April 18, 2021

Numerical methods

John D. Fenton

Institute of Hydraulic Engineering and Water Resources Management
Vienna University of Technology, Karlsplatz 13/222,
1040 Vienna, Austria

URL: *http://johndfenton.com/*
URL: *mailto:JohnDFenton@gmail.com*

Abstract

These notes provide an introduction to numerical methods for the solution of physical problems. Extensive use will be made of *Excel Solver* for the solving or approximating the solution of systems of equations.

Table of Contents

Accompanying documents

The following Excel documents can be found in the same directory as this file:

URL: *http://johndfenton.com/Lectures/Numerical-Methods/*

and are referred to in the subsequent notes

File name	Description
1-FUNCTIONS.XLS	Basic *Excel* functions
2-SOLUTION.XLS	Solution of a single equation in a single variable
3-SOLVER.XLS	*Solver* applied to solution of equations, interpolation, and approximation
4-FITTING.XLS	A curve fitting example where using *Excel Trendline* gave poor results
5-DIFF-EQNS.XLS	Different problems solved by different methods
6-SPLINES.XLS	Use of cubic splines for interpolation
SPLINES.XLS	Contains the spline functions necessary for the previous spreadsheet

1. Introduction

Through the use of numerical methods many problems can be solved that would otherwise be thought to be insoluble. In the past, solving problems numerically often meant a great deal of programming and numerical problems. Programming languages such as Fortran, Basic, Pascal and C have been used extensively by scientists and engineers, but they are often difficult to program and to debug. Modern commonly-available software has gone a long way to overcoming such difficulties. Matlab, Maple, Mathematica, and MathCAD for example, are rather more user-friendly, as many operations have been modularised, such that the programmer can see rather more clearly what is going on. However, spreadsheet programs provide engineers and scientists with very powerful tools. The two which will be referred to in these lectures are *Microsoft Excel* and *Libre Office Calc*. Spreadsheets are more intuitive than using high-level languages, and one can easily learn to use a spreadsheet to a certain level. Yet often users do not know how to translate powerful numerical procedures into spreadsheet calculations.

It is the aim of these lectures to present the theory of the most useful numerical methods and to show how to implement them, usually in a spreadsheet, but occasionally also in a programming language, for sometimes spreadsheets are not adequate for large-scale computations.

The two spreadsheets we have mentioned are:

- *Microsoft Excel* – widely known and used. Some of its effectiveness for numerical computations comes from

a pair of modules, *Goal Seek* and *Solver*, which obviate the need for much programming and computations. *Goal Seek*, is easy to use, but it is limited – with it one can solve a single equation, however complicated or however many spreadsheet cells are involved, whether the equation is linear or nonlinear. *Solver* is much more powerful. It was originally designed for optimisation problems, where one has to find values of a number of different parameters such that some quantity is minimised, usually the sum of errors of a number of equations. With this tool one can find such optimal solutions, or solutions of one or many equations, even if they are nonlinear. In this course of lectures we will use it to simplify many procedures. It is somewhat annoying, however, that Solver is not automatically installed. You should open Excel, then click on the Tools tab. If Solver is not there you will have to click on Add-ins, and proceed to install it.

- *Libre Office or OpenOffice Calc* – Libre Office is a shareware version of Microsoft Office, with a word processor, spreadsheet, presentation program, and drawing program; it can be downloaded from the site, `https://www.libreoffice.org/`. The package was originally Open Office, but that has not received much maintenance in recent years The spreadsheet is called *Calc*. It has most of the features of *Excel*, including *Goal Seek*, but at the time of writing, its version of *Solver* is not quite as good as that of Excel. It uses a Genetic Algorithm, and is slower. In this course we will concentrate on *Excel* and will use that as a generic name for the two programs.

2. Accuracy, errors and computer arithmetic

2.1 Accuracy

Excel stores and calculates with 15 digit accuracy. This is equivalent to double precision in some programming languages, and is accurate enough that most calculations do not suffer from significant loss of accuracy. Whenever numbers are stored on a machine a small error usually occurs. *Excel* can store numbers in the range from $2^{-2^{10}} = 2^{-1024} \approx 10^{-308}$ to $2^{2^{10}} = 2^{1024} \approx 10^{308}$. If the number is less than the minimum it stores it as 0, if greater than the maximum it records it as an *overflow* in the form `#NUM!`. Unlike programming languages, *Excel* does not distinguish between integers and floating point numbers

2.2 Rounding

- *Excel* displays numbers rounded to the accuracy of the display. For example if you evaluate 2/3 and the cell has been formatted to display 4 decimal places, it will appear as 0.6667.
- If you need to round a number there is a function `ROUND(number,decimal_places)` which rounds a number to a specified number of decimal places. If `decimal_places` is 0, then the number is rounded to the nearest integer, which is often useful in programming.

  Example: `ROUND(3.14159,3)` gives `3.142`.

2.3 Errors

1. Blunders: These are not really errors, but are mistakes, such as typing the wrong digit.
2. Errors in the model: A mathematical model in civil and environmental engineering does not usually represent every aspect of a real problem, such as the neglect of turbulence in hydraulics.
3. Errors in the data: Most data from a physical problem contain errors or uncertainties, due to the limited accuracy of the measuring device.
4. Truncation error: This is the error made when a limiting process is truncated before one has reached the limiting value such as when an infinite series is broken off after a finite number of terms.

   Example: computing $\sin x$ from the first three terms of its power series expansion $x - x^3/3! + x^5/5!$.
5. Roundoff error: This is the error caused by the limited accuracy of the computer, and a roundoff error occurs whenever numbers are stored and arithmetic operations performed.

In this course we will be concerned mainly with the last two types of errors.

2.3.1 Absolute and relative errors

Let x be the approximate value of a number whose exact value is X.

The *absolute error* in x is $e = x - X$.

The *relative error* in x is $r = e/x = (x - X)/X$. The relative error is often given as a percentage.

> A number which is correct to n decimal places has an error of less than $1/2$ in the nth decimal place,
> Absolute error $|e| \leqslant \frac{1}{2} \times 10^{-n}$
> A number which is correct to n significant digits has an error of less than $1/2$ in the nth digit,
> Relative error $|r| \leqslant \frac{1}{2} \times 10^{1-n}$

Example: Consider $X = 23.494$ and $x = 23.491$. x is correct to 4 significant figures and 2 decimal places:

$$\begin{aligned} |e| &= 0.003 < \tfrac{1}{2} \times 10^{-2} \\ |r| &\approx 0.00013 < \tfrac{1}{2} \times 10^{-3}. \end{aligned}$$

2.3.2 Roundoff error – an example

If we subtract two nearly equal numbers this leads to a loss of significant digits which can cause serious error if used in further calculations. As an example, consider the quadratic formula for solving $ax^2 + bx + c = 0$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \tag{2.1}$$

and if b is a large number, then $b \approx \sqrt{b^2 - 4ac}$ and subtracting the two to give the smaller root will be inaccurate due to roundoff error. For example, take $a = 1$, $b = -10^7$, and $c = 1$. Using *Excel* (with 15 figure accuracy) to evaluate the expression for the smaller root gives 0.9965×10^{-7}, which shows roundoff error. The correct answer to 15 figures is $x = 1. \times 10^{-7}$.

Some different procedures Here as an example of different procedures we might adopt in other problems we obtain the smaller root by different methods for the case $a = 1$, $b = -10^7$, and $c = 1$. **Note that we leave the best method second!**:

1. As an example of the power and utility of *Solver*, and without requiring a great deal of mathematical or computational knowledge, *Solver* was used to obtain the root, by choosing an approximate small value of $x = 0$ and finding the solution of $x^2 - 10^7 x + 1 = 0$, which gave an answer of $1.00000000000001 \times 10^{-7}$, correct to the full accuracy of *Excel*. The numerical optimisation method which *Solver* uses had no roundoff error problems here.

2. In January 2019, John D. Sahr of Electrical and Computer Engineering at the University of Washington wrote a letter to the author, suggesting the following general and elegant solution. We should all use it as the standard algorithm for solving a quadratic!

   It uses also the alternative solution of the quadratic that can be obtained by re-writing the quadratic $ax^2 + bx + c = 0$ as $c/x^2 + b/x + a = 0$, here solving for $1/x$ using the usual formula, giving the alternative solutions

   $$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}. \tag{2.2}$$

   Although this has the same problem, the solution liable to roundoff error here is for the other root, and so we can use the algorithm

   - If $b \geqslant 0$ compute $d = -b - \sqrt{b^2 - 4ac}$
   - If $b < 0$ compute $d = -b + \sqrt{b^2 - 4ac}$
   - Then d will no longer be the small difference of large numbers, and the two roots will be, first from the usual formula (2.1), then (2.2):

   $$x = \frac{d}{2a}, \frac{2c}{d}$$

Example 2.1 Using 6-figure arithmetic, solve the above example, $a = 1$, $b = -10^7$, and $c = 1$:

$b \geqslant 0$, $d = -b - \sqrt{b^2 - 4ac} = 2.0 \times 10^{-7}$

$x = \frac{2.0 \times 10^{-7}}{2 \times 1} = 1.000000 \times 10^{-7}$ and $\frac{2}{2.0 \times 10^{-7}} = 1.000000 \times 10^{+7}$.

