التحليل العددي

أستاذ المادة :- الأستاذ المساعد ناجي مطر سحيب

Chapter 1

Computer arithmetic

1.1 Introduction

What are numeric methods? They are algorithms that compute approximations to solutions of equations or similar things.

Such algorithms should be implemented (programmed) on a computer.

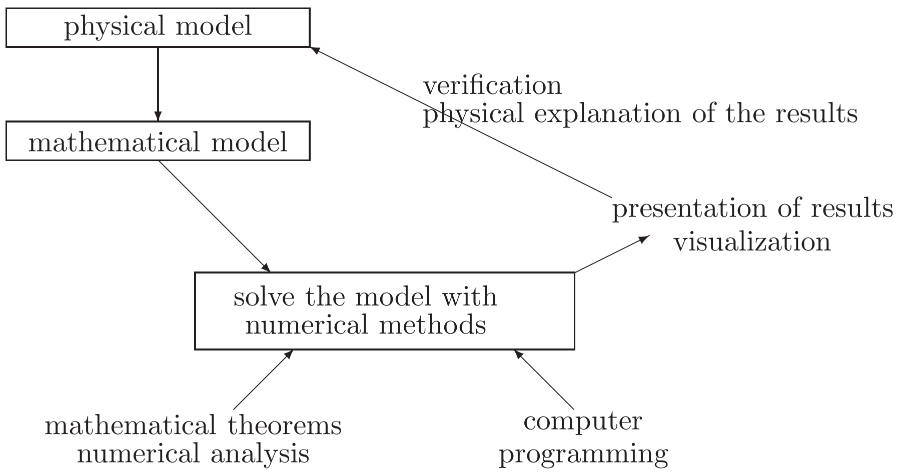

Figure 1.1: The big picture

Numerical methods are not about numbers. It is about mathematical insights.

We will study some basic classical types of problems:

- development of algorithms;
- implementation;

- a little bit of analysis, including error-estimates, convergence, stability etc.

We will use Matlab throughout the course for programming purpose.

1.2 Representation of numbers in different bases

Some bases for numbers:

10: decimal, daily use;

2: binary, computer use;

8: octal;

18: hexadecimal, ancient China;

20: ancient France;

- etc...

In principle, one can use any number β as the base.

$$\begin{aligned} & \Big(\overbrace{a_n a_{n-1} \cdots a_1 a_0}^{\text{integer part}} . \overbrace{b_1 b_2 b_3 \cdots}^{\text{fractional part}}\Big)_\beta \\ = \quad & a_n\beta^n + a_{n-1}\beta^{n-1} + \cdots + a_1\beta + a_0 \qquad \text{(integer part)} \\ & +b_1\beta^{-1} + b_2\beta^{-2} + b_3\beta^{-3} + \cdots \qquad \text{(fractonal part)} \end{aligned}$$

Converting between different bases:

Example 1. octal $\rightarrow$ decimal

$$(45.12)_8 = 4 \times 8^2 + 5 \times 8 + 1 \times 8^{-1} + 2 \times 8^{-2} = (37.15625)_{10}$$

Example 2. octal $\rightarrow$ binary

Observe

$$\begin{aligned}
(1)_8 &= (1)_2 \\
(2)_8 &= (10)_2 \\
(3)_8 &= (11)_2 \\
(4)_8 &= (100)_2 \\
(5)_8 &= (101)_2 \\
(6)_8 &= (110)_2 \\
(7)_8 &= (111)_2 \\
(8)_8 &= (1000)_2
\end{aligned}$$

Then,

$$(5034)_8 = (\underbrace{101}_{5}\,\underbrace{000}_{0}\,\underbrace{011}_{3}\,\underbrace{100}_{4})_2$$

and

$$(110\,010\,111\,001)_2 = (\underbrace{6}_{110}\quad\underbrace{2}_{010}\quad\underbrace{7}_{111}\quad\underbrace{1}_{001})_8$$

Example 3. decimal → binary: write $(12.45)_{10}$ in binary base.

Answer. integer part

$$\begin{array}{r|ll}
2 & \underline{12} & 0 \\
2 & \underline{6} & 0 \\
2 & \underline{3} & 1 \\
2 & \underline{1} & 1
\end{array} \quad \Rightarrow (12)_{10} = (1100)_2$$

fractional part

$$\begin{array}{l|ll}
0.45 & \times & 2 \\
\underline{0}.9 & \times & 2 \\
\underline{1}.8 & \times & 2 \\
\underline{1}.6 & \times & 2 \\
\underline{1}.2 & \times & 2 \\
\underline{0}.4 & \times & 2 \\
\underline{0}.8 & \times & 2 \\
\underline{1}.6 & \times & 2 \\
\cdots & &
\end{array} \quad \Rightarrow (0.45)_{10} = (0.01110011001100\cdots)_2.$$

Put together:

$$(10.45)_{10} = (1100.01110011001100\cdots)_2$$

1.3 Floating point representation

normalized scientific notation

decimal: $x = \pm r \times 10^n, \quad 10^{-1} \le r < 1$. (ex: $r = 0.d_1 d_2 d_3 \cdots, \quad d_1 \ne 0$)

binary: $x = \pm r \times 10^n, \quad 2^{-1} \le r < 1$

octal: $x = \pm r \times 8^n, \quad 8^{-1} \le r < 1$

r: normalized mantissa

n: exponent

Computers represent numbers with finite length. These are called *machine numbers*.

In a 32-bit computer, with single-precision:

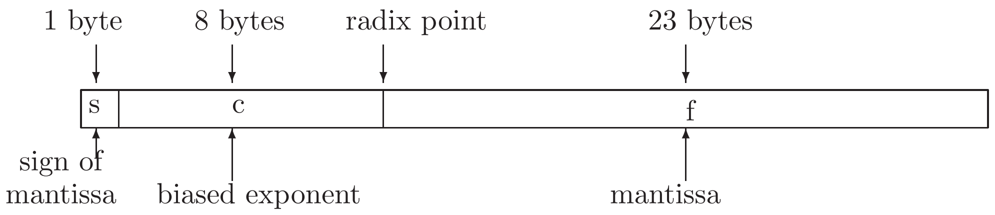

Figure 1.2: 32-bit computer with single precision

The exponent: $2^8 = 256$. It can represent numbers from -127 to 128.

The value of the number:

$$(-1)^s \times 2^{c-127} \times (1.f)_2$$

This is called: single-precision IEEE standard floating-point.

smallest representable number: $x_{\min} = 2^{-127} \approx 5.9 \times 10^{-39}$.

largest representable number: $x_{\max} = 2^{128} \approx 2.4 \times 10^{38}$.

We say that x underflows if $x < x_{\min}$, and consider $x = 0$.

We say that x overflows if $x > x_{\max}$, and consider $x = \infty$.

Computer errors in representing numbers:

- round off relative error: $\le 0.5 \times 2^{-23} \approx 0.6 \times 10^{-7}$
- chopping relative error: $\le 1^{-23} \approx 1.2 \times 10^{-7}$

Floating point representation of a number x: call it $\mathrm{fl}(x)$

$$\mathrm{fl}(x) = x \cdot (1 + \delta)$$

$$\text{relative error: } = \frac{\mathrm{fl}(x) - x}{x} = \delta$$

$$\text{absolute error: } = \text{fl}(x) - x = \delta \cdot x$$

$|\delta| \le \varepsilon$, where ε is called *machine epsilon*, which represents the smallest positive number detectable by the computer, such that $\text{fl}(1+\varepsilon) > 1$.

In a 32-bit computer: $\varepsilon = 2^{-23}$.

Error propagation (through arithmetic operation)

Example 1. Addition, $z = x + y$.

Let

$$\text{fl}(x) = x(1+\delta_x), \qquad \text{fl}(y) = y(1+\delta_y)$$

Then

$$\begin{aligned}
\text{fl}(z) &= \text{fl}\left(\text{fl}(x) + \text{fl}(y)\right) \\
&= \left(x(1+\delta_x) + y(1+\delta_y)\right)(1+\delta_z) \\
&= (x+y) + x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z) + (x\delta_x\delta_z + y\delta_y\delta_z) \\
&\approx (x+y) + x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z)
\end{aligned}$$

$$\begin{aligned}
\text{absolute error} &= \text{fl}(z) - (x+y) = x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z) \\
&= \underbrace{\underbrace{x\cdot\delta_x}_{\substack{\text{abs. err.}\\ \text{for } x}} + \underbrace{y\cdot\delta_y}_{\substack{\text{abs. err.}\\ \text{for } y}}}_{\text{propagated error}} + \underbrace{(x+y)\cdot\delta_z}_{\text{round off err}}
\end{aligned}$$

$$\text{relative error} = \frac{\text{fl}(z)-(x+y)}{x+y} = \underbrace{\frac{x\delta_x + y\delta_y}{x+y}}_{\text{propagated err}} + \underbrace{\delta_z}_{\text{round off err}}$$

1.4 Loss of significance

This typically happens when one gets too few significant digits in subtraction.

For example, in a 8-digit number:

$$x = 0.d_1d_2d_3\cdots d_8 \times 10^{-a}$$

d_1 is the most significant digit, and d_8 is the least significant digit.

Let $y = 0.b_1b_2b_3\cdots b_8 \times 10^{-a}$. We want to compute $x - y$.

If $b_1 = d_1$, $b_2 = d_2$, $b_3 = d_3$, then

$$x - y = 0.000c_4c_5c_6c_7c_8 \times 10^{-a}$$

We lose 3 significant digits.

Example 1. Find the roots of $x^2 - 40x + 2 = 0$. Use 4 significant digits in the computation.

Answer. The roots for the equation $ax^2 + bx + c = 0$ are

$$r_{1,2} = \frac{1}{2a}\left(-b \pm \sqrt{b^2 - 4ac}\right)$$

In our case, we have

$$x_{1,2} = 20 \pm \sqrt{398} \approx 20 \pm 19.95$$

so

$$x_1 \approx 20 + 19.95 = 39.95, \quad \text{(OK)}$$

$$x_2 \approx 20 - 19.95 = 0.05, \quad \text{not OK, lost 3 sig. digits}$$

To avoid this: change the algorithm. Observe that $x_1x_2 = c/a$. Then

$$x_2 = \frac{c}{ax_1} = \frac{2}{1 \cdot 39.95} \approx 0.05006$$

We get back 4 significant digits in the result.

1.5 Review of Taylor Series

Given $f(x)$, smooth function. Expand it at point $x = c$:

$$f(x) = f(c) + f'(c)(x-c) + \frac{1}{2!}f''(c)(x-c)^2 + \frac{1}{3!}f'''(c)(x-c)^3 + \cdots$$

or using the summation sign

$$f(x) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(c)(x-c)^k.$$

This is called *Taylor series of f at the point c*.

Special case, when $c = 0$, is called *Maclaurin series*:

$$f(x) = f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(0)x^3 + \cdots = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(0)x^k.$$

Some familiar examples

$$\begin{aligned}
e^x &= \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad |x| < \infty \\
\sin x &= \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \qquad |x| < \infty \\
\cos x &= \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \qquad |x| < \infty \\
\frac{1}{1-x} &= \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + x^3 + x^4 + \cdots, \qquad |x| < 1 \\
\text{etc.}
\end{aligned}$$

This is actually how computers calculate many functinos!
For example:

$$e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}$$

for some large integer N such that the error is sufficiently small.

Example 1. Compute e to 6 digit accuracy.

Answer. We have

$$e = e^1 = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots$$

And

$$\begin{aligned}
\frac{1}{2!} &= 0.5 \\
\frac{1}{3!} &= 0.166667 \\
\frac{1}{4!} &= 0.041667 \\
\cdots \\
\frac{1}{9!} &= 0.0000027 \qquad \text{(can stop here)}
\end{aligned}$$

so

$$e \approx 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots \frac{1}{9!} = 2.71828$$

Error and convergence: Assume $f^{(k)}(x)$ $(0 \le k \le n)$ are continuous functions. Call

$$f_n(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(c)(x-c)^k$$

the first $n+1$ terms in Taylor series.

Then, the error is

$$E_{n+1} = f(x) - f_n(x) = \sum_{k=n+1}^{\infty} \frac{1}{k!} f^{(k)}(c)(x-c)^k = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x-c)^{n+1}$$

where ξ is some point between x and c.

Observation: A Taylor series convergence rapidly if x is near c, and slowly (or not at all) if x is far away from c.

Special case: $n = 0$, we have the "Mean-Value Theorem":

If f is smooth on the interval (a, b), then

$$f(a) - f(b) = (b-a)f'(\xi), \qquad \text{for some } \xi \text{ in } (a, b).$$

See Figure 1.3.

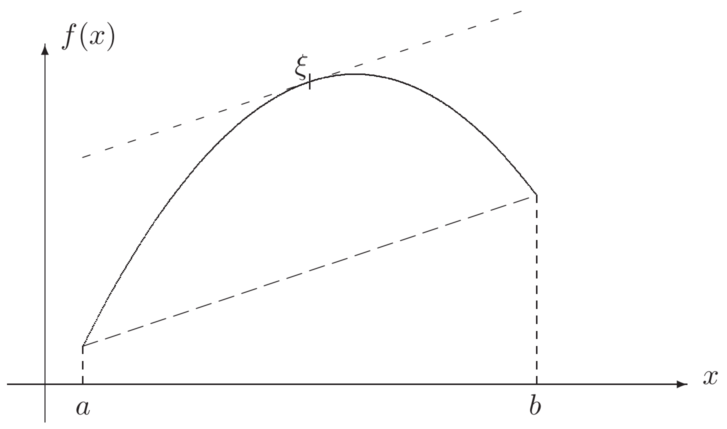

Figure 1.3: Mean Value Theorem

This implies

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

So, if a, b are close to each other, this can be used as an approximation for f'.

Given $h > 0$ very small, we have

$$\begin{aligned} f'(x) &\approx \frac{f(x+h) - f(x)}{h} \\ f'(x) &\approx \frac{f(x) - f(x-h)}{h} \\ f'(x) &\approx \frac{f(x+h) - f(x-h)}{2h} \end{aligned}$$

Another way of writing Taylor Series:

$$f(x+h) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(x) h^k = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x) h^k + E_{n+1}$$

where

$$E_{n+1} = \sum_{k=n+1}^{\infty} \frac{1}{k!} f^{(k)}(x) h^k = \frac{1}{(n+1)!} f^{(n+1)}(\xi) h^{n+1}$$

for some ξ that lies between x and $x+h$.

Chapter 2

Polynomial interpolation

2.1 Introduction

Problem description:

Given $(n+1)$ points, (x_i, y_i), $i = 0, 1, 2, \cdots, n$, with distinct x_i such that

$$x_0 < x_1 < x_2 < \cdots < x_n,$$

find a polynomial of degree n

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

such that it interpolates these points:

$$P_n(x_i) = y_i, \qquad i = 0, 1, 2, \cdots, n$$

Why should we do this?

- Find the values between the points;
- To approximate a (probably complicated) function by a polynomial

Example 1. Given table

x_i	0	1	2/3
y_i	1	0	0.5

Note that

$$y_i = \cos(\pi/2) x_i$$

Interpolate with a polynomial with degree 2.

Answer. Let

$$P_2(x) = a_0 + a_1 x + a_2 x^2$$

Then

$$\begin{aligned} x=0,\ y=1 &\ :\ P_2(0)=a_0=1 \\ x=1,\ y=0 &\ :\ P_2(1)=a_0+a_1+a_2=0 \\ x=2/3,\ y=0.5 &\ :\ P_2(2/3)=a_0+(2/3)a_1+(4/9)a_2=0.5 \end{aligned}$$

In matrix-vector form

$$\begin{pmatrix}1 & 0 & 0\\ 1 & 1 & 1\\ 1 & \frac{2}{3} & \frac{4}{9}\end{pmatrix}\begin{pmatrix}a_0\\ a_1\\ a_2\end{pmatrix}=\begin{pmatrix}1\\ 0\\ 0.5\end{pmatrix}$$

Easy to solve in Matlab (homework 1)

$$a_0=1, \quad a_1=-1/4, \quad a_2=-3/4.$$

Then

$$P_2(x)=1-\frac{1}{4}x-\frac{3}{4}x^2.$$

Back to general case with $(n+1)$ points:

$$P_n(x_i)=y_i, \qquad i=0,1,2,\cdots,n$$

We will have $(n+1)$ equations:

$$\begin{aligned} P_n(x_0)=y_0 &\ :\ a_0+x_0a_1+x_0^2a_2+\cdots+x_0^na_n &= y_0 \\ P_n(x_1)=y_1 &\ :\ a_0+x_1a_1+x_1^2a_2+\cdots+x_1^na_n &= y_1 \\ \cdots & & \\ P_n(x_n)=y_n &\ :\ a_0+x_na_1+x_n^2a_2+\cdots+x_n^na_n &= y_n \end{aligned}$$

In matrix-vector form

$$\begin{pmatrix}1 & x_0 & x_0^2 & \cdots & x_0^n\\ 1 & x_1 & x_1^2 & \cdots & x_1^n\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & x_n & x_n^2 & \cdots & x_n^n\end{pmatrix}\begin{pmatrix}a_0\\ a_1\\ \vdots\\ a_n\end{pmatrix}=\begin{pmatrix}y_0\\ y_1\\ \vdots\\ y_n\end{pmatrix}$$

or

$$\mathbf{X}\,\vec{a}=\vec{y}$$

where

x	:	$(n+1)\times(n+1)$ matrix, given, (van der Monde matrix)
$\vec{a}$	:	unknown vector, $(n+1)$
$\vec{y}$	:	given vector, $(n+1)$

Known: if x_i's are distinct, then $\mathbf{X}$ is invertible, therefore $\vec{a}$ has a unique solution.
In Matlab, the command `vander`$([x_1, x_2, \cdots, x_n])$ gives this matrix.
But: $\mathbf{X}$ has very large condition number, not effective to solve.
Other more efficient and elegant methods

- Lagrange polynmial
- Newton's divided differences

2.2 Lagrange interpolation

Given points: $x_0, x_1, \cdots, x_n$
Define the *cardinal functions*: $l_0, l_1, \cdots, l_n :\in \mathcal{P}^n$ (polynomials of degree n)

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j \end{cases} \qquad i = 0, 1, \cdots, n$$

The **Lagrange form of the interpolation polynomial** is

$$P_n(x) = \sum_{i=0}^{n} l_i(x) \cdot y_i.$$

We check the interpolating property:

$$P_n(x_j) = \sum_{i=0}^{n} l_i(x_j) \cdot y_i = y_j, \qquad \forall j.$$

$l_i(x)$ can be written as

$$\begin{aligned} l_i(x) &= \prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \\ &= \frac{x - x_0}{x_i - x_0} \cdot \frac{x - x_1}{x_i - x_1} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n} \end{aligned}$$

One can easily check that $l_i(x_i) = 1$ and $l_i(x_j) = 0$ for $i \neq j$, i.e., $l_i(x_j) = \delta_{ij}$.

Example 2. Consider again (same as in Example 1)

x_i	0	1	2/3
y_i	1	0	0.5

Write the Lagrange polynomial.

