Introduction to Numerical Methods

Ahmed H. Sameh and Vivek Sarin
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398

Contents

List of Figures

List of Tables

Chapter 1

MATLAB Primer

Elementary

Starting MATLAB	`> matlab`
Online help	`help` <*topic*>
	`help` <*function*>
Useful help topics	`ops lang elmat elfun graph2d`
	`graphics uitools strfun iofun`
Keyword search	`lookfor` <*keyword*>
Other commands	`what which`
Interrupt	`Control-C`
Exit	`quit`

1.1 Basic Features

- Simple Math

```
>> 4+2/4
ans =
    4.5000
```

- Variables: case-sensitive, must start with a letter

```
>> a=4, b=5; c=a+b
a =
     4
c =
     9
>> who
Your variables are:
a         ans       b         c
```

- Useful commands

  `save` <*filename*>
  `load` <*filename*>
  `clear` <*variable list*> (Caution: when used without <*variable list*>, your entire workspace will be cleaned!)
  `fprintf sprintf`

1.2 Scientific features

- Math functions: trigonometric, exponential, complex, rounding, remainder.

```
>> x=sqrt(2)
x =
    1.4142
>> y=sin(pi/2)
y =
     1
```

1.3 Vectors

- Simple vector

```
>> x=[1 4 9]
x =
     1     4     9
>> y=sqrt(x)
y =
     1     2     3
```

- Addressing elements

```
>> y(2)
ans =
     2
>> y(3:-1:1) % start=3, step=-1, end=1
ans =
     3     2     1
>> y([3 1 1])
ans =
     3     1     1
```

- Constructing vectors

```
>> x=[0:0.1:1];
>> x=linspace(0,1,11);
>> a=1:5, b=1:2:9
a =
     1     2     3     4     5
b =
     1     3     5     7     9
>> c = [b a]
c =
     1     3     5     7     9     1     2     3     4     5
>> d = [b; a]
d =
     1     3     5     7     9
     1     2     3     4     5
```

- Scalar-Vector Operations

```
>> a+1
ans =
     2     3     4     5     6
>> 2*a+1
ans =
     3     5     7     9    11
```

- Vector Operations

```
>> a+b
ans =
     2     5     8    11    14
>> a.*b
ans =
     1     6    15    28    45
>> a./b
ans =
    1.0000    0.6667    0.6000    0.5714    0.5556
>> a.^2
ans =
     1     4     9    16    25
>> b.^a
ans =
    1           9         125        2401       59049
```

- Matrix Orientation

```
>> c = [1 2; 3 4; 5 6+i]
c =
   1.0000             2.0000
   3.0000             4.0000
   5.0000             6.0000 + 1.0000i
>> d = c'
d =
   1.0000             3.0000             5.0000
   2.0000             4.0000             6.0000 - 1.0000i
>> e = c.'
e =
   1.0000             3.0000             5.0000
   2.0000             4.0000             6.0000 + 1.0000i
```

1.4 Figures

```
>> x=linspace(0,pi,20);    % create equally spaced point vector
>> y = sin(x);             % function of x
>> plot(x,y)               % plot y
>> figure(2);              % another figure
>> plot(x,y,':',x,y,'o');  % plot y twice - line and circles
>> xlabel('variable x')    % label x axis
```

```
>> ylabel('function of x') % label y axis
>> title('sample plot')    % label title
```

This produces the following figures shown in Fig. 1.1 To save these figures in files, we need to use the

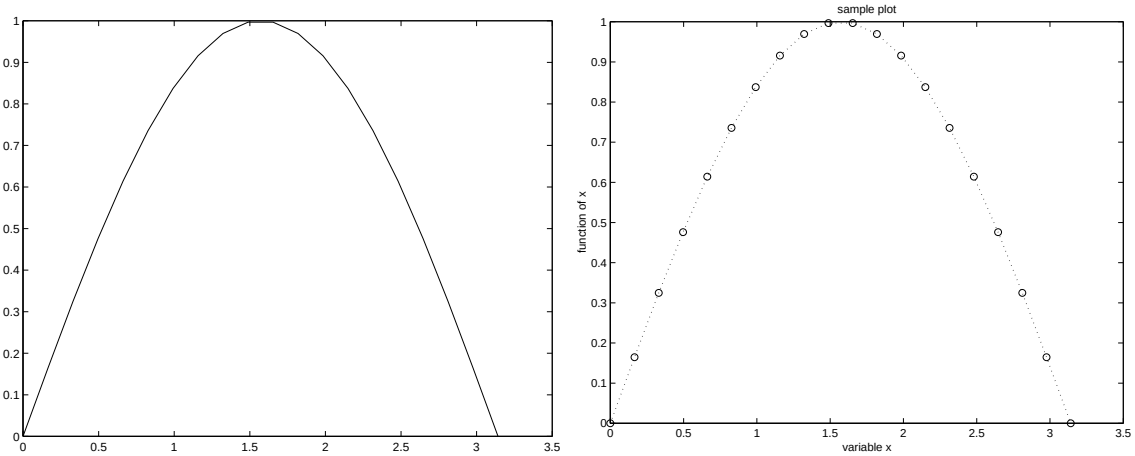


Figure 1.1: Sample figures in MATLAB

command `print`

```
>> help print

 PRINT  Print graph or SIMULINK system; or save graph to M-file.

  Syntax: PRINT [ -ddevice] [ -options ] <filename>

  PostScript devices are:
    -dps    - PostScript for black and white printers
    -deps   - Encapsulated PostScript

  Options only for use with PostScript and GhostScript devices:
    -append    - Append, not overwrite, the graph to
 PostScript file

  Other options are:
    -Pprinter  - Specify the printer to use (Unix & VMS only)
    -fhandle   - Handle Graphics handle of Figure to print

>> print -dps -f1 first.ps
>> print -dps -f2 second.ps
```

1.5 Script files

We can generate a script file with a sequence of commands that can be executed in MATLAB by typing the name of the file. For example, by typing the commands from the previous section into a file called `samplescript.m`, we can generate the figures in Fig. 1.1 by typing:

```
>> samplescript
```

The file `samplescript.m` can be viewed by the following command.

```
>> type samplescript
```

```
x=linspace(0,pi,20);    % create equally spaced point vector
y = sin(x);             % function of x
plot(x,y)               % plot y
figure(2);              % another figure
plot(x,y,':',x,y,'o');  % plot y twice - line and circles
xlabel('variable x')    % label x axis
ylabel('function of x') % label y axis
title('sample plot')    % label title
```

1.6 Control Flow

- FOR statement

```
      for variable = expr,
statement, ..., statement
      end

      for I = 1:N,
          for J = 1:N,
              A(I,J) = 1/(I+J-1);
          end
      end
```

  The following example steps S with increments of -0.1

```
     for S = 1.0: -0.1: 0.0,
      ...
     end
```

- IF statement

```
      if expression
        statements
      elseif expression
        statements
      else
        statements
      end
```

- WHILE statement

```
      while expression
        statements
      end
```

- Other

```
break
return
pause <seconds>
```

1.7 Functions

- Sample function (in file called `myqr.m`)

```
function [Q,R] = myqr(A)
%
% Sample funtion file myqr.m
% This is wrapper around the QR function provided
% in MATLAB
%
% Usage: [Q,R] = myqr(A)
%   A: input matrix of (size m x n)
%   Q: orthogonal basis for columns of A (size m x m)
%   R: upper triangular matrix s.t. A = QR (size m x n)
%
  [X, Y] = qr(A);
  Q = X;          % initialize output variables
  R = Y;          % initialize output variables
  return;         % not necessary
```

- Information about the function

```
>> what

M-files in the current directory /home/sarin/CS414

myqr

>> help myqr

  Sample funtion file myqr.m
  This is wrapper around the QR function provided
  in MATLAB

  Usage: [Q,R] = myqr(A)
    A: input matrix of (size m x n)
    Q: orthogonal basis for columns of A (size m x m)
    R: upper triangular matrix s.t. A = QR (size m x n)
```

- Calling the function

```
>> A = rand(4);
>> [Q,R]= myqr(A);
>> Q

Q =

   -0.7606   -0.1354    0.4523    0.4457
   -0.1850   -0.8151   -0.5489   -0.0062
   -0.4858    0.0754    0.0617   -0.8686
   -0.3890    0.5582   -0.7002    0.2163
```

```
>> R

R =

   -1.2492   -1.0478   -1.3141   -1.0811
         0   -0.6971    0.0148   -0.4867
         0         0   -0.3892   -0.2615
         0         0         0    0.3409
```

- Content of the file `myqr.m`

```
>> type myqr

function [Q,R] = myqr(A)
%
% Sample funtion file myqr.m
% This is wrapper around the QR function provided
% in MATLAB
%
% Usage: [Q,R] = myqr(A)
%   A: input matrix of (size m x n)
%   Q: orthogonal basis for columns of A (size m x m)
%   R: upper triangular matrix s.t. A = QR (size m x n)
%
  [X, Y] = qr(A);
  Q = X;          % initialize output variables
  R = Y;          % initialize output variables
  return;         % not necessary
```

Chapter 2

Introduction

2.1 Scientific Computing

- Mathematical Modeling
- Numerical Analysis
- Software Development

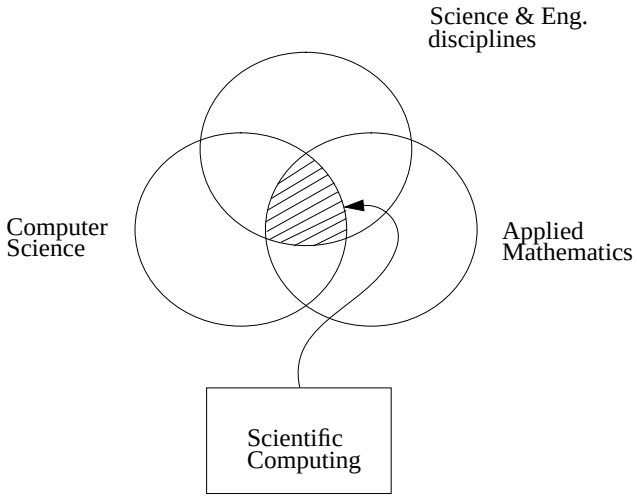


Figure 2.1: Scientific Computing is the interface of Computer Science, Engineering and Applied Mathematics.

Applications of Scientific Computing

- fluid dynamics
- materials science
- semiconductor device simulation
- circuit simulation

- weather modeling
- computational biology
- econometrics and computational finance, etc.

Example 2.1 Consider the electrical network shown below: Given that the values of the resistance are as

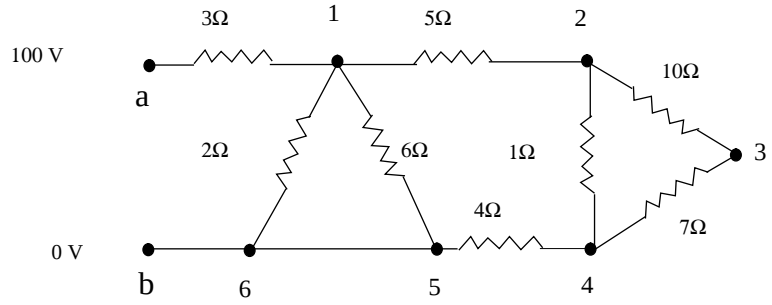


Figure 2.2: Electrical network

shown in Ohms and the potential applied between a & b is 100 volts, obtain the voltages at nodes 1 through 6.

Solution From Ohm's law, the current flowing from node p to node q is given by,

$$I_{pq} = \frac{v_p - v_q}{R_{pq}}$$

where v_p & v_q are the voltages at nodes p and q, respectively, and R_{pq} is the resistance segment pq. Also from Kirchoff's law: *the sum of currents arriving or leaving each node is zero.*
Node 1:

$$\begin{aligned} I_{a1} + I_{61} + I_{51} + I_{21} &= 0 \\ \frac{100 - v_1}{3} + \frac{v_6 - v_1}{2} + \frac{v_5 - v_1}{6} + \frac{v_2 - v_1}{5} &= 0. \end{aligned}$$

Since $v_b = v_6 = v_5 = 0$, we get

$$\begin{aligned} \frac{100 - v_1}{3} - \frac{2v_1}{3} + \frac{v_2 - v_1}{5} &= 0 \\ \Rightarrow 18v_1 - 3v_2 &= 500. \end{aligned}$$

Repeated application at nodes 2 to 6 yields the *linear system*

$$\begin{array}{rcrcrcrcl} 18v_1 & - & 3v_2 & & & & & = & 500 \\ 2v_1 & - & 13v_2 & + & v_3 & + & 10v_4 & = & 0 \\ & & 7v_2 & - & 17v_3 & + & 10v_4 & = & 0 \\ & & 28v_2 & + & 4v_3 & - & 39v_4 & = & 0 \end{array}$$

or

$$\begin{pmatrix} 18 & -3 & 0 & 0 \\ 2 & -13 & 1 & 10 \\ 0 & 7 & -17 & 10 \\ 0 & 28 & 4 & -39 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} 500 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$Av = f$$

MATLAB

```
 A = [ 18 -3 0 0
       2 -13 1 10
       0 7 -17 10
       0 28 4 -39 ]
f = [ 500; 0; 0; 0 ];
v = A \ f;
```

The preceding code give the value

```
v = [ 30.288
      15.059
      13.367
      12.183 ]
```

□

Example 2.2 Plot the function

$$f(x) = 2\sin x + 3\sin 2x + 7\sin 3x + 5\sin 4x$$

on the interval [–10,10] using MATLAB.

Solution

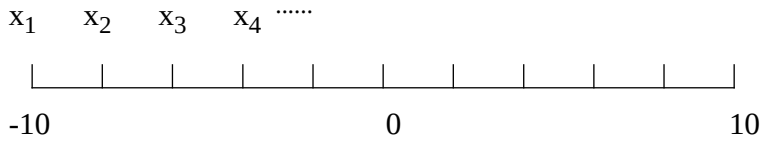


Figure 2.3: Equispaced points in the interval [–10,10] for plotting $f(x)$.

$$\begin{aligned} f(x) &= (2,3,7,5)\begin{pmatrix} \sin x \\ \sin 2x \\ \sin 3x \\ \sin 4x \end{pmatrix} \\ &= a^T b \quad \text{(inner product)} \\ &= b^T a \end{aligned}$$

$$a^T := \text{row vector}; \qquad b := \text{col. vector}$$

$$\begin{aligned} f(x_k) &= (\sin x_k, \sin 2x_k, \sin 3x_k, \sin 4x_k) * a \\ f_k &= b_k^T \cdot a \end{aligned}$$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{200} \end{bmatrix} = \begin{bmatrix} b_1^T \\ b_2^T \\ b_3^T \\ \vdots \\ b_{200}^T \end{bmatrix} * a \qquad \leftarrow \text{Matrix\_vector product}$$

200×1 200×4
vector *matrix*
$f = B * a$

MATLAB

```
n  = 200;
m  = 4;
xx = linspace(-10,10,n);         % row vector
x  = xx';                        % column vector
B  = zeros(m,n);                 % initialize
B  = [sin(x) sin(2*x) sin(3*x) sin(4*x)];
a  = [2; 3; 7; 5];
f  = B*a;
plot(x,f);
title('f(x)=2sin(x)+3sin(2x)+7sin(3x)+5sin(4x)');  % try it!!
```

□

2.2 Computational Errors

Roundoff Errors. Consider the following computation

$$\begin{aligned} a &= 9.99 \times 10^4 \\ b &= 9.99 \times 10^{-4} \end{aligned}$$

$$\begin{aligned} a+b &= (9.99 + 9.99 * 10^{-8}) * 10^4 \\ &= (9.99 + 0.0000000999) * 10^4 \\ &= 9.9900000999 * 10^4 \quad (11 \text{ digits}) \end{aligned}$$

Note that computing to 6 digits only, b would have no effect!

$$\begin{aligned} a/b &= 10^8 \\ a * b &= 99.8001 \end{aligned}$$

Truncation Errors. Consider evaluating the exponential of x using the formula

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

Since the terms in the series decrease in magnitude, e^x can be approximated by a truncated series with n terms, where n is a positive integer. The *Taylor approximation* for e^x is

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

where $0 < \eta < x$. The truncation error consists of the second term on the right hand side. As we increase the number of terms in the approximation, i.e., $n \to \infty$, the truncation error reduces to zero and the partial sums converge to the exact value of e^x. The following MATLAB code computes the relative error for approximations with n terms for $n = 1, \ldots, 50$, and plots the logarithm of the relative error versus n (see Fig. 1.4).

MATLAB

```
n = 50;
x = 10;
term = 1;
sum  = 1;
result = exp(x);
```

```
for k=1:n,
  term = x * term/k;
  sum = sum + term;
  error(k) = abs(result-sum);
end;
relerr = error/result;          % relative error
semilogy(1:n, relerr);
```

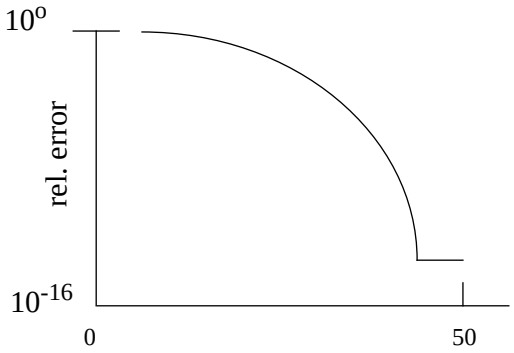


Figure 2.4: Relative error in computing e^x decreases with increase in the number of terms of the Taylor approximation.

Other examples of approximation with truncation errors include computation of derivative and integrals of functions. In this case, we use Taylor's theorem to define a function in its neighborhood as follows.

$$\begin{aligned} f(a+h) &= f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + \cdots + \frac{h^n}{n!}f^{(n)}(a) + R_n \\ R_n &= \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(z) \end{aligned}$$

where z is between a, and $(a+h)$. Similarly,

$$f(a-h) = f(a) - hf'(a) + \frac{h^2}{2!}f''(a) - \cdots$$

Taking the difference of the above equations, and dropping terms with h^3, h^5, etc., we get

$$f(a+h) - f(a-h) \simeq 2hf'(a)$$

or

$$f'(a) \simeq \frac{f(a+h) - f(a-h)}{2h}.$$

A computation is acceptable if the *computed* value differs from the *exact* value by no more than a specified *error tolerance*, i.e.,

$$|\text{computed solution} - \text{exact solution}| \leq \text{tolerance}$$

Main Idea.

> Obtaining exact answers for most problems is either impossible or impractical. An acceptable answer is one within a specified tolerance from the exact solution.

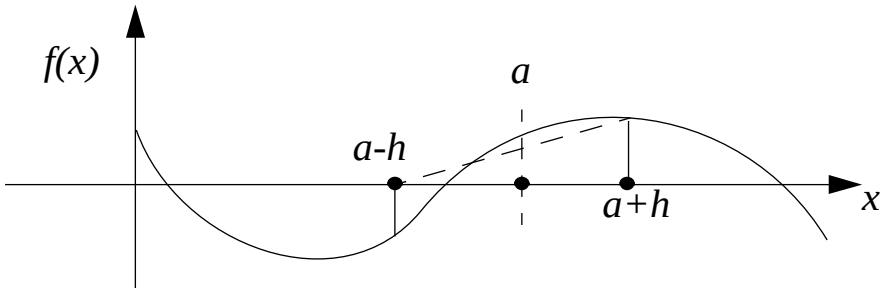


Figure 2.5: The derivative $f'(a)$ can be computed using Taylor series approximation of $f(x)$ in the neighborhood of $x = a$.

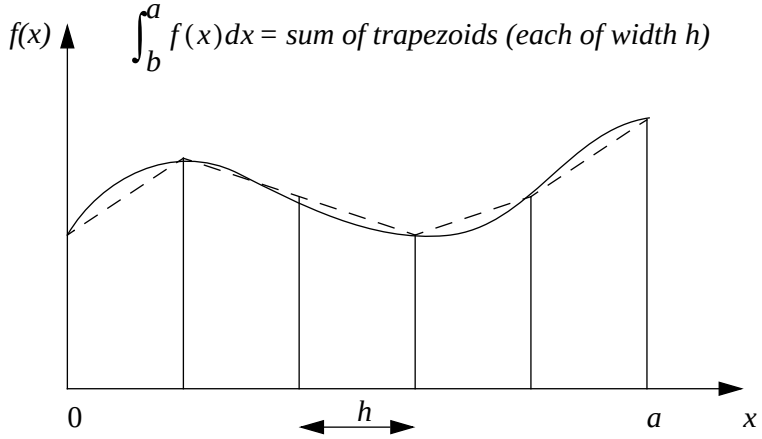


Figure 2.6: Approximating the integral of $f(x)$ by the sum of the areas of the trapezoids.

Error tolerances should be part of the specifications of an algorithm. For example, in computing e correct to 1 *unit* in the 8-th significant digit, both $e = 2.7182818$ & $e = 2.7182819$ are acceptable. There is a price to be paid for high accuracy, however!

> An *algorithm* is a complete description of well-defined operations through which each permissible input data is transformed into the output.

Let $\tilde{a}$ be an approximate value of a quantity whose exact value is a,

$$\begin{aligned} \text{error} &= \tilde{a} - a \\ \text{rel. error} &= \frac{\tilde{a} - a}{a} \\ \tilde{a} &= a(1 + \text{rel. error}) \end{aligned}$$

A convenient measure of accuracy is to measure the error relative to 1 ulp (one **u**nit in the **l**ast **p**lace). It requires, however, an explicit representation of the approximate value.

We define one unit in the last place to be the positive value obtained by replacing all the digits of the approximate value by zeros except for the last, which is replaced by 1. For example,

$$\begin{gathered} \pi \simeq 3.14160 \\ 1\ ulp = 0.00001 \\ (\pi = 3.14159265\ldots) \end{gathered}$$

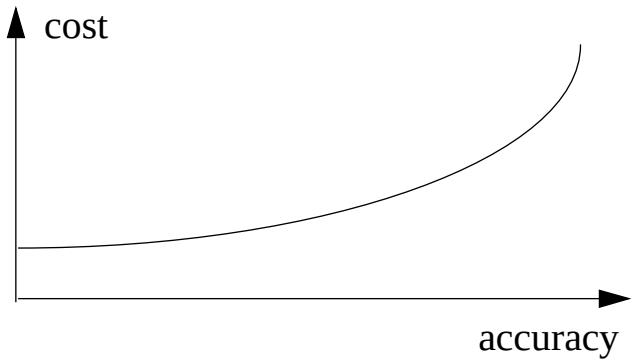


Figure 2.7: The cost of an algorithm increases with the accuracy desired.

The number of decimal places of accuracy, d, for fixed error

$$|\text{error}| = 10^{-d}$$

is given by

$$d = -\log_{10} |\text{error}|,$$

i.e., the approximation has d accurate decimal places.

Consider the following

$$\begin{aligned} a &= 2718.281828 \\ \tilde{a} &= 2718.282137 \\ |\tilde{a} - a| &= |0.000309|. \end{aligned}$$

The error and relative error are

$$\begin{aligned} |\text{error}| &= |0.000309| < 10^{-3} \\ |\text{rel. error}| &= \frac{|0.000309|}{2178.281828} \simeq 10^{-7}, \end{aligned}$$

The approximation $\tilde{a}$ is correct to 3 decimal places. To compute the number of digits of accuracy in the approximation, we assume that $|\text{rel. error}| = 10^{-\delta}$. Therefore,

$$\delta = -\log_{10} |\text{rel. error}| \simeq 7,$$

i.e., $\tilde{a}$ has 7 significant digits.

As another example, let

$$\begin{aligned} a &= 0.00299823 \\ \tilde{a} &= 0.00300000 \\ |\tilde{a} - a| &= 0.00000177 < 10^{-5} \\ \left|\frac{\tilde{a} - a}{a}\right| &= 0.00059 < 10^{-3} \end{aligned}$$

In this case, $\tilde{a}$ is correct to 5 decimals but only 3 significant digits.

2.3 Error Propagation

The representation of physical quantities is accompanied by errors due to finite precision arithmetic in computers. Computation based on this erroneous data incurs additional errors due to roundoff and truncation. Each step in the computation introduces errors that accumulate over the course of the computation. Fig. 1.8 shows that data error $a - \tilde{a}$ causes an error of magnitude ϵ_1 in $f(x)$ when the function is computed exactly. An additional error ϵ_2 may be introduced when the computation of $f(x)$ in prone to computational errors.

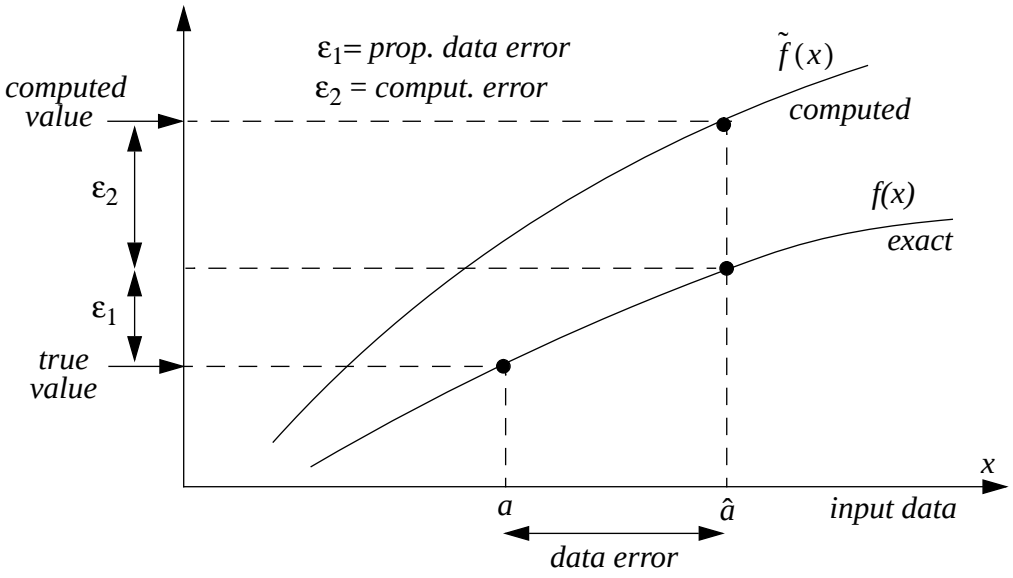


Figure 2.8: Propagation of errors in computation and data.

Therefore,

data error			=	$a - \tilde{a}$
propogated data error	=	ϵ_1	=	$f(\tilde{a}) - f(a)$
computational error	=	ϵ_2	=	$\tilde{f}(\tilde{a}) - f(\tilde{a})$
total error	=	$\epsilon_1 + \epsilon_2$	=	$\tilde{f}(\tilde{a}) - f(a)$

Example 2.3 Compute the length of the side c of the triangle in Fig. 1.9 for $\gamma = 1°$.

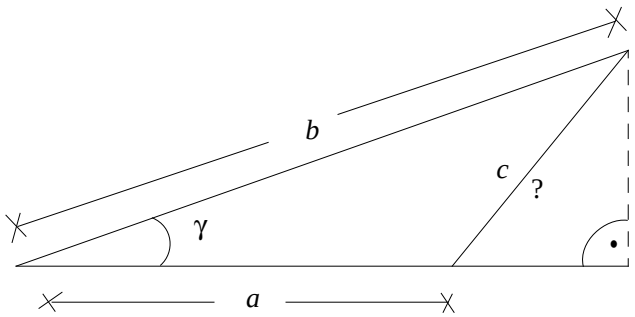


Figure 2.9: Computing sides of a triangle.

Solution We know that

$$c^2 = (b\cos\gamma - a)^2 + (b\sin\gamma)^2$$

$$\begin{aligned} &= a^2 + b^2 - 2ab\cos\gamma \\ &= a^2 + b^2 - 2ab(1 - 2\sin^2\frac{\gamma}{2}) \\ &= (b-a)^2 + 4ab\sin^2\frac{\gamma}{2} \end{aligned}$$

Therefore,

$$c = \left[(b-a)^2 + 4ab\sin^2\frac{\gamma}{2}\right]^{1/2}.$$

When $\gamma = 1^\circ = \frac{\pi}{180} \simeq 0.01745329$ radians, $c \simeq 2.0190487$. Table 1.1 shows the relative error in computing intermediate values and final answer. A relative error of 0.1% in data translates to a relative error of –2.3% in the final result. □

	True	Approx.	Rel. Error
a	100	100.1	0.1%
$b-a$	1	0.9	–10%
$(b-a)^2$	1	0.81	–19%
$4ab\sin^2\frac{\gamma}{2}$	3.0765...	3.0796...	0.1%
c^2	4.0765...	3.8896...	–4.6%
c	2.0190...	1.9722...	–2.3%

Table 2.1: Relative error in computation of intermediate values when computing the side of a triangle.

Example 2.4 Consider the problem of obtaining the roots of a polynomial

$$\begin{aligned} p(z) &= z^{20} - 210z^{19} + \cdots + 20! \\ &= (z-1)(z-2)(z-3)\cdots(z-20). \end{aligned}$$

If we perturb the coefficient (–210) to (–210+10^{-7}), some of the roots of the resulting polynomial became complex. □

This example shows that the *sensitivity* of a problem is indicated by the relative dependence of output error to input error. Formally, we define *condition number* of a problem as

$$\text{Condition Number} = \frac{|\text{rel. error in result}|}{|\text{rel. error in data}|}$$

A problem with a small condition number is considered *well-conditioned*, and demonstrates insensitivity to relative error in data. On the other hand, a problem with condition number much larger than 1 is considered *ill-conditioned*, and hence, has a greater sensitivity to relative error in data.

2.4 Error Propagation in Basic Arithmetic Operations

Let

$$\begin{aligned} \tilde{a} = a + \alpha &= a\left(1 + \frac{\alpha}{a}\right) = a(1+\epsilon_a) \\ \tilde{b} = b + \beta &= b\left(1 + \frac{\beta}{b}\right) = b(1+\epsilon_b) \end{aligned}$$

where ϵ_a and ϵ_b are the relative errors in $\tilde{a}$ and $\tilde{b}$, respectively.

Addition. Let

$$\begin{aligned}(\tilde{a}+\tilde{b}) &= (a+b)+(\alpha+\beta)\\ &= (a+b)\left[1+\frac{\alpha+\beta}{a+b}\right]\\ &= (a+b)(1+\epsilon_{a+b})\end{aligned}$$

where

$$\begin{aligned}\epsilon_{a+b} &= \frac{\alpha+\beta}{a+b}\\ &= \frac{a}{a+b}\cdot\frac{\alpha}{a}+\frac{b}{a+b}\cdot\frac{\beta}{b}\\ &= \frac{a}{a+b}\epsilon_a+\frac{b}{a+b}\epsilon_b\end{aligned}$$

From this, it is clear that if $a \simeq -b$, then ϵ_{a+b} may be very large. For example, if

$$\begin{aligned}a &= 80.0499 \qquad \epsilon_a = 10^{-6}\\ b &= -80.0000 \qquad \epsilon_b = -10^{-6},\ \text{then}\\ \epsilon_{a+b} &= \left(\frac{80.0499}{0.0499}+\frac{80}{0.0499}\right)\times 10^{-6} \simeq 10^{-3},\end{aligned}$$

i.e., the computed sum has only 3 significant digits! This is an example of *catastrophic cancellation*.

Multiplication. Suppose

$$\begin{aligned}\tilde{a}\cdot\tilde{b} &= ab(1+\epsilon_a)(1+\epsilon_b)\\ &\simeq ab(1+\epsilon_a+\epsilon_b) \qquad \text{[Note: } \epsilon_a\epsilon_b \ll 1\text{]}\\ &= ab(1+\epsilon_{a*b})\\ \Rightarrow \epsilon_{a*b} &= \epsilon_a+\epsilon_b.\end{aligned}$$

Division. Again, consider

$$\begin{aligned}\frac{\tilde{a}}{\tilde{b}} &= \frac{a(1+\epsilon_a)}{b(1+\epsilon_b)}\\ &\simeq \frac{a}{b}(1+\epsilon_a)(1-\epsilon_b)\\ &\simeq \frac{a}{b}(1+\epsilon_a-\epsilon_b) \qquad \text{[Note: } \epsilon_a\epsilon_b \ll 1\text{]}\\ &= \frac{a}{b}(1+\epsilon_{a/b})\\ \Rightarrow \epsilon_{a/b} &= \epsilon_a-\epsilon_b.\end{aligned}$$

Over here, we have used the approximation

$$\frac{1}{1+\epsilon_b} = 1-\epsilon_b+\epsilon_b^2-\ldots \simeq 1-\epsilon_b$$

where we assume that $\epsilon_b^2 \ll 1$.

Avoiding Catastrophic Cancellation. One can avoid catastrophic cancellation in some cases. For instance, consider the following computation

$$y = \sqrt{x+\delta} - \sqrt{x} \qquad \text{for } x = 100, \delta = 0.1$$

using only 2-decimal arithmetic in evaluating the square roots. We get

$$\sqrt{x+\delta} = \sqrt{100.1} = 10.0049987\ldots$$

and therefore, $\tilde{y} = 0$ giving

$$|\text{rel. error}| = \left|\frac{\tilde{y} - y}{y}\right| = 1$$

This situation can be remedied by computing the solution in a different way. We use the following equivalent formula

$$y = [\sqrt{x+\delta} - \sqrt{x}]\left[\frac{\sqrt{x+\delta} + \sqrt{x}}{\sqrt{x+\delta} + \sqrt{x}}\right] = \frac{\delta}{\sqrt{x+\delta} + \sqrt{x}}.$$

In 2-decimal arithmetic, the solution is

$$\tilde{y} = \frac{0.1}{20} = 0.005$$

and the relative error is

$$|\text{rel. error}| = \left|\frac{0.005 - 0.0049987}{0.0049987}\right| \simeq 2.6 \times 10^{-4}.$$

An alternate approach relies on the Taylor expansion to compute small difference in the value of a function. The Taylor expansion of a function $f(x)$ about x is

$$f(x+\delta) = f(x) + \delta \cdot f'(x) + \frac{\delta^2}{2!} f''(x) + \cdots$$

In our example, we use

$$f(x) = \sqrt{x}, \qquad f'(x) = \frac{1}{2\sqrt{x}}, \qquad f''(x) = -\frac{1}{4x\sqrt{x}}$$

Using the first three terms of the Taylor expansion, we have

$$f(x+\delta) - f(x) \simeq \delta \cdot f'(x) + \frac{\delta^2}{2!} f''(x)$$

Therefore, in 2-decimal arithmetic,

$$\sqrt{x+\delta} - \sqrt{x} \simeq \frac{\delta}{2\sqrt{x}} - \frac{\delta^2}{8x\sqrt{x}} = \frac{0.1}{20} - \frac{.01}{8000} \simeq 0.005$$

Chapter 3

Floating-point Computation

3.1 Positional Number System

An integer N in a number system of base (or radix) β may be written as

$$N = a_n\beta^n + a_{n-1}\beta^{n-1} + \cdots + a_1\beta + a_0 = P_n(\beta)$$

where a_i are positive integers less than β and $P_n(\beta)$ is a polynomial of degree n. For example, in the decimal system:

$$\beta = 10, \qquad 0 \le a_i \le 9$$

while in the binary system

$$\beta = 2, \qquad 0 \le a_i \le 1.$$

A fraction z in base β may be written as

$$z = a_{-1}\beta^{-1} + a_{-2}\beta^{-2} + \cdots + a_{-k}\beta^{-k} + \cdots = \sum_{j=1}^{\infty} a_{-j}\beta^{-j}$$

Note that the above series always converges. For example, if all $a_{-j} = 1$, then, for $\beta = 2$

$$z = \sum_{j=1}^{\infty} 2^{-j} = \frac{1/2}{1 - 1/2} = 1.$$

Recall that the sum of a *geometric series* with n terms is given by

$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = a\left(\frac{1 - r^n}{1 - r}\right).$$

Also, the geometric series with infinite terms, i.e., $n = \infty$,

$$a + ar + ar^2 + ar^3 + \cdots = \frac{a}{(1 - r)}$$

provided $-1 < r < 1$.

Example 3.1 Convert the binary integer $N = (110100010)_2$ to decimal.

Solution Before we compute N, we first describe an efficient way to compute the value of a polynomial. As an example, we can rewrite a degree 3 polynomial as follows

$$\begin{aligned} P_3(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 \\ &= \underbrace{a_0 + x(a_1 + x(\underbrace{a_2 + x \cdot a_3}))}_{} \end{aligned}$$

and evaluate the terms in the order of the parenthesis. This is equivalent to the *Horner's rule* for evaluating polynomials.

Horner's rule for polynomials

$b_0 = a_n$
for $k = 1, 2, \ldots, n$
    $b_k = a_{n-k} + b_{k-1} * x$
end;
Operation Count: $2n$ arithmetic steps

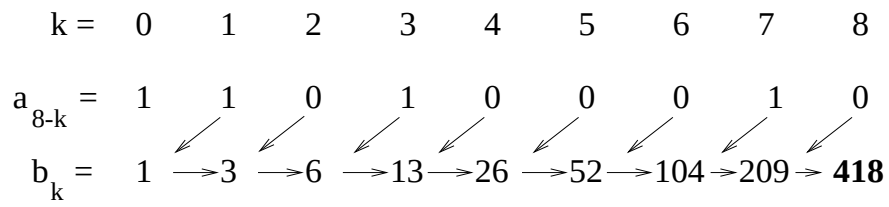


Figure 3.1: Converting binary integer $(110100010)_2$ to decimal.

