

**Numerical Methods
Natural Sciences Tripos 1B
Lecture Notes
Lent Term 1999**

© **Stuart Dalziel**

Department of Applied Mathematics and Theoretical Physics
University of Cambridge

Phone: (01223) 337911

E-mail: *s.dalziel@damtp.cam.ac.uk*

WWW: <http://www.damtp.cam.ac.uk/user/fdl/people/sd103/>

Lecture Notes: <http://www.damtp.cam.ac.uk/user/fdl/people/sd103/lectures/>

Formatting and visibility in this version:

Subsections

Sub-subsections

Fourth order subsections

All versions

- Lecture points

Common equations

Handout/lecture equations

Handout/lecture figures

Tables

CONTENTS

- Page numbers in Table of Contents may change for subsequent handouts

1 Introduction

1.1 Objective

- What can be done?
- How can we do it?
- Confidence to try
- No knowledge of computer languages expected or required

1.2 Books

General:

More specialised:

- Listed in printed notes
- *Numerical Recipes – The Art of Scientific Computing*
 - ◇ by Press, Flannery, Teukolsky & Vetterling (CUP)
 - ◇ The “bible” for many

1.3 Programming

- Programming **not** required for course
- Program listing provided only for **completeness** and to **illustrate** the approach.

1.4 Tools

- Wide variety now available.

1.4.1 Software libraries

- NAG
- IMFL
- Numerical Recipes

1.4.2 Maths systems

- Shrink-wrapped
 - ◊ Derive
 - ◊ Maple
 - ◊ Matlab
 - ◊ Mathematica
 - ◊ Reduce
- Easy to use and powerful
- Can tackle wide range of problems
- Not yet suitable for large-scale calculations

1.5 Course Credit

- Two examination questions
- First examined in 1995-96.
- Some topics not examinable
 - ◊ These are indicated by asterisks in the section headings

1.6 Versions

- Multiple versions of notes available through different routes

1.6.1 Notes distributed during lectures

- Fill in the blanks
- Contain details which may not be covered as fully in lectures
- Some discussion given in lectures not contained in notes

1.6.2 Acrobat

- On the web: <http://www.damtp.cam.ac.uk/user/fdl/people/sd103/lectures/>
 - ◊ Will try to keep up to date with handouts
- All the blanks filled in
- Some discussion given in lectures not contained in notes

1.6.3 HTML

Table 1: Correspondence between colour and characters.

- Notes from previous years in HTML format
- Includes hypertext links within notes
- Does not contain latest revisions
- Not all characters and fonts correctly supported

1.6.4 Copyright

2 Key Idea

- Much of the material in course derived from Taylor Series

◇ In 1D

$$f(x + \delta x) = \quad (1)$$

◇ In 2D

$$f(x + \delta x, y + \delta y) = \quad (2)$$

- Truncation error

◇ The terms neglected

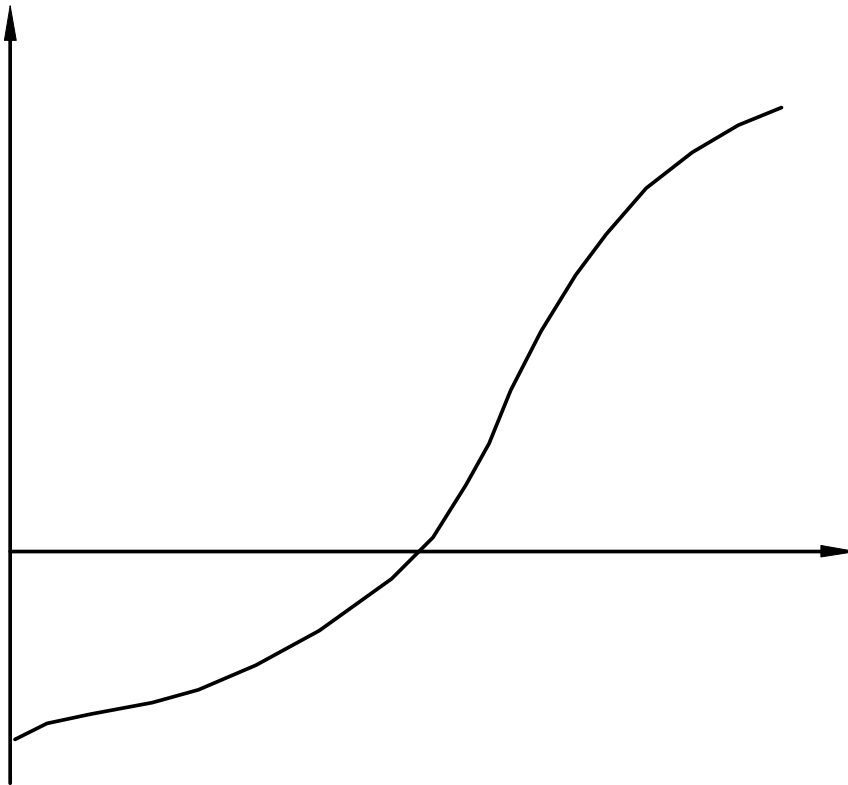
3 Root finding in one dimension

3.1 Why?

- Solution to $f(x) = 0$ often not possible analytically

3.2 Bisection

- Also known as *binary chopping*
- Simplest method
- Two initial guesses bracket root
 - ◇ Can be difficult to find suitable initial guesses
 - ◇ All subsequent guesses will also bracket root



- Algorithm:
 - ◇ $x_c =$
 - ◇ If f_c
 - ◇ elseif
 - ◇ else

◇ Repeat if root not found

3.2.1 Convergence

- x_a, x_b always bracket root
- Error estimate

$$e_n < |x_a - x_b|. \quad (3)$$

- Interval (x_a, x_b) halved for each iteration

$$e_{n+1} \sim \quad (4)$$

- If the root is x^* such that $f(x^*) = 0$, then the actual error is

$$\epsilon_n = \quad (5)$$

- General relationship between errors

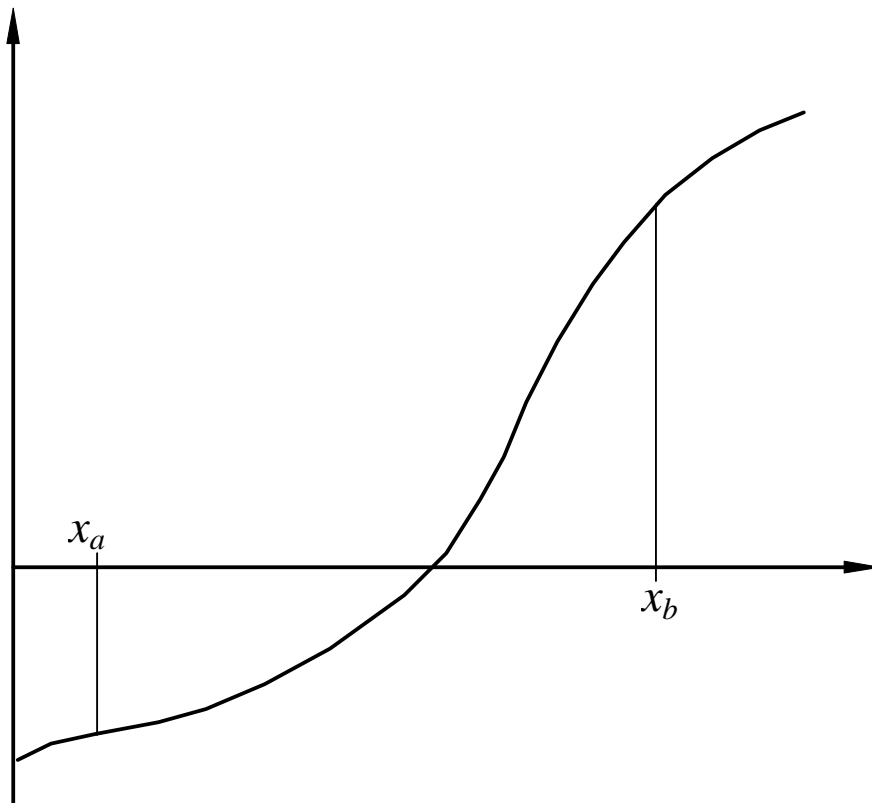
$$|\epsilon_{n+1}| \sim \quad (6)$$

- Applies asymptotically
- p gives the order of the scheme, e.g. $p = 1$ is a linear scheme
- Need $\epsilon_{n+1} < \epsilon_n$ for convergence
- For bisection method
 - ◇ Estimate $\epsilon_n \approx e_n$
 - ◇ See that $p = 1$ (first order) and $C = 1/2$, so converges

3.2.2 Criteria

- $x_n \rightarrow x^*$ as $n \rightarrow \infty$, but normally $x_n \neq x^*$
- Continue iterating until *converged*
 - ◇ $e_n < tol_1$
 - ◇ $|f(x_n)| < tol_2$
- Only continue if converging
 - ◇ $\epsilon_{n+1} < \epsilon_n$
 - ◇ Guaranteed for bisection method

3.3 Linear interpolation (regula falsi)



- Algorithm
 - ◇ Let $x_c =$
 - ◇ If f_c
 - ◇ elseif
 - ◇ else
 - ◇ Repeat until converged
- Each guess brackets root
- First order
- Exact for linear f .

3.4 Newton-Raphson

- Taylor Series about $x = x_0$

$$f(x) =$$

(7)

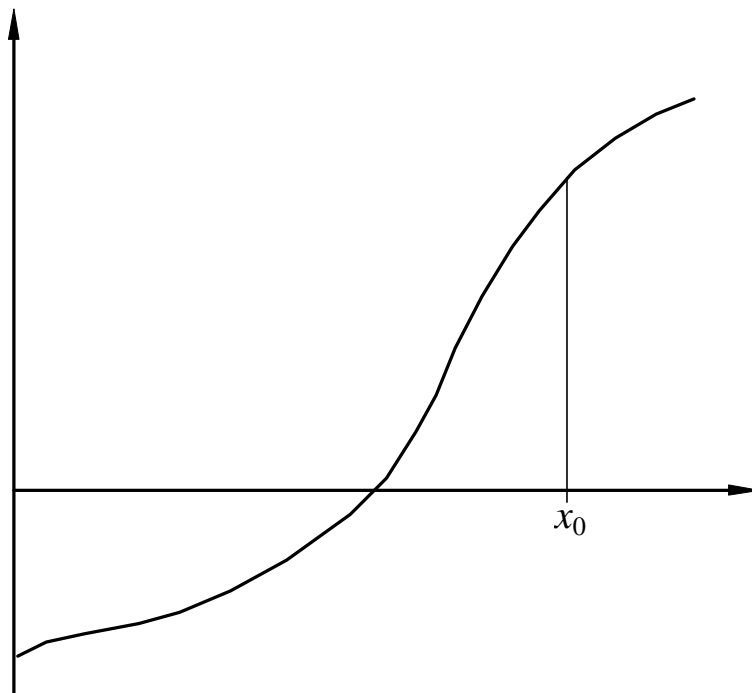
- Setting $f(x) = 0$ and neglecting second and higher-order terms gives improved estimate for root

$$x_1 = \quad (8)$$

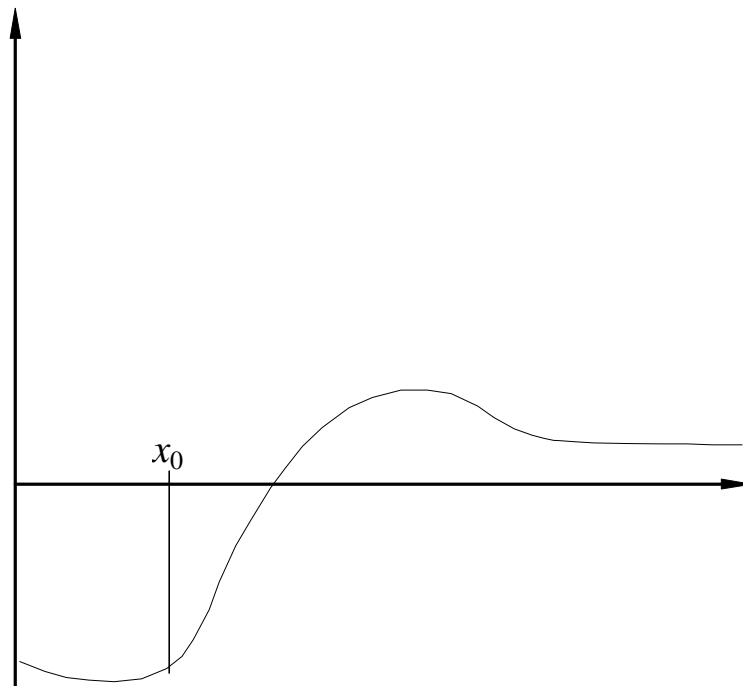
- Subsequent iterations in same manner

$$x_{n+1} = \quad (9)$$

- Geometrical interpretation in terms of tangent to curve



- *If* it converges, then Newton-Raphson efficient
- *But* may not always converge



3.4.1 Convergence

- Taylor-Series expansion around root x^*

$$f(x) = \quad (10)$$

- Substitute into iteration formula

$$\begin{aligned} \epsilon_{n+1} &= x_{n+1} - x^* \\ &= x_n - x^* - \frac{f(x_n)}{f'(x_n)} \\ &= \end{aligned}$$

◇ Since $f(x^*) = 0$

(11)

- Of form $|\epsilon_{n+1}| = C |\epsilon_n|^p$
 - ◇ $C = 1/2 f''/f'$
 - ◇ Quadratic convergence ($p = 2$)
- Convergence fails if C becomes too large in neighbourhood of root, *e.g.* if f' vanishes

3.5 Secant (chord)

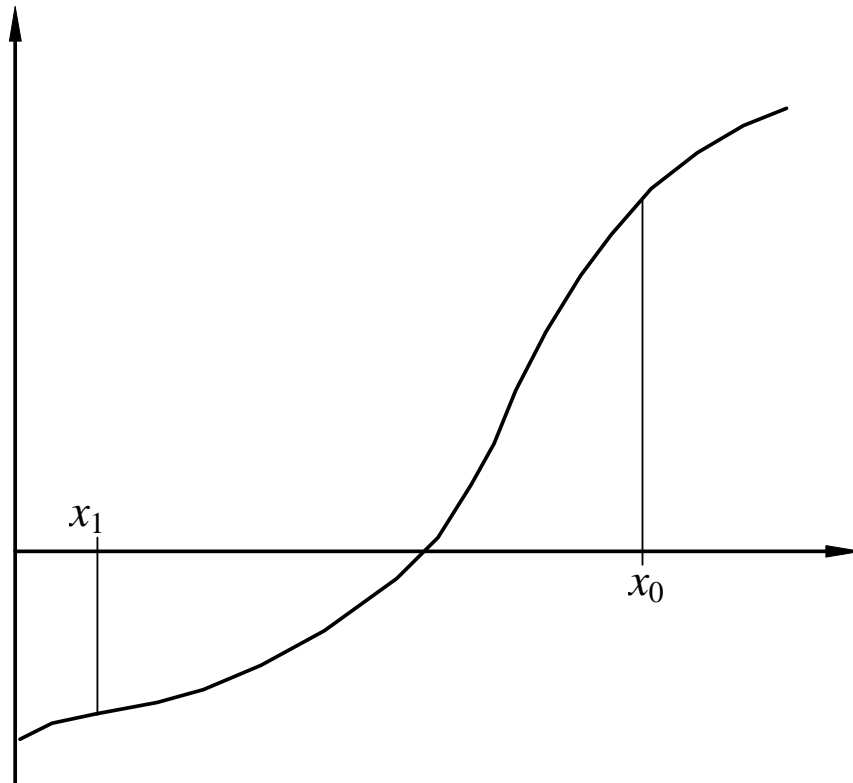
- Similar to Newton-Raphson, but
 - ◇ Approximate f' by finite difference
 - ◇ Uses two points, not just one

$$f'(x_n) \approx \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n} \quad (12)$$