3. Excel functions

Accompanying document Excel workbook 1-FUNCTIONS.XLS sets out the main commands which are necessary at this stage. In the experience of the lecturer, Excel commands are accurate and robust although there has been some discussion about some of the statistical routines.

	A	B	C	D	E	F	G	H
1	**Functions in Excel**		This document is "1-Functions.xls"					
2								
3	**Basic numerical operations**							
4		=ABS(-1)	1	Absolute value				
5		=INT(9.9)	9	Truncates down to the integer value				
6		=INT(-9.9)	-10	As we wrote, truncates DOWN, rather than closer to zero				
7		=RAND()	0.237513	A pseudo-random number between 0 and 1				
8		=90+(100-90)*RAND()	98.5124	We use a more general form for any real number in a range, such as 90 to 100 here				
9								
10	**Powers**							
11		=2^4	16.0000	General exponentiation				
12		=2^0.5	1.4142					
13		=SQRT(2)	1.4142	Square root				
14								
15	**Significant digits**							
16		=ROUND(C11,D11-1-INT(LOG10(ABS(C11))))	120	123.4567	2	A possibly-useful function		
17		=Significant(1234.56789,6)	1234.57	Programmed up as a VBA function in this worksheet				
18								
19	**Maximum value of an array of numbers**							
20		=MAX(1,2,3)	3.0000	Internally, all numbers are real numbers, which shows if formatted				
21		=MAX(D23:E24)	4					
22		Array whose largest element is required:			1	2		
23					3	4		
24	**Exponential and related functions**							
25		=EXP(1)	2.7183					
26		=COSH(1)	1.5431					
27		=SINH(1)	1.1752					
28	**Logarithms**							
29		=LOG(8,2)	3.0000	LOG(Number,Base) is the general form				
30		=LN(EXP(1))	1.0000	LN is the natural logarithm				
31		=LOG(100)	2.0000	If base is omitted, it is assumed to be 10. Naughty Microsoft.				
32	**Constants**							
33		=PI()	3.1416	Useful. Remember the brackets.				
34								
35	**Trigonometric functions**							
36		=SIN(PI()/2)	1.0000	All arguments of trigonometric functions are in radians.				
37		=ASIN(1.)	1.5708	Results of all inverse trigonometric functions are in				
38		=COS(90)	-0.4481	This is the cosine of 90 radians				
39		=SIN(RADIANS(90))	1.0000	There is an easy conversion if you must work in degrees				
40		=SIN(90*PI()/180)	1.0000	But so is the usual conversion				
41		=ATAN(1)	0.7854	ATAN returns values in the range -PI/2 to +PI/2				
42		=DEGREES(ATAN(1))	45.0000					
43		=DEGREES(ATAN2(-1,-1))	-135	ATAN2 Generalised inverse tangent function				
44								
45	**Array formulae**							
46								
47		These three numbers have been named "data"			1	2	3	
48		=SUM(data)	6	Named block				
49		=SUM(E48:G48)	6	More usual				
50		=AVERAGE(data)	2.0000	Straightforward average				
51		{=SUM(data^2)}	14	Sum of squares - entered with CTL-ALT-ENTER				

Figure 3-1. Elementary functions

	A	B	C	D	E	F	G	H
52	**Matrix Functions**							
53		Type numbers into array:	1	3				
54			2	4				
55								
56		For matrix inverse the array must be square, then select another square region, type =MINVERSE(You can now select your first region), then CTL-SHIFT-ENTER						
57		=MINVERSE(C54:D55)	-2	1.5				
58			1	-0.5				
59								
60		Now to check, multiply the matrix by its inverse, select a blank array of the correct size, type =MMULT(select one array,select another array) then CTL-SHIFT-ENTER. Remember that an m*n matrix multiplied by an n*p matrix gives an m*p matrix						
61		=MMULT(C54:D55,C58:D59)	1	0				
62			0	1				
63		Transpose - similar to the above, but with TRANSPOSE(…) then CTL-SHIFT-ENTER						
64		=TRANSPOSE(C58:D59)	-2	1				
65			1.5	-0.5				
66								
67		In each of the above array operations, selecting an array and completing typing with CTL-SHIFT-ENTER inserts the command in all elements. If you only get one element you have forgotten to (a) select a range before entering formula or (b) complete with CTL-SHIFT-ENTER						
68								
69	**Decision Functions**							
70								
71		=IF(Condition,true-value,false-value)		The IF function is used when you want a formula to return different results depending on the value of a				
72		=IF(2<3,-1,0)	-1.0000					
73		=IF(2<3,"True","False")	True					
74		The condition can be any valid Excel expression, and will usuallycontain a comparison operator:						
75			=	Equal to				
76			>	Greater than				
77			>=	Greater than or equal to				
78			<	Less than				
79			<=	Less than or equal to				
80			<>	Not equal to				
81		=IF(1/5=0.2,"Tested correctly","Did not evaluate to true in real arithmetic")	Tested correctly	Be careful using real arithmetic to test for equality or non-equality. In some software to machine accuracy the two apparently-equal expressions might be different, although here it seemed all right.				
82		Boolean statements may be used in the condition:						
83		AND(), OR(), NOT()						
84		=IF(AND(1=1,2=2,3=3),"All true","Not all true")	All true	AND: all statements in the argument list are true				
85		=IF(OR(1=1,2=1,3=1),"At least one true","None true")	At least one true	OR: at least one statement is true				
86								
87	**Complex numbers**							
88								
89		=COMPLEX(1,2)	1+2i					
90		=IMPRODUCT("1+2i","1+i")	-3+4i	Special routines have to be used - of which there are				
91		=IMPRODUCT(C90,C90)	-3+4i					

Figure 3-2. Advanced functions

4. Solutions of nonlinear equations

4.1 The problem

The problem is: if we have one equation in one unknown, and we need to find one, some, or all of the values of that unknown which satisfy the equation. That is, we have to solve $f(x) = 0$, where $f(x)$ is a known function of x. This is finding the value of x where the graph of $y = f(x)$ intersects the x axis. There may be more than one solution.

Examples

$x - 2 = 0$	A linear equation, with a simple solution $x = 2$.
$x^2 - 4x + 3 = 0$	A quadratic – we might be able to factorise it, or use the formula for solving quadratics.
$x^5 - 3x^3 + x - 1 = 0$	A higher degree polynomial – we might be able to factorise and solve, but in general it is unlikely
$\sin x - x + 1 = 0$	A *transcendental* equation, which we are most unlikely to be able to solve.

In the case of the last two equations we have to resort to numerical methods.

4.2 Methods

A number of different methods are presented here, for each has certain characteristics which favours its use in certain situations. For the purposes of this lecture course, however, we will see that *Goal Seek* and *Solver* are the simplest and best to use, but in general any one of the others might have advantages.

4.2.1 Trial and error

This involves simply guessing a value of x, evaluate $f(x)$, compare with zero, and then guess again. It is often used for one-off problems, but we often need to be much more systematic about it.

4.2.2 Newton's method

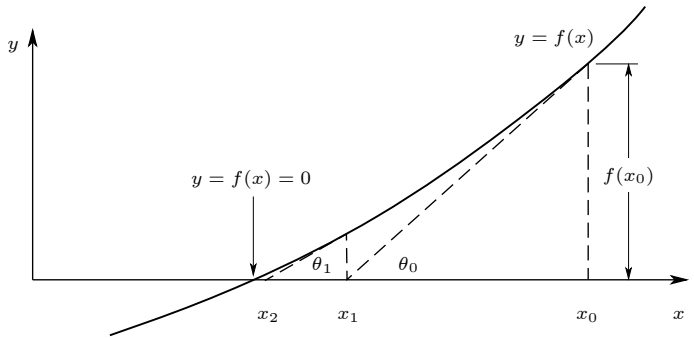


Figure 4-1. Newton's method

This method, if it converges, does so very quickly. Instead of just computing $f(x)$, the derivative $f'(x)$ is also calculated, and it is assumed that the next estimate of the *root*, or solution, is where the tangent crosses the x axis. The values of $f(x)$ and $f'(x)$ there are calculated and the process repeated until it converges. The process is shown in figure 4-1, such that at iteration n

$$\tan\theta_n = \frac{df}{dx}(x_n) = f'(x_n) = \frac{f(x_n)}{x_n - x_{n+1}},$$

from which an expression for the next estimate x_{n+1} is obtained

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n + \delta_n \quad \text{where } \delta_n = -\frac{f(x_n)}{f'(x_n)}. \tag{4.1}$$

The convergence of this method is quadratic, such that the number of correct digits doubles at each step. However,

if a double root occurs, such that the curve just touches the graph at the root and then curves away again, convergence is less rapid. Usually the correction at each stage $\delta_n = -f(x_n)/f'(x_n)$ is calculated and monitored, and when it is less than a certain amount in magnitude, the process terminated.

