$

$$\vec{F} = \begin{bmatrix} F_x \\ F_y \end{bmatrix} = K_{2 \times 2} \begin{bmatrix} x \\ y \end{bmatrix}$$

where $K_{2 \times 2}$ is symmetric.

Suppose that \vec{v} is an eigenvector of $K_{2 \times 2}$. Then $K\vec{v} = \lambda\vec{v}$, which means that there is a special relationship between the forces and the positions. In fact, what will happen is that if we pull the mass in the direction of a position eigenvector and let go, the mass will oscillate in a *straight line*; if we pull the mass in any direction that is **not** an eigenvector, it will oscillate in an orbital trajectory!

Eigenvectors are also useful in describing linear systems. Suppose that $Z = \{\vec{z}_0, \vec{z}_1, \dots, \vec{z}_n\}$ is a set of linearly independent vectors. This means that if we add scalar multiples of them together to get a vector \vec{x} :

$$\vec{x} = \alpha_0 \vec{z}_0 + \alpha_1 \vec{z}_1 + \dots + \alpha_n \vec{z}_n$$

then $\vec{x} = \vec{0}$ if and only if every $\alpha_i = 0$. Given \vec{x} , we can compute $\alpha_i = \vec{x} \cdot \vec{z}_i$, and α_i is called the component of \vec{x} in the direction \vec{z}_i . We call Z a **basis** for the vector space \mathcal{R}^{n+1} .

This is important because the eigenvectors V of a nonsingular matrix A form a special basis. Using the convention that eigenvectors are of unit length, i.e., we require that $\|\vec{v}_i\| = 1$, we can represent \vec{x} as

$$\begin{aligned}\vec{x} &= \alpha_0 \vec{v}_0 + \alpha_1 \vec{v}_1 + \cdots + \alpha_n \vec{v}_n \\ &= \sum_{i=0}^n a_i \vec{v}_i\end{aligned}$$

When we perform the multiplication $A\vec{x} = \vec{b}$ we get

$$\begin{aligned}A\vec{x} &= a_0 A\vec{v}_0 + a_1 A\vec{v}_1 + \cdots + a_n A\vec{v}_n \\ &= \sum_{i=0}^n a_i \lambda_i \vec{v}_i \\ &= \sum_{i=0}^n \lambda_i (a_i \vec{v}_i)\end{aligned}$$

so each of the original terms $a_i \vec{v}_i$ is multiplied by the eigenvalue λ_i . If λ_i is large, then the term $a_i \vec{v}_i$ greatly increases. If the vector \vec{x} is perturbed to $\hat{\vec{x}}$ by an error \vec{e} of the form

$$\vec{e} = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_i \\ \vdots \\ \epsilon_n \end{bmatrix}$$

so that $\hat{\vec{x}} = \vec{x} + \vec{e}$, then $A\hat{\vec{x}} = \hat{\vec{b}}$ will have a large error due to $\lambda_i(\hat{a}_i \vec{v}_i)$ and the size of λ_i tells us about the stability of A .

There are many mathematically useful decompositions of a matrix A , and many of the decompositions are related to eigenvalues and eigenvectors. One, called the Schur decomposition, states that for any matrix A there is some unitary matrix U such that

$$U^{-1}AU = T$$

where T is an upper triangular matrix and U is a unitary matrix ($U^*U = UU^* = I$ and $\|U\vec{x}\|_2 = \|\vec{x}\|_2$). This is important because the characteristic equation of T is

$$0 = \det(T - \lambda I) = \prod_{i=0}^n (t_{ii} - \lambda_i)$$

so the diagonal entries of T are its eigenvalues.

It is easy to prove that, if $U^{-1}AU = T$, and $A\vec{v}_i = \lambda_i\vec{v}_i$, then $TU\vec{v}_i = \lambda_i U\vec{v}_i$ so if we could find U then we could recover the eigenvalues (and the eigenvectors) of A .

Example: For a symmetric A , how stable is $A\vec{x} = \vec{b}$?