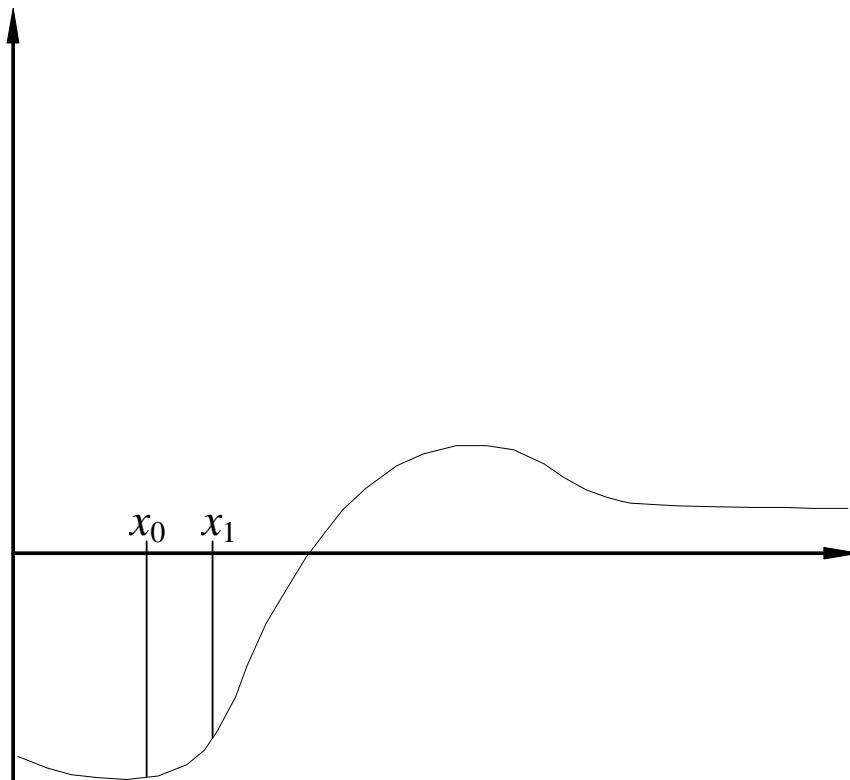
- Substitute into Newton-Raphson

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (13)$$

- Formula identical to Linear Interpolation, but
 - ◇ Use two newest points rather than those bracketting the root
 - ◇ Do not need initial guesses to bracket root



- Convergence not guaranteed



3.5.1 Convergence

- Expand about root x^*

$$f(x_n) = \tag{14a}$$

$$f(x_{n-1}) = \tag{14b}$$

- Substitute into iteration formula

$$\begin{aligned} \epsilon_{n+1} &= x_{n+1} - x^* \\ &= x_n - x^* - \frac{f(x_n)}{f(x_n) - f(x_{n-1})} (x_n - x_{n-1}) \\ &= \end{aligned}$$

(15)

- Expression for ϵ_{n+1} not of the form $C|\epsilon_n|^p$

◇ Assume we can convert to this form

$$\epsilon_{n+1} = \tag{16}$$

◇ Substitute into (15)

$$\begin{aligned} \epsilon_{n+1} &= \left(\frac{f''(x^*)}{2f'(x^*)} \right) \epsilon_n \epsilon_{n-1} \\ &= \end{aligned}$$

(17)

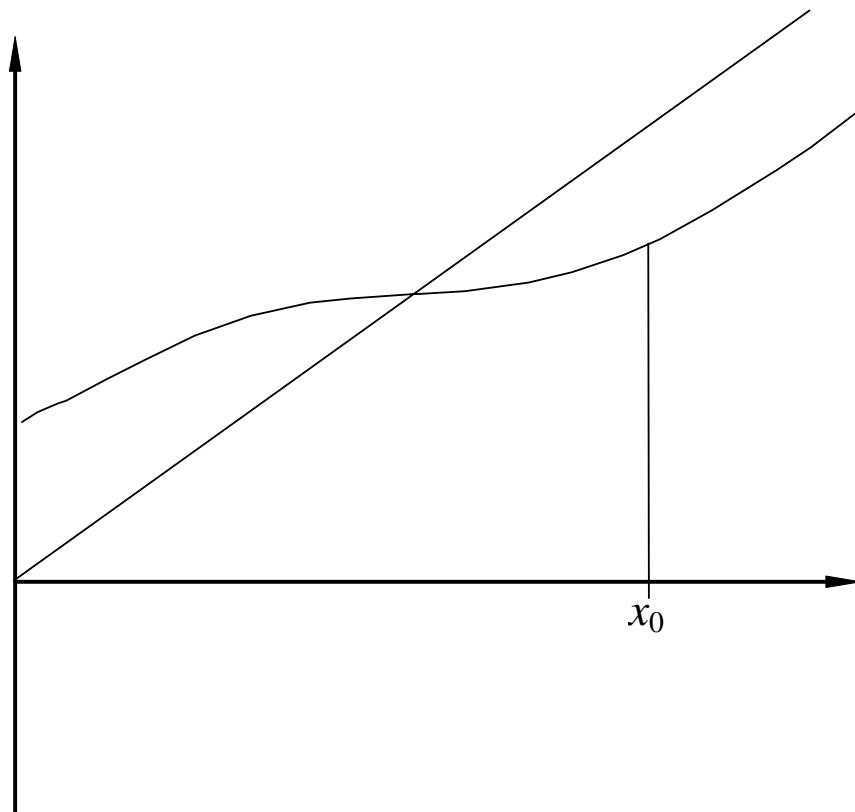
◇ Equate with assumed relationship

$$\alpha = \frac{1 + \alpha}{\alpha} =$$
$$\beta = \frac{\alpha}{1 + \alpha} =$$
(18)

3.6 Direct iteration

- Transform $f(x)$ to $g(x) = T(f(x), x)$
 - ◇ Choose $T(f, x)$ such that $x = g(x)$ corresponds to $f(x) = 0$
- Iteration formula

$$x_{n+1} =$$
(19)



3.6.1 Convergence

- Use similar approach to previous methods
- Expand $g(x)$ around x^* :

$$g(x_n) = \quad (20)$$

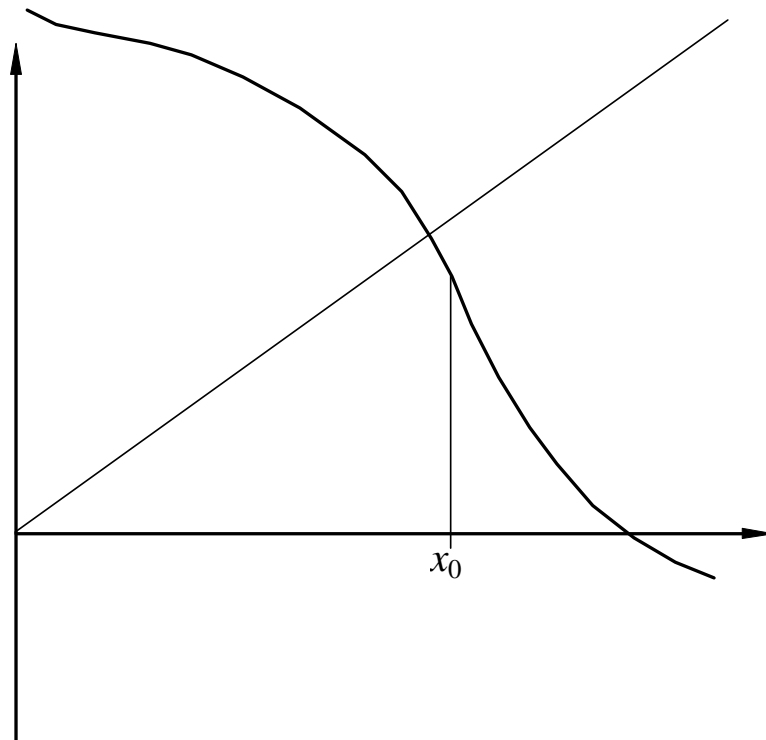
- Substitute into equation for error

$$\epsilon_{n+1} = x_{n+1} - x$$

(21)

- Method first order

- Convergence factor $C \equiv g'(x^*) < 1$
- Convergence requires $|C| < 1$
- Rate of convergence maximised if $T(f,x)$ chosen so that $g'(x)$ minimised
 ◊ Need to make sure $|g''|$ does not become too large!
- If $g'(x) < 0$ then get oscillatory behaviour
- If $|g'(x)| > 1$ then get divergence



3.7 Examples

$$f(x) = \cos x - 1/2. \tag{22}$$

3.7.1 Bisection method

- Initial guesses $x = 0$ and $x = \pi/2$.
- Expect linear convergence: $|\epsilon_{n+1}| \sim |\epsilon_n|/2$.

Iteration	Error	e_{n+1}/e_n
0	-0.261799	-0.500001909862
1	0.130900	-0.4999984721161
2	-0.0654498	-0.5000015278886
3	0.0327250	-0.4999969442322
4	-0.0163624	-0.5000036669437
5	0.00818126	-0.4999951107776
6	-0.00409059	-0.5000110008581

7	0.00204534	-0.4999755541866
8	-0.00102262	-0.5000449824959
9	0.000511356	-0.4999139542706
10	-0.000255634	-0.5001721210794
11	0.000127861	-0.4996574405018
12	-0.0000638867	-0.5006848060707
13	0.0000319871	-0.4986322611303
14	-0.0000159498	-0.5027411002019
15	0.00000801862	

3.7.2 Linear interpolation

- Initial guesses $x = 0$ and $x = \pi/2$.
- Expect linear convergence: $|\epsilon_{n+1}| \sim c|\epsilon_n|$.

Iteration	Error	e_{n+1}/e_n
0	-0.261799	0.1213205550823
1	-0.0317616	0.0963178807113
2	-0.00305921	0.09340810209172
3	-0.000285755	0.09312907910623
4	-0.0000266121	0.09310313729469
5	-0.00000247767	0.09310037252741
6	-0.000000230672	0.09310059304987
7	-0.0000000214757	0.09310010849472
8	-0.00000000199939	0.09310039562066
9	-0.000000000186144	0.09310104005501
10	-0.0000000000173302	0.09310567679542
11	-0.00000000000161354	0.09316100003719
12	-0.000000000000150319	0.09374663216227
13	-0.0000000000000140919	0.10000070962752
14	-0.0000000000000014092	0.1620777746239
15	-0.0000000000000002284	

- Convergence linear, but fast.

3.7.3 Newton-Raphson

- Initial guess: $x = \pi/2$.
- Note that can not use $x = 0$ as derivative vanishes here.
- Expect quadratic convergence: $\epsilon_{n+1} \sim c\epsilon_n^2$.

Iteration	Error	e_{n+1}/e_n	e_{n+1}/e_n^2
0	0.0235988	0.00653855280777	0.2770714107399
1	0.000154302	0.0000445311143083	0.2885971297087
2	0.00000000687124	0.000000014553	-
3	1.0E-15		
4	Machine accuracy		

- Solution found to roundoff error ($O(10^{-15})$) in three iterations.

3.7.4 Secant method

- Initial guesses $x = 0$ and $x = \pi/2$.
- Expect convergence: $\epsilon_{n+1} \sim c\epsilon_n^{1.618}$.

Iteration	Error	e_{n+1}/e_n	$ e_{n+1} / e_n ^{1.618}$
0	-0.261799	0.1213205550823	0.2777
1	-0.0317616	-0.09730712558561	0.8203
2	0.00309063	-0.009399086917554	0.3344
3	-0.0000290491	0.0008898244696049	0.5664
4	-0.0000000258486	-0.000008384051747483	0.4098
5	0.000000000000216716		
6	Machine accuracy		

- Convergence substantially faster than linear interpolation.

3.7.5 Direct iteration

3.7.5.1 Addition of x

$$x_{n+1} = g(x) = x_n + \cos x - 1/2 \tag{23}$$

- Initial guess: $x = 0$ (also works with $x = \pi/2$)
- Expect convergence: $\epsilon_{n+1} \sim g'(x^*) \epsilon_n \sim 0.13 \epsilon_n$.

Iteration	Error	e_{n+1}/e_n
0	-0.547198	0.30997006568
1	-0.169615	0.1804233116175
2	-0.0306025	0.1417596601585
3	-0.00433820	0.1350620072841
4	-0.000585926	0.1341210323488
5	-0.0000785850	0.1339937647134
6	-0.0000105299	0.1339775306508
7	-0.00000141077	0.1339750632633
8	-0.000000189008	0.1339747523914
9	-0.0000000253223	0.1339747969181
10	-0.00000000339255	0.1339744440023
11	-0.00000000454515	0.1339748963181
12	-0.000000000608936	0.1339759843399
13	-0.0000000000815828	0.1339878013503
14	-0.0000000000109311	0.1340617138257
15	-0.000000000001465442	

3.7.5.2 Multiplication by x

$$x_{n+1} = g(x) = 2x \cos x \tag{24}$$

- Initial guess: $x = \pi/2$ (fails with $x = 0$ as this is a new solution to $g(x)=x$)
- Expect convergence: $\epsilon_{n+1} \sim g'(x^*) \epsilon_n \sim 0.81 \epsilon_n$.

Iteration	Error	e_{n+1}/e_n
0	0.0635232	-0.9577980958138
1	-0.0608424	-0.6773664418235
2	0.0412126	-0.9070721090152
3	-0.0373828	-0.7297714456916
4	0.0272809	-0.8754733164962
5	-0.0238837	-0.7600455540808
6	0.0181527	-0.854809477378
7	-0.0155171	-0.778843985023
8	0.0120854	-0.8410892481838
9	-0.0101649	-0.7908921878228
10	0.00803934	-0.8319464035605
11	-0.00668830	-0.7987216482514
12	0.00534209	-0.8258546748557
13	-0.00441179	-0.8038528579103
14	0.00354643	-0.8218010788314
15	-0.00291446	

3.7.5.3 Approximating $f'(x)$

$$f(x) = f(x) + (x-x)h(x) = 0. \tag{25}$$

$$x_{n+1} = g(x_n) = x_n - f(x_n)/h(x_n). \tag{26}$$

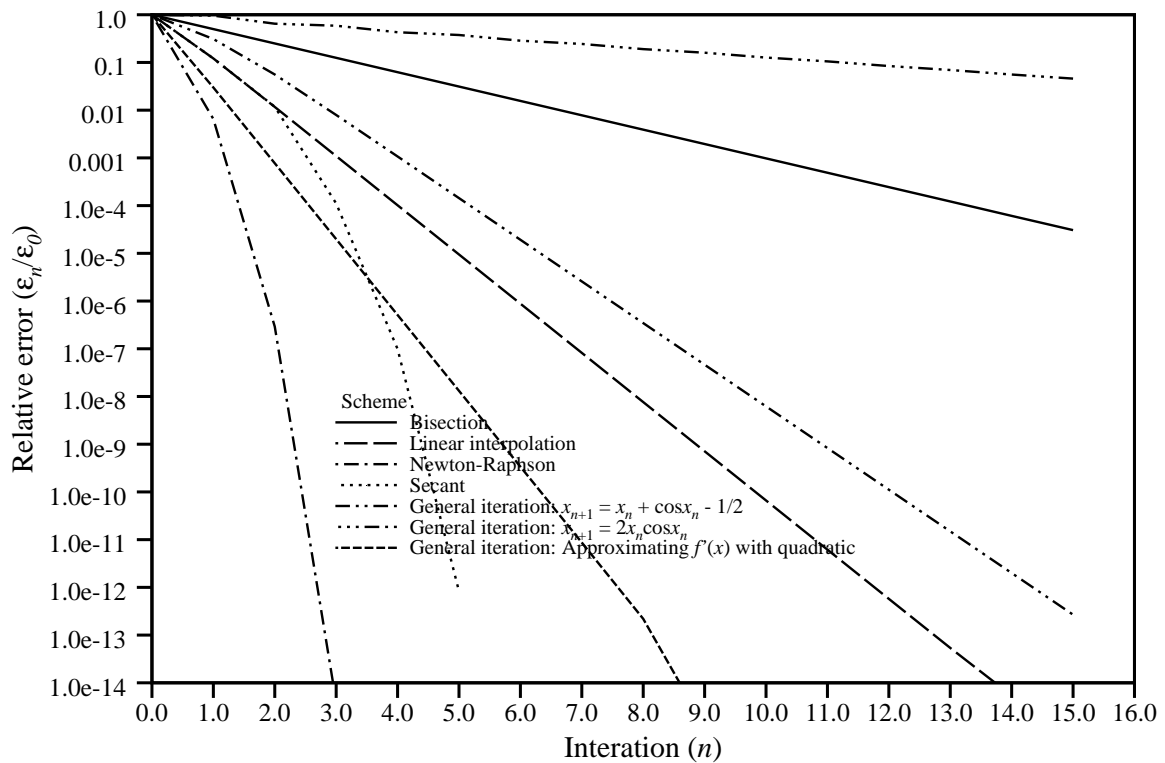
$$h(x) = 4x(x - \pi)/\pi^2 \tag{27}$$

- Initial guess: $x = 0$ (fails with $x = \pi/2$ as $h(x)$ vanishes).
- Expect convergence: $\epsilon_{n+1} \sim g'(x^*) \epsilon_n \sim 0.026 \epsilon_n$.