Answer. We have

$$\begin{aligned}
l_0(x) &= \frac{x-2/3}{0-2/3}\cdot\frac{x-1}{0-1} = \frac{3}{2}(x-\frac{2}{3})(x-1)\\
l_1(x) &= \frac{x-0}{2/3-0}\cdot\frac{x-1}{2/3-1} = -\frac{9}{2}x(x-1)\\
l_2(x) &= \frac{x-0}{1-0}\cdot\frac{x-2/3}{1-2/3} = 3x(x-\frac{2}{3})
\end{aligned}$$

so

$$\begin{aligned}
P_2(x) &= l_0(x)y_0 + l_1(x)y_1 + l_2(x)y_2\\
&= \frac{3}{2}(x-\frac{2}{3})(x-1) - \frac{9}{2}x(x-1)(0.5) + 0\\
&= -\frac{3}{4}x^2 - \frac{1}{4}x + 1
\end{aligned}$$

This is the same as in Example 1.

Pros and cons of Lagrange polynomial:

- Elegant formula, (+)
- slow to compute, each $l_i(x)$ is different, (-)
- Not flexible: if one changes a points x_j, or add on an additional point x_{n+1}, one must re-compute all l_i's. (-)

2.3 Newton's divided differences

Given a data set

x_i	x_0	x_1	$\cdots$	x_n
y_i	y_0	y_1	$\cdots$	y_n

$n=0$: $P_0(x) = y_0$

$n=1$: $P_1(x) = P_0(x) + a_1(x-x_0)$
Determine a_1: set in $x = x_1$, then $P_1(x_1) = P_0(x_1) + a_1(x_1 - x_0)$
so $y_1 = y_0 + a_1(x_1-x_0)$, we get $a_1 = \dfrac{y_1-y_0}{x_1-x_0}$

$n=2$: $P_2(x) = P_1(x) + a_2(x-x_0)(x-x_1)$
set in $x = x_2$: then $y_2 = P_1(x_2) + a_2(x_2-x_0)(x_2-x_1)$
so $a_2 = \dfrac{y_2 - P_1(x_2)}{(x_2-x_0)(x_2-x_1)}$

General expression for a_n:

Assume that $P_{n-1}(x)$ interpolates (x_i, y_i) for $i = 0, 1, \cdots, n-1$. We will find $P_n(x)$ that interpolates (x_i, y_i) for $i = 0, 1, \cdots, n$, in the form

$$P_n(x) = P_{n-1}(x) + a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

where

$$a_n = \frac{y_n - P_{n-1}(x_n)}{(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1})}$$

Check by yourself that such polynomial does the interpolating job!

Newtons' form:

$$\begin{aligned} p_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots \\ &\quad + a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}) \end{aligned}$$

The constants a_i's are called *divided difference*, written as

$$a_0 = f[x_0], \quad a_1 = f[x_0, x_1] \quad \cdots a_i = f[x_0, x_1, \cdots, x_i]$$

And we have (see textbook for proof)

$$f[x_0, x_1, \cdots, x_k] = \frac{f[x_1, x_1, \cdots, x_k] - f[x_0, x_1, \cdots, x_{k-1}]}{x_k - x_0}$$

Compute f's through the table:

x_0	$f[x_0] = y_0$				
x_1	$f[x_1] = y_1$	$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$			
x_2	$f[x_2] = y_2$	$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$	$f[x_0, x_1, x_2] = \cdots$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
x_n	$f[x_n] = y_n$	$f[x_{n-1}, x_n] = \frac{f[x_n] - f[x_{n-1}]}{x_n - x_{n-1}}$	$f[x_{n-2}, x_{n-1}, x_n] = \cdots$	$\ldots$	$f[x_0, x_1, \cdots, x_n]$

Example : Use Newton's divided difference to write the polynomial that interpolates the data

x_i	0	1	2/3	1/3
y_i	1	0	1/2	0.866

Answer. Set up the trianglar table for computation

0	$\boxed{1}$			
1	0	$\boxed{-1}$		
2/3	0.5	-1.5	$\boxed{-0.75}$	
1/3	0.8660	-1.0981	-0.6029	$\boxed{0.4413}$

So

$$P_3(x) = \boxed{1} + \boxed{-1}x + \boxed{-0.75}x(x-1) + \boxed{0.4413}x(x-1)(x-2/3).$$

Flexibility of Newton's form: easy to add additional points to interpolate.

Nested form:

$$\begin{aligned} P_n(x) &= a_0 + a_1(x-x_0) + +a_2(x-x_0)(x-x_1) + \cdots \\ &\qquad + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}) \\ &= a_0 + (x-x_0)\left(a_1 + (x-x_1)(a_2 + (x-x_2)(a_3 + \cdots + a_n(x-x_{n-1})))\right) \end{aligned}$$

Effective to compute in a program:

```
• $p = a_n$
• for $k = n-1, n-1, \cdots, 0$
    – $p = p(x - x_k) + a_k$
• end
```

Some theoretical parts:

Existence and Uniqueness theorem for polynomial interpolation:

Given $(x_i, y_i)_{i=0}^n$, with x_i's distinct. There exists one and only polynomial $P_n(x)$ of degree $\leq n$ such that

$$P_n(x_i) = y_i, \qquad i = 0, 1, \cdots, n$$

Proof. : Existence: OK from construction

Uniqueness: Assume we have two polynomials, call them $p(x)$ and $q(x)$, of degree $\leq n$, both interpolate the data, i.e.,

$$p(x_i) = y_i, \qquad q(x_i) = y_i, \qquad i = 0, 1, \cdots, n$$

Now, let $g(x) = p(x) - q(x)$, which will be a polynomial of degree $\leq n$. Furthermore, we have

$$g(x_i) = p(x_i) - q(x_i) = y_i - y_i = 0, \qquad i = 0, 1, \cdots, n$$

So $g(x)$ has $n+1$ zeros. We must have $g(x) \equiv 0$, therefore $p(x) \equiv q(x)$.

2.4 Errors in Polynomial Interpolation

Given a function $f(x)$, and $a \le x \le b$, a set of distinct points x_i, $i = 0, 1, \cdots, n$, and $x_i \in [a, b]$. Let $P_n(x)$ be a polynomial of degree $\le n$ that interpolates $f(x)$ at x_i, i.e.,

$$P_n(x_i) = f(x_i), \qquad i = 0, 1, \cdots, n$$

Define the error

$$e(x) = f(x) - P_n(x)$$

Theorem *There exists a point $\xi \in [a, b]$, such that*

$$e(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i), \qquad \textit{for all} \;\; x \in [a, b].$$

Proof. . If $f \in \mathcal{P}_n$, then $f(x) = P_n(x)$, trivial.

Now assume $f \notin \mathcal{P}_n$. For $x = x_i$, we have $e(x_i) = f(x_i) - P_n(x_i) = 0$, OK.

Now fix an a such that $a \ne x_i$ for any i. We define

$$W(x) = \prod_{i=0}^{n} (x - x_i) \quad \in \mathcal{P}_{n+1}$$

and a constant

$$c = \frac{f(a) - P_n(a)}{W(a)},$$

and another function

$$\varphi(x) = f(x) - P_n(x) - cW(x).$$

Now we find all the zeros for this function φ:

$$\varphi(x_i) = f(x_i) - P_n(x_i) - cW(x_i) = 0, \qquad i = 0, 1, \cdots, n$$

and

$$\varphi(a) = f(a) - P_n(a) - cW(a) = 0$$

So, φ has at least $(n+2)$ zeros.

Here goes our deduction:

$$\begin{array}{rccl}
\varphi(x) \text{ has at least} & n+2 & \text{zeros.} & \\
\varphi'(x) \text{ has at least} & n+1 & \text{zeros.} & \\
\varphi''(x) \text{ has at least} & n & \text{zeros.} & \\
 & \vdots & & \\
\varphi^{(n+1)}(x) \text{ has at least} & 1 & \text{zero.} & \boxed{\text{Call it } \xi.}
\end{array}$$

So we have

$$\varphi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - 0 - cW^{(n+1)}(\xi) = 0.$$

Use

$$W^{(n+1)} = (n+1)!$$

we get

$$f^{(n+1)}(\xi) = cW^{(n+1)}(\xi) = \frac{f(a) - P_n(a)}{W(a)}(n+1)!.$$

Change a into x, we get

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

Example $n = 1$, $x_0 = a, x_1 = b$, $b > a$.

We have an upper bound for the error, for $x \in [a, b]$,

$$|e(x)| = \frac{1}{2} \left| f''(\xi) \right| \cdot |(x-a)(x-b)| \le \frac{1}{2} \left\| f'' \right\|_\infty \frac{(b-a)^2}{4} = \frac{1}{8} \left\| f'' \right\|_\infty (b-a)^2.$$

Observation: Different distribution of nodes x_i would give different errors.

Uniform nodes: equally distribute the space. Consider an interval $[a, b]$, and we distribute $n+1$ nodes uniformly as

$$x_i = a + ih, \qquad h = \frac{b-a}{n}, \qquad i = 0, 1, \cdots, n.$$

One can show that

$$\prod_{i=0}^{n} |x - x_i| \ \le\ \frac{1}{4} h^{n+1} \cdot n!$$

(Try to prove it!)

This gives the error estimate

$$|e(x)| \le \frac{1}{4(n+1)} \left| f^{(n+1)}(x) \right| h^{n+1} \le \frac{M_{n+1}}{4(n+1)} h^{n+1}$$

where

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

Example Consider interpolating $f(x) = \sin(\pi x)$ with polynomial on the interval $[-1, 1]$ with uniform nodes. Give an upper bound for error, and show how it is related with total number of nodes with some numerical simulations.

Answer. We have
$$\left|f^{(n+1)}(x)\right| \;\leq\; \pi^{n+1}$$
so the upper bound for error is
$$|e(x)| = |f(x) - P_n(x)| \leq \frac{\pi^{n+1}}{4(n+1)} \left(\frac{2}{n}\right)^{n+1}.$$

Below is a table of errors from simulations with various n.

n	error bound	measured error
4	4.8×10^{-1}	1.8×10^{-1}
8	3.2×10^{-3}	1.2×10^{-3}
16	1.8×10^{-9}	6.6×10^{-10}

Problem with uniform nodes: peak of errors near the boundaries. See plots.

Chebychev nodes: equally distributing the error.

Type I: including the end points.

For interval $[-1,1]$: $\bar{x}_i = \cos(\frac{i}{n}\pi), \quad i = 0, 1, \cdots, n$

For interval $[a,b]$: $\bar{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos(\frac{i}{n}\pi), \quad i = 0, 1, \cdots, n$

With this choice of nodes, one can show that
$$\prod_{k=0}^{n} |x - \bar{x}_k| \;=\; 2^{-n} \;\leq\; \prod_{k=0}^{n} |x - x_k|$$
where x_k is any other choice of nodes.

This gives the error bound:
$$|e(x)| \;\leq\; \frac{1}{(n+1)!}\left|f^{(n+1)}(x)\right| 2^{-n}.$$

Example Consider the same example with uniform nodes, $f(x) = \sin \pi x$. With Chebyshev nodes, we have
$$|e(x)| \;\leq\; \frac{1}{(n+1)!}\pi^{n+1} 2^{-n}.$$

The corresponding table for errors:

n	error bound	measured error
4	1.6×10^{-1}	1.15×10^{-1}
8	3.2×10^{-4}	2.6×10^{-4}
16	1.2×10^{-11}	1.1×10^{-11}

The errors are much smaller!

Type II: Chebyshev nodes can be chosen strictly inside the interval $[a, b]$:

$$\bar{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos(\frac{2i+1}{2n+2}\pi), \quad i = 0, 1, \cdots, n$$

See slides for examples.

Theorem *If $P_n(x)$ interpolates $f(x)$ at $x_i \in [a, b]$, $i = 0, 1, \cdots, n$, then*

$$f(x) - P_n(x) = f[x_0, x_1, \cdots, x_n, x] \cdot \prod_{i-0}^{n}(x - x_i), \quad \forall x \neq x_i.$$

Proof. Let $a \neq x_i$, let $q(x)$ be a polynomial that interpolates $f(x)$ at $x_0, x_1, \cdots, x_n, a$. Newton's form gives

$$q(x) = P_n(x) + f[x_0, x_1, \cdots, x_n, a]\prod_{i=0}^{n}(x - x_i).$$

Since $q(a) = f(a)$, we get

$$f(a) = q(a) = P_n(a) + f[x_0, x_1, \cdots, x_n, a]\prod_{i=0}^{n}(a - x_i).$$

Switching a to x, we prove the Theorem.

As a consequence, we have:

$$f[x_0, x_1, \cdots, x_n] = \frac{1}{n!}f^{(n)}(\xi), \qquad \xi \in [a, b].$$

Proof. Let $P_{n-1}(x)$ interpolate $f(x)$ at $x_0, \cdots, x_{n-1}$. The error formula gives

$$f(x_n) - P_{n-1}(x_n) = \frac{1}{n!}f^{(n)}(\xi)\prod_{i=0}^{n}(x_n - x_i), \qquad \xi \in (a, b).$$

From above we know

$$f(x_n) - P_{n-1}(x_n) = f[x_0, \cdots, x_n]\prod_{i=0}^{n}(x_n - x_i)$$

Comparing the rhs of these two equation, we get the result.

Observation: Newton's divided differences are related to derivatives.

$n = 1$: $f[x_0, x_1] = f'(\xi)$, $\xi \in (x_0, x_1)$
$n = 2$: $f[x_0, x_1, x_2] = f''(\xi)$. Let $x_0 = x - h, x_1 = x, x_2 = x + h$, then

$$f[x_0, x_1, x_2] = \frac{1}{2h^2}[f(x+h) - 2f(x) + f(x+h)] = \frac{1}{2}f''(\xi), \quad \xi \in [x-h, x+h].$$

2.5 Numerical differentiations

Finite difference:

$$\begin{array}{llll} (1) & f'(x) & \approx & \frac{1}{h}\left(f(x+h) - f(x)\right) \\ (2) & f'(x) & \approx & \frac{1}{h}\left(f(x) - f(x-h)\right) \\ (3) & f'(x) & \approx & \frac{1}{2h}\left(f(x+h) - f(x-h)\right) \qquad \text{(central difference)} \\ & f''(x) & \approx & \frac{1}{h^2}\left(f(x+h) - 2f(x) + f(x-h)\right) \end{array}$$

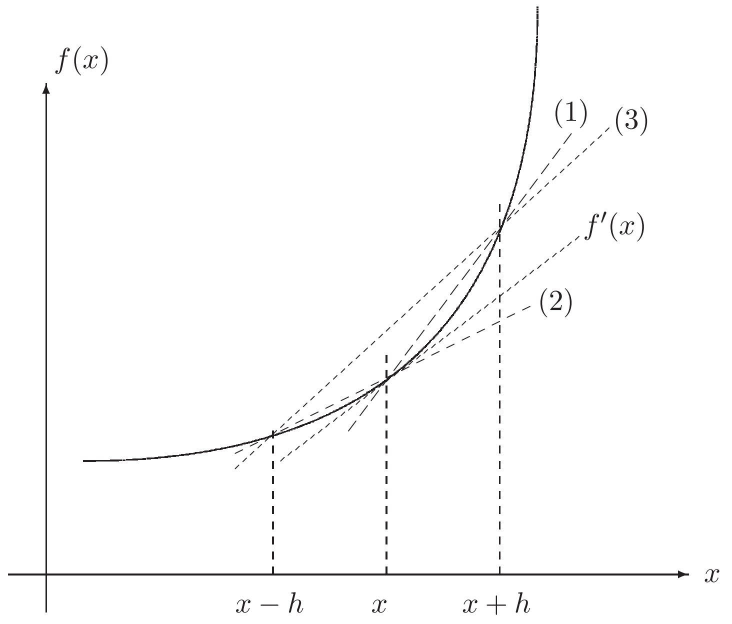

Figure 2.1: Finite differences to approximate derivatives

Truncation erros in Taylor expansion

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4) \\ f(x-h) &= f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4) \end{aligned}$$

Then,

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{1}{2}hf''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h), \quad (1^{st}\text{order})$$

similarly

$$\frac{f(x)-f(x-h)}{h} = f'(x) - \frac{1}{2}hf''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h), \quad (1^{st}\text{order})$$

and

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x) - \frac{1}{6}h^2 f'''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^2), \quad (2^{nd}\text{order})$$

finally

$$\frac{f(x+h)-2f(x)+f(x-h)}{h^2} = f''(x) + \frac{1}{12}h^2 f^{(4)}(x) + \mathcal{O}(h^4) = f''(x) + \mathcal{O}(h^2), \quad (2^{nd}\text{order})$$

Richardson Extrapolation : will be discussed later, in numerical integration, with Romberg's algorithm

Chapter 3

Piece-wise polynomial interpolation. Splines

3.1 Introduction

Usage:

- visualization of discrete data
- graphic design –VW car design

Requirement:

- interpolation
- certain degree of smoothness

Disadvantages of polynomial interpolation $P_n(x)$

- n-time differentiable. We do not need such high smoothness;
- big error in certain intervals (esp. near the ends);
- no convergence result;
- Heavy to compute for large n

Suggestion: use piecewise polynomial interpolation.

Problem setting : Given a set of data

x	t_0	t_1	$\cdots$	t_n
y	y_0	y_1	$\cdots$	y_n

Find a function $\mathcal{S}(x)$ which interpolates the points $(t_i, y_i)_{i=0}^n$.
The set $t_0, t_1, \cdots, t_n$ are called knots.
$\mathcal{S}(x)$ consists of piecewise polynomials

$$\mathcal{S}(x) \doteq \begin{cases} \mathcal{S}_0(x), & t_0 \le x \le t_1 \\ \mathcal{S}_1(x), & t_1 \le x \le t_2 \\ \vdots & \\ \mathcal{S}_{n-1}(x), & t_{n-1} \le x \le t_n \end{cases}$$

$\mathcal{S}(x)$ is called *a spline of degree* n, if

- $\mathcal{S}_i(x)$ is a polynomial of degree n;
- $\mathcal{S}(x)$ is $(n-1)$ times continuous differentiable, i.e., for $i = 1, 2, \cdots, n-1$ we have

$$\begin{aligned} \mathcal{S}_{i-1}(t_i) &= \mathcal{S}_i(t_i), \\ \mathcal{S}'_{i-1}(t_i) &= \mathcal{S}'_i(t_i), \\ &\vdots \\ \mathcal{S}^{(n-1)}_{i-1}(t_i) &= \mathcal{S}^{(n-1)}_i(t_i), \end{aligned}$$

Commonly used ones:

- $n = 1$: linear splines (simplest)
- $n = 1$: quadratic splines
- $n = 3$: cubic splines (most used)

3.2 First degree and second degree splines

Linear splines: $n = 1$. Piecewise linear interpolation, i.e., straight line between 2 neighboring points. See Figure 3.1.
So

$$\mathcal{S}_i(x) = a_i + b_i x, \qquad i = 0, 1, \cdot, n-1$$

Requirements:

$$\begin{aligned} \mathcal{S}_0(t_0) &= y_0 \\ \mathcal{S}_{i-1}(t_i) = \mathcal{S}_i(t_i) &= y_i, \qquad i = 1, 2, \cdots, n-1 \\ \mathcal{S}_{n-1}(t_n) &= y_n. \end{aligned}$$

Easy to find: write the equation for a line through two points: (t_i, y_i) and (t_{i+1}, y_{i+1}),

$$\mathcal{S}_i(x) = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i}(x - t_i), \qquad i = 0, 1, \cdots, n-1.$$

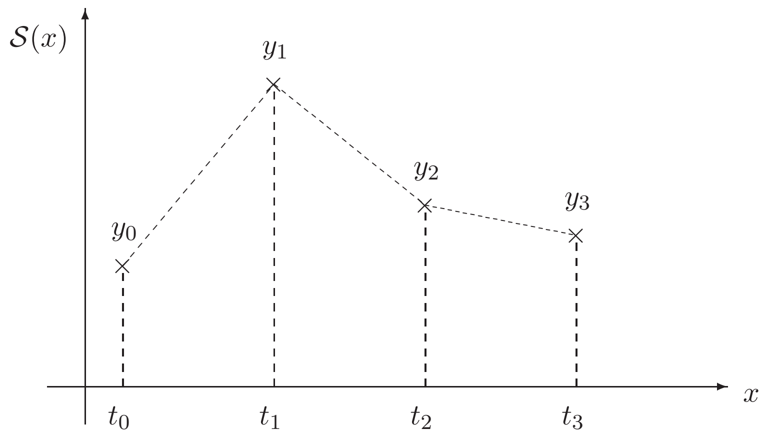

Figure 3.1: Linear splines

Accuracy Theorem for linear splines: Assume $t_0 < t_1 < t_2 < \cdots < t_n$, and let

$$h = \max_i (t_{i+1} - t_i)$$

Let $f(x)$ be a given function, and let $\mathcal{S}(x)$ be a linear spline that interpolates $f(x)$ s.t.

$$\mathcal{S}(t_i) = f(t_i), \qquad i = 0, 1, \cdots, n$$

We have the following, for $x \in [t_0, t_n]$,

(1) If f' exists and is continuous, then

$$|f(x) - \mathcal{S}(x)| \leq \frac{1}{2} h \max_x |f'(x)| .$$

(2) If f'' exits and is continuous, then

$$|f(x) - \mathcal{S}(x)| \leq \frac{1}{8} h^2 \max_x |f''(x)| .$$

Quadratics splines. read the textbook if you want.