Example 4.1 With this method we can develop, for example, an algorithm to calculate the square root of a number. Let the function $f(x) = x^2 - N = 0$, where N is the number whose square root we want. Now, $f'(x) = 2x$, and substituting into equation (4.1):

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

which is a simple and perhaps obvious algorithm: take the mean of your estimate and the number divided by that quantity. Now, let's find the square root of 100, starting with 100/2 as the first estimate:

Step	x_n	$\frac{1}{2}\left(x_n + \frac{N}{x_n}\right)$
1	50	$\frac{1}{2}\left(50 + \frac{100}{50}\right) = 26.0$
2	26.0	$\frac{1}{2}\left(26 + \frac{100}{26}\right) = 14.923$
3	14.923	$\frac{1}{2}\left(14.923 + \frac{100}{14.923}\right) = 10.812$
4	10.812	$\frac{1}{2}\left(10.812 + \frac{100}{10.812}\right) = 10.0305$
5	10.0305	$\frac{1}{2}\left(10.0305 + \frac{100}{10.0305}\right) = 10.00005$

4.2.3 The secant method

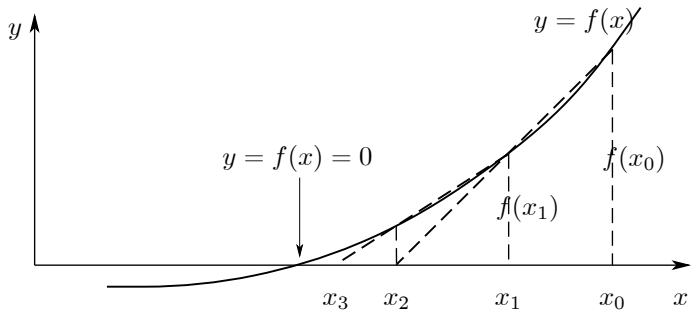


Figure 4-2. Secant method

Newton's method is unpleasant to apply if the function $f()$ is complicated, such that differentiation is a problem. In such cases it is appropriate to approximate the derivative by the secant approximation to the tangent. This means that $f'(x_n)$ is approximated by

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} = \frac{f(x_n) - f(x_{n-1})}{\delta_{n-1}},$$

and the scheme requires two starting values $(x_0, f(x_0))$ and $(x_1, f(x_1))$ and it becomes

$$x_{n+1} = x_n + \delta_n \quad \text{where } \delta_n = \frac{-f(x_n)\delta_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Example 4.2 We repeat the above problem using the secant method, using 0 and 50 as the first two estimates. Application of the method is shown in the following table.

Step	x_n	$\delta_n = \frac{N-x_n^2}{x_n^2-x_{n-1}^2}\delta_{n-1}$
0	0.	50
1	50	$\frac{100-2500}{2500-0}50 = -48.0$
2	$50 - 48.0 = 2.0$	$\frac{100-2^2}{2^2-50^2} \times -48 = 1.846$
3	$2 + 1.846 = 3.846$	$\frac{100-3.846^2}{3.846^2-2^2} \times 1.846 = 14.575$
...	...	...
9	9.999979	0.000021
10	10.000000	0.000000

Note that the method was somewhat slowly convergent as both numerator and denominator of the expression went to zero in this special case. However, it still worked and the method has the decided advantage for complicated functions that it is not necessary to obtain the derivative of the function.

4.2.4 Fixed point, direct, or simple iteration

This is a simple and obvious method and it is simple enough that if it works it can be very useful – *but* it goes wrong in about 50% of cases! We will investigate the conditions below. Once again, our problem is to solve the equation $f(x) = 0$. However, if we can re-arrange this in the form $x = g(x)$, where $g(x)$ is some function of x, then this gives the scheme

$$x_{n+1} = g(x_n),$$

meaning, assume some initial value x_0, evaluate $g(x_0)$ and use this value as the next estimate x_1 and so on.

Example 4.3 Use direct iteration to solve the equation $x^3 - x - 1 = 0$.

This obviously suggests the scheme $x_{n+1} = x_n^3 - 1$. We start with $x_0 = 1$:

x_n	$x_n^3 - 1$
1	$1^3 - 1 = 0$
0	$0^3 - 1 = -1$
-1	$-1^3 - 1 = -2$
-2	$-2^3 - 1 = -9$
-9	$-9^3 - 1 = -730$

It is obvious that the scheme is unstable and is not converging to a solution. Let us now try re-arranging the equation in the form: $x_{n+1} = (x_n + 1)^{1/3}$, with results:

x_n	$(x_n + 1)^{1/3}$
1	$(1 + 1)^{1/3} = 1.259$
1.259	$(1.259 + 1)^{1/3} = 1.3121$
1.3121	$(1.3121 + 1)^{1/3} = 1.3223$
1.3223	$(1.3223 + 1)^{1/3} = 1.3243$
1.3243	$(1.3243 + 1)^{1/3} = 1.3246$

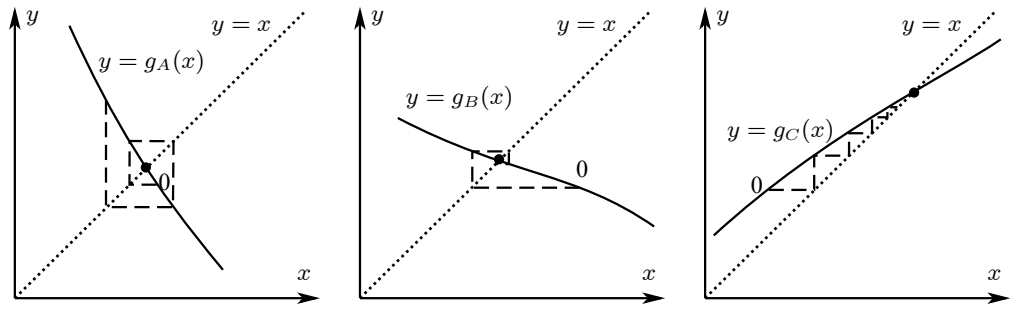
It can be seen that the process is converging quite well.

Why did one method work and not the other? The reason can be explained graphically, if we consider solving the equation $x = g(x)$ to be equivalent to solving the pair of simultaneous equations

$$y = x, \quad \text{and} \quad y = g(x),$$

so that we plot the two graphs – the first is simply a straight line of gradient 1 passing through the origin, and the problem is to determine where the second graph crosses it.

The iteration procedure for different cases is shown in Figure 4-3. In Case A, the function has a gradient which is greater than 1, and although we started near the solution at point 0, we were taken away from it. In the next Case B the function still has a negative slope, but it is less than 1 in magnitude and it can be seen how the solution spirals in to converge. The last Case C is for a positive slope which is less than 1, and the process converges. These figures



Case A: $|g'_A(x)| > 1$, unstable Cases B and C: $\left|g'_{B,C}(x)\right| < 1$, stable

Figure 4-3. Unstable and stable behaviour of direct iteration scheme for different gradients of the function

demonstrate the condition for convergence, that the direct iteration scheme $x_{n+1} = g(x_n)$ is stable if the gradient of the curve is less than one in magnitude in the vicinity of the root being sought, that is,

$$\left|\frac{dg}{dx}(x_n)\right| < 1 \text{ for stability.}$$

4.2.5 The bisection method

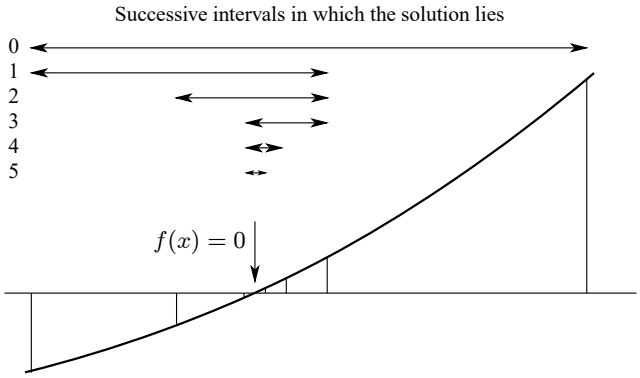


Figure 4-4. Bisection method, showing the successive halvings of the interval in which a solution is known to lie

This is a non-gradient method which uses a simple algorithm which can handle the most difficult of functions. The method will *always* converge to one solution provided a range which contains the solution is selected at the start. It proceeds by halving the range, then testing whether the solution is in the left or right half, and then repeating, halving the interval at each step. It is similar to a dictionary game, where one player selects a word, and another player has to guess the word by taking half the dictionary, asking if the word is in the first or second half of the dictionary, then in which half of that half and so on. If there are N words, then it should be possible to get the word in m tries, where $N = 2^m$, or $m = \log_2 N$. In the case of the a typical dictionary, with something like 80000 words, $m = \log_2 80000 \approx 16$, *i.e.* 16 tries.