Therefore, $N = (110100010)_2 = (418)_{10}$. □

Example 3.2 Convert the decimal integer $N = (418)_{10}$ to binary.

Solution To convert from decimal to binary, we make use of the fact that if

$$N = a_n 2^n + a_{n-1} 2^{n-1} + \cdots + a_1 * 2 + a_0,$$

then for

$$\begin{aligned} N : \text{even} &\Rightarrow a_0 = 0 \\ N : \text{odd} &\Rightarrow a_0 = 1. \end{aligned}$$

Using this fact, we get

$$\begin{array}{lll} N_0 = 418 & = a_n 2^n + a_{n-1} 2^{n-1} + \cdots + a_1 * 2 + a_0 & \Rightarrow a_0 = 0 \\ N_1 = \frac{N_0 - a_0}{2} = 209 & = a_n 2^{n-1} + \cdots + a_2 * 2 + a_1 & \Rightarrow a_1 = 1 \\ N_2 = \frac{N_1 - a_1}{2} = 104 & = a_n 2^{n-2} + \cdots + a_3 * 2 + a_2 & \Rightarrow a_2 = 0, \text{etc.} \end{array}$$

Therefore, $N = (418)_{10} = (110100010)_2$. □

Example 3.3 Let us turn our attention to decimal fractions. It is important to note that not all decimal fractions are exactly representable in the binary system. As an example, convert $z = (0.1)_{10}$ to binary.

Solution The algorithm for converting a decimal fraction to binary is

k =	0	1	2	3	4	5	6	7	8	9
N_k =	418	209	104	52	26	13	6	3	1	0
a_k =	0	1	0	0	0	1	0	1	1	

Figure 3.2: Converting decimal integer $(418)_2$ to binary.

Converting decimal fraction to binary

$z_1 = z$
for $k = 1, 2, 3, \ldots$
$\quad a_{-k} = \begin{cases} 1 & \text{if } 2z_k \geq 1 \\ 0 & \text{if } 2z_k < 1 \end{cases}$
$\quad z_{k+1} = 2z_k - a_{-k}$
end;

Let

$$z_1 = (0.1)_{10} = a_{-1}2^{-1} + a_{-2}2^{-2} + a_{-3}2^{-3} + \cdots$$

Therefore,

$$2z_1 = a_{-1} + a_{-2}2^{-1} + a_{-3}2^{-2} + \cdots$$

Following the algorithm,

$$\begin{array}{ll} z_1 = (0.1)_{10}; & 2z_1 = 0.2 < 1 \Rightarrow a_{-1} = 0 \\ z_2 = 2z_1 - a_{-1} = 0.2; & 2z_2 = 0.4 < 1 \Rightarrow a_{-2} = 0 \\ z_3 = 2z_2 - a_{-2} = 0.4; & 2z_3 = 0.8 < 1 \Rightarrow a_{-3} = 0 \\ z_4 = 2z_3 - a_{-3} = 0.8; & 2z_4 = 1.6 > 1 \Rightarrow a_{-4} = 1 \\ z_5 = 2z_4 - a_{-4} = 0.6; & 2z_5 = 1.2 > 1 \Rightarrow a_{-5} = 1 \\ z_6 = 2z_5 - a_{-5} = 0.2; & 2z_6 = 0.4 < 1 \Rightarrow a_{-6} = 0 \\ \vdots & \end{array}$$

Complete this example to show that

$$z = (0.1)_{10} = (0.0\underbrace{0011}\underbrace{0011}\underbrace{0011}\cdots)_2,$$

i.e., z has a nonterminating binary representation! □

Terminating an infinite binary fraction. Suppose we terminate an infinite binary fraction after t digits, and convert to decimal. The magnitude of the maximum error cannot exceed

$$\begin{aligned} y &= 2^{-(t+1)} + 2^{(t+2)} + 2^{(t+3)} + \cdots \\ &= 2^{-(t+1)}[1 + 2^{-1} + 2^{-2} + \cdots] \\ &= 2^{-(t+1)}\left(\tfrac{1}{1-\frac{1}{2}}\right) = 2^{-t}. \end{aligned}$$

3.2 Floating-point Arithmetic

Scientific calculations are carried out in floating-point arithmetic. This is a number system which uses a finite number of digits to approximate the real number system we use in exact computation.

Let $x \neq 0$ be in the real number system,

$$x = \pm 10^e (0.d_1 d_2 d_3 \cdots),$$

$$d_1 \neq 0, \qquad 0 \leq d_j \leq 9, \qquad j > 1.$$

The t-digit floating-point (decimal) representation of x is given by

$$fl(x) = \pm 10^e (0.\delta_1 \delta_2 \cdots \delta_t),$$

$$\delta_1 \neq 0, \qquad 0 \leq \delta_j \leq 9, \qquad j \geq 2$$

The condition $\delta_1 \neq 0$ implies that it is a *normalized* floating-point number. Here, e is called the *exponent*, and $(\delta_1 \delta_2 \cdots \delta_t)$ is called the *mantissa* or fraction. Usually,

$$-N \leq e \leq M,$$

and

$$e < -N \qquad \Rightarrow \text{underflow}$$

$$e > M \qquad \Rightarrow \text{overflow}$$

The floating-point numbers are **not** uniformly distributed, unlike the mathematician's real number system, but are clustered around zero. This can be seen from the following examples for decimal and binary systems.

Example 3.4 $\beta = 10; \quad t = 3; \quad N = M = 3.$

Aside from zero, the numbers are

$$\left.\begin{array}{l} \pm 0.100 * 10^{-3} \\ \pm 0.101 * 10^{-3} \\ \vdots \\ \pm 0.999 * 10^{-3} \end{array}\right\} \quad \text{spacing} = 0.001 * 10^{-3} = 10^{-6}$$

$$\left.\begin{array}{l} \pm 0.100 * 10^{-2} \\ \pm 0.101 * 10^{-2} \\ \vdots \\ \pm 0.999 * 10^{-2} \end{array}\right\} \quad \text{spacing} = 0.001 * 10^{-2} = 10^{-5}$$

$$\vdots$$

$$\left.\begin{array}{l} \pm 0.100 * 10^{+3} \\ \pm 0.101 * 10^{+3} \\ \vdots \\ \pm 0.999 * 10^{+3} \end{array}\right\} \quad \text{spacing} = 0.001 * 10^{+3} = 1$$

□

Example 3.5 $\beta = 2; \quad t = 3; \quad N = 1; \quad M = 2.$

Aside from zero, the numbers are:

$$\left.\begin{array}{l}\pm 0.100 * 2^{-1}\\ \pm 0.101 * 2^{-1}\\ \pm 0.110 * 2^{-1}\\ \pm 0.111 * 2^{-1}\end{array}\right\} \quad \text{spacing} = (0.001)_2 * \frac{1}{2} = \frac{1}{16}$$

$$\left.\begin{array}{l}\pm 0.100 * 2^{+0}\\ \pm 0.101 * 2^{+0}\\ \pm 0.110 * 2^{+0}\\ \pm 0.111 * 2^{+0}\end{array}\right\} \quad \text{spacing} = (0.001)_2 * 1 = \frac{1}{8}$$

$$\vdots$$

$$\left.\begin{array}{l}\pm 0.100 * 2^{+2}\\ \pm 0.101 * 2^{+2}\\ \pm 0.110 * 2^{+2}\\ \pm 0.111 * 2^{+2}\end{array}\right\} \quad \text{spacing} = (0.001)_2 * 4 = \frac{1}{2}$$

□

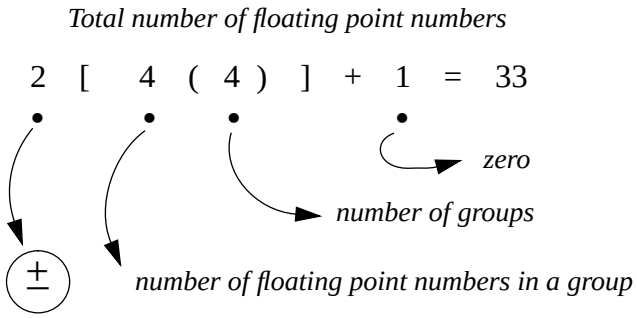


Figure 3.3: A count of the floating point numbers.

In general, for a floating point system with radix β and mantissa of length t

$$fl(x) = \pm\beta^e(0.\delta_1\delta_2\cdots\delta_t) \qquad \delta_1 \neq 0; \qquad 0 \leq \delta_1 \leq \beta - 1,$$

floating point numbers in a given group are given by

$$\boxed{\begin{array}{llll}\pm & 0.100 & \cdots & 0\ 0\ *\ \beta^e\\ \pm & 0.100 & \cdots & 0\ 1\ *\ \beta^e\\ & \vdots & & \\ \pm & 0.vvv & \cdots & v\ v\ *\ \beta^e\end{array}}$$

$$\pm \quad 0.100 \quad \cdots \quad 0\ 0\ *\ \beta^{e+1}$$

$$v = \beta - 1$$

$$\text{spacing} = \gamma = \beta^e * \beta^{-t} = \beta^{e-t}$$

Also, the number of floating point numbers in a group is given by

$$\frac{\beta^e - \beta^{e-1}}{\beta^{e-t}} = \frac{\beta - 1}{\beta^{1-t}}$$

Therefore, total number of floating point numbers in the system is given by

$$2\left[\left(\frac{\beta-1}{\beta^{1-t}}\right)(M+N+1)\right]+1.$$

Also, the floating point number with *smallest* magnitude is $\pm\beta^{-N-1}$ whereas the floating point number with *largest* magnitude is $\beta^M-\beta^{M-t}$.

Next, we describe how to obtain floating point digits from the exact representation of a number. Consider

$$x=\pm\beta^e(0.d_1d_2d_3\cdots), \qquad d_1\neq 0; \qquad 0\leq d_j\leq\beta-1,$$

i.e., $\beta^{e-1}\leq x<\beta^e$. We present *only* two ways in which the floating point digits δ_i are obtained from the exact digits d_i.

Chopping.

$$\delta_i=d_i \quad i=1,2,\ldots,t$$

Observing that, in the interval $[\beta^{e-1},\beta^e]$, the floating point numbers are uniformly distributed with a separation of β^{e-t}, we see that

$$|fl(x)-x|\leq\beta^{e-t}$$

and

$$\left|\frac{fl(x)-x}{x}\right|\leq\frac{\beta^{e-t}}{|x|}\leq\frac{\beta^{e-t}}{\beta^{e-1}}=\beta^{1-t},$$

i.e.,

$$fl(x)=x(1+\mu_c),$$

where $|\mu_c|\leq\beta^{1-t}$.

Rounding.

$$(\delta_1\delta_2\cdots\delta_t)=\left\lfloor d_1d_2\cdots d_t\cdot d_{t+1}+\frac{1}{2}\right\rfloor$$

where $\lfloor y\rfloor$ is the greatest integer less than or equal to y. Therefore,

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

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

or

$$fl(x)=x(1+\mu_R)$$

where $|\mu_R|\leq\frac{1}{2}\beta^{1-t}$.

$$fl(x)=x(1+\delta); \qquad |\delta|\leq\epsilon \qquad [\epsilon=\text{unit roundoff}]$$

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

We assume throughout our error analysis of basic arithmetic operations that our hypothetical computer performs each arithmetic operation correctly to $2t$ *digits*. Hence, for two normalized numbers

$$a = 0 \cdot a_1 a_2 \cdots a_t * \beta^e$$
$$b = 0 \cdot b_1 b_2 \cdots b_t * \beta^f$$

we have

$$fl(a \bullet b) = (a \bullet b)(1 + \delta)$$

where $\bullet$ represents operations such as $+$, $*$, or $/$, and $|\delta| \leq \epsilon$ (unit roundoff).

Example 3.6

$$\begin{aligned} fl(x + y + z) &:= fl[fl(x + y) + z] \\ &= [(x + y)(1 + \delta_1) + z](1 + \delta_2) \\ &= (x + y)(1 + \delta_1)(1 + \delta_2) + z(1 + \delta_2) \end{aligned}$$

Therefore,

$$fl(x + y + z) \cong (x + y + z) + [(x + y)(\delta_1 + \delta_2) + z \cdot \delta_2]$$

and

$$|fl(x + y + z) - (x + y + z)| \leq 2\epsilon|x + y + z|.$$

In general,

$$\left| fl\left(\sum_{i=1}^{n} x_i\right) - \left(\sum_{i=1}^{n} x_i\right) \right| \leq (n-1)\epsilon \left(\sum_{i=1}^{n} |x_i|\right)$$

□

Example 3.7 The order of computation is important. For example, one may obtain different errors depending on the order in which a set of numbers are added. Let us analyse the error in the following computations.

(i) $fl(x_1 + x_2 + x_3) \qquad x_1 > x_2 > x_3$

(ii) $fl(x_3 + x_2 + x_1) \qquad x_1 > x_2 > x_3$

Solution

(i)
$$\begin{aligned} z_1 &= x_1 \\ z_2 &= x_2 \hat{+} z_1 \\ &= [10^4(0.00125400 \hat{+} 0.38270000)] \\ &= fl[10^4(0.38395400)] \\ &= 10^4(0.3840) \\ z_3 &= x_3 \hat{+} z_2 \\ &= [10^4(0.38400000 \hat{+} 0.00015670)] \\ &= fl[10^4(0.38415670)] \\ &= 10^4(0.3842) \end{aligned}$$

Exact answer := $10^4(0.38411070)$, i.e., we have one unit error in the last figure.

(ii) $z_1 = x_3$
$z_2 = x_2 \hat{+} z_1$
$= [10^2(0.12540000 \hat{+} 0.01567000)]$
$= fl[10^2(0.14107000)]$
$= 10^2(0.1411)$
$z_3 = x_1 \hat{+} z_2$
$= [10^4(0.38270000 \hat{+} 0.00141100)]$
$= fl[10^4(0.38411100)]$
$= 10^4(0.3841)$, i.e., correct to 4 decimals.

□

If error in $\sum_{i=1}^{n} x_i$ is to be minimized, we should have the numbers of largest magnitude be added last; i.e., associated with the smallest possible error; and the numbers of smallest magnitude added first (i.e., associated with the largest error).

3.3 Condition of Problems and Stability of Algorithms

Poor accuracy in the output can depend either on the problem or the algorithm. The problem can be ill-conditioned; i.e., small perturbations in the input data could lead to large perturbations in the output. Or, the algorithm may be poorly constructed, i.e., unstable.

Example 3.8 Consider the linear system

$$\begin{aligned} 0.913x_1 + 0.659x_2 &= 0.254 \\ 0.780x_1 + 0.563x_2 &= 0.217 \end{aligned}$$

whose exact solution is:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1.0 \\ -1.0 \end{pmatrix}.$$

Compute this solution using 3-decimal arithmetic with chopping.

Solution Multiply the first equation by

$$\begin{aligned} m &= -fl\left(\frac{0.780}{0.913}\right) \\ &= -fl(0.854326) = -0.854 \end{aligned}$$

and add to the second equation. Thus we get,

$$\begin{aligned} 0 * x_1 + x_2 * fl[0.563 - fl(0.659 * 0.854)] \\ = fl[0.217 - fl(0.254 * 0.854)] \end{aligned}$$

or

$$\begin{aligned} [fl(0.563 - 0.562)]x_2 = fl(0.217 - 0.216) \\ 0.001x_2 = 0.001 \\ \text{i.e. } \tilde{x_2} = 1. \end{aligned}$$

Substituting in the 1^{st} equation, we get

$$0.913x_1 = fl(0.254 - 0.659) = -0.405$$

or

$$\tilde{x}_1 = fl(-0.405/0.913) = -0.443.$$

□

Let us compare the computed solution to the exact solution.

$$\text{Computed: } \begin{pmatrix} -0.443 \\ 1 \end{pmatrix} \qquad \text{Exact: } \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

The computed solution does not even have the correct sign! What went wrong? The disastrous calculation was that of computing x_2, i.e.,

$$[fl(0.563 - 0.562)]x_2 = fl(0.217 - 0.216)$$

Clearly, we cancelled all the significant digits and obtained a result heavily contaminated with rounding errors. The situation, however, is readily corrected if one uses 6-decimal arithmetic.

Example 3.9 Solve for the smaller root of the quadratic

$$x^2 - 6.433x + 0.009474 = 0$$

using 4-decimal arithmetic with rounding.

Solution The roots of the quadratic equation

$$ax^2 + bx + c = 0$$

are given by

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The smaller root is given by

$$\alpha = \frac{1}{2}\left[6.433 - \sqrt{(6.433)^2 - 4(0.009474)}\right]$$

and the calculation is organized as follows:

1. $u = fl[10^1(0.6433)]^2 = 10^2(0.4138)$
2. $v = fl[10^1(0.4000) * 10^{-2}(0.9474)] = 10^{-1}(0.3790)$
3. $w = fl(u - v) = fl[10^2(0.41380000 - 0.00037900)] = 10^2(0.4134)$
4. $y = fl(\sqrt{w}) = 10^1(0.6430)$
5. $z = fl[10^1(0.6433 - 0.6430)] = 10^{-2}(0.3000)$
6. $\tilde{\alpha} = fl(z/2.0) = 10^{-2}(0.1500)$

□

The computed value $\tilde{\alpha}$ fails to agree with the true value $\alpha = 0.0014731$ in its second significant digit. Again, the disastrous calculation occurs in step 5 when we subtract nearly equal quantities.

Both examples 8 and 9 appear similar; in both cases **catastrophic cancellation** results in poor solutions. In spite of the similarity, the two examples fail for quite different reasons. The difficulty in example 9 is

caused by a poorly constructed algorithm. This difficulty can be avoided by an alternate expression for the smaller root:

$$\begin{aligned} x_- &= \frac{1}{2a}(-b-\sqrt{b^2-4ac})\cdot\left[\frac{-b+\sqrt{b^2-4ac}}{-b+\sqrt{b^2-4ac}}\right] \\ &= \frac{1}{2a}\cdot\left[\frac{4ac}{-b+\sqrt{b^2-4ac}}\right] \end{aligned}$$

Thus, we compute α as follows:

$$\alpha = \frac{1}{2}\left(6.433-\sqrt{\cdots}\right) * \left(\frac{6.433+\sqrt{\cdots}}{6.433+\sqrt{\cdots}}\right).$$

Now,

$$\begin{aligned} \tilde{\alpha} &= \frac{1}{2} * \frac{4*(0.009474)}{6.433+\sqrt{(6.433)^2-4*(0.009474)}} \\ &= fl\left[\frac{2*(.009474)}{fl[10^1(0.6433)+10^1(0.6430)]}\right] \\ &= 10^{-2}(0.1473) \end{aligned}$$

which agrees with the exact solution to 4 significant digits.

On the other hand, there are no alternative rearrangements of the computations in example 8 that will allow us to solve the problem accurately using only 3-digit arithmetic (with chopping).

For example 9, we see that a stable algorithm for computing the roots of a quadratic $ax^2+bx+c=0$ may be outlined as follows:

$$\rho_1 = -\left[\frac{b+sign(b)*\sqrt{b^2-4ac}}{2a}\right], \qquad \rho_2 = \frac{c}{a*\rho_1}.$$

Note that $\rho_1+\rho_2 = -b/a$, $\rho_1*\rho_2 = c/a$.

3.4 Ill-Conditioned Problems

The ill-conditioning of a problem is measured by a number K_P called the condition number of the problem P. The larger K_P is the more ill-conditioned is the problem.

Let P be the problem of computing the value of $f(x)$ at the point $x=\alpha$ (solution). Let the relative perturbation to the data be of magnitude $\left|\frac{\Delta\alpha}{\alpha}\right|$. Then

$$K_P * \left|\frac{\Delta\alpha}{\alpha}\right| = \left|\frac{f(\alpha+\Delta\alpha)-f(\alpha)}{f(\alpha)}\right|.$$

From Taylor series

$$f(\alpha+\Delta\alpha) \simeq f(\alpha)+\Delta\alpha f'(\alpha)+\frac{(\Delta\alpha)^2}{2!}f''(\alpha)+\cdots$$

Therefore,

$$K_P * \left|\frac{\Delta\alpha}{\alpha}\right| \simeq \left|\frac{\Delta\alpha f'(\alpha)}{f(\alpha)}\right|,$$

i.e.,

$$K_P \simeq \left|\frac{\alpha f'(\alpha)}{f(\alpha)}\right|.$$

Example 3.10 Consider the two equations in x, and y

$$\begin{aligned} x + \beta y &= 1 \\ \beta x + y &= 0; \qquad \beta > 0, \end{aligned}$$

for which the exact solution is given by

$$x = 1/(1-\beta^2) \;\; \& \;\; y = \beta/(\beta^2 - 1).$$

Clearly, x & y suffer the consequences of severe cancellation when β is close to 1. The condition number of the problem of determining $x(\beta)$ alone, is given by

$$K_P \simeq \left| \frac{\beta * x'(\beta)}{x(\beta)} \right| = \left| \beta * \frac{2\beta}{(1-\beta^2)^2} * (1-\beta^2) \right| = \left| \frac{2\beta^2}{(1-\beta^2)} \right|,$$

which becomes large for $\beta \simeq 1$. Hence, the problem of determining x is ill-conditioned for β near 1. On the other hand, the problem for determining

$$z(\beta) = x + y = 1/(1+\beta)$$

is well-conditioned for any value of $\beta > 0$. In fact,

$$K_{P'} \simeq \left| \frac{\beta * z'(\beta)}{z(\beta)} \right| = \left| \beta * \frac{-1}{(1+\beta)^2} * (1+\beta) \right| = \left| \frac{\beta}{(1+\beta)} \right|,$$

which is much smaller than K_P for $\beta \simeq 1$. □

Figure 3.4 shows an ill-conditioned problem. Although ill-conditioned problems are intrinsically difficult

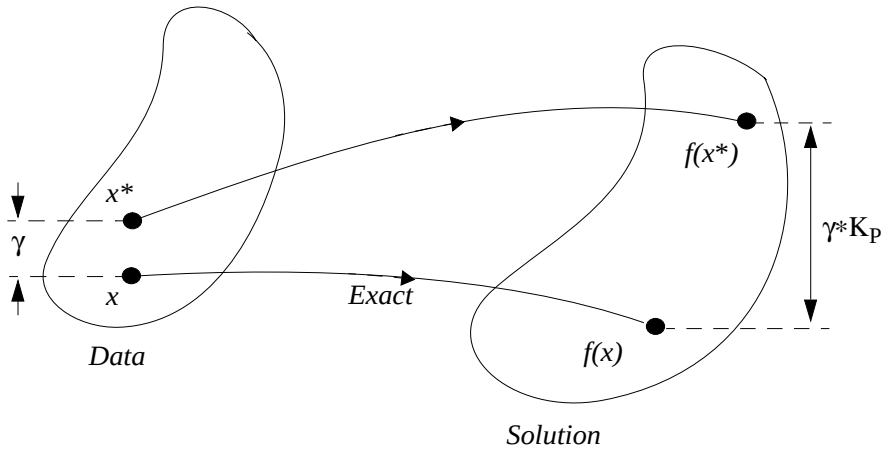


Figure 3.4: An ill-conditioned problem is characterized by a large perturbation in the computed solution due to a small perturbation in the data.

to solve on computer, one can still design effective algorithms for these problems. A *stable algorithm* for an ill-conditioned problem is an algorithm for which the computed solution is near the exact solution of another slightly perturbed problem. Fig. 3.5 shows an ill-conditioned problem for which an algorithm computes the solution $f^*(x)$ at point x. The algorithm is said to be stable since the computed solution is near the exact solution $f(x^*)$ at point x^*, where x^* is assumed to a slight perturbation of x.

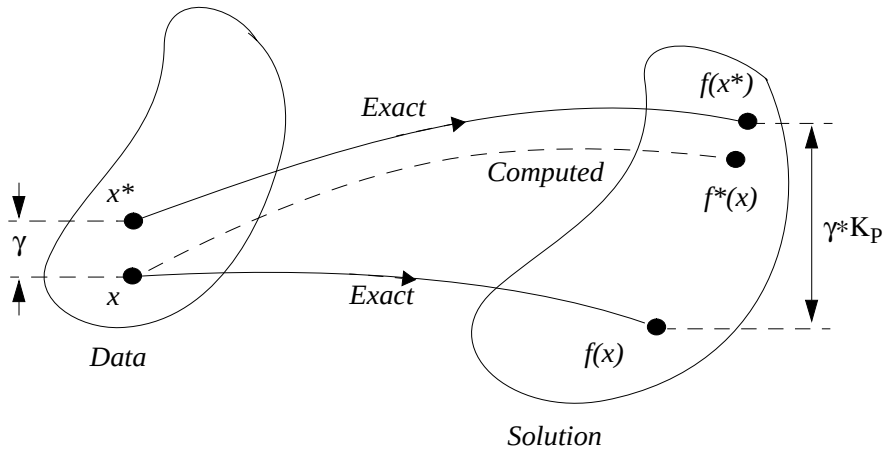


Figure 3.5: A stable algorithm for ill-conditioned problem generates a computed solution that is close to the solution of a slightly perturbed problem.

Chapter 4

Matrix Computations

The main objective of this chapter is to solve the linear system

$$A\mathbf{x} = \mathbf{f},$$

where A is a square $n \times n$ matrix, and $\mathbf{x}$ and $\mathbf{f}$ are vectors of order n.

4.1 Basics

Matrix.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = [a_{ij}] \qquad i, j = 1, 2, ..., n.$$

Column Vectors.

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

Row Vectors.

$$\mathbf{x}^T = (x_1, x_2, \ldots, x_n)$$

$$\mathbf{f}^T = (f_1, f_2, \ldots, f_n)$$

Inner Products.

$$\mathbf{a}^T\mathbf{b} = (a_1, a_2, \ldots, a_n) \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \sum_{i=1}^{n} a_i b_i \leftarrow \text{scalar}$$

Operation count: n multiplications, $n-1$ additions $\simeq 2n$ operations.

Outer Products.

$$\mathbf{ab}^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} (b_1, b_2, \ldots, b_n) = \begin{pmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & & \vdots \\ a_nb_1 & a_nb_2 & \cdots & a_nb_n \end{pmatrix} \leftarrow \text{matrix}$$

Operation count: n^2 multiplications $= n^2$ operations.

4.1.1 Setting up matrix problems in MATLAB

Hilbert matrix. Suppose we want to construct a matrix A s.t.

$$A_{ij} = \frac{1}{i+j-1}$$

MATLAB

```
A = zeros(n,n);
for i = 1:n,
  for j = 1:n,
      A(i,j) = 1/(i+j-1);
  end;
end;
```

This is known as a Hilbert matrix, and can also be created in MATLAB by `A = hilb(n)`. For $n = 5$, this matrix is of the form

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{pmatrix}$$

Observing that the Hilbert matrix is symmetric, the above MATLAB code fragment could be modified to take advantage of this symmetry:

MATLAB

```
A = zeros(n,n);
for i = 1:n,
  for j = i:n,
      A(i,j) = 1/(i+j-1);
      A(j,i) = A(i,j);
  end;
end;
```

Another matrix. As another example of matrix set-up, consider the following MATLAB code:

MATLAB

```
P = zeros(n,n);
P(:,1) = ones(n,1);   % P(:,1) = 1st column of P
for i = 2:n,
  for j = 2:i,
      P(i,j) = P(i-1,j-1)+P(i-1,j);
  end;
end;
```

For $n = 5$, the above code generates the following matrix

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

Next, we present two special matrices that can be created given a vector of n components, i.e., the elements of the matrix depend only on n parameters.

Vandermonde matrix. This a special type of matrix that can be created from a vector of n components. Given a vector

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

we can create a matrix V as follows:

MATLAB

```
n = length(x);
V(:,1) = ones(n,1);
for j = 2:n,
    V(:,j) = x .* V(:,j-1);
end;
```

Over here, the jth column of V is obtained by element-wise multiplication of the vector x with the $(j-1)$th column. For $n = 4$, V is of the form

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \end{pmatrix}$$

Circulant matrix. Another matrix that can be created from a vector of size n is the circulant matrix. Given a vector

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

we can create a circulant matrix C as follows:

MATLAB

```
function C = circulant(a)
  n = length(a);
  C(1,:) = a';
  for j = 2:n,
      C(i,:)=[C(i-1,n) C(i-1,1:n-1)];
  end;
```

Over here, the ith row of C is obtained by permuting $(i-1)$th row; the last element of $(i-1)$th row is permuted to the start of the ith row. For $n = 4$, C is of the form

$$C = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ a_4 & a_1 & a_2 & a_3 \\ a_3 & a_4 & a_1 & a_2 \\ a_2 & a_3 & a_4 & a_1 \end{pmatrix}$$

4.1.2 The Structure of Matrices

The nonzero elements of a matrix determine its structure. Following are commonly occurring matrices.

$$\begin{matrix} \star & 0 & 0 & 0 & 0 \\ 0 & \star & 0 & 0 & 0 \\ 0 & 0 & \star & 0 & 0 \\ 0 & 0 & 0 & \star & 0 \\ 0 & 0 & 0 & 0 & \star \end{matrix}$$
Diagonal

$$\begin{matrix} \star & \star & 0 & 0 & 0 \\ \star & \star & \star & 0 & 0 \\ 0 & \star & \star & \star & 0 \\ 0 & 0 & \star & \star & \star \\ 0 & 0 & 0 & \star & \star \end{matrix}$$
Tridiagonal

$$\begin{matrix} \star & 0 & 0 & 0 & 0 \\ \star & \star & 0 & 0 & 0 \\ \star & \star & \star & 0 & 0 \\ \star & \star & \star & \star & 0 \\ \star & \star & \star & \star & \star \end{matrix}$$
Lower triangular

$$\begin{matrix} \star & \star & \star & \star & \star \\ 0 & \star & \star & \star & \star \\ 0 & 0 & \star & \star & \star \\ 0 & 0 & 0 & \star & \star \\ 0 & 0 & 0 & 0 & \star \end{matrix}$$
Upper triangular

$$\begin{matrix} \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star \\ 0 & \star & \star & \star & \star \\ 0 & 0 & \star & \star & \star \\ 0 & 0 & 0 & \star & \star \end{matrix}$$
Upper Hessenberg

A *diagonal* matrix can be constructed from a vector d, where

$$d = [d_1\ d_2\ d_3\ \cdots\ d_n];$$

by the MATLAB statement `D = diag(d)`. This constructs the diagonal matrix D, s.t.,

$$D = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}$$

MATLAB also provides a way to specify diagonal entries other than the main diagonal. For example, consider the following code:

MATLAB

```
m = 3; k = 2;
v = [10 20 30];
A = diag(v,k);
```

This constructs a matrix A with the vector v forming the kth diagonal of A:

$$A = \begin{pmatrix} 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Also, the statement `v = diag(A,k)` extracts the kth diagonal in the form of a vector. A negative k refers to diagonals below the main diagonal while $k = 0$ refers to the main diagonal.

Block structures. Matrices with block structure can also be constructed in MATLAB. A block structured matrix consists of elements that are in turn smaller matrices. For example, a block structured matrix of the form

$$A = \begin{pmatrix} A_{11} & A_{12} \\ (3\times 3) & (3\times 2) \\ \\ A_{21} & A_{22} \\ (2\times 3) & (2\times 2) \end{pmatrix}$$

can be constructed from the following code

MATLAB

```
A11 = rand(3,3);
A12 = rand(3,2);
A21 = [1 2 3 ; 4 5 6];
A22 = [7 8 ; 9 10];
A  =  [A11 A12 ; A21 A22];
```

4.2 Matrix Operations

Matrix-Vector Multiplication. Given an $m \times n$ matrix A and a vector $\mathbf{x}$ of size n, we wish to compute the vector $\mathbf{y}$ of size m s.t.

$$\mathbf{y} = A\mathbf{x}$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

gives the vector $\mathbf{y}$ with elements

$$y_i = \sum_{k=1}^{n} a_{ik}x_k \qquad i = 1, 2, \ldots, m$$

Operation count: mn multiplications, $m(n-1)$ additions $\simeq 2mn$ operations.

MATLAB

```
y = zeros(m,1);
for i = 1:m,
  for j = 1:n,
      y(i) = y(i) + A(i,j) * x(j)
  end;
end;
```

This can be computed in a number of ways.
Alternative 1: (row-oriented)

$$A = \begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_m^T \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \Rightarrow \mathbf{y} = \begin{pmatrix} \mathbf{a}_1^T\mathbf{x} \\ \mathbf{a}_2^T\mathbf{x} \\ \vdots \\ \mathbf{a}_m^T\mathbf{x} \end{pmatrix}$$

The corresponding MATLAB program uses m *inner products*:

MATLAB

```
for i = 1:m,
    y(i) = A(i,:) * x;
end;
```

Alternative 2: (column-oriented)

$$A = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n), \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \Rightarrow \mathbf{y} = \sum_{j=1}^{n} x_j \mathbf{b}_j$$

The corresponding MATLAB program uses *summation of n vectors*, i.e., "scalar * vector + vector":

MATLAB

```
for j = 1:n,
    y = y + x(j) * A(:,j);
end;
```

Matrix-Matrix Multiplication. Given an $m \times r$ matrix A and an $r \times n$ matrix B we wish to compute the product C of size $m \times n$ s.t.

$$\begin{array}{ccccc} C & = & A & * & B \\ \downarrow & & \downarrow & & \downarrow \\ (m \times n) & & (m \times r) & & (r \times n) \end{array}$$

The matrix C can be represented as

$$C = [c_{ij}], \qquad \begin{array}{l} i = 1, 2, \ldots, m \\ j = 1, 2, \ldots, n \end{array}$$

where

$$c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj}$$

Operation count: mnr multiplications, $mn(r-1)$ additions $\simeq 2mnr$ operations.

MATLAB

```
C = zeros(m,n);
for j = 1:n,
  for i = 1:m,
    for k = 1:r,
      C(i,j) = C(i,j) + A(i,k)*B(k,j);
    end;
  end;
end;
```

Alternative 1: (inner products)

$$A = \begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_m^T \end{pmatrix}, \qquad B = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n) \qquad \Rightarrow c_{ij} = \mathbf{a}_i^T \mathbf{b}_j$$

MATLAB

```
for j = 1:n,
  for i = 1:m,
      C(i,j) = A(i,:)*B(:,j);
  end;
end;
```

Alternative 2: (outer products)

$$A = (\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_r) \qquad B = \begin{pmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \vdots \\ \mathbf{h}_r^T \end{pmatrix}, \qquad \Rightarrow C = \sum_{k=1}^{r} \mathbf{g}_k \mathbf{h}_k^T \leftarrow m \times n \text{ matrix}$$

MATLAB

```
for k = 1:r,
    C = C + A(:,k)*B(k,:);
end;
```

Alternative 3: (matrix-vector products)

$$B = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n) \qquad \Rightarrow C = (A\mathbf{b}_1, A\mathbf{b}_2, \ldots, A\mathbf{b}_n)$$

MATLAB

```
for j = 1:n,
    C(:,j) = A*B(:,j);
end;
```

4.2.1 Matrix Norms, the Inverse of a Matrix and the Sensitivity of Linear Systems

The Inverse of a Matrix. If A is a square matrix of order n, then A is *nonsingular* if there exists a unique $n \times n$ matrix X such that

$$AX = XA = I = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}$$

The inverse, X, is also denoted by A^{-1}. The following property holds for inverse of a product of two matrices:

$$(A * B)^{-1} = B^{-1} * A^{-1}.$$

Note also that

$$(A * B)^T = B^T * A^T.$$

The *identity* matrix I is often expressed as

$$I = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n), \qquad \mathbf{e}_i^T = (0, 0, \ldots, 0, 1, 0, \ldots, 0),$$

i.e., $\mathbf{e}_i$ has a one in the ith position and zero everywhere else.

Vector and matrix norms. Suppose that $\mathbf{x}'$ and $\mathbf{x}''$ are two approximations of $\mathbf{x}$. Let

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \mathbf{x}' = \begin{pmatrix} 1.01 \\ 1.01 \\ 1.01 \end{pmatrix}, \qquad \mathbf{x}'' = \begin{pmatrix} 1 \\ 1.02 \\ 1 \end{pmatrix}.$$

Based on the errors in the approximation, can we decide which one is a better approximation of $\mathbf{x}$?

$$\Delta\mathbf{x}' = \mathbf{x}' - \mathbf{x} = \begin{pmatrix} 0.01 \\ 0.01 \\ 0.01 \end{pmatrix}, \qquad \Delta\mathbf{x}'' = \mathbf{x}'' - \mathbf{x} = \begin{pmatrix} 0.0 \\ 0.02 \\ 0.0 \end{pmatrix}.$$

A norm is a mapping that assigns to each vector a nonnegative real number, i.e., it is a generalization of the scalar absolute value. We define the *maximum norm* or the *infinity norm* of a vector as the largest absolute value of all the elements, i.e.,

$$\|\mathbf{x}\| = \max_{1 \le i \le n} |x_i| = \|\mathbf{x}\|_\infty$$

In MATLAB this is computed by `norm(x,inf)`.