Since $\vec{x} = A^{-1}\vec{b}$, and A^{-1} is symmetric, we have

$$\vec{x} = (UDU^T)\vec{b}$$

for orthogonal U , and diagonal D . So,

$$\begin{aligned}\vec{x} &= UD(U^T\vec{b}) \\ &= UD\vec{C} \\ &= U \begin{bmatrix} \lambda_0 C_0 \\ \lambda_1 C_1 \\ \lambda_2 C_2 \\ \vdots \\ \lambda_n C_n \end{bmatrix}\end{aligned}$$

If $|\lambda_n|$ is huge, then small perturbations in C_n mean large perturbations in \vec{x} .

LOCATING EIGENVALUES

For $A_{n \times n}$, define

$$r_i = \sum_{j=0}^{i-1} |a_{ij}| + \sum_{j=i+1}^n |a_{ij}|$$

Assuming that A is complex, construct a circle in the complex plane centred at a_{ii} and of radius r_i :

$$Z_i = \{z \in C \mid |z - a_{ii}| \leq r_i\}.$$

Then

- a.) There is an eigenvalue $\lambda \in Z_i$;
- b.) If m circles form a connected set S that is disjoint from all other $n - m$ circles, then S contains m eigenvalues (counted by algebraic multiplicity).

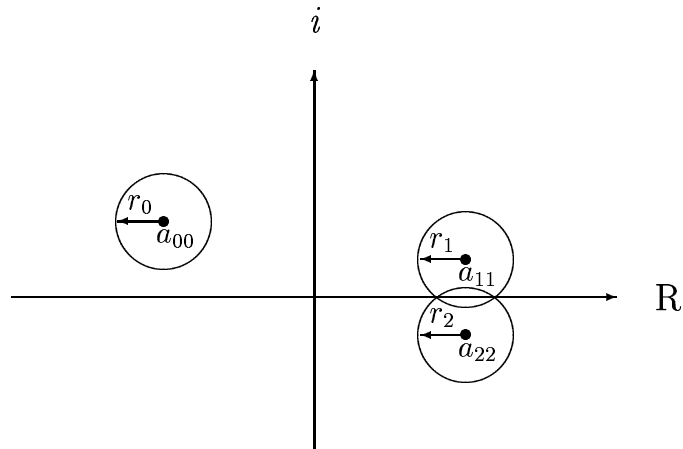


Figure 30.1: Three complex eigenvalues

This situation is shown in Figure 30.1.

Example

For the matrix

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 0 & -1 \\ 1 & 1 & -4 \end{bmatrix}$$

the eigenvalues are in

$$|\lambda - 4| \leq 1, |\lambda| \leq 2, |\lambda + 4| \leq 2$$

as shown in Figure 30.2.

It can be shown that for a real symmetric A $\lambda \in R$ or both λ, λ^* are present.

Therefore, one $\lambda \in [3, 5]$. Since $\lambda = -2$ is not possible from the characteristic equation, then there is one $\lambda \in [-6, -2)$ and one $\lambda \in (-2, 2]$. (Actual answer:s $\lambda \simeq -3.76, -.443, 4.20$).

PERTURBATIONS

Theorem (Bauer-Fike):

Let A be such that $P^{-1}AP = D = \text{diag}(\lambda_0, \dots, \lambda_n)$ and let λ be an eigenvalue of $A + E$. Then

$$\min_{0 \leq i \leq n} |\lambda - \lambda_i| \leq \|P\| \|P^{-1}\| \|E\|.$$

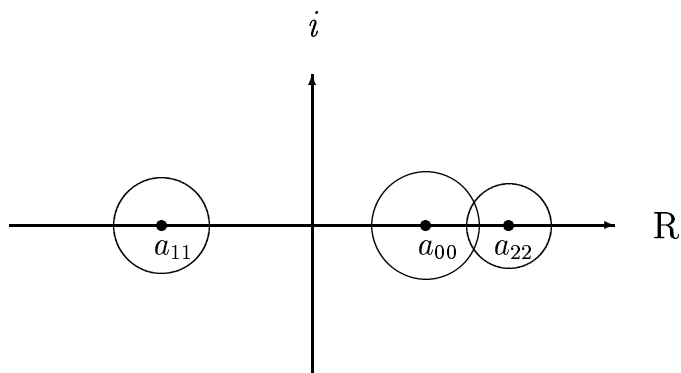


Figure 30.2: Three real eigenvalues

Corollary:

If A is symmetric (Hermitian) then

$$\min_{0 \leq i \leq n} |\lambda - \lambda_i| \leq \|E\|_2.$$

So, perturbations have little effect.