The method is to bracket the solution initially, *i.e.* to find a_0 and b_0 such that the solution lies between the two, and then calculate the mid-point $x_m = (a_0 + b_0)/2$ and by the sign of $f(x_m)$, determine whether the solution is in the left or right half, and then to repeat until the interval is small enough. The algorithm can be written, where a and b are the left and right ends of the initial interval and ε is the required accuracy:

$f_a = f(a),\ f_b = f(b)$	Calculate the values of the function at the ends
if($f_a \times f_b > 0$) exit and choose another a or b	The function seems not to change sign in the interval
while ($b - a > \varepsilon$)	Repeat until the interval is small enough
$x_m = (a + b)/2$	Calculate the mid-point
$f_m = f(x_m)$	Calculate the function there
if ($f_m \times f_a < 0$)	Does the sign of the function change in the left half?
$b = x_m$	If so, reset the right boundary to the mid-point
else	
$a = x_m, f_a = f_m$	Otherwise set the left boundary to the mid-point
endif	

This method is not so easily programmed in a spreadsheet itself. Rather it is easily programmed in Visual Basic, the programming language which is available behind the spreadsheet. This can be accessed any time by using Alt-F11 in Excel Workbook 2-SOLUTION.XLS.

4.3 Excel Solver

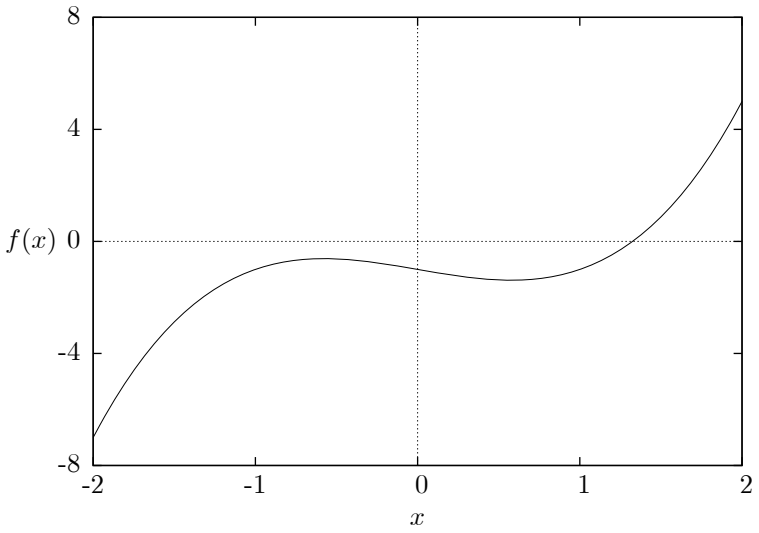


Figure 4-5. Cubic function $f(x) = x^3 - x - 1$, with difficult behaviour for root-finding

Workbook 2-SOLUTION.XLS shows the use of Excel Solver to obtain the solution of a single equation in a single unknown for the example used above, $f(x) = x^3 - x - 1$, to give an introduction to its use and to help us feel familiar with it. This simple cubic actually has a very nasty property, in that it has one solution, but the nature of the function, with a turning point where there is no solution is such as to throw off both Newton's method and Excel Solver, unless one starts sufficiently close to the solution. Plotting the function is always a good idea. Generally, however, using Excel Solver seems to be the best and simplest way of solving equations, most of which have rather better properties than the example given.

5. Systems of equations

5.1 Solution by optimisation

Solver can be programmed to solve a much wider family of problems. It was originally designed for optimisation problems, where one has to find values of a number of different parameters such that some quantity is minimised, usually the sum of errors of a number of equations. With this tool one can find such optimal solutions, or solutions of one or many equations, even if they are nonlinear. All one has to do is to formulate the problem as one or more equations.

Let our system of equations be $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{f}$ and $\mathbf{x}$ are vectors such that

$$\mathbf{f}(\mathbf{x}) = [f_i(x_j,\, j = 1, \ldots, N) = 0,\, i = 1, \ldots, M]\,,$$

namely we have a system of M equations f_i in N unknowns x_j. If the equations are linear, a number of linear algebra methods such as matrix inversion, *etc.* are possible, but the problems are usually more easily soluble with *Solver*. This even applies to the more general case, where the equations are nonlinear. In this process, ε, the sum of the squares of the errors for all the equations are evaluated:

$$\varepsilon = \sum_{i=1}^{M} f_i^2(x_j),$$

and the values of the x_i found such that ε will be a minimum. If $M = N$ it should be possible to find a solution such that $\varepsilon = 0$ and the equations are solved. If there are fewer variables than equations, $N < M$, then it will not be possible to solve the equations, but minimising ε produces a useful solution anyway. This is particularly so in the case of finding approximating functions.In the formulations of the present problems, we have to write all the equations with all terms taken over to one side, and then to implement an optimisation procedure such as *Solver*. It does not matter how complicated the equations are – it is just necessary to be able to evaluate them. Most methods for minimising ε are gradient methods, such that in the N-dimensional solution space, a direction is determined, and the minimum of the error function in that direction found, at the minimum a new direction found, usually orthogonal to the previous one, and so on. For two dimensions it can be shown as finding the minimum value of a surface, as in Figure 5-1; the three dimensional case can be imagined as finding the hottest point in a room by successively travelling in a number of different directions, finding the maximum in each, and then setting off at right angles, and so on. In either case the gradients are not obtained analytically but numerically by evaluating the functions at different values.

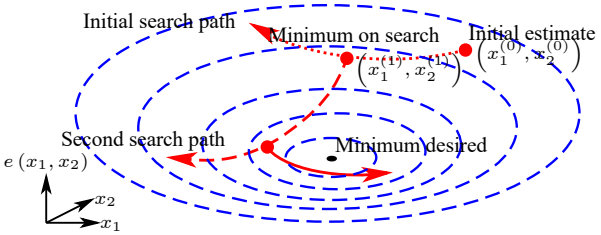


Figure 5-1. Typical search procedure to find a minimum – here a function of two variables

This procedure usually means that we have simply to write down the equations, with initial trial values of the unknowns, and call the optimising routine. It is very general and powerful, and we can usually write down the equations in the simplest form, so that no algebraic manipulation is necessary or favourable. It can be seen that it works simply, and no attention has to be paid to the solution process.

5.2 Systems of linear equations

Very commonly throughout engineering one encounters systems of linear simultaneous equations. These can be solved by direct methods such as *Gaussian elimination*, or *LU decomposition* into upper and lower triangular form. Computationally, a very popular method for 20 years has been *Singular value decomposition*, which also works well on equations which are poorly conditioned, such that the solution is very sensitive to numerical accuracy. Of course, other procedures for solving systems of linear equations are matrix methods, but these are often not as computationally efficient as the previously mentioned ones.

Solver, however, can be used very simply to solve linear equations as well. An example is given in the Workbook 3-SOLVER.XLS. This contains some simple examples in Worksheet LINEAR EQUATIONS. In the first example we compare the use of typing out the equation with matrix notation.

A well-conditioned system – two approaches Consider the system of equations

$$\begin{aligned} x + 0 \times y - z &= 1 \\ y &= 1 \\ x + 0 \times y + z &= 1 \end{aligned}$$

1. Firstly, the equations $x-z-1=0$, $y-1=0$, and $x+z-1=0$ are typed and solved: $y=1$, now eliminate z from the first and third equations by adding to give $2x=2$, and back substitute $x=1$ to give $z=0$. This system is well-conditioned, in that the solutions are not sensitive to numerical operations. Sometimes, when two or more of the three intersecting surfaces are very similar, the actual intersection point is poorly defined and the system is not well-conditioned.
2. Whereas the writing and solution of the three equations was simple and obvious, it is sometimes easier to write the equations as a matrix equation

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Solver can be used to actually solve the system of equations or to find the solution such that the sum of the squares of the errors in each equation is a minimum. In this case *Solver* obtains the solution to six figures $\begin{bmatrix} 1.000000 & 1.000000 & 0.000000 \end{bmatrix}^{\mathrm{T}}$.

A singular system: in this case we consider

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix},$$

where the last row is simply -1 times the first row. Solving this is equivalent to finding where one plane crosses two planes which are coincident so that there is an infinity of solutions along a line where the planes intersect, so there is no unique solution. *Solver* found the solution $\begin{bmatrix} 0.49999945 & 0.99999999 & -0.50000051 \end{bmatrix}^{\mathrm{T}}$, and several others besides, just depending where the initial solution was.