Iteration	Error	e_{n+1}/e_n
0	0.0235988	0.02985973863078
1	0.000704654	0.02585084310882
2	0.0000182159	0.02572477890195
3	0.000000468600	0.02572151088348
4	0.0000000120531	0.02572134969427
5	0.000000000310022	0.02572107785899
6	0.00000000000797410	0.02570835580191
7	0.000000000000205001	0.02521207213623
8	0.00000000000000516850	
9	Machine accuracy	

3.7.6 Comparison

Convergence for $\cos(x) = 1/2$



3.7.7 Fortran program*

- Program include for completeness and illustration
- ◊ No knowledge of any computer language required for this course

```

PROGRAM Roots
  INTEGER*4 i,j
  REAL*8    x,xa,xb,xc,fa,fb,fc,pi,xStar,f,df
  REAL*8    Error(0:15,0:15)
  f(x)=cos(x)-0.5
  df(x) = -SIN(x)
  pi = 3.141592653
  xStar = ACOS(0.5)
  WRITE(6,*)'# ',xStar,f(xStar)
C=====Bisection
  xa = 0
  fa = f(xa)
  xb = pi/2.0
  fb = f(xb)
  DO i=0,15
    xc = (xa + xb)/2.0
    fc = f(xc)
    IF (fa*fc .LT. 0.0) THEN
      xb = xc
      fb = fc
    
```

```

        ELSE
            xa = xc
            fa = fc
        ENDIF
        Error(0,i) = xc - xStar
    ENDDO
C=====Linear interpolation
    xa = 0
    fa = f(xa)
    xb = pi/2.0
    fb = f(xb)
    DO i=0,15
        xc = xa - (xb-xa)/(fb-fa)*fa
        fc = f(xc)
        IF (fa*fc .LT. 0.0) THEN
            xb = xc
            fb = fc
        ELSE
            xa = xc
            fa = fc
        ENDIF
        Error(1,i) = xc - xStar
    ENDDO
C=====Newton-Raphson
    xa = pi/2.0
    DO i=0,15
        xa = xa - f(xa)/df(xa)
        Error(2,i) = xa - xStar
    ENDDO
C=====Secant
    xa = 0
    fa = f(xa)
    xb = pi/2.0
    fb = f(xb)
    DO i=0,15
        IF (fa .NE. fb) THEN
C          If fa = fb then either method has converged (xa=xb)
C          or will diverge from this point
            xc = xa - (xb-xa)/(fb-fa)*fa
            xa = xb
            fa = fb
            xb = xc
            fb = f(xb)
        ENDIF
        Error(3,i) = xc - xStar
    ENDDO
C=====Direct iteration using  $x + f(x) = x$ 
    xa = 0.0
    DO i=0,15
        xa = xa + f(xa)
        Error(4,i) = xa - xStar
    ENDDO
C=====Direct iteration using  $xf(x)=0$  rearranged for x
C-----Starting point prevents convergence

```

```
    xa = pi/2.0
    DO i=0,15
        xa = 2.0*xa*(f(x)-0.5)
        Error(5,i) = xa - xStar
    ENDDO
C====Direct iteration using xf(x)=0 rearranged for x
    xa = pi/4.0
    DO i=0,15
        xa = 2.0*xa*COS(xa)
        Error(6,i) = xa - xStar
    ENDDO
C====Direct iteration using 4x(x-pi)/pi/pi to approximate f'
    xa = pi/2.0
    DO i=0,15
        xa = xa - f(xa)*pi*pi/(4.0*xa*(xa-pi))
        Error(7,i) = xa - xStar
    ENDDO
C====Output results
    DO i=0,15
        WRITE(6,100)i,(Error(j,i),j=0,7)
    ENDDO
100  FORMAT(1x,i4,8(1x,g12.6))
    END
```

4 Linear equations

- Linear systems fundamental to many mathematical techniques
- Many different ways of determining the solution
 - ◇ Different sensitivity to truncation or round-off error
 - ◇ Exact and approximate solution
 - ◇ Make use of particular system structure

4.1 Gauss elimination

- The simple way
- What you might do, algebraically, by hand

$$\begin{array}{rcl}
 x & + & 2y & + & 3z & = & 6 \\
 2x & + & 2y & + & 3z & = & 7, \\
 x & + & 4y & + & 4z & = & 9
 \end{array}
 \tag{28}$$

- Linear combinations of solutions still solutions
 - ◇ Subtract 2 times the first equation from the second equation
 - ◇ Subtract 1 times the first equation from the third equation

(29)

- ◇ Add second and third equations

(30)

- ◇ Divide third equation by -2
- ◇ Add 3z to second equation and divide by -2

◇ Similarly for first equation

(31)

- Can write system in matrix format

(32)

- For our example

(33)

- Subtract 2 times first *row* from second *row*, and likewise subtract third row from first row

(34)

- Divide second row by -2 to obtain 1 on diagonal

◇ This normally simplifies subsequent operations

(35)

- Subtract 2 times second row from third row

(36)

(37)

- Divide third row by -2 to obtain z

(38)

- Back substitute to get rest of answer

(39)

(40)

- Can do the same for a general system

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n &= r_1 \\
 a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n &= r_2 \\
 a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \dots + a_{3n}x_n &= r_3, \\
 &\vdots \\
 a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n &= r_n
 \end{aligned}
 \tag{41}$$

- ◇ Divide first row by a_{11}
- ◇ Subtract a_{21} times new first row from second row, a_{31} times new first row from third row *etc.*
- ◇ Must remember to do right-hand sides too

$$\begin{bmatrix}
 1 & a_{12}/a_{11} & a_{13}/a_{11} & \dots & a_{1n}/a_{11} \\
 0 & a_{22} - (a_{21}/a_{11})a_{12} & a_{23} - (a_{21}/a_{11})a_{13} & \dots & a_{2n} - (a_{21}/a_{11})a_{1n} \\
 0 & a_{32} - (a_{31}/a_{11})a_{12} & a_{33} - (a_{31}/a_{11})a_{13} & \dots & a_{3n} - (a_{31}/a_{11})a_{1n} \\
 & & \vdots & & \vdots \\
 0 & a_{n2} - (a_{n1}/a_{11})a_{12} & a_{n3} - (a_{n1}/a_{11})a_{13} & \dots & a_{nn} - (a_{n1}/a_{11})a_{1n}
 \end{bmatrix}
 \begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 \vdots \\
 x_n
 \end{pmatrix}
 =
 \begin{pmatrix}
 r_1/a_{11} \\
 r_2 - (a_{21}/a_{11})r_1 \\
 r_3 - (a_{31}/a_{11})r_1 \\
 \vdots \\
 r_n - (a_{n1}/a_{11})r_1
 \end{pmatrix}
 \tag{42}$$

- Repeat until get upper-triangular

$$\begin{bmatrix}
 1 & & & & \\
 & \dots & & & \\
 & & \dots & & \\
 & & & \vdots & \\
 & & & & \dots
 \end{bmatrix}
 \begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 \vdots \\
 x_n
 \end{pmatrix}
 =
 \begin{pmatrix}
 \hat{r}_1 \\
 \hat{r}_2 \\
 \hat{r}_3 \\
 \vdots \\
 \hat{r}_n
 \end{pmatrix}
 \tag{43}$$

- Back substitute for solution

$$\begin{aligned}
 x_n &= \\
 x_{n-1} &= \\
 x_{n-2} &= \\
 &\vdots \\
 x_1 &=
 \end{aligned}
 \tag{44}$$

- Solution process takes $O(n^3)$ operations
- Technique fails if a zero appears on diagonal

$$\begin{bmatrix} 0 & 3 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix},
 \tag{45}$$

- ◇ Solution obviously $x_1 = x_2 = x_3 = 1$
- ◇ The way around this problem is *Pivoting*

4.2 Pivoting

- Difficulties disappear if we interchange rows one and two

(46)

- Or columns one and two

(47)

- If we have

$$\begin{bmatrix} 1 & 4 & 1 & 8 & 3 & 2 & \dots & 5 \\ 0 & 10^{-6} & 1 & 10 & 201 & 13 & & 4 \\ 0 & 9 & 4 & 6 & -8 & 2 & & 18 \\ 0 & 3 & 2 & -3 & 4 & 6003 & & 15 \\ 0 & 15 & 1 & 9 & 33 & -2 & & 1 \\ 0 & -155 & 23 & 4 & 25 & 73 & & 2 \\ & & & \vdots & & & & \vdots \\ 0 & 8 & 56 & 4 & -4 & 4 & \dots & 88 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \hat{r}_1 \\ \hat{r}_2 \\ \hat{r}_3 \\ \hat{r}_4 \\ \hat{r}_5 \\ \hat{r}_6 \\ \vdots \\ \hat{r}_n \end{pmatrix} \quad (48)$$

◇ Problems due to finite precision arithmetic are likely to arise when calculating the next step

4.2.1 Partial pivoting

- Exchange rows to get the largest value on diagonal before division step

$$\begin{array}{c} \left. \begin{array}{l} \rightarrow \\ \rightarrow \end{array} \right\} \begin{bmatrix} 1 & 4 & 1 & 8 & 3 & 2 & \dots & 5 \\ 0 & -155 & 23 & 4 & 25 & 73 & & 2 \\ 0 & 9 & 4 & 6 & -8 & 2 & & 18 \\ 0 & 3 & 2 & -3 & 4 & 6003 & & 15 \\ 0 & 15 & 1 & 9 & 33 & -2 & & 1 \\ 0 & 10^{-6} & 1 & 10 & 201 & 13 & & 4 \\ & & & \vdots & & & & \vdots \\ 0 & 8 & 56 & 4 & -4 & 4 & \dots & 88 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \hat{r}_1 \\ \hat{r}_6 \\ \hat{r}_3 \\ \hat{r}_4 \\ \hat{r}_5 \\ \hat{r}_2 \\ \vdots \\ \hat{r}_n \end{pmatrix} \left. \begin{array}{l} \leftarrow \\ \leftarrow \end{array} \right\} \end{array} \quad (49)$$

- ◇ Variables remain in same order
- ◇ Equations put into a different order

4.2.2 Full pivoting

- Could swap columns instead
 - ◇ Get numerically larger value onto diagonal
 - ◇ Interchange variables, but keep equations in same order

$$\begin{bmatrix} 1 & 3 & 1 & 8 & 3 & 2 & \dots & 5 \\ 0 & 201 & 1 & 10 & 10^{-6} & 13 & & 4 \\ 0 & -8 & 4 & 6 & 9 & 2 & & 18 \\ 0 & 4 & 2 & -3 & 3 & 6003 & & 15 \\ 0 & 33 & 1 & 9 & 15 & -2 & & 1 \\ 0 & 25 & 23 & 4 & -155 & 73 & & 2 \\ & & & \vdots & & & & \vdots \\ 0 & -4 & 56 & 4 & 8 & 4 & \dots & 88 \end{bmatrix} \begin{pmatrix} x_1 \\ x_5 \\ x_3 \\ x_4 \\ x_2 \\ x_6 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \\ \hat{b}_4 \\ \hat{b}_5 \\ \hat{b}_6 \\ \vdots \\ \hat{b}_n \end{pmatrix} \quad (50)$$

- Higher accuracy achieved if interchange both rows and columns

$$\begin{bmatrix} 1 & 4 & 1 & 8 & 3 & 2 & \dots & 5 \\ 0 & 6003 & 2 & -3 & 4 & 3 & & 4 \\ 0 & 2 & 4 & 6 & -8 & 9 & & 18 \\ 0 & 13 & 1 & 10 & 201 & 10^{-6} & & 15 \\ 0 & -2 & 1 & 9 & 33 & 15 & & 1 \\ 0 & 73 & 23 & 4 & 25 & -155 & & 2 \\ & & & \vdots & & & & \vdots \\ 0 & 4 & 56 & 4 & -4 & 8 & \dots & 88 \end{bmatrix} \begin{pmatrix} x_1 \\ x_6 \\ x_3 \\ x_4 \\ x_5 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \hat{r}_1 \\ \hat{r}_4 \\ \hat{r}_3 \\ \hat{r}_2 \\ \hat{r}_5 \\ \hat{r}_6 \\ \vdots \\ \hat{r}_n \end{pmatrix} \quad (51)$$

- May be more efficient to do this only if the diagonal element *too small*
- If can not rearrange to remove zeros from diagonal, then **A** is singular

4.3 LU factorisation

- Decompose **A**

$$\mathbf{A} = \quad (52)$$

- ◊ **L** is a lower triangular matrix
- ◊ **U** is an upper triangular matrix
- ◊ By convention, **L** is scaled to have 1's on the diagonal

$$= \mathbf{b} \quad (53)$$

$$\mathbf{x} = \quad (54)$$

- Solution of system as efficient as Gauss elimination
- Factorisation process takes $O(n^3)$ operations
- Once we have \mathbf{L} and \mathbf{U} then this takes only $O(n^2)$ operations
- Since \mathbf{L} is unity on the diagonal, we do not need to store this, so the factorisation takes no more space than the original matrix
- ◊ Often just overwrite the original matrix to do factorisation
- Algorithm given in notes

```

# Factorisation
FOR i=1 TO n
  FOR p=i TO n
    
$$a_{pi} = a_{pi} - \sum_{k=1}^{i-1} a_{pk} a_{ki}$$

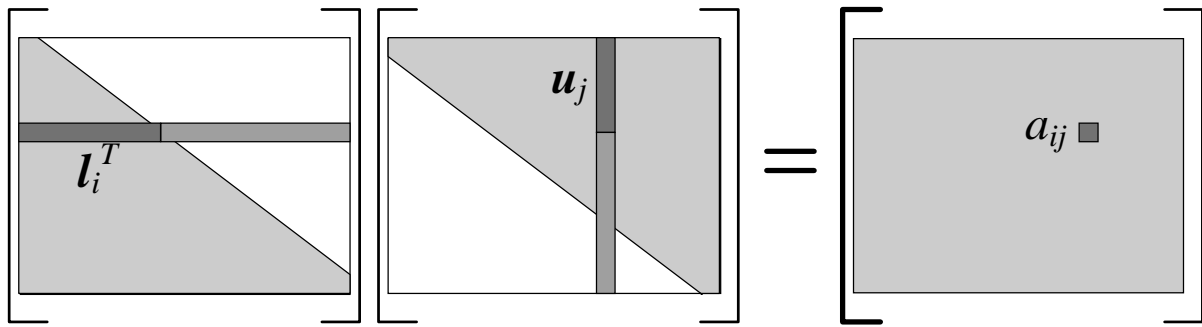
  NEXT p
  FOR q=i+1 TO n
    
$$a_{iq} = \frac{a_{iq} - \sum_{k=1}^{i-1} a_{ik} a_{kq}}{a_{ii}}$$

  NEXT q
NEXT i
# Forward Substitution
FOR i=1 TO n
  FOR q=n+1 TO n+m
    
$$a_{iq} = \frac{a_{iq} - \sum_{k=1}^{i-1} a_{ik} a_{kq}}{a_{ii}}$$

  NEXT q
NEXT i
# Back Substitution
FOR i=n-1 TO 1
  FOR q=n+1 TO n+m
    
$$a_{iq} = a_{iq} - \sum_{k=i+1}^n a_{ik} a_{kq}$$

  NEXT q
NEXT i

```



- If pivoting necessary, then need to store row or column interchanges to make use of factorisation later
- Factorisation gives you direct access to the eigen values

4.4 Banded matrices

- Factorisation readily modified to deal efficiently with banded matrices
- If zero turns up on diagonal during factorisation, need to be careful not to destroy structure
- Making use of banded structure leads to significant savings in time and memory

4.5 Tridiagonal matrices

- A special form of banded matrix with nonzeros only on, just above and just below diagonal

$$\begin{bmatrix}
 b_1 & c_1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 a_2 & b_2 & c_2 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
 0 & a_3 & b_3 & c_3 & 0 & \dots & 0 & 0 & 0 & 0 \\
 0 & 0 & a_4 & b_4 & c_4 & \dots & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & a_5 & b_5 & \dots & 0 & 0 & 0 & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
 0 & 0 & 0 & 0 & 0 & \dots & b_{n-3} & c_{n-3} & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & \dots & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\
 0 & 0 & 0 & 0 & 0 & \dots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\
 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & a_n & b_n
 \end{bmatrix}
 \begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 x_4 \\
 x_5 \\
 \vdots \\
 x_{n-3} \\
 x_{n-2} \\
 x_{n-1} \\
 x_n
 \end{pmatrix}
 =
 \begin{pmatrix}
 r_1 \\
 r_2 \\
 r_3 \\
 r_4 \\
 r_5 \\
 \vdots \\
 r_{n-3} \\
 r_{n-2} \\
 r_{n-1} \\
 r_n
 \end{pmatrix}
 \quad (55)$$

or

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = r_i \quad (56)$$

- Solution by Gauss elimination
- Set $a_1 = 0 = c_n$

```
# Factorisation
FOR i=1 TO n
  b_i = b_i - a_i c_{i-1}
  c_i = c_i / b_i
NEXT i

# Forward Substitution
FOR i=1 TO n
  r_i = (r_i - a_i r_{i-1}) / b_i
NEXT i

# Back Substitution
FOR i=n-1 TO 1
  r_i = r_i - c_i r_{i+1}
NEXT i
```