In our example,

$$\|\Delta\mathbf{x}'\| = 0.01 \qquad \|\Delta\mathbf{x}''\| = 0.02.$$

Thus, using the max-norm, we conclude that $\mathbf{x}'$ is a better approximation of $\mathbf{x}$. (The outcome could be different if we use a different norm.)

Norms can be defined for matrices in a similar way. The matrix norm corresponding to the vector max-norm is given by,

$$\|A\| = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| = \|A\|_\infty$$

Clearly, this equals the maximum row sum of the absolute values of the matrix. In MATLAB this is computed by `norm(A,inf)`.

Example 4.1

$$A = \begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix} \qquad \Rightarrow \|A\| = \max\{3, 1\} = 3$$

□

A number of norms can be defined. The most common of these are presented below:

1. $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$.
2. $\|\mathbf{x}\|_2 = (\mathbf{x}^T\mathbf{x})^{\frac{1}{2}}$.
3. $\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$ (max. col. sum).
4. $\|A\|_F = \left[\sum_j \sum_i |a_{ij}|^2 \right]^{\frac{1}{2}}$ (Frobenius).
5. $\|A\|_2 = \max_{\|\mathbf{x}\|_2 = 1} \|A\mathbf{x}\|_2$ (spectral).

Properties of norms.

1. $\|\mathbf{x}\| > 0$; moreover, $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = 0$.
2. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$.
3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.
4. $\|A\| > 0$; moreover, $\|A\| = 0$ if and only if $A = 0$.
5. $\|\alpha A\| = |\alpha|\|A\|$.
6. $\|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|$.
7. $\|A + B\| \leq \|A\| + \|B\|$.

Theorem 4.1 *If $\hat{A}$ is the stored version of the matrix A on a computer with ϵ unit roundoff, then $\hat{A} = A+E$ where $\|E\| \leq \epsilon * \|A\|$.*

Proof We know that

$$\hat{A} = [\hat{a}_{ij}], \qquad \hat{a}_{ij} = fl(a_{ij}) = a_{ij}(1 + \epsilon_{ij})$$

with $|\epsilon_{ij}| \leq \epsilon$ (unit roundoff). Now,

$$\begin{aligned} \|E\| &= \|\hat{A} - A\| \\ &= \max_{1\leq i\leq n} \sum_{j=1}^{n} |\hat{a}_{ij} - a_{ij}| \\ &\leq \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}\epsilon_{ij}| \\ &\leq \epsilon * \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| \\ &= \epsilon * \|A\|. \end{aligned}$$

□

Residual and Error Vectors. Consider the linear system $A\mathbf{x} = \mathbf{f}$. Suppose that the computed solution is $\hat{\mathbf{x}}$. Then we define the following:

- residual: $\mathbf{r} = \mathbf{f} - A\hat{\mathbf{x}}$
- error: $\delta\mathbf{x} = \hat{\mathbf{x}} - \mathbf{x}$

Therefore,

$$\mathbf{r} = \mathbf{f} - A\hat{\mathbf{x}} = A\mathbf{x} - A\hat{\mathbf{x}} = A(\mathbf{x} - \hat{\mathbf{x}}) = -A\delta\mathbf{x}$$

In some situations one requires a small residual, i.e., $\|\mathbf{r}\| \leq$ tolerance, and the size of the error is of no consequence. In other situations, however, one needs to obtain small errors, i.e., $\|\delta\mathbf{x}\| \leq$ small quantity. For example, consider the linear system

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}.$$

The solution is given by

$$\mathbf{x} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Consider an approximate solution $\hat{\mathbf{x}}_a$

$$\hat{\mathbf{x}}_a = \begin{pmatrix} 0.341 \\ -0.087 \end{pmatrix}, \qquad \mathbf{r}_a = \begin{pmatrix} 10^{-6} \\ 0 \end{pmatrix}, \qquad \delta\mathbf{x}_a = \hat{\mathbf{x}}_a - \mathbf{x} = \begin{pmatrix} -0.659 \\ 0.913 \end{pmatrix}.$$

The residual and error norms are $\|\mathbf{r}_a\| = 10^{-6}$ and $\|\delta\mathbf{x}_a\| \simeq 0.9$, respectively. On the other hand, for another computed solution $\hat{\mathbf{x}}_b$,

$$\hat{\mathbf{x}}_b = \begin{pmatrix} 0.999 \\ -1.000 \end{pmatrix}, \qquad \mathbf{r}_b = \begin{pmatrix} 0.000780 \\ 0.000913 \end{pmatrix}, \qquad \delta\mathbf{x}_b = \hat{\mathbf{x}}_b - \mathbf{x} = \begin{pmatrix} -0.001 \\ 0 \end{pmatrix},$$

Clearly, in the first case, the solution is meaningless in spite of the smaller residual norm!

Problem Sensitivity. In the preceding 2×2 system, the matrix

$$A = \begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix}$$

has a determinant, $\det(A) \simeq 10^{-6}$. In fact

$$\tilde{A} = \begin{pmatrix} 0.780 & 0.563001095\ldots \\ 0.913 & 0.659 \end{pmatrix}$$

is exactly singular; i.e., a perturbation of roughly 10^{-6} of the element in the (1,2) position renders the problem insoluble. In other words, the above problem is *ill-conditioned*.

Consider the hypothetical situation in which there is no roundoff error during the entire solution process except when A and $\mathbf{f}$ are stored. Thus, we have the system

$$(A + \Delta A)(\mathbf{x} + \Delta\mathbf{x}) = (\mathbf{f} + \Delta\mathbf{f}).$$

It can be shown that

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \beta * \left[\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta\mathbf{f}\|}{\|\mathbf{f}\|}\right]$$

where the "magnification factor" β is given by

$$\beta = \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\Delta A\|}{\|A\|}}$$

in which

$$\kappa(A)\frac{\|\Delta A\|}{\|A\|} < 1.$$

Here,

$$\kappa(A) = \|A\|\|A^{-1}\|$$

is called the *condition number* of A with respect to the max. norm. Note that $\kappa(A) \geq 1$. To see that, recall that $AA^{-1} = I$, and

$$\|I\| = 1 = \|AA^{-1}\| \leq \|A\|\|A^{-1}\|.$$

Moreover, if

$$\frac{\|\Delta A\|}{\|A\|} \leq \epsilon, \qquad \frac{\|\Delta\mathbf{f}\|}{\|\mathbf{f}\|} \leq \epsilon, \qquad [\epsilon = \text{unit roundoff}]$$

then

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \lesssim \kappa(A) * \epsilon,$$

i.e., $\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|}$ is less than a quantity that is proportional to $\kappa(A) * \epsilon$. Furthermore, if $\kappa(A) * \epsilon$ is close to 1 then A is "numerically" singular.

Example 4.2 Consider the linear system $A\mathbf{x} = \mathbf{f}$ in which the representation error of A is given by

$$\frac{\|\Delta A\|}{\|A\|} \simeq 10^{-16}.$$

If $\kappa(A) \simeq 10^{11}$, give an upper bound on the rel. error in the solution $\|\Delta \mathbf{x}\|/\|\mathbf{x}\|$.

Solution

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\Delta A\|}{\|A\|}} \left[\frac{\|\Delta A\|}{\|A\|}\right] \simeq \frac{10^{11}}{1 - 10^{-5}}(10^{-16}) \simeq 10^{-5}.$$

□

Example 4.3 Let

$$A = \begin{pmatrix} 4.1 & 2.8 \\ 9.7 & 6.6 \end{pmatrix}. \qquad A^{-1} = \begin{pmatrix} -66 & 28 \\ 97 & -41 \end{pmatrix}.$$

Verify that $AA^{-1} = I$. Assuming that the relative perturbation in the right-hand side is

$$\frac{\|\Delta \mathbf{f}\|}{\|\mathbf{f}\|} \simeq 0.000613,$$

obtain an upper bound for the rel. error in the solution $\|\Delta \mathbf{x}\|/\|\mathbf{x}\|$.

Solution We have

$$\begin{array}{lcl} \|A\| & = & \max\{6.9, 16.3\} = 16.3 \\ \|A^{-1}\| & = & \max\{94, 138\} = 138 \\ \kappa(A) & = & (16.3)(138) \simeq 2249.4 \end{array}$$

Now,

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A) \left[\frac{\|\Delta \mathbf{f}\|}{\|\mathbf{f}\|}\right] \simeq 1.38$$

In fact, if

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \hat{\mathbf{x}} = \begin{pmatrix} 0.06 \\ 2.38 \end{pmatrix}, \qquad \text{i.e., } \Delta \mathbf{x} = \begin{pmatrix} -0.94 \\ 1.38 \end{pmatrix},$$

then

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} = 1.38$$

exactly! □

4.3 Solution of Linear System

4.3.1 Gaussian Elimination

Consider the system of equations:

$$\begin{array}{rcrcrcll} 3x_1 & + & 6x_2 & + & 9x_3 & = & 39 & Eq.(1) \\ 2x_2 & + & 5x_2 & - & 2x_3 & = & 3 & Eq.(2) \\ x_1 & + & 3x_2 & - & x_3 & = & 2 & Eq.(3) \end{array}$$

These equations can be solved by the process of *Gaussian Elimination*. Gaussian Elimination consists of two stages - forward elimination and back substitution.

Stage I. Forward elimination. The goal of this stage is to use the equations numbered (1)–(k) to eliminate the terms with unknowns $x_1, x_2, \ldots, x_k$ from equations numbered (k+1)–(n). This is achieved in $n-1$ steps. For our example system, the steps are as follows.

Step 1: Eliminate x_1 from Eq.(2) by adding $-\frac{2}{3}*$ Eq.(1) to Eq.(2), and from Eq.(3) by adding $-\frac{1}{3}*$ Eq.(1) to Eq.(3). The linear system becomes:

$$\begin{array}{rcrcrcrl} 3x_1 & + & 6x_2 & + & 9x_3 & = & 39 & Eq.(1) \\ 0 & + & x_2 & - & 8x_3 & = & -23 & Eq.(2) \\ 0 & + & x_2 & + & 4x_3 & = & -11 & Eq.(3) \end{array}$$

Note that the right-hand side of Eq.(2) and Eq.(3) are also updated similarly.

Step 2: Eliminate x_2 from Eq.(3) by subtracting Eq.(2) from Eq.(3). Now, the linear system is

$$\begin{array}{rcrcrcrl} 3x_1 & + & 6x_2 & + & 9x_3 & = & 39 & Eq.(1) \\ & & x_2 & - & 8x_3 & = & -23 & Eq.(2) \\ & & & & 4x_3 & = & 12 & Eq.(3) \end{array}$$

At the end of forward elimination stage, Eq. (k) has terms with unknowns $x_k, \ldots, x_n$ only, i.e., the linear system consists of an upper triangular matrix. The next stage computes the values of unknowns in reverse order, i.e., $x_n, x_{n-1}, \ldots, x_1$, by using equations in that same order.

Stage II. Back substitution. The linear system can be rewritten as follows:

$$\begin{array}{rcrcrcrl} 3x_1 & + & 6x_2 & + & 9x_3 & = & 39 & Eq.(1) \\ & & x_2 & - & 8x_3 & = & -23 & Eq.(2) \\ & & & & 4x_3 & = & 12 & Eq.(3) \end{array}$$

Step 1: Obtain x_3 from Eq. (3), i.e., $x_3 = 3$.

Step 2: Obtain x_2 from Eq. (2), i.e., $x_2 = 8x_3 - 23 = 1$.

Step 3: Substitute x_2 and x_3 in Eq. (1) to get x_1, i.e.,

$$3x_1 = 39 - 6x_2 - 9x_3 = 6 \qquad \Rightarrow x_1 = 2.$$

The solution obtained at the conclusion of the algorithm is

$$\mathbf{x} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}.$$

Gaussian Elimination in Matrix Notation. Our example can be written in matrix form:

$$\begin{pmatrix} 3 & 6 & 9 \\ 2 & 5 & -2 \\ 1 & 3 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 39 \\ 3 \\ 2 \end{pmatrix},$$

where

$$A = \begin{pmatrix} 3 & 6 & 9 \\ 2 & 5 & -2 \\ 1 & 3 & -1 \end{pmatrix} \qquad \mathbf{f} = \begin{pmatrix} 39 \\ 3 \\ 2 \end{pmatrix}.$$

The operations in the forward elimination stage can also be represented by matrices multiplying A. Let $A_{i-1}\mathbf{x} = \mathbf{f}_{i-1}$ be the linear system at the start of step i, and let M_i be the matrix that is multiplied to A_{i-1} to obtain A_i.

Stage I. Forward elimination.

Step 1: At the start of this step, we have the linear system $A_0\mathbf{x} = \mathbf{f}_0$, where $A_0 = A$ and $\mathbf{f}_0 = \mathbf{f}$. The transformation of the linear system at this step is given by $M_1 A_0 \mathbf{x} = M_1 \mathbf{f}_0$. It is easy to see that

$$\begin{pmatrix} 1 & & \\ -\frac{2}{3} & 1 & \\ -\frac{1}{3} & & 1 \end{pmatrix} \begin{pmatrix} 3 & 6 & 9 \\ 2 & 5 & -2 \\ 1 & 3 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & & \\ -\frac{2}{3} & 1 & \\ -\frac{1}{3} & & 1 \end{pmatrix} \begin{pmatrix} 39 \\ 3 \\ 2 \end{pmatrix},$$

$$\Rightarrow \begin{pmatrix} 3 & 6 & 9 \\ 0 & 1 & -8 \\ 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 39 \\ -23 \\ -11 \end{pmatrix}$$

and thus,

$$M_1 = \begin{pmatrix} 1 & & \\ -\frac{2}{3} & 1 & \\ -\frac{1}{3} & & 1 \end{pmatrix}$$

Step 2: The next transformation is given by $M_2 A_1 \mathbf{x} = \mathbf{f}_1$, where A_1 and $\mathbf{f}_1$ are obtained in the previous step. Observe that

$$\begin{pmatrix} 1 & & \\ & 1 & \\ & -1 & 1 \end{pmatrix} \begin{pmatrix} 3 & 6 & 9 \\ & 1 & -8 \\ & 1 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & & \\ & 1 & \\ & -1 & 1 \end{pmatrix} \begin{pmatrix} 39 \\ -23 \\ -11 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 3 & 6 & 9 \\ & 1 & -8 \\ & & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 39 \\ -23 \\ 12 \end{pmatrix}$$

and thus,

$$M_2 = \begin{pmatrix} 1 & & \\ & 1 & \\ & -1 & 1 \end{pmatrix}$$

The linear system at the end of this step is $A_2\mathbf{x} = \mathbf{f}_2$. The matrix A_2 is an upper triangular matrix, and the corresponding system is solved by back substitution.

Gaussian Elimination for General n. Let us develop the Gaussian Elimination algorithm to solve the linear system $A\mathbf{x} = \mathbf{f}$ where A is a nonsingular matrix of size $n \times n$ and $\mathbf{x}$ and $\mathbf{f}$ are vectors of size n. The forward elimination stage consists of $n-1$ steps to convert A into an upper triangular matrix.

Forward elimination

$A_0 = A$, $\mathbf{f}_0 = \mathbf{f}$
for $k = 1, 2, \ldots, n-1$
    $M_k A_{k-1}\mathbf{x} = M_k \mathbf{f}_{k-1}$
    $[A_k = M_k A_{k-1}, \mathbf{f}_k = M_k \mathbf{f}_{k-1}]$
end;

Back-substitution is used to solve the upper triangular system obtained from the forward elimination stage: $A_{n-1}\mathbf{x} = \mathbf{f}_{n-1}$.

Let us now compute the matrices M_k, $k = 1, 2, \ldots, n-1$. Suppose

$$A_0 = A = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \cdots & a_{2n}^{(0)} \\ a_{31}^{(0)} & a_{32}^{(0)} & \cdots & a_{3n}^{(0)} \\ \vdots & \vdots & & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \cdots & a_{nn}^{(0)} \end{pmatrix}.$$

where the symbol $a_{ij}^{(k)}$ is used to denote the value of the element a_{ij} at the start of the kth step.

Forward elimination step 1. The matrix M_1 that transforms all the elements below a_{11} in the first column of A_0 to zero is given by

$$M_1 = \begin{pmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ -m_{31} & & 1 & & \\ \vdots & & & \ddots & \\ -m_{n1} & & & & 1 \end{pmatrix}$$

where

$$m_{i1} = \frac{a_{i1}^{(0)}}{a_{11}^{(0)}}, \quad i = 2, \ldots, n$$

The transformed matrix $A_1 = M_1 A_0$ is given by

$$A_1 = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

where

$$a_{ij}^{(1)} = a_{ij}^{(0)} - m_{i1} a_{1j}^{(0)}, \quad \begin{matrix} i = 2, \ldots, n \\ j = 1, \ldots, n \end{matrix}$$

It is important to note that the first row remains unchanged.

Forward elimination step 2. At this step, M_2 is used to transforms all the elements below a_{22} in the second column of A_1 to zero. This is given by

$$M_2 = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & -m_{32} & 1 & & \\ & -m_{42} & & 1 & \\ & \vdots & & & \ddots \\ & -m_{n2} & & & 1 \end{pmatrix},$$

where

$$m_{i2} = \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}, \quad i = 3, \ldots, n$$

The transformed matrix $A_2 = M_2 A_1$ is given by

$$A_2 = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix}$$

where

$$a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i2} a_{2j}^{(1)} \ , \quad \begin{matrix} i = 3, \ldots, n \\ j = 2, \ldots, n \end{matrix}$$

Observe that at this step, the first and second rows remain unchanged.

Forward elimination step k. At step k, the matrix A_{k-1} has the following structure

$$A_{k-1} = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1k}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2k}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & a_{33}^{(2)} & \cdots & a_{3k}^{(2)} & \cdots & a_{3n}^{(2)} \\ & & & \ddots & \vdots & & \vdots \\ & 0 & & & a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ & & & & \vdots & & \vdots \\ & & & & a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{pmatrix}$$

and the matrix M_k used to transform all the elements below a_{kk} in the kth column of A_{k-1} to zero is given by

$$M_k = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & -m_{k+1,k} & 1 & & & \\ & & -m_{k+2,k} & & 1 & & \\ & & \vdots & & & \ddots & \\ & & -m_{nk} & & & & 1 \end{pmatrix},$$

where

$$m_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \ , \quad i = k+1, \ldots, n$$

The elements of the transformed matrix $A_k = M_k A_{k-1}$ are

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} - m_{ik} a_{kj}^{(k-1)}, & i, j = k+1, \ldots, n \\ 0, & i \geq k+1, j \leq k \\ a_{ij}^{(k-1)}, & i \leq k \end{cases}$$

The reader is encouraged to verify that the kth column below diagonal in A_k is zero.

At the kth step, the right-hand side is also transformed by M_k, and the resulting vector $\mathbf{f}_k = M_k \mathbf{f}_{k-1}$ is given as

$$f_i^{(k)} = \begin{cases} f_i^{(k-1)} - m_{ik} f_k^{(k-1)}, & i = k+1, \ldots, n \\ f_i^{(k-1)}, & i \leq k \end{cases}$$

The forward elimination algorithm has the potential to break down when $a_{kk}^{(k-1)} = 0$ or $a_{kk}^{(k-1)} \approx 0$. This situation can be rectified by using pivoting techniques; this will be discussed in later sections.

Back substitution. At this stage, we need to solve the upper triangular system $U\mathbf{x} = g$, where

$$U = A_{n-1} = \begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{pmatrix} \qquad \mathbf{g} = \mathbf{f}_{n-1} = \begin{pmatrix} f_1^{(0)} \\ f_2^{(1)} \\ \vdots \\ f_n^{(n-1)} \end{pmatrix}$$

Gaussian Elimination in MATLAB. We now present a MATLAB program for the forward elimination and back substitution stages of Gaussian Elimination.

MATLAB

```
%   FORWARD ELIMINATION
    m = zeros(n,1);
    for k=1:n-1,
%     Compute kth column of M{k}
      m(k+1:n) = A(k+1:n,k)/A(k,k);
%     Multiply M{k} with A{k-1}
      for i=k+1:n,
          A(i,k+1:n)=A(i,k+1:n)-m(i)*A(k,k+1:n);
      end;
%     Update f{k}
      f(k+1:n) = f(k+1:n) - m(k+1:n) * f(k);
    end;
    U = triu(A); % Extract upper triangular part

%   BACK SUBSTITUTION
    x(n) = f(n)/U(n,n);
    for k=n-1:-1:1,
        f(1:k) = f(1:k)-U(1:k,k+1)*f(k+1);
        f(k) = f(k)/U(k,k);
    end;
```

The operation count for the forward elimination stage is given below:
Matrix Operations:

$$\begin{aligned} \text{Div:} \quad & \sum_{k=1}^{n-1}(n-k) = \frac{1}{2}n(n-1) && \text{(compute } M_k\text{)} \\ \text{Mult:} \quad & \sum_{k=1}^{n-1}(n-k)^2 = \frac{1}{6}n(n-1)(2n-1) && \text{(compute } A_k\text{)} \\ \text{Add:} \quad & \sum_{k=1}^{n-1}(n-k)^2 = \frac{1}{6}n(n-1)(2n-1) && \text{(compute } A_k\text{)} \\ \text{Total:} \quad & T_u = \tfrac{2}{3}n^3 - \tfrac{1}{2}n^2 - \tfrac{1}{6}n \end{aligned}$$

Right-Hand Side:

$$\begin{aligned} \text{Mult:} \quad & \sum_{k=1}^{n-1}(n-k) = \frac{1}{2}n(n-1) \\ \text{Add:} \quad & \sum_{k=1}^{n-1}(n-k) = \frac{1}{2}n(n-1) \\ \text{Total:} \quad & T_f = n(n-1) \end{aligned}$$

The operation count for the back substitution stage is given below:

$$\begin{array}{ll} \text{Div:} & n \\ \text{Mult:} & \displaystyle\sum_{k=1}^{n-1}(n-k) = \frac{1}{2}n(n-1) \\ \text{Add:} & \displaystyle\sum_{k=1}^{n-1}(n-k) = \frac{1}{2}n(n-1) \\ \text{Total:} & T_b = n^2 \end{array}$$

Recall the formulas for summation:

$$\sum_{j=1}^{m} j = \frac{1}{2}m(m+1), \qquad \sum_{j=1}^{m} j^2 = \frac{1}{6}m(m+1)(2m+1)$$

Thus, the *overall time* for the solution of a linear system of order n using Gaussian Elimination is

$$T^* = T_u + T_f + T_b = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

In general, we say that

- Gaussian Elimination takes $O(n^3)$ operations.
- Back-substitution takes $O(n^2)$ operations.

4.3.2 Triangular Factorizations: The LU Decomposition

Let us again consider the system $A\mathbf{x} = \mathbf{f}$. The forward elimination process may be written as:

$$M_{n-1}M_{n-1}\cdots M_2M_1A = A_{n-1} = U.$$

Multiplying both sides by the matrix $M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}$, we have

$$A = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}U$$

Recall that each M_k is a lower triangular matrix. In fact, M_k is a *unit* lower triangular matrix which is a lower triangular matrix with 1's on diagonal. Furthermore, each M_k^{-1} is unit lower triangular, and so is the product matrix $M_1^{-1}\cdots M_{n-1}^{-1}$. Let L be the unit lower triangular matrix defined as

$$L = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1},$$

then we have

$$A = LU.$$

With this result, we have expressed a general nonsingular matrix A as a product of a unit lower and an upper triangular matrix. The result would be much more useful if we could construct L easily.

We now describe how to obtain L from the forward elimination stage. The following assertions are easy to verify.

- It can be shown that

$$M_k^{-1} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & m_{k+1,k} & 1 & & \\ & & m_{k+2,k} & & 1 & \\ & & \vdots & & & \ddots \\ & & m_{nk} & & & 1 \end{pmatrix},$$

  i.e., M_k^{-1} is identical to M_k with the exception that the sign on m_{ik} is reversed.

- It can be also be shown that

$$M_k^{-1}M_{k+1}^{-1} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & m_{k+1,k} & 1 & & \\ & & m_{k+2,k} & m_{k+2,k+1} & \ddots & \\ & & \vdots & \vdots & & \\ & & m_{nk} & m_{nk+1} & & 1 \end{pmatrix}$$

- Finally, using the previous result, it can be shown that

$$L = \begin{pmatrix} 1 & & & & & \\ m_{21} & 1 & & & & \\ m_{31} & m_{32} & 1 & & & \\ \vdots & \vdots & & \ddots & & \\ \vdots & \vdots & & & 1 & \\ m_{n1} & m_{n2} & & & m_{n,n-1} & 1 \end{pmatrix}.$$

Solution of Linear Systems Using LU Decomposition. The factorization of a matrix A into the triangular factors L and U can be used to solve the linear system $A\mathbf{x} = \mathbf{f}$. Since $A = LU$, we need to solve the system

$$LU\mathbf{x} = \mathbf{f}.$$

Assuming $\mathbf{y} = U\mathbf{x}$, the solution consists of the following two steps.

1. Solve $L\mathbf{y} = \mathbf{f}$.
2. Solve $U\mathbf{x} = \mathbf{y}$.

The reader can verify that the first step corresponds to the transformation of $\mathbf{f}$ during the forward elimination stage, and that indeed $\mathbf{y} = \mathbf{f}_{n-1}$.

The operation count for LU decomposition of a matrix is

$$T_u < \frac{2}{3}n^3$$

and for the solution of the triangular systems with matrices L and U is

$$T_s < 2n^2.$$

The advantage of computing LU decomposition over Gaussian Elimination is evident when solving a system with multiple right-hand sides. For example, when solving $AX = F$, where F is an $n \times m$ matrix, the operation count is

$$T_{LU}^* = T_u + T_s \approx \frac{2}{3}n^3 + 2mn^2$$

whereas using Gaussian Elimination for each right-hand side, the operation count would have been

$$T_{GE}^* = \frac{2}{3}mn^3,$$

which is a factor of m more!

Example 4.4 Compute the LU decomposition of A from our earlier example:

$$A = \begin{pmatrix} 3 & 6 & 9 \\ 2 & 5 & -2 \\ 1 & 3 & -1 \end{pmatrix}$$

We know that

$$M_1 = \begin{pmatrix} 1 & & \\ -\frac{2}{3} & 1 & \\ -\frac{1}{3} & & 1 \end{pmatrix}, \qquad M_2 = \begin{pmatrix} 1 & & \\ & 1 & \\ & -1 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 3 & 6 & 9 \\ 0 & 1 & -8 \\ 0 & 0 & 4 \end{pmatrix}$$

Thus,

$$L = M_1^{-1} M_2^{-1} = \begin{pmatrix} 1 & & \\ \frac{2}{3} & 1 & \\ \frac{1}{3} & & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ \frac{2}{3} & 1 & \\ \frac{1}{3} & 1 & 1 \end{pmatrix}$$

□

4.3.3 Pivoting for Stability

Gaussian elimination, as described above, will break down if $a_{kk}^{(k-1)} = 0$ for any $k = 1, \ldots, n-1$. the following example illustrates this for a 2×2 linear system.

Example 4.5 Consider the system

$$\begin{aligned} 0 * x_1 + 1 * x_2 &= 1 \\ 1 * x_1 + 1 * x_2 &= 2 \end{aligned}$$

or

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

with the exact solution:

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Clearly, Gaussian elimination fails in the first step since $a_{11}^{(0)} = 0$. If we exchange the rows, however, we get

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

and proceed with back substitution. Row 2, which is now row 1, is called the "pivotal row". □

An important concept in numerical computations is the following:

> If a mathematical process fails when a particular parameter achieves a certain value, then the corresponding numerical algorithm will most likely be unstable when his parameter is *near* the critical value.

A slight perturbation in the linear system from the previous example serves to illustrate this point.

Example 4.6 Consider the linear system $A\mathbf{x} = \mathbf{f}$ where

$$A = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 1 \end{pmatrix} \qquad \mathbf{f} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Solve the above linear system with Gaussian elimination, using 3-digit decimal arithmetic with rounding.

Solution The first step in forward elimination gives

$$m_{21} = 1/10^{-4} = 10^4, \qquad M_1 = \begin{pmatrix} 1 & 0 \\ -10^4 & 1 \end{pmatrix}$$

and

$$fl(M_1 A) = \begin{pmatrix} 10^{-4} & 1 \\ 0 & fl(1 - 10^4) \end{pmatrix}, \qquad fl(M_1 \mathbf{f}) = \begin{pmatrix} 1 \\ fl(2 - 10^4) \end{pmatrix}.$$

But,

$$\begin{aligned} fl(1 - 10^4) &= fl[10^5(-0.100000) + 10^1(0.100000)] \\ &= fl[10^5(-0.100000 + .000010)] \\ &= fl[10^5(-.099990)] \\ &= -10^5(0.100) = -10^4, \\ fl(2 - 10^4) &= -10^4. \end{aligned}$$

Thus, we have

$$\begin{pmatrix} 10^{-4} & 1 \\ 0 & -10^4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -10^4 \end{pmatrix}$$

and the approximate solution is

$$\hat{\mathbf{x}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The exact solution, however, is given by

$$\mathbf{x} = \begin{pmatrix} 1.000100010001 \\ 0.999899989998 \end{pmatrix}.$$

Now, if we exchange rows so that $|m_{21}| < 1$, i.e., $m_{21} = 10^{-4}$, then

$$fl(A_1) = \begin{pmatrix} 1 & 1 \\ 0 & fl(1 - 10^{-4}) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad fl(\mathbf{f}_1) = \begin{pmatrix} 2 \\ 1 \end{pmatrix},$$

and the solution is

$$\hat{\mathbf{x}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

which is a much better computed solution! □

> The pivoting strategy adopted in this example is known as **partial pivoting**. Such a strategy exchanges rows so that all multipliers m_{ik} are less than or equal to 1 in magnitude.

Observe that in Example 6 $m_{21} = 10^4$ is very large compared to the sizes of the original elements of the linear system. As a result, elements of A_1 (or A_k, in general) can grow unacceptably large. These, in turn, can result in loss of accuracy in fixed precision arithmetic, giving inaccurate results.

Partial Pivoting Strategy. At the k^{th} step of Gaussian elimination, find row i, $i \geq k$, for which

$$|a_{ik}^{(k-1)}| \geq |a_{jk}^{(k-1)}|, \qquad j = k, \ldots, n$$

and exchange it with row k. This ensures that $|m_{jk}| < 1$ at this step. Such a row permutation is applied at each step to ensure that $|m_{ik}| < 1$ for all k. In most cases this keeps the growth of elements of A_k, $k = 1, 2, \ldots, n-1$ within acceptable limits.

Triangular Factorization with Partial Pivoting. To demonstrate this strategy, let us consider the LU decomposition of the matrix

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 5 & -2 \\ 3 & 6 & 9 \end{pmatrix}.$$

We will also use a vector, $IPIV$, to keep track of the row exchanges made during the factorization. The steps in the forward elimination proceed as follows.

Step 1: Row 3 is the pivotal row since

$$\max_{1 \le i \le 3} |a_{i1}^{(0)}| = a_{31}^{(0)} = 3;$$

therefore, we exchange rows 1 and 3, and initialize the first element of $IPIV$ with 3. Now, we have

$$\tilde{A}_0 = \begin{pmatrix} 3 & 6 & 9 \\ 2 & 5 & -2 \\ 1 & 0 & 1 \end{pmatrix}, \qquad IPIV = \begin{pmatrix} 3 \\ \bullet \\ \bullet \end{pmatrix}.$$

The transformed matrix $A_1 = M_1 \tilde{A}_0$ is given by

$$A_1 = M_1 \tilde{A}_0 = \begin{pmatrix} 1 & & \\ -\frac{2}{3} & 1 & \\ -\frac{1}{3} & & 1 \end{pmatrix} \begin{pmatrix} 3 & 6 & 9 \\ 2 & 5 & -2 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 9 \\ 0 & 1 & -8 \\ 0 & -2 & -2 \end{pmatrix}$$

Storing the multipliers $-m_{21}$ and $-m_{31}$ in A_1, we get

$$A_1 = \begin{pmatrix} 3 & 6 & 9 \\ \boxed{\frac{2}{3}} & 1 & -8 \\ \boxed{\frac{1}{3}} & -2 & -2 \end{pmatrix}$$

Step 2: Again, row 3 is the pivotal row since

$$\max_{2 \le i \le 3} |a_{i2}^{(1)}| = a_{31}^{(1)} = 2;$$

therefore, we exchange rows 2 and 3, and initialize the second element of $IPIV$ with 3. Now, we have

$$\tilde{A}_1 = \begin{pmatrix} 3 & 6 & 9 \\ \boxed{\frac{2}{3}} & -2 & -2 \\ \boxed{\frac{1}{3}} & 1 & -8 \end{pmatrix}, \qquad IPIV = \begin{pmatrix} 3 \\ 3 \\ \bullet \end{pmatrix}.$$

The transformed matrix $A_2 = M_2 \tilde{A}_1$ is given by

$$A_2 = M_2 \tilde{A}_1 = \begin{pmatrix} 1 & & \\ & 1 & \\ & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 3 & 6 & 9 \\ & -2 & -2 \\ & 1 & -8 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 9 \\ & -2 & -2 \\ & 0 & -9 \end{pmatrix}$$

Storing the multiplier $-m_{32}$ in A_2, we get

$$\tilde{A}_2 = \begin{pmatrix} 3 & 6 & 9 \\ \boxed{\frac{2}{3}} & -2 & -2 \\ \boxed{\frac{1}{3}} & \boxed{-\frac{1}{2}} & -9 \end{pmatrix}$$

Instead of the factorization $A = LU$, we have computed a factorization of a matrix whose rows have been permuted according to the vector $IPIV$. In other words, we have

$$PA = L * U,$$

where

$$L = \begin{pmatrix} 1 & & \\ \frac{2}{3} & 1 & \\ \frac{1}{3} & -\frac{1}{2} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 3 & 6 & 9 \\ & -2 & -2 \\ & & -9 \end{pmatrix}, \qquad P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Note that the permutation matrix P is obtained by applying the row exchanges stored in $IPIV$ to the identity matrix I. The exchanges *must* be applied sequentially in the order specified by $IPIV(i)$, for $i = 1, \ldots, n-1$.

$$IPIV = \begin{pmatrix} 3 \\ 3 \\ \bullet \end{pmatrix} \begin{matrix} \leftarrow \text{row to be exchanged with row 1} \\ \leftarrow \text{row to be exchanged with row 2} \\ \leftarrow \text{not used} \end{matrix}$$

Now, let us solve the linear system $A\mathbf{x} = \mathbf{f}$, where A is given above, and

$$\mathbf{f} = \begin{pmatrix} 2 \\ 5 \\ 18 \end{pmatrix}.$$

We know that $PA\mathbf{x} = P\mathbf{f}$, or $LU\mathbf{x} = P\mathbf{f}$, and so we solve

$$LU\mathbf{x} = P\mathbf{f} = \tilde{\mathbf{f}} = \begin{pmatrix} 18 \\ 2 \\ 5 \end{pmatrix}.$$

Forward Sweep: Solve $L\mathbf{y} = \hat{\mathbf{f}}$

$$\begin{pmatrix} 1 & & \\ \frac{1}{3} & 1 & \\ \frac{2}{3} & -\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 18 \\ 2 \\ 5 \end{pmatrix} \quad \Rightarrow \begin{matrix} y_1 = 18 & \\ y_2 + 6 = 2 & \Rightarrow y_2 = -4 \\ y_3 + 2 + 12 = 5 & \Rightarrow y_3 = -9 \end{matrix}$$

Backward Sweep: Solve $U\mathbf{x} = \mathbf{y}$

$$\begin{pmatrix} 3 & 6 & 9 \\ & -2 & -2 \\ & & -9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 18 \\ -4 \\ -9 \end{pmatrix} \quad \Rightarrow \begin{matrix} x_3 = 1 & \\ -2x_2 - 2 = -4 & \Rightarrow x_2 = 1 \\ 3x_1 + 6 + 9 = 18 & \Rightarrow x_1 = 1 \end{matrix}$$

Therefore,

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Next, we give the MATLAB programs for the triangular factorization and solution of triangular systems.

```
function [L,U,p] = lutx(A)
%LUTX Triangular factorization, textbook version
%   [L,U,p] = lutx(A) produces a unit lower triangular matrix L,
%   an upper triangular matrix U, and a permutation vector p,
%   so that L*U = A(p,:)

[n,n] = size(A);
```

```
p = (1:n)'n;

for k = 1:n-1

   % Find index of largest element below diagonal in k-th column
   [r,m] = max(abs(A(k:n),k)));
   m = m+k-1;

   % Skip elimination if column is zero
   if (A(m,k) ~= 0)

   % Swap pivot row
   if (m ~= k)
      A([k m],:) = A([m k],:);
      p([k m]) = p([m k]);
   end

      % Compute multipliers
      A(k+1:n,k) = A(k+1:n,k)/A(k,k);

      % Update the remainder of the matrix
      A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n,k)*A(k,k+1:n);

      % rank-1
      update
   end
end

% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
```

Solving unit lower triangular systems

```
function x = forward_sweep(L,x)
   [n,n] = size(L);
   for k = 2:n
      x(k:n) = x(k:n) - x(k-1)*L(k:n,k-1);
end
```

Solving upper triangular systems

```
function x = back_sweep (U,x)
   [n,n] = size (U);
   for k = n:-1:1
      x(k) = x(k)/U(k,k);
      x(1:k-1) = x(1:k-1) - x(k)*U(1:k-1,k);
end
```

The MATLAB code that solves a linear system $A\mathbf{x} = \mathbf{f}$ using triangular factorization with partial pivoting is given as

MATLAB

```
[L,U,p] = lutx(A);      % p has row permutation
y = forward_sweep(L,f(p)); % f(p) is permuted rhs
x = back_sweep(U,y);
```

Let us pay a closer look at `lutx(A)`. At stage k, after partial pivoting, let $\tilde{A}_k$ be of the form

$$\tilde{A}_k = \begin{pmatrix} \star & \star & \star & \star & \star & \star \\ o & \star & \star & \star & \star & \star \\ o & o & a_{33}^{(k)} & \star & \star & \star \\ o & o & a_{43}^{(k)} & \star & \star & \star \\ o & o & a_{53}^{(k)} & \star & \star & \star \\ o & o & a_{63}^{(k)} & \star & \star & \star \end{pmatrix}, \qquad M_k = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & \star & 1 & & \\ & & \star & & 1 & \\ & & \star & & & 1 \end{pmatrix},$$

i.e.,

$$\tilde{A}_k = \begin{pmatrix} U_1 & F_1 \\ 0 & \tilde{B}_k \end{pmatrix} \qquad M_k = \begin{pmatrix} I_2 & 0 \\ 0 & \tilde{M}_k \end{pmatrix}$$

Therefore,

$$A_{k+1} = M_k \tilde{A}_k = \begin{pmatrix} U_1 & F_1 \\ 0 & \tilde{M}_k B_k \end{pmatrix}$$

Let us focus on $\tilde{M}_k * B_k$:

$$\begin{pmatrix} 1 & 0 \\ \mathbf{m} & I_3 \end{pmatrix} \begin{pmatrix} a_{33}^{(k)} & \mathbf{b}^T \\ \mathbf{c} & E_k \end{pmatrix} = \begin{pmatrix} a_{33}^{(k)} & \mathbf{b}^T \\ a_{33}^{(k)}\mathbf{m} + \mathbf{c} & E_k + \mathbf{m}\mathbf{b}^T \end{pmatrix},$$

Note that $\mathbf{m} = -\mathbf{c}/a_{33}^{(k)}$; in fact, $-\mathbf{m}$ is computed by the MATLAB statement `A(k+1:n,k)=A(k+1:n,k)/A(k,k)`. Also, $E_k + \mathbf{m}\mathbf{b}^T$ is computed by the statement `A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-A(k+1:n,k)*A(k,k+1:n)`.