A poorly-conditioned system: in this case the system of equations is such that the system is not exactly singular, but almost so, such that solutions depend very much on the numerical accuracy. Consider the example which is simple but is notoriously poorly-conditioned:

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \theta \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

In this case *Solver* had great trouble, and found a solution but with quite a large error, and it was not possible to refine this because of the poorly-conditioned nature of the system. It was interesting, however, that the matrix inverse facility provided as part of *Excel* gave the correct solution correct to 16 decimal places, $\begin{bmatrix} -4 & 60 & -180 & 140 \end{bmatrix}^{\mathrm{T}}$.

5.3 Nonlinear equations

In many physical problems, however, the system is not poorly conditioned, and *Solver* can obtain highly accurate solutions. As it uses a scheme for minimising the sum of errors of a system of equations it does not matter whether the system is linear or nonlinear, which the next example in the spreadsheet shows.

Example 5.1 Find where the parabola $y=x^2$ crosses the circle $x^2+y^2=2^2$.

In *Solver* we have two variable cells for x and y, plus a cell for the sum of the squares of the errors of the equations. However, it is usually clearer to set up a cell for the evaluation of each of the equations plus one cell where the sum of the squares is evaluated, and this is the cell that is minimised or solved. A single command can be used: SUMSQ(...). Here is the initial setup on Worksheet NONLINEAR SYSTEMS in Workbook 3-SOLVER.XLS, with an initial estimate of $x=1$, $y=1$:

		Column C	Column D
		Solution	
		x	y
	Row 5	1	1
		Evaluate functions, each of which should be zero for a solution	
Mathematics:		Excel contents	Result
$y - x^2$	Row 9	=C5-B5*B5	0
$x^2 + y^2 - 2^2$	Row 10	= B5*B5+C5*C5-4	-2
Sum of squares:	Row 11	=SUMSQ(D9:D10)	4

Now, running *Solver* with the Target Cell D11, Changeable Cells B5:C5, and seeking a *Solution* or *Minimising the sum of the errors* the method converges, $x = 1.24962...$, $y = 1.56155....$ Unpleasantly, with the initial solution $x = 0$, $y = 0$, it does not converge – so one has to be sufficiently close to a solution to converge to it. For problems that are not so nonlinear, this is not such a problem. Note this did not find another solution which was the negative of both those numbers, unless an initial estimate of the solution closer to that one was assumed. If it finds one solution, it provides no information as to how many exist or where the other ones are.

Exercise: consider the problem of finding the intersection of the function $y = x^{1/3}$ with the circle $x^2 + y^2 = 1$. *Solver* can find this very accurately as $x = 0.563623889$ and $y = 0.826031289$.

6. Interpolation of data

An important problem is the determination of a mathematical function which passes through a number of data points. This is equivalent to the solution of a number of equations, equal to the number of data points and to the number of coefficients in the interpolating function. Usually such problems are linear, for example, determining the coefficients a_i in the polynomial $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$.

6.1 The nature of the problem

Consider the following pairs of x, y points: $(0, 0)$, $(1, 1)$, and $(2, 4)$. With three data points, this suggests using a quadratic equation $y = ax^2 + bx + c$, where we can solve the equation at three points to give the values of a, b, and c. We obtain the three equations

$$\begin{aligned} 0 &= c \\ 1 &= a + b + c \\ 4 &= 4a + 2b + c \end{aligned}$$

These are easily solved: $a = 1$, $b = 0$, and $c = 0$, and *Solver* can be used for such problems to good effect. Figure 6-1, shows how the function that interpolates the data points passes through each. It is capable of analytical differentiation or integration.. If the data points are irrgular, then we cannot use *interpolation*, but rather *approximation*, as described in §7 below.

6.2 Scaling of the dependent variable

There are some subtleties in the numerical operations which can destroy the accuracy of interpolation and approximation. It is always a good idea to scale the independent variable back to the range $[0, 1]$ or even better $[-1, 1]$, rather than in absolute terms, *especially in civil engineering problems* where the values of x might be huge, corresponding to distances along a road, railway or river. For example, consider the problem of finding the interpolating quadratic which passes through three points on a road curve near the $20\,\mathrm{km}$ mark: $(20200, 0)$, $(20300, 100)$, and $(20400, 150)$. Assuming a function of the form $y = a_0 + a_1 x + a_2 x^2$ and using *Solver* gave terrible results, as would almost any other method. If on the other hand, the results are scaled as above, then *Solver* obtains the results to 6 figures: $a_0 = 99.99996$, $a_1 = 74.99995$, and $a_2 = -25.00000$. This is in Worksheet INTERPOLATION in Workbook 3-SOLVER.XLS. A description of such problems has been given by Fenton (1994). In particular the use of Divided Differences and Newton Interpolation described there is very powerful.

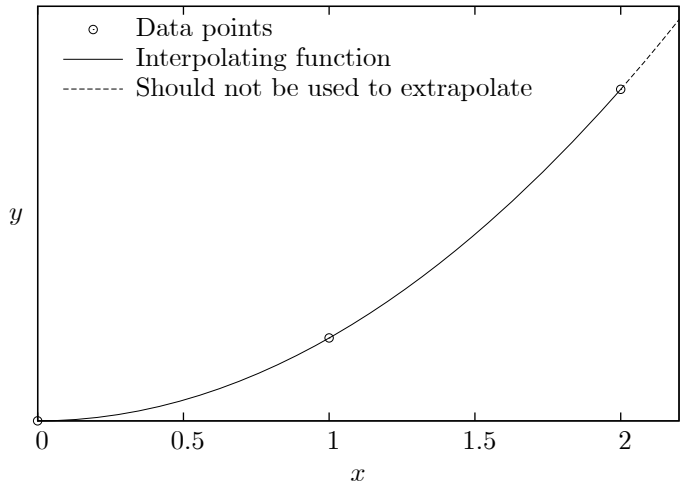


Figure 6-1. Interpolation of three data points by a second-degree polynomial

Also powerful is the use of orthogonal functions, to be described below in §7.2.

7. Approximation of data

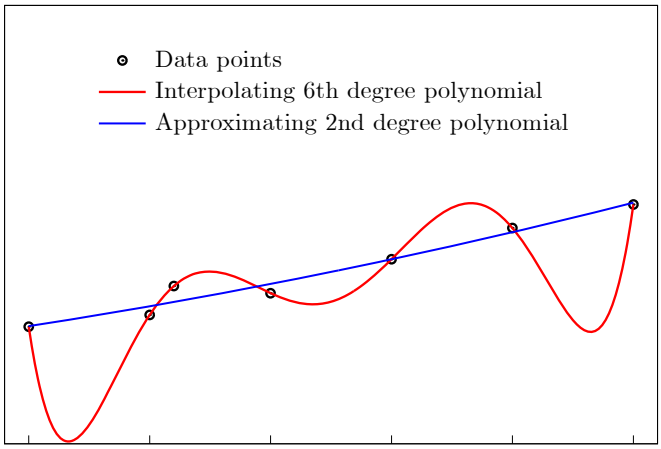


Figure 7-1. Irregular data points, showing the superiority of approximation to interpolation

Interpolation can only be used for smooth regular data points, such as from the numerical solution of equations. For experimental data, with irregularities, it is completely inappropriate, such as the figure here shows. It is much better to approximate the data by functions that contain fewer degrees of freedom, such as lower-degree polynomials, such that they do not pass through every point. They do not *interpolate*, they *approximate*.

If we use *Solver*, almost any method of approximation becomes possible. Mostly, this will consist of a relatively low-order polynomial. One advantage of using optimisation methods, as done by *Solver* is that *nonlinear* methods can be used. As an example, consider the function

$$y = a_0 + a_1 e^{a_2 X},$$

where in this case, finding the a_0, a_1, and a_2 is a nonlinear problem. We have used the transformed variable X again, although it is often not necessary. In the accompanying *Excel* worksheet 3-SOLVER.XLS such a problem is solved. It is interesting that analytically nonlinear problems are very difficult indeed, but for optimisation methods and *Solver* they are no more difficult than linear problems.

This can be used in a very similar way to interpolation of data as described above in Section 6, but where the number of degrees of freedom (the number of coefficients of the approximating function) are rather fewer than the

number of data points. In this case *Excel* cannot solve the equations exactly, as there are not enough coefficients, but it can determine the coefficients such that the sum of the squares of the errors is minimised.