Example

$$H_3 = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix} \quad \begin{array}{l} \lambda_0 = 1.408319 \\ \lambda_1 = 0.1223271 \\ \lambda_2 = 0.002687340 \end{array}$$

$$\hat{H}_3 = \begin{bmatrix} 1.000 & .5000 & .3333 \\ .5000 & .3333 & .2500 \\ .3333 & .2500 & .2000 \end{bmatrix} \quad \begin{array}{l} \lambda_0 = 1.408294 \\ \lambda_1 = 0.1223415 \\ \lambda_2 = 0.002664489 \end{array}$$

Example

$$A = \begin{bmatrix} 101 & -90 \\ 110 & -98 \end{bmatrix} \quad \begin{array}{l} \lambda_0 = 1 \\ \lambda_1 = 2 \end{array}$$

$$A + E = \begin{bmatrix} 100.999 & -90.001 \\ 110 & -98 \end{bmatrix} \quad \begin{array}{l} \lambda_0 = 1.298 \\ \lambda_1 = 1.701 \end{array}$$

CISC 271 Class 33

The Power Method

Suppose that the eigenvalues of $A_{(n+1) \times (n+1)}$ are ordered

$$|\lambda_0| > |\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$$

with one eigenvalue, λ_0 , dominant.

If A has n distinct eigenvalues, then any vector \vec{z} can be written as

$$\vec{z} = \sum_{j=0}^n \alpha_j \vec{v}_j \quad \text{where } \alpha_j = \vec{z} \cdot \vec{v}_j$$

Let us examine the iteration $\vec{z}(i) = A\vec{z}(i-1)$, where $\alpha_0 \neq 0$:

$$\vec{z}(1) = A\vec{z}(0) = \sum_{j=0}^n \alpha_j A\vec{v}_j = \sum_{j=0}^n \alpha_j \lambda_j \vec{v}_j$$

$$\vec{z}(2) = \sum_{j=0}^n \alpha_j (\lambda_j)^2 \vec{v}_j$$

and so

$$\vec{z}(k) = \sum_{j=0}^n \alpha_j (\lambda_j)^k \vec{v}_j$$

Multiply the RHS by $\left(\frac{\lambda_0}{\lambda_0}\right)^k$, to find that

$$\vec{z}(k) = (\lambda_0)^k \sum_{j=0}^n \alpha_j \left(\frac{\lambda_j}{\lambda_0}\right)^k \vec{v}_j$$

Because $\lambda_j/\lambda_0 < 1$, $\lim_{k \rightarrow \infty} \left(\frac{\lambda_j}{\lambda_0}\right)^k = 0$ and so for large k ,

$$\vec{z}(k) \simeq (\lambda_0)^k \alpha_0 \vec{v}_0$$

since $\frac{\lambda_j}{\lambda_0} = 1$ for $j = 0$.

This suggests that it is possible to find the dominant eigenvalue/eigenvector pair by repeated application of A :

$$A^k \vec{z}(0) \simeq (\lambda_0)^k \alpha_0 \vec{v}_0$$

Notes:

1. This is valid iff $\vec{z}(i) \cdot \vec{v}_0 \neq 0$ Usually, round-off error *helps* to produce a non-zero component after a few iterations.
2. To avoid overflow/underflow, $\vec{z}(i)$ should be normalized. The easiest norm is L_∞ .
3. A good termination condition is that $\vec{z}(i)$ and $\vec{z}(i - 1)$ are almost scalar multiples.