3.3 Natural cubic splines

Given $t_0 < t_1 < \cdots < t_n$, we define the *cubic spline* $\mathcal{S}(x) = \mathcal{S}_i(x)$ for $t_i \leq x \leq t_{i+1}$. We require that $\mathcal{S}, \mathcal{S}', \mathcal{S}''$ are all continuous. If in addition we require $\mathcal{S}_0''(t_0) = \mathcal{S}_{n-1}''(t_n) = 0$, then it is called *natural cubic spline.*

Write

$$\mathcal{S}_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \qquad i = 0, 1, \cdots, n-1$$

Total number of unknowns$= 4 \cdot n$.

Equations we have

$$\left.\begin{array}{llll} & \text{equation} & & \text{number} \\ (1) & \mathcal{S}_i(t_i) = y_i, & i = 0, 1, \cdots, n-1 & n \\ (2) & \mathcal{S}_i(t_{i+1}) = y_{i+1}, & i = 0, 1, \cdots, n-1 & n \\ (3) & \mathcal{S}'_i(t_{i+1}) = \mathcal{S}'_{i+1}(t_i), & i = 0, 1, \cdots, n-2 & n-1 \\ (4) & \mathcal{S}''_i(t_{i+1}) = \mathcal{S}''_{i+1}(t_i), & i = 0, 1, \cdots, n-2 & n-1 \\ (5) & \mathcal{S}''_0(t_0) = 0, & & 1 \\ (6) & \mathcal{S}''_{n-1}(t_n) = 0, & & 1. \end{array}\right\} \text{total} = 4n.$$

How to compute $\mathcal{S}_i(x)$? We know:

$\mathcal{S}_i$	:	polynomial of degree 3
$\mathcal{S}'_i$	:	polynomial of degree 2
$\mathcal{S}''_i$	:	polynomial of degree 1

procedure:

- Start with $\mathcal{S}''_i(x)$, they are all linear, one can use Lagrange form,
- Integrate $\mathcal{S}''_i(x)$ twice to get $\mathcal{S}_i(x)$, you will get 2 integration constant
- Determine these constants by (2) and (1). Various tricks on the way...

Details: Define z_i as

$$z_i = \mathcal{S}''(t_i), \qquad i = 1, 2, \cdots, n-1, \qquad z_0 = z_n = 0$$

NB! These z_i's are our unknowns.

Introduce the notation $h_i = t_{i+1} - t_i$.

Lagrange form

$$\mathcal{S}''_i(x) = \frac{z_{i+1}}{h_i}(x - t_i) - \frac{z_i}{h_i}(x - t_{i+1}).$$

Then

$$\begin{aligned} \mathcal{S}'_i(x) &= \frac{z_{i+1}}{2h_i}(x - t_i)^2 - \frac{z_i}{2h_i}(x - t_{i+1})^2 + C_i - D_i \\ \mathcal{S}_i(x) &= \frac{z_{i+1}}{6h_i}(x - t_i)^3 - \frac{z_i}{6h_i}(x - t_{i+1})^3 + C_i(x - t_i) - D_i(x - t_{i+1}). \end{aligned}$$

(You can check by yourself that these $\mathcal{S}_i, \mathcal{S}'_i$ are correct.)

Interpolating properties:
(1). $\mathcal{S}_i(t_i) = y_i$ gives

$$y_i = -\frac{z_i}{6h_i}(-h_i)^3 - D_i(-h_i) = \frac{1}{6}z_i h_i^2 + D_i h_i \quad \Rightarrow \quad D_i = \frac{y_i}{h_i} - \frac{h_i}{6}z_i$$

(2). $\mathcal{S}_i(t_{i+1}) = y_{i+1}$ gives

$$y_{i+1} = \frac{z_{i+1}}{6h_i}h_i^3 + C_i h_i, \quad \Rightarrow \quad C_i = \frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1}.$$

We see that, once z_i's are known, then (C_i, D_i)'s are known, and so $\mathcal{S}_i, \mathcal{S}_i'$ are known.

$$\begin{aligned}
\mathcal{S}_i(x) &= \frac{z_{i+1}}{6h_i}(x-t_i)^3 - \frac{z_i}{6h_i}(x-t_{i+1})^3 + \left(\frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1}\right)(x-t_i) \\
&\quad - \left(\frac{y_i}{h_i} - \frac{h_i}{6}z_i\right)(x-t_{i+1}). \\
\mathcal{S}_i'(x) &= \frac{z_{i+1}}{2h_i}(x-t_i)^2 - \frac{z_i}{2h_i}(x-t_{i+1})^2 \frac{y_{i+1}-y_i}{h_i} - \frac{z_{i+1}-z_i}{6}h_i.
\end{aligned}$$

How to compute z_i's? Last condition that's not used yet: continuity of $\mathcal{S}'(x)$, i.e.,

$$\mathcal{S}_{i-1}'(t_i) = \mathcal{S}_i'(t_i), \qquad i = 1, 2, \cdots, n-1$$

We have

$$\begin{aligned}
\mathcal{S}_i'(t_i) &= -\frac{z_i}{2h_i}(-h_i)^2 + \underbrace{\frac{y_{i+1}-y_i}{h_i}}_{b_i} - \frac{z_{i+1}-z_i}{6}h_i \\
&= -\frac{1}{6}h_i z_{i+1} - \frac{1}{3}h_i z_i + b_i \\
\mathcal{S}_{i-1}'(t_i) &= \frac{1}{6}z_{i-1}h_{i-1} + \frac{1}{3}z_i h_{i-1} + b_{i-1}
\end{aligned}$$

Set them equal to each other, we get

$$\begin{cases} h_{i-1}z_{i-1} + 2(h_{i-1}+h_i)z_i + h_i z_{i+1} = 6(b_i - b_{i-1}), & i = 1, 2, \cdots, n-1 \\ z_0 = z_n = 0. \end{cases}$$

In matrix-vector form:

$$\mathbf{H} \cdot \vec{z} = \vec{b}$$

where

$$\mathbf{H} = \begin{pmatrix}
2(h_0+h_1) & h & & & & \\
h_1 & 2(h_1+h_2) & h_2 & & & \\
& h_2 & 2(h_2+h_3) & h_3 & & \\
& \ddots & \ddots & \ddots & & \\
& & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\
& & & h_{n-2} & 2(h_{n-2}+h_{n-1})
\end{pmatrix}$$

and

$$\vec{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} 6(b_1 - b_0) \\ 6(b_2 - b_1) \\ 6(b_3 - b_2) \\ \vdots \\ 6(b_{n-2} - b_{n-3}) \\ 6(b_{n-1} - b_{n-2}) \end{pmatrix}.$$

Here, **H** is a tri-diagonal matrix, symmetric, and diagonal dominant

$$2\,|h_{i-1} + h_i| > |h_i| + |h_{i-1}|$$

which implies unique solution for $\vec{z}$.

See slides for Matlab codes and solution graphs.

Theorem on smoothness of cubic splines. *If $\mathcal{S}$ is the natural cubic spline function that interpolates a twice-continuously differentiable function f at knots*

$$a = t_0 < t_1 < \cdots < t_n = b$$

then

$$\int_a^b \left[\mathcal{S}''(x)\right]^2 \, dx \le \int_a^b \left[f''(x)\right]^2 \, dx.$$

Note that $\int (f'')^2$ is related to the curvature of f.

Cubic spline gives the least curvature, $\Rightarrow$ most smooth, so best choice.

Proof. Let

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

Then

$$g(t_i) = 0, \qquad i = 0, 1, \cdots, n$$

and $f'' = \mathcal{S}'' + g''$, so

$$(f'')^2 = (\mathcal{S}'')^2 + (g'')^2 + 2\mathcal{S}''g''$$

$$\Rightarrow \quad \int_a^b (f'')^2 \, dx = \int_a^b (\mathcal{S}'')^2 \, dx + \int_a^b (g'')^2 \, dx + \int_a^b 2\mathcal{S}''g'' \, dx$$

Claim that

$$\int_a^b \mathcal{S}''g'' \, dx = 0$$

then this would imply

$$\int_a^b (f'')^2 \, dx \ge \int_a^b (\mathcal{S}'')^2 \, dx$$

and we are done.

Proof of the claim: Using integration-by-parts,

$$\int_a^b \mathcal{S}'' g''\, dx = \mathcal{S}'' g' \Big|_a^b - \int_a^b \mathcal{S}''' g'\, dx$$

Since $g(a) = g(b) = 0$, so the first term is 0. For the second term, since $\mathcal{S}'''$ is piecewise constant. Call

$$c_i = \mathcal{S}'''(x), \quad \text{for} \quad x \in [t_i, t_{i+1}].$$

Then

$$\int_a^b \mathcal{S}''' g'\, dx = \sum_{i=0}^{n-1} c_i \int_{t_i}^{t_{i+1}} g'(x)\, dx = \sum_{i=0}^{n-1} c_i \left[g(t_{i+1}) - g(t_i)\right] = 0,$$

(b/c $g(t_i) = 0$).

Chapter 4

Numerical integration

4.1 Introduction

Problem: Given a function $f(x)$ on interval $[a, b]$, find an approximation to the integral

$$I(f) = \int_a^b f(x)\,dx$$

Main idea:

- Cut up $[a, b]$ into smaller sub-intervals
- In each sub-interval, find a polynomial $p^i(x) \approx f(x)$
- Integrate $p^i(x)$ on each sub-interval, and sum up

4.2 Trapezoid rule

The grid: cut up $[a, b]$ into n sub-intervals:

$$x_0 = a, \quad x_i < x_{i+1}, \quad x_n = b$$

On interval $[x_i, x_{i+1}]$, approximate $f(x)$ by a linear polynomial
We use

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx \int_{x_i}^{x_{i+1}} p^i(x)\,dx = \frac{1}{2}\left(f(x_{i+1}) + f(x_i)\right)(x_{i+1} - x_i)$$

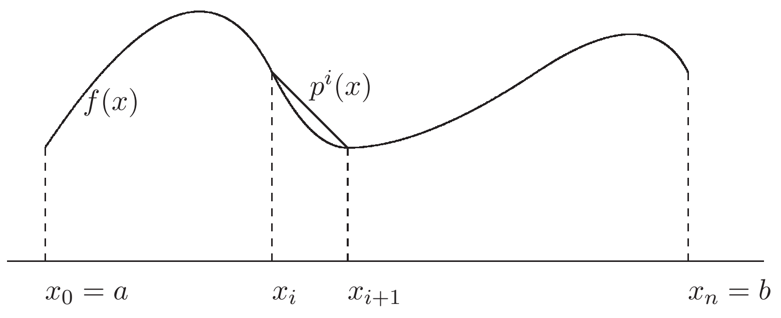

Figure 4.1: Trapezoid rule: straight line approximation in each sub-interval.

Summing up all the sub-intervals

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx \\
&\approx \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} p^i(x)\,dx \\
&= \sum_{i=0}^{n-1} \frac{1}{2}\left(f(x_{i+1}) + f(x_i)\right)(x_{i+1} - x_i)
\end{aligned}$$

Uniform grid: $h = \frac{b-a}{n}, x_{i+1} - x_i = h$,

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{i=0}^{n-1} \frac{h}{2}\left(f(x_{i+1}) + f(x_i)\right) \\
&= \underbrace{h\left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right]}_{T(f;h)}
\end{aligned}$$

so we can write

$$\int_a^b f(x) \approx T(f;h)$$

Error estimates.

$$E_T(f;h) = I(f) - T(f;h) = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} \left[f(x) - p^i(x)\right]\,dx$$

Known from interpolation:

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

Basic error: error on each sub-interval:

$$E_{T,i}(f;h) \;=\; \frac{1}{2} f''(\xi_i) \int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1})\, dx \;=\; -\frac{1}{12} h^3 f''(\xi_i).$$

Total error:

$$E_T(f;h) \;=\; \sum_{i=0}^{n-1} E_{T,i}(f;h) = \sum_{i=0}^{n-1} -\frac{1}{12} h^3 f''(\xi_i) \;=\; -\frac{1}{12} h^3 \underbrace{\left[\sum_{i=0}^{n-1} f''(\xi_i)\right] \cdot \frac{1}{n}}_{= f''(\xi)} \cdot \underbrace{\frac{b-a}{h}}_{= n}$$

Total error is

$$E_T(f;h) \;=\; \frac{b-a}{12} h^2 f''(\xi), \qquad \xi \in (a,b).$$

Error bound

$$E_T(f;h) \;\le\; \frac{b-a}{12} h^2 \max_{x \in (a,b)} \left| f''(x) \right| .$$

Example Consider function $f(x) = e^x$, and the integral

$$I(f) = \int_0^2 e^x \, dx$$

Require error $\le 0.5 \times 10^{-4}$. How many points should be used in the Trapezoid rule?

Answer. We have

$$f'(x) = e^x, \quad f''(x) = e^x, \quad a = 0, \quad b = 2$$

so

$$\max_{x \in (a,b)} \left| f''(x) \right| = e^2.$$

By error bound, it is sufficient to require

$$\begin{aligned} & |E_T(f;h)| \le \frac{1}{6} h^2 e^2 \le 0.5 \times 10^{-4} \\ \Rightarrow\quad & h^2 \le 0.5 \times 10^{-4} \times 6 \times e^{-2} \approx 4.06 \times 10^{-5} \\ \Rightarrow\quad & \frac{2}{n} = h \le \sqrt{4.06 \times 10^{-5}} = 0.0064 \\ \Rightarrow\quad & n \ge \frac{2}{0.0064} \approx 313.8 \end{aligned}$$

We need at least 314 points.

4.3 Simpson's rule

Cut up $[a, b]$ into $2n$ equal sub-intervals

$$x_0 = a, \quad x_{2n} = b, \quad h = \frac{b-a}{2n}, \quad x_{i+1} - x_i = h$$

Consider the interval $[x_{2i}, x_{2i+2}]$. Find a 2nd order polynomial that interpolates $f(x)$ at the points

$$x_{2i}, x_{2i+1}, x_{2i+2}$$

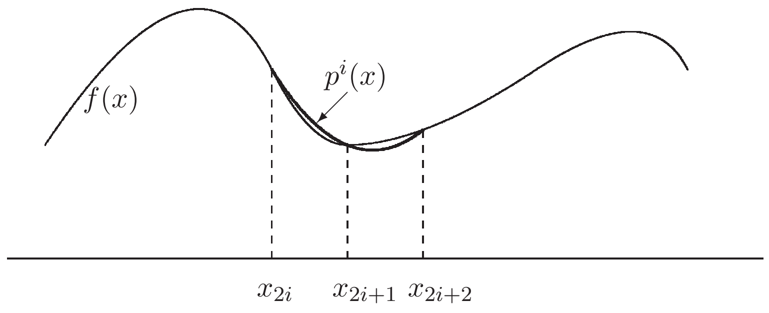

Figure 4.2: Simpson's rule: quadratic polynomial approximation (thick line) in each sub-interval.

Lagrange form gives

$$\begin{aligned}
p^i(x) &= f(x_{2i})\frac{(x-x_{2i+1})(x-x_{2i+2})}{(x_{2i}-x_{2i+1})(x_{2i}-x_{2i+2})} + f(x_{2i+1})\frac{(x-x_{2i})(x-x_{2i+2})}{(x_{2i+1}-x_{2i})(x_{2i+1}-x_{2i+2})} \\
&\quad + f(x_{2i+2})\frac{(x-x_{2i})(x-x_{2i+1})}{(x_{2i+2}-x_{2i})(x_{2i+2}-x_{2i+1})} \\
&= \frac{1}{2h^2}f(x_{2i})(x-x_{2i+1})(x-x_{2i+2}) - \frac{1}{h^2}f(x_{2i+1})(x-x_{2i})(x-x_{2i+2}) \\
&\quad + \frac{1}{2h^2}f(x_{2i+2})(x-x_{2i})(x-x_{2i+1})
\end{aligned}$$

Then

$$\begin{aligned}
\int_{x_{2i}}^{x_{2i+2}} f(x)\,dx &\approx \int_{x_{2i}}^{x_{2i+2}} p^i(x)\,dx \\
&= \frac{1}{2h^2} f(x_{2i}) f(x_{2i}) \underbrace{\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i+1})(x - x_{2i+2})\,dx}_{\frac{2}{3}h^3} \\
&\quad - \frac{1}{h^2} f(x_{2i+1}) \underbrace{\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+2})\,dx}_{-\frac{4}{3}h^3} \\
&\quad + \frac{1}{2h^2} f(x_{2i+2}) \underbrace{\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+1})\,dx}_{\frac{2}{3}h^3} \\
&= \frac{h}{3}\left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right]
\end{aligned}$$

Putting together

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx S(f;h) \\
&= \sum_{i=0}^{n-1} \int_{x_{2i}}^{x_{2i+2}} p^i(x)\,dx \\
&= \frac{h}{3} \sum_{i=0}^{n-1} \left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right]
\end{aligned}$$

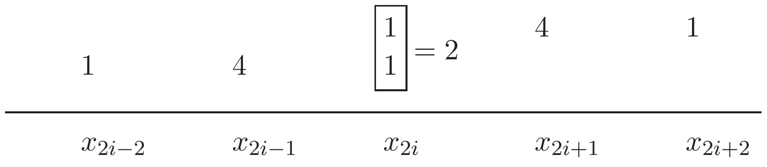

Figure 4.3: Simpson's rule: adding the constants in each node.

See Figure 4.3 for the counting of coefficients on each node. Wee see that for x_0, x_{2n} we get 1, and for odd indices we have 4, and for all remaining even indices we get 2.