7.1 Curve fitting by Excel

There is a facility in *Excel* by which "trendlines" can be very simply added to sets of experimental points, which are none other than approximating data by a relatively low-degree function as we described immediately above. Generally the program behind it is very robust, and certainly uses the subtraction of a mid-value of x as we described in Section 6. One right clicks on one of the plotted points and is given the option of adding what *Excel* calls a "trendline", and one is given the option of several different types of function. It can only do linear problems, which is not a big disadvantage, except, for example, if we wanted the exponential fit shown above. If one needs the actual function which *Excel* has obtained, that can be displayed on the figure as well. However, it has a flaw, in that however robustly it calculates the approximating function internally, it displays it in expanded form, and sometimes with few significant digits in the coefficients, so that round-off errors might be huge. For example, it might calculate the function obtained above internally as $y = 100.001 + 0.75\,(x - 20300) - 0.0025001\,(x - 20300)^2$, but when asked for the function it expands it as, for example here, it should expand to $-1.\,045\,391\,208\times10^6 + 102.\,254\,06\,x - 0.002\,500\,1\,x^2$, but unless specifically required it displays only few figures, and in the above case it actually just displayed $y = -1E + 06 + 102.25\,x - 0.0025\,x^2$, which gives nothing like the desired results. One can change the number of digits by right-clicking on the formula and changing the Number of the Format of the line, but it is better to do the approximation oneself, in a form such that one has control of it. To do the approximation oneself, simply use *Solver* as described above.

The above example has been programmed in 4-FITTING.XLS.

7.2 The use of orthogonal functions

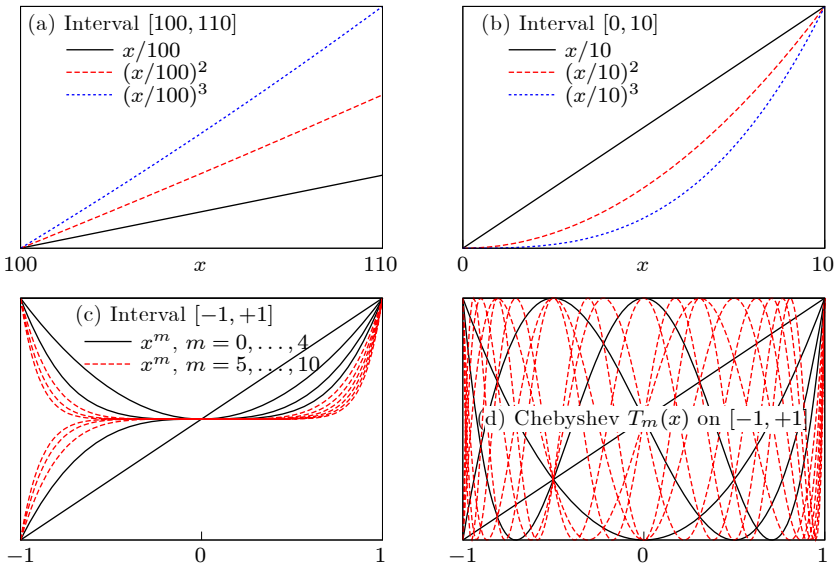


Figure 7-2. Comparison of the first three monomials on the intervals $[100, 110]$ and $[0, 10]$ and the first ten monomials and Chebyshev polynomials on $[-1, +1]$

Consider the general approximating function a linear combination of different functions $p_m(x)$:

$$p\,(x) = \sum_{m=0}^{M} a_m p_m(x) = a_0 p_0(x) + a_1 p_1(x) + \ldots + a_M p_M(x)\,. \tag{7.1}$$

The simplest approximating function is a polynomial, with the individual contributions $p_m(x)$ to be the monomials x^m. Figure 7-2 shows different aspects of the approximation:

Fig. 7-2(a) shows the first three monomials (after $p_0(x) = x^0 = 1$) on the interval $[100, 110]$, scaled for plotting purposes. It can be seen that all three look almost like straight lines on such an interval. If the function to be

approximated had any curvature, those functions would have to work very hard, in the form of large oscillating values of the coefficients a_m.

Fig. 7-2(b) shows what happens if we were to transform the interval to $[0, 10]$, where the basis functions now look less similar, and as a result they could better approximate a more general function with curvature. That holds only to a limited extent, however, as seen in the right half of the next figure.

Fig. 7-2(c) shows a better solution where the interval of approximation has been transformed to $[-1, +1]$, and the first 4 or 5 monomials are really quite different, however for larger values of m they all start to look similar to each other, making general approximation more difficult. That would be especially so if we were to consider just the interval $[0, 1]$ in the right half of the figure.

Fig. 7-2(d) shows the best solution of all, to use orthogonal Chebyshev polynomials for x also scaled to $[-1, +1]$: $T_m(x) = \cos(m \arccos(x))$, $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$, *etc*. They all show behaviour quite different from each other, and hence have a much better ability to describe arbitrarily varying quantities.

8. Differentiation and integration

The calculation of derivatives and integrals from numerical data is very important. As integration is a smoothing operation, its results are very much more robust than differentiation, which is well-known to be sensitive to "noisy" data.

8.1 Differentiation

Consider the Taylor expansion for a function $f(x)$ about a point x:

$$f(x+h) = f(x) + h\frac{df}{dx}(x) + \frac{h^2}{2!}\frac{d^2f}{dx^2}(x) + \ldots . \tag{8.1}$$

We can re-arrange this, neglecting the term in h^2 to give

$$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x)}{h} + O(h), \tag{8.2}$$

which is a *forward difference approximation*. This states that to obtain the value of a derivative, calculate the value of the function at a nearby point $x + h$, subtract the value at x and divide by h. The error is shown by the term $O(h)$ which means that it is directly proportional to the distance h. Results are shown by the dotted line on Figure 8-1, where it can be seen that the gradient of this approximation is not a particularly accurate approximation to the gradient of the tangent at x, that which is required.

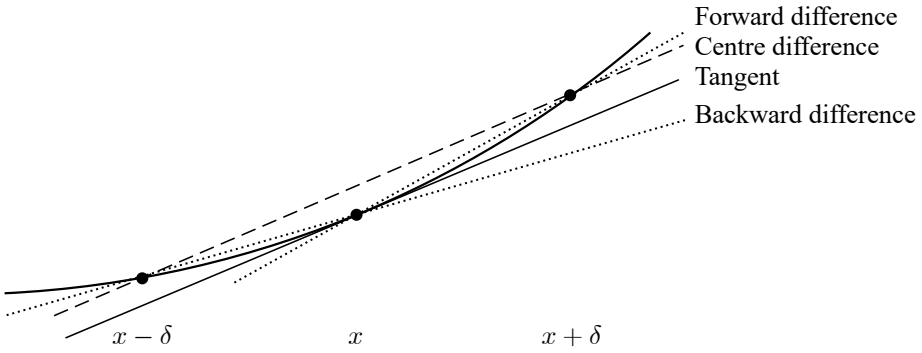


Figure 8-1. Approximations to the slope of the curve at x in terms of function values at $x - \delta$, x, and $x + \delta$

Now consider equation (8.1), which we write, replacing h by $-h$:

$$f(x-h) = f(x) - h\frac{df}{dx}(x) + \frac{h^2}{2!}\frac{d^2f}{dx^2}(x) + \ldots . \tag{8.3}$$

Now if we subtract equation 8.3 from equation 8.1:

$$f(x+h) - f(x-h) = 2h\frac{df}{dx}(x) + O(h^3),$$

and re-arranging gives

$$\frac{df}{dx}(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2). \tag{8.4}$$

Results are shown by the dashed line on Figure 8-1, and it can be seen that this *centred difference approximation* is rather more accurate, as shown mathematically by the error term $O(h^2)$, which means that if the step size is halved the error will be one quarter.

These approximations, discretising in both space *and* time are widely used for numerical solution of partial differential equations throughout engineering.

8.2 Integration

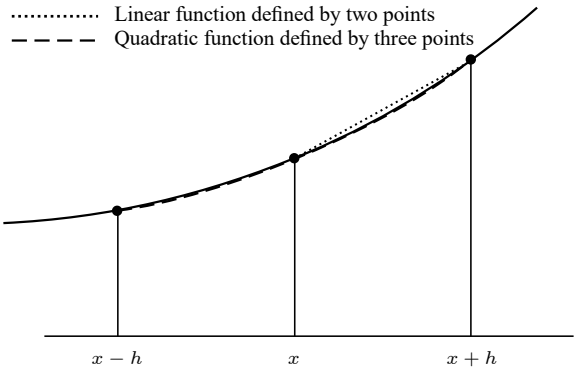


Figure 8-2. Interpolating functions for integration in terms of function values at $x-h$, x, and $x+h$

Figure 8-2 shows two different approximations for integrating the function by finding the area between the function and the x axis. The first is the *trapezoidal rule*, which consists of taking the function value at two points, fitting a straight line between them (the dotted line on the figure) and finding the area of that trapezium. It is

$$\int_x^{x+h} f(x)\,dx \approx \frac{h}{2}\left(f(x) + f(x+h)\right). \tag{Trapezoidal rule}$$

On the other hand, if three points are used, a quadratic function can be used to interpolate the three, shown dashed in the figure, and the area under the quadratic obtained. The result is *Simpson's rule*

$$\int_{x-h}^{x+h} f(x)\,dx \approx \frac{h}{3}\left(f(x-h) + 4f(x) + f(x+h)\right). \tag{Simpson's rule}$$

In fact this result is also exact for a cubic function, and the accuracy of Simpson's rule goes like $O(h^3)$, which is surprisingly accurate.