Example 4.7 How should one compute $\alpha = \mathbf{c}^T A^{-1}\mathbf{d}$?

Solution Since

$$PA = LU,$$

taking the inverse of both sides, we get

$$A^{-1}P^{-1} = U^{-1}L^{-1} \qquad \Rightarrow A^{-1} = U^{-1}L^{-1}P.$$

Therefore, $\alpha = \mathbf{c}^T U^{-1}L^{-1}P\mathbf{d}$. Let $\mathbf{g} = P\mathbf{d}$,

- solve $L\mathbf{y} = \mathbf{g}$,
- solve $U\mathbf{x} = \mathbf{y}$, and
- compute $\alpha = \mathbf{c}^T\mathbf{x}$.

Alternately, using MATLAB functions developed earlier,

MATLAB

```
[L,U,p] = lutx(A);
g = d(p);
y = forward_sweep(L,g);
x = back_sweep(U,y);
alpha = c'*x;
```

□

Iterative Refinement. An important technique to improve the accuracy of a computed solution relies on repeated solution of linear systems with successive residuals as the right hand side. In other words, one solves the linear system $A\mathbf{x}^{(2)} = \mathbf{f} - A\mathbf{x}^{(1)}$, where $\mathbf{x}^{(1)}$ is the initial solution. The procedure is outlined below:

Iterative Refinement

1. Compute $PA = LU$.
2. Solve $LU\mathbf{x} = P\mathbf{f} = \hat{\mathbf{f}}$, i.e.,
   $L\mathbf{y} = \hat{\mathbf{f}}$,
   $U\mathbf{x} = \mathbf{y}$ $\Rightarrow \hat{\mathbf{x}}$ [computed solution].
3. Let $\mathbf{x}^{(1)} = \hat{\mathbf{x}}$; the residual is $\mathbf{r} = \mathbf{f} - A\mathbf{x}^{(1)} = A(\mathbf{x} - \mathbf{x}^{(1)}) = -A\delta\mathbf{x}^{(1)}$.
4. Solve $A\delta\mathbf{x}^{(1)} = -\mathbf{r}$ for error in solution.
5. Compute improved solution: $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \delta\mathbf{x}^{(1)}$ (repeat if needed)

Example 4.8 Consider the system $A\mathbf{x} = \mathbf{f}$, i.e.,

$$\begin{pmatrix} 20 & 200000 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 200000 \\ 4 \end{pmatrix}$$

The triangular factorization yields

$$L = \begin{pmatrix} 1 & 0 \\ 0.1 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 20 & 200000 \\ 0 & -20000 \end{pmatrix}, \qquad IPIV = \begin{pmatrix} 1 \\ \bullet \end{pmatrix}$$

Thus, the computed solution is

$$\mathbf{x}^{(1)} = \begin{pmatrix} 0 \\ 1.0 \end{pmatrix}.$$

Using 4-digit decimal arithmetic the residual is computed as

$$\mathbf{r}^{(1)} = \mathbf{f} - A\mathbf{x}^{(1)} = \begin{pmatrix} 0 \\ 2.0 \end{pmatrix}.$$

Solving $A\delta\mathbf{x}^{(1)} = -\mathbf{r}^{(1)}$ for the error $\delta\mathbf{x}^{(1)}$, we get

$$\delta\mathbf{x}^{(1)} = \begin{pmatrix} -1.0 \\ 0.0001 \end{pmatrix} \qquad \Rightarrow \mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \delta\mathbf{x}^{(1)} = \begin{pmatrix} 1.0 \\ 0.9999 \end{pmatrix}.$$

Compute $\boldsymbol{r}^{(2)}$ and show it is much smaller than $\boldsymbol{r}^{(1)}$. □

4.4 Least Squares Problems

Least Squares Fitting. LS fitting problems arise in many applications. Consider a *linear model* for computing the height of a plant as a linear function of four nutrients, i.e.,

$$h = a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4,$$

where h is the plant height, and a_1, a_2, a_3, and a_4 are nutrient concentrations in the soil. The unknown parameters x_1, x_2, x_3, and x_4 can be determined from m observations of height and concentrations for $m > 4$. Typically, a large number of experiments are conducted to obtain the values for h, a_1, a_2, a_3, and a_4. Suppose the observations for the ith experiment are given by h_i, a_{i1}, a_{i2}, a_{i3}, a_{i4}. If the model is perfect, then

$$h_i = a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + a_{i4}x_4$$

for $i = 1, 2, \ldots, m$. In matrix notation we would then have $\mathbf{h} = A\mathbf{x}$, or,

$$\begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix},$$

where A is an $m \times 4$ matrix.

In general, however, the model will not be perfect, making it impossible to find that vector $\mathbf{x}$ which exactly satisfies the above equation. In such situations, the main objective is to obtain an $\mathbf{x}$ for which

$$\|\mathbf{h} - A\mathbf{x}\|_2$$

is minimized. Recall that for a vector $\mathbf{y}$, $\|\mathbf{y}\|_2^2 = \mathbf{y}^T\mathbf{y}$.

LS fitting arises also in the approximation of known functions. Suppose we wish to approximate $f(x) = \sqrt{x}$ on [0.25,1] via a linear approximation of the form

$$l(y) = \alpha + \beta y$$

Thus, given the table

x	x_1	x_2	$\cdots$	x_m
$\sqrt{x}$	f_1	f_2	$\cdots$	f_m

in which $f_i = \sqrt{x_i}$, determine the best least squares fit parameters α and β. Let,

$$\mathbf{r} = \mathbf{f} - A\mathbf{a} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{pmatrix} - \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix},$$

then we seek to determine $\mathbf{a}$ so that $\|\mathbf{r}\|_2^2 = \|\mathbf{f} - A\mathbf{a}\|_2^2$ is a minimum.

Least Squares Problem. The least squares problem is stated as follows:

$$\min_{\mathbf{x}} \|\mathbf{f} - A\mathbf{x}\|_2^2$$

where A is an $m \times n$ matrix with $m \geq n$, and $\mathbf{f}$, $\mathbf{x}$ are vectors of order m and n, respectively. Also, we assume that A has linearly independent columns, i.e., A is of full column-rank.

We seek a factorization of A of the form,

$$Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

where R is an $n \times n$ upper triangular matrix, and Q is orthogonal, i.e., $QQ^T = Q^TQ = I_m$. Since the vector 2-norm is invariant under orthogonal transformations, i.e.,

$$\|Q^T\mathbf{r}\|_2^2 = (Q^T\mathbf{r})^T(Q^T\mathbf{r}) = \mathbf{r}^TQQ^T\mathbf{r} = \mathbf{r}^T\mathbf{r} = \|\mathbf{r}\|_2^2$$

we have,

$$\begin{aligned} \|\mathbf{f} - A\mathbf{x}\|_2^2 &= \|Q^T(\mathbf{f} - A\mathbf{x})\|_2^2 \\ &= \left\| \begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \end{pmatrix} - \begin{pmatrix} R \\ 0 \end{pmatrix} \mathbf{x} \right\|_2^2 \\ &= \left\| \begin{pmatrix} \mathbf{g}_1 - R\mathbf{x} \\ \mathbf{g}_2 \end{pmatrix} \right\|_2^2 \end{aligned}$$

This quantity is minimized for $\mathbf{x} = \hat{\mathbf{x}}$, for which $R\hat{\mathbf{x}} = \mathbf{g}_1$; then $\hat{\mathbf{x}}$ is called the *least squares solution*, and $\|\hat{\mathbf{r}}\|_2 = \|\mathbf{f} - A\mathbf{x}\|_2 = \|\mathbf{g}_2\|_2$.

Next, we describe how to construct the *orthogonal* factorization

$$Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

Consider the 2×2 orthogonal matrix

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

in which $c = \cos\theta$ and $s = \sin\theta$. It can be easily verified that

$$GG^T = G^T G = \begin{pmatrix} c^2 + s^2 & 0 \\ 0 & c^2 + s^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

In fact, given a vector $\mathbf{x}$ of size 2, we can construct G such that the vector $\mathbf{y} = G\mathbf{x}$ has zero as the second element. To do this, let

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ 0 \end{pmatrix},$$

then

$$-sx_1 + cx_2 = 0 \qquad \Rightarrow \frac{s}{c} = \frac{x_2}{x_1}$$

Using the identity $c^2 + s^2 = 1$, we have

$$1 + \frac{x_2^2}{x_1^2} = \frac{1}{c^2}$$

that further gives the following expression for c and s:

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \qquad s = \frac{x_2}{\sqrt{x_1^2 + x_2^2}}$$

Note that

$$y_1 = cx_1 + sx_2 = \sqrt{x_1^2 + x_2^2} = \|\mathbf{x}\|_2.$$

Now, we can construct a 2×2 orthogonal matrix that introduces a zero in the second position of a vector of size 2. We use this idea to introduce zeros in the lower triangular part of A to obtain the required upper triangular form by repeated multiplication with suitable 2×2 orthogonal matrices. The matrix

$$G^{(1)}_{m-1,m} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & c^{(1)}_{m-1,m} & s^{(1)}_{m-1,m} \\ & & & -s^{(1)}_{m-1,m} & c^{(1)}_{m-1,m} \end{pmatrix}$$

is used to introduce a zero in the last position, i.e., mth location in the first column. After that, the matrix

$$G^{(1)}_{m-2,m-1} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & c^{(1)}_{m-2,m-1} & s^{(1)}_{m-2,m-1} & \\ & & & -s^{(1)}_{m-2,m-1} & c^{(1)}_{m-2,m-1} & \\ & & & & & 1 \end{pmatrix}$$

is used to introduce a zero in the $(m-1)$th position of the first column. The process of introducing zeros in the first column is continued by forming the matrix $G^{(1)}_{j-1,j}$, for $j = m-2, m-3, \ldots, 2$, that introduce zeros in the jth location. Let us define Q_1^T to be the product of these matrices, i.e.,

$$Q_1^T = G^{(1)}_{1,2} \cdots G^{(1)}_{m-2,m-1} G^{(1)}_{m-1,m}$$

The same technique can be used to zero out the elements below the diagonal in the second column, followed by the third column all the way up to the last column. Note that introducing zeros in this manner doesn't effect the zeros introduced earlier. Let $G^{(k)}_{j-1,j}$ be the matrix used to zero out the element in the (j,k) position. Let us also define

$$\begin{aligned} Q_2^T &= G^{(2)}_{2,3} \cdots G^{(2)}_{m-2,m-1} G^{(2)}_{m-1,m} \\ \vdots \quad &\quad \vdots \\ Q_n^T &= G^{(n)}_{n-1,n} \cdots G^{(n)}_{m-2,m-1} G^{(n)}_{m-1,m}, \end{aligned}$$

then,

$$Q_n^T \cdots Q_2^T Q_1^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

Note that since each $G^{(k)}_{j-1,j}$ is orthogonal, each Q_k^T is also orthogonal. Finally, defining

$$Q^T = Q_n^T \cdots Q_2^T Q_1^T,$$

we have the required orthogonal matrix that transforms A into an upper triangular matrix. The same transformations applied to $\mathbf{f}$ give

$$\mathbf{g} = Q_n^T \cdots Q_2^T Q_1^T \mathbf{f}.$$

For example, if A is a 5×3 matrix (m=5, n=3), Q_1^T introduces zeros below the diagonal in the first column of A, Q_2^T introduces zeros below the diagonal in the second column, and so on until

$$Q_n^T \cdots Q_1^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

In other words, zeros in locations 1, 2, 3, and 4 are introduced by Q_1^T, zeros in locations 5, 6, and 7 are introduced by Q_2^T, and zeros in locations 8 and 9 are introduced by Q_3^T.

$$\begin{pmatrix} \star & \star & \star \\ \boxed{4} & \star & \star \\ \boxed{3} & \boxed{7} & \star \\ \boxed{2} & \boxed{6} & \boxed{9} \\ \boxed{1} & \boxed{5} & \boxed{8} \end{pmatrix}$$

MATLAB

```
function [xLS,res] = LSq (A,f)
[m,n] = size(A);
for j=1:n,
  for i=m:-1:j+1
      x = [A(i-1,j),A(i,j)];
      [c,s] = x/norm(x);
      A(i-1:i,j:n) = [c s; -s c]*A(i-1:i,j:n);
```

```
        f(i-1:i) = [c s; -s c]*f(i-1:i);
      end;
    end;
    xLS = back_sweep(A(1:n,1:n), f(1:n))
    if m=n,
       res = 0;
    else
       res = norm(f(n+1:m));
    end;
```

Chapter 5

Polynomial Interpolation

In this chapter, we consider the important problem of approximating a function $f(x)$, whose values at a set of distinct points $x_0, x_1, x_2, \ldots, x_n$ are known, by a polynomial $P(x)$ such that $P(x_i) = f(x_i)$, $i = 0, 1, 2, \ldots, n$. Such a polynomial is known as an interpolating polynomial. An approximation of this function is desirable if $f(x)$ is difficult to evaluate or manipulate like the error function

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt,$$

for example. Note that polynomials are easily evaluated, differentiated or integrated. Another purpose of such an approximation is that a very long table of function values $f(x_i)$ may be replaced by a short table and a compact interpolation subroutine.

5.1 Linear Interpolation

Let $f(x)$ be given at two distinct points x_i and x_{i+1}. Now, given $f(x_i) = f_i$ and $f(x_{i+1}) = f_{i+1}$, we wish to determine the interpolating polynomial $P_1(x)$ of degree 1 (i.e., a straight line) that approximates $f(x)$ on the interval $[x_i, x_{i+1}]$.

Let $P_1(x) = \alpha x + \beta$. From the conditions $P_1(x_i) = f_i$ and $P_1(x_{i+1}) = f_{i+1}$ we obtain the two equations

$$\begin{aligned} \alpha x_i + \beta &= f_i \\ \alpha x_{i+1} + \beta &= f_{i+1}, \end{aligned}$$

i.e.,

$$\begin{pmatrix} x_i & 1 \\ x_{i+1} & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix}.$$

This system is clearly nonsingular since $x_i \neq x_{i+1}$, and it has a unique solution:

$$\alpha = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} \qquad \beta = \frac{f_i x_{i+1} - f_{i+1} x_i}{x_{i+1} - x_i}.$$

Therefore,

$$P_1(x) = f_i + \left(\frac{f_{i+1} - f_i}{x_{i+1} - x_i} \right) (x - x_i).$$

Example 1 Using linear interpolation, approximate sin 6.5° from a table which gives

(a) $\sin 6^\circ = 0.10453$; $\sin 7^\circ = 0.12187$.

(b) $\sin 0^\circ = 0$; $\sin 10^\circ = 0.17365$.

Solution

(a) $P_1(6.5) = 0.10453 + \left(\frac{0.12187-0.10453}{7-6}\right) \approx 0.11320$.

(b) $P_1(6.5) = 0 + \frac{0.17365}{10}(6.5) \approx 0.11287$.

The first answer is correct to 5 decimals whereas the second answer is correct only to 2 decimals! □

Conclusion: Linear interpolation is suitable only over small intervals.

5.2 Polynomial Interpolation

Since linear interpolation is not adequate unless the given points are closely spaced, we consider higher order interpolating polynomials. Let $f(x)$ be given at the selected sample of $(n+1)$ points: $x_0 < x_1 < \cdots < x_n$, i.e., we have $(n+1)$ pairs (x_i, f_i), $i = 0, 1, 2, \ldots, n$. The objective now is to find the *lowest degree* polynomial that passes through this selected sample of points.

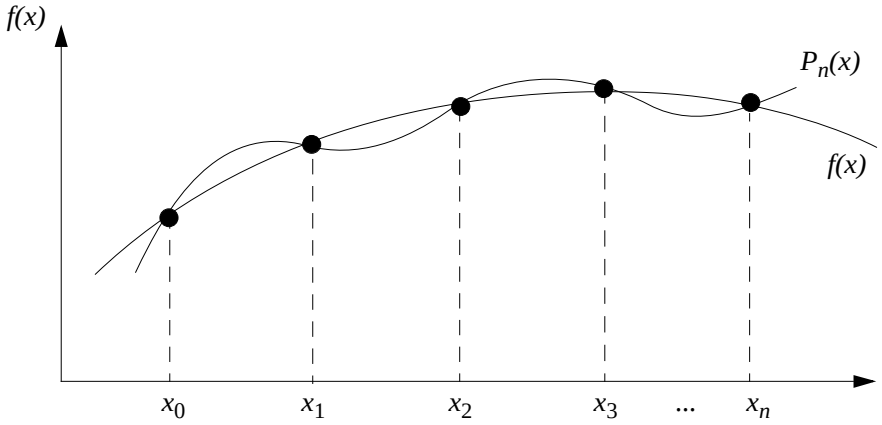


Figure 5.1: Interpolating the function $f(x)$ by a polynomial of degree n, $P_n(x)$.

Consider the nth degree polynomial

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

We wish to determine the coefficients a_j, $j = 0, 1, \ldots, n$, such that

$$P_n(x_j) = f(x_j), \qquad j = 0, 1, 2, \ldots, n.$$

These $(n+1)$ conditions yield the linear system

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n &= f_0 \\
a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n &= f_1 \\
a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_n x_2^n &= f_2 \\
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots & \\
a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n &= f_n
\end{aligned}$$

or $V\mathbf{a} = \mathbf{f}$, where $\mathbf{a}^T = (a_0, a_1, \ldots, a_n)$, $\mathbf{f}^T = (f_0, f_1, \ldots, f_n)$, and V is an $(n+1) \times (n+1)$ matrix given by

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}.$$

The matrix V is known as the Vandermonde matrix. It is nonsingular with nonzero determinant

$$\det(V) = \prod_{\substack{j=0 \\ i>j}}^{n-1} (x_i - x_j).$$

For $n = 3$, for example,

$$\det(V) = (x_1 - x_0)(x_2 - x_0)(x_3 - x_0)(x_2 - x_1)(x_3 - x_1)(x_3 - x_2)$$

Hence $\mathbf{a}$ can be uniquely determined as $\mathbf{a} = V^{-1}\mathbf{f}$. Another proof of the uniqueness of the interpolating polynomial can be given as follows.

Theorem 5.1 Uniqueness of interpolating polynomial. *Given a set of points $x_0 < x_1 < \cdots < x_n$, there exists only one polynomial that interpolates a function at those points.*

Proof Let $P(x)$ and $Q(x)$ be two interpolating polynomials of degree at most n, for the same set of points $x_0 < x_1 < \cdots < x_n$. Also, let

$$R(x) = P(x) - Q(x).$$

Then $R(x)$ is also a polynomial of degree at most n. Since

$$P(x_i) = Q(x_i) = f_i,$$

we have, $R(x_i) = 0$ for $i = 0, 1, \ldots, n$. In other words $R(x)$ has $(n+1)$ roots. From the *Fundamental Theorem of Algebra* however, $R(x)$ cannot have more than n roots. A contradiction! Thus, $R(x) \equiv 0$, and $P(x) \equiv Q(x)$. □

Example 2 Given the following table for the function $f(x)$, obtain the lowest degree interpolating polynomial.

x_k	–1	2	4
f_k	–1	–4	4

Solution Since the number of nodes $= 3 = n + 1$, therefore $n = 2$. Let

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

that satisfies the following conditions at the points $x_0 = 1$; $x_1 = 2$, $x_2 = 4$:

$$P(-1) = -1; \quad P(2) = -4; \quad P(4) = 4.$$

$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 16 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -1 \\ -4 \\ 4 \end{pmatrix}.$$

Using Gaussian elimination with partial pivoting, we compute the solution

$$\mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -4 \\ -2 \\ 1 \end{pmatrix},$$

Therefore, the interpolating polynomial is given by

$$P_2(x) = -4 - 2x + x^2.$$

We can use this approximation to estimate the root of $f(x)$ that lies in the interval $[2, 4]$:

$$\gamma = \frac{2 + \sqrt{4 + 16}}{2} = 1 + \sqrt{5}.$$

□

An important remark is in order. One, in general, should not determine the interpolating polynomial by solving the Vandermonde linear system. These systems are surprisingly ill-conditioned for n no larger than 10. For example, for $0 < x_0 < x_1 < \cdots < x_n = 1$ uniformly distributed in [0,1], large n yields a Vandermonde matrix with almost linearly dependent columns, and the Vandermonde system becomes almost singular.

A more satisfactory form of the interpolating polynomial is due to Lagrange,

$$P_n(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x).$$

Here $L_j(x)$, for $0 \le j \le n$, are polynomials of degree n called *fundamental polynomials* or *Lagrange polynomials*. From the $(n + 1)$ conditions

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

we see that

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

The above property is certainly satisfied if we choose

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

i.e.,

$$L_j(x) = \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}.$$

The fact that $L_j(x)$ is unique follows from the uniqueness of the interpolation polynomial.

Recall that there is one and only one polynomial of degree n or less that interpolates $f(x)$ at the $(n+1)$ nodes $x_0 < x_1 < \cdots < x_n$. Thus, for this given set of nodes, both forms yield the same polynomial!

Vandermonde Approach.

$$P_n(x) = \sum_{i=0}^{n} a_i x^i \qquad V\mathbf{a} = \mathbf{f}$$

Operation count: It can be shown that the Vandermonde system $V\mathbf{a} = \mathbf{f}$ can be solved in $O(n^2)$ arithmetic operations rather than $O(n^3)$ operations because of the special form of the Vandermonde matrix.

Lagrange Approach.

$$P_n(x) = \sum_{j=0}^{n} f_j L_j(x) = \sum_{j=0}^{n} g_j \left[\frac{\prod_{i=0}^{n}(x - x_i)}{(x - x_j)} \right] = \sum_{j=0}^{n} g_j \cdot \nu_j$$

where

$$g_j = \frac{f(x_j)}{\prod_{\substack{i=0 \\ i \neq j}}^{n}(x_j - x_i)}.$$

Operation count: The Lagrange form avoids solving the ill-conditioned Vandermonde linear, systems, and evaluates the polynomials

$$\nu_j = \frac{1}{(x - x_j)} \prod_{i=0}^{n}(x - x_i)$$

for a given x. The cost of these operations for each $j = 0, 1, 2, \ldots, n$ are:

$$\begin{aligned}
u(x) &= \prod_{i=0}^{n}(x - x_i) && \rightarrow (2n+1) \text{ ops} \\
\nu_j &= \frac{u(x)}{(x - x_j)} && \rightarrow (n+1) \text{ ops} \\
g_j &= \frac{f_j}{\prod_{\substack{i=0 \\ i \neq j}}^{n}(x_j - x_i)} && \rightarrow 2n(n+1) \text{ ops (evaluated once only)}
\end{aligned}$$

Now, the cost for evaluating the polynomial

$$P_n(x) = \sum_{j=0}^{n} g_j \nu_j \qquad \rightarrow (2n+1) \text{ ops}$$

Therefore, the total number of operations are $5n + 3$ once g_j's are evaluated.

5.3 Error in Polynomial Interpolation

Let $e_n(x)$ be the error in polynomial interpolation given by

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

Since $P_n(x_i) = f(x_i)$, we have

$$e_n(x_i) = 0 \qquad x_0 < x_1 < \cdots < x_n.$$

In other words $e_n(x)$ is a function with at least $(n + 1)$ roots. Hence, we can write

$$e_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)h(x) = \pi(x) \cdot h(x)$$

where $h(x)$ is a function of x. Let us define

$$g(z) = e_n(z) - \pi(z)h(x)$$

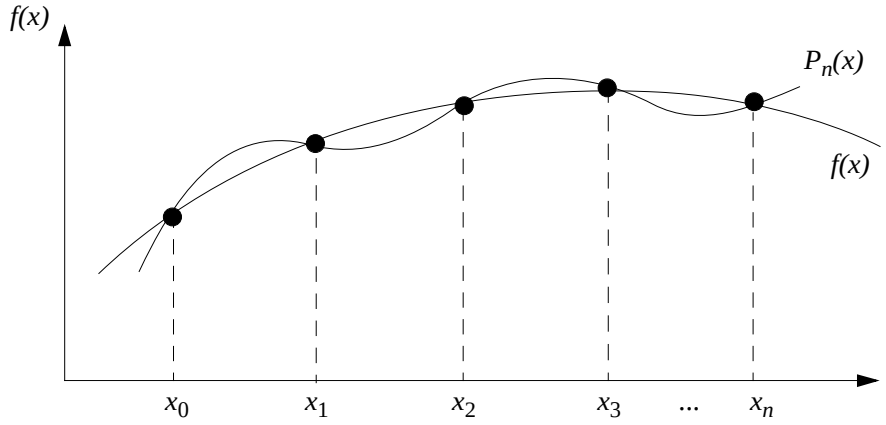


Figure 5.2: Computing the error in polynomial interpolation.

where $x \neq x_j$, for $0 \leq j \leq n$. Thus,

$$\begin{aligned} g(x) &= 0, \\ g(x_j) &= 0, \qquad 0 \leq j \leq n, \end{aligned}$$

i.e., $g(z)$ has $(n+2)$ distinct roots. Assuming that $f(x)$ has at least $n+1$ continuous derivatives, the $(n+1)$th derivative of $g(z)$, i.e.,

$$\frac{d^{(n+1)}}{dz^{(n+1)}} g(z) \equiv g^{(n+1)}(z)$$

has at least one root x^*. Hence,

$$g^{(n+1)}(z) = e_n^{(n+1)}(z) - h(x)\pi^{(n+1)}(z),$$

where $\pi^{(n+1)}(z) = (n+1)!$ Moreover, since $e_n(z) = f(z) - P_n(z)$, therefore,

$$e_n^{(n+1)}(z) = f^{(n+1)}(z) - P_n^{(n+1)}(z)$$

in which the last term $P_n^{(n+1)}(z)$ is zero. Using the above equations, we have

$$g^{(n+1)}(z) = f^{(n+1)}(z) - h(x)(n+1)!$$

We had shown earlier that $g^{(n+1)}(z)$ has at least one root x^* in the interval containing x and x_j, $j = 0, \ldots, n$. This implies that $g^{(n+1)}(x^*) = 0$, and therefore,

$$h(x) = \frac{f^{(n+1)}(x^*)}{(n+1)!}$$

Consequently, we have the following representation of the error in polynomial interpolation

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

where x^* is in the interval containing x and x_j, $j = 0, \ldots, n$.

Example 3 Obtain the Lagrange interpolating polynomial from the data

$$\sin 0 = 0, \qquad \sin \frac{\pi}{6} = \frac{1}{2}, \qquad \sin \frac{\pi}{3} = \frac{\sqrt{3}}{2}, \qquad \sin \frac{\pi}{2} = 1,$$

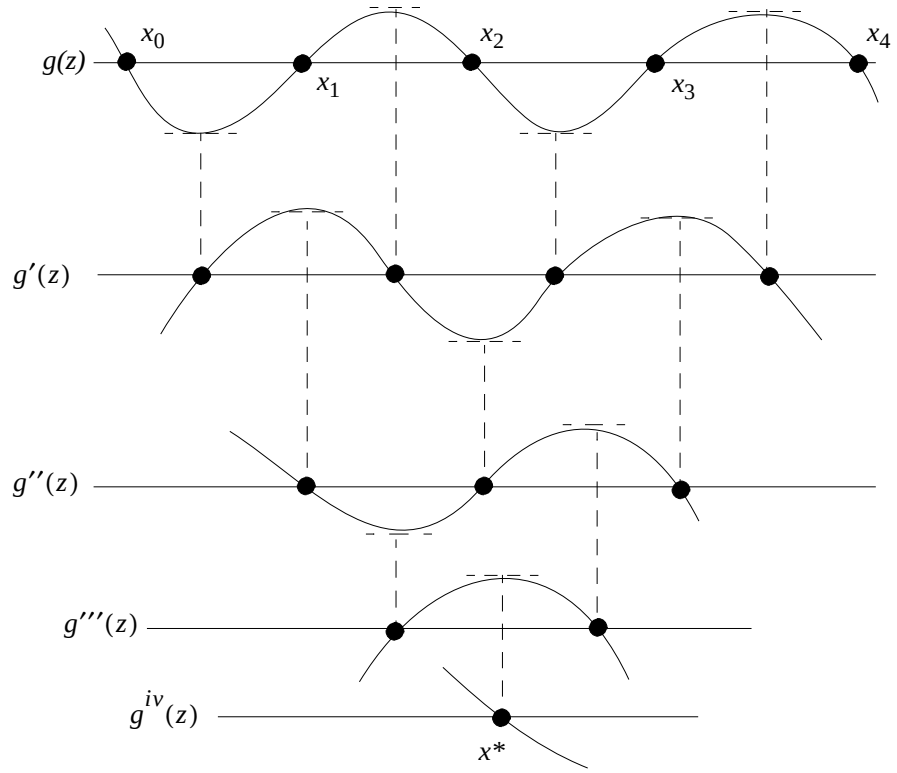


Figure 5.3: The $(n+1)$th derivative of a function $g(x)$ with $n+2$ roots $(n=3)$ must have at least one root.

and use it to evaluate $\sin\frac{\pi}{12}$ and $\sin\frac{\pi}{4}$

Solution The interpolating polynomial is given by

$$\begin{aligned} P_3(x) &= f_0L_0(x) + f_1L_1(x) + f_2L_2(x) + f_3L_3(x) \\ &= \frac{1}{2}L_1(x) + \frac{\sqrt{3}}{2}L_2(x) + L_3(x) \end{aligned}$$

At $x = \frac{\pi}{12}$,

$$L_1\left(\frac{\pi}{12}\right) = \frac{\left(\frac{\pi}{12}-0\right)\left(\frac{\pi}{12}-\frac{\pi}{3}\right)\left(\frac{\pi}{12}-\frac{\pi}{2}\right)}{\left(\frac{\pi}{6}-0\right)\left(\frac{\pi}{6}-\frac{\pi}{3}\right)\left(\frac{\pi}{6}-\frac{\pi}{2}\right)} = \frac{15}{16}$$

$$L_2\left(\frac{\pi}{12}\right) = \frac{\left(\frac{\pi}{12}-0\right)\left(\frac{\pi}{12}-\frac{\pi}{6}\right)\left(\frac{\pi}{12}-\frac{\pi}{2}\right)}{\left(\frac{\pi}{3}-0\right)\left(\frac{\pi}{3}-\frac{\pi}{6}\right)\left(\frac{\pi}{3}-\frac{\pi}{2}\right)} = -\frac{5}{16}$$

$$L_3\left(\frac{\pi}{12}\right) = \frac{\left(\frac{\pi}{12}-0\right)\left(\frac{\pi}{12}-\frac{\pi}{6}\right)\left(\frac{\pi}{12}-\frac{\pi}{3}\right)}{\left(\frac{\pi}{2}-0\right)\left(\frac{\pi}{2}-\frac{\pi}{6}\right)\left(\frac{\pi}{2}-\frac{\pi}{3}\right)} = \frac{1}{16}$$

Therefore,

$$P_3\left(\frac{\pi}{12}\right) = 0.26062.$$

Since $\sin\frac{\pi}{12} = 0.25882$, we have $|error| = 0.0018$.

At $x = \frac{\pi}{4}$,

$$L_1\left(\frac{\pi}{4}\right) = 0.5625, \qquad L_2\left(\frac{\pi}{4}\right) = 0.5625, \qquad L_3\left(\frac{\pi}{4}\right) = 0.0625.$$

Therefore

$$P_3\left(\frac{\pi}{4}\right) = 0.70589.$$

Since $\sin\frac{\pi}{4} = 0.70711$, we have $|error| = 0.00122$.

Let us estimate the error $e_n(x)$ at $x = \frac{\pi}{4}$:

$$e_3\left(\frac{\pi}{4}\right) = \left(\frac{\pi}{4} - 0\right)\left(\frac{\pi}{4} - \frac{\pi}{6}\right)\left(\frac{\pi}{4} - \frac{\pi}{3}\right)\left(\frac{\pi}{4} - \frac{\pi}{2}\right)\frac{f^{(4)}(x^*)}{4!}$$

Since $f(x) = \sin x$; $f^{(4)}(x) = \sin x$, and thus,

$$|f^{(4)}(x)| \leq 1.$$

Therefore,

$$\left|e_3\left(\frac{\pi}{4}\right)\right| \leq \frac{\pi}{4}\cdot\frac{\pi}{12}\cdot\frac{\pi}{12}\cdot\frac{\pi}{4}\cdot\frac{1}{4!} = 1.76\times 10^{-3}.$$

Similarly, $e_n(x)$ at $x = \frac{\pi}{12}$ is bounded by

$$\left|e_3\left(\frac{\pi}{12}\right)\right| \leq \frac{\pi}{12}\cdot\frac{\pi}{12}\cdot\frac{\pi}{4}\cdot\frac{5\pi}{12}\cdot\frac{1}{4!} = 2.94\times 10^{-3}.$$

□

5.4 Runge's Function and Equidistant Interpolation

In this section, we discuss a problem that arises when one constructs a global interpolation polynomial on a fairly large number of equally spaced nodes. Consider the problem of approximating the following function on the interval $[-1, 1]$

$$f(x) = \frac{1}{1+25x^2}$$

using the interpolation polynomial $P_n(x)$ on the equidistant nodes

$$x_j = -1 + \frac{2j}{n}, \qquad j = 0, 1, \ldots, n.$$

Figure 5.4 shows how the polynomials $P_4(x)$, $P_8(x)$, $P_{12}(x)$, and $P_{16}(x)$ approximate $f(x)$ on $[-1, 1]$. We see that as n increases, the interpolation error in the central portion of the interval diminishes, while increasing rapidly near the ends of the interval. Such behavior is typical in equidistant interpolation with polynomials of high degree. This is known as Runge's phenomenon, and the function $f(x)$ is known as Runge's function.