The algorithm looks like:

$$S(f;h) = \frac{h}{3}\left[f(x_0) + 4\sum_{i=1}^{n} f(x_{2i-1}) + 2\sum_{i=1}^{n-1} f(x_{2i}) + f(x_{2n})\right]$$

Error estimate. basic error is

$$-\frac{1}{90}h^5 f^{(4)}(\xi_i), \qquad \xi_i \in (x_{2i}, x_{2i+2})$$

so

$$E_S(f;h) = I(f) - S(f;h) = -\frac{1}{90}h^5 \sum_{i=0}^{n-1} f^{(4)}(\xi_i)\frac{1}{n} \cdot \frac{b-a}{2h} = -\frac{b-a}{180}h^4 f^{(4)}(\xi), \quad \xi(a,b)$$

Error bound

$$|E_S(f;h)| \le \frac{b-a}{180}h^4 \max_{x\in(a,b)} \left| f^{(4)}(x) \right|.$$

Example With $f(x) = e^x$ in $[0,2]$, now use Simpson's rule, to achieve an error $\le 0.5 \times 10^{-4}$, how many points must one take?

Answer. We have

$$\begin{aligned} |E_S(f;h)| &\le \frac{2}{180}h^4 e^2 \le 0.5 \times 10^{-4} \\ \Rightarrow \quad & h^4 \le 0.5^{-4} \times 180/e^2 = 1.218 \times 10^{-3} \\ \Rightarrow \quad & h \le 0.18682 \\ \Rightarrow \quad & n = \frac{b-a}{2h} = 5.3 \approx 6 \end{aligned}$$

We need at least $2n+1 = 13$ points.

Note: This is much fewer points than using Trapezoid Rule.

4.4 Recursive trapezoid rule

These are called *composite schemes*.

Divide $[a,b]$ into 2^n equal sub-intervals.

$$h_n = \frac{b-a}{2^n}, \qquad h_{n+1} = \frac{1}{2}h$$

So

$$\begin{aligned} T(f;h_n) &= h_n \cdot \left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{i=1}^{2^n-1} f(a+ih_n) \right] \\ T(f;h_{n+1}) &= h_{n+1} \cdot \left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{i=1}^{2^{n+1}-1} f(a+ih_{n+1}) \right] \end{aligned}$$

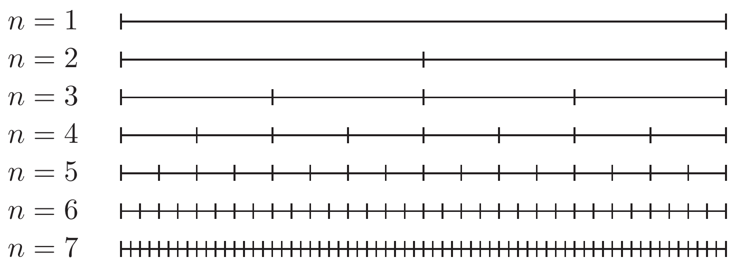

Figure 4.4: Recursive division of intervals, first few levels

We can re-arrange the terms in $T(f; h_{n+1})$:

$$\begin{aligned} T(f; h_{n+1}) &= \frac{h_n}{2}\left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{i=1}^{2^n-1} f(a+ih_n) + \sum_{j=1}^{2^n-1} f(a+(2j+1)h_{n+1})\right] \\ &= \frac{1}{2}T(f; h_n) + h_{n+1}\sum_{j=1}^{2^n-1} f(a+(2j+1)h_{n+1}) \end{aligned}$$

Advantage: One case keep the computation for a level n. If this turns out to be not accurate enough, then add one more level to get better approximation. $\Rightarrow$ flexibility.

4.5 Romberg Algorithm

If $f^{(n)}$ exists and is bounded, then we have the Euler MacLaurin's formula for error

$$\begin{aligned} E(f; h) &= I(f) - T(f; h) = a_2h^2 + a_4h^4 + a_6h^6 + \cdots + a_nh^n \\ E(f; \frac{h}{2}) &= I(f) - T(f; \frac{h}{2}) = a_2(\frac{h}{2})^2 + a_4(\frac{h}{2})^4 + a_6(\frac{h}{2})^6 + \cdots + a_n(\frac{h}{2})^n \end{aligned}$$

Here a_n depends on the derivatives $f^{(n)}$.

We have

$$\begin{aligned} (1) \quad I(f) &= T(f; h) + a_2h^2 + a_4h^4 + a_6h^6 + \cdots \\ (2) \quad I(f) &= T(f; \frac{h}{2}) + a_2(\frac{h}{2})^2 + a_4(\frac{h}{2})^4 + a_6(\frac{h}{2})^6 + \cdots + a_n(\frac{h}{2})^n \end{aligned}$$

The goal is to use the 2 approximations $T(f; h)$ and $T(f; \frac{h}{2})$ to get one that's more accurate, i.e., we wish to cancel the leading error term, the one with h^2.

Multiply (2) by 4 and subtract (1), gives

$$\begin{aligned} 3 \cdot I(f) &= 4 \cdot T(f; h/2) - T(f; h) + a_4'h^4 + a_6'h^6 + \cdots \\ \Rightarrow I(f) &= \underbrace{\frac{4}{3}T(f; h/2) - \frac{1}{3}T(f; h)}_{U(h)} + \tilde{a}_4h^4 + \tilde{a}_6h^6 + \cdots \end{aligned}$$

$R(h)$ is of 4-th order accuracy! Better than $T(f;h)$. We now write:

$$U(h) = T(f;h/2) + \frac{T(f;h/2) - T(f;h)}{2^2 - 1}$$

This idea is called the *Richardson extrapolation*.

Take one more step:

$$\begin{array}{lrcl} (3) & I(f) & = & U(h) + \tilde{a}_4 h^4 + \tilde{a}_6 h^6 + \cdots \\ (4) & I(f) & = & U(h/2) + \tilde{a}_4 (h/2)^4 + \tilde{a}_6 (h/2)^6 + \cdots \end{array}$$

To cancel the term with h^4: (4) $\times 2^4 -$ (3)

$$(2^4 - 1)I(f) \quad = \quad 2^4 U(h/2) - U(h) + \tilde{a}'_6 h^6 + \cdots$$

Let

$$V(h) = \frac{2^4 U(h/2) - U(h)}{2^4 - 1} = U(h/2) + \frac{U(h/2) - U(h)}{2^4 - 1}.$$

Then

$$I(f) = V(h) + \tilde{a}'_6 h^6 + \cdots$$

So $V(h)$ is even better than $U(h)$.

One can keep doing this several layers, until desired accuracy is reached.

This gives the **Romberg Algorithm**: Set $H = b - a$, define:

$$\begin{aligned} R(0,0) &= T(f;H) = \frac{H}{2}(f(a) + f(b)) \\ R(1,0) &= T(f;H/2) \\ R(2,0) &= T(f;H/(2^2)) \\ &\vdots \\ R(n,0) &= T(f;H/(2^n)) \end{aligned}$$

Here $R(n,0)$'s are computed by the recursive trapezoid formula.

Romberg triangle: See Figure 4.5.

The entry $R(n,m)$ is computed as

$$R(n,m) = R(n,m-1) + \frac{R(n,m-1) - R(n-1,m-1)}{2^{2m-1}}$$

Accuracy:

$$I(f) = R(n,m) + \mathcal{O}(h^{2(m+1)}), \qquad h = \frac{H}{2^n}.$$

Algorithm can be done either column-by-column or row-by-row.

Here we give some pseudo-code, using column-by-column.

$$\begin{tikzcd}
R(0,0) \arrow[dr] & & & & \\
R(1,0) \arrow[r] \arrow[dr] & R(1,1) \arrow[dr] & & & \\
R(2,0) \arrow[r] \arrow[dr] & R(2,1) \arrow[r] \arrow[dr] & R(2,2) \arrow[dr] & & \\
R(3,0) \arrow[r] \arrow[d, dash, dotted] & R(3,1) \arrow[r] \arrow[d, dash, dotted] & R(3,2) \arrow[r] \arrow[d, dash, dotted] & {} \arrow[dr, dash, dashed] & \\
R(n,0) & R(n,1) & R(n,3) \arrow[rr, dash, dashed] & & R(n,n)
\end{tikzcd}$$

Figure 4.5: Romberg triangle

```
$R$ =romberg($f, a, b, n$)
    $R = n \times n$ matrix
    $h = b - a$; $R(1,1) = [f(a) + f(b)] * h/2$;
    for $i = 1$ to $n - 1$ do   %1st column recursive trapezoid
        $R(i+1,1) = R(i,1)/2$;
        $h = h/2$;
        for $k = 1$ to $2^{i-1}$ do
            $R(i+1,1) = R(i+1,1) + h * f(a + (2k-1)h)$
        end
    end
    for $j = 2$ to $n$ do   %2 to $n$ column
        for $i = j$ to $n$ do
            $R(i,j) = R(i,j-1) + \frac{1}{4^j-1}[R(i,j-1) - R(i-1,j-1)]$
        end
    end
```

4.6 Adaptive Simpson's quadrature scheme

Same idea can be adapted to Simpson's rule instead of trapezoid rule.
For interval $[a, b]$, $h = \frac{b-a}{2}$,

$$S_1[a,b] = \frac{b-a}{6}\left[f(a) + 4f(\frac{a+b}{2}) + f(b)\right]$$

Error form:

$$E_1[a,b] = -\frac{1}{90}h^5 f^{(4)}(\xi), \qquad \xi(a,b)$$

Then

$$I(f) = S_1[a,b] + E_1[a,b]$$

Divide $[a,b]$ up in the middle, let $c = \frac{a+b}{2}$.

$$\begin{aligned} I(f)[a,b] &= I(f)[a,c] + I(f)[c,b] \\ &= S_1[a,c] + E_1[a,c] + S_1[c,b] + E_1[c,b] \\ &= S_2[a,b] + E_2[a,b] \end{aligned}$$

where

$$\begin{aligned} S_2[a,b] &= S_1[a,c] + S_1[c,b] \\ E_2[a,b] &= E_1[a,c] + E_1[c,b] = -\frac{1}{90}(h/2)^5\left[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)\right] \end{aligned}$$

Assume $f^{(4)}$ does NOT change much, then $E_1[a,c] \approx E_1[c,b]$, and

$$E_2[a,b] \approx 2E_1[a,c] = 2\frac{1}{2^5}E_1[a,b] = \frac{1}{2^4}E_1[a,b]$$

This gives

$$S_2[a,b] - S_1[a,b] = (I - E_2[a,b]) - (I - E_1[a,b]) = E_1 - E_2 = 2^4E_2 - E_2 = 15E_2$$

This means, we can compute the error E_2:

$$E_2 = \frac{1}{15}(S_2 - S_1)$$

If we wish to have $|E_2| \le \varepsilon$, we only need to require

$$\frac{S_2 - S_1}{2^4 - 1} \le \varepsilon$$

This gives the idea of an adaptive recursive formula:

$$\begin{aligned} &(A) \qquad I = S_1 + E_1 \\ &(B) \qquad I = S_2 + E_2 = S_2 + \frac{1}{2^4}E_1 \end{aligned}$$

$(B) * 2^4 - (A)$ gives

$$(2^4 - 1)I = 2^4S_2 - S_1$$

$$\Rightarrow \quad I = \frac{2^4S_2 - S_1}{2^4 - 1} = S_2 + \frac{S_2 - S_1}{15}$$

Note that this gives the best approximation when $f^{(4)} \approx const$.

Pseudocode: f: function, $[a,b]$ interval, ε: tolerance for error

```
answer=simpson($f, a, b, \varepsilon$)
compute $S_1$ and $S_2$
If $|S_2 - S_1| < 15\varepsilon$
    answer= $S_2 + (S_2 - S_1)/15$;
else
    $c = (a + b)/2$;
    Lans=simpson($f, a, c, \varepsilon/2$);
    Rans=simpson($f, c, b, \varepsilon/2$);
    answer=Lans+Rans;
end
```

In Matlab, one can use `quad` to compute numerical integration. Try `help quad`, it will give you info on it. One can call the program by using:

```
a=quad('fun',a,b,tol)
```

It uses adaptive Simpson's formula.

See also `quad8`, higher order method.

4.7 Gaussian quadrature formulas

We seek numerical integration formulas of the form

$$\int_a^b f(x)\,dx \approx A_0 f(x_0) + A_1 f(x_1) + \cdots + A_n f(x_n),$$

with the weights A_i, $(i = 0, 1, \cdots, n)$ and the *nodes*

$$x_i \in (a, b), \qquad i = 0, 1, \cdots, n$$

How to find nodes and weights?

Nodes x_i: are roots of Legendre polynomials $q_{n+1}(x)$. These polynomials satisfies

$$\int_a^b x^k q_{n+1}(x)\,dx = 0, \qquad (0 \le k \le n)$$

Examples for $n \le 3$, for interval $[-1, 1]$

$$\begin{aligned} q_0(x) &= 1 \\ q_1(x) &= x \\ q_2(x) &= \frac{3}{2}x^2 - \frac{1}{2} \\ q_3(x) &= \frac{5}{2}x^3 - \frac{3}{2}x \end{aligned}$$

The roots are

$$\begin{aligned} q_1 &: \; 0 \\ q_2 &: \; \pm 1/\sqrt{3} \\ q_3 &: \; 0, \quad \pm\sqrt{3/5} \end{aligned}$$

For general interval $[a, b]$, use the transformation:

$$t = \frac{2x - (a+b)}{b-a}, \qquad x = \frac{1}{2}(b-a)t + \frac{1}{2}(a+b)$$

so for $-1 \le t \le 1$ we have $a \le x \le b$.

Weights A_i: Recall $l_i(x)$, the Cardinal form in Lagrange polynomial:

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Then

$$A_i = \int_{-a}^{b} l_i(x)\, dx.$$

There are tables of such nodes and weights (Table 5.1 in textbook).

We skip the proof. See textbook if interested.

Advantage: Since all nodes are in the interior of the interval, these formulas can handle integrals of function that tends to infinite value at one end of the interval (provided that the integral is defined). Examples:

$$\int_0^1 x^{-1}\, dx, \qquad \int_0^1 (x^2-1)^{1/3} \sqrt{\sin(e^x - 1)}\, dx.$$

Chapter 5

Numerical solution of nonlinear equations.

5.1 Introduction

Problem: $f(x)$ given function, real-valued, possibly non-linear. Find a root r of $f(x)$ such that $f(r) = 0$.

Example 1. Quadratic polynomials: $f(x) = x^2 + 5x + 6$.

$$f(x) = (x+2)(x+3) = 0, \quad \Rightarrow \quad r_1 = -2, \quad r_2 = -3.$$

Roots are not unique.

Example 2. $f(x) = x^2 + 4x + 10 = (x+2)^2 + 6$. There are no real r that would satisfy $f(r) = 0$.

Example 3. $f(x) = x^2 + \cos x + e^x + \sqrt{x+1}$. Roots can be difficult/impossible to find analytically.

Our task now: use a numerical method to find an approximation to a root.

Overview of the chapter:

- Bisection (briefly)
- Fixed point iteration (main focus): general iteration, and Newton's method
- Secant method

Systems (*) optional...

5.2 Bisection method

Given $f(x)$, continuous function.

- Initialization: Find a, b such that $f(a) \cdot f(b) < 0$.
  This means there is a root $r \in (a, b)$ s.t. $f(r) = 0$.
- Let $c = \frac{a+b}{2}$, mid-point.
- If $f(c) = 0$, done (lucky!)
- else: check if $f(c) \cdot f(a) < 0$ or $f(c) \cdot f(b) < 0$.
- Pick that interval $[a, c]$ or $[c, b]$, and repeat the procedure until stop criteria satisfied.

Stop Criteria:

1) interval small enough
2) $|f(c_n)|$ almost 0
3) max number of iteration reached
4) any combination of the previous ones.

Convergence analysis: Consider $[a_0, b_0]$, $c_0 = \frac{a_0+b+0}{2}$, let $r \in (a_0, b_0)$ be a root. The error:

$$e_0 = |r - c_0| \leq \frac{b_0 - a_0}{2}$$

Denote the further intervals as $[a_n, b_n]$ for iteration no. n. Then

$$e_n = |r - c_n| \leq \frac{b_n - a_n}{2} \leq \frac{b_0 - a_0}{2^{n+1}} = \frac{e_0}{2^n}.$$

If the error tolerance is ε, we require $e_n \leq \varepsilon$, then

$$\frac{b_0 - a_0}{2^{n+1}} \leq \varepsilon \quad \Rightarrow \quad n \geq \frac{\log(b-a) - \log(2\varepsilon)}{\log 2}, \quad (\# \text{ of steps})$$

Remark: very slow convergence.

5.3 Fixed point iterations

Rewrite the equation $f(x) = 0$ into the form $x = g(x)$.

Remark: This can always be achieved, for example: $x = f(x) + x$. The catch is that, the choice of g makes a difference in convergence.

Iteration algorithm:

- Choose a start point x_0,
- Do the iteration $x_{k+1} = x_k$, $k = 0, 1, 2, \cdots$ until meeting stop crietria.

Stop Criteria: Let ε be the tolerance

- $|x_k - x_{k-1}| \le \varepsilon$,
- $|x_k - g(x_k)| \le \varepsilon$,
- max # of iteration reached,
- any combination.

Example 1. $f(x) = x - cosx$.
Choose $g(x) = \cos x$, we have $x = \cos x$.
Choose $x_0 = 1$, and do the iteration $x_{k+1} = \cos(x_k)$:

$$\begin{aligned}
x_1 &= \cos x_0 = 0.5403 \\
x_2 &= \cos x_1 = 0.8576 \\
x_3 &= \cos x_2 = 0.6543 \\
&\vdots \\
x_{23} &= \cos x_{22} = 0.7390 \\
x_{24} &= \cos x_{23} = 0.7391 \\
x_{25} &= \cos x_{24} = 0.7391 \quad \text{stop here}
\end{aligned}$$

Our approximation to the root is 0.7391.

Example 2. Consider $f(x) = e^{-2x}(x-1) = 0$. We see that $r = 1$ is a root.
Rewrite as

$$x = g(x) = e^{-2x}(x-1) + x$$

Choose an initial guess $x_0 = 0.99$, very close to the real root. Iterations:

$$\begin{aligned}
x_1 &= \cos x_0 = 0.9886 \\
x_2 &= \cos x_1 = 0.9870 \\
x_3 &= \cos x_2 = 0.9852 \\
&\vdots \\
x_{27} &= \cos x_{26} = 0.1655 \\
x_{28} &= \cos x_{27} = -0.4338 \\
x_{29} &= \cos x_{28} = -3.8477 \quad \text{Diverges. It does not work.}
\end{aligned}$$

Convergence depends on x_0 and $g(x)$!

Convergence analysis. Let r be the exact root, s.t., $r = g(r)$.

Our iteration is $x_{k+1} = g(x_k)$.

Define the error: $e_k = x_k - r$. Then,

$$\begin{aligned} e_{k+1} &= x_{k+1} - r = g(x_k) - r = g(x_k) - g(r) \\ &= g'(\xi)(x_k - r) \qquad (\xi \in (x_k, r), \text{ since } g \text{ is continuous}) \\ &= g'(\xi)e_k \end{aligned}$$

$$\Rightarrow \quad |e_{k+1}| \leq \left|g'(\xi)\right| |e_k|$$

Observation:

- If $|g'(\xi)| < 1$, error decreases, the iteration convergence. (linear convergence)
- If $|g'(\xi)| \geq 1$, error increases, the iteration diverges.

Convergence condition: There exists an interval around r, say $[r-a, r+a]$ for some $a > 0$, such that $|g'(x)| < 1$ for almost all $x \in [r-a, r+a]$, and the initial guess x_0 lies in this interval.

In Example 1, $g(x) = \cos x$, $g'(x) = \sin x$, $r = 0.7391$,

$$\left|g'(r)\right| = |\sin(0.7391)| < 1. \qquad \text{OK, convergence.}$$

In Example 2, we have

$$\begin{aligned} g(x) &= e^{-2x}(x-1) + x, \\ g'(x) &= -2e^{-2x}(x-1) + x^{-2x} + 1 \end{aligned}$$

With $r = 1$, we have

$$\left|g'(r\right| = e^{-2} + 1 > 1$$

Divergence.