Figure 8-3. Typical subdivision for numerical integration

Of course in practice we need to find the area over a finite range, and so we use compound versions of these, as shown in Figure 8-3. The trapezoidal rule can be written

$$\int_{x_0}^{x_N} f(x)\,dx \approx \frac{h}{2}\left(f(x_0) + f(x_N) + 2\sum_{i=1}^{n-1} f(x_i)\right). \tag{Compound trapezoidal rule}$$

The compound Simpson's rule can only be applied for even numbers of panels (odd numbers of points, from 0 to

N, such that N is an even number). It is

$$\int_{x_0}^{x_N} f(x)\,dx \approx \frac{h}{3}\left(f(x_0)+f(x_N)+4\sum_{i=1,\text{ odd}}^{N-1} f(x_i)+2\sum_{i=2,\text{ even}}^{N-2} f(x_i)\right). \qquad \text{(Compound Simpson's rule)}$$

9. Numerical solution of ordinary differential equations

Consider the problem of solving the first-order differential equation

$$\frac{dy}{dt} = f(t,y) \tag{9.1}$$

such that we know the rate of change of a quantity as a function of time and the quantity itself. The problem is to obtain a solution for $y(t)$, possibly in a general form, but in practical problems usually where an initial condition $y(t_0) = y_0$ is known. In many problems this can be solved analytically, for example in the case of radioactive decay, where the rate of change of the quantity of material is proportional to the quantity present,

$$\frac{dy}{dt} = -k\,y, \tag{9.2}$$

where k is the constant of proportionality. In this case we can separate the variables on each side of the equation and integrate:

$$\begin{aligned}\int \frac{dy}{y} &= -k\int dt,\\ \ln y &= -kt + C,\end{aligned}$$

and substituting in the *initial condition* $y(t_0) = y_0$ to determine the constant of integration C we find $C = \ln y_0 + kt_0$ and the solution can be written

$$y = y_0 e^{-k(t-t_0)},$$

showing the well-known exponential decay.

In general such an integration is not possible, for example, the differential equation $dy/dt = -y^2 - t$, and we have to solve the equation numerically, which usually means obtaining a sequence of numerical values $y_1, y_2, \ldots$ corresponding to specified values $t_1, t_2, \ldots$.

9.1 Euler's method

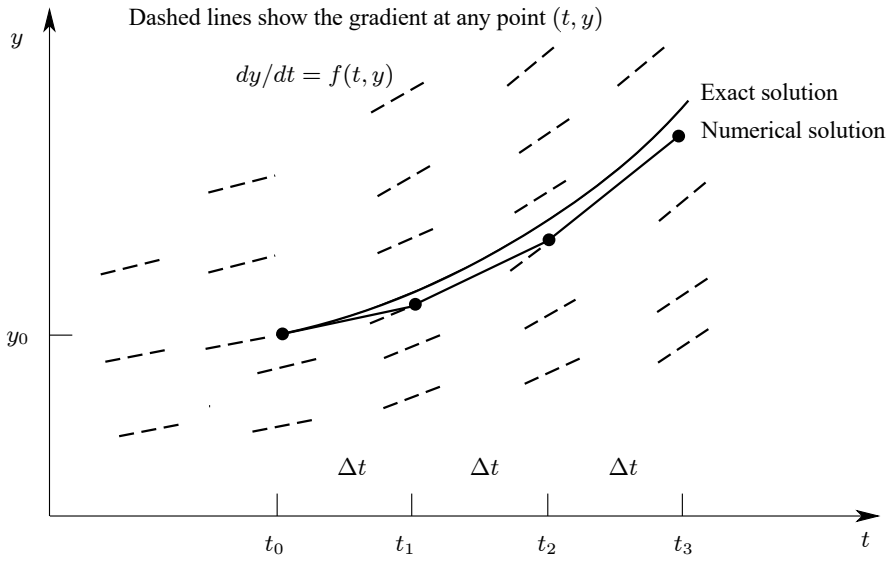


Figure 9-1. The nature of the numerical solution of a differential equation

Consider the plot of the gradient $dy/dt = f(t, y)$ as shown by the dashed lines on Figure 9-1, such that for any t and y the value of the derivative can be calculated. Numerical solution begins at the given initial point (t_0, y_0), and the problem is to advance the solution in time along a curve across the plane shown in the figure. The simplest (Euler[1]) scheme to advance the solution from one computational solution point (t_i, y_i) to another (t_{i+1}, y_{i+1}) is to consider a step Δt in t such that $t_{i+1} = t_i + \Delta t$, for all i, and to approximate the derivative in equation (9.1) by a simple forward difference approximation

$$\frac{\Delta y}{\Delta t} \approx f\left(t, y\right),$$

such that considering points i and $i + 1$ we have

$$\frac{y_{i+1} - y_i}{\Delta t} \approx f\left(t_i, y_i\right),$$

such as shown on Figure 9-1. This gives the explicit expression for the new value of y_{i+1}:

$$y_{i+1} \approx y_i + \Delta t\, f\left(t_i, y_i\right), \tag{9.3}$$

and solution proceeds as shown in the figure, in general deviating somewhat from the exact solution. This is the simplest but least accurate of all methods.

If we were to examine the approximation of the scheme in equation (9.3) we would find that the neglected terms are of a magnitude proportional to the square of the time step, $(\Delta t)^2$. This is called the truncation error, because we have truncated a series approximation to the differential equation, here after the very first term. To carry out the calculations to a certain point in time t, the number of time steps necessary is $t/\Delta t$, and so multiplying the truncation error at each step by the number of steps we conclude that the *global error* at a particular time varies like

$$\text{Error} \sim \frac{1}{\Delta t} \times (\Delta t)^2 \sim \Delta t.$$

We have used the symbol "$\sim$" or tilde to show how the error varies with Δt in the limit of small Δt. As errors here go like this first power of the time step $(\Delta t)^1$, we call this a *first-order* scheme. If we were to halve the time step, the error at a certain time would also be halved.

Example 9.1 Solve the equation for radioactive decay, equation (9.2) using Euler's method, with $k = 2$, with initial condition $t = 0$, $y = 1$, using (a) $\Delta t = 0.2$ and (b) $\Delta t = 0.1$.

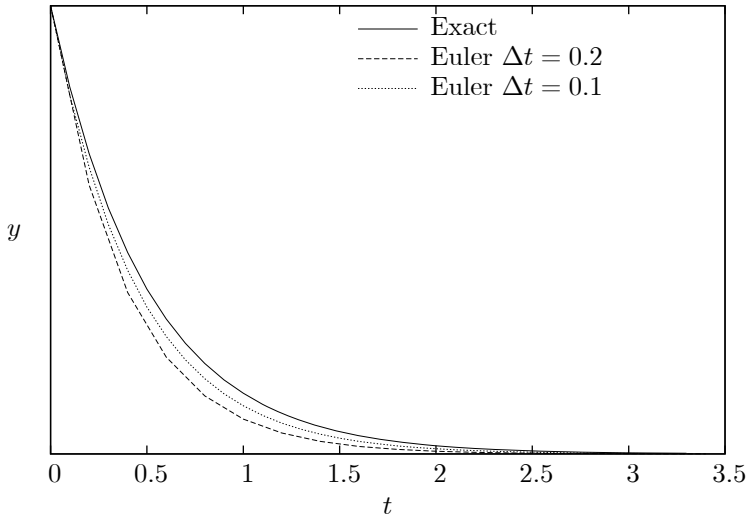


Figure 9-2. Comparison of results from Euler's method

This was done in spreadsheet 5-DIFF-EQNS.XLS and the results are shown in the figure. It can be seen that as the computational step was halved, then the error at any point was also roughly halved. This is characteristic of this first-order method. We note that the results give a rough idea of the scale of the solution, but are not quite accurate enough to be considered satisfactory.

[1] Leonhard Euler (1707-1783), Swiss mathematician and physicist, remarkable genius.

As a general rule in computations it is good practice to demonstrate that the numerical solution is a sufficiently accurate one by considering smaller steps, to show that the process has converged. In most problems a relatively slow convergence to the exact solution as in the above example is not acceptable. A simple way of overcoming the problem of insufficient accuracy is to take successively smaller time steps, until the process has converged to the desired accuracy. To gain better accuracy most simply, one can just take smaller steps Δt. However higher accuracy methods such as Richardson extrapolation and Heun's method are usually considered better, and which will be described below.