4.6 Other approaches to solving linear systems

- Iterative methods give approximate solution to linear systems
 - ◇ Based on the direct iteration of §3.6
 - ◇ Often much faster than solving exactly, especially for large sparse systems
- Linear systems often arise when solving pde's numerically
 - ◇ We will look at iterative solutions in this context in §8.1.2

4.7 Over determined systems*

- Over determined systems
 - ◇ Number of equations exceeds number of unknowns
 - ◇ Often produced from experimental work or when analysing observational data
 - ◇ Fitting $a + bx + cx^2 + de^x = y$ to data points x_i, y_i

$$\begin{bmatrix} 1 & x_0 & x_0^2 & e^{x_0} \\ 1 & x_1 & x_1^2 & e^{x_1} \\ 1 & x_2 & x_2^2 & e^{x_2} \\ 1 & x_3 & x_3^2 & e^{x_3} \\ 1 & x_4 & x_4^2 & e^{x_4} \\ 1 & x_5 & x_5^2 & e^{x_5} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & e^{x_{n-1}} \end{bmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \vdots \\ y_{n-1} \end{pmatrix} \tag{57}$$

which is of the form $\mathbf{Ax} = \mathbf{r}$. While the *solution* to $\mathbf{Ax} = \mathbf{r}$ will not exist in an algebraic sense, it can be valuable to determine the solution in an approximate sense. The *error* in this approximate solution is then

- System of the form $\mathbf{Ax} = \mathbf{r}$
- There is no exact solution (unless $m-n$ equations redundant)
- Any approximate solution \mathbf{x} has an error

$$\mathbf{e} = \tag{58}$$

- Construct approximate solution by *optimising* the error
- The *Least Squares* solution is the most common and useful
 - ◊ Minimises the *residual sum of squares* (L_2 norm)

$$rss = \tag{59}$$

- Substituting in for \mathbf{e}

$$rss = \tag{60}$$

- Differentiating and looking for minimum value

$$\frac{\partial r_{ss}}{\partial \mathbf{x}} = \quad (61)$$

- Solution to n by n problem $\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{r}$ gives us the least squares solution
- $\mathbf{A}^T \mathbf{A}$ is often poorly conditioned
 - ◇ Nearly singular
 - ◇ Determinant close to zero
 - * Redundant or unnecessary terms in equations/variables in \mathbf{x}
 - * Small differences between big numbers
 - ◇ Need to be careful how we solve it
 - * Singular Value Decomposition
 - * Householder transformation
- Householder transformation
 - ◇ Suppose \mathbf{Q} is an orthogonal matrix

$$\mathbf{Q}^T \mathbf{Q} = \mathbf{I}, \quad (62)$$

- Choose \mathbf{Q} such that

$$\mathbf{Q} \mathbf{A} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}, \quad (63)$$

- Right-hand side of $\mathbf{Q} \mathbf{A} \mathbf{x} = \mathbf{Q} \mathbf{r}$

$$\mathbf{Q} \mathbf{r} = \begin{pmatrix} b \\ c \end{pmatrix}, \quad (64)$$

- Turning points

$$\begin{aligned}\frac{\partial r_{SS}}{\partial \mathbf{x}} &= 2[\mathbf{A}^T \mathbf{A} \mathbf{x} - \mathbf{A}^T \mathbf{r}] \\ &= 2[\mathbf{A}^T \quad \mathbf{A} \mathbf{x} - \mathbf{A}^T \quad \mathbf{r}]\end{aligned}\tag{65}$$

- For non-trivial solution

$$\mathbf{R} \mathbf{x} = \mathbf{b}.\tag{66}$$

- Details of how to choose \mathbf{Q} are beyond the scope of
- this course

4.8 Under determined systems*

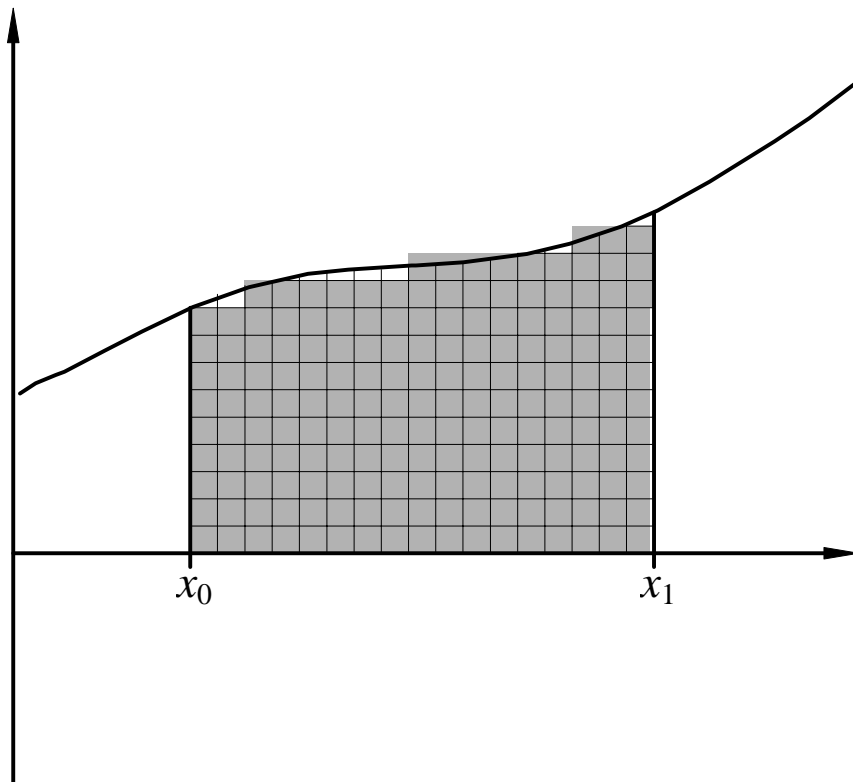
- More variables than equations
- Need optimisation strategy to set additional constraints
- Linear programming
 - ◇ Optimise an *objective* function $z = \mathbf{c}^T \mathbf{x}$
 - ◇ *Operations Research*

5 Numerical integration

- Analytical integration not always possible
- Need to integrate tabulated or sampled data
- As part of broader ranging numerical scheme
- Many will have covered some of these methods at school

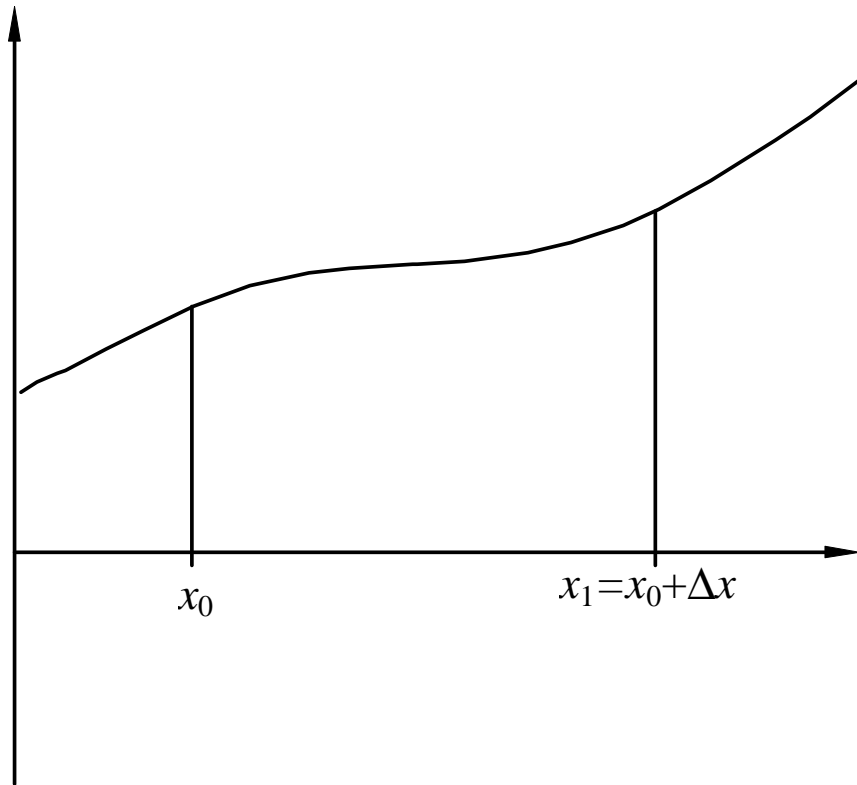
5.1 Manual method

- Counting squares
 - ◇ May be degrading, but still valid
 - ◇ Often the quickest way to get a rough estimate, especially for complex shapes



5.2 Constant rule

- Simplest possible rule
 - ◇ Assume $f(x)$ is constant over interval



$$\begin{aligned}
 \int_{x_0}^{x_0+\Delta x} f(x)dx &= \int_{x_0}^{x_0+\Delta x} \\
 &= \\
 &=
 \end{aligned}
 \tag{67}$$

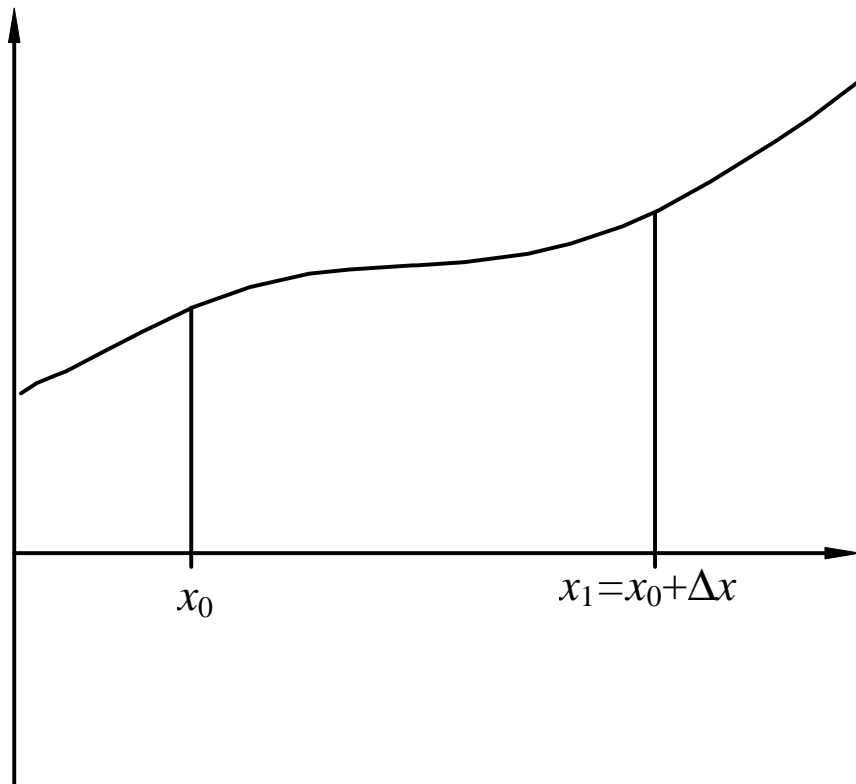
- Makes a difference if lower or upper bound is used
- Rule not used in practice, but helps with solution of ode's.

5.3 Trapezium rule

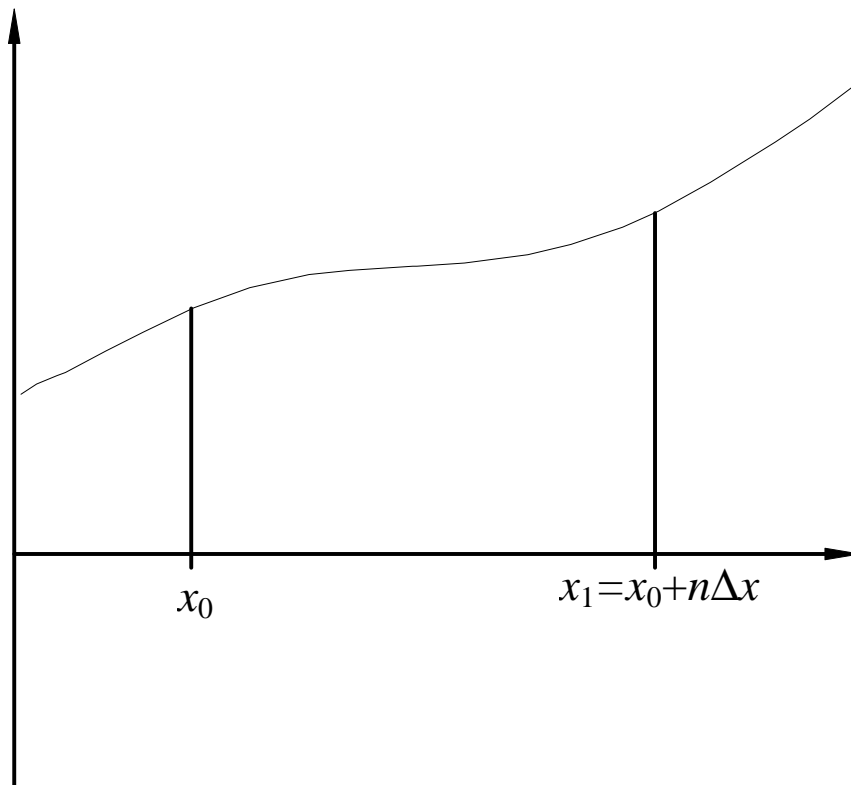
- Taylor Series expansion integrated

$$\begin{aligned}
 \int_{x_0}^{x_0+\Delta x} f(x)dx &= \int_{x_0}^{x_0+\Delta x} f(x_0) + f'(x_0)\Delta x + \frac{1}{2}f''(x_0)\Delta x^2 + \dots dx \\
 &= f(x_0)\Delta x + \frac{1}{2}f'(x_0)\Delta x^2 + \frac{1}{6}f''(x_0)\Delta x^3 + \dots . \\
 &=
 \end{aligned}
 \tag{68}$$

- Trapezium Rule has obvious geometric interpretation



- Error $O(\Delta x^3)$
 - ◇ Halving Δx , decreases error by factor of 8
 - * Size of domain halved
 - * Integrate twice to cover same region
 - * Total error $2 \times O((\Delta x/2)^2) = \frac{1}{4} O(\Delta x^3)$



- Compound Trapezium Rule

$$\begin{aligned}
 \int_{x_0}^{x_1} f(x) dx &= \sum_{i=0}^{n-1} \int_{x_0+i\Delta x}^{x_0+(i+1)\Delta x} f(x) dx \\
 &\approx \sum_{i=0}^{n-1} \dots \\
 &= \dots
 \end{aligned}
 \tag{69}$$

- If n steps, then total error in interval $n \Delta x$ is $O(\Delta x^2)$
 - ◇ Could use smaller Δx in regions where curvature greater

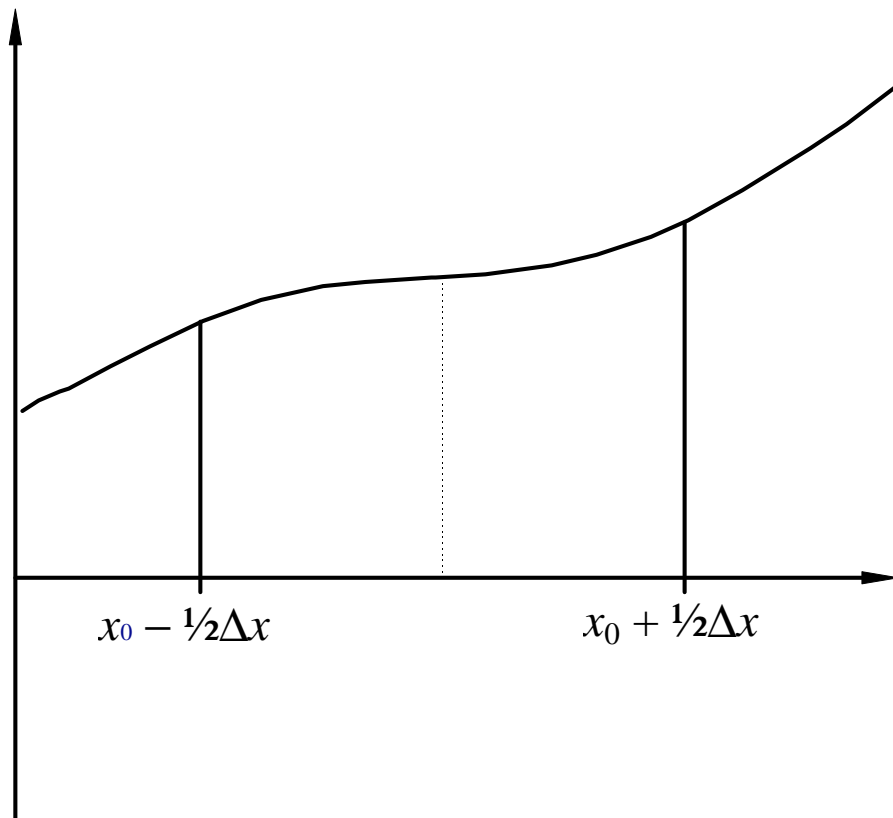
5.4 Mid-point rule

- Variant on the Trapezium Rule
- Integrate Taylor Series from $x_0 - \frac{1}{2}\Delta x$ to $x_0 + \frac{1}{2}\Delta x$

$$\int_{x_0 - \frac{1}{2}\Delta x}^{x_0 + \frac{1}{2}\Delta x} f(x) dx = \int_{x_0 - \frac{1}{2}\Delta x}^{x_0 + \frac{1}{2}\Delta x} f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots dx \quad (70)$$