Pseudo code:

```
r=fixedpoint('g', x,tol,nmax}
  r=g(r);
  nit=1;
  while (abs(r-g(r))>tol and nit < nmax) do
    r=g(r);
    nit=nit+1;
  end
```

How to compute the error?
Assume $|g'(x)| \le m < 1$ in $[r-a, r+a]$.
We have $|e_{k+1}| \le m\,|e_k|$.
This gives

$$|e_1| \le m\,|e_0|\,, \quad |e_2| \le m\,|e_1| \le m^2\,|e_0|\,, \quad \cdots \quad |e_k| \le m^k\,|e_0|$$

We also have

$$|e_0| = |r - x_0| = |r - x_1 + x_1 - x_0| \le |e_1| + |x_1 - x_0| \le m\,|e_0| + |x_1 - x_0|$$

then

$$|e_0| \le \frac{1}{1-m}\,|x_1 - x_0|\,, \qquad \text{(can be computed)}$$

Put together

$$|e_k| \le \frac{m^k}{1-m}\,|x_1 - x_0|\,.$$

If the error tolerance is ε, then

$$\frac{m^k}{1-m}\,|x_1 - x_0| \le \varepsilon, \quad \Rightarrow \quad m^k \le \frac{\varepsilon(1-m)}{|x_1 - x_0|} \quad \Rightarrow \quad k \ge \frac{\ln(\varepsilon(1-m)) - \ln|x_1 - x_0|}{\ln m}$$

which give the maximum number of iterations needed to achieve an error $\le \varepsilon$.

Example $\cos x - x = 0$, so

$$x = g(x) = \cos x, \qquad g'(x) = -\sin x$$

Choose $x_0 = 1$. We know $r \approx 0.74$. We see that the iteration happens between $x = 0$ and $x = 1$. For $x \in [0,1]$, we have

$$\left|g'(x)\right| \le \sin 1 = 0.8415 = m$$

And $x_1 = \cos x_0 = \cos 1 = 0.5403$. Then, to achieve an error $\le \varepsilon = 10^{-5}$, the maximum # iterations needed is

$$k \ge \frac{\ln(\varepsilon(1-m)) - \ln|x_1 - x_0|}{\ln m} \approx 73.$$

Of course that is the worst situation. Give it a try and you will find that $k = 25$ is enough.

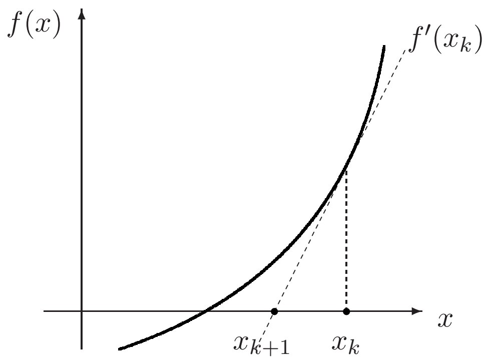

Figure 5.1: Newton's method: linearize $f(x)$ at x_k.

5.4 Newton's method

Goal: Given $f(x)$, find a root r s.t. $f(r) = 0$.

Choose an initial guess x_0.

Once you have x_k, the next approximation x_{k+1} is determined by treating $f(x)$ as a linear function at x_k. See Figure 5.1.

We have

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

which gives

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

Connection with fixed point iterations:

$$f(x) = 0, \quad \Rightarrow \quad b(x)f(x) = 0, \quad \Rightarrow \quad x = x - b(x)f(x) = g(x)$$

Here the function $b(x)$ is chosen in such a way, to get fastest possible convergence. We have

$$g'(x) = 1 - b'(x)f(x) - b(x)f'(x)$$

Let r be the root, such that $f(r) = 0$, and $r = g(r)$. We have

$$g'(r) = 1 - b(r)f'(r), \qquad \text{smallest possible:} \quad \left|g'(r)\right| = 0.$$

Choose now

$$1 - b(x)f'(x) = 0, \quad \Rightarrow \quad b(x) = \frac{1}{f'(x)}$$

we get a fixed point iteration for

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

Convergence analysis. Let r be the root so $f(r) = 0$ and $r = g(r)$. Define error:

$$e_{k+1} = |x_{k+1} - r| = |g(x_k) - g(r)|$$

Taylor expansion for $g(x_k)$ at r:

$$g(x_k) = g(r) + (x_k - r)g'(r) + \frac{1}{2}(x_k - r)^2 g''(\xi), \quad \xi \in (x_k, r)$$

Since $g'(r) = 0$, we have

$$g(x_k) = g(r) + \frac{1}{2}(x_k - r)^2 g''(\xi)$$

Back to the error, we now have

$$e_{k+1} = \frac{1}{2}(x_k - r)^2 \left|g''(\xi)\right| = \frac{1}{2}e_k^2 \left|g''(\xi)\right|$$

Write again $m = \max_x |g''(\xi)|$, we have

$$e_{k+1} \le m\, e_k^2$$

This is called *Quadratic convergence.* Guaranteed convergence if e_0 is small enough! (m can be big, it would effect the convergence!)

Proof for the convergence: (can drop this) We have

$$e_1 \le m\, e_0 e_0$$

If e_0 is small enough, such that $m\, e_0 < 1$, then $e_1 < e_0$.

Then, this means $m\, e_1 < m e_0 < 1$, and so

$$e_2 \le m\, e_1 e_1 < e_1, \quad \Rightarrow \quad m\, e_2 < m e_1 < 1$$

Continue like this, we conclude that $e_{k+1} < e_k$ for all k, i.e., error is strictly decreasing after each iteration. $\Rightarrow$ convergence.

Example Find an numerical method to compute $\sqrt{a}$ using only $+, -, *, /$ arithmetic operations. Test it for $a = 3$.

Answer. It's easy to see that $\sqrt{a}$ is a root for $f(x) = x^2 - a$.

Newton's method gives

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^2 - a}{2x_k} = \frac{x_k}{2} + \frac{a}{2x_k}$$

Test it on $a = 3$: Choose $x_0 = 1.7$.

	error
$x_0 = 1.7$	7.2×10^{-2}
$x_1 = 1.7324$	3.0×10^{-4}
$x_2 = 1.7321$	2.6×10^{-8}
$x_3 = 1.7321$	4.4×10^{-16}

Note the extremely fast convergence. Usually, if the initial guess is good (i.e., close to r), usually a couple of iterations are enough to get an very accurate approximation.

Stop criteria: can be any combination of the following:

- $|x_k - x_{k-1}| \le \varepsilon$
- $|f(x_k)| \le \varepsilon$
- max number of iterations reached.

Sample Code:

```
r=newton('f','df',x,nmax,tol)
n=0; dx=f(x)/df(x);
while (dx > tol) and (f(x) > tol) and (n<nmax) do
  n=n+1;
  x=x-dx;
  dx=f(x)/df(x);
end
r=x-dx;
```

5.5 Secant method

If $f(x)$ is complicated, $f'(x)$ might not be available.
Solution for this situation: using approximation for f', i.e.,

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

This is *secant method*:

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

Advantages include

- No computation of f';
- One $f(x)$ computation each step;
- Also rapid convergence.

A bit on convergence: One can show that

$$e_{k+1} \le C e_k^{\alpha}, \qquad \alpha = \frac{1}{2}(1+\sqrt{5}) \approx 1.62$$

This is called *super linear convergence.* $(1 < \alpha < 2)$

It converges for all function f if x_0 and x_1 are close to the root r.

Example Use secant method for computing $\sqrt{a}$.

Answer. The iteration now becomes

$$x_{k+1} = x_k - \frac{(x_k^2 - a)(x_k - x_{k-1})}{(x_k^2 - a) - (x_{k-1}^2 - a)} = x_k - \frac{x_k^2 - a}{x_k + x_{k+1}}$$

Test with $a = 3$, with initial data $x_0 = 1.65$, $x_1 = 1.7$.

	error
$x_1 = 1.7$	7.2×10^{-2}
$x_2 = 1.7328$	7.9×10^{-4}
$x_3 = 1.7320$	7.3×10^{-6}
$x_4 = 1.7321$	1.7×10^{-9}
$x_5 = 1.7321$	3.6×10^{-15}

It is a little but slower than Newton's method, but not much.

5.6 System of non-linear equations

Consider the system

$$\mathbf{F}(\vec{x}) = 0, \quad \mathbf{F} = (f_1, f_2, \cdots, f_n)^t, \quad \vec{x} = (x_1, x_2, \cdots, x_n)^t$$

write it in detail:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) & = & 0 \\ f_2(x_1, x_2, \cdots, x_n) & = & 0 \\ & \vdots & \\ f_n(x_1, x_2, \cdots, x_n) & = & 0 \end{cases}$$

We use fixed point iteration. Same idea: choose $\vec{x}_0$. Rewrite as

$$\vec{x} = \mathbf{G}(\vec{x})$$

The iteration is simply:

$$\vec{x}_{k+1} = \mathbf{G}(\vec{x}_k).$$

Newton's mathod:

$$\vec{x}_{k+1} = \vec{x}_k - D_f(\vec{x}_k)^{-1} \cdot \mathbf{F}(\vec{x}_k)$$

where $D_f(\vec{x}_k)$ is a Jacobian matrix of f

$$D_f = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{pmatrix}$$

and $D_f(\vec{x}_k)^{-1}$ is the inverse matrix of $D_f(\vec{x}_k)$.

Chapter 6

Direct methods for linear systems

6.1 Introduction

The problem:

$$(A): \quad \left\{ \begin{array}{lcll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = & b_1 & \qquad (1) \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & = & b_2 & \qquad (2) \\ & \vdots & & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n & = & b_n & \qquad (n) \end{array} \right.$$

We have n equations, n unknowns, can be solved for x_i if a_{ij} are "good".
In compact form, for equation i:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, \cdots, n.$$

Or in matrix-vector form:

$$A\vec{x} = \vec{b},$$

where $A \in I\!R^{n\times n}$, $\quad \vec{x} \in I\!R^n$, $\quad \vec{b} \in I\!R^n$

$$A = \{a_{ij}\} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Our goal: Solve for $\vec{x}$. — We will spend 3 weeks on it.
Methods and topics:

- Different types of matrix A:

1. Full matrix
2. Large sparse system
3. Tri-diagonal or banded systems
4. regularity and condition number

- Methods

  A. Direct solvers (exact solutions): slow, for small systems
    - $*$ Gaussian elimination, with or without pivoting
    - $*$ LU factorization

  B. Iterative solvers (approximate solutions, for large sparse systems)
    - $*$ more interesting for this course
    - $*$ details later...

6.2 Gaussian elimination, simplest version

Consider system (A). The basic Gaussian elimination takes two steps:
Step 1: Make an upper triangular system – forward elimination.

```
for $k = 1, 2, 3, \cdots, n-1$
$(j) \leftarrow (j) - (k) \times \frac{a_{jk}}{a_{kk}}, \quad j = k+1, k+2, \cdots, n$
```

You will make lots of zeros, and the system becomes:

$$(B): \quad \left\{ \begin{array}{rcll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = & b_1 & (1) \\ a_{22}x_2 + \cdots + a_{2n}x_n & = & b_2 & (2) \\ & \vdots & & \\ a_{nn}x_n & = & b_n & (n) \end{array} \right.$$

Note, here the a_{ij} and b_i are different from those in (A).
Step 2: Backward substitution – you get the solutions.

$$\begin{aligned} x_n &= \frac{b_n}{a_{nn}} \\ x_i &= \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right), \qquad i = n-1, n-2, \cdots, 1. \end{aligned}$$

Potential problem: In step 1, if some a_{kk} is very close to or equal to 0, then you are in trouble.

Example 1. Solve

$$\begin{cases} x_1 + x_2 + x_3 &= 1 \qquad (1) \\ 2x_1 + 4x_2 + 4x_3 &= 2 \qquad (2) \\ 3x_1 + 11x_2 + 14x_3 &= 6 \qquad (3) \end{cases}$$

Forward elimination:

$$\begin{array}{rrcll} (1)*(-2)+(2): & 2x_2 + 2x_3 &=& 0 & (2') \\ (1)*(-3)+(3): & 8x_2 + 11x_3 &=& 3 & (3') \\ (2')*(-4)+(3'): & 3x_3 = 3 \quad (3'') \end{array}$$

Your system becomes

$$\begin{cases} x_1 + x_2 + x_3 &= 1 \\ 2x_2 + 2x_3 &= 0 \\ 3x_3 &= 3 \end{cases}$$

Backward substitution:

$$\begin{aligned} x_3 &= 1 \\ x_2 &= \frac{1}{2}(0 - 2x_3) = -1 \\ x_1 &= 1 - x_2 - x_3 = 1 \end{aligned}$$

It works fine here, but not always.

6.3 Gaussian Elimination with scaled partial pivoting

First, an example where things go wrong with Gaussian elimination.

Example 2. Solve the system with 3 significant digits.

$$\begin{cases} 0.001x_1 \quad -x_2 &= -1 \qquad (1) \\ x_1 \quad 2x_2 &= 3 \qquad (2) \end{cases}$$

Answer. Write it with 3 significant digits

$$\begin{cases} 0.00100x_1 \quad -1.00x_2 &= -1.00 \qquad (1) \\ 1.00x_1 \quad 2.00x_2 &= 3.00 \qquad (2) \end{cases}$$

Now, $(1)*(-1000)+(2)$ gives

$$\begin{aligned} & (1000+2)x_2 = 1000+3 \\ \Rightarrow \quad & 1.00 \cdot 10^3 x_2 = 1.00 \cdot 10^3 \\ \Rightarrow \quad & x_2 = 1.00 \end{aligned}$$

Put this back into (1) and solve for x_1:

$$x_1 = \frac{1}{0.001}(-1.00 + 1.00x_2) = \frac{1}{0.001} \cdot 0 \ 0$$

.

Note that x_1 is wrong!!

What is the problem? We see that 0.001 is a very small number!

One way around this difficulty: Change the order of two equations.

$$\begin{cases} 1.00x_1 & 2.00x_2 & = & 3.00 & (1) \\ 0.00100x_1 & -1.00x_2 & = & -1.00 & (2) \end{cases}$$

Now run the whole procedure again: $(1) * (-0.001) + (2)$ will give us $x_2 = 1.00$.

Set it back in (1):

$$x_1 = 3.00 - 2.00x_2 = 1.00$$

Solution now is correct for 3 digits.

Conclusion: The order of equations can be important!

Consider

$$\begin{cases} a_{11}x_1 + a_{12}x_2 & = & b_1 \\ a_{21}x_1 + a_{22}x_2 & = & b_2 \end{cases}$$

Assume that we have computed $\tilde{x}_2 = x_2 + \varepsilon_2$ where ε_2 is the error (machine error, round off error etc).

We then compute x_1 with this $\tilde{x}_2$:

$$\begin{aligned} \tilde{x}_1 &= \frac{1}{a_{11}}(b_1 - a_{12}\tilde{x}_2) \\ &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{12}\varepsilon_2) \\ &= \underbrace{\frac{1}{a_{11}}(b_1 - a_{12}x_2)}_{x_1} - \underbrace{\frac{a_{12}}{a_{11}}\varepsilon_2}_{\varepsilon_1} \\ &= \quad x_1 \quad - \quad \varepsilon_1 \end{aligned}$$

Note that $\varepsilon_1 = \frac{a_{12}}{a_{11}}\varepsilon_2$. Error in x_2 propagates with a factor of $\frac{a_{12}}{a_{11}}$.

For best results, we wish to have $|a_{11}|$ as big as possible.

Scaled Partial Pivoting. Idea: use $\max_{k \le i \le n} |a_{ik}|$ for a_{ii}.

Procedure:

1. Compute a scaling vector

$$\vec{s} = [s_1, s_2, \cdots, s_n], \quad \text{where} \quad s_i = \max_{1 \le j \le n} |a_{ij}|$$

   Keep this $\vec{s}$ for the rest of the computation.

2. Find the index k s.t.
$$\left|\frac{a_{k1}}{s_k}\right| \geq \left|\frac{a_{i1}}{s_i}\right|, \quad i = 1, \cdots, n$$
Exchange eq (k) and (1), and do 1 step of elimination. You get
$$\left\{\begin{array}{rcll} a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n &=& b_k & (1) \\ a_{22}x_2 + \cdots + a_{2n}x_n &=& b_2 & (2) \\ &\vdots& & \\ a_{12}x_2 + \cdots + a_{1n}x_n &=& b_1 & (k) \\ &\vdots& & \\ a_{n2}x_2 + \cdots + a_{nn}x_n &=& b_n & (n) \end{array}\right.$$
3. Repeat (2) for the remaining $(n-1) \times (n-1)$ system, and so on.

Example Solve the system using scaled partial pivoting.
$$\left\{\begin{array}{rcll} x_1 + 2x_2 + x_3 &=& 3 & (1) \\ 3x_1 + 4x_2 + 0x_3 &=& 3 & (2) \\ 2x_1 + 10x_2 + 4x_3 &=& 10 & (3) \end{array}\right.$$

Answer. We follow the steps.

1. Get $\vec{s}$.
$$\vec{s} = [2, 4, 10]$$
2. We have
$$\frac{a_{11}}{s_1} = \frac{1}{2}, \quad \frac{a_{21}}{s_2} = \frac{3}{4}, \quad \frac{a_{31}}{s_3} = \frac{2}{10}, \quad \Rightarrow \quad k = 2$$
Exchange eq (1) and (2), and do one step of elimination
$$\left\{\begin{array}{rcll} 3x_1 + 4x_2 + 0x_3 &=& 3 & (2) \\ \frac{2}{3}x_2 + x_3 &=& 2 & (1') \quad = (1) + (2) * (\frac{1}{3}) \\ \frac{22}{3}x_2 + 4x_3 &=& 8 & (3') \quad = (3) + (2) * (-\frac{2}{3}) \end{array}\right.$$
3. For the 2×2 system,
$$\frac{\bar{a}_{12}}{s_1} = \frac{2/3}{2} = \frac{1}{3}, \qquad \frac{\bar{a}_{32}}{s_3} = \frac{22/3}{10} = \frac{22}{30} \quad \Rightarrow \quad k = 3.$$
Exchange (3') and (1') and do one step of elimination
$$\left\{\begin{array}{rcll} 3x_1 + 4x_2 + 0x_3 &=& 3 & (2) \\ \frac{22}{3}x_2 + 4x_3 &=& 8 & (3') \\ \frac{7}{11}x_3 &=& \frac{14}{11} & (1'') \quad = (1') + (3') * (-\frac{1}{11}) \end{array}\right.$$

4. Backward substitution gives

$$x_3 = 2, \qquad x_2 = 0, \qquad x_1 = 1.$$

In MAtlab, to solve $Ax = b$, one can use:

```
> x= A\b;
```

6.4 LU-Factorization

Without pivoting: One can write $A = LU$ where

L: lower triangular matrix with unit diagnal

U: upper triangular matrix

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1,(n-1)} & u_{1n} \\ 0 & u_{22} & \cdots & u_{2,(n-1)} & u_{2n} \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & u_{(n-1),(n-1)} & u_{(n-1),n} \\ 0 & 0 & \cdots & 0 & u_{nn} \end{pmatrix}$$

Theorem *If $Ax = b$ can be solved by Gaussian elimination without pivoting, then we can write $A = LU$ uniquely.*

Use this to solve $Ax = b$: Let $y = Ux$, then we have

$$\begin{cases} U\,x &= y \\ L\,y &= b \end{cases}$$

Two triangular system. We first solve y (by forward substitution), then solve x (by backward substitution).