If the $(n+1)$ interpolation nodes are not chosen equally spaced, but rather placed near the ends of the interval at the roots of the so called Chebyshev polynomial of degree $(n+1)$, the problem with Runge's function disappears (see Fig. 5.5). The roots of the Chebyshev polynomial of degree $(n+1)$ are given by

$$x_k = -\cos\left(\frac{k+\frac{1}{2}}{n+1}\pi\right), \qquad k = 0, 1, \ldots, n.$$

5.5 The Newton Representation

The Lagrange form for the interpolating polynomial is not suitable for practical computations. The Newton form of the interpolating polynomial proves to be much more convenient. Let us obtain a different form of the lowest degree interpolating polynomial at the distinct interpolation nodes given by

$$x_0 < x_1 < x_2 < \cdots < x_{n-1}$$

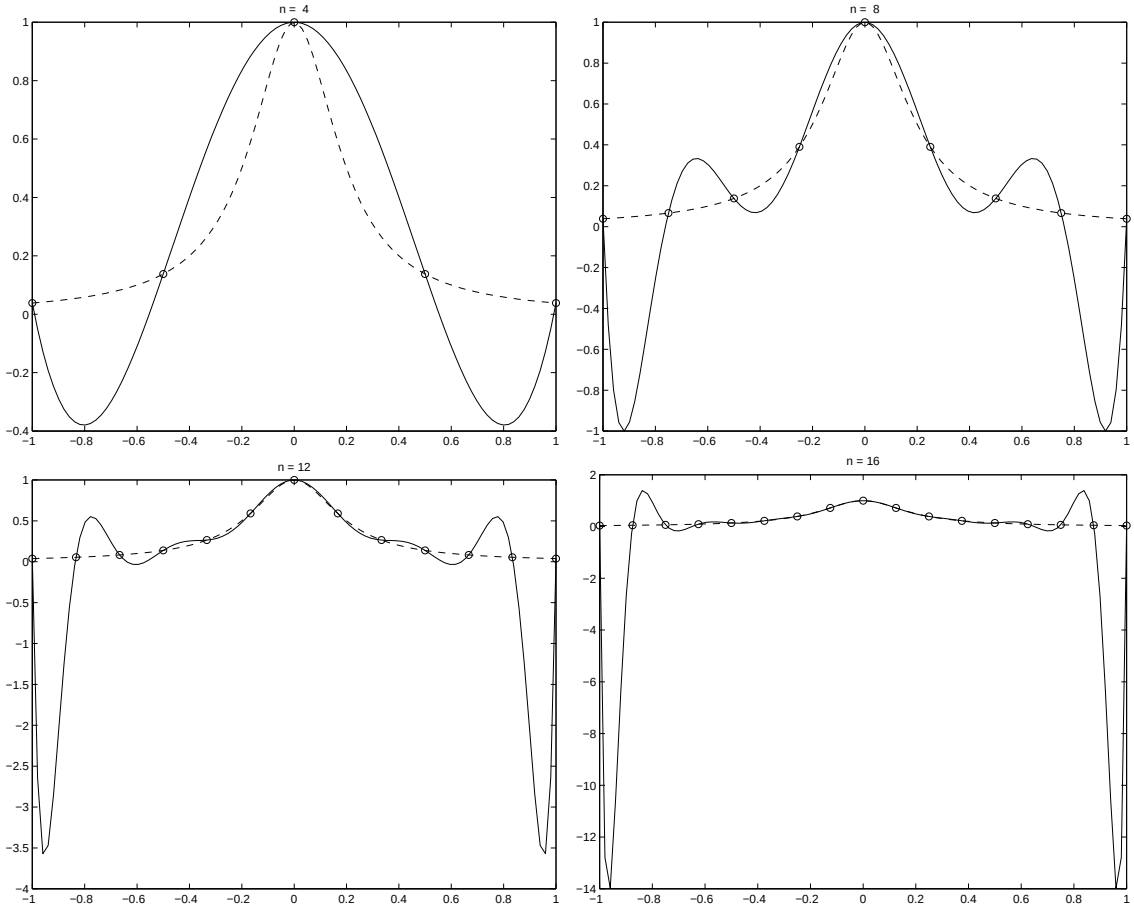


Figure 5.4: Equal spaced interpolation of Runge's function.

where $f(x_j) = f_j$, $j = 0, 1, \ldots, n-1$. Let the interpolating polynomial be

$$\begin{aligned} P_n(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \\ &\quad c_3(x - x_0)(x - x_1)(x - x_2) + \cdots + \\ &\quad c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \end{aligned}$$

Using the n conditions

$$P_n(x_j) = f_j, \qquad j = 0, 1, \ldots, n-1,$$

we can construct a system of equations in the unknown c_j. For $n = 3$, for example, this yields a system of linear equations of the form $L\mathbf{c} = \mathbf{f}$, where L is given by,

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & (x_1 - x_0) & 0 & 0 \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & 0 \\ 1 & (x_3 - x_0) & (x_3 - x_0)(x_3 - x_1) & (x_3 - x_0)(x_3 - x_1)(x_3 - x_2) \end{pmatrix}.$$

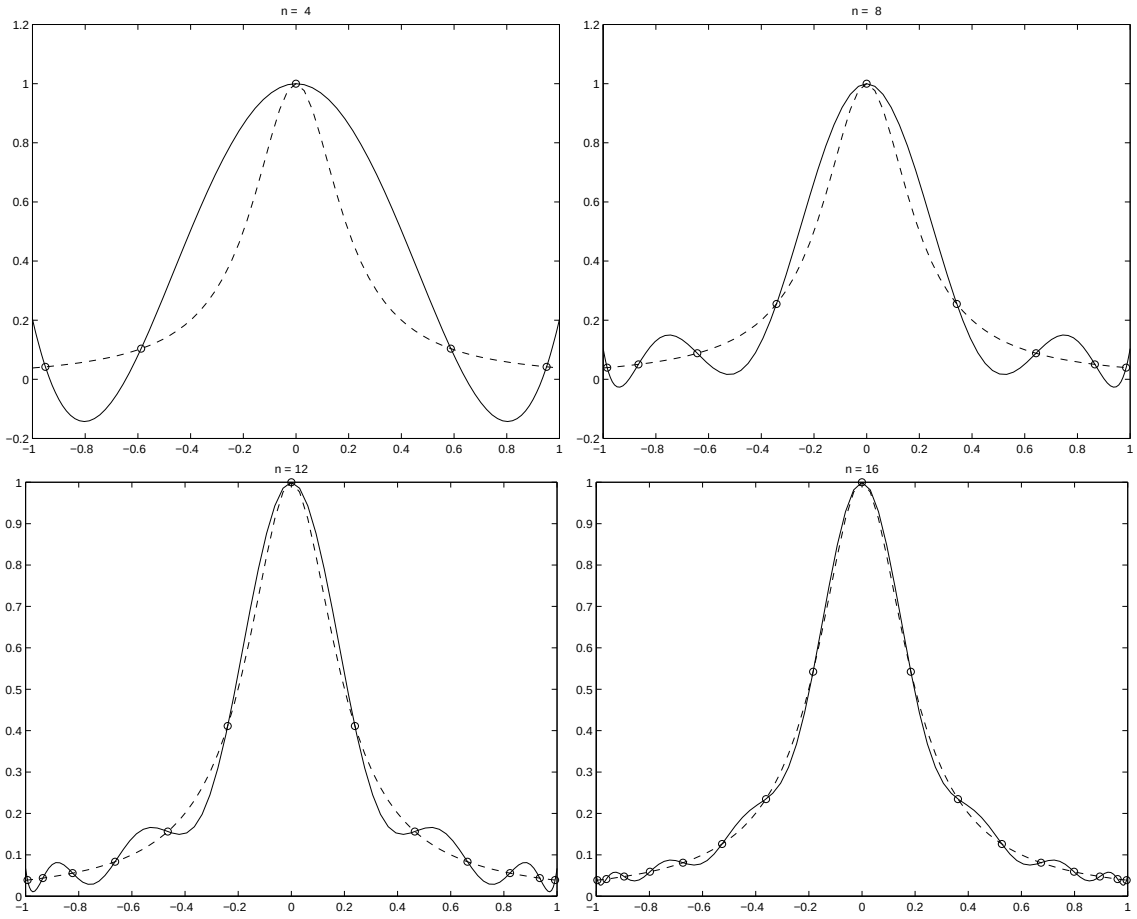


Figure 5.5: Chebyshev interpolation of Runge's function.

Note that the evaluation of L for n nodes requires $O(n^2)$ arithmetic operations. Assuming that the nodes are equidistant, i.e., $x_{i+1} - x_i = h$, the system $L\mathbf{c} = \mathbf{f}$ is given by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & h & 0 & 0 \\ 1 & 2h & 2h^2 & 0 \\ 1 & 3h & 6h^2 & 6h^3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix}.$$

Step 1.

$$\begin{aligned} c_0 &= f_0 \\ f_1^{(1)} &= (f_1 - f_0)/h \\ f_2^{(1)} &= (f_2 - f_0)/2h \\ f_3^{(1)} &= (f_3 - f_0)/3h. \end{aligned}$$

Then we have the system order 3:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & h & 0 \\ 1 & 2h & 2h^2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} f_1^{(1)} \\ f_2^{(1)} \\ f_3^{(1)} \end{pmatrix}.$$

Step 2.

$$\begin{aligned} c_1 &= f_1^{(1)} \\ f_2^{(2)} &= (f_2^{(1)} - f_1^{(1)})/h \\ f_3^{(2)} &= (f_3^{(1)} - f_1^{(1)})/2h \end{aligned}$$

Then we get the system of order 2:

$$\begin{pmatrix} 1 & 0 \\ 1 & h \end{pmatrix} \begin{pmatrix} c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} f_2^{(2)} \\ f_3^{(2)} \end{pmatrix}.$$

Step 3.

$$\begin{aligned} c_2 &= f_2^{(2)} \\ f_3^{(3)} &= (f_3^{(2)} - f_2^{(2)})/h \end{aligned}$$

Step 4.

$$c_3 = f_3^{(3)}$$

MATLAB

```
  function c = InterpolateNewton(x,f)
%  x: interpolating nodes
%  f: function values at x
%  c: coef. of Newton's form of interpolating polynomial
    n = length(x);
    for k = 1:n-1
        f(k+1:n) = (f(k+1:n)-f(k))./(x(k+1:n)-x(k));
    end
    c = y;
```

5.6 Spline Interpolation

Draftsmen used to draw smooth curves through data points by using *splines*. These are thin flexible strips of plastic or wood which were laid on paper and held with weights so as to pass through the required data points (or nodes). The weights are constructed in such a way that the spline is free to slip. As a result, the flexible spline straightens out as much as it can subject to passing over these points. Theory of elasticity suggests that this mechanical spline is given by a cubic polynomial (degree 3 polynomial) in each subinterval $[x_i, x_{i+1}]$, $i = 1, 2, \ldots, n-1$, with adjacent cubics joint continuously with continuous first and second derivatives. Let

$$s(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3,$$

be the form of each cubic spline. Since we have $(n-1)$ subintervals $[x_i, x_{i+1}]$, $1 \le i \le n-1$, then we have $4(n-1)$ unknowns; (4 parameters for each subinterval: x_0,a_1,a_2, and a_3). Suppose $s_i(x)$ denotes the cubic spline the ith interval. The conditions to be satisfied by these cubic splines are:

1. Continuity at each interior node:

$$s_{i-1}(x_i) = s_i(x_i), \qquad i = 2, 3, \ldots, n-1.$$

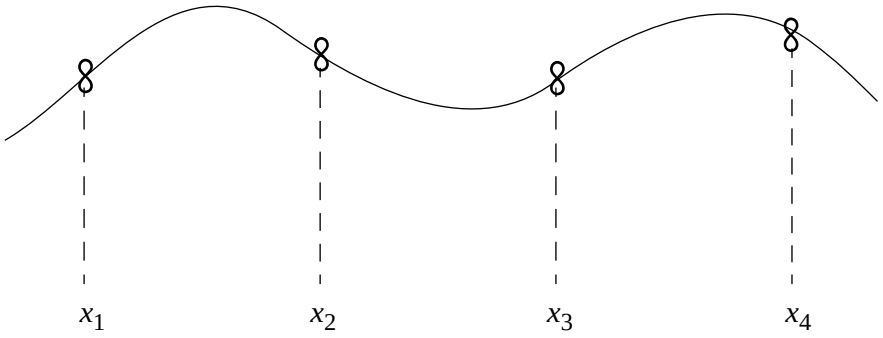


Figure 5.6: Spline interpolation.

2. Continuity of first derivative at each interior node:

$$s'_{i-1}(x_i) = s'_i(x_i), \qquad i = 2, 3, \ldots, n-1.$$

3. Continuity of second derivative at each interior node:

$$s''_{i-1}(x_i) = s''_i(x_i), \qquad i = 2, 3, \ldots, n-1.$$

4. Interpolation of function at each node:

$$s(x_j) = f(x_j), \qquad j = 1, 2, \ldots, n.$$

These provide $3(n-2) + n = 4n - 6$ conditions; however, in order to completely specify the cubic splines, we still need 2 *additional* conditions. These are conditions specified at the end points x_1 and x_n. For the "mechanical" or "natural" spline we set

$$s''(x_1) = s''(x_n) = 0.$$

Let the nodes x_j, $j = 1, 2, \ldots, n$, be equidistant, i.e.,

$$x_{j+1} - x_j = h$$

and let,

$$\sigma_j = \frac{1}{6} s''(x_j)$$

i.e.,

$$s''_{i-1}(x_i) = s''_i(x_i) = 6\sigma_i$$

for all i satisfying condition 3. Since

$$s(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3,$$

we have

$$\begin{aligned} s'(x) &= a_1 + 2a_2 x + 3a_3 x^2 \\ s''(x) &= 2a_2 + 6a_3 x, \end{aligned}$$

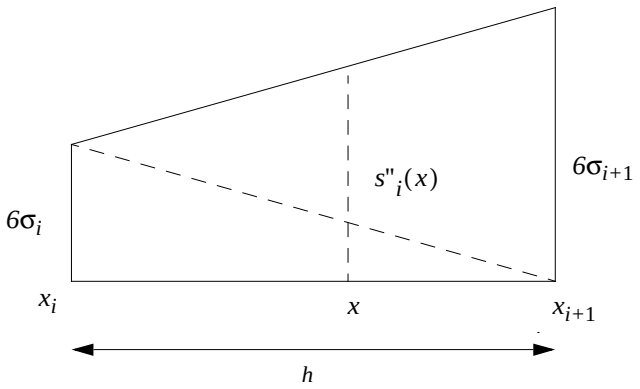


Figure 5.7: Computing spline within an interval.

i.e., $s''(x)$ is a straight line. Thus, $s_i''(x)$ takes the following form (see Fig. 5.7):

$$s_i''(x) = \frac{x_{i+1} - x}{h}(6\sigma_i) + \frac{(x - x_i)}{h}(6\sigma_{i+1}), \qquad i = 1, 2, \ldots, n-1.$$

Integrating once, we have

$$s_i'(x) = \frac{-6\sigma_i}{h} \cdot \frac{(x_{i+1} - x)^2}{2} + \frac{6\sigma_{i+1}}{h} \cdot \frac{(x - x_i)^2}{2} + \beta_1$$

and integrating once more, we have

$$s_i(x) = \frac{\sigma_i}{h}(x_{i+1} - x)^3 + \frac{\sigma_{i+1}}{h}(x - x_i)^3 + \beta_1 x + \beta_2$$

where β_1 and β_2 are constants. Clearly we can write $s_i(x)$ as follows,

$$s_i(x) = \frac{\sigma_i}{h}(x_{i+1} - x)^3 + \frac{\sigma_{i+1}}{h}(x - x_i)^3 + \gamma_1(x - x_i) + \gamma_2(x_{i+1} - x).$$

Using conditions 4 and 1,

$$\begin{array}{lll} s_i(x_i) = f_i & \Rightarrow & f_i = \sigma_i h^2 + \gamma_2 h \\ s_i(x_{i+1}) = f_{i+1} & \Rightarrow & f_{i+1} = \sigma_{i+1} h^2 + \gamma_1 h \end{array}$$

Therefore,

$$\begin{aligned} \gamma_1 &= \frac{f_{i+1}}{h} - \sigma_{i+1} h, \\ \gamma_2 &= \frac{f_i}{h} - \sigma_i h, \end{aligned}$$

and

$$\begin{aligned} s_i(x) &= \frac{\sigma_i}{h}(x_{i+1} - x)^3 + \frac{\sigma_{i+1}}{h}(x - x_i)^3 + \left(\frac{f_{i+1}}{h} - \sigma_{i+1} h\right)(x - x_i) \\ &\quad + \left(\frac{f_i}{h} - \sigma_i h\right)(x_{i+1} - x) \\ s_i'(x) &= \frac{-3\sigma_i}{h}(x_{i+1} - x)^2 + \frac{3\sigma_{i+1}}{h}(x - x_i)^2 + \left(\frac{f_{i+1}}{h} - \sigma_{i+1} h\right) \\ &\quad - \left(\frac{f_i}{h} - \sigma_i h\right). \end{aligned}$$

Using condition 2, i.e., $s_i'(x_i) = s_{i-1}'(x_i)$, we get

$$-3\sigma_i h + \left(\frac{f_{i+1} - f_i}{h}\right) - h(\sigma_{i+1} - \sigma_i) = 3\sigma_i h + \left(\frac{f_i - f_{i-1}}{h}\right) - h(\sigma_i - \sigma_{i-1})$$

or

$$\sigma_{i+1} + 4\sigma_i + \sigma_{i-1} = \frac{\Delta_i - \Delta_{i-1}}{h} \qquad i = 2, 3, \ldots, n-1,$$

where

$$\Delta_i = \frac{f_{i+1} - f_i}{h}.$$

Using the additional conditions $\sigma_1 = \sigma_n = 0$, we have

$$\begin{array}{rcll} 4\sigma_2 + \sigma_3 & = & (\Delta_2 - \Delta_1)/h & \\ \sigma_{i-1} + 4\sigma_i + \sigma_{i+1} & = & (\Delta_i - \Delta_{i-1})/h & i = 3, 4, \ldots, n-2 \\ \sigma_{n-2} + 4\sigma_{n-1} & = & (\Delta_{n-1} - \Delta_{n-2})/h. & \end{array}$$

Now, assigning

$$g_j = \frac{\Delta_j - \Delta_{j-1}}{h} = \frac{f_{j+1} - 2f_j + f_{j-1}}{h^2},$$

we have the linear system

$$\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix} \begin{pmatrix} \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_{n-2} \\ \sigma_{n-1} \end{pmatrix} = \begin{pmatrix} g_2 \\ g_3 \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{pmatrix}.$$

The tridiagonal system above, denoted by $[1, 4, 1]$, may be factored as follows:

$$\begin{pmatrix} 1 & & & & \\ \lambda_1 & 1 & & & \\ & \lambda_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda_{n-3} & 1 \end{pmatrix} \begin{pmatrix} \mu_1 & 1 & & & \\ & \mu_2 & 1 & & \\ & & \mu_3 & \ddots & \\ & & & \ddots & 1 \\ & & & & \mu_{n-2} \end{pmatrix}$$

The following conditions must be satisfied:

$$\mu_1 = 4, \qquad \lambda_i \mu_i = 1, \qquad \lambda_i + \mu_{i+1} = 4$$

The algorithm for computing Ł and U is given below.

Triangular decomposition for system [1,4,1]

$\mu_1 = 4$
for $i = 1, 2, \ldots, n-3$
    $\lambda_i = \mu_i^{-1}$
    $\mu_{i+1} = 4 - \lambda_i$
end;

Once Ł and U are obtained, the above system is solved as shown in Chapter 4.

Chapter 6

Numerical Integration

Numerical integration is the study of how the numerical value of an integral can be found. Methods of function approximation discussed in Chapter 5, i.e., function approximation via the global interpolation polynomial or spline interpolation, provides a basis for numerical integration techniques. Let the definite integral under consideration be

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

where $[a, b]$ is a *finite* closed interval. In this chapter, we primarily consider approximations to $I\{f\}$ that are of the form

$$I_{n+1}\{f\} = \sum_{j=0}^{n} a_j f(x_j)$$

where the *quadrature nodes* are given by,

$$a \leq x_0 < x_1 < x_2 < \ldots < x_n \leq b$$

and the real coefficients a_j are known as the *quadrature coefficients*. The nodes x_j are preassigned and often equally spaced.

Numerical approximation of definite integrals is desirable in two cases:

1. a closed form of $I\{f\}$ is not easily obtained, or
2. the available closed form of $I\{f\}$ is too complicated for efficient numerical evaluation, as the following example clearly illustrates:

$$\begin{aligned}\int_0^x \frac{dt}{1+t^4} &= \frac{1}{4\sqrt{2}} \log_e \left[\frac{x^2 + x\sqrt{2} + 1}{x^2 - x\sqrt{2} + 1}\right] + \\ &\quad \frac{1}{2\sqrt{2}} \left[\tan^{-1} \frac{x}{\sqrt{2} - x} + \tan^{-1} \frac{x}{\sqrt{2} + x}\right].\end{aligned}$$

In *interpolatory quadrature*, the only type we will be studying in this chapter in addition to Romberg and adaptive quadrature, we approximate the function $f(x)$ by the interpolating polynomial (Lagrange form)

$$P_n(x) = \sum_{j=0}^{n} f(x_j) L_j(x)$$

where

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

Hence,

$$I_{n+1}\{f\} = \int_a^b P_n(x)dx = \sum_{j=0}^{n} f(x_j) \int_a^b L_j(x)dx = \sum_{j=0}^{n} a_j f(x_j)$$

in which

$$a_j = \int_a^b L_j(x)dx, \qquad j = 0, 1, \ldots, n.$$

Note that the quadrature coefficients a_j are completely determined by the end points a and b and the interpolation nodes x_j, $j = 0, 1, \ldots, n$. Correspondingly, the *quadrature error* or *truncation error* is given by

$$E_{n+1}\{f\} = I\{f\} - I_{n+1}\{f\} = \int_a^b [f(x) - P_n(x)]dx = \int_a^b e_n(x)dx$$

in which $e_n(x)$ is the interpolation error

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

where $x^* \equiv x^*(x) \in (x_0, x_1, \ldots, x_n, x)$. If $f(x)$ is a polynomial of degree at most n, then $f^{(n+1)}(x) = 0$ and hence $E_{n+1}\{f\} = 0$. We will use this observation to obtain some basic interpolatory quadrature rules.

6.1 Method of Exact Matching

This can also be called *Method of Undetermined Coefficients*.

6.1.1 The Trapezoidal Rule

The trapezoidal rule uses the function value at two points x_0 and x_1 to compute the integral of the function in the interval $[x_0, x_1]$. For $x_0 = 0$ and $x_1 = h$, the integral

$$I\{f\} = \int_0^h f(x)dx$$

is approximated by the trapezoidal rule

$$I_2\{f\} = a_0 f_0 + a_1 f_1$$

where the quadrature coefficients a_0 and a_1 need to be determined. Observe that the trapezoidal rule is exact for polynomials of degree zero and one, i.e., for a constant and a straight line. Therefore, it is exact for the following functions:

1. $f(x) = 1$
2. $f(x) = x$

From (1) we see that

$$\int_0^h 1 \cdot dx = a_0 + a_1 = h,$$

and from (2) we have

$$\int_0^h x dx = a_1 h = \frac{h^2}{2}.$$

Using these equations, we get the values of the coefficients, $a_0 = a_1 = h/2$, and the trapezoidal rule can be written as

$$I_2\{f\} = \frac{h}{2}(f_0 + f_1),$$

where $h = x_1 - x_0$. Observe that this is also the area of the shaded trapezoid in Fig. 6.1. Consequently,

$$I\{f\} = \int_{x_0}^{x_1} f(x)dx = I_2\{f\} + E_2\{f\} = \frac{h}{2}[f_0 + f_1] + E_2\{f\},$$

where $E_2\{f\}$ is the error in trapezoidal rule.

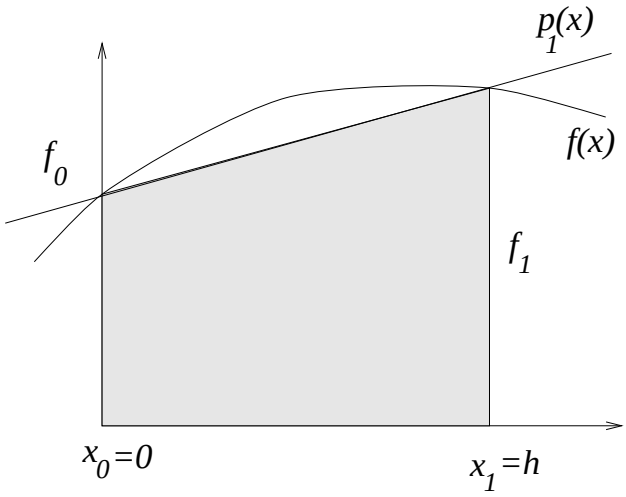


Figure 6.1: Trapezoidal rule.

Composite Trapezoidal Rule. From the basic trapezoidal rule we can construct a quadrature rule to compute an integral over the interval $[a, b]$ by dividing the interval into N equal subintervals and using the basic trapezoidal rule for each subinterval. Suppose the interval is divided into N subintervals of equal length using the nodes $x_0, x_1, \ldots, x_N$, where $x_i = a + ih$, for $i = 0, \ldots, N$, and $h = \frac{b-a}{N}$ is the size of each subinterval. Assuming $f(x_i) = f_i$, the composite trapezoidal rule is given as

$$\int_a^b f(x)dx = \frac{h}{2}[f_0 + 2f_1 + 2f_2 + \cdots + 2f_{N-1} + f_N] + E_T,$$

where E_T is the error in the composite trapezoidal rule.

6.1.2 Simpson's Rule

Simpson's rule uses 3 interpolation nodes and a quadratic interpolating polynomial. The general form is

$$I_3\{f\} = a_{-1}f_{-1} + a_0 f_0 + a_1 f_1$$

Since the Simpson's rule is exact for polynomials of degree zero, one, and two, it is exact for the following functions.

1. $f(x) = 1$,
2. $f(x) = x$, and

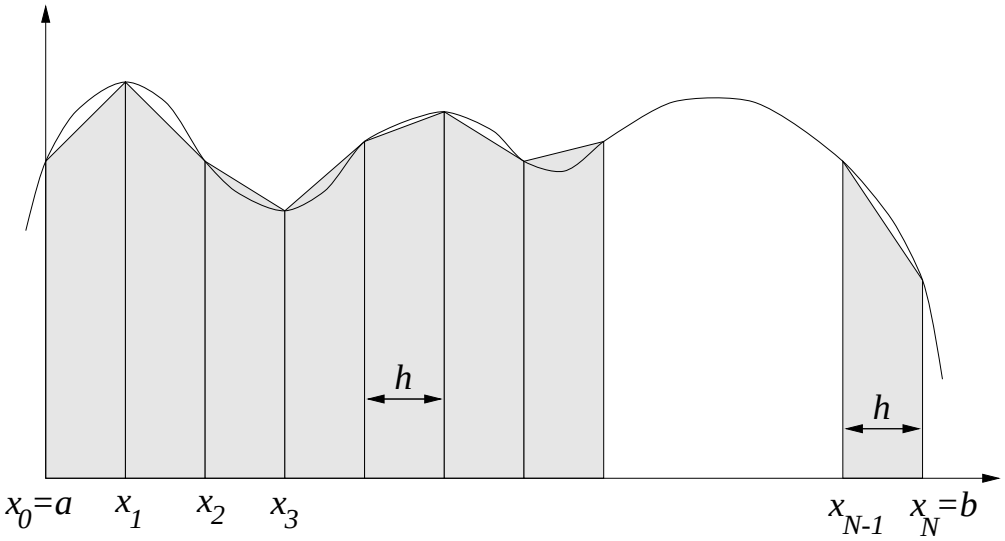


Figure 6.2: Composite trapezoidal rule for equidistant points.

3. $f(x) = x^2$.

Thus, from (1) we have

$$\int_{-h}^{h} 1 \cdot dx = a_{-1} + a_0 + a_1 = 2h,$$

from (2), we get

$$\int_{-h}^{h} x dx = -ha_{-1} + ha_1 = 0,$$

and from (3) we obtain

$$\int_{-h}^{h} x^2 dx = h^2 a_{-1} + h^2 a_1 = \frac{2}{3}h^3$$

These equations yield the quadrature coefficients

$$a_{-1} = a_1 = \frac{1}{3}h, \qquad a_0 = \frac{4}{3}h.$$

Hence, the Simpson's rule is given as

$$I_3\{f\} = \frac{h}{3}[f_{-1} + 4f_0 + f_1],$$

and

$$I\{f\} = \int_{x_0}^{x_2} f(x)dx = I_3\{f\} + E_3\{f\} = \frac{h}{3}[f_{-1} + 4f_0 + f_1] + E_3\{f\},$$

where $h = x_2 - x_1 = x_1 - x_0$, and $E_3\{f\}$ is the error in Simpson's rule.

Composite Simpson's Rule. Now, let us derive the composite Simpson's rule for computing integral over the interval $[a, b]$. Suppose we have N equal subintervals of width $2h$, i.e., $b - a = 2hN$. We also define $2N + 1$ equally spaced points $x_j = a + jh$, for $j = 0, \ldots, 2N$ in interval $[a, b]$. The ith subinterval has the

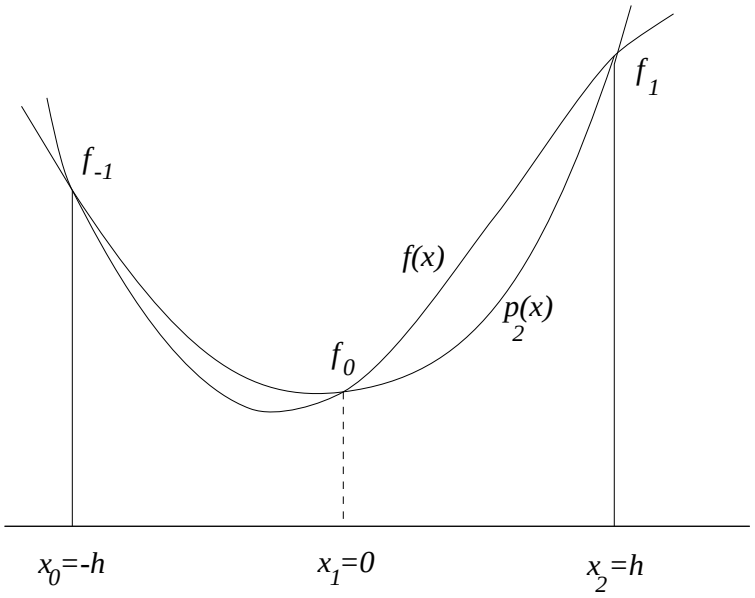


Figure 6.3: Simpson's rule.

endpoints x_{2i-2} and x_{2i}, and the midpoint x_{2i-1}, for $i = 1, \ldots, N$. Applying the basic Simpson's rule over each subinterval, we obtain the composite rule

$$\begin{aligned}\int_a^b f(x)dx &= [\,\frac{h}{3}(f_0 + 4f_1 + f_2) + \frac{h}{3}(f_2 + 4f_3 + f_4) + \cdots \\ &\quad + \frac{h}{3}(f_{2N-2} + 4f_{2N-1} + f_{2N})\,] + E_S \\ &= \frac{h}{3}\left[f_0 + 4\sum_{i=1}^{N} f_{2i-1} + 2\sum_{i=1}^{N-1} f_{2i} + f_{2N}\right] + E_S,\end{aligned}$$

where E_S is the error in the composite Simpson's rule.

6.2 The Truncation Error

6.2.1 The Trapezoidal Rule

Recall that

$$I\{f\} = \int_0^h f(x)dx = \frac{h}{2}[f_0 + f_1] + E_2\{f\}$$

Let us compute the error in the trapezoidal rule for the polynomials of degree 0, 1, and 2, i.e., for $f(x) = 1, x, x^2$.

$$\begin{aligned}E_2\{1\} &= \int_0^h 1 \cdot dx - \frac{h}{2}[1 + 1] = 0, \\ E_2\{x\} &= \int_0^h x dx - \frac{h}{2}[0 + h] = 0, \\ E_2\{x^2\} &= \int_0^h x^2 dx - \frac{h}{2}[0 + h^2] = -\frac{1}{6}h^3 \neq 0.\end{aligned}$$

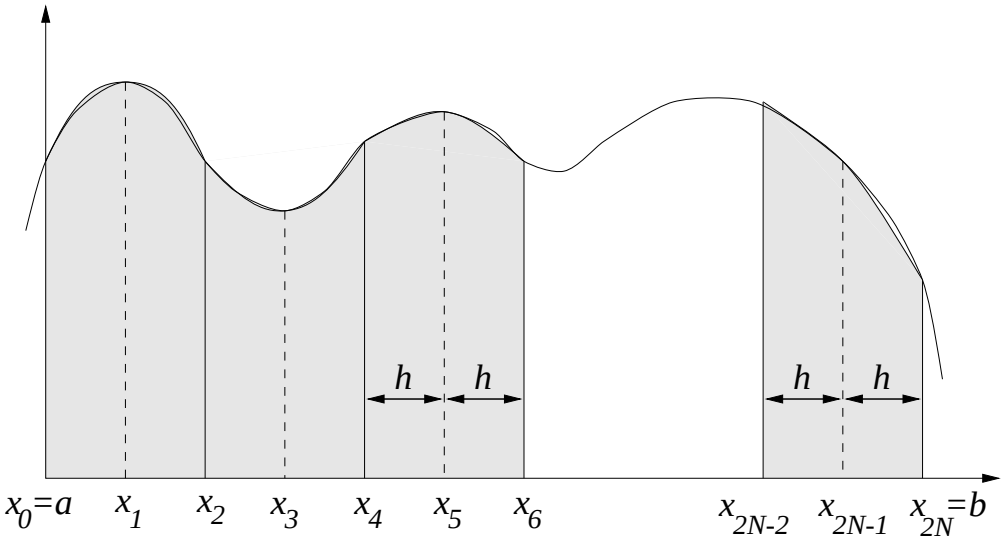


Figure 6.4: Composite Simpson's rule.

Since the error is zero for polynomials of degree 1 or less, the trapezoidal rule is said to have *degree of precision* = 1. By Taylor's formula,

$$\begin{aligned} f(x) &= f_0 + xf_0' + \frac{x^2}{2!}f_0'' + \frac{x^3}{3!}f_0''' + \cdots \\ &= (\text{straight line}) + \frac{x^2}{2!}f_0'' + \frac{x^3}{3!}f_0''' + \cdots \end{aligned}$$

Therefore, we have the following expression for error

$$E_2\{f(x)\} = E_2\left\{f_0 + xf_0'\right\} + E_2\left\{\frac{x^2}{2}f_0'' + \frac{x^3}{6}f_0''' + \cdots\right\}$$

Since the error for a straight line is zero, the first term on the right hand side is zero, i.e., $E_2\{f_0 + xf_0'\} = 0$. Thus,

$$\begin{aligned} E_2\{f(x)\} &= 0 + \frac{1}{2}f_0''E_2\{x^2\} + \frac{1}{6}f_0'''E_2\{x^3\} + \cdots \\ &= -\frac{h^3}{12}f_0'' - \frac{h^4}{24}f_0''' - \cdots \\ &= -\frac{h^3}{12}f_0'' + O(h^4) \end{aligned}$$

The above expression of the truncation error in the trapezoidal rule is known as an *asymptotic error estimate*. Over here, the term $O(h^4)$ has the following meaning: a function $q(h) = O(h^\alpha)$ as $h \to 0$, (read as $q(h)$ is of the order h^α), means that there exist constants h_0 and K such that

$$|q(h)| \leq Kh^\alpha, \qquad 0 < h \leq h_0.$$

6.2.2 Simpson's Rule

Recall that

$$I\{f\} = \int_{-h}^{h} f(x)dx = \frac{h}{3}[f_{-1} + 4f_0 + f_1] + E_3\{f\}$$

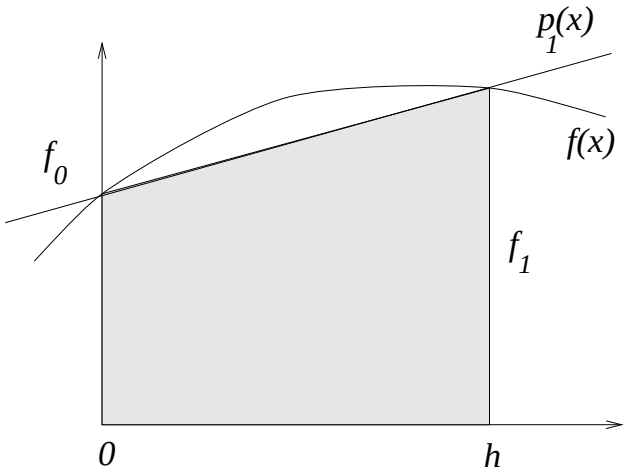


Figure 6.5: Truncation error in trapezoidal rule.