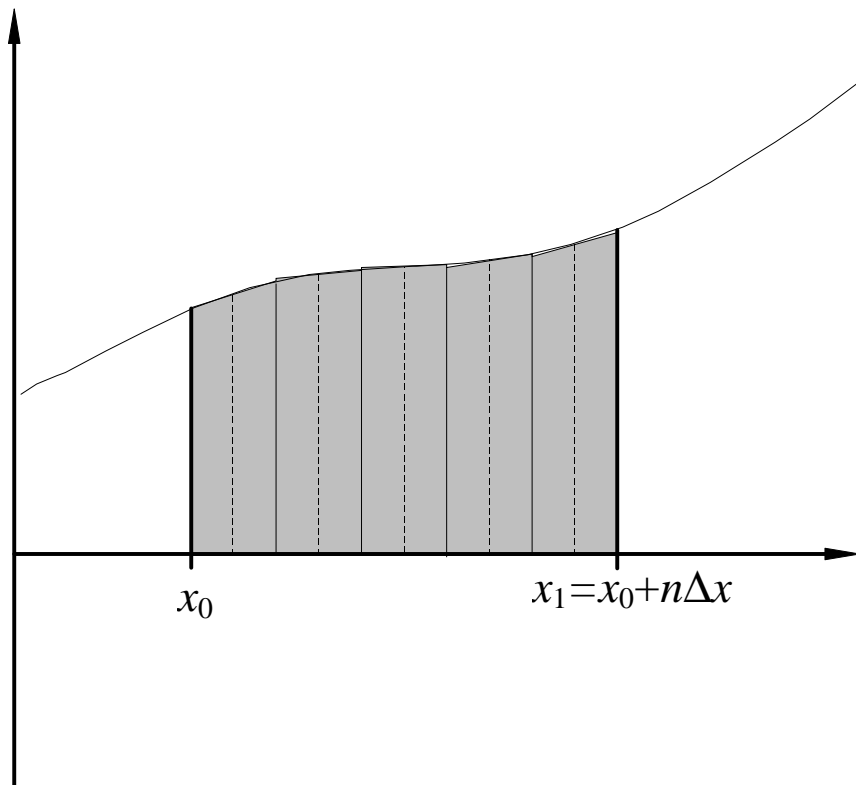
$$=$$

- Evaluating $f(x)$ once at mid-point gives error $O(\Delta x^3)$
 - ◊ Same order as Trapezium Rule achieved with two values of $f(x)$
 - ◊ Coefficient $f''/24$ is smaller than the $f''/12$ of the Trapezium Rule

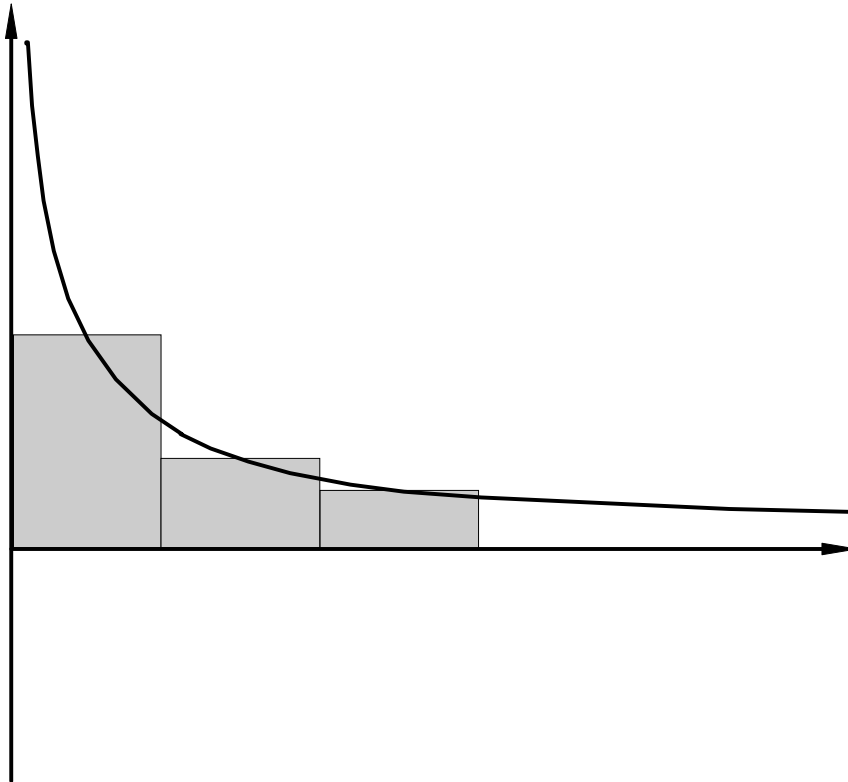


- Construct Compound Mid-point Rule in same way as Compound Trapezium Rule

$$\int_{x_0}^{x_1} f(x) dx \approx \sum_{i=0}^{n-1} \quad (71)$$

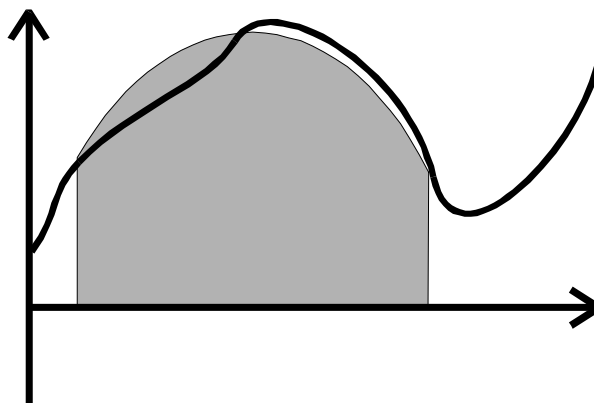


- Only one fewer evaluations than Compound Trapezium Rule
 - ◇ Phase difference
- Integrable singularities
 - ◇ May appear to be able to integrate non-integrable singularities
 - * No convergence as Δx decreases



5.5 Simpson's rule

- Alternative to decreasing Δx : Increase accuracy of function used to approximate $f(x)$



- Integrate Taylor Series over interval $2\Delta x$

$$\begin{aligned}
 \int_{x_0}^{x_0+2\Delta x} f(x) dx &= & (72) \\
 &= \frac{\Delta x}{3} [\\
 & f(x_0) \\
 & + 4(f(x_0) + f'(x_0)\Delta x + \frac{1}{2}f''(x_0)\Delta x^2 + \frac{1}{6}f'''(x_0)\Delta x^3 + \frac{1}{24}f^{iv}(x_0)\Delta x^4 + \dots) \\
 & + (f(x_0) + 2f'(x_0)\Delta x + 2f''(x_0)\Delta x^2 + \frac{4}{3}f'''(x_0)\Delta x^3 + \frac{2}{3}f^{iv}(x_0)\Delta x^4 + \dots) \\
 & - \frac{17}{30}f^{iv}(x_0)\Delta x^4 \dots] \\
 & =
 \end{aligned}$$

- Error $O(\Delta x^5)$
 - ◊ Two orders more accurate than Trapezium Rule
- Compound Simpson's Rule

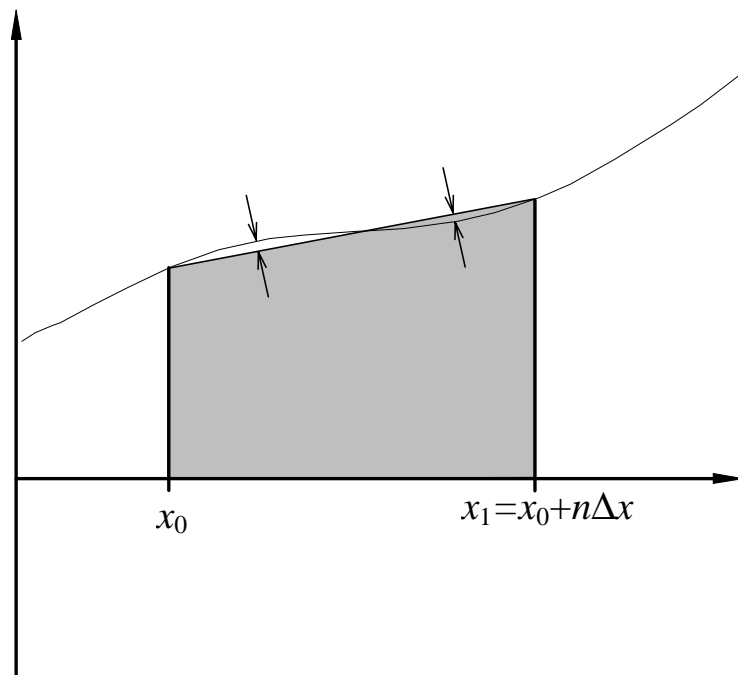
$$\begin{aligned}
 \int_{x_0}^{x_1} f(x) dx &\approx \frac{\Delta x}{3} \sum_{i=0}^{m-1} f(x_0 + 2i\Delta x) + 4f(x_0 + (2i+1)\Delta x) + f(x_0 + (2i+2)\Delta x) \\
 &= & (73)
 \end{aligned}$$

- Error $O(n\Delta x^5) = O(\Delta x^4)$

5.6 Quadratic triangulation*

- Simple manual method
 - ◊ Approximate by parabolas
 - ◊ More accurate than using triangles

$$area = \frac{2}{3} \times chord \times maxDeviation, \tag{74}$$



5.7 Romberg integration

- Way of combining integrals to improve accuracy
- Compound Trapezium Rule
 - ◊ Approximation: $T(\Delta x) = I + c\Delta x^2 + O(\Delta x^4)$
 - * Second approximation: $T(\Delta x/2) = I + \frac{1}{4}c\Delta x^2 + O(\Delta x^4)$
- Take linear combination

$$\begin{aligned} T^{(1)}(\Delta x/2) &= \alpha T(\Delta x/2) + (1-\alpha)T(\Delta x) \\ &= \alpha[I + c\Delta x^2/4 + O(\Delta x^4)] + (1-\alpha)[I + c\Delta x^2 + O(\Delta x^4)]. \end{aligned} \tag{75}$$

- Eliminate $c\Delta x^2$ term by choosing $\alpha = 4/3$:

$$T^{(1)}(\Delta x/2) = [4T(\Delta x/2) - T(\Delta x)]/3. \tag{76}$$

- Substitute in the Compound Trapezium Rule
 - ◊ Gives 1 4 2 4 2 4 ... 2 4 1 pattern of Simpson’s Rule
- Can repeat this trick to eliminate higher order errors

$$T^{(m)}(\Delta x/2) = [2^{2m}T^{(m-1)}(\Delta x/2) - T^{(m-1)}(\Delta x)]/(2^{2m}-1). \tag{77}$$

5.8 Gauss quadrature

- Careful selection of points where function evaluated can improve accuracy
 - ◊ The mid-point rule is the simplest example
- Most common example is Gauss Quadrature
 - ◊ Exact integration of cubics with only two function evaluations
 - * Simpson’s Rule is exact for cubics, but requires three $f(x)$

- Gauss Quadrature:

$$\int_{x_0}^{x_1=x_0+\Delta x} f(x)dx \approx \frac{\Delta x}{2} \left[f\left(x_0 + \left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)\Delta x\right) + f\left(x_0 + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)\Delta x\right) \right] + O(\Delta x^4). \tag{78}$$

- With M evaluations, can get exact integration for all polynomials of degree $2M - 1$ or less.

$$\begin{aligned} \int_{x_0}^{x_1=x_0+\Delta x} f(x)dx &= \Delta x f(x_0) + \frac{\Delta x^2}{2} f'(x_0) + \frac{\Delta x^3}{6} f''(x_0) + \frac{\Delta x^4}{24} f'''(x_0) + O(\Delta x^5) \\ &= \frac{\Delta x}{2} \left[f(x_0) + \alpha \Delta x f'(x_0) + \frac{(\alpha \Delta x)^2}{2} f''(x_0) + \frac{(\alpha \Delta x)^3}{6} f'''(x_0) + \dots \right. \\ &\quad \left. + f(x_0) + \beta \Delta x f'(x_0) + \frac{(\beta \Delta x)^2}{2} f''(x_0) + \frac{(\beta \Delta x)^3}{6} f'''(x_0) + \dots \right] \tag{79} \\ &= \Delta x f(x_0) + (\alpha + \beta) \frac{\Delta x^2}{2} f'(x_0) + (\alpha^2 + \beta^2) \frac{\Delta x^3}{4} + (\alpha^3 + \beta^3) \frac{\Delta x^4}{12} + \dots \end{aligned}$$

- Equating terms

$$(80)$$

- Solution gives $\alpha, \beta = \frac{1}{2} \pm \sqrt{3}/6$
- If use three $f(x)$ for interval $x_0 - \Delta x$ to $x_0 + \Delta x$ then get error $O(\Delta x^8)$
 - ◊ Symmetry suggests evaluate at x_0 and $x_0 \pm \alpha$
 - ◊ Weightings $2\Delta x A$ and $2\Delta x B$

- Taylor Series of itegral

$$\int_{x_0-\Delta x}^{x_0+\Delta x} f(x)dx = 2\Delta x \left[f(x_0) + \frac{\Delta x^2}{6} f''(x_0) + \frac{\Delta x^4}{120} f^{iv}(x_0) + \frac{\Delta x^6}{5040} f^{vi}(x_0) + O(\Delta x^8) \right], (81)$$

- Taylor Series of points

$$\begin{aligned} \int_{x_0-\Delta x}^{x_0+\Delta x} f(x)dx &= 2\Delta x [Af(x_0) + Bf(x_0 - \alpha) + Bf(x_0 + \alpha)] \\ &= 2\Delta x [Af \\ &+ B(f - \alpha f' + \frac{1}{2}\alpha^2 f'' - \frac{1}{6}\alpha^3 f''' + \frac{1}{24}\alpha^4 f^{iv} - \frac{1}{120}\alpha^5 f^v + \frac{1}{720}\alpha^6 f^{vi} - \frac{1}{5040}\alpha^7 f^{vii} + O(\alpha^8)) \\ &+ B(f + \alpha f' + \frac{1}{2}\alpha^2 f'' + \frac{1}{6}\alpha^3 f''' + \frac{1}{24}\alpha^4 f^{iv} + \frac{1}{120}\alpha^5 f^v + \frac{1}{720}\alpha^6 f^{vi} + \frac{1}{5040}\alpha^7 f^{vii} + O(\alpha^8))] \\ &= 2\Delta x [(A + 2B)f + B\alpha^2 f'' + \frac{1}{12} B\alpha^4 f^{iv} + \frac{1}{360} \alpha^6 f^{vi} + O(\alpha^8)]. \end{aligned} \quad (82)$$

- Comparing terms

$$A + 2B = 1$$

$$B\alpha^2 = \Delta x^2/6$$

$$B\alpha^4/12 = \Delta x^4/120 \quad (83)$$

- Solving

$$(B\alpha^4/12)/B\alpha^2 = (\Delta x^4/120) / (\Delta x^2/6)$$

$$\Rightarrow \alpha^2 = (6/10)\Delta x^2$$

$$\Rightarrow \alpha = (3/5)^{1/2} \Delta x$$

$$B = 5/18$$

$$A = 4/9, \tag{84}$$

$$\int_{x_0-\Delta x}^{x_0+\Delta x} f(x) dx = \frac{1}{9} \Delta x \left[8f(x_0) + 5f\left(x_0 - \left(\frac{3}{5}\right)^{1/2} \Delta x\right) + 5f\left(x_0 + \left(\frac{3}{5}\right)^{1/2} \Delta x\right) \right]. \tag{85}$$