With pivoting:

$$LU = PA$$

where P is the pivoting matrix.

Used in Matlab:

```
> [L,U]=lu(A);
> y = L \ b;
> x = U \ y;
```

+ transparences.
Work amount for direct solvers for $A \in \mathbb{R}^{n\times n}$: operation count
flop: one float number operation (+, -, *, /)
Elimination: $\frac{1}{3}(n^3 - n)$ flops
Backward substitution: $\frac{1}{2}(n^2 - n)$ flops
Total work amount is about $\frac{1}{3}n^3$ for large n.
This is $\underline{\text{very slow}}$ for large n. We will need something more efficient.

6.5 Review of linear algebra

Consider a square matrix $A = \{a_{ij}\}$.

Diagonal dominant system. If

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, \cdots, n$$

then A is called *strictly diagonal dominant*. And A has the following properties:

- A is regular, invertible, A^{-1} exists, and $Ax = b$ has a unique solution.
- $Ax = b$ can be solved by Gaussian Elimination without pivoting.

One such example: the system from natural cubic spline.

Vector and matrix norms:
A norm: measures the "size" of the vector and matrix.

General norm properties: $x \in \mathbb{R}^n$ or $x \in \mathbb{R}^{n\times n}$. Then, $\|x\|$ satisfies

1. $\|x\| \geq 0$, equal if and only if $x = 0$;
2. $\|ax\| = |a| \cdot \|x\|$, $\quad a$: is a constant;
3. $\|x + y\| \leq \|x\| + \|y\|$, triangle inequality.

Examples of vector norms: $x \in I\!R^n$

1. $\|x\|_1 = \sum_{i=1}^{n} |x_i|$, l_1-norm
2. $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$, l_2-norm
3. $\|x\|_\infty = \max_{1 \le i \le n} |x_i|$, l_∞-norm

Matrix norm related to the corresponding vector norm, $A \in I\!R^{n\times n}$

$$\|A\| = \max_{\vec{x}\neq 0} \frac{\|Ax\|}{\|x\|}$$

Obviously we have

$$\|A\| \ge \frac{\|Ax\|}{\|x\|} \quad \Rightarrow \quad \|Ax\| \le \|A\| \cdot \|x\|$$

In addition we have

$$\|I\| = 1, \qquad \|AB\| \le \|A\| \cdot \|B\| .$$

Examples of matrix norms:

$$\begin{aligned} l_1 - \text{norm} \quad &: \quad \|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}| \\ l_2 - \text{norm} \quad &: \quad \|A\|_2 = \max_i |\lambda_i| , \qquad \lambda_i : \text{ eigenvalues of } A \\ l_\infty - \text{norm} \quad &: \quad \|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}| \end{aligned}$$

Eigenvalues λ_i for A:

$$Av = \lambda v, \qquad \lambda : \text{ eigenvalue}, \qquad v : \text{ eigenvector}$$

$$(A - \lambda I)v = 0, \quad \Rightarrow \quad \det(A - \lambda I) = 0 : \quad \text{polynomial of degree } n$$

In general, one gets n eigenvalues counted multiplicity.
Property:

$$\lambda_i(A^{-1}) = \frac{1}{\lambda_i(A)}$$

This implies

$$\left\|A^{-1}\right\|_2 = \max_i \left|\lambda_i(A^{-1})\right| = \max_i \frac{1}{|\lambda_i(A^{-1})|} = \frac{1}{\min_i |\lambda_i(A^{-1})|}$$

Condition number of a matrix A: Want to solve $Ax = b$. Put some perturbation:

$$A\bar{x} = b + p$$

Relative error in perturbation:

$$e_b = \frac{\|p\|}{\|b\|}$$

Relative change in solution is

$$e_x = \frac{\|\bar{x} - x\|}{\|x\|}$$

We wish to find a relation between e_x and e_b. We have

$$A(\bar{x} - x) = p, \quad \Rightarrow \quad \bar{x} - x = A^{-1}p$$

so

$$e_x = \frac{\|\bar{x} - x\|}{\|x\|} = \frac{\left\|A^{-1}p\right\|}{\|x\|} \leq \frac{\left\|A^{-1}\right\| \cdot \|p\|}{\|x\|}.$$

By using the following

$$Ax = b \quad \Rightarrow \quad \|AX\| = \|b\| \quad \Rightarrow \quad \|A\| \, \|x\| \geq \|b\| \quad \Rightarrow \quad \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|}$$

we get

$$e_x \leq \left\|A^{-1}\right\| \cdot \|p\| \frac{\|A\|}{\|b\|} = \|A\| \left\|A^{-1}\right\| e_b = \kappa(A) e_b,$$

Here

$$\kappa(A) = \|A\| \left\|A^{-1}\right\| = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$$

is called *the condition number* of A. Error in b propagates with a factor of $\kappa(A)$ into the solution.

If $\kappa(A)$ is very large, $Ax = b$ is very sensitive to perturbation, therefore difficult to solve. We call this *ill-conditioned system*.

In Matlab: `cond(A)` gives the condition number of A.

6.6 Tridiagonal and banded systems

Tridiagonal system is when A is a tridiagonal matrix:

$$A = \begin{pmatrix} d_1 & c_1 & 0 & \cdots & 0 & 0 & 0 \\ a_1 & d_2 & c_2 & \cdots & 0 & 0 & 0 \\ 0 & a_2 & d_3 & \ddots & 0 & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & d_{n-2} & c_{n-2} & 0 \\ 0 & 0 & 0 & \cdots & a_{n-2} & d_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1} & d_n \end{pmatrix}$$

or we can write

$$A = \text{tridiag}(a_i, d_i, c_i).$$

This can be solved very efficiently:

- Elimination (without pivoting)

```
for $i = 2, 3, \cdots, n$
    $d_i \leftarrow d_i - \frac{a_{i-1}}{d_{i-1}} c_{i-1}$
    $b_i \leftarrow b_i - \frac{a_{i-1}}{d_{i-1}} b_{i-1}$
end
```

  Now the A matrix looks like

$$A = \begin{pmatrix} d_1 & c_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & d_2 & c_2 & \cdots & 0 & 0 & 0 \\ 0 & 0 & d_3 & \ddots & 0 & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & d_{n-2} & c_{n-2} & 0 \\ 0 & 0 & 0 & \cdots & 0 & d_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 & d_n \end{pmatrix}$$

- Backward substitution

```
$x_n \leftarrow b_n / d_n$
for $i = n-1, n-2, \cdots, 1$
    $x_i \leftarrow \frac{1}{d_i}(b_i - c_i x_{i+1})$
end
```

Amount of work: $\mathcal{O}(Cn)$ where C is a fixed small constant.

Penta-diagonal system. We can write

$$A = \text{pentadiag}(e_i, a_i, d_i, c_i, f_i)$$

which means

$$A = \begin{pmatrix} d_1 & c_1 & f_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_1 & d_2 & c_2 & f_2 & 0 & \cdots & 0 & 0 & 0 \\ e_1 & a_2 & d_3 & c_3 & f_3 & \cdots & 0 & 0 & 0 \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots & \vdots \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & 0 & 0 & & & \cdots & d_{n-2} & c_{n-2} & f_{n-2} \\ 0 & 0 & 0 & 0 & & \cdots & a_{n-2} & d_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \cdots & e_{n-2} & a_{n-1} & d_n \end{pmatrix}$$

Band matrix: This is a more general description:

$$A = \begin{pmatrix} d_1 & \ddots & \ddots & \ddots & & & 0 \\ \ddots & d_2 & \ddots & \ddots & \ddots & & \\ \ddots & \ddots & d_3 & \ddots & \ddots & \ddots & \\ \ddots & \ddots & \ddots & d_4 & \ddots & \ddots & \ddots \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & & & \ddots & \ddots & \ddots & d_n \end{pmatrix}$$
$$|\leftarrow k \rightarrow|$$

Here $k \geq 0$ is the *band width*, meaning $a_{ij} = 0$ for all $|i - j| > k$.
We have

- diagonal matrix: $k = 0$,
- tridiagonal matrix: $k = 1$,
- pentadiagonal matrix: $k = 2$.

Gaussian elimination is efficient if $k << n$.

Some Matlab commands:

```
[L,U] = lu(A);    % LU-factorization
norm(x);          % vector norm
eig(A);           % eigenvalue/eigen vector of a matrix
cond(A);          % condition number of A
```

Chapter 7

Iterative solvers for linear systems

7.1 General introduction

Want to solve $Ax = b$, where $A \in I\!R^{n \times n}$, n is very large, and A is sparse.
Two classes of solvers:

- Direct solvers (Gaussian Eliminatio, LU factorizatio etc): very slow.
- Iterative methods: finding only approximation to solutions. Useful for large sparse systems, usually coming from discretization of differential equations.

Properties of such systems

- Large, n is very big, for example $n = \mathcal{O}(10^6)$.
- A is sparse, with a large percent of 0 entries.
- A is structured. (meaning: the product Ax can be computed efficiently)

Two classes of iterative methods

1. Fixed-point iterations
    - Jacobi
    - Gauss-Seidal
    - SOR
2. Krylove technics (not covered in this course, though widely used)

7.2 Jacobi iterations

Want to solve $Ax = b$. Write it out

$$\left\{\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &=& b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &=& b_2 \\ &\vdots& \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &=& b_n \end{array}\right.$$

Rewrite it in another way:

$$\left\{\begin{array}{rcl} x_1 &=& \frac{1}{a_{11}}\left(b_1 - a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n\right) \\ x_2 &=& \frac{1}{a_{22}}\left(b_2 - a_{21}x_1 + a_{22}x_2 + \cdots a_{2n}x_n\right) \\ &\vdots& \\ x_n &=& \frac{1}{a_{nn}}\left(b_2 - a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nn}x_n\right) \end{array}\right.$$

or in a compact for:

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j\right), \qquad i = 1, 2, \cdots, n$$

This gives the Jacobi iterations:

- Choose a start point, $x^0 = (x_1^0, x_2^0, \cdots, x_n^0)^t$. For example, one may choose $x_i^0 = 1$ for all i, or $x_i = b_i/a_{ii}$.
- for $k = 0, 1, 2, \cdots$ until stop criteria

```
for $i = 1, 2, \cdots, n$
    $x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^k\right)$
  end
end
```

Stop Criteria could be any combinations of the following

- x^k close enough to x^{k-1}, for example $\left\|x^k - x^{k-1}\right\| \leq \varepsilon$ for certain vector norms.
- Residual $r^k = Ax^k - b$ is small: for example $\left\|r^k\right\| \leq \varepsilon$.
- or others...

About the algorithm:

- Must make 2 vectors for the computation, x^k and x^{k+1}.
- Great for parallel computing.

Example 1. Solve the following system with Jacobi iterations.

$$\left\{\begin{array}{rcl} 2x_1 - x_2 & = & 0 \\ -x_1 + 2x_2 - x_3 & = & 1 \\ -x_2 + 2x_3 & = & 2 \end{array}\right.$$

given the exact solution $x = (1, 2, 2)^t$.

Answer. Choose x^0 by $x_i^0 = b_i/a_{ii}$:

$$x^0 = (1,\ 1/2,\ 1)^t$$

The iteration is

$$\left\{\begin{array}{rcl} x_1^{k+1} & = & \frac{1}{2}x_2^k \\ x_2^{k+1} & = & \frac{1}{2}(1 + x_1^k + x_3^k) \\ x_3^{k+1} & = & \frac{1}{2}(2 + x_2^k) \end{array}\right.$$

We run a couple of iterations, and get

$$\begin{array}{rcl} x^1 & = & (0.25,\ 1,\ 1.25)^t \\ x^2 & = & (0.5,\ 1.25,\ 1.5)^t \\ x^3 & = & (0.625,\ 1.5,\ 1.625)^t \end{array}$$

Observations:

- Looks like it is converging. Need to run more steps to be sure.
- Rather slow convergence rate.

7.3 Gauss-Seidal iterations

An improved version: observe that in Jacobi iteration, we write

$$\begin{aligned} x_i^{k+1} &= \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^k\right) \\ &= \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^k\right) \end{aligned}$$

In the first summation term, all x_j^k are already computed for step $k+1$.

We will replace all these x_j^k with x_j^{k+1}. This gives:

Gauss-Seidal iterations: use the latest computed values of x_i.

```
for $k = 0, 1, 2, \cdots$, until stop criteria
    for $i = 1, 2, \cdots, n$
        $x_i^{k+1} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right)$
    end
end
```

About the algorithm:

- Need only one vector for both x^k and x^{k+1}, saves memory space.
- Not good for parallel computing.

Example 2. Try it on the same Example 1, with $x^0 = (0, 0.5, 1)^t$. The iteration now is:

$$\begin{cases} x_1^{k+1} &= \frac{1}{2} x_2^k \\ x_2^{k+1} &= \frac{1}{2}(1 + x_1^{k+1} + x_3^k) \\ x_3^{k+1} &= \frac{1}{2}(2 + x_2^{k+1}) \end{cases}$$

We run a couple of iterations:

$$\begin{aligned} x^1 &= (0.25,\ 1.125,\ 1.5625)^t \\ x^2 &= (0.5625,\ 1.5625,\ 1.7813)^t \end{aligned}$$

Observation: Converges a bit faster than Jacobi iterations.

7.4 SOR

SOR (Successive Over Relaxation) is a more general iterative method. It is based on Gauss-Seidal.

$$x_j^{k+1} = (1-w) x_i^k + w \cdot \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right)$$

Note the second term is the Gauss-Seidal iteration multiplied with w.

w: relaxation parameter.

Usual value: $0 < w < 2$ (for convergence reason)

- $w = 1$: Gauss-Seidal
- $0 < w < 1$: under relaxation
- $1 < w < 2$: over relaxation

Example Try this on the same example with $w = 1.2$. General iteration is now:

$$\begin{cases} x_1^{k+1} &= & -0.2x_1^k + 0.6x_2^k \\ x_2^{k+1} &= & -0.2x_2^k + 0.6 * (1 + x_1^{k+1} + x_3^k) \\ x_3^{k+1} &= & -0.2x_3^k + 0.6 * (2 + x_2^{k+1}) \end{cases}$$

With $x^0 = (0,\ 0.5,\ 1)^t$, we get

$$\begin{aligned} x^1 &= (0.3, 1.28, 1.708)^t \\ x_2 &= (0.708, 1.8290, 1.9442)^t \end{aligned}$$

Recall the exact solution $x = (1, 2, 2)^t$.
Observation: faster convergence than both Jacobi and G-S.

7.5 Writing all methods in matrix-vector form

Want to solve $Ax = b$.
Splitting of the matrix A:

$$A = L + D + U$$

where

- L is the lower triangular part of A:

$$L = \{l_{ij}\}, \qquad l_{ij} = \begin{cases} a_{ij}, & i > j \\ 0 & i \leq j \end{cases}$$

- D is the diagonal part of A:

$$D = \{d_{ij}\}, \qquad d_{ij} = \begin{cases} a_{ij} = a_{ii}, & i = j \\ 0 & i \neq j \end{cases}$$

- U is the upper triangular part of A:

$$U = \{u_{ij}\}, \qquad u_{ij} = \begin{cases} a_{ij}, & i < j \\ 0 & i \geq j \end{cases}$$

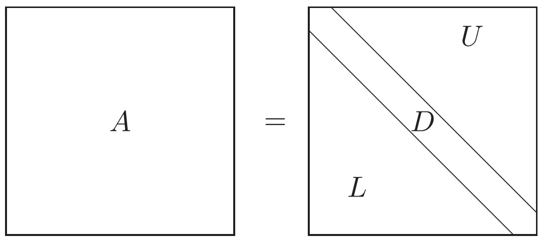

Figure 7.1: Splitting of A.

See the graph in Figure 7.1 for an illustration.

Now we have

$$Ax = (L + D + U)x = Lx + Dx + Ux = b$$

Jacobi iterations:

$$Dx^{k+1} = b - Lx^k - Ux^k$$

so

$$x^{k+1} = D^{-1}b - D^{-1}(L + U)x^k = y_J + M_J x^k$$

where

$$y_J = D^{-1}b, \qquad M_J = -D^{-1}(L + U).$$

Gauss-Seidal:

$$Dx^{k+1} + Lx^{k+1} = b - Ux^k$$

so

$$x^{k+1} = (D + L)^{-1}b - (D + L)^{-1}Ux^k = y_{GS} + M_{GS}x^k$$

where

$$y_{GS} = (D + L)^{-1}b, \qquad M_{GS} = -(D + L)^{-1}U.$$

SOR:

$$x^{k+1} = (1 - w)x^k + wD^{-1}(b - Lx^{k+1} - Ux^k)$$

$$\Rightarrow \quad Dx^{k+1} = (1 - w)Dx^k + wb - wLx^{k+1} - wUx^k$$

$$\Rightarrow \quad (D + wL)x^{k+1} = wb + [(1 - w)D - wU]x^k$$

so

$$x^{k+1} = (D + wL)^{-1}b + (D + wL)^{-1}[(1 - w)D - wU]x^k = y_{SOR} + M_{SOR}x^k$$

where

$$y_{SOR} = (D + wL)^{-1}b, \qquad M_{SOR} = (D + wL)^{-1}[(1 - w)D - wU].$$

7.6 Analysis for errors and convergence

Consider an iteration of the form

$$x^{k+1} = y + Mx^k$$

Assume s is the solution s.t. $As = b$.
This means s is a fixed point of the iteration: $s = y + Ms$.
Define the error:

$$e^k = x^k - s$$

We have

$$e^{k+1} = x^{k+1} - s = y + Mx^k - (y + Ms) = M(x^k - s) = Me^k.$$

This gives the propagation of error:

$$e^{k+1} = M\,e^k.$$

Take the norm on both sides:

$$\left\| e^{k+1} \right\| = \left\| Me^k \right\| \leq \|M\| \cdot \left\| e^k \right\|$$

This implies:

$$\left\| e^k \right\| \leq M^k \left\| e^0 \right\|, \qquad e^0 = x^0 - s.$$

Theorem *If $\|M\| < 1$ for some norm $\|\cdot\|$, then the iterations converge.*

NB! Convergence only depends on the iteration matrix M.

Check our methods: $A = D + L + U$.

- Jacobi: $M = -D^{-1}(L + U)$, given by A;
- G-S: $M = -(D + L)^{-1}U$, given by A;
- SOR: $M = (D + wL)^{-1}[(1 - w)D - wU]$, can adjust w to get a smallest posible $\|M\|$. More flexible.

Example Let's check the same example we have been using. We have

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix},$$

so

$$L = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}, \qquad D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \qquad U = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

The iteration matrix for each method:

$$M_J = \begin{pmatrix} 0 & 0.5 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0 \end{pmatrix}, \qquad M_{GS} = \begin{pmatrix} 0 & 0.5 & 0 \\ 0 & 0.25 & 0.5 \\ 0 & 0.125 & 0.25 \end{pmatrix}, \qquad M_{SOR} = \begin{pmatrix} -0.2 & 0.6 & 0 \\ -0.12 & 0.16 & 0.6 \\ -0.072 & 0.096 & 0.16 \end{pmatrix}$$

We list their various norms:

M	l_1 norm	l_2 **norm**	l_∞ norm
Jacobi	1	**0.707**	1
G-S	0.875	**0.5**	0.75
SOR	0.856	**0.2**	0.88

The l_2 norm is the most significant one. We see now why SOR converges fastest.