ALGORITHM FOR THE POWER METHOD

```

Select a non-zero  $z[0..n](0)$  with  $||z[0..n](0)||_{\infty} = 1$ ;
Set  $\mu[-1] = 0$ ;  $\mu[0] = 1$ ;
 $i = 0$ ;
while  $|\mu[i] - \mu[i-1]|$  greater than or equal to epsilon
     $i = i + 1$ ;
     $Y[0..n] = A * z[0..n](i-1)$ ;
     $\mu[i] = ||Y[0..n]||_{\infty}$ ;
     $z[0..n](i) = Y[0..n] / \mu[i]$ ;
end while;
result:  $\mu[k] \approx \lambda[0]$ ;  $z[0..n](k) \approx v[0..n][0]$ ;

```

The largest eigenvalue is associated with the eigenvector that dominates the matrix. How do we find the *smallest* one?

Because the eigenvalues of A^{-1} are $\frac{1}{\lambda_i}$, we can use the inverse power method:

$$\vec{x}(i) = A^{-1}\vec{x}(i - 1)$$

Instead of inverting A , we can observe that

$$A\vec{x}(i) = \vec{x}(i - 1)$$

is an equivalent computation. This can be done efficiently by first computing $A = LU$, and just back-substituting:

$$LU\vec{x}(i) = \vec{x}(i - 1)$$

DEFLATION

Let the dominant eigenvalue/eigenvector of A be λ_0 and \vec{v}_0 . Pick \vec{Y} so that $\vec{y} \cdot \vec{v}_0 = 1$ (for example, \vec{y} is all zero except for 1 element).

It can be show that the matrix

$$B = A - \lambda_0 \vec{v}_0 \vec{y}^T$$

has eigenvalues $0, \lambda_1, \lambda_2, \dots, \lambda_n$ and eigenvectors $\vec{v}_0, \vec{w}_1, \vec{w}_2, \dots, \vec{w}_n$ where

$$\vec{v}_i = (\lambda_i - \lambda_0) \vec{w}_i + \lambda_0 (\vec{Y} \cdot \vec{w}_i) \vec{v}_i$$

for $i = 1, 2, \dots, n$.

CISC 271 Class 34

QR and SVD

A powerful theoretical result, for which there is a practical algorithm, is called the QR decomposition:

For any square matrix A , there is an orthogonal Q and an upper-triangular (right) R such that

$$A = QR$$

Calculation is complicated, but there are many good codes available.

The most common method for finding all the eigenvalues is the *QR iteration*. Observe that if $A(0) = A$, then we can always decompose

$$A(i) = Q(i)R(i)$$

Form

$$A(i+1) = R(i)Q(i)$$

Such that

$$\begin{aligned} A(i+1) &= Q^T(i)Q(i)R(i)Q(i) \\ &= Q^T(i)A(i)Q(i) \end{aligned}$$

so $A(i+1)$ is similar to $A(i)$ and has the same eigenvalues.

Eventually, $A(i)$ converges to

- a.) a diagonal matrix, *or*
- b.) a matrix that is “nearly” diagonal but which has easily calculated eigenvalues.

For case a.), $A(i) \rightarrow D$ and

$$\|D - A(i)\| \leq c \cdot \max_{j=0}^n \left| \frac{\lambda_{j+1}}{\lambda_j} \right|$$

Note: Inverse iteration (1944) is still best for extracting eigenvectors.

SINGULAR VALUE DECOMPOSITION

One of the most useful theorems of 20th - century linear algebra is the singular-value decomposition.

For any $M_{m \times n}$, there is a unitary $U_{n \times n}$ and $V_{m \times m}$, and a “diagonal” $\Sigma_{m \times n}$ such that

$$A = V\Sigma U^T$$

and

$$\Sigma = \begin{bmatrix} \sigma_0 & & \cdots & & 0 \\ 0 & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \\ \vdots & & & & & 0 \\ 0 & & \cdots & & & 0 \end{bmatrix}$$

where

- a.) $\sigma_i \in R$
- b.) $\sigma_i > 0$
- c.) $\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_r > 0$ and
- d.) r is the rank of A .

Corollary:

If A is real and square ($n \times n$), then $\{\sigma_i^2\}$ are the eigenvalues of $A^T A$.