5.9 Example of numerical integration

$$\int_0^{\pi} \sin x \, dx = 2, \tag{86}$$

No. intervals	No. $f(x)$	Trapezium Rule	Error Ratio: e_{2n}/e_n
1	2	-2.00000000	0.2146
2	3	-0.429203673	0.24203
4	5	-0.103881102	0.24806
8	9	-0.0257683980	0.24952
16	17	-0.00642965622	0.24988
32	33	-0.00160663902	0.24997
64	65	-0.000401611359	0.24999
128	129	-0.000100399815	0.25
256	257	-0.0000250997649	0.25
512	513	-0.00000627492942	0.25
1024	1025	-0.00000156873161	0.25
2048	2049	-0.000000392182860	0.25
4096	4097	-0.0000000980457133	0.25
8192	8193	-0.0000000245114248	0.25
16384	16385	-0.00000000612785222	0.25
32768	32769	-0.00000000153194190	0.24999
65536	65537	-0.000000000382977427	0.24994
131072	131073	-0.0000000000957223189	0.25014
262144	262145	-0.0000000000239435138	0.24898
524288	524289	-0.00000000000596145355	

Table 2: Error in Trapezium Rule. Note error ratio $\rightarrow 2^{-2} (\Delta x^2)$

No. intervals	No. $f(x)$	Midpoint Rule	Error Ratio: e_{2n}/e_n
1	1	1.14189790	0.19392
2	2	0.221441469	0.23638
4	4	0.0523443059	0.24662
8	8	0.0129090855	0.24916
16	16	0.00321637816	0.24979
32	32	0.000803416309	0.24995
64	64	0.000200811728	0.24999

128	128	0.0000502002859	0.25
256	256	0.0000125499060	0.25
512	512	0.00000313746618	0.25
1024	1024	0.000000784365898	0.25
2048	2048	0.000000196091438	0.25
4096	4096	0.0000000490228564	0.25
8192	8192	0.0000000122557182	0.25
16384	16384	0.00000000306393221	0.25
32768	32768	0.000000000765979280	0.25
65536	65536	0.000000000191497040	0.24979
131072	131072	0.0000000000478341810	0.25081
262144	262144	0.0000000000119970700	0.25286
524288	524288	0.00000000000303357339	

Table 3: Error in Mid-point Rule. Note error ratio $\rightarrow 2^{-2} (\Delta x^2)$

No. intervals	No. $f(x)$	Simpson's Rule	Error Ratio: e_{2n}/e_n
1	3	0.0943951023	0.0483
2	5	0.00455975498	0.05903
4	9	0.000269169948	0.06164
8	17	0.0000165910479	0.06228
16	33	0.00000103336941	0.06245
32	65	0.0000000645300022	0.06249
64	129	0.00000000403225719	0.0625
128	257	0.000000000252001974	0.0625
256	513	0.0000000000157500679	0.0624
512	1025	0.000000000000982769421	

Table 4: Error in Simpson's Rule. Note error ratio $\rightarrow 2^{-4} (\Delta x^4)$

No. intervals	No. $f(x)$	Gauss Quadrature	Error Ratio: e_{2n}/e_n
1	2	-0.0641804253	0.0476
2	4	-0.00305477319	0.05881
4	8	-0.000179666460	0.06158
8	16	-0.0000110640837	0.06227
16	32	-0.000000688965642	0.06244
32	64	-0.0000000430208237	0.06249
64	128	-0.00000000268818500	0.0625
128	256	-0.000000000168002278	0.06249
256	512	-0.0000000000104984909	0.06248
512	1024	-0.000000000000655919762	

Table 5: Error in Gauss Quadrature. Note error ratio $\rightarrow 2^{-4} (\Delta x^4)$

No. intervals	No. $f(x)$	Gauss Quadrature - three point	Error Ratio: e_{2n}/e_n
1	3	0.001388913607743625	0.01169
2	6	0.00001624311099668319	0.01464
4	12	0.0000002378219958742989	0.01538
8	24	0.000000003657474767493341	0.01556
16	48	0.00000000005692291082937118	0.0156
32	96	0.0000000000008881784197001252	0.016
64	192	0.00000000000001421085471520200	-0.01563
128	384	-0.0000000000000002220446049250313	1
256	768	-0.0000000000000002220446049250313	

Table 6: Error in Three point Gauss Quadrature. Note error ratio $\rightarrow 2^{-6} (\Delta x^6)$

No. Intervals	Trapezium Rule	Midpoint Rule	Simpson's Rule	Gauss Quadrature	3 Pnt Gauss Quadrature
1	-2.0000E+00	1.1418E+00	9.4395E-02	-6.4180E-02	1.3889E-03
2	-4.2920E-01	2.2144E-01	4.5597E-03	-3.0547E-03	1.6243E-05
4	-1.0388E-01	5.2344E-02	2.6916E-04	-1.7966E-04	2.3782E-07
8	-2.5768E-02	1.2909E-02	1.6591E-05	-1.1064E-05	3.6574E-09
16	-6.4296E-03	3.2163E-03	1.0333E-06	-6.8896E-07	5.6922E-11
32	-1.6066E-03	8.0341E-04	6.4530E-08	-4.3020E-08	8.8817E-13
64	-4.0161E-04	2.0081E-04	4.0322E-09	-2.6881E-09	1.4210E-14
128	-1.0039E-04	5.0200E-05	2.5200E-10	-1.6800E-10	-2.2204E-16
256	-2.5099E-05	1.2549E-05	1.5750E-11	-1.0498E-11	-2.2204E-16
512	-6.2749E-06	3.1374E-06	9.8276E-13	-6.5591E-13	
1024	-1.5687E-06	7.8436E-07			
2048	-3.9218E-07	1.9609E-07			
4096	-9.8045E-08	4.9022E-08			
8192	-2.4511E-08	1.2255E-08			
16384	-6.1278E-09	3.0639E-09			
32768	-1.5319E-09	7.6597E-10			
65536	-3.8297E-10	1.9149E-10			
131072	-9.5722E-11	4.7834E-11			
262144	-2.3943E-11	1.1997E-11			
524288	-5.9614E-12	3.0335E-12			
1048576					

Table 7: Error in numerical integration of (86) as a function of the number of subintervals.

5.9.1 Program for numerical integration*

- Program include for completeness and illustration
 - ◇ No knowledge of any computer language required for this course
- Program written for clarity and simplicity rather than speed
 - ◇ Compound rules written as sum of simple rules

```

PROGRAM Integrat
REAL*8    x0,x1,Value,Exact,pi
INTEGER*4  i,j,nx
C=====Functions
REAL*8    TrapeziumRule
REAL*8    MidpointRule
REAL*8    SimpsonsRule
REAL*8    GaussQuad
C=====Constants
pi = 2.0*ASIN(1.0D0)
Exact = 2.0
C=====Limits
x0 = 0.0
x1 = pi
C=====
C= Trapezium rule =
C=====
WRITE(6,*)
WRITE(6,*) 'Trapezium rule'

nx = 1
DO i=1,20
  Value = TrapeziumRule(x0,x1,nx)
  WRITE(6,*)nx,Value,Value - Exact
  nx = 2*nx

```

```

C=====ENDDO
C= Midpoint rule =
C=====
WRITE(6,*)
WRITE(6,*)'Midpoint rule'
nx = 1
DO i=1,20
  Value = MidpointRule(x0,x1,nx)
  WRITE(6,*)nx,Value,Value - Exact
  nx = 2*nx
ENDDO
C=====
C= Simpson's rule =
C=====
WRITE(6,*)
WRITE(6,*)'Simpson''s rule'
WRITE(6,*)
nx = 2
DO i=1,10
  Value = SimpsonsRule(x0,x1,nx)
  WRITE(6,*)nx,Value,Value - Exact
  nx = 2*nx
ENDDO
C=====
C= Gauss Quadrature =
C=====
WRITE(6,*)
WRITE(6,*)'Gauss quadrature'
nx = 1
DO i=1,10
  Value = GaussQuad(x0,x1,nx)
  WRITE(6,*)nx,Value,Value - Exact
  nx = 2*nx
ENDDO
END

FUNCTION f(x)
C=====parameters
REAL*8 x,f
f = SIN(x)
RETURN
END

REAL*8 FUNCTION TrapeziumRule(x0,x1,nx)
C=====parameters
INTEGER*4 nx
REAL*8 x0,x1
C=====functions
REAL*8 f
C=====local variables
INTEGER*4 i
REAL*8 dx,xa,xb,fa,fb,Sum
dx = (x1 - x0)/DFLOAT(nx)
Sum = 0.0
DO i=0,nx-1
  xa = x0 + DFLOAT(i)*dx
  xb = x0 + DFLOAT(i+1)*dx
  fa = f(xa)
  fb = f(xb)

  Sum = Sum + fa + fb
ENDDO
Sum = Sum * dx / 2.0
TrapeziumRule = Sum
RETURN

```

```

END
REAL*8 FUNCTION MidpointRule(x0,x1,nx)
C=====parameters
INTEGER*4 nx
REAL*8 x0,x1
C=====functions
REAL*8 f
C=====local variables
INTEGER*4 i
REAL*8 dx,xa,fa,Sum
dx = (x1 - x0)/DFLOAT(nx)
Sum = 0.0
DO i=0,nx-1
  xa = x0 + (DFLOAT(i)+0.5)*dx
  fa = f(xa)
  Sum = Sum + fa
ENDDO
Sum = Sum * dx
MidpointRule = Sum
RETURN
END

REAL*8 FUNCTION SimpsonsRule(x0,x1,nx)
C=====parameters
INTEGER*4 nx
REAL*8 x0,x1
C=====functions
REAL*8 f
C=====local variables
INTEGER*4 i
REAL*8 dx,xa,xb,xc,fa,fb,fc,Sum
dx = (x1 - x0)/DFLOAT(nx)
Sum = 0.0
DO i=0,nx-1,2
  xa = x0 + DFLOAT(i)*dx
  xb = x0 + DFLOAT(i+1)*dx
  xc = x0 + DFLOAT(i+2)*dx
  fa = f(xa)
  fb = f(xb)
  fc = f(xc)
  Sum = Sum + fa + 4.0*fb + fc
ENDDO
Sum = Sum * dx / 3.0
SimpsonsRule = Sum
RETURN
END

REAL*8 FUNCTION GaussQuad(x0,x1,nx)
C=====parameters
INTEGER*4 nx
REAL*8 x0,x1
C=====functions
REAL*8 f
C=====local variables
INTEGER*4 i
REAL*8 dx,xa,xb,fa,fb,Sum,dx1,dxr
dx = (x1 - x0)/DFLOAT(nx)
dx1 = dx*(0.5D0 - SQRT(3.0D0)/6.0D0)
dxr = dx*(0.5D0 + SQRT(3.0D0)/6.0D0)

Sum = 0.0
DO i=0,nx-1
  xa = x0 + DFLOAT(i)*dx + dx1
  xb = x0 + DFLOAT(i)*dx + dxr
  fa = f(xa)

```

```
      fb = f(xb)
      Sum = Sum + fa + fb
ENDDO
Sum = Sum * dx / 2.0
GaussQuad = Sum
RETURN
END
```

6 First order ordinary differential equations

- Introduce basic techniques
- ODE techniques help with PDEs

6.1 Taylor series

- Key idea
 - ◇ Combine function values at different points or times to approximate derivatives
 - ◇ Finite difference approximations

6.2 Finite difference

- Consider

$$\frac{dy}{dt} = \tag{87}$$

- ◇ Boundary/initial condition $y = c$ at $t = t_0$
- ◇ Discretise time t to $t_0, t_0+\Delta t, t_0+2\Delta t, t_0+3\Delta t, \dots$
 - * Exact solution at $t = t_n \equiv t_0 + n\Delta t$ is $y_n \equiv y(t=t_n)$
 - * Approximate solution at t_n is Y_n

- Look to solve

$$\tag{88}$$

- ◇ Y'_n is approximation to dy/dt

- Taylor Series

...

$$Y_{n-2} =$$

$$Y_{n-1} =$$

$$\begin{aligned}
 Y_n &= \\
 Y_{n+1} &= \\
 Y_{n+2} &= \\
 &\dots
 \end{aligned}
 \tag{89}$$

- Take linear combinations

$$Y'_n \approx \sum_{i=a}^b \alpha_i Y_{n+i} . \tag{90}$$

◇ To eliminate Y_n require $\tag{91}$

- Select α_i to approximate Y' to varying degrees of accuracy
- If using Y_n and Y_{n-1} then

◇ $\alpha_1 = \alpha, \alpha_2 = -\alpha$

$$\begin{aligned}
 Y'_n &\approx \alpha Y_n - \alpha Y_{n-1} \\
 &=
 \end{aligned}
 \tag{92}$$

◇ $\alpha = 1/\Delta t$

$$\frac{Y_n - Y_{n-1}}{\Delta t} = Y'_n + \frac{1}{2} \Delta t Y''_n + O(\Delta t^2) . \tag{93}$$

6.3 Truncation error

- *Global truncation error*
- ◇ The absolute error in the n^{th} step

$$E_n = \tag{94}$$

* Measure of how accurate the solution is

- *Local truncation error*

- ◇ The error introduced in the n^{th} step

$$e_n = \quad (95)$$

- * y_n^* is exact solution given that $y_{n-1}^* = Y_{n-1}$

- Global truncation error E_n is not

$$\sum_{i=1}^n e_n, \quad (96)$$

- ◇ Depends on *stability* of the method

- * We shall look at this in §6.7

- ◇ Aim for $E_n = O(e_n)$, not $O(n e_n)$

6.4 Euler method

- Simplest scheme

$$(97)$$

- In ode $y' = f(t, y)$

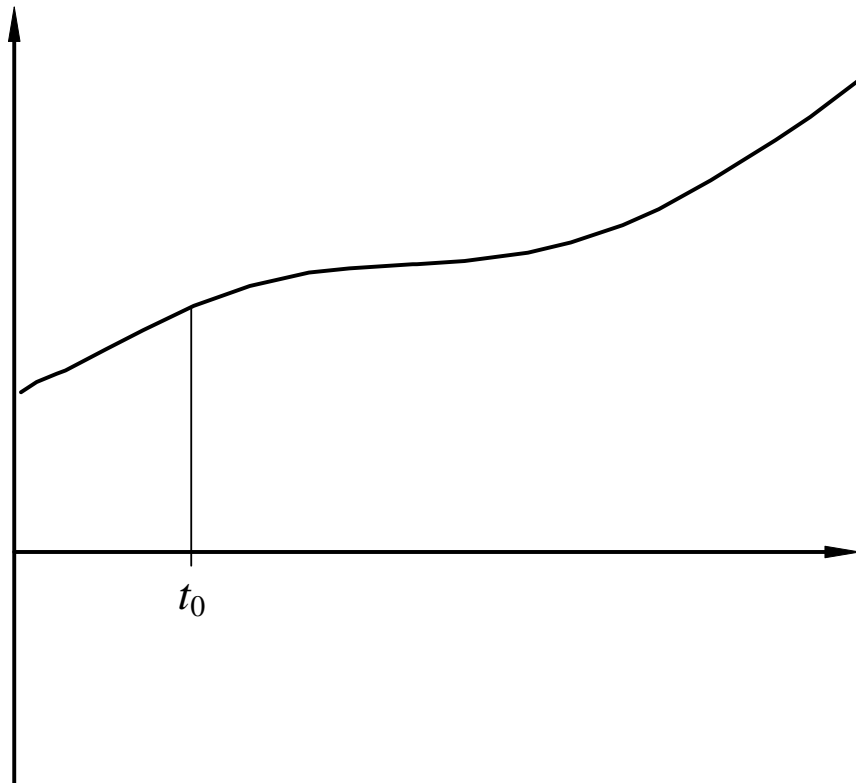
$$(98)$$

- From ic/bc $Y_0 = c$, obtain

- ◇ $Y_1 = Y_0 + \Delta t f(t_0, Y_0)$

- ◇ $Y_2 = Y_1 + \Delta t f(t_1, Y_1)$

- ◇ ...