Convergence Theorem. *If A is diagonal dominant, i.e.,*

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \qquad \forall i = 1, 2, \cdots, n.$$

Then, all three iteration methods converge for all initial choice of x^0.

NB! If A is not diagonal dominant, it might still converge, but there is no guarantee.

Chapter 8

The Method of Least Squares

8.1 Problem description

Given data set

x	x_0	x_1	x_2	$\cdots$	x_m
y	y_0	y_1	y_2	$\cdots$	y_M

Data come from observation (measured) or experiments.

These y_i's can have error (called "noise") in measuring or experimenting.

y has a relation with x from physical model: $y = y(x)$.

Then, our data is

$$y_k = y(x_k) + e_k$$

where e_k is error.

8.2 Linear regression and basic derivation

Example 1. If $y = ax + b$, this is called *linear regression*. Your lab data would not lie exact on a straight line (or your lab instructor will be very suspicious!). Our job now is to find a straight line that "best" fit our data. See Figure 8.1 for an illustration.

A more specific way of saying the same thing: Find a, b, such that when we use $y = ax+b$, the "error" becomes smallest possible.

How to measure error?

1. $\max\limits_k |y(x_k) - y_k|$ — l_∞ norm

2. $\sum\limits_{k=0}^{m} |y(x_k) - y_k|$ — l_1 norm

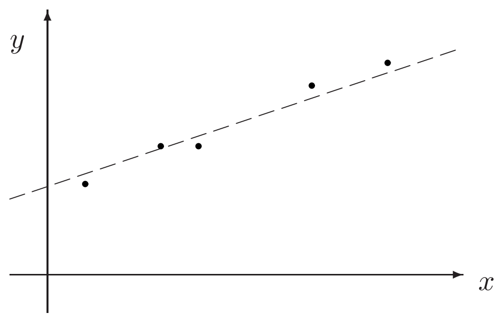

Figure 8.1: Linear regression.

3. $\sum_{k=0}^{m}[y(x_k)-y_k]^2$ — l_2 norm, used in Least Square Method. (LSM)

Our problem can now be stated as a minimization problem:

Find a and b such that the error function $\psi(a,b)$ defined as

$$\psi(a,b)=\sum_{k=0}^{m}(ax_k+b-y_k)^2$$

is minimized.

How to find a and b?

At the minimum of a function, we have

$$\frac{\partial\psi}{\partial a}=\frac{\partial\psi}{\partial b}=0$$

In detail:

$$\frac{\partial\psi}{\partial a}=0 \quad : \quad \sum_{k=0}^{m}2(ax_k+b-y_k)x_k=0, \qquad (I)$$

$$\frac{\partial\psi}{\partial b}=0 \quad : \quad \sum_{k=0}^{m}2(ax_k+b-y_k)=0, \qquad (II)$$

Solve $(I),(II)$ for (a,b). Rewrite it as a system

$$\begin{cases} \left(\sum_{k=0}^{m}x_k^2\right)\cdot a+\left(\sum_{k=0}^{m}x_k\right)\cdot b &= \sum_{k=0}^{m}x_k\cdot y_k \\ \left(\sum_{k=0}^{m}x_k\right)\cdot a+(m+1)\cdot b &= \sum_{k=0}^{m}y_k \end{cases}$$

These are called the normal equations.

Example 2. Consider the data set (from textbook, p.428)

T_k	0	10	20	30	40	80	90	95
S_k	68.0	67.1	66.4	65.6	64.6	61.8	61.0	60.0

where

S: surface tension in a liquid

T: temperature

From physics we know that they have a linear relation

$$S = aT + b$$

Use MLS to find the best fitting a, b.

Answer. We have $m = 7$, and

$$\begin{aligned}
\sum_{k=0}^{7} T_k^2 &= 0 + 10^2 + 20^2 + \cdots + 90^2 + 95^2 = 26525 \\
\sum_{k=0}^{7} T_k &= \cdots = 365 \\
\sum_{k=0}^{7} T_k S_k &= \cdots = 22685 \\
\sum_{k=0}^{7} S_k &= \cdots = 514.5
\end{aligned}$$

The normal equations are

$$\begin{cases} 26525\, a + 365\, b = 22685 \\ 365\, a + 8\, b = 514.5 \end{cases}$$

Solve it

$$a = -0.079930, \qquad b = 67.9593.$$

So

$$S(T) = -0.079930\, T + 67.9593.$$

8.3 LSM with parabola

Example 3. Given $m+1$ data $(x_k, y_k)_{k=0}^m$. Find $y(x) = ax^2 + bx + c$ that best fit our data.

Answer. Define the error (in least squares sense)

$$\psi(a,b,c) = \sum_{k=0}^{m} \left(ax_k^2 + bx_k + c - y_k\right)^2$$

At minimum, we have

$$\frac{\partial \psi}{\partial a} = \frac{\partial \psi}{\partial b} = \frac{\partial \psi}{\partial c} = 0$$

In detail:

$$\begin{aligned}
\frac{\partial \psi}{\partial a} = 0 \quad &: \qquad \sum_{k=0}^{m} 2\left(ax_k^2 + bx_k + c - y_k\right) \cdot x_k^2 = 0 \\
\frac{\partial \psi}{\partial b} = 0 \quad &: \qquad \sum_{k=0}^{m} 2\left(ax_k^2 + bx_k + c - y_k\right) \cdot x_k = 0 \\
\frac{\partial \psi}{\partial c} = 0 \quad &: \qquad \sum_{k=0}^{m} 2\left(ax_k^2 + bx_k + c - y_k\right) = 0
\end{aligned}$$

The normal equations:

$$\left\{\begin{array}{lcl}
\left(\sum_{k=0}^m x_k^4\right) \cdot a + \left(\sum_{k=0}^m x_k^3\right) \cdot b + \left(\sum_{k=0}^m x_k^2\right) \cdot c & = & \sum_{k=0}^m x_k^2 y_k \\
\left(\sum_{k=0}^m x_k^3\right) \cdot a + \left(\sum_{k=0}^m x_k^2\right) \cdot b + \left(\sum_{k=0}^m x_k\right) \cdot c & = & \sum_{k=0}^m x_k y_k \\
\left(\sum_{k=0}^m x_k^2\right) \cdot a + \left(\sum_{k=0}^m x_k\right) \cdot b + (m+1) \cdot c & = & \sum_{k=0}^m y_k
\end{array}\right.$$

We have 3 linear equations for 3 unknown. The system is symmetric.

Need to compute only:

$$\sum x_k^4, \quad \sum x_k^3, \quad \sum x_k^2, \quad \sum x_k, \quad \sum x_k^2 y_k, \quad \sum x_k y_k, \quad \sum y_k$$

8.4 LSM with non-polynomial

Example 4. Non-polynomial example. Given data set (x_k, y_k) for $k = 0, 1, \cdots, m$. Find a

$$y(x) = a \cdot f(x) + b \cdot g(x) + c \cdot h(x)$$

which best fit the data. This means we need to find (a, b, c).

Here $f(x), g(x), h(x)$ are given functions, for example

$$f(x) = e^x, \qquad g(x) = \ln(x), \qquad h(x) = \cos x,$$

but not restricted to these.

Define the error function:

$$\psi(a,b,c) = \sum_{k=0}^{m} (y(x_k) - y_k)^2 = \sum_{k=0}^{m} \left(a \cdot f(x_k) + b \cdot g(x_k) + c \cdot h(x_k) - y_k\right)^2$$

At minimum, we have

$$\begin{aligned}
\frac{\partial \psi}{\partial a} = 0 \quad &: \qquad \sum_{k=0}^{m} 2\Big[a \cdot f(x_k) + b \cdot g(x_k) + c \cdot h(x_k) - y_k\Big] \cdot f(x_k) = 0 \\
\frac{\partial \psi}{\partial b} = 0 \quad &: \qquad \sum_{k=0}^{m} 2\Big[a \cdot f(x_k) + b \cdot g(x_k) + c \cdot h(x_k) - y_k\Big] \cdot g(x_k) = 0 \\
\frac{\partial \psi}{\partial c} = 0 \quad &: \qquad \sum_{k=0}^{m} 2\Big[a \cdot f(x_k) + b \cdot g(x_k) + c \cdot h(x_k) - y_k\Big] \cdot h(x_k) = 0
\end{aligned}$$

The normal equations are

$$\left\{\begin{array}{lcl}
\left(\sum_{k=0}^{m} f(x_k)^2\right) a + \left(\sum_{k=0}^{m} f(x_k)g(x_k)\right) b + \left(\sum_{k=0}^{m} f(x_k)h(x_k)\right) c & = & \sum_{k=0}^{m} f(x_k) y_k \\
\left(\sum_{k=0}^{m} f(x_k)g(x_k)\right) a + \left(\sum_{k=0}^{m} g(x_k)^2\right) b + \left(\sum_{k=0}^{m} h(x_k)g(x_k)\right) c & = & \sum_{k=0}^{m} g(x_k) y_k \\
\left(\sum_{k=0}^{m} f(x_k)h(x_k)\right) a + \left(\sum_{k=0}^{m} f(x_k)h(x_k)\right) b + \left(\sum_{k=0}^{m} h(x_k)^2\right) c & = & \sum_{k=0}^{m} h(x_k) y_k
\end{array}\right.$$

We note that the system of normal equations is always symmetric. We only need to compute half of the entries.

8.5 General linear LSM

Let $g_0, g_1, g_2, \cdots g_n$ be $n+1$ given functions (they don't need to be linear).

Given data set (x_k, y_k), $k = 0, 1, \cdots, m$. (m and n are in general different).

We search for a function in the form

$$y(x) = \sum_{i=0}^{n} c_i g_i(x)$$

that best fit the data.

Here g_i's are called *basis functions.*

How to choose the basis functions? They are chosen such that the system of the normal equations is regular (invertible) and well-conditioned.

Requirements. g_i's must be a set of linearly independent functions. Meaning: one can not be written as linear combination of the others, or

$$\sum_{i=0}^{n} c_i g_i(x) = 0, \quad \text{if and only if} \quad c_0 = c_1 = c_2 = \cdots = c_n = 0.$$

Define error function

$$\psi(c_0, c_1, \cdots, c_n) = \sum_{k=0}^{m} \Big[y(x_k) - y_k \Big]^2 = \sum_{k=0}^{m} \left[\sum_{i=0}^{n} c_i g_i(x) - y_k \right]^2$$

At minimum, we have

$$\frac{\partial \psi}{\partial c_j} = 0, \qquad j = 0, 1, \cdot, n.$$

This gives:

$$\sum_{k=0}^{m} 2 \left[\sum_{i=0}^{n} c_i g_i(x_k) - y_k \right] g_j(x_k) = 0$$

Re-arranging the ordering of summation signs:

$$\sum_{i=0}^{n} \left(\sum_{k=0}^{m} g_i(x_k) g_j(x_k) \right) c_i = \sum_{k=0}^{m} g_j(x_k) y_k, \qquad j = 0, 1, \cdot, n.$$

This gives the system of normal equations:

$$A\vec{c} = \vec{b}$$

where $\vec{c} = (c_0, c_1, \cdots, c_n)^t$ and

$$A = \{a_{ij}\}, \qquad a_{ij} = \sum_{k=0}^{m} g_i(x_k) g_j(x_k)$$

$$\vec{b} = \{b_j\}, \qquad b_j = \sum_{k=0}^{m} g_j(x_k) y_k.$$

We note that this A is symmetric.

8.6 Non-linear LSM

Next is an example of quasi-linear LSM.

Example 5. Consider fitting the data with the function

$$y(x) = a \cdot b^x$$

This means, we need to find (a, b) such that this $y(x)$ best fit the data.

Do a variable change:

$$\ln y = \ln a + x \cdot \ln b\,.$$

Let

$$S = \ln y, \qquad \bar{a} = \ln a, \qquad \bar{b} = \ln b.$$

Given data set (x_k, y_k). Compute $S_k = \ln y_k$ for all k.

We can now find $(\bar{a}, \bar{b})$ such that S_k best fits (x_k, S_k).

Then, transform back to the original variable

$$a = \exp\{\bar{a}\}, \qquad b = \exp\{\bar{b}\}.$$

Example 6. Non-linear LSM. For example,

$$y(x) = ax \cdot \sin(bx).$$

We can not find a variable change that can change this problem into a linear one. So we will now deal with it as a non-linear problem.

Define error

$$\psi(a,b) = \sum_{k=0}^{m} \Big[y(x_k) - y_k \Big]^2 = \sum_{k=0}^{m} \Big[a x_k \cdot \sin(b x_k) - y_k \Big]^2.$$

At minimum:

$$\frac{\partial \psi}{\partial a} = 0 : \qquad \sum_{k=0}^{m} 2 \Big[a x_k \cdot \sin(b x_k) - y_k \Big] \cdot [x_k \cdot \sin(b x_k)] = 0$$

$$\frac{\partial \psi}{\partial a} = 0 : \qquad \sum_{k=0}^{m} 2 \Big[a x_k \cdot \sin(b x_k) - y_k \Big] \cdot [a x_k \cdot \cos(b x_k) x_k] = 0$$

We now have a 2×2 system of non-linear equations to solve for (a, b)!

May use Newton's method to find a root.

May have several solutions, including all the maximum, minimum and saddle points. (see slides)

Chapter 9

Numerical solution of ordinary differential equations (ODE)

9.1 Introduction

Definition of ODE: an equation which contains one or more ordinary derivatives of an unknown function.

Example 1. Let $x = x(t)$ be the unknown function of t, ODE examples can be

$$x' = x^2, \qquad x'' + x \cdot x' + 4 = 0, \qquad \text{etc.}$$

We consider the initial-value problem for first-order ODE

$$(*) \qquad \begin{cases} x' = f(t,x), & --\text{differential equation} \\ x(t_0) = x_0 & ---\text{initial condition, given} \end{cases}$$

Some examples:

$$\begin{aligned} x' = x+1, \quad x(0) = 0. \qquad & \text{solution:} \qquad x(t) = e^t - 1 \\ x' = 2, \quad x(0) = 0. \qquad & \text{solution:} \qquad x(t) = 2t. \end{aligned}$$

In many situations, exact solutions can be very difficult/impossible to obtain.

Numerical solutions: Given (*), find $x_n = x(t_n)$, $n = 1, 2, \cdots, N$, and $t_0 < t_1 < \cdots < t_N$. Here t_N is final computing time.

Take uniform time step: Let h be the time step length

$$t_{n+1} - t_n = h, \qquad t_k = t_0 + kh$$

Overview:

- Taylor series method, and error estimates
- Runge-Kutta methods
- Multi-step methods
- System of ODE
- High order equations and systems
- Stiff systems
- Matlab solvers

9.2 Taylor series methods for ODE

Given

$$x'(t) = f(t, x(t)), \qquad x(t_0) = x_0.$$

Let's find $x_1 = x(t_1) = x(t_0 + h)$. Taylor expansion gives

$$x(t_0 + h) = x(t_0) + hx'(t_0) + \frac{1}{2}h^2 x''(t_0) + \cdots = \sum_{m=0}^{\infty} \frac{1}{m!} h^m x^{(m)}(t_0)$$

Taylor series method of order m: take the first $(m+1)$ terms in Taylor expansion.

$$x(t_0 + h) = x(t_0) + hx'(t_0) + \frac{1}{2}h^2 x''(t_0) + \cdots + \frac{1}{m!} h^m x^{(m)}(t_0).$$

Error in each step:

$$x(t_0 + h) - x_1 = \sum_{k=m+1}^{\infty} \frac{1}{k!} h^k x^{(k)}(t_0) = \frac{1}{(m+1)!} h^{m+1} x^{(m+1)}(\xi)$$

for some $\xi \in (t_0, t_1)$.

For $m = 1$, we have Euler's method:

$$x_1 = x_0 + hx'(t_0) = x_0 + h \cdot f(t_0, x_0)$$

General formula for step number k:

$$x_{k+1} = x_k + h \cdot f(t_k, x_k), \qquad k = 0, 1, 2, \cdots$$

For $m = 2$, we have

$$x_1 = x_0 + hx'(t_0) + \frac{1}{2}h^2 x''(t_0)$$

Using

$$x''(t_0) = \frac{d}{dt} f(t_0, x(t_0)) = f_t(t_0, x_0) + f_x(t_0, x_0) \cdot x'(t_0) = f_t(t_0, x_0) + f_x(t_0, x_0) \cdot f(t_0, x_0)$$

we get

$$x_1 = x_0 + h f(t_0, x_0) + \frac{1}{2} h^2 \left[f_t(t_0, x_0) + f_x(t_0, x_0) \cdot f(t_0, x_0) \right]$$

For general step k, we have

$$x_{k+1} = x_k + h f(t_k, x_k) + \frac{1}{2} h^2 \left[f_t(t_k, x_k) + f_x(t_k, x_k) \cdot f(t_k, x_k) \right]$$

Example 2. Set up Taylor series methods with $m = 1, 2$ for

$$x' = -x + e^{-t}, \qquad x(0) = 0.$$

(Exact solution is $x(t) = te^{-t}$.)

Answer. The initial data gives $x_0 = 0$.
For $m = 1$, we have

$$x_{k+1} = x_k + h(-x_k + e^{-t_k}) = (1 - h)x_k + he^{-t_k}$$

For $m = 2$, we have

$$x'' = (-x + e^{-t})' = -x' - e^{-t} = x - e^{-t} - e^{-t} = x - 2e^{-t}$$

so

$$\begin{aligned} x_{k+1} &= x_k + hx'_k + \frac{1}{2} h^2 x''_k \\ &= x_k + h\left(-x_k + \exp\{-t_k\}\right) + \frac{1}{2} h^2 \left[x_k - 2\exp\{-t_k\} \right] \\ &= (1 - h + \frac{1}{2} h^2) x_k + (h - h^2) \exp\{-t_k\} \end{aligned}$$

Example 3. Set up Taylor series methods with $m = 1, 2, 3, 4$ for

$$x' = x, \qquad x(0) = 1.$$

(Exact solution $x(t) = e^t$)

Answer. We set $x_0 = 1$. Note that

$$x'' = x' = x, \quad x''' = x'' = x, \quad \cdots \quad x^{(m)} = x$$

We have

$$\begin{aligned}
m=1: &\qquad x_{k+1} = x_k + hx_k = (1+h)x_k \\
m=2: &\qquad x_{k+1} = x_k + hx_k + \frac{1}{2}h^2x_k = (1+h+\frac{1}{2}h^2)x_k \\
m=3: &\qquad x_{k+1} = x_k + hx_k + \frac{1}{2}h^2x_k + \frac{1}{6}h^3x_k = (1+h+\frac{1}{2}h^2+\frac{1}{6}h^3)x_k \\
m=4: &\qquad x_{k+1} = (1+h+\frac{1}{2}h^2+\frac{1}{6}h^3+\frac{1}{24}h^4)x_k
\end{aligned}$$

See slides.

Error analysis. Given ODE

$$x' = f(t,x), \qquad x(t_0) = x_0.$$

<u>Local truncation error</u> (error in each time step) for Taylor series method of order m is

$$e_L^{(k)} = |x_{k+1} - x(t_k+h)| = \left|\frac{h^{m+1}}{(m+1)!}x^{(m+1)}(\xi)\right| = \left|\frac{h^{m+1}}{(m+1)!}\frac{d^m f(\xi)}{dt^m}\right|, \quad \xi \in (t_k, t_{k+1}).$$

Here we use the fact that

$$x^{(m+1)} = \frac{d^m f}{dt^m}$$

Assume now that

$$\left|\frac{d^m f}{dt^m}\right| \le M.$$

We have

$$e_L^{(k)} \le \frac{M}{(m+1)!}h^{m+1} = \mathcal{O}(h^{m+1}).$$

<u>Total error:</u> sum over all local errors.