Let us compute the error for polynomials of degree 3 or less.

$$\begin{aligned}
E_3\{1\} &= \int_{-h}^{h} 1dx - \frac{h}{3}[1+4+1] = 0, \\
E_3\{x\} &= \int_{-h}^{h} xdx - \frac{h}{3}[-h+0+h] = 0, \\
E_3\{x^2\} &= \int_{-h}^{h} x^2dx - \frac{h}{3}[h^2+0+h^2] = 0, \\
E_3\{x^3\} &= \int_{-h}^{h} x^3dx - \frac{h}{3}[-h^3+0+h^3] = 0.
\end{aligned}$$

We expected Simpson's rule to be exact for polynomials of degree 2 or less. It turns out that Simpson's rule is also exact for cubics. For $f(x) = x^4$,

$$E_3\{x^4\} = \int_{-h}^{h} x^4dx - \frac{h}{3}[h^4+0+h^4] = -\frac{4}{15}h^5 \neq 0.$$

Therefore, Simpson's rule has a degree precision = 3. Using Taylor's formula,

$$\begin{aligned}
f(x) &= f_0 + xf_0' + \frac{x^2}{2!}f_0'' + \frac{x^3}{3!}f_0''' + \frac{x^4}{4!}f_0^{(iv)} + \cdots \\
&= (\text{degree 3 polynomial}) + \frac{x^4}{24}f_0^{(iv)} + \frac{x^5}{120}f_0^{v} + \cdots
\end{aligned}$$

We have shown above that Simpson's rule is exact for cubics. Thus,

$$E_3\{f\} = 0 + \frac{1}{24}f_0^{(iv)}E_3\{x^4\} + \frac{1}{120}f_0^{v}E_3\{x^5\} + \cdots$$

and the asymptotic error estimate is

$$E_3\{f\} = \frac{-h^5}{90}f_0^{(iv)} + O(h^7).$$

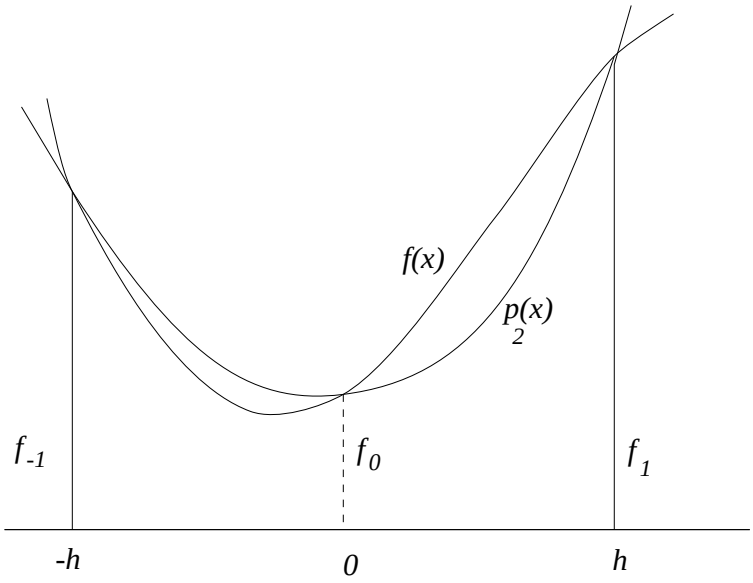


Figure 6.6: Truncation error in Simpson's rule.

We should mention here that one can obtain *strict* error estimates for the above two quadrature rules, rather than just asymptotic estimates. A complete discussion of this question, however, is outside the scope of this book. We only state the result as follows.

Theorem 6.1 *If an interpolatory quadrature formula has a degree of precision m, then its truncation error is given by*

$$E_{n+1}\{f\} = \frac{f^{(m+1)}(z)}{(m+1)!} E_{n+1}\{x^{m+1}\}$$

where $a < z < b$.
Thus, for the trapezoidal rule $m = 1$, and

$$E_2\{f\} = \frac{f''(z)}{2!} E_2\{x^2\} = -\frac{h^3}{12} f''(z), \qquad x_0 < z < x_1.$$

While for Simpson's rule, $m = 3$ and

$$E_3\{f\} = \frac{f^{(iv)}(z)}{4!} E_3\{x^4\} = -\frac{h^5}{90} f^{(iv)}(z), \qquad x_0 < z < x_2.$$

The truncation error for the composite trapezoid rule is given by

$$E_T = -\frac{h^3 N}{12} f''(z^*) = -\frac{(b-a)h^2}{12} f''(z^*), \qquad b - a = Nh,$$

where $x_0 < z^* < x_n$. Similarly, the error for the composite Simpson's rule is given by

$$E_S = -\frac{h^5}{90}\left[f^{(iv)}(y_1) + f^{(iv)}(y_2) + \cdots + f^{(iv)}(y_N)\right]$$

where $x_{2i-2} < y_i < x_{2i}$, for $i = 1, 2, \ldots, N$. This simplifies to

$$E_S = -\frac{h^5 N}{90} f^{(iv)}(y^*) = -\frac{(b-a)h^4}{180} f^{(iv)}(y^*), \qquad b - a = 2Nh$$

where $a < y^* < b$.

6.3 Spline Quadrature

Consider the definite integral

$$I\{f\} = \int_a^b f(x)dx.$$

If we approximate $f(x)$ via spline interpolation, such as the cubic interpolatory spline $s(x)$ discussed in Chapter 5, where

$$a = x_1 < x_2 < \cdots < x_n = b, \qquad h_i = x_{i+1} - x_i,$$

then we obtain an interesting approximation of $I\{f\}$ that is given by

$$I_{sp}\{f\} = \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} s_i(x)dx,$$

where $s_i(x)$ is the cubic spline function for the ith subinterval $[x_i, x_{i+1}]$, and is given by

$$\begin{aligned} s_i(x) &= \frac{\sigma_i}{h_i}(x_{i+1} - x)^3 + \frac{\sigma_{i+1}}{h_i}(x - x_i)^3 + \left(\frac{f_{i+1}}{h_i} - \sigma_{i+1}h_i\right)(x - x_i) \\ &\quad + \left(\frac{f_i}{h_i} - \sigma_i h_i\right)(x_{i+1} - x) \end{aligned}$$

(see Chapter 5). Let us define

$$w = \frac{1}{h_i}(x - x_i), \qquad \bar{w} = 1 - w = \frac{1}{h_i}(x_{i+1} - x)$$

Thus,

$$\begin{aligned} s_i(x) = s_i(w) &= h_i^2\sigma_i\bar{w}^3 + h_i^2\sigma_{i+1}w^3 + w[f_{i+1} - \sigma_{i+1}h_i^2] + \bar{w}[f_i - \sigma_i h_i^2] \\ &= \bar{w}f_i + wf_{i+1} + h_i^2[\sigma_{i+1}(w^3 - w) + \sigma_i(\bar{w}^3 - \bar{w})], \end{aligned}$$

and

$$\int_{x_i}^{x_{i+1}} s_i(x)dx = h_i \int_0^1 s_i(w)dw.$$

Moreover, observing that

$$\int_0^1 w\ dw = \int_0^1 \bar{w}\ d\bar{w} = \frac{1}{2},$$

and

$$\int_0^1 (w^3 - w)dw = \int_0^1 (\bar{w}^3 - \bar{w})d\bar{w} = -\frac{1}{4},$$

we obtain

$$\int_{x_1}^{x_{i+1}} s_i(x)dx = \frac{h_i}{2}[f_i + f_{i+1}] - \frac{h_i^3}{4}[\sigma_i + \sigma_{i+1}].$$

Note that this equation can be regarded as the trapezoidal rule plus a correction term. This correction term is given by

$$\tau_i = -\frac{h^3}{24}[s''(x_i) + s''(x_{i+1})],$$

which indicates that if $f''(x)$ is not too badly behaved, then

$$\tau_i \simeq -\frac{h_i^3}{12}f''(\theta) \simeq E_2\{f\}, \qquad x_i < \theta < x_{i+1}.$$

In other words if $f''(x)$ is well behaved, we have

$$\int_{x_i}^{x_{i+1}} s_i(x)dx \cong \frac{h_i}{2}[f_i + f_{i+1}] + E_2\{f\}$$

which can be remarkably close to the integral

$$\int_{x_i}^{x_{i+1}} f(x)dx.$$

6.4 Richardson's Extrapolation

In many calculations what one would really like to know is the limiting value of a certain quantity $F(h)$ as $F \to 0$. Needless to say, the work required for computing $F(h)$ increases sharply as h approaches zero. Furthermore, the effects of rounding errors set a practical limit on how small h can be chosen. Usually, one has some knowledge of how the truncation error $[F(0) - F(h)]$ behaves as $h \to 0$. Let

$$F(h) = F(0) + \alpha_1 h^p + O(h^r)$$

where $r > p$, and α_1 is an unknown. Compute F for two step lengths: h and qh, $(q > 1)$

$$\begin{aligned} F(h) &= F(0) + \alpha_1 h^p + O(h^r) \\ F(qh) &= F(0) + \alpha_1 (qh)^p + O(h^r). \end{aligned}$$

Multiplying the first equation by q^p, the second by -1, and adding, we get

$$a^p F(h) - F(qh) = [q^p - 1]F(0) + O(h^r),$$

or

$$F(0) = \left[F(h) + \frac{F(h) - F(qh)}{q^p - 1}\right] + O(h^r).$$

This simple technique known as *Richardson's extrapolation* improves the asymptotic error bound from $O(h^p)$ to $O(h^r)$.

The repeated application of Richardson's extrapolation to numerical integration is known as *Romberg Integration*. As an illustration, consider the composite trapezoidal rule on N panels, each of width h,

$$T(h) = \frac{h}{2}\left[f_0 + 2\sum_{j=1}^{N-1} f_j + f_N\right].$$

In Appendix 6.7, we show that

$$I\{f\} \equiv I = T(h) + \alpha_1 h^2 + \alpha_2 h^4 + \alpha_3 h^6 + \cdots$$

Using panels of half the width, we get

$$I = T\left(\frac{h}{2}\right) + \alpha_1\left(\frac{h^2}{4}\right) + \alpha_2\left(\frac{h^4}{16}\right) + \alpha_3\left(\frac{h^6}{64}\right) + \cdots$$

Multiplying this equation by $-1/4$ and adding to the previous equation, we obtain

$$\frac{3}{4}I = \left[T\left(\frac{h}{2}\right) - \frac{1}{4}T(h)\right] - \frac{3}{16}\alpha_2 h^4 - \cdots$$

or

$$I = \frac{1}{3}\left[4T\left(\frac{h}{2}\right) - T(h)\right] - \frac{1}{4}\alpha_2 h^4 + O(h^6) \tag{6.1}$$

$$= \left[T\left(\frac{h}{2}\right) + \frac{1}{3}\left(T\left(\frac{h}{2}\right) - T(h)\right)\right] + O(h^4) \tag{6.2}$$

Suppose $b - a = Nh$, and let us denote the composite formula with N panels by T_N, and the formula with $2N$ panels by T_{2N}. Thus,

$$T_N = T(h), \qquad T_{2N} = T\left(\frac{h}{2}\right).$$

The formula using Richardson's extrapolation is given by

$$T_{2N}^{(1)} = \frac{1}{3}(4T_{2N} - T_N) = T_{2N} + \frac{1}{3}(T_{2N} - T_N).$$

Now, equation 6.2 can be written as

$$I = T_{2N}^{(1)} - \frac{1}{4}\alpha_2 h^4 + O(h^6). \tag{6.3}$$

Similarly

$$I = T_{4N}^{(1)} - \frac{1}{4}\alpha_2 \left(\frac{h}{2}\right)^4 + O(h^6), \tag{6.4}$$

in which

$$T_{4N}^{(1)} = \frac{1}{3}(4T_{4N} - T_{2N}) = T_{4N} + \frac{1}{3}(T_{4N} - T_{2N}).$$

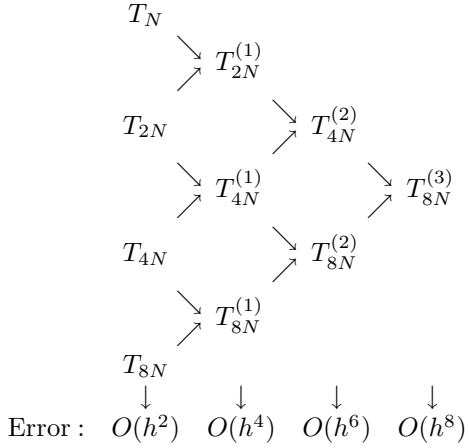
Eliminating α_2 from equations (6.3) and (6.4), we have

$$I = \frac{1}{15}\left[16T_{4N}^{(1)} - T_{2N}^{(1)}\right] + O(h^6)$$

$$= \left[T_{4N}^{(1)} + \frac{1}{15}\left(T_{4N}^{(1)} - T_{2N}^{(1)}\right)\right] + O(h^6)$$

$$= T_{4N}^{(2)} + O(h^6).$$

If we systematically halve the interval, taking $h, \frac{h}{2}, \frac{h}{4}, \ldots$, etc., we can construct the table of which a section is shown below.



In general, $I = T^{(j)} + O(h^r)$ in which $r = 2(j+1)$, and

$$T_{2i}^{(j)} = \frac{4^j T_{2i}^{(j-1)} - T_i^{(j-1)}}{(4^j - 1)} = T_{2i}^{(j-1)} + \frac{T_{2i}^{(j-1)} - T_i^{(j-1)}}{(4^j - 1)}.$$

Example 6.1 Compute $\int_0^{0.8} \frac{\sin x}{x} dx$ using Romberg's integration.

Solution

$$\begin{array}{l|l} h & \\ 0.8 & T_1 = .758680 \\ & \quad \searrow \\ & \quad \nearrow T_2^{(1)} = .772120 \\ 0.4 & T_2 = .768760, \quad \searrow \\ & \quad \nearrow T_4^{(2)} = .772095 \\ & \quad \searrow \\ & \quad \nearrow T_4^{(1)} = .772096, \\ 0.2 & T_4 = .771262, \quad \searrow \\ & \quad \nearrow T_8^{(2)} = \underline{.772095} \\ & \quad \searrow \\ & \quad \nearrow T_8^{(1)} = .772095, \\ 0.1 & T_8 = .771887, \end{array}$$

□

6.5 Adaptive Quadrature

An adaptive quadrature algorithm uses one or two basic quadrature rules, namely, the trapezoidal rule and Simpson's rule, and determines the subinterval sizes so that the computed result meets some prescribed accuracy requirement. In this way, an attempt is made to provide a result with the prescribed accuracy at the lowest cost possible. By cost we mean computer time, which in turn is directly proportional to the number of function evaluations necessary to obtain the result. The user of an adaptive quadrature routine specifies the interval $[a, b]$, provides a subroutine which computes the function $f(x)$ for any $x \in [a, b]$, and chooses a tolerance ϵ. The adaptive routine attempts to compute a quantity Q such that

$$\left| Q - \int_a^b f(x)dx \right| \leq \epsilon.$$

Example 6.2 Consider the problem of approximation $I = \int_a^b f(x)dx$ using an adaptive Simpson's rule with a prescribed tolerance ϵ.

We first approximate I using the *basic* Simpson's rule over the interval $[a, b]$, i.e., using one panel, thus

$$P_1 = \frac{H}{6}\left[f(a) + 4f\left(a + \frac{H}{2}\right) + f(b)\right].$$

Divide $[a, b]$ into two equal subintervals, and apply Simpson's rule to each. For the left half we get

$$P_{11} = \frac{H}{12}\left[f(a) + 4f\left(a + \frac{H}{4}\right) + f\left(a + \frac{H}{2}\right)\right],$$

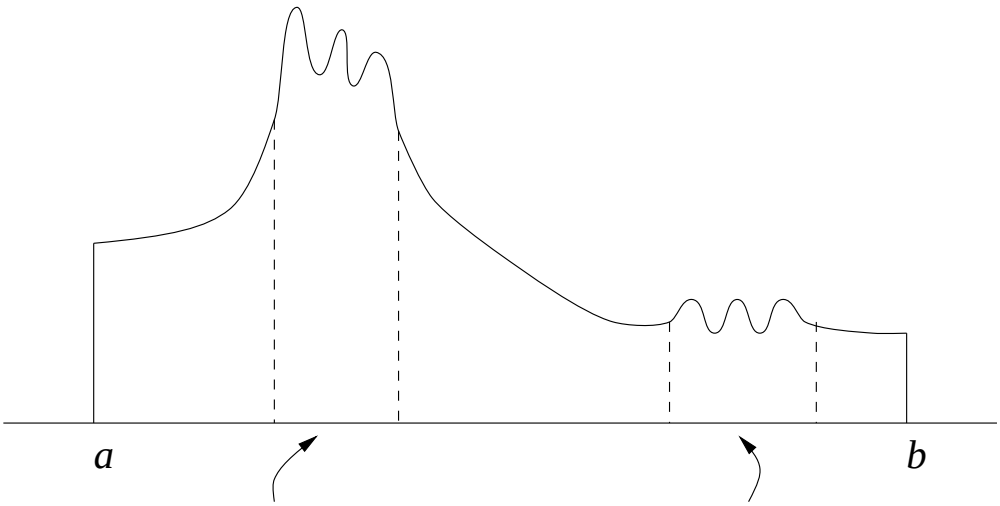


Figure 6.7: Adaptive quadrature.

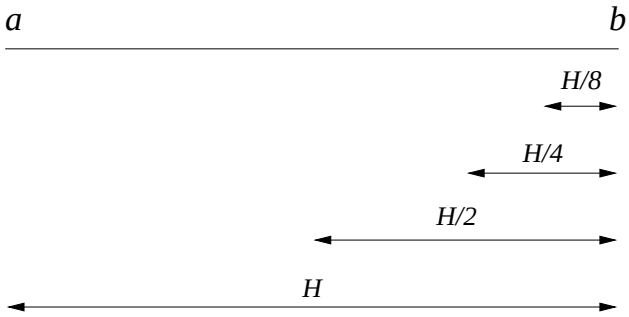


Figure 6.8: Division of interval $[a, b]$ for adaptive quadrature.

and for the right half we obtain

$$P_{12} = \frac{H}{12}\left[f\left(a + \frac{H}{2}\right) + 4f\left(a + \frac{3}{4}H\right) + f(b)\right].$$

Note that we need only compute $f\left(a + \frac{H}{4}\right)$ and $f\left(a + \frac{3}{4}H\right)$, since the other values of f are available from the previous *level*. Now, compute

$$Q = P_{11} + P_{12}$$

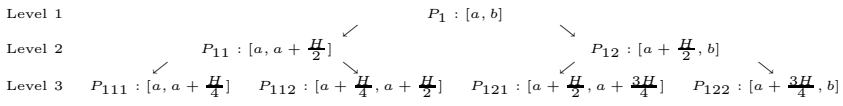
and compare with P_1. If $|P_1 - Q| \leq \epsilon$ for a prescribed tolerance ϵ, report Q as the desired approximation to I. If not, set the right half $\left[a + \frac{H}{2}, b\right]$ aside for the moment and proceed in the same way with the left half. In other words, compute

$$\begin{aligned} P_{111} &= \frac{H}{24}\left[f(a) + 4f\left(a + \frac{H}{8}\right) + f\left(a + \frac{H}{4}\right)\right], \\ P_{112} &= \frac{H}{24}\left[f\left(a + \frac{H}{4}\right) + 4f\left(a + \frac{3}{8}H\right) + f\left(a + \frac{H}{2}\right)\right]. \end{aligned}$$

We accept $(P_{111} + P_{112})$ as an approximation to $\int_a^{a+H/2} f(x)dx$ if

$$|P_{11} - (P_{111} + P_{112})| \leq \frac{\epsilon}{2},$$

and repeat the process for the right half $\left[a + \frac{H}{2}, b\right]$. Thus, we have constructed the following tree



If the above test fails, however, set the right half of $[a, a + \frac{H}{2}]$ aside and repeat the process for the left half $[a, a + \frac{H}{4}]$. □

In practice, we have to limit the number of levels used since the number of function evaluations increases rapidly. Hence, when we reach the maximum level permitted the last approximation is accepted and we move to the right.

In general, let us assume that on an interval $[x_i, x_{i+1}]$ the basic Simpson's rule yields an approximation S_i to the true value $I_i = \int_{x_i}^{x_{i+1}} f(x)dx$. Then

$$I_i - S_i = -\frac{(h_i/2)^5}{90} f^{(iv)}(z_i) \tag{6.5}$$

where $x_i < z_i < x_{i+1}$ and $h_i = x_{i+1} - x_i$. Let $S_i^{(L)}$ and $S_i^{(R)}$ be the results of the basic Simpson's rules on $[x_i, x_i + \frac{h_i}{2}]$ and $[x_i + \frac{h_i}{2}, x_{i+1}]$, respectively. Also, let $Q_i = S_i^{(L)} + S_i^{(R)}$, then

$$I_i - Q_i = -\frac{1}{2^4} \cdot \frac{(h_i/2)^5}{90} f^{(iv)}(y_i) \tag{6.6}$$

where $x_i < y_i < x_{i+1}$. Assuming that h_i is small enough so that $f^{(iv)}x)$ is essentially constant, i.e., $f^{(iv)}(\eta) \cong f^{(iv)}(y_i) \cong f^{(iv)}(z_i)$, equations (6.5) and (6.6) yield

$$Q_i - S_i \cong -\frac{(h_i/2)^5}{90} f^{(iv)}(\eta) \left[1 - \frac{1}{2^4}\right]$$

or

$$I_i \cong Q_i + \frac{1}{2^4 - 1}[Q_i - S_i]. \tag{6.7}$$

Note that (6.7) is one step of *Richardson's extrapolation*. In fact, similar to Appendix 6.7, we can show that

$$\begin{aligned} I_i &= S_i + \beta_1 h_i^4 + \beta_2 h_i^6 + \cdots + \\ I_i &= Q_i + \beta_1 \left(\frac{h_i}{2}\right)^4 + \beta_2 \left(\frac{h_i}{2}\right)^6 + \cdots \end{aligned}$$

and

$$I_i = Q_i + \frac{1}{2^4 - 1}[Q_i - S_i] + O(h_i^6).$$

Therefore, the basic task of a typical routine is to bisect each subinterval until the following inequality is satisfied

$$\frac{|S_i - Q_i|}{2^4 - 1} \leq \frac{h_i}{b - a}\epsilon$$

where ϵ is the user specified tolerance. To see this, let

$$\begin{aligned} Q &= \sum_{i=0}^{N-1} Q_i, \\ I &= \sum_{i=0}^{N-1} I_i = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx. \end{aligned}$$

Thus,

$$\begin{aligned} \left| Q - \int_a^b f(x)dx \right| &= \left| \sum_{i=0}^{N-1} (Q_i - I_i) \right| \\ &\leq \sum_{i=0}^{N-1} |Q_i - I_i| \\ &\simeq \sum_{i=0}^{N-1} \frac{1}{2^4 - 1} |Q_i - S_i| \\ &\leq \frac{\epsilon}{b-a} \sum_{i=0}^{N-1} h_i = \epsilon, \end{aligned}$$

which is the desired goal!

In this example we have assumed that the routine is using an *absolute error* criterion

$$\left| Q - \int_a^b f(x)dx \right| \leq \epsilon.$$

It is usually preferable to use the *relative error* tolerance

$$\frac{\left| Q - \int_a^b f(x)dx \right|}{\int_a^b |f(x)|dx} \leq \sigma.$$

Finally, we would like to note that it is not difficult to construct a function $f(x)$ for which a given adaptive quadrature routine fails. For example, the above adaptive Simpson's rule fails for the integral

$$I = \int_b^b f(x)dx, \qquad f(x) = x^2(x-1)^2(x-2)^2(x-3)^2(x-4)^2.$$

In this case, $P_1 = 0$, and both $P_{11} = 0$ and $P_{12} = 0$ because $f(x)$ has roots at $x = 0, 1, 2, 3$, and 4. And since $P_1 = P_{11} + P_{12} = 0$, the quadrature rule will assume the answer to be accurate!

6.6 Some Difficulties in Numerical Integration

Discontinuous functions. When $f(x)$ is has a discontinuity at the point c lying within the interval $[a, b]$, we can split the integral as shown below

$$\int_a^b f(x)dx = \int_a^b f(x)dx + \int_c^b f(x)dx.$$

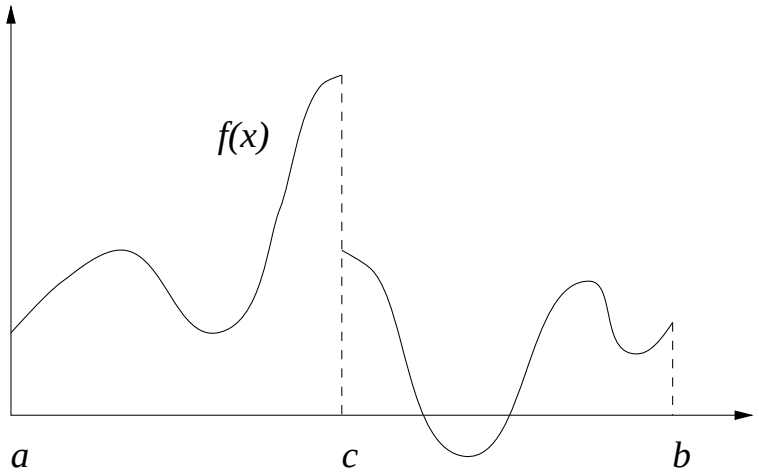


Figure 6.9: Discontinuous function.

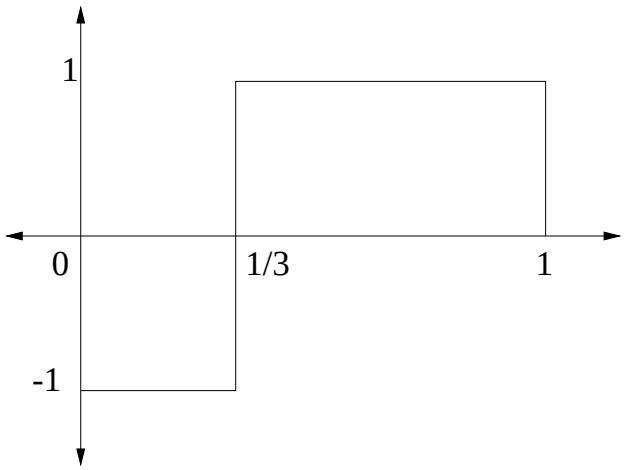


Figure 6.10: Pulse function.

Example 6.3 Compute the integral $I = \int_0^1 f(x)dx$ for the pulse function

$$f(x) = \begin{cases} 1 & x \geq 1/3 \\ -1 & x < 1/3 \end{cases}$$

Without splitting, the adaptive Simpson's procedure converges slowly:

$$P_1 = 2/3, \qquad P_{11} = -1/3, \qquad P_{12} = 1/2, \qquad P_1 \neq P_{11} + P_{12},$$

and

$$P_{111} = 1/4, \qquad P_{112} = 1/6, \qquad P_{11} \neq P_{111} + P_{112}, \qquad \ldots$$

and so on. On the other hand, we get immediate convergence by splitting the integral

$$I = \int_a^{1/3} f(x)dx + \int_{1/3}^1 f(x)dx.$$

□

Integration over infinite interval. When the range of integration is infinite, i.e.,

$$I = \int_a^\infty f(x)dx, \qquad a > 0.$$

we can resort to change of variables. Let $y = \frac{1}{x}$, then

$$I = \int_0^{1/a} f\left(\frac{1}{y}\right)\frac{dy}{y^2}.$$

Example 6.4 Compute the infinite integral

$$I = \int_1^\infty \frac{e^{-x}}{x}dx.$$

Let $y = \frac{1}{x}$; then $x = 1 \Rightarrow y = 1$, and $x = \infty \Rightarrow y = 0$. Moreover, $dx = \frac{-dy}{y^2}$. Hence,

$$I = \int_0^1 \frac{e^{-1/y}}{y}dy = \int_0^1 g(y)dy,$$

where we can easily evaluate $g(y)$ for any $y > 0$. For $y = 0$ we see that

$$\lim_{y\to 0}\frac{e^{-1/y}}{y} = \lim_{y\to 0}\frac{1}{y(1 + y^{-1} + \frac{1}{2!}y^{-2} + \cdots)} = 0.$$

□

Singularity. Singularity at a point $c \in [a, b]$ is tackled as follows: We split the function into two parts

$$I = \int_a^b f(x)dx = \int_a^b f_1(x)dx + \int_a^b f_2(x)dx,$$

where $f_1(x)$ is smooth and can be integrated using an adaptive procedure, and $f_2(x)$ contains the singularity but can be integrated analytically.

Example 6.5 Compute the integral

$$I = \int_0^1 \frac{\cos x}{\sqrt{x}}dx$$

which has a singularity at $x = 0$.

Clearly, the function can be split as follows

$$\frac{\cos x}{\sqrt{x}} = \frac{\cos x - 1}{\sqrt{x}} + \frac{1}{\sqrt{x}} = f_1(x) + f_2(x).$$

Even though $f_2(x)$ has a singularity at $x = 0$, its integral can be obtained analytically,

$$\int_0^1 f_2(x)dx = \int_0^1 \frac{1}{\sqrt{x}}dx = 2.$$

Integral of $f_1(x)$ can now be handled by an adaptive quadrature routine provided we observe that

$$\begin{aligned}\lim_{x\to 0}\frac{\cos x - 1}{\sqrt{x}} &= \lim_{x\to 0}\frac{1}{\sqrt{x}}\left[-1 + \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots\right)\right] \\ &= \lim_{x\to 0}\left[-\frac{x^{3/2}}{2!} + \frac{x^{7/2}}{4!} - \frac{x^{11/2}}{6!} + \cdots\right] \\ &= 0,\end{aligned}$$

and in the neighborhood of $x = 0$, $\cos x - 1 \simeq -\frac{x^2}{2}$. □

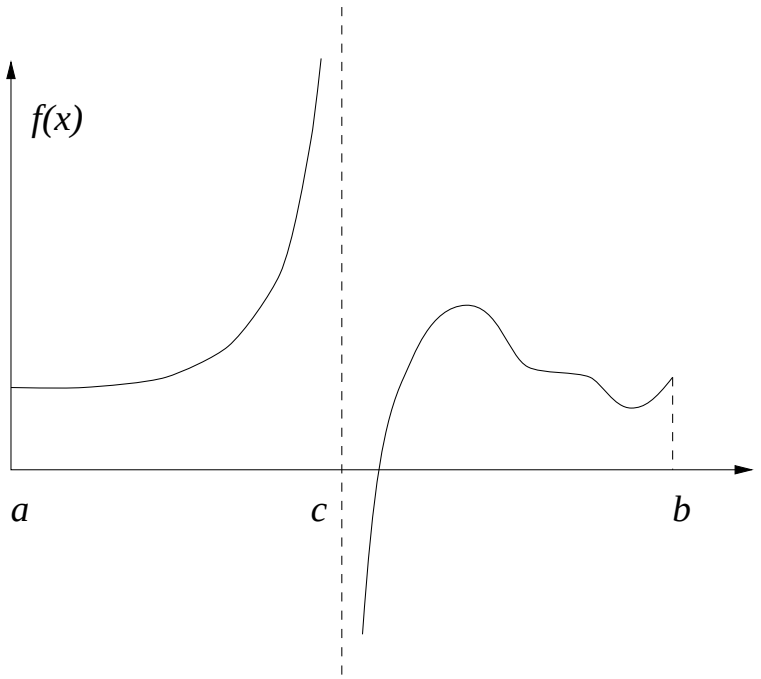


Figure 6.11: Singular function.

6.7 Appendix: Numerical Integration

We wish to show that the truncation error in trapezoidal rule is given by the following

$$I\{f\} = \int_a^b f(x)dx = T(h) + \alpha_1 h^2 + \alpha_2 h^4 + \alpha_3 h^6 + \cdots$$

where

$$T(h) = \frac{h}{2}\left[f_0 + 2\sum_{i=1}^{N-1} f_i + f_N\right], \qquad h = \frac{b-a}{N}.$$

Consider the panel $[x_i, x_{i+1}]$ with the midpoint $y_i = (x_i + x_{i+1})/2$. Then

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

Observing that

$$\int_{x_i}^{x_{i+1}} (x-y_i)^m dx = \begin{cases} h & m=0 \\ 0 & m=1 \\ \frac{h^3}{12} & m=2 \\ 0 & m=3 \\ \frac{h^5}{80} & m=4 \end{cases}$$

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

$$\int_{x_i}^{x_{i+1}} f(x)dx = hf(y_i) + \frac{h^3}{24}f''(y_i) + \frac{h^5}{1920}f^{(iv)}(y_i) + \cdots \tag{6.8}$$

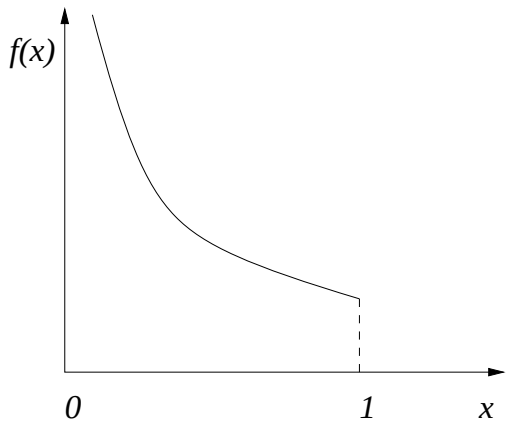


Figure 6.12: Graph of $f(x) = \frac{\cos x}{\sqrt{x}}$.

However,

$$\begin{aligned} f(x_i) &= f(y_i) - \frac{h}{2}f'(y_i) + \frac{h^2}{8}f''(y_i) - \frac{h^3}{48}f'''(y_i) + \frac{h^4}{384}f^{(iv)}(y_i) - \dots \\ f(x_{i+1}) &= f(y_i) + \frac{h}{2}f'(y_i) + \frac{h^2}{8}f''(y_i) + \frac{h^3}{48}f'''(y_i) + \frac{h^4}{384}f^{(iv)}(y_i) + \cdots \end{aligned}$$

Hence,

$$T_i = \frac{h}{2}[f(x_i) + f(x_{i+1})] = hf(y_i) + \frac{h^3}{8}f''(y_i) + \frac{h^5}{384}f^{(iv)}(y_i) + \cdots$$

Substituting in (6.8), we obtain

$$\int_{x_i}^{x_{i+1}} f(x)dx = T_i - \frac{h^3}{12}f''(y_i) - \frac{h^5}{480}f^{(iv)}(y_i) - \cdots$$

Consequently,

$$\begin{aligned} \int_a^b f(x)dx &= \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx \\ &= T(h) - \frac{h^2}{12}(b-a)f''(\eta_1) - \frac{h^4}{480}(b-a)f^{(iv)}(\eta_2)\cdots \end{aligned}$$

in which $a < \eta_1, \eta_2 < b$. In other words,

$$I\{f\} = T(h) + \alpha_1 h^2 + \alpha_2 h^4 + \cdots$$

Chapter 7

Nonlinear Equations

Given a nonlinear function $f(x)$, a value r such that $f(r) = 0$, is called a root or a zero of $f(x)$. For example, for $f(x) = xe^{-x} - 0.16064$, Fig. 7.1 gives the set of points satisfying $y = xe^{-x} - 0.16064$. Most numerical methods for solving $f(x) = 0$ require only the ability to evaluate $f(x)$ for any x.

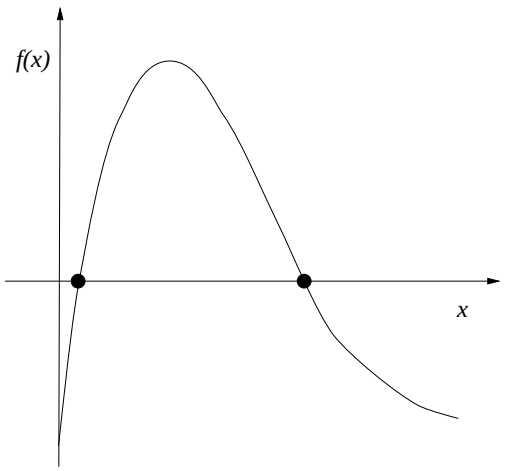


Figure 7.1: The function $f(x) = xe^{-x} - 0.16064$ has two positive roots.

7.1 The Bisection Method

This is the simplest method for obtaining a root of the function $f(x)$ that lies in an interval (a, b) for which $f(a) * f(b) < 0$. Consider the function shown in Fig. 7.2, where we assume that only *one* root exists in (a, b). The bisection method divides (a, b) into two equal halves, and the subinterval containing the root is selected. This half is further divided into two equal halves, and the process is repeated. After several steps, the interval containing the root reduces to a size smaller than the specified tolerance. The mid point of the smallest interval is taken to be the root.

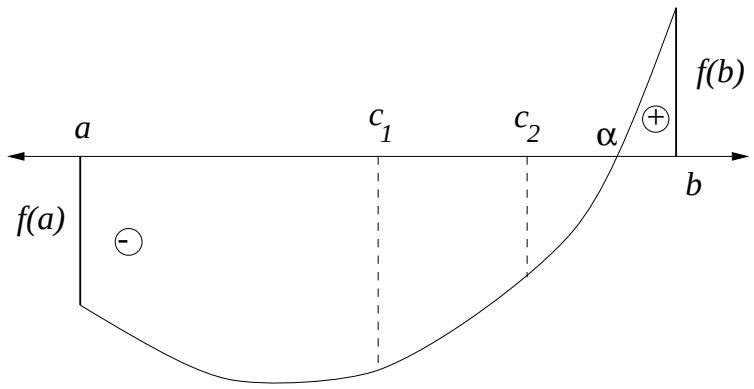


Figure 7.2: Bisection method.