- *Explicit method*
 - ◊ Explicit expression for Y_{n+1} in terms of *known* values at t_n
- Error: compare (98) with expansion for Y_{n+1}
 - ◊ $e_n = O(\Delta t^2)$
- Comparison with *Constant Rule* integration

6.5 Implicit methods

- Explicit methods update formula $Y_{n+1} = g(Y_n, t_n, \Delta t)$
 - ◊ Cheap per step
- Implicit methods update formula $Y_{n+1} = h(Y_n, Y_{n+1}, t_n, \Delta t)$
 - ◊ Normally more expensive per step
 - ◊ More stable (§6.7), so can take larger (and hence fewer) steps

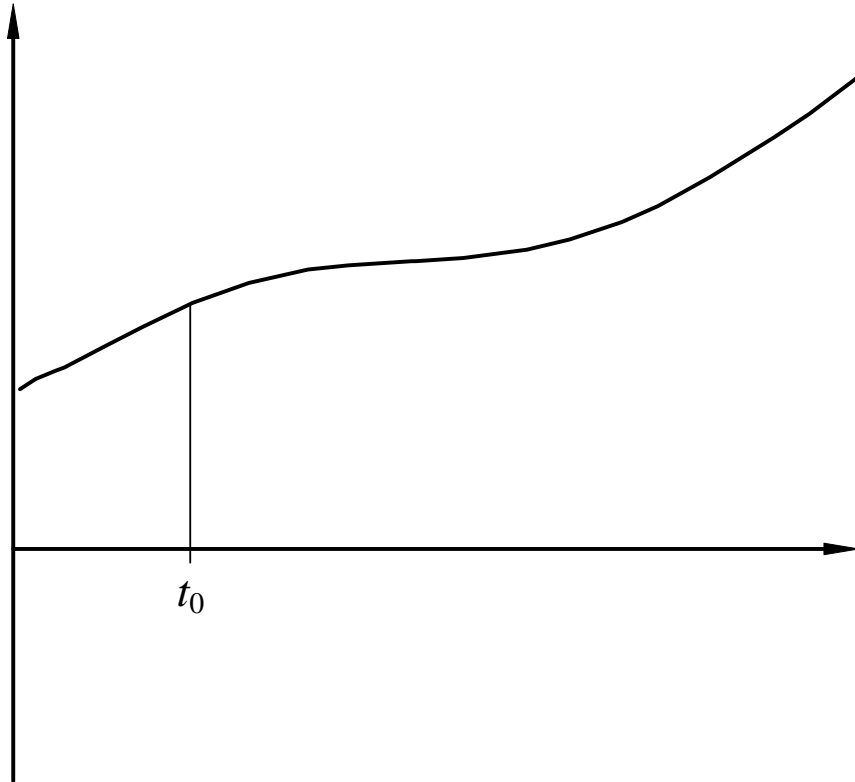
6.5.1 Backward Euler

- Simplest implicit method
- Derivative approximated backwards in time
 - ◊ *Backward difference*

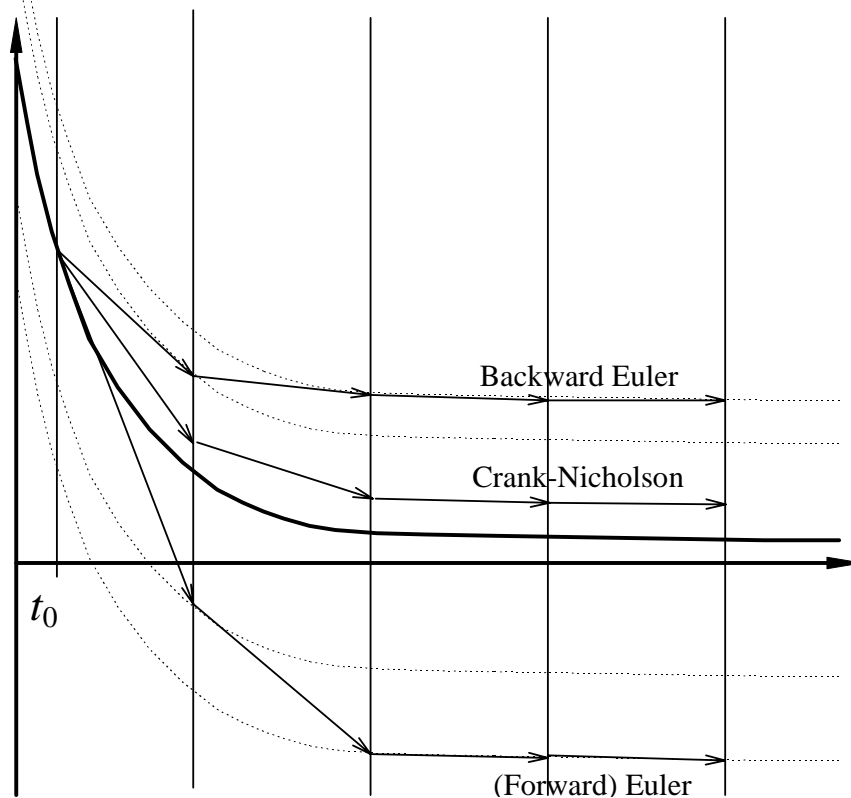
$$Y'_n \approx \quad (99)$$

- Evolution equation

$$Y_{n+1} = \quad (100)$$



- Truncation error same as for Euler method
 - ◇ Errors $O(\Delta t^2)$
 - ◇ More stable and hence more accurate, especially for *stiff* problems



6.5.2 Richardson extrapolation

- Similar to Romberg integration (§5.7)
- Error in estimates depends on step size
 - ◊ $Y(t, \Delta t) = y(t) + c\Delta t^2$
 - * $Y(t, 1/2\Delta t) = y(t) + 1/4 c\Delta t^2$
- Take linear combination to remove leading order error term

$$Y^{(1)}(t, \Delta t/2) = [4Y(t, \Delta t/2) - Y(t, \Delta t)] / 3. \tag{101}$$

- Not very often used in practice
 - ◊ But does allow easy monitoring of the truncation error
 - ◊ Can help reduce numerical dispersion

6.5.3 Crank-Nicholson

- Use *central difference*
 - ◊ Eliminates the $\Delta t^2 Y''$ term in the error
 - * Second order approximation for Y'

$$\tag{102}$$

- Substitution into ode

$$(103)$$

- Need to have $f(t_{n+1/2}, Y_{n+1/2})$

◇ Only have values at t_n

* Once step complete, t_{n+1}

◇ Use linear interpolation

$$Y_{n+1} - Y_n = \tag{104}$$

- Implicit
- Proof that second-order more complex than Euler

$$y_{n+1} = y_n + \frac{dy}{dt} \Delta t + \frac{1}{2} \frac{d^2y}{dt^2} \Delta t^2 + O(\Delta t^3) \tag{105a}$$

=

$$f(t_{n+1}, y_{n+1}) = \tag{105b}$$

- Substitution into (104)

◇ Left-hand side

$$y_{n+1} - y_n = f_n \Delta t + \frac{1}{2} \left(\frac{\partial f}{\partial t} + f \frac{\partial f}{\partial y} \right) \Delta t^2 + O(\Delta t^3) \tag{106a}$$

◇ Right-hand side

$$\begin{aligned} \frac{1}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1}))\Delta t &= \frac{1}{2}\left(f_n + f_n + \left(\frac{\partial f}{\partial t} + f \frac{\partial f}{\partial y}\right)\Delta t + O(\Delta t^2)\right)\Delta t \\ &= f_n\Delta t + \frac{1}{2}\left(\frac{\partial f}{\partial t} + f \frac{\partial f}{\partial y}\right)\Delta t^2 + O(\Delta t^3) \end{aligned} \tag{106b}$$

- Are equal up to $O(\Delta t^3)$

6.6 Multistep methods

- Improve accuracy by linear combination of additional points
 - ◇ Using $Y_{n-s+1}, Y_{n-s+2}, \dots, Y_n$ allows estimation of derivatives of orders 1 to s at t_n

- If $s = 2$

◇ Have Y_{n-1}, Y_n

* So know f_{n-1}, f_n and

$$Y'_n =$$

$$Y''_n \approx \tag{107}$$

- Taylor series expansion

$$Y_{n+1} = Y_n + \Delta t Y'_n + \frac{1}{2}\Delta t^2 Y''_n$$

$$= \tag{108}$$

- For $s = 3$

◇ Also have $Y''_n = f''_n = (f_n - 2f_{n-1} + f_{n-2})/\Delta t^2$

$$Y_{n+1} = Y_n + \Delta t Y'_n + \frac{1}{2}\Delta t^2 Y''_n + \frac{1}{6}\Delta t^3 Y'''_n$$

$$= \tag{109}$$

- These are called *Adams-Bashforth* methods

- $s = 1$ recovers the Euler method
- If we use f_{n+1} as well, then method is *implicit*
- ◊ The $s = 2$ implicit method uses

$$Y'_n = f_n,$$

$$Y''_n \approx$$

$$Y'''_n \approx \tag{110}$$

◊ Giving

$$\begin{aligned} Y_{n+1} &= Y_n + \Delta t Y'_n + \frac{1}{2} \Delta t^2 Y''_n + \frac{1}{6} \Delta t^3 Y'''_n \\ &= Y_n + (1/12) \Delta t (5f_{n+1} + 8f_n - f_{n-1}). \end{aligned} \tag{111}$$

- Family of implicit methods called *Adams-Moulton* methods

6.7 Stability

- Accuracy is function of
 - ◊ Truncation error
 - * Reducing Δt decreases truncation error
 - ◊ Stability
- Stability often more limiting
- Prototype equation

(112)

◊ Complex λ

◊ Exact solution bounded provided $\text{Re}(\lambda) \leq 0$

* Look to see how numerical solution behaves

- Euler method

$$Y_{n+1} = Y_n + \Delta t f(t_n, Y_n)$$

=

$$(113)$$

- If Y_n to remain bounded as $n \rightarrow \infty$ then

$$(114)$$

- Condition very restrictive if $\lambda \ll 0$
 - ◊ Must then use very small time steps $\Delta t < 2/\lambda$
- Why did we look at this form of ode?
 - ◊ Tells us something about the behaviour of small errors

$$y_s = \tag{115}$$

- Substitute into ode

$$\begin{aligned} \frac{dy_s}{dt} &= \frac{dy}{dt} + \frac{d\varepsilon}{dt} \\ &= f(t, y_s) \\ &= \end{aligned}$$

$$(116)$$

- Error obeys ode of form (112)
 - ◊ Growth $\lambda = \partial f / \partial y$
 - * Equation inherently unstable if $\text{Re}(\lambda) > 0$
 - ◊ For method to be stable, then scheme must have errors ε decay
 - * If errors decay, then $E_n = O(e_n)$
- Can do same analysis on Backward Euler

$$Y_{n+1} = Y_n + \Delta t f(t_{n+1}, Y_{n+1})$$

$$=$$
(117)

- Rearranging for explicit expression for Y_{n+1}
 - ◇ Possible since equation linear

$$Y_{n+1} =$$
(118)

- Stability requires

$$(119)$$

- ◇ For $\text{Re}(\lambda) \leq 0$ this is always satisfied
 - * Backward Euler is unconditionally stable

- Crank-Nicholson method

$$(1 - \lambda \Delta t / 2) Y_{n+1} = (1 + \lambda \Delta t / 2) Y_n,$$
(120)

- ◇ Thus

$$Y_{n+1} = [(1 + \lambda \Delta t / 2) / (1 - \lambda \Delta t / 2)]^{n+1} Y_0,$$
(121)

- * So need $|(1 + \lambda \Delta t / 2) / (1 - \lambda \Delta t / 2)| < 1$
 - \Rightarrow Always for $\text{Re}(\lambda) < 0$

- Explicit methods
 - ◇ Cheap per step
 - ◇ Need small steps to keep stable
 - * Accuracy more dependent on stability than truncation error
- Implicit methods
 - ◇ Expensive per step
 - ◇ Stability not an issue

6.8 Predictor-corrector methods

- Consider Backward Euler

$$Y_{n+1} = \tag{122}$$

- Using theory for direct iteration (§3.6), write

$$Y_{n+1,i} = \tag{123}$$

◇ Let $Y_{n+1,0} = Y_n$

◇ Iterate until converged

* Convergence requires

$$\tag{124}$$

* Compare with criterion for stability of Euler method

* No real advantages

⇒ Scheme same order, but more expensive!

⇒ Must take smaller steps

- Crank-Nicholson

◇ Second order scheme

◇ Iterate

$$Y_{n+1,i} = \tag{125}$$

* Convergence requires $\frac{1}{2} \Delta t \partial f / \partial Y < 1$

⇒ Same as stability of Euler

⇒ Second order, but may be costly

- What happens we we iterate finite number of times rather than until convergence?

- Predictor-Corrector schemes

◇ Predict using explicit scheme

◇ Correct using semi-implicit scheme based on earlier predictions

6.8.1 Improved Euler method

- The simplest scheme

◇ Prediction using Euler

$$Y_{n+1}^{(1)} = \tag{126}$$

◇ One correction looks like Backward Euler

$$Y_{n+1}^{(2)} = \tag{127}$$

◇ Final solution the mean of these

$$Y_{n+1} = \tag{128}$$

* This is identical to first iteration of Crank-Nicholson given in (125)

- Stability?

◇ Look at $y' = \lambda y$

$$Y_{n+1}^{(1)} = Y_n + \lambda \Delta t Y_n, \tag{129a}$$

$$Y_{n+1}^{(2)} = Y_n + \lambda \Delta t Y_{n+1}^{(1)}$$

$$=$$

$$=, \tag{129b}$$

$$Y_{n+1} = (Y_{n+1}^{(1)} + Y_{n+1}^{(2)})/2$$

$$=$$

$$=$$

$$= \tag{129c}$$

- ◇ Stability requires $|1 + \lambda\Delta t + \frac{1}{2}\lambda^2\Delta t^2| < 1$ (for $\text{Re}\lambda < 0$)
 - * $\Delta t < 2|\lambda|^{-1}$
 - * Same as for Euler method
 - * Same as convergence criterion for iterative Crank Nicholson (125)
 - * Comparison with Taylor Series shows second order

6.8.2 Runge-Kutta methods

- Improved Euler simplest of family
- Can construct higher order methods by having more than one corrector step
- Fourth order often used

$$Y_{n+1} - Y_n = k = \Delta t f(t, Y)$$

$$k^{(1)} = \Delta t f(t_n, Y_n), \quad (130a)$$

$$k^{(2)} = \Delta t f(t_n + \frac{1}{2}\Delta t, Y_n + \frac{1}{2}k^{(1)}), \quad (130b)$$

$$k^{(3)} = \Delta t f(t_n + \frac{1}{2}\Delta t, Y_n + \frac{1}{2}k^{(2)}), \quad (130c)$$

$$k^{(4)} = \Delta t f(t_n + \Delta t, Y_n + k^{(3)}), \quad (130d)$$

$$Y_{n+1} = Y_n + (k^{(1)} + 2k^{(2)} + 2k^{(3)} + k^{(4)})/6. \quad (130e)$$

- To analyse,
 - ◇ Construct Taylor Series expansions for $k^{(2)}$, $k^{(3)}$ and $k^{(4)}$
 - * $k^{(2)} = \Delta t f(t_n + \frac{1}{2}\Delta t, Y_n + \frac{1}{2}k^{(1)}) = \Delta t [f(t_n, Y_n) + (\Delta t/2)(\partial f/\partial t + f\partial f/\partial y)]$
 - ◇ Compare with TS for Y_{n+1} . Need

$$\begin{aligned}
 Y'' &= df/dt \\
 &= \partial f/\partial t + \partial y/\partial t \partial f/\partial y \\
 &=
 \end{aligned}
 \tag{131}$$

$$\begin{aligned}
 Y''' &= d^2f/dt^2 \\
 &= \partial^2 f/\partial t^2 + 2f \partial^2 f/\partial t \partial y + \partial f/\partial t \partial f/\partial y + f^2 \partial^2 f/\partial y^2 + f (\partial f/\partial y)^2,
 \end{aligned}
 \tag{132}$$

$$Y'''' =$$

- Can match all terms up to and including order Δt^4

7 Higher order ordinary differential equations

- Solving higher order odes
- Techniques
 - ◇ Based on first order ode methods
 - ◇ Based on linear equations

7.1 Initial value problems

- Write explicit expression for highest order derivative

(133)

- Initial value problem
 - ◇ Initial conditions on $y, dy/dt, d^2y/dt^2, \dots, d^{n-1}y/dt^{n-1}$
- Split into system of first order equations
 - ◇ Write
 - * $x_0 = y$
 - * $x_1 = dy/dt$
 - * ...
 - * $x_{n-1} = d^{n-1}y/dt^{n-1}$
 - $x_0' =$
 - $x_1' =$
 - $x_2' =$
 -

$$\begin{aligned}
 x_{n-2}' &= \\
 x_{n-1}' &=
 \end{aligned}
 \tag{134}$$

- Use standard ode methods to advance system
- Euler method: advance all equations to t_1 using quantities at t_0
- In some cases it may be better to use a mixture of old and new values for right-hand side
 - ◊ Beyond scope of course

7.2 Boundary value problems

- Often not all boundary conditions at same boundary
 - ◊ Can not march from the known into the unknown (?)

7.2.1 Shooting method

- Analogy with shooting at a target
 - ◊ If you miss the first time, change the initial conditions (*i.e.* where the rifle is pointing) and try again
- Consider $y'' = f(t, y, y')$ with $y(0) = c_0$ and $y(1) = c_1$
 - ◊ Guess $y'(0) = \alpha$
 - ◊ Advance solution from $t = 0$ to $t = 1$ using $y'(0) = \alpha$
 - ◊ Compare $y(1; \alpha)$ with c_1
 - ◊ If *same*, solution found
 - ◊ Else revise α and try again
- Root finding
 - ◊ Use algorithms of §3
- For system of order n
 - ◊ Require n boundary conditions
 - ◊ If m specified at $t = t_0$, then
 - * Need to guess $n - m$ boundary conditions at $t = t_0$
 - * Optimisation problem tricky!