Detail: We want to compute $x(T)$ for some time $t = T$. Choose an h. Then total number of steps is

$$N = fracTh, \qquad i.e., \quad T = Nh.$$

Then

$$\begin{aligned}
E &= \sum_{k=1}^{N}\left|e_L^{(k)}\right| \le \sum_{k=1}^{N}\frac{M}{(m+1)!}h^{m+1} \\
&= N\frac{M}{(m+1)!}h^{m+1} = (Nh)\frac{M}{(m+1)!}h^m = \frac{MT}{(m+1)!}h^m = \mathcal{O}(h^m)
\end{aligned}$$

Therefore, the method is of order m.

In general: If the local truncation error is of order $\mathcal{O}(h^{a+1})$, then the total error is of $\mathcal{O}(h^a)$, i.e., one order less.

9.3 Runge Kutta methods

Difficulty in high order Taylor series methods:

$x'', x''', \cdots$, might be very difficult to get.

A better method: should only use $f(t,x)$, not its derivatives.
1st order method: The same as Euler's method.
2nd order method: Let $h = t_{k+1} - t_k$. Given x_k, the next value x_{k+1} is computed as

$$x_{k+1} = x_k + \frac{1}{2}(K_1 + K_2)$$

where

$$\begin{cases} K_1 &= h \cdot f(t_k, x_k) \\ K_2 &= h \cdot f(t_{k+1}, x_k + K_1) \end{cases}$$

This is called Heun's method.

Proof that this is a second order method: Taylor expansion in two variables

$$f(t+h, x+K_1) = f(t,x) + hf_t(t,x) + K_1 f_x(t,x) + \mathcal{O}(h^2, K_1^2).$$

We have $K_1 = hf(t,x)$, abd

$$K_2 = h\Big[f(t,x) + hf_t(t,x) + hf(t,x)f_x(t,x) + \mathcal{O}(h^2)\Big]$$

Then, our method is:

$$\begin{aligned} x_{k+1} &= x_k + \frac{1}{2}\Big[hf + hf + h^2 f_t + h^2 f f_x + \mathcal{O}(h^3)\Big] \\ &= x_k + hf + \frac{1}{2}h^2[f_t + f f_x] + \mathcal{O}(h^3) \end{aligned}$$

Compare this with Taylor expansion for $x(t_{k+1}) = x(t_k + h)$

$$\begin{aligned} x(t_k + h) &= x(t_k) + hx'(t_k) + \frac{1}{2}h^2 x''(t_k) + \mathcal{O}(h^3) \\ &= x(t_k) + hf(t_k, x_k) + \frac{1}{2}h^2[f_t + f_x x'] + \mathcal{O}(h^3) \\ &= x(t_k) + hf + \frac{1}{2}h^2[f_t + f_x f] + \mathcal{O}(h^3). \end{aligned}$$

We see the first 3 terms are identical, this gives the local truncation error:

$$e_L = \mathcal{O}(h^3)$$

meaning that this is a 2nd order method.

In general, Rung-Kutta methods of order m takes the form

$$x_{k+1} = x_k + w_1 K_1 + w_2 K_2 + \cdots + w_m K_m$$

where

$$\begin{cases} K_1 &= h \cdot f(t_k, x_k) \\ K_2 &= h \cdot f(t_k + a_2 h, x + b_2 K_1) \\ K_3 &= h \cdot f(t_k + a_3 h, x + b_3 K_1 + c_3 K_2) \\ &\vdots \\ K_m &= h \cdot f(t_k + a_m h, x + \sum_{i=1}^{m-1} \phi_i K_i) \end{cases}$$

The parameters w_i, a_i, b_i, ϕ_i are carefully chosen to guarantee the order m. NB! The choice is NOT unique!

The classical RK4 : a 4th order method takes the form

$$x_{k+1} = x_k + \frac{1}{6}\Big[K_1 + 2K_2 + 2K_3 + K_4\Big]$$

where

$$\begin{aligned} K_1 &= h \cdot f(t_k,\ x_k) \\ K_2 &= h \cdot f(t_k + \frac{1}{2}h,\ x_k + \frac{1}{2}K_1) \\ K_3 &= h \cdot f(t_k + \frac{1}{2}h,\ x_k + \frac{1}{2}K_2) \\ K_4 &= h \cdot f(t_k + h,\ x_k + K_3) \end{aligned}$$

See slides for codes.

9.4 An adaptive Runge-Kutta-Fehlberg method

In general, we have

1. smaller time step h gives smaller error;
2. The error varies at each step, depending on f.

Optimal situation: h varies each step to get uniform error at each step.

This leads to *adaptive methods.*

Key point: How to get an error estimate at each time step?

One possibility:

- Compute $x(t+h)$ from $x(t)$ with step h;
- Compute $x(t+\frac{1}{2}h)$ from $x(t)$ with step $\frac{1}{2}h$, then compute $x(t+h)$ from $x(t+\frac{1}{2}h)$ with step $\frac{1}{2}h$; Call this $\bar{x}(t+h)$;
- Then, $|x(t+h)-\bar{x}(t+h)|$ gives a measure to error;
- if error $>>$ tol, half the step size;
- if error $<<$ tol, double the step size;
- if error $\approx$ tol, keep the step size;

But this is rather wasteful of computing time. Although the idea is good.

A better method, by Fehlberg, building upon R-K methods. He has a 4th order method:

$$x(t+h) = x(t) + \frac{25}{216}K_1 + \frac{1408}{2565}K_3 + \frac{2197}{4104}K_4 - \frac{1}{5}K_5,$$

where

$$\begin{aligned}
K_1 &= h \cdot f(t,x) \\
K_2 &= h \cdot f(t+\frac{1}{4}h,\ x+\frac{1}{4}K_1) \\
K_3 &= h \cdot f(t+\frac{3}{8}h,\ x+\frac{3}{32}K_1+\frac{9}{32}K_2) \\
K_4 &= h \cdot f(t+\frac{12}{13}h,\ x+\frac{1932}{2197}K_1-\frac{7200}{2197}K_2+\frac{7296}{2197}K_3) \\
K_5 &= h \cdot f(t+h,\ x+\frac{439}{216}K_1-8K_2+\frac{3680}{513}K_3-\frac{845}{4104}K_4)
\end{aligned}$$

Adding an additional term:

$$K_6 = h \cdot f(t+\frac{1}{2}h,\ x-\frac{8}{27}K_1+2K_2-\frac{3544}{2565}K_3+\frac{1859}{4104}K_4-\frac{11}{40}K_5)$$

We obtain a 5th order method:

$$\bar{x}(t+h) = x(t) + \frac{16}{135}K_1 + \frac{6656}{12825}K_3 + \frac{28561}{56430}K_4 - \frac{9}{50}K_5 + \frac{2}{55}K_6.$$

Main interests here: The difference $|x(t+h)-\bar{x}(t+h)|$ gives an estimate for the error.

Pseudo code for adaptive RK45, with time step controller

```
Given $t_0, x_0, h_0, n_{max}, e_{min}, e_{max}, h_{min}, h_{max}$
set $h = h_0, x = x_0, k = 0$,
while $k < n_{max}$ do
    if $h < h_{min}$ then $h = h_{min}$
    else if $h > h_{max}$ then $h = h_{max}$
    end
    Compute RK4, RK5, and $e = |\text{RK4} - \text{RK5}|$
    if $e > e_{max}$, then $h = h/2$;
    else
        k=k+1; t=t+h; x=RK5;
        If $e < e_{min}$, the $h = 2 * h$; end
    end
end (while)
```

9.5 Multi-step methods

Given

$$x' = f(t, x), \quad x(t_0) = x_0,$$

Let $t_n = t_0 + nh$. If $x(t_n)$ is given, the exact value for $x(t_{n=1})$ would be

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} x'(s)\, ds$$

Assume we know $x_n, x_{n-1}, x_{n-2}, \cdots, x_{n-k}$ one can approximate the integrand $x'(s)$ by using interpolating polynomial.

Example Consider $k = 1$. Given x_n, x_{n-1}, we can compute f_n, f_{n-1} as

$$f_n = f(t_n, x_n), \qquad f_{n-1} = f(t_{n-1}, x_{n-1})$$

Use now linear approximation, i.e., straight line interpolation,

$$x'(s) \approx P_1(s) = f_{n-1} + \frac{f_n - f_{n-1}}{h}(s - t_{n-1}).$$

Then

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} P_1(s)\, ds = x_n + \frac{h}{2}(3f_n - f_{n-1}).$$

Adams-Bashforth method: The explicit version. Given

$$x, x_{n-1}, \cdots, x_{n-k}$$

and

$$f_n, f_{n-1}, \cdots, , f_{n-k}$$

find interpolating polynomial $P_k(t)$ that interpolates $(t_i, f_i)_{i=n-k}^{n}$. Compute

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} P_k(s)\,ds$$

which will always be in the form

$$x_{n+1} = x_n + h \cdot (b_0 f_n + b_1 f_{n-1} + b_2 f_{n-2} + \cdots + b_k f_{n-k}).$$

Good sides: Simple, minimum number of $f(\cdot)$ evaluations. Fast.

Disadvantage: Here we use interpolating polynomial to approximate a function outside the interval of interpolating points. This gives bigger error.

Improved version. implicit method, Adams-Bashforth-Moulton (ABM) method. Find interpolating polynomial $P_{k+1}(t)$ that interpolates

$$(f_{n+1}, t_{n+1}), \quad (f_n, t_n), \quad \cdots, \quad (f_{n-k}, t_{n-k}),$$

and use

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} P_{k+1}(s)\,ds$$

which will always be in the form

$$x_{n+1} = x_n + h \cdot (b_{-1} f_{n+1} + b_0 f_n + b_1 f_{n-1} + b_2 f_{n-2} + \cdots + b_k f_{n-k})$$

where $f_{n+1} = f(t_{n+1}, x_{n+1})$ which is unknown. Therefore, we get a non-linear equation. Can be solved by fixed-point iteration (Newton's method). Use AB solution as the initial guess, it will converge in 2-3 iterations.

Example For a couple of k values, we have

$$k = -1: \qquad x_{n+1} = x_n + h \cdot f_{n+1}, \qquad \text{(implicit backward Euler's method)}$$

$$k = 0: \qquad x_{n+1} = x_n + \frac{h}{2}(f_n + f_{n+1}), \qquad \text{(trapezoid rule)}$$

$$k = 1: \qquad x_{n+1} = x_n + h \cdot \left[\frac{5}{12} f_{n+1} + \frac{3}{12} f_n - \frac{1}{12} f_{n-1}\right]$$

With fixed point iteration:

Given $x_n, x_{n-1}, f_n, f_{n-1}$, compute AB solution with $k = 1$:

$$(P) \qquad \begin{cases} x^*_{n+1} &= x_n + h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right) \\ f^*_{n=1} &= f\left(t_{n+1}, x^*_{n+1}\right). \end{cases}$$

Do one iteration of Newton's method, to correct the error:

$$(C) \qquad \begin{cases} x_{n+1} &= x_n + \frac{h}{2}\left(f^*_{n+1} + f_n\right) \\ f_{n+1} &= f(t_{n+1}, x_{n+1}) \end{cases}$$

Here step (P) is called the *predictor*, and step (C) is the *corrector*.

This is called *predictor-corrector's method.*

9.6 Methods for first order systems of ODE

We consider

$$\vec{x}\,' = F(t, \vec{x}), \qquad \vec{x}(t_0) = \vec{x}_0$$

Here $\vec{x} = (x_1, x_2, \cdots, x_n)^t$ is a vector, and $F = (f_1, f_2, \cdots, f_n)^t$ is a vector-values function.

Write it out

$$\begin{cases} x'_1 &= f_1(t, x_1, x_2, \cdots, x_n) \\ x'_2 &= f_2(t, x_1, x_2, \cdots, x_n) \\ & \cdots \\ x'_n &= f_n(t, x_1, x_2, \cdots, x_n) \end{cases}$$

Remark: <u>All</u> methods for scalar equation can be used for systems!

Taylor series methods:

$$\vec{x}(t+h) = \vec{x} + h\vec{x}\,' + \frac{1}{2}h^2\vec{x}\,'' + \cdots + \frac{1}{m!}h^m\vec{x}^{(m)}$$

Example Consider

$$\begin{cases} x'_1 &= x_1 - x_2 + 2t - t^2 - t^3 \\ x'_2 &= x_1 + x_2 - 4t^2 + t^3 \end{cases}$$

We will need the high order derivatives:

$$\begin{cases} x''_1 &= x'_1 - x'_2 + 2 - 2t - 3t^2 \\ x''_2 &= x_1 + x_2 - 8t + 3t^2 \end{cases}$$

and

$$\begin{cases} x_1''' &= x_1'' - x_2'' - 2 - 6t \\ x_2''' &= x_1'' + x_2'' - 8 + 6t \end{cases}$$

and so on...

Runge-Kutta methods take the same form. For example, RK4:

$$\vec{x}_{k+1} = \vec{x}_k + \frac{1}{6}\Big[\vec{K}_1 + 2\vec{K}_2 + 2\vec{K}_3 + \vec{K}_4\Big]$$

where

$$\begin{aligned} \vec{K}_1 &= h \cdot F(t_k,\ \vec{x}_k) \\ \vec{K}_2 &= h \cdot F(t_k + \frac{1}{2}h,\ \vec{x}_k + \frac{1}{2}\vec{K}_1) \\ \vec{K}_3 &= h \cdot F(t_k + \frac{1}{2}h,\ \vec{x}_k + \frac{1}{2}\vec{K}_2) \\ \vec{K}_4 &= h \cdot F(t_k + h,\ \vec{x}_k + \vec{K}_3) \end{aligned}$$

Here everything is a vector instead of a scalar value.

9.7 Higher order equations and systems

Treatment: Rewrite it into a system of first order equations.

Example Higher order ODE

$$x^{(n)} = f(t, x, x', x'', \cdots, x^{(n-1)})$$

with

$$x(t_0), x'(t_0), x''(t_0), \cdots, x^{(n-1)}(t_0)$$

given.

Introduce a systematic change of variables

$$x_1 = x, \quad x_2 = x', \quad x_3 = x'', \quad \cdots \quad x_n = x^{(n-1)}$$

We then have

$$\begin{cases} x_1' &= x' = x_2 \\ x_2' &= x'' = x_3 \\ x_3' &= x''' = x_4 \\ & \vdots \\ x_{n-1}' &= x^{(n-1)} = x_n \\ x_n' &= x^{(n)} = f(t, x_1, x_2, \cdots, x_n) \end{cases}$$

This is a system of 1st order ODEs.

Systems of high-order equations are treated in the same way.

9.8 Stiff systems

A system is called *stiff* if the solution consists of components that vary with very different speed/frequency.

Example Consider

$$x' = -ax, \qquad x(0) = 1$$

the exact solution is

$$x(t) = e^{-t}$$

We see that

$$(P1) \qquad\qquad x \to 0 \quad \text{as} \quad t \to +\infty.$$

Solve it by Euler's method:

$$x_{n+1} = x_n - ahx_n = (1-ah)x_n, \quad \Rightarrow \quad x_n = (1-ah)^n x_0 = (1-ah)^h$$

In order to keep the property $(P1)$, in the approximate solution, i.e.,

$$x_n \to 0, \quad \text{as} \quad n \to +\infty,$$

We must have

$$|1-ah| < 1, \quad \Rightarrow \quad h < \frac{2}{a}$$

which gives a restriction to the time step size: it has to be sufficiently small.

Example Now consider a system

$$\left\{\begin{array}{rcl} x' &=& -20x - 19y \\ y' &=& -19x - 20y \end{array}\right. \qquad \left\{\begin{array}{rcl} x(0) &=& 2 \\ y(0) &=& 0 \end{array}\right.$$

The exact solution is

$$\left\{\begin{array}{rcl} x(t) &=& e^{-39t} + e^{-t} \\ y(t) &=& e^{-39t} - e^{-t} \end{array}\right.$$

Observations:

- The solution tends to 0, i.e., $x \to 0, y \to 0$ as $t \to +\infty$.
- Two components in the solution, e^{-39t} and e^{-t};
- They decay at a very different rate. The term e^{-39t} tends to 0 much faster than the term e^{-t};
- For large values of t, the term e^{-t} dominate.
- Therefore, e^{-39t} is called the *transient term*.

Solve it with Euler's method:

$$\begin{cases} x_{n+1} &= x_n + h \cdot (-20x_n - 19y_n) \\ y_{n+1} &= y_n + h \cdot (-19x_n - 20y_n) \end{cases} \qquad \begin{cases} x_0 &= 2 \\ y_0 &= 0 \end{cases}$$

One can show by induction that

$$\begin{cases} x_n &= (1-39h)^n + (1-h)^n \\ y_n &= (1-39h)^n - (1-h)^n \end{cases}$$

We must require that

$$x_n \to 0, \qquad y_n \to 0 \qquad \text{as} \quad n \to +\infty.$$

This gives the conditions

$$|1-39h| < 1 \qquad \text{and} \qquad |1-h| < 1$$

which implies

$$(1): \; h < \frac{2}{39} \qquad \text{and} \qquad (2): \; h < 2$$

We see that condition (1) is much stronger than condition (2), therefore it must be satisfied.

Condition (1) corresponds to the term e^{-39t}, which is the transient term and it tends to 0 very quickly as t grows. Unfortunately, time step size is restricted by this transient term.

A more stable method: Backward Euler, implicit method

$$\begin{cases} x_{n+1} &= x_n + h \cdot (-20x_{n+1} - 19y_{n+1}) \\ y_{n+1} &= y_n + h \cdot (-19x_{n+1} - 20y_{n+1}) \end{cases}$$

Let

$$A = \begin{pmatrix} -20 & -19 \\ -19 & -20 \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}, \qquad \vec{x}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}.$$

We can write

$$\begin{aligned} & \vec{x}_{n+1} = \vec{x}_n + A \cdot \vec{x}_{n+1}, \\ \Rightarrow \qquad & (I - hA)\vec{x}_{n+1} = \vec{x}_n \\ \Rightarrow \qquad & \vec{x}_{n+1} = (I - hA)^{-1}\vec{x}_n \end{aligned}$$

Take some vector norm on both sides

$$\|\vec{x}_{n+1}\| = \left\|(I - hA)^{-1}\vec{x}_n\right\| \leq \left\|(I - hA)^{-1}\right\| \|\vec{x}_n\|.$$

We see that if $\left\|(I - hA)^{-1}\right\| < 1$, then $\vec{x}_n \to 0$ as $n \to +\infty$.
Use the l_2 norm:

$$\left\|(I - hA)^{-1}\right\| = \max_i \left|\lambda_i (I - hA)^{-1}\right| = \max_i \frac{1}{|(1 - h \cdot \lambda_i(A))|}$$

We have

$$\lambda_1(A) = -1, \qquad \lambda_2(A) = -39$$

They are both negative, therefore $1 - h\lambda_i > 1$, implying

$$\left\|(I - hA)^{-1}\right\| < 1$$

independent of the value of h.
Therefore, this method is called *unconditionally stable.*

Advantage: Can choose large h, always stable. Suitable for stiff systems.

Disadvantage: Must solve a system of linear equations at each time step

$$(I - hA)\vec{x}_{n+1} = \vec{x}_n$$

Longer computing time for each step. Not recommended if the system is not stiff.