Bisection Method

1. Let $f(a) * f(b) < 0$.
2. $c = (a + b)/2$.
3. if sign$[f(c)]$ = sign$[f(a)]$, then $a = c$, else $b = c$.
4. if $|b - a| < \epsilon$ (specified tolerance),
   then $\alpha = (a + b)/2$ (root), stop,
   else go to step 2.

The following points must be kept in mind.

1. For floating point arithmetic with $\beta \neq 2$, expressing c as $[a + (b - a)/2]$ is more accurate.
2. The algorithm converges linearly to the root α, i.e., the ratio e_{k+1}/e_k is a constant, where e_k is the error $\alpha - c$ at the kth step. This is implied from the fact that $|\alpha - c_2| < |\alpha - c_1|$, where c_1 and c_2 are midpoints of the interval in two consecutive iterations.

The work required by this algorithm is summarized in Table 7.1

Step	Interval Width	Function Evaluations	
		Number	Value
0	H	2	$f(a), f(b)$
1	H/2	1	$f(c)$
2	H/4	1	$f(c)$
3	H/8	1	$f(c)$
⋮	⋮	⋮	⋮
k	H/2^k	1	$f(c)$
⋮	⋮	⋮	⋮

Table 7.1: Operation count in bisection method for initial interval width $H = b - a$.

Example 7.1 How many function evaluations are required to obtain the root α with an error of 2^{-10} for an initial interval width $H = b - a = 1$.?

Solution At step k, width of interval is $(b - a)/2^k = 2^{-k}$. In order to have an error not exceeding 2^{-10}, we must have $2^{-k}/2 = 2^{-10}$, therefore $k = 9$, i.e., the number of function evaluations is 9+2 = 11. □

7.2 Newton's Method

Unlike the bisection method, the Newton scheme does not generate a sequence of nested intervals that bracket the root, but rather a sequence of points that with "luck", converge to the root. Given an initial estimate for the root, say x_0, Newton's method generates a sequence of estimates x_k, for $k = 1, 2, \ldots$ that are expected to be improved approximations of the root. At the kth iteration, the next estimate for the root x_{k+1} is obtained by intersecting the tangent to $f(x)$ from the point $(x_k, f(x_k))$ with the x-axis (See Fig. 7.3). The equation for the tangent is

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

and at the point of intersection with the x-axis, $f(x_{k+1}) = 0$, giving the following formula for x_{k+1}

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

Newton's Method

Let x_0 be an initial estimate for the root.
for $k = 1, 2, \ldots$
    $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
    if $|f(x_{k+1})| \leq \epsilon$, stop
end

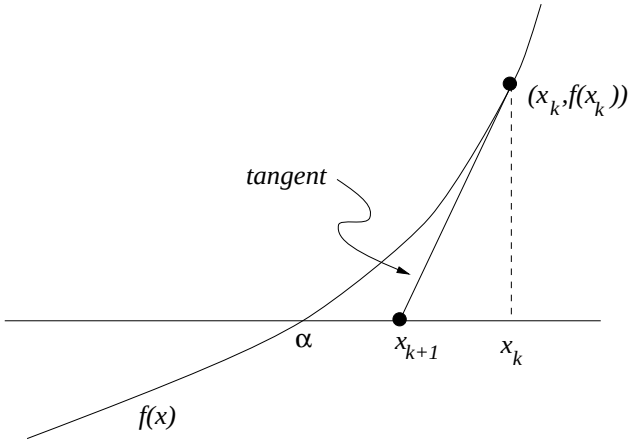


Figure 7.3: Newton's method.

Example 7.2 Obtain the square root of a positive number a.

Solution We need to obtain the root of

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

Newton's method for finding the root is given as follows:

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

For $a = 4$, we have

$$x_0 = 1, \qquad x_1 = \frac{1}{2}\left(1 + \frac{4}{1}\right) = \frac{5}{2}, \qquad x_2 = \frac{1}{2}\left(\frac{5}{2} + \frac{8}{5}\right) = \frac{41}{20}, \text{ etc.}$$

□

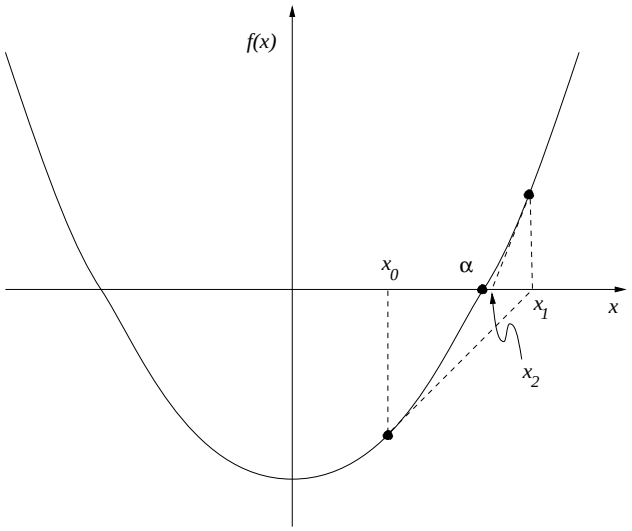


Figure 7.4: Newton's method for finding square root of a positive number.

Example 7.3 Obtain the reciprocal of a number a.

Solution We have to obtain the root of

$$f(x) = \frac{1}{x} - a$$

Newton's method for the reciprocal of a number is given as follows:

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

For $a = 3$,

$$x_0 = \frac{1}{2}, \qquad x_1 = \frac{1}{2}(2 - \frac{3}{2}) = \frac{1}{4}, \qquad x_2 = \frac{1}{4}(2 - \frac{3}{4}) = \frac{5}{16}, \text{ etc.}$$

□

7.2.1 Convergence of Newton's Method

In Example 2, if x_0 were chosen zero, the method would have failed as $f'(x_0) = 0$. Similarly, in Example 3, if x_0 were chosen $\frac{2}{a}$, then $x_1 = x_0(2 - a * x_0) = 0$; and since, $f(x_1) = \infty$, the method fails.

Theorem 7.1 *If r is a root of $f(x)$, i.e., $f(r) = 0$, such that $f'(r) \neq 0$, and $f'(x)$ is a continuous function, then there is an open interval containing the root such that for any x_0 in this interval, Newton's iterates converge to the root r.*

Example 7.4 Non-convergence due to cyclic iterates. Let $f(x) = \frac{x}{|x|^{\frac{1}{2}}}$. This function causes Newton's method to cycle between the first two iterates, irrespective of the initial iterate x_0.

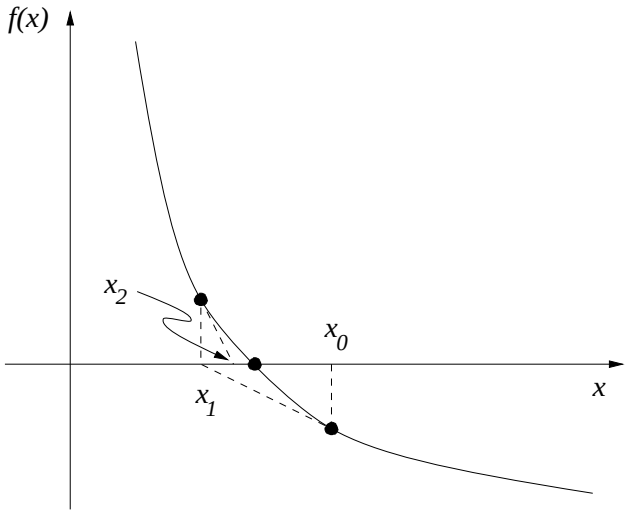


Figure 7.5: Newton's method for the reciprocal of a number.

Example 7.5 Conditional convergence of Newton's method.

$$x_{k+1} = x_k - \frac{f_k}{f'_k}$$

Consider the function

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

whose derivative is given by

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

For two successive iterates to be identical in magnitude with opposite sign, i.e., $x_{k+1} = -x_k$, we must have

$$-x_k = x_k - \left(\frac{x_k}{1+x_k^2}\right)\left(\frac{(1+x_k^2)^2}{1-x_k^2}\right),$$

or

$$2x_k = x_k \frac{(1+x_k^2)}{(1-x_k^2)},$$

Therefore, $x_k = \frac{1}{\sqrt{3}}$.

Example 7.6 Divergence of Newton's method. See Fig. 7.8

Example 7.7 Newton's method trapped in a local minima. See Fig. 7.9

7.2.2 Order of convergence

Let us now try to determine how fast Newton's method converges to the root.

Theorem 7.2 *If $f(r) = 0$; $f'(r) \neq 0$, and $f''(x)$ is continuous, then for x_0 close enough to r,*

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \frac{f''(r)}{2f'(r)}$$

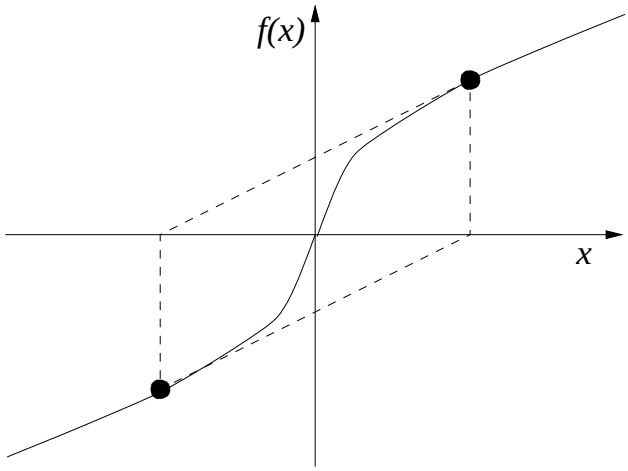


Figure 7.6: Example of non-convergence of Newton's method due to cyclic iterates.

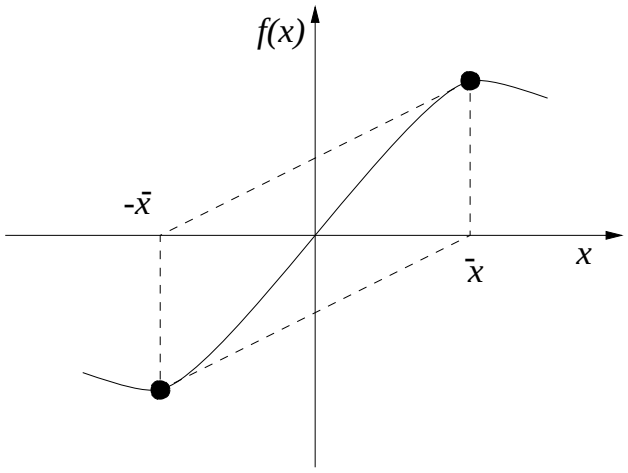


Figure 7.7: Example of conditionally convergent Newton's method.

where e_k is the error at the kth step, given by $e_k = x_k - r$.

For practical purposes, we can think of this result as stating

$$e_{k+1} \simeq \frac{f''(r)}{2f'(r)} e_k^2$$

Alternately,

$$-\log_{10}|e_{k+1}| \simeq 2\left[-\log_{10}|e_k| - \log_{10}\left|\frac{f''(r)}{2f'(r)}\right|\right]$$

The left hand side denotes the number of accurate decimal digits at the $(k+1)$th iteration and the first term on the right hand side denotes the same for the kth iteration. Since we can assume that the term $\frac{f''(r)}{2f'(r)}$ is a constant, the equation implies that the number of accurate decimal digits doubles at each iteration.

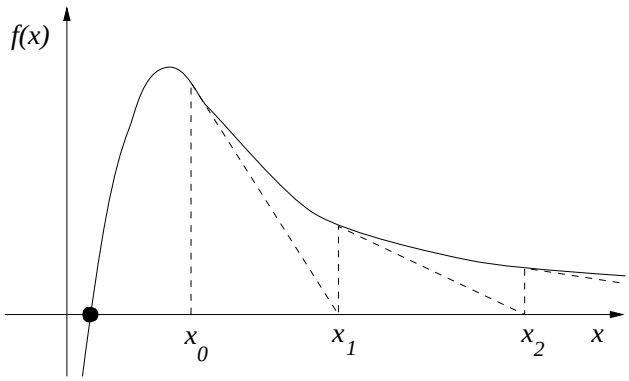


Figure 7.8: Divergence of Newton's method.

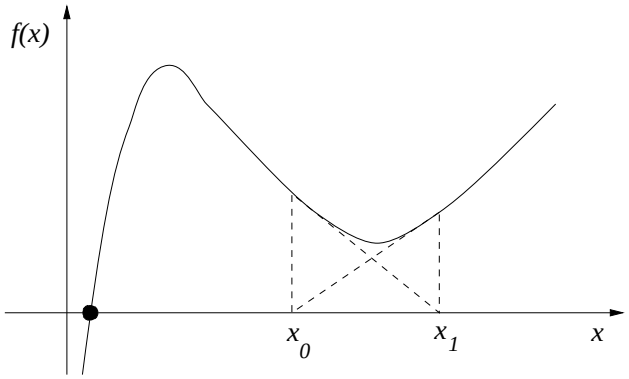


Figure 7.9: Example of Newton's method trapped in a local minimum.

Note: In general, for other iterative methods

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = C$$

where C is the asymptotic error constant, and p is the order of convergence ($p \geq 1$).

Comparison of Newton with Bisection method. At each iteration, bisection method requires a function evaluation whereas Newton's method requires a function evaluation as well as the evaluation of first derivative of the function. The first derivative may be computed using software such as Mathematica or Maple.

Multiple Roots. Note that Newton can be slower than bisection (i.e., Newton is no longer of 2^{nd} order convergence) when $f(r) = f'(r) = \cdots = f^{(m-1)}(r) = 0$. In this case, Newton's method has linear convergence with asymptotic error constant

$$C = \frac{m-1}{m}$$

where m is the multiplicity of the root.

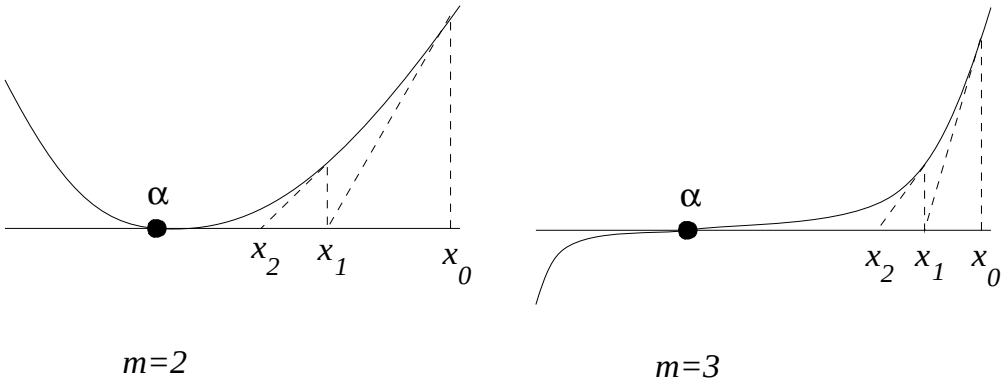


Figure 7.10: Newton's method has linear convergence in the presence of multiple roots.

7.3 Secant Method

Newton's method requires computing the first derivative of the function along with the function at each iteration. Quite often, the first derivative may not be available, or may be expensive to compute. The main motivation in developing the secant method is to overcome this drawback of Newton's method. In the secant method the first derivative is approximated via numerical differentiation. Instead of the iteration used in the Newton's method, i.e.,

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

the secant method computes the new estimate for the root as follows

$$x_{k+1} = x_k - \frac{f_k}{f[x_k, x_{k-1}]},$$

where $f[x_k, x_{k-1}]$ is an approximation to the first derivative $f'(x_k)$, and is given by

$$f[x_k, x_{k-1}] = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

The new iterate x_{k+1} is the point at which the secant of $f(x)$ at x_k and x_{k-1} intersects the x-axis. Note that this requires the function value at *two* points, x_k and x_{k-1}. Therefore, the secant method must be initialized at two points, x_0 and x_1.

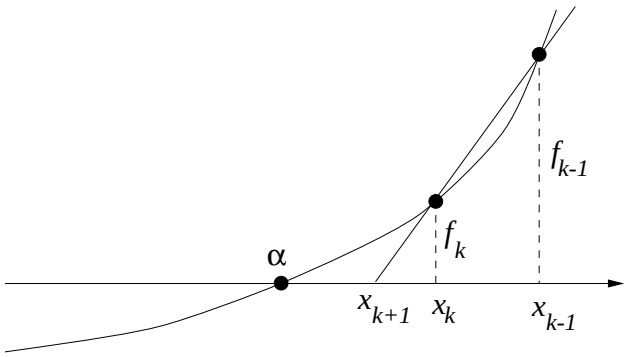


Figure 7.11: A single iteration of the Secant method.

Example 7.8 Obtain the square root of 3 using the secant method, with $x_0 = 1$, and $x_1 = 2$.

We need to compute the root of the function $f(x) = x^2 - 3$. The iterates computed by the secant method for the function are given below.

k	x_k	f_k	$f[x_k, x_{k-1}]$	$-f(x_k)/f[x_k, x_{k-1}]$
0	1.0	–2.0		
1	0.20	1.0	3.00...	–0.33...
2	1.666...	–0.222...	3.66...	0.060...
3	1.72...	–0.0165...	3.39...	0.0048...
4	1.73...	0.00031...	3.45...	–0.00009...

The actual root is $\sqrt{3} \simeq 1.7320508\ldots$.

7.3.1 Convergence of Secant Method

Theorem 7.3 *If $f(r) = 0$, $f'(r) \neq 0$, and $f'(x)$ is continuous, then there is an open interval containing r such that x_0 and x_1 in this interval will generate secant iterates $x_k \to r$ as $k \to \infty$.*

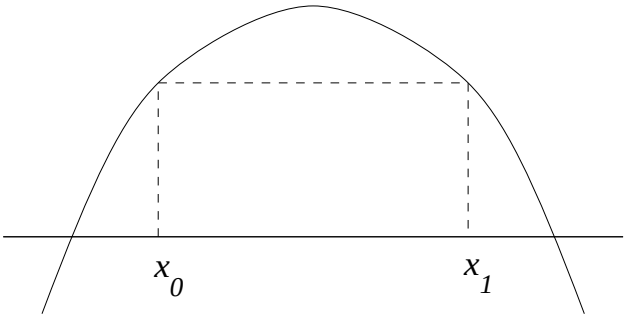


Figure 7.12: Failure of the Secant method.

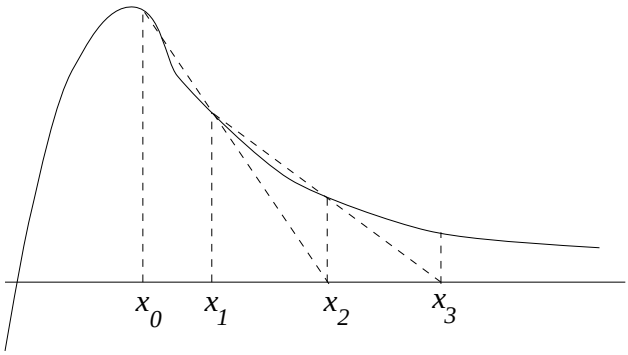


Figure 7.13: Divergence of the Secant method.

7.3.2 Order of convergence

Theorem 7.4 *If $f(r) = 0$; $f'(r) \neq 0$, and $f''(x)$ is continuous, then for x_0, x_1 close enough to r,*

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^\phi} = \left|\frac{f''(r)}{2f'(r)}\right|^{\phi-1}$$

where $e_k = x_k - r$ is the error, and $\phi = (\sqrt{5}+1)/2 \cong 1.618$ is the root of the equation $\phi^2 = \phi + 1$, and is called the **Golden Ratio**.

In practice,

$$e_{k+1} \simeq \frac{f''(r)}{2f'(r)} \cdot (e_{k-1} \cdot e_k)$$

In contrast, for Newton iterations,

$$e_{k+1} \simeq \frac{f''(r)}{2f'(r)} e_k^2.$$

Thus, Newton's method is more powerful than the secant method. A better comparison of the "effectiveness" of these two methods should also consider the "work" needed for convergence.

For the secant method,

$$\begin{aligned} |e_{k+2}| &\simeq \left|\frac{f''(r)}{2f'(r)}\right|^{\phi-1} |e_{k+1}|^\phi \\ &= \left|\frac{f''(r)}{2f'(r)}\right|^{\phi-1} \left[\left|\frac{f''(r)}{2f'(r)}\right|^{\phi-1} |e_k|^\phi\right]^\phi \\ &= \left|\frac{f''(r)}{2f'(r)}\right|^{(\phi-1)+(\phi^2-\phi)} * |e_k|^{\phi^2}. \end{aligned}$$

But, $\phi^2 = \phi + 1$, therefore

$$|e_{k+2}| \simeq \left|\frac{f''(r)}{2f'(r)}\right|^{\phi} \cdot |e_k|^{\phi+1},$$

Thus, two steps of the secant method have an order of convergence $\simeq 2.618$, which is greater than that of Newton.

7.4 Function Iteration

We are concerned with solving equations of the form

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

where $g(x)$ is a function of x, via the iterative scheme,

$$x_{k+1} = g(x_k) \qquad k = 0, 1, 2, \ldots,$$

where x_0 is chosen as an approximation of the fixed point $\alpha = g(\alpha)$. Here, α is a root of the function $f(x) = x - g(x)$.

Clearly, the iterations fail if $g(x)$ is not defined at some point x_k. Therefore, we assume that:

Assumption 1. $g(x)$ is defined on some interval $I = [a, b]$, i.e., $a \leq x \leq b$, and

Assumption 2. $a \leq g(x) \leq b$.

These assumptions are not sufficient, however, to guarantee that $x = g(x)$ has a solution $\alpha = g(\alpha)$. For example, $g(x)$ may be discontinuous. Thus, we introduce another assumption: Thus, we introduce another assumption:

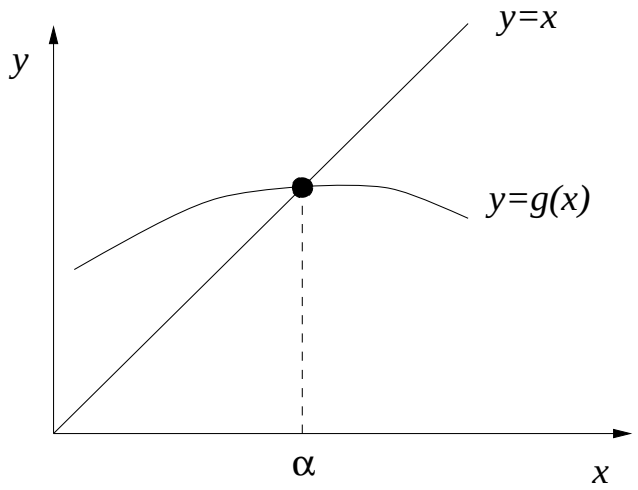


Figure 7.14: Function iteration for solving $x = g(x)$ computes the intersection of $y = x$ and $y = g(x)$.

Assumption 3. $g(x)$ is continuous on $[a, b]$.

From assumptions 1–3, we see that $x = g(x)$ has **at least one** solution if $f(x) = x - g(x)$ has a change in sign in $[a, b]$. If we wish to have the interval $[a, b]$ contain *exactly* one root, then we must make yet another assumption that guarantees that the function $g(x)$ "does not vary too rapidly" in $[a, b]$.

Assumption 4. $|g'(x)| \leq L < 1$ for $a \leq x \leq b$.

Using the differential mean-value theorem, we obtain from assumption 4

$$|g(x_1) - g(x_2)| = |g'(z)||x_1 - x_2|,$$

where $x_1 < z < x_2$. Hence

$$|g(x_1) - g(x_2)| \leq L|x_1 - x_2| < |x_1 - x_2|.$$

Theorem 7.5 *Let $g(x)$ satisfy assumptions 1–4. Then for any x_0 in $[a, b]$,*

(a) There exists one and only one root $\alpha = g(\alpha)$ in $[a, b]$.

(b) Since $a \leq g(x) \leq b$ for all x in $[a, b]$, then all the iterates $x_{k+1} = g(x_k)$, $k = 0, 1, 2, \ldots$, lie in $[a, b]$.

(c) The sequence $\{x_k\}$ defined by $x_{k+1} = g(x_k)$ converges to the unique fixed point $\alpha = g(\alpha)$.

Proof

(a) Let α_1 and α_2 be two solutions. Then from assumption 4,

$$|g(\alpha_1) - g(\alpha_2)| < |\alpha_1 - \alpha_2|,$$

or

$$|\alpha_1 - \alpha_2| < |\alpha_1 - \alpha_2|.$$

A contradiction!

(b) Obvious!

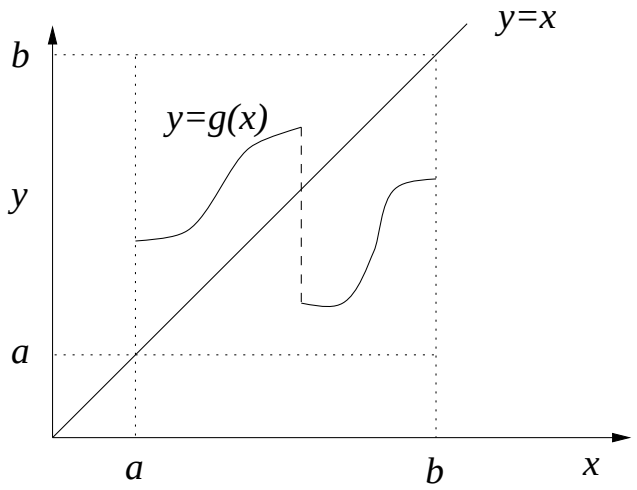


Figure 7.15: Function iteration cannot be used for a function that is discontinuous in $[a, b]$.

(c) Consider the error at the k^{th} iterate: $e_k = x_k - \alpha$. Since $x_k = g(x_{k-1})$ and $\alpha = g(\alpha)$, then by assumption 4,

$$|g(x_{k-1}) - g(\alpha)| \leq L|x_{k-1} - \alpha|,$$

i.e., $|x_k - \alpha| \leq L|x_{k-1} - \alpha|$. Therefore,

$$|e_k| \leq L|e_{k-1}|.$$

Consequently,

$$|e_1| \leq L|e_0|, \qquad |e_2| \leq L^2|e_0|, \qquad \ldots \qquad |e_k| \leq L^k|e_0|.$$

Since $|g'(x)| \leq L < 1$, then $\lim_{k\to\infty} L^k = 0$, and $\lim_{k\to\infty} |e_k| = 0$, proving convergence of the iterative scheme $x_{k+1} = g(x_k)$ under assumptions 1–4.

□

Observe that

$$e_{k+1} = x_{k+1} - \alpha = g(x_k) - \alpha = g(\alpha + e_k) - \alpha$$

From Taylor's theorem,

$$g(\alpha + e_k) = g(\alpha) + e_k g'(\beta)$$

where β lies between α and x_k. Therefore,

$$e_{k+1} = g(\alpha) + e_k g'(\beta) - \alpha = e_k g'(\beta).$$

Hence

$$\frac{|e_{k+1}|}{|e_k|} = |g'(\beta)|.$$

But, we have just proved that (Theorem 5) $\lim_{k\to\infty} x_k = \alpha$. Therefore,

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|} = \lim_{k\to\infty} |g'(\beta)| = |g'(\alpha)|$$

Consequently, the iterative scheme $x_{k+1} = g(x_k)$ under assumptions 1–4 is 1^{st} order, with an asymptotic error constant (or asymptotic convergence factor)

$$C = |g'(\alpha)|.$$

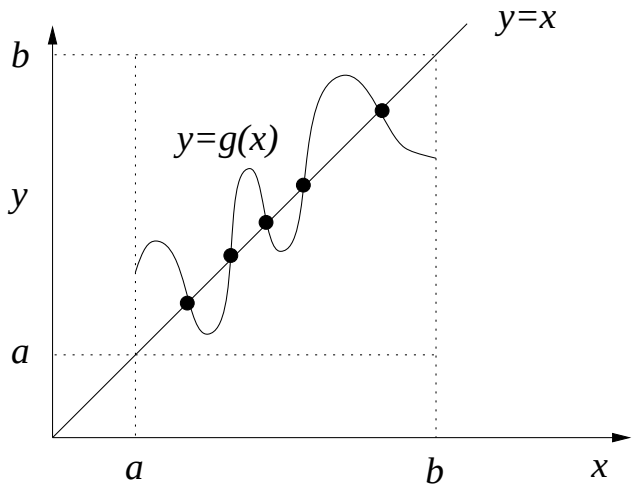


Figure 7.16: A function with multiple roots in $[a, b]$.

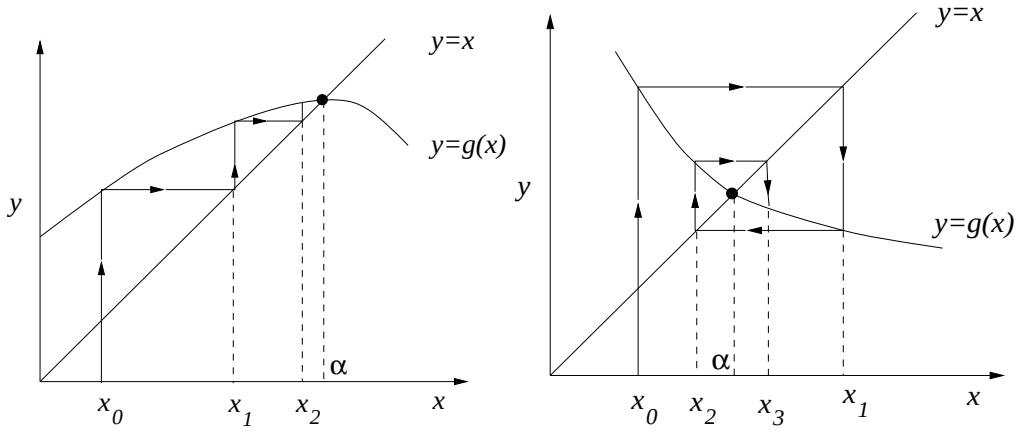


Figure 7.17: Function iteration converges for $|g'(x)| < 1$.

2^{nd} order function iterations. Let us consider the question: What happens when $g'(\alpha) = 0$? Let $g'(x)$ and $g''(x)$ be continuous on the interval containing α and x_k (for any k). Then from Taylor's Theorem

$$\begin{aligned} e_{k+1} &= g(\alpha + e_k) - \alpha \\ &= [g(\alpha) + e_k g'(\alpha) + \frac{e_k^2}{2!} g''(\gamma)] - \alpha \\ &= \frac{1}{2} e_k^2 g''(\gamma), \end{aligned}$$

where γ is between α and x_k. Therefore,

$$\frac{|e_{k+1}|}{|e_k|^2} = \frac{1}{2}|g''(\gamma)|,$$

This implies that when $g'(\alpha) = 0$, we have a 2^{nd}-order method, and

$$\lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^2} = \frac{1}{2}|g''(\alpha)|.$$

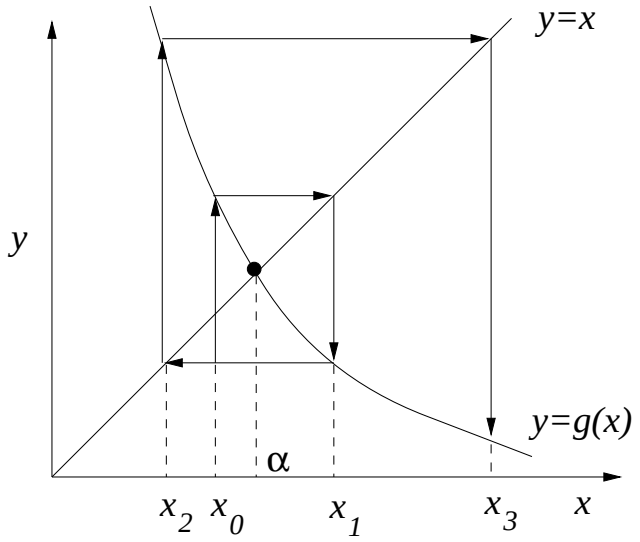


Figure 7.18: Function iteration diverges for $|g'(x)| > 1$.

Note also that if $\kappa = \frac{1}{2}|g''(\gamma)|$, then

$$|e_k| \leq \kappa |e_{k-1}|^2 \leq \kappa * \kappa^2 |e_{k-2}|^4 \leq \kappa * \kappa^2 * \kappa^4 |e_{k-3}|^8$$

which gives the relation

$$|e_k| \leq \kappa^s |e_0|^{2^k}$$

where

$$s = 1 + 2 + 4 + 8 + \cdots + 2^{k-1} = \frac{2^k - 1}{2 - 1} = 2^k - 1$$

Therefore $|e_k| \leq \kappa^{2^k - 1} |e_0|^{2^k}$, or

$$|e_k| \leq [\kappa |e_0|]^{2^k - 1} |e_0|.$$

Therefore, provided that

$$\kappa |e_0| < 1,$$

the method converges, and

$$\lim_{k \to \infty} |e_k| = 0.$$

Let us compare the number of iterations required to reduce the magnitude of the initial error $|e_0|$ by a factor of at least 10^ν for 1^{st} and 2^{nd} order methods, assuming that $|g'(x)| \leq L < 1$, and $\kappa |e_0| \leq L < 1$.

$$\begin{array}{ll} 1^{st} \text{ order method:} & |e_M| \leq L^M |e_0| \\ 2^{nd} \text{ order method:} & |e_N| \leq L^{2^N - 1} |e_0| \end{array}$$

Therefore,

$$10^{-\nu} = L^M = L^{2^N - 1} \qquad \Rightarrow M = 2^N - 1 \qquad \Rightarrow N = \log_2(M + 1)$$

In other words, if a first order method requires 255 iterations to reduce $|e_0|$ by a factor of 10^ν, a second order method will require only $\log_2(255 + 1) = 8$ iterations to reduce the same error $|e_0|$ by a factor of 10^ν.

Notice that Newton's method can be derived from the function iteration $x_{k+1} = g(x_k)$ by choosing

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

where $0 < |h(x)| < \infty$, and

$$h(x) = \frac{1}{f'(x)}$$

assuming that $f'(x)$ and $f''(x)$ are continuous on an interval $[a, b]$ containing the root α and that $f'(x) \neq 0$ on $[a, b]$.

Now,

$$g'(x) = 1 - h(x)f'(x) - h'(x)f(x),$$

or

$$g'(\alpha) = 1 - h(\alpha)f'(\alpha) - 0 = 0 \qquad \Rightarrow 2^{nd} \text{ order method}$$

or

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

Note that if α is a multiple root, then $f'(\alpha) = 0$, violating the assumption that $f'(x) \neq 0$ on $[a, b]$ which contains the root α, and Newton's method becomes only a 1^{st} order iteration.

Returning to the 2^{nd} order scheme, the asymptotic error constant

$$\begin{aligned} C &= \frac{1}{2}|g''(\alpha)| \\ &= \frac{1}{2} \cdot \frac{1}{|f'^4(\alpha)|} \left| f'^3(\alpha)f''(\alpha) + f(\alpha)f'^2(\alpha)f'''(\alpha) - 2f(\alpha)f'(\alpha)f''^2(\alpha) \right| \\ &= \frac{1}{2} \left| \frac{f''(\alpha)}{f'(\alpha)} \right| \end{aligned}$$

or

$$\lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^2} = \frac{1}{2} \left| \frac{f''(\alpha)}{f'(\alpha)} \right|.$$

Assuming x_0 is "sufficiently close" to the root α, convergence of Newton's method is assured if

$$\frac{1}{2} \left| \frac{f''(\alpha)}{f'(\alpha)} \right| e_0| < 1.$$

Chapter 8

Ordinary Differential Equations

Differential equations are an extremely important tool in various science and engineering disciplines. Laws of nature are most often expressed as different equations. Just as it is often difficult to evaluate an integral analytically, it is often difficult to solve an Ordinary Differential Equation (ODE) analytically. Simple examples in which analytical methods (e.g., separation of variables) are successful are illustrated below.