7.2.2 Linear equations

- For a linear system of equations

(135)

- Boundary conditions

$$\diamond y(t_0) = \alpha$$

$$\diamond y(t_1) + y'(t_1) = \beta$$

- Write as finite difference approximation

$$y'_i = \quad (136a)$$

$$y''_i = \quad (136b)$$

- Substitution

(137)

- Boundary conditions

$$\diamond \text{At } t = t_0$$

(138)

$$\diamond \text{At } t = t_1$$

(139)

* Not ideal!

- This leads to system of linear equations

...

(140)

- ◇ Tridiagonal system (see §4.5)
- Our approximations in interior second order
 - ◇ But our boundary condition at t_1 first order
 - ◇ So overall solution only first order
- Would like to make solution second order
 - ◇ Need to make boundary condition second order
 - ◇ Could use second order one-sided approximation

(141)

* Last equation becomes

(142)

⇒ System no longer tridiagonal!

- Alternative
 - ◇ Introduce a dummy mesh point t_{n+1}
 - * Assume ode applies for t_n as well
 - * Apply bc using central difference

$$(1 + \frac{1}{2}a\Delta t)Y_{n-2} + (b\Delta t^2 - 2)Y_{n-1} + (1 - \frac{1}{2}a\Delta t)Y_n = c\Delta t^2,$$

(143)

- ◇ Tridiagonal structure maintained
- Can do same tricks if boundary condition on y' at t_0
- Can cope with higher order linear odes
 - ◇ Normally created banded structure
 - ◇ Use solver which maintains this
- Nonlinear odes
 - ◇ Combine this approach with root finding
 - ◇ Beyond the scope of this course

7.3 Other considerations*

7.3.1 Truncation error*

7.3.2 Error and step control*

8 Partial differential equations

- Concentrate second order linear pdes

8.1 Laplace equation

- In 2D

$$(144)$$

◇ In rectangular domain

* $[x_0, x_1], [y_0, y_1]$

◇ Discretise into $m+1$ by $n+1$ points

$$x_i = \quad (145a)$$

$$y_j = \quad (145b)$$

* $i = 0, m$

* $j = 0, n$

* $\Delta x = (x_1 - x_0)/m$

* $\Delta y = (y_1 - y_0)/n$

- The solution

$$\Phi_{ij} = \quad (146)$$

- Let $\Phi_{ij} \approx \phi_{ij}$ be approximate solution
- Taylor series for points near ϕ_{ij} :

$$\Phi_{i+1,j} = \quad (147a)$$

$$\varphi_{i-1,j} = \quad (147b)$$

$$\varphi_{i,j+1} = \quad (147c)$$

$$\varphi_{i,j-1} = \quad (147d)$$

- Finite difference approximation of $\partial^2 \varphi / \partial x^2$ and $\partial^2 \varphi / \partial y^2$

(148)

◇ For internal points

◇ Boundary conditions (on φ and/or normal derivative of φ)

8.1.1 Direct solution

- Linear system (provided bc's linear)

(149)

◇ $(m+1) \times (n+1)$ mesh points

◇ \mathbf{A} has $[(m+1) \times (n+1)]^2$ elements

$$\Phi^*_{ij} = \tag{154}$$

◇

◇

• Example

- Stencil or template
- Very slow to converge

8.1.2.2 Gauss-Seidel

- Less memory required (there is no need to store Φ^*).
- Faster convergence (although still relatively slow).
- Example

8.1.2.3 Red-Black ordering

- Variant of Gauss-Seidel
 - ◇ Uses two passes in chess board pattern
 - ◇ No interdependence of the solution updates within a single pass aids vectorisation.
 - ◇ Faster convergence at low wave numbers.

--	--	--	--	--

8.1.2.4 Successive Over Relaxation (SOR)

- Jacobi, Gauss-Seidel and Red-Black are all slow to converge
 - ◊ Normally converge monotonically
 - ◊ Try to speed up convergence
 - * Instead of

$$\Phi^*_{ij} = (\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1})/4,$$

* Try

$$\Phi^*_{ij} = \sigma \Phi_{ij} + (1-\sigma)(\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1})/4 \tag{155}$$

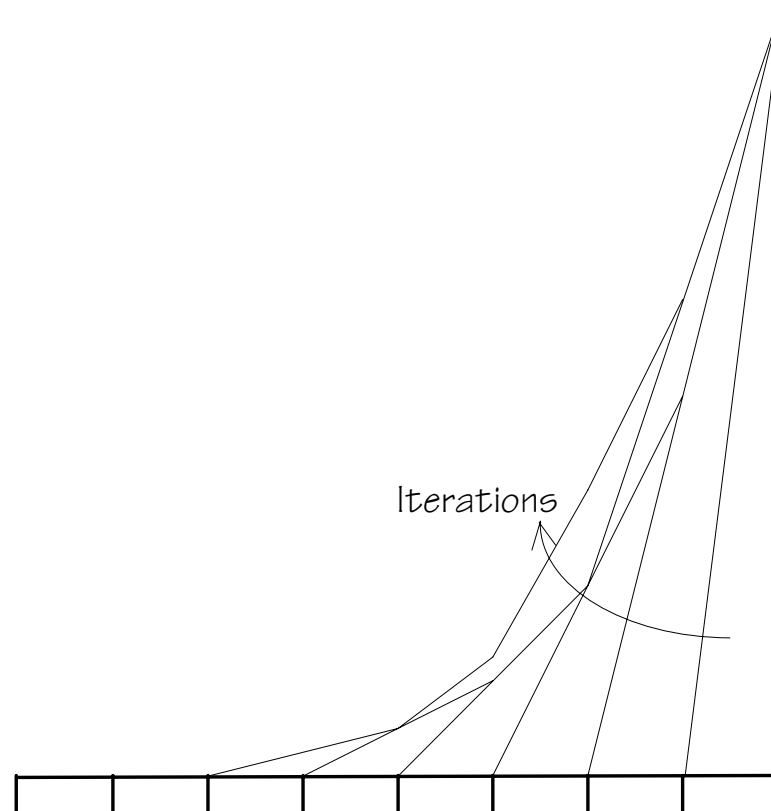
* σ some constant

* For given problem there is an optimal value, but typically 1.2 to 1.4

8.1.3 Multigrid*

- Not examinable
- Relaxation good at removing high wavenumber (frequency errors)

- Very slow at removing low wavenumber errors
 - ◇ Jacobi takes $m/2$ iterations for boundary conditions to influence centre of grid



- Multigrid looks at hierarchy of grids
 - ◇ Coarse grids for removing low wavenumber errors
 - ◇ Fine grids for removing high wavenumber errors

- Algorithm

- ◇ 1. Select the initial finest grid resolution $p=P_0$ and set $\mathbf{b}^{(p)} = 0$ and make some initial guess at the solution $\Phi^{(p)}$
 - ◇ 2. If at coarsest resolution ($p=0$) then solve $\mathbf{A}^{(p)}\Phi^{(p)}=\mathbf{b}^{(p)}$ *exactly* and jump to step 7
 - ◇ 3. Relax the solution at the current grid resolution, applying boundary conditions
 - ◇ 4. Calculate the error $\mathbf{r} = \mathbf{A}\Phi^{(p)}-\mathbf{b}^{(p)}$
 - ◇ 5. Coarsen the error $\mathbf{b}^{(p-1)}\leftarrow\mathbf{r}$ to the next coarser grid and decrement p
 - ◇ 6. Repeat from step 2
 - ◇ 7. Refine the correction to the next finest grid $\Phi^{(p+1)} = \Phi^{(p+1)}+\alpha\Phi^{(p)}$ and increment p
 - ◇ 8. Relax the solution at the current grid resolution, applying boundary conditions
 - ◇ 9. If not at current finest grid (P_0), repeat from step 7
 - ◇ 10. If not at final desired grid, increment P_0 and repeat from step 7
 - ◇ 11. If not converged, repeat from step 2.
- Can converge very rapidly
 - * $O(n)$ operations
 - \Rightarrow Direct solution $O(n^3)$

⇒Hockney's $O(n \log n)$

⇒FFT (§8.1.5) $O(n \log n)$

- Irregular geometries and complex boundary conditions easy
- Can also be applied to other types of ode and some more general linear systems

8.1.4 The mathematics of relaxation*

- Not examinable
- Why do relaxation methods work?
- Look at problem as one of root finding
 - ◇ Seek $f(x) = \mathbf{0}$ where

$$f(x) = \mathbf{Ax} - \mathbf{b}, \quad (156)$$

- Use Direct Iteration approach

$$\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n), \quad (157)$$

- ◇ Selecting

$$\mathbf{g}(x) = \mathbf{D}^{-1}\{[\mathbf{A}+\mathbf{D}]\mathbf{x} - \mathbf{b}\}, \quad (158)$$

* \mathbf{D} arbitrary (not singular!); normally taken to be diagonal

- ◇ Analyse convergence as for single variable direct iteration

$$\begin{aligned} \boldsymbol{\varepsilon}_{n+1} &= \mathbf{x}_{n+1} - \mathbf{x}^* \\ &= \mathbf{D}^{-1}\{[\mathbf{A}+\mathbf{D}]\mathbf{x}_n - \mathbf{b}\} - \mathbf{D}^{-1}\{[\mathbf{A}+\mathbf{D}]\mathbf{x}^* - \mathbf{b}\} \\ &= \mathbf{D}^{-1}[\mathbf{A}+\mathbf{D}](\mathbf{x}_n - \mathbf{x}^*) \\ &= \mathbf{D}^{-1}[\mathbf{A}+\mathbf{D}]\boldsymbol{\varepsilon}_n \\ &= \{\mathbf{D}^{-1}[\mathbf{A}+\mathbf{D}]\}^{n+1} \boldsymbol{\varepsilon}_0. \end{aligned} \quad (159)$$

- ◇ Convergence requires

$$\|\boldsymbol{\varepsilon}_{n+1}\| = \|\mathbf{B}\boldsymbol{\varepsilon}_n\| < \|\boldsymbol{\varepsilon}_n\|, \quad (160)$$

* $\mathbf{B} = \mathbf{D}^{-1}[\mathbf{A}+\mathbf{D}]$

- ◇ Need consider only the eigen values (these give the *scale* of \mathbf{B})

$$\mathbf{B}_{SOR} - \lambda \mathbf{I} = \sigma \left[\mathbf{B}_J - \frac{\lambda + \sigma - 1}{\sigma} \mathbf{I} \right] \quad (169)$$

- Eigen values (λ_{SOR}) related to those for Jacobi and Gauss-Seidel (λ_J) by

$$\lambda_{SOR} = 1 + \sigma(\lambda_J - 1) \quad (170)$$

8.1.4.3 Other equations*

- Relaxation can be used for other pdes
- Also for general linear systems
- Will typically converge if \mathbf{A} is diagonally dominant
- May need to have $\sigma < 1$ for some cases

8.1.5 FFT*

- Not examinable
- FFT \equiv *Fast Fourier Transform*
- Requires $n = 2^p$ points
- Uses $O(n \log n)$ operations
- Divide and conquer

8.1.6 Boundary elements*

- Not examinable

8.1.7 Finite elements*

- Not examinable

8.2 Poisson equation

- Treat in identical way to Laplace except have non-zero right-hand side

8.3 Diffusion equation

- Time dependent
- In 2D

(171)

- Boundary conditions
 - ◊ $u = 0$ on $x = 0,1, y = 0,1$
- Initial conditions
 - ◊ $u(x,y,t=0) = u_0(x,y)$

8.3.1 Semi-discretisation

- Discretise in space THEN solve system of odes in time
- Take $\Delta x = \Delta y = 1/m$ and $D = 1$
- In the interior

$$\frac{\partial u_{i,j}}{\partial t} \approx \quad (172)$$

- On boundaries $u_{0,j} = u_{m,j} = u_{i,0} = u_{i,m} = 0$
- System of coupled equations

(173)

- Can use any of the time stepping methods discussed earlier
 - ◊ Need to watch out for stability!

8.3.2 Euler method

- Euler method $Y_{n+1} = Y_n + \Delta t f(Y_n, t_n)$ leads to

(174)

- The Courant number is

$$\mu = \quad (175)$$

8.3.3 Stability

- Approach similar to ode's
- Choose

$$U^{(0)}_{i,j} = \quad (176)$$

- α and β chosen to satisfy boundary conditions
 - ◇ $\alpha = p\pi/m$
 - ◇ $\beta = q\pi/m$

- Now $\sin(A-B) = \sin A \cos B - \cos A \sin B$
 and $\sin(A+B) = \sin A \cos B + \cos A \sin B$
 so $\sin(A-B) + \sin(A+B) = 2 \sin A \cos B$

$$(177)$$

- Applying repeatedly gives

$$U^{(n)}_{i,j} = \tag{178}$$

- So $|1 - 4\mu[\sin^2(\alpha/2) + \sin^2(\beta/2)]| < 1$, or

$$\Delta t < \tag{179}$$

- If stable, then $E_n = O(e_n) = O(\Delta t^2)$
- Doubling spatial resolution requires time step to decrease by factor of four
 - ◊ Overall computational cost increases by factor of 16
- Diffusion equation in 1D or 3D handled in same manner

8.3.4 Model for general initial conditions

- Why did we choose initial conditions of form $\sin(k\pi x/L_x) \sin(l\pi y/L_y)$?
 - ◊

8.3.6 ADI*

8.4 Advection*

8.4.1 Upwind differencing*

8.4.2 Courant number*

8.4.3 Numerical dispersion*

8.4.4 Shocks*

8.4.5 Lax-Wendroff*

8.4.6 Conservative schemes*

9. Number representation*

9.1. Integers*

Decimal	Binary	Two's compliment (16 bit)
1	1	0000000000000001
5	101	0000000000000101
183	10110111	000000010110111
255	11111111	0000000111111111
32767	1111111111111111	0111111111111111
 	Decimal	Two's compliment (16 bit)
<i>A</i>	5	0000000000000101
xor 11111111111111		1111111111111010
+ 1	-5	1111111111111011
 	Decimal	Two's compliment (16 bit)
<i>-A</i>	-5	1111111111111011
xor 11111111111111		0000000000000100
+ 1	5	0000000000000101
 	Decimal	Two's compliment (16 bit)
Decimal	Binary	Two's compliment (16 bit)
-1	-1	1111111111111111
-5	-101	1111111111111011
-183	-10110111	1111111101001001
-255	-11111111	1111111100000001
-32767	-1111111111111111	1000000000000001
-32768	-1111111111111110	1000000000000000
 	Decimal	Two's compliment (16 bit)
	-5	1111111111111011
+	+ 183	000000010110111
binary sum	???	1000000010110010

discard carry bit

178

0000000010110010

9.2. Floating point*

Decimal	Sign	Exponent	Mantissa	Value Stored
				seeeeeee emmmmmmmmm mmmmmmmmm mmmmmmmmm
0.0	0_2	0 00000000_2	0.0 0.00000_2	0.0 $00000000\ 00000000\ 00000000\ 00000000_2$
1.0	$+$ 0_2	0 01111111_2	1.0 1.00000_2	1.0 $00111111\ 10000000\ 00000000\ 00000000_2$
8.0	$+$ 0_2	3 10000100_2	1.0 1.00000_2	8.0 $01000001\ 00000000\ 00000000\ 00000000_2$
3.0	$+$ 0_2	1 10000000_2	1.5 1.10000_2	3.0 $01000000\ 01000000\ 00000000\ 00000000_2$
-3.0	$-$ 1_2	1 10000000_2	1.5 1.10000_2	-3.0 $11000000\ 01000000\ 00000000\ 00000000_2$
0.25	$+$ 0_2	-2 01111101_2	1.0 1.00000_2	0.25 $00111110\ 10000000\ 00000000\ 00000000_2$
0.2	$+$ 0_2	-3 01111100_2	1.6 1.10011_2	$0.2 + 0.0000000149\dots$ $00111110\ 01001100\ 11001100\ 11001101$

9.3. Rounding and truncation error***9.4. Endians*****10. Computer languages*****10.1. Procedural verses Object Oriented*****10.2. Fortran 90****10.2.1. Procedural oriented***10.2.2. Fortran enhancements****10.3. C++****10.3.1. C***10.3.2. Object Oriented***10.3.3. Weaknesses****10.4. Others****10.4.1. Ada***10.4.2. Algol***10.4.3. Basic***10.4.4. Cobol***10.4.5. Delphi***10.4.6. Forth***10.4.7. Lisp***10.4.8. Modula-2**

*10.4.9. Pascal**

*10.4.10. PL/I**

*10.4.11. PostScript**

*10.4.12. Prolog**

*10.4.13. Smalltalk**

*10.4.14. Visual Basic**