Proof: $A^T A = U\Sigma^T V^T V\Sigma U^T = U\Sigma^2 U^T = U D U^T$

There are many uses for the Singular Value Decomposition. One depends on its unusual numerical stability:

$$A^{-1} = (V\Sigma U^T)^{-1} = (U^T)^{-1} \Sigma^{-1} (U)^{-1} = U \Sigma^{-1} V^T$$

where

$$\Sigma^{-1} = \text{diag}\left(\frac{1}{\sigma_0}, \frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_n}\right)$$

So to solve $A\vec{x} = \vec{b}$, we compute

$$\vec{x} = U\Sigma^{-1}V^T\vec{b}$$

LEAST SQUARES DATA-FITTING

In general, given m values $\{(x_i, Y_i)\}$ that presumably come from some function – $Y_i = g(x_i)$, we may want to find the weights w_i of n arbitrary $\phi_j(x)$ so that

$$\sum_{j=0}^n w_j \phi_j(x_i) \simeq Y_i$$

For example, linear least squares: $\phi_0(x) = 1, \phi_1(x) = x$, find $w_0 + w_1 x_i \simeq Y_i$.

This is usually done by minimizing the error due to the weights:

$$\begin{aligned} \min_{\vec{w}} E(\vec{w}) &= \left(\frac{1}{m} \sum_{i=0}^m \left(Y_i - \sum_{j=0}^n w_j \phi_j(x_i) \right)^2 \right)^{\frac{1}{2}} \\ &= \frac{1}{\sqrt{m}} \|\vec{Y} - A\vec{w}\|_2 \end{aligned}$$

where $a_{ij} = \phi_j(x_i)$.

Setting the partial derivatives to zero, we seek a solution to

$$A^T A \vec{w} = A^T \vec{Y}$$

SVD approach

Using the singular value decomposition, and substituting $A = V\Sigma U^T$ into $A^T A \vec{w} = A^T \vec{Y}$,

$$\begin{aligned} U\Sigma^T \Sigma U^T \vec{w} &= U\Sigma^T V^T \vec{Y} \\ \Sigma^T \Sigma U^T \vec{w} &= \Sigma^T V^T \vec{Y} \\ \Sigma U^T \vec{w} &= V^T \vec{Y} \\ U^T \vec{w} &= \Sigma^{-1} V^T \vec{Y} \\ \vec{w} &= U\Sigma^{-1} V^T \vec{Y} \end{aligned}$$

where the first line was premultiplied by U^{-1} and the second line was premultiplied by Σ^{-1} . The solution has a stability of

$$\text{cond}(A^T A)_2 = \frac{\sigma_0^2}{\sigma_n^2}$$

QR Solution

The QR method for solution is as follows. Factor $A = QR$ into the normal equations:

$$\begin{aligned}\sqrt{m}E(\vec{w}) &= \|A\vec{w} - \vec{Y}\|_2 \\ &= \|Q^T A\vec{w} - Q^T \vec{Y}\|_2 \quad \text{because } Q \text{ is orthogonal} \\ &= \|Q^T QR\vec{w} - Q^T \vec{Y}\|_2 \\ &= \|R\vec{w} - Q^T \vec{Y}\|_2\end{aligned}$$

Because A is $m \times n$, the specific for R is

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \longrightarrow \begin{array}{ll} n \times n & R_1 \text{ nonsingular} \\ (m-n) \times n & \end{array}$$

Rewrite

$$Q^T \vec{Y} = \begin{bmatrix} \vec{z}_1 \\ \vec{z}_2 \end{bmatrix}$$

where \vec{z}_1 is a vector of length n , and \vec{z}_2 is a vector of length $m - n$.

Then

$$\begin{aligned}\sqrt{m}E(\vec{w}) &= \|A\vec{w} - \vec{Y}\|_2 \\ &= \|R\vec{w} - \vec{z}\|_2 \\ &= \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \vec{w} - \begin{bmatrix} \vec{z}_1 \\ \vec{z}_2 \end{bmatrix} \right\|_2 \\ &= \left(\|R_1 \vec{w} - \vec{z}_1\|_2^2 + \|\vec{z}_2\|_2^2 \right)^{\frac{1}{2}}\end{aligned}$$

\vec{z}_2 is constant, so this is minimized by \vec{w} that satisfy

$$R_1 \vec{w} = \vec{z}_1$$

This is the preferred way of solving least-square problems.