Example 8.1 Consider the equation

$$\frac{dy}{dt} = y.$$

Rearranging the terms,

$$\frac{dy}{y} = dt.$$

Taking the integral for both sides,

$$\int \frac{dy}{y} = \int dt,$$

which gives

$$\log |y| = t + C,$$

where C is a constant. Therefore,

$$y = \pm e^C \cdot e^t.$$

□

Example 8.2 Here is a somewhat more complicated equation

$$\frac{dy}{dt} = 1 - |y|y.$$

Rearranging the equation and taking the integral,

$$\int \frac{dy}{1 - |y|y} = \int dt,$$

we obtain

$$\int \frac{dy}{1 - |y|y} = t + C,$$

Case 1: Assume $y > 0$; therefore,

$$\int \frac{dy}{1 - y^2} = t + C.$$

Computing the integral, we have

$$\begin{aligned} \tanh^{-1} y &= t + C, \quad y < 1 \\ \coth^{-1} y &= t + C \quad y > 1 \end{aligned}$$

which gives the following solution

$$y = \begin{cases} \tanh(t + C), & y < 1 \\ \coth(t + C), & y > 1 \end{cases}$$

Case 2: Assume $y < 0$; therefore,

$$\int \frac{dy}{1 + y^2} = t + C,$$

which can be simplified to

$$\tan^{-1} y = t + C,$$

giving the solution

$$y = \tan(t + C)$$

□

Example 8.3 Solve the equation

$$y'(t) = -ty(t)$$

with the condition $y(0) = 1$. Again, rearranging and integrating,

$$\int \frac{dy}{y} = -\int t dt,$$

which gives the solution

$$\log_e y = -\frac{t^2}{2} + C.$$

Now, we enforce the condition $y(0) = 1$ to get

$$\log_e 1 = 0 + C \qquad \Rightarrow C = 0$$

The solution can now be written as

$$y(t) = e^{-\frac{t^2}{2}}$$

□

8.1 The Initial-Value Problem for a Single ODE

The general form of the initial value problem for a single ODE is

$$y'(t) = f(t, y(t)), \qquad y(a) = \eta$$

The first part, $y'(t) = f(t, y(t))$ gives the slope of the function $y(t)$ at t. This condition determines a family of curves in the $t - y$ plane that satisfy the given equation. To specify a particular curve, we need an additional condition. The second part, $y(a) = \eta$ specifies the function value at η. Such a condition is called the initial condition, and determines a unique solution.

Suppose

$$y'(t) = f(t) \qquad \Rightarrow \frac{dy}{dt} = f(t)$$

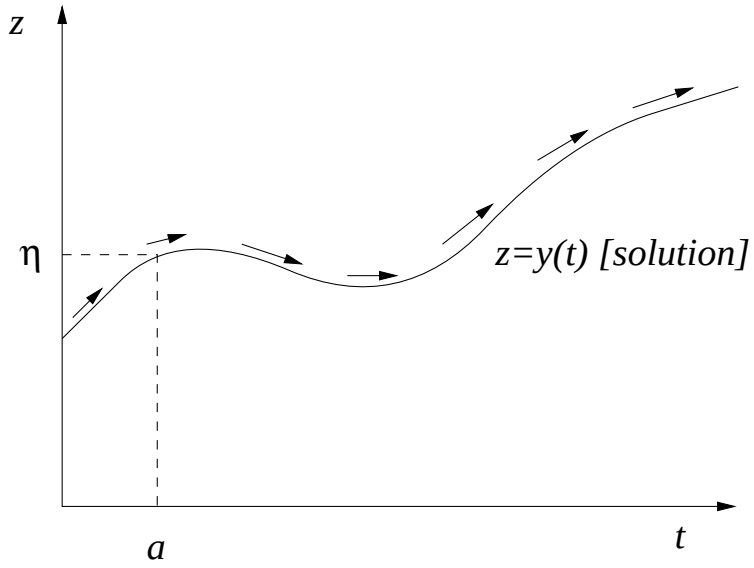


Figure 8.1: The initial value problem for a single ODE.

Therefore,

$$\frac{dy}{dt} = f(t),$$

and integrating both sides, we have

$$\int_a^t dy = \int_a^t f(t)dt.$$

Along with the initial condition, $y(a) = \eta$, we get the following solution

$$y(t) = \eta + \int_a^t f(s)ds$$

Example 8.4 Suppose that an ODE has as its solution $y(t) = 3 + Ce^{-2t}$ where C is an arbitrary constant

1. Show that there exists a solution through any point (t_0, y_0) in the $t - y$ plane.
2. Show that this is not true if instead $y(t) = 3 + C * t$.

Solution

1. We know that $3+C*e^{-2t_0} = y_0$, which implies that $C = (y_0-3)e^{2t_0}$. Therefore, the following function $y(t)$ passes through (t_0, y_0):
$$y(t) = 3 + (y_0 - 3)e^{-2(t-t_0)}$$
2. If $3 + C * t_0 = y_0$, then $C = (y_0 - 3)/t_0$. For $t_0 = 0$ and $y_0 \neq 3$, we cannot satisfy $3 + C * t_0 = y_0$.

□

Example 8.5 Obtain the general solution of

$$y'(t) = y(t)/t,$$

and also the particular solution for the initial condition $y(1) = \frac{1}{2}$.

Solution Rearranging the equation,

$$\frac{dy}{y} = \frac{dt}{t}.$$

Integrating both sides,

$$\log_e y = \log_e t + C_1,$$

and thus,

$$y = e^{C_1 + \log_e t} = e^{C_1} * t.$$

Since $y(1) = \frac{1}{2}$, we have

$$e^{C_1} = \frac{1}{2},$$

and therefore,

$$y(t) = 0.5t$$

□

Example 8.6 Solve

$$y'(t) = y(t)[1 - y(t)]$$

for the initial condition $y(0) = 1/2$.

Solution Rearranging the equation,

$$\frac{dy}{y(1-y)} = dt$$

The integral of the left hand side is given as

$$\int \frac{dy}{y(1-y)} = \int \frac{dy}{y} + \int \frac{dy}{1-y} = \log_e y - \log_e(1-y).$$

Thus,

$$\log_e y - \log_e(1-y) = t + C,$$

or

$$\log_e \frac{y}{1-y} = t + C.$$

At $t = 0$, $y(0) = \frac{1}{2}$, which implies that $C = 0$. This gives the solution

$$\frac{y(t)}{1-y(t)} = e^t$$

or

$$y(t) = \frac{e^t}{1+e^t}.$$

□

Example 8.7 Solve

$$y'(t) = \frac{y^2(t)}{t}, \qquad t \geq 1,$$

given that $y(1) = \eta$.

Solution Rearranging,

$$\frac{dy}{y^2} = \frac{dt}{t},$$

and integrating,

$$-y^{-1} = \log_e t + C.$$

The initial condition implies

$$-\frac{1}{\eta} = \log_e^1 + C \qquad \Rightarrow C = -\frac{1}{\eta}.$$

Thus,

$$y(t) = \frac{\eta}{1 - \eta \log_e t}.$$

□

8.2 Stability

The single ODE $y'(t) = y$ has the family of solutions $y(t) = \gamma e^t$. Note that the error ϵ at $t = 0$ grows by a factor of e^t at time t. On the other hand, for $y'(t) = -y(t)$, we have the family of solutions $y(t) = \gamma e^{-t}$, and the error ϵ at $t = 0$, decreases as t increases by a factor of e^{-t}. Figs. 8.2 and 8.3 represent this fact graphically.

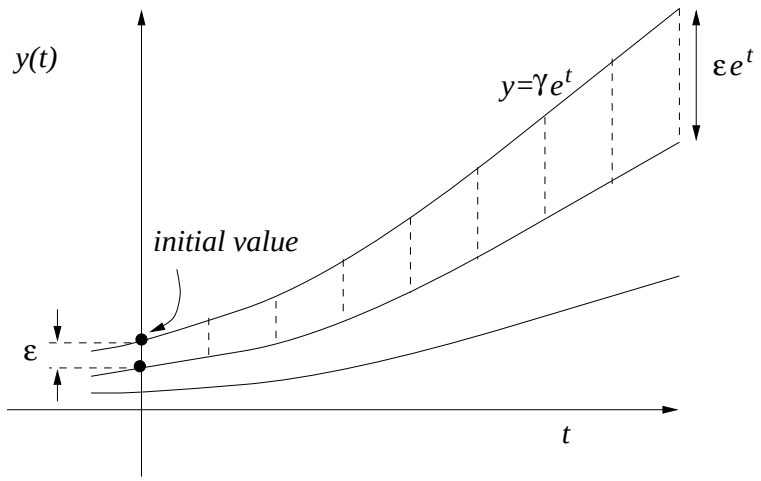


Figure 8.2: Unstable family of solutions.

8.3 Euler's Method

In order to solve the equation,

$$\frac{dy}{dt} = f(t, y),$$

let us consider the simple finite difference approximation in time

$$\frac{y_{k+1} - y_k}{h} = f(t_k, y_k) \qquad k = 0, 1, 2, \ldots, N$$

where we have chosen a uniform *mesh* for the time axis. The time instance t_k is given by

$$t_k = k * h + a \qquad h = \frac{b - a}{N}$$

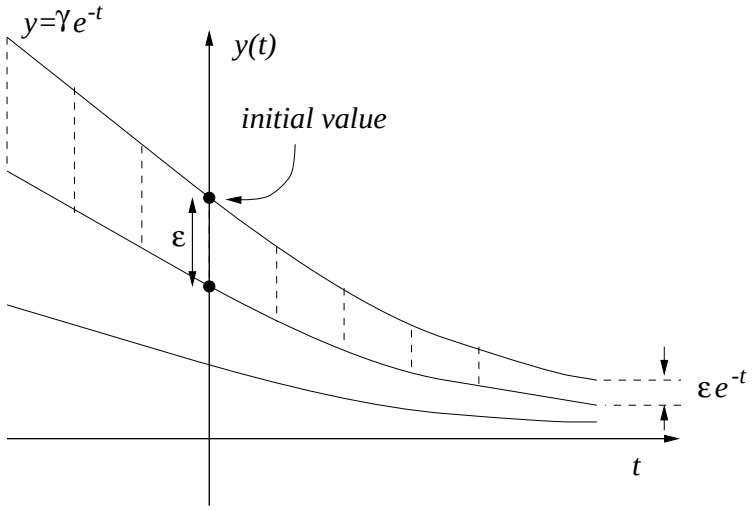


Figure 8.3: A stable family of solutions.

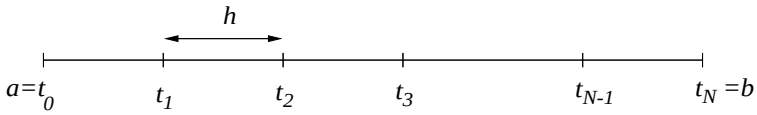


Figure 8.4: A uniform mesh for the time axis.

where a is the starting instance and b is the final time. Thus,

$$y_{k+1} = y_k + h * f(t_k, y_k).$$

Example 8.8 Consider $y'(t) = 1 - y^2(t)$ with the initial condition $y(0) = 0$.
Let $h = 0.1$; since $y(0) = 0$, we have

$$\begin{aligned} y_1 &= y_0 + (0.1)[1 - y_0^2] = 0.1, \\ y_2 &= y_1 + (0.1)(1 - y_1^2) = 0.1 + (0.1)(1 - .01) = 0.199. \end{aligned}$$

Similarly, $y_3 = 0.295, \ldots$, etc. □

8.3.1 Local and Global Error in Euler's Method

We need a measure of the error by which the exact solution of $y' = f(t, y)$ fails to satisfy the difference equation $y_{k+1} = y_k + hf(t_k, y_k)$. This is known as the *local truncation error*, or *discretization error* and is define by ...

At time t_k we are at the point (t_k, y_k). Most likely it is not on the true solution but rather on some nearby solution $U(t)$, i.e.,

$$y_k = U(t_k), \qquad U'(t) = f(t, U(t)).$$

We call $U(t)$ the local solution. Therefore, the *local error* is given by

$$d_{k+1} = y_{k+1} - U(t_{k+1}).$$

But,

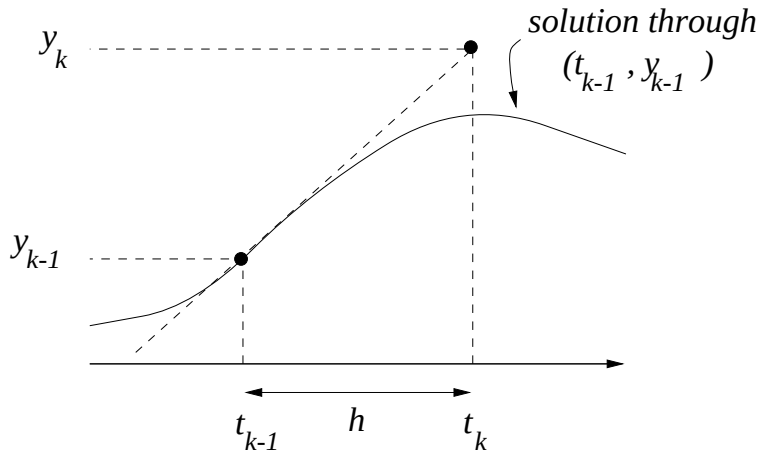


Figure 8.5: Euler's method.

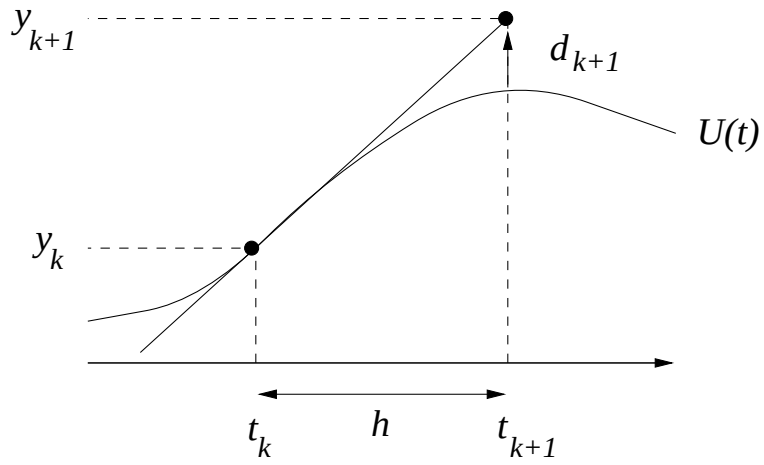


Figure 8.6: Local error.

$$y_{k+1} = U(t_k) + hU'(t_k)$$

and since

$$U(t_{k+1}) = U(t_k) + hU'(t_k) + \frac{h^2}{2}U''(\tau_{k+1}),$$

where the last term on the right hand side is the error. This is obvious since

$$U(t_{k+1}) = y_{k+1} + \frac{h^2}{2}U''(\tau_{k+1})$$

and thus,

$$d_{k+1} = -\frac{h^2}{2}U''(\tau_{k+1})$$

The important "error" here however, is the *global error*,

$$e_k = y_k - y(t_k).$$

which is the total error accrued over N steps. In fact,

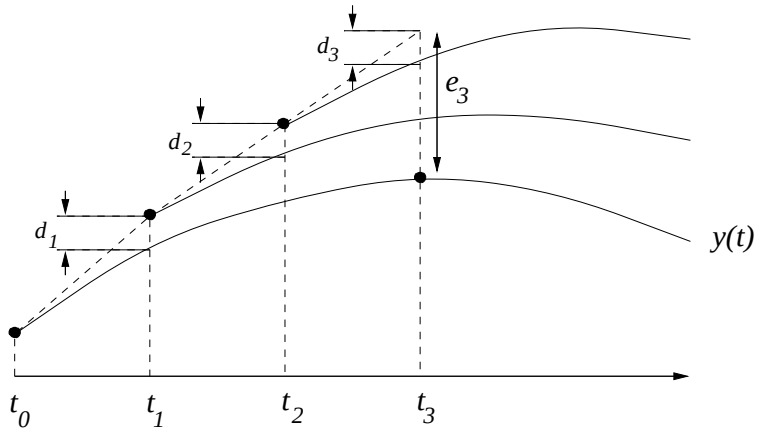


Figure 8.7: Global error.

$$e_N = \sum_{k=1}^{N-1} G_{N,k} d_k + d_N$$

where G_{ij} are the *magnification factors*.

Example 8.9 Give a closed-form expression for y_N obtained by Euler's method with step-size $\frac{1}{N}$ for

$$y'(t) = y(t), \qquad y(0) = 1.$$

Solution

$$\begin{aligned} y_{k+1} &= y_k + hf(t_k, y_k) \\ &= y_k + hy'(t_k) \\ &= y_k + hy_k \\ &= (1+h)y_k, \end{aligned}$$

where $h = \frac{1}{N}$. Therefore,

$$\begin{aligned} y_1 &= \left(1 + \frac{1}{N}\right) y_0 \\ y_2 &= \left(1 + \frac{1}{N}\right) y_1 = \left(1 + \frac{1}{N}\right)^2 y_0, \qquad \ldots \\ y_k &= \left(1 + \frac{1}{N}\right)^k y_0. \end{aligned}$$

Since $y_0 = 1$, we have

$$y_k = \left(1 + \frac{1}{N}\right)^k$$

□

Example 8.10 For the differential equation $y'(t) = y(t)$, a local error introduced into the numerical solution at mesh point t_n will have an effect on the numerical solution at some later mesh point t_N that is $e^{(t_N - t_n)}$

times greater. Suppose we have a mesh $0 < 0.1 < 0.3 < 0.6 < 1$ with local errors $0.02, 0.01, 0.05, 0.04$ introduced at the end of each of the four steps in the order given. What is the global error at $t = 1.0$?

Solution We know that the solution of

$$y'(t) = y(t)$$

is given by

$$y(t) = \gamma e^t.$$

Therefore, the global error at $t = 1.0$ is

$$\begin{aligned} e_4 &= 0.02 * e^{(1-0.1)} + 0.01 * e^{(1-0.3)} + 0.05 * e^{(1-0.6)} + 0.04 \\ &= 0.02e^{0.9} + 0.01e^{0.7} + 0.05e^{0.4} + 0.04 \\ &\cong 0.183921. \end{aligned}$$

□

8.4 Systems of ODE's

A general system of n ODE's can be written as

$$\begin{aligned} \frac{d}{dt}y^{(1)}(t) &= f_1(t, y^{(1)}(t), y^{(2)}(t), \ldots, y^{(n)}(t)), \quad y^{(1)}(a) = \eta_1 \\ \frac{d}{dt}y^{(2)}(t) &= f_2(t, y^{(1)}(t), y^{(2)}(t), \ldots, y^{(n)}(t)), \quad y^{(2)}(a) = \eta_2 \\ &\vdots \qquad\qquad\qquad\qquad \vdots \qquad\qquad \vdots \\ \frac{d}{dt}y^{(n)}(t) &= f_n(t, y^{(1)}(t), y^{(2)}(t), \ldots, y^{(n)}(t)), \quad y^{(n)}(a) = \eta_n \end{aligned}$$

Alternately, this can be expressed as the following system of ODE's

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(a) = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_n \end{pmatrix}.$$

Euler's method for this system is given by

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(t_k, \mathbf{y}_k).$$

For instance, consider the following system for $n = 2$

$$\begin{aligned} x'(t) &= f(t, x(t), y(t)), \\ y'(t) &= g(t, x(t), y(t)). \end{aligned}$$

Euler's method is given by

$$\begin{aligned} x_{k+1} &= x_k + hf(t_k, x_k, y_k), \\ y_{k+1} &= y_k + hg(t_k, x_k, y_k). \end{aligned}$$

Example 8.11 Consider the equations

$$\begin{aligned} r'(t) &= 2r(t) - .01r(t)f(t), \\ f'(t) &= -f(t) + .01r(t)f(t), \end{aligned}$$

which represent the rate of change of the populations of rabbits and foxes (predator-prey relationship).

1. Is there a particular solution to this problem for which the rabbit and fox population stay constant?
2. How many foxes must there be to bring about a decrease in the rabbit population?
3. What happens to the rabbit population if there are no foxes?

Solution

1. For the populations to stay constant
$$f'(t) = r'(t) = 0$$
which gives the solution
$$r(t) = 100, \qquad f(t) = 200$$
2. A decrease in rabbit population corresponds to the condition
$$r'(t) < 0,$$
which implies
$$r(t)[2 - .01f(t)] < 0 \qquad \Rightarrow 2 - .01f(t) < 0 \qquad \Rightarrow f(t) > 200.$$
3. If $f(t) = 0$, then
$$r'(t) = 2r(t) \qquad \Rightarrow r(t) = \gamma \cdot e^{2t}$$
which implies exponential growth!

Note: This is realistic for small population r. For large r, however, there are limiting factors such as availability of food and space.

□

Example 8.12 Determine t_n, r_n, f_n, r'_n, f'_n for $n = 0, 1$ for Euler's method applied to

$$\begin{aligned} r'(t) &= 2r(t) - 0.1r(t)f(t), \\ f'(t) &= -f(t) + 0.1r(t)f(t), \end{aligned}$$

with $r(0) = 300$, $f(0) = 150$ and $h = .001$.

Solution

$$\begin{aligned} r'_0 &= 600 - 0.1(300)(150) = -3900, \\ f'_0 &= -150 + (0.1)(300)(150) = 4350, \\ r_1 &= r_0 + hr'_0 = 300 - (.001)(3900) = 296.1, \\ f_1 &= f_0 + hf'_0 = 150 + (.001)(4350) = 154.35, \\ r'_1 &= 2r_1 - 0.1r_1 * f_1 = -3978.1, \\ f'_1 &= -f_1 + 0.1r_1f_1 = 4415.95. \end{aligned}$$

□

8.5 Taylor-series Methods

Euler's method can be improved using the Taylor-series expansion of $y(t)$. We know that

$$y(t) \approx y(t_k) + (t - t_k)y'(t_k) + \frac{(t - t_k)^2}{2}y''(t_k),$$

where

$$y'(t_k) = f(t_k, y(t_k)),$$

is provided to us. The last term, $y''(t_k)$, can be obtained simply by differentiating the ODE and evaluating the resulting expression at $t = t_k$. The following example illustrates the concept.

Example 8.13 Consider the ODE

$$y' = t^2 + y^2.$$

Differentiating w.r.t. t, we get

$$y'' = 2t + 2yy'.$$

At time t_k,

$$y'_k = t_k^2 + y_k^2,$$

and

$$y''_k = 2t_k + 2y_k y'_k.$$

Using these equations in the Taylor-series approximation,

$$y_{k+1} = y_k + hy'_k + \frac{h^2}{2}y''_k,$$

For $t_k = 2$, $y_k = 1$, $h = 0.1$, we get

$$y'_k = 5, \qquad y''_k = 14,$$

and $y_{k+1} = 1.57$. □

The local truncation error is given by

$$d_{k+1} = \frac{-h^3}{3!}y'''(\tau_{k+1}),$$

with $t_k < \tau_{k+1} < t_{k+1}$. An error like this is committed at every step, and after a number of steps proportional to $\frac{1}{h}$, it accumulates to give a global error of $O(h^2)$. Hence we have a method whose *order of accuracy* is 2, which we call *2^{nd}-order Taylor Method*.

Example 8.14 For the equation

$$y'(t) = ty^2(t),$$

where $y(2) = 1$, determine the value of $y'''(2)$.

Solution From the equation, we have

$$\begin{aligned} y' &= ty^2 \\ y'' &= y^2 + 2tyy' \\ y''' &= 2yy' + 2yy' + 2t(y')^2 + 2tyy''. \end{aligned}$$

Thus,

$$\begin{aligned} y(2) &= 1 \\ y'(2) &= 2 * 1 = 2 \\ y''(2) &= 1 + 4(1)(2) = 9 \\ y'''(2) &= 2 * 1 * 2 + 2 * 1 * 2 + 2 * 2 * (2)^2 + 2 * 2 * 1 * 9 \\ &= 4 + 4 + 16 + 36 = 60. \end{aligned}$$

□

Example 8.15 Given

$$y''(t) = ty'(t) + y^2(t),$$

and $y(1) = 1$, $y'(1) = 2$, determine $y'''(1)$.

Solution Differentiating the above equation,

$$y'''(t) = y'(t) + ty''(t) + 2y(t)y'(t),$$

and therefore,

$$y'''(1) = 2 + (2 + 1) + 2 * 1 * 2 = 9$$

□

Example 8.16 Obtain the equations for one step of the 3^{rd} order Taylor-series method applied to $y'(t) = ty(t)$.

Solution The Taylor-series expansion is given as

$$U(t) = U(t_k) + (t - t_k)U'(t_k) + \frac{(t - t_k)^2}{2!}U''(t_k) + \frac{(t - t_k)^3}{3!}U'''(t_k) + \cdots$$

Therefore, the method is

$$\begin{aligned} y_{k+1} &= y_k + hy'_k + \frac{h^2}{2!}y''_k + \frac{h^3}{3!}y'''_k \\ &= y_k + h\left[y'_k + \frac{h}{2}\left(y''_k + \frac{h}{3}y'''_k\right)\right] \end{aligned}$$

For $y' = ty$, we have

$$\begin{aligned} y'_k &= t_k y_k, \\ y''_k &= y_k + t_k y'_k, \\ y'''_k &= 2y'_k + t_k y''_k. \end{aligned}$$

□

8.6 Runge-Kutta Methods

Runge-Kutta methods are a class of methods that were devised to avoid computing analytic derivatives of y' without sacrificing the accuracy of Taylor methods.

8.6.1 Runge's Midpoint Method

The simplest method in this class of methods can be derived using the observation that

$$y(t_k + h) - y(t_k) = \int_{t_k}^{t_k+h} y'(t)dt,$$

where the right hand side requires computing the average value of $y'(t)$ over the interval $[t_k, t_k + h]$. The integral can be approximated in a number of different ways. Euler's method uses the approximation

$$\int_{t_k}^{t_k+h} y'(t)dt \approx hy'(t_k).$$

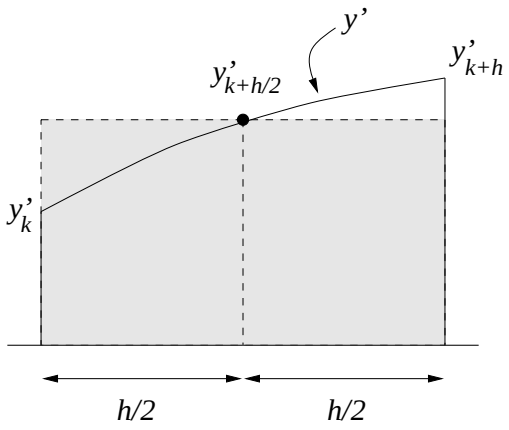


Figure 8.8: Approximating $\int_{t_k}^{t_k+h} y'(t)dt$ by the midpoint quadrature rule.

A better approximation can be obtained using the value of y' at the midpoint of the interval $[t_k, t_k + h]$

$$\int_{t_k}^{t_k+h} y'(t)dt \approx hy'(t_k + h/2).$$

Although it is not possible to compute $y'(t_k + h/2)$, we can approximate it with the value obtained using Euler's method with step size $h/2$, i.e.,

$$y'(t_k + h/2) \approx f(t_k + h/2, y(t_k + h/2))$$

where $y(t_k + h/2)$ is given by

$$y(t_k + h/2) = y(t_k) + \frac{h}{2} f(t_k, y_k).$$

Runge's midpoint method

$$\begin{aligned} g_1 &= f(t_k, y_k) \\ y_{k+\frac{1}{2}} &= y_k + \frac{h}{2} g_1 \\ g_2 &= f\left(t_k + \frac{h}{2}, y_{k+\frac{1}{2}}\right) \\ y_{k+1} &= y_k + hg_2 \end{aligned}$$

8.6.2 Runge's Trapezoid Method

Another way to approximate the integral of $y'(t)$ on the interval $[t_k, t_k + h]$ uses the trapezoid rule. The algorithm is stated below.

Runge's Trapezoid method

$$\begin{aligned} g_1 &= f(t_k, y_k) \\ g_2 &= f(t_k + h, y_k + hg_1) \\ y_{k+1} &= y_k + \frac{h}{2}(g_1 + g_2) \end{aligned}$$

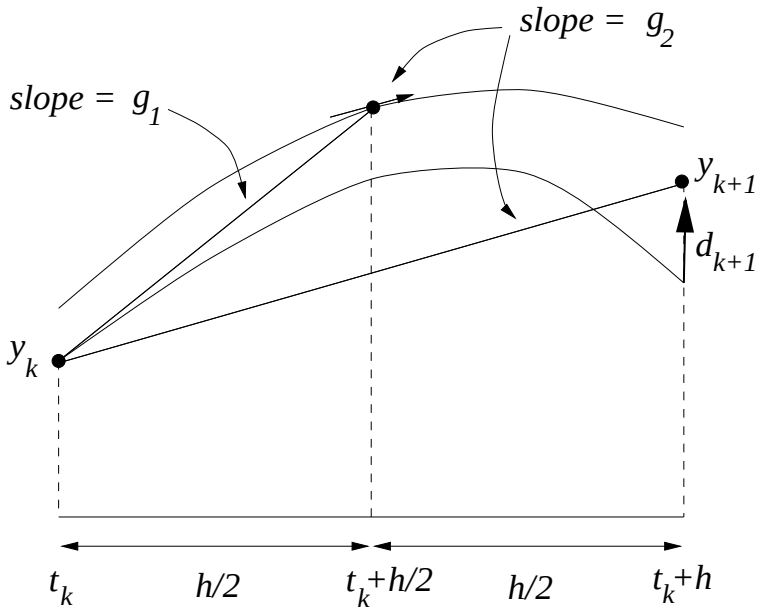


Figure 8.9: Runge's midpoint method.

Example 8.17 Suppose $t_k = 2$, $y_k = 1$, and $h = 0.1$. Then,

$$\begin{aligned} g_1 &= f(t_k, y_k) = t_k^2 + y_k^2 = 4 + 1 = 5 \\ g_2 &= f(t_k + h, y_k + hg_1) \\ &= (t_k + h)^2 + (y_k + hg_1)^2 \\ &= (2.1)^2 + (1 + 0.1 * 5)^2 = 6.66 \\ y_{k+1} &= y_k + \frac{h}{2}(g_1 + g_2) = 1 + \frac{0.1}{2}(5 + 6.66) \\ &= 1 + \frac{1.166}{2} = 1 + 0.583 = 1583. \end{aligned}$$

□

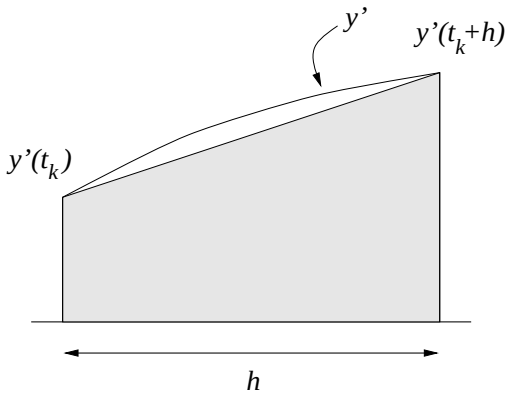


Figure 8.10: Approximating $\int_{t_k}^{t_k+h} y'(t)dt$ by the trapezoid rule.

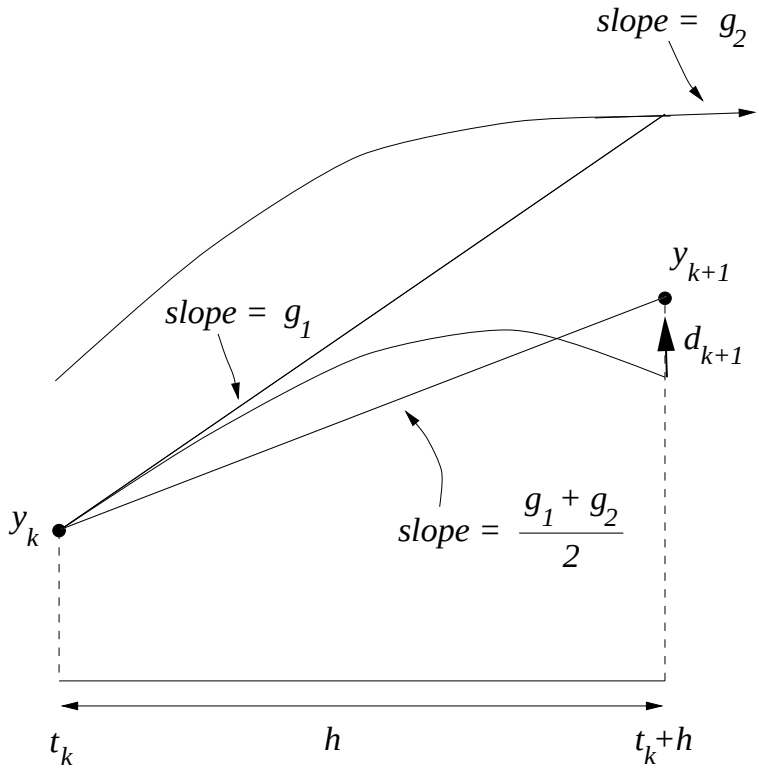


Figure 8.11: Runge's trapezoid method.

8.6.3 General 2^{nd} order 2-stage Runge-Kutta Methods

The two methods described above are special cases of general 2-stage Runge-Kutta methods which are 2^{nd} order accurate. The general form can be rewritten as

$$\begin{aligned} g_1 &= f(t_k, y_k) \\ g_2 &= f(t_k + \alpha h, y_k + \alpha h g_1) \\ y_{k+1} &= y_k + h\left[\left(1 - \frac{1}{2\alpha}\right) g_1 + \frac{1}{2\alpha} g_2\right] \end{aligned}$$

Clearly, when $\alpha = 1/2$, we obtain Runge's midpoint method, where

$$\begin{aligned} g_1 &= f(t_k, y_k) \\ g_2 &= f(t_k + \frac{1}{2}h, y_k + \frac{1}{2}h g_1) \\ y_{k+1} &= y_k + h g_2 \end{aligned}$$

while $\alpha = 1$ gives Runge's trapezoid rule

$$\begin{aligned} g_1 &= f(t_k, y_k) \\ g_2 &= f(t_k + h, y_k + h g_1) \\ y_{k+1} &= y_k + \frac{h}{2}\left[g_1 + g_2\right] \end{aligned}$$

Observe that the general form can be written as

$$\begin{aligned} y_{k+1} &= y_k + \left(1 - \frac{1}{2\alpha}\right) h y'_k + \frac{h}{2\alpha} f(t_k + \alpha h, y_k + \alpha h y'_k) \\ &= y_k + h y'_k + \frac{h^2}{2} \left[\frac{f(t_k + \alpha h, y_k + \alpha h y'_k) - f(t_k, y_k)}{\alpha h} \right]. \end{aligned}$$

Also, note that

$$\lim_{\alpha \to 0} \left[\frac{f(t_k + \alpha h, y_k + \alpha h y'_k) - f(t_k, y_k)}{\alpha h} \right] = f'(t_k, y_k) = y''(t_k).$$

Therefore, as $\alpha \to 0$, the method approaches the 2^{nd} order Taylor's method.

8.6.4 Classical 4^{th} Order Runge-Kutta Method

A popular 4^{th} order accurate method, commonly known as *the* Runge-Kutta method, uses the slopes at four specially selected points,

$$\begin{aligned} g_1 &= f(t_k, y_k), \\ g_2 &= f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} g_1\right), \\ g_3 &= f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} g_2\right), \\ g_4 &= f(t_k + h, y_k + h g_3), \end{aligned}$$

to compute the function at the next step,

$$y_{k+1} = y_k + h \left[\frac{1}{6} g_1 + \frac{1}{3} g_2 + \frac{1}{3} g_3 + \frac{1}{6} g_4 \right].$$

This method is also called *Kutta-Simpson's* method.

Example 8.18 Given

$$y' = 1 + y^2, \qquad y(0) = 0,$$

calculate $y(\pi/4)$ with step size $h = \pi/4$ using Kutta-Simpson's method.

$$\begin{aligned} g_1 &= f(t_0, y_0) = 1 + 0 = 1 \\ g_2 &= f\left(\frac{h}{2}, y_0 + \frac{h}{2} g_2\right) = 1 + \left(y_0 + \frac{h}{2} g_1\right)^2 \\ &= 1 + \left(0 + \frac{\pi}{8} * 1\right)^2 = 1.1542 \\ g_3 &= f\left(\frac{h}{2}, y_0 + \frac{h}{2} g_2\right) = 1 + \left(0 + \frac{\pi}{8} * 1.1542\right)^2 = 1.2054 \\ g_4 &= f\left(\frac{h}{2}, y_0 + h g_3\right) = 1 + \left(0 + \frac{\pi}{4} * 1.2054\right)^2 = 1.8963 \end{aligned}$$

Therefore,

$$\begin{aligned} y_1 &= y_0 + h \left[\frac{1}{6} g_1 + \frac{1}{3} g_2 + \frac{1}{3} g_3 + \frac{1}{6} g_4 \right] \\ &= 0 + \frac{\pi}{4} \left[\frac{1}{6} + \frac{1}{3}(1.1542 + 1.2054) + \frac{1}{6}(1.8963) \right] \\ &= \frac{\pi}{4} * \frac{1}{6} [2.8963 + 4.7192] = 0.99687 \end{aligned}$$

Here, y_1 has a relative error of -0.3%. In contrast, Euler's method with the step size $\pi/16$ gives the solution $y_4^{(Euler)} = 0.90188$ for the same time instant. This has a relative error of -9.8%.

This is due to the fact that Kutta-Simpson's method is 4^{th} order with $O(h^4)$ error whereas Euler's method is only 1^{st} order with $O(h)$ error. Thus, a reduction in the step size from h to $h/2$ improves the error in Kutta-Simpson's method by a factor of 16, but only by a factor of 2 in Euler's method. □