9.2 Higher-order schemes

9.2.1 Heun's method

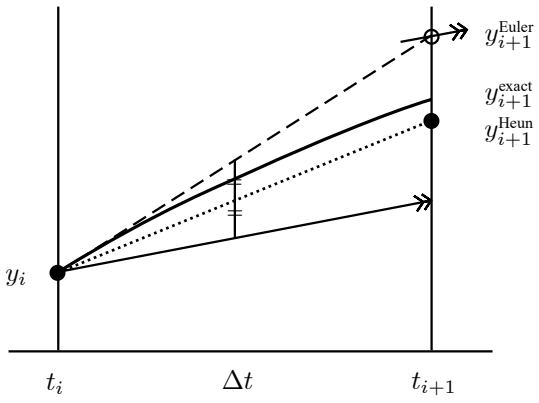


Figure 9-3. One step of the Heun scheme

In Euler's method there was no attempt made to include information from the point t_{i+1} at which the solution is being obtained. Heun's method uses Euler's method but uses the result y^*_{i+1} to calculate the gradient there and then, whereas Euler's method just uses $f(t_i, y_i)$ for the gradient over the interval, Heun's method uses the mean of that value and the approximately calculated value at t_{i+1}. The scheme can be written (*cf.* equation 9.3):

$$y^*_{i+1} = y_i + \Delta t\, f(t_i, y_i), \tag{9.4}$$

$$y_{i+1} \approx y_i + \frac{\Delta t}{2}\left(f(t_i, y_i) + f\left(t_{i+1}, y^*_{i+1}\right)\right). \tag{9.5}$$

This is shown graphically on Figure 9-3. The dotted line has a gradient $f(t_i, y_i)$ giving the point $\left(t_{i+1}, y^*_{i+1}\right)$ marked by an open circle, corresponding to equation (9.4), Euler's method. At this point the gradient is calculated from the differential equation, giving $f\left(t_{i+1}, y^*_{i+1}\right)$, and the mean of the two gradients calculated and used in equation (9.5). On Figure 9-3 this has been shown by transferring a dashed line of gradient $f\left(t_{i+1}, y^*_{i+1}\right)$ back to the original point and then drawing a dotted line which is the mean of the two gradients and then using this to calculate the y_{i+1}. This method has a local error of $O\left(\Delta t^3\right)$, and hence a global error of $O\left(\Delta t^2\right)$, more accurate than Euler's method.

Results for this are shown on spreadsheet 5-DIFF-EQNS.XLS.

Example 9.2 Repeat example 0 using Heun's method.

When executed using a spreadsheet and plotted, the results were indistinguishable from the exact solution. The value obtained at $t = 1$ for comparison with the results from Richardson's method was $y(1) = 0.1374$, compared with the exact result of 0.1353, and an error of 1.6%, very much better than Euler's method, but comparable with the results from the Richardson enhancement.

9.2.2 Predictor-corrector method – Trapezoidal method

This is simply an iteration of the last method, whereby the steps in equations (9.4) and (9.5) are repeated several times, at each stage setting y^*_{i+1} equal to the updated value of y_{i+1}. This gives a slightly more accurate method. On Figure 9-3 this would correspond to calculating the gradient at the updated point y_{i+1}, transferring this back

to the original point again, calculating the mean of this and that back at the original point and re-calculating the estimated y_{i+1}, so that the sequence of approximations on Figure 9-3 would be successive points at t_{i+1} gradually converging to the best solution obtainable with this method.

Example 9.3 Repeat the above examples but iterating Heun's procedure at each step

The results were, as expected, even more accurate than in Example 0, and indistinguishable from the exact solution when plotted. The value obtained at $t = 1$ for comparison with the results in Example 0 was $y(1) = 0.1344$, compared with the exact result of 0.1353, and an error of -0.7%, more accurate than, but comparable with, the results from the Richardson enhancement and Heun's method.

9.2.3 Runge-Kutta methods

These are a family of methods which are capable of high accuracy. Heun's method is actually the second-order component of the family. A popular version is the fourth-order member, for which reference can be made to any book on numerical methods. They require the ability to calculate the right side of the differential equation at a number of points intermediate between t_i and t_{i+1}.

9.3 Higher-order differential equations

There are many problems where second or higher-order differential equations have to be solved. The procedure in this case is to convert the nth-order differential equation to n separate first order equations by introducing the derivatives as extra variables. Then, the procedures considered above can be used.

Consider the second-order differential equation with two initial conditions,

$$\frac{d^2y}{dt^2} = F\left(t, y, \frac{dy}{dt}\right) \qquad \text{with} \qquad y(t_0) = y_0,\ \frac{dy}{dt}(t_0) = y_0'.$$

Introducing the variable $u = dy/dt$, the system becomes

$$\left.\begin{array}{l} \frac{du}{dt} = F\left(t, y, u\right), \\ \frac{dy}{dt} = u \end{array}\right\} \text{with } y(t_0) = y_0 \text{ and } u(t_0) = y_0'.$$

Euler's equation (9.3) applied to each differential equation becomes, with $y(t_0) = y_0$ and $u(t_0) = y_0'$,

$$\begin{array}{l} u_{i+1} = u_i + \Delta t\, F\left(t_i, y_i, u_i\right), \\ y_{i+1} = y_i + \Delta t\, u_i \end{array} \tag{9.6}$$

We could continue the process to differential equations of arbitrary order, $v = du/dt = d^2y/dt^2$, hence $d^3y/dt^3 = dv/dt$, *etc*.

Example 9.4 Consider a vibrating mass or pendulum with equation

$$\frac{d^2y}{dt^2} + y = 0,$$

with initial conditions $y(0) = 0$ and $y'(0) = 1$.It is easily shown that the analytical solution is

$$y = \sin t,$$

but here we will solve it numerically as far as $t = \pi$ using steps of $\Delta t = 0.1$.

Letting $u = dy/dt$, the equations become

$$\left[\begin{array}{l} \frac{du}{dt} = -y \\ \frac{dy}{dt} = u \end{array}\right]$$

and Euler's method (9.6) becomes

$$\left[\begin{array}{l} u_{i+1} = u_i - \Delta t\, y_i \\ y_{i+1} = y_i + \Delta t\, u_i \end{array}\right]$$

Results are shown in Figure 9-4. Again it can be seen that Euler's method is not accurate, although that could easily be remedied by taking shorter steps or using Richardson extrapolation. Heun's method is much more accurate for

this problem too.

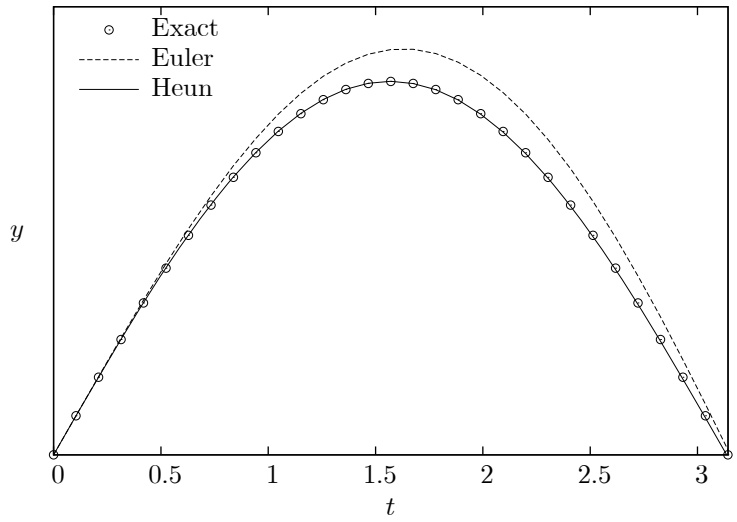


Figure 9-4. Comparison of results from solving a 2nd order differential equation

9.4 Boundary value problems

All the previous problems and methods studied have been *Initial Value Problems* (IVPs), where all conditions are known at a particular initial point. In practice there are many problems which have boundary conditions not just at one point. Examples including vibrating beams and many others. These problems, known as *Boundary Value Problems* (BVPs), are rather more difficult to solve.

9.4.1 Shooting methods

The traditional way of solving such a problem is to turn it into an initial value problem and solve it iteratively, that is, try a number of different conditions at a single point and vary them until the remaining condition(s) are satisfied. This method is rather more difficult to implement on a spreadsheet.

9.4.2 A spectral method developed for this course

With the ability of *Excel Solver* to handle systems of equations, the prospect now becomes a possibility of routinely solving such problems by using series of known functions (a *spectral* representation), where all that is necessary to do is to find the coefficients of those functions. Let

$$y(t) = a_0\phi_0(t) + a_1\phi_1(t) + a_2\phi_2(t) + \ldots, \tag{9.7}$$

where we will choose the functions $\phi_i(t)$ before we begin. Let us suppose we have a general differential equation

$$D\left(t, y, \frac{dy}{dt}, \frac{d^2y}{dt^2}, \ldots\right) = 0, \tag{9.8}$$

where $D(\ldots)$ is a function of all the variables shown. If we substitute $y(t)$ from equation (9.7) into this, the result is a function of all the unknown coefficients, which is possibly nonlinear. We can find the optimal values of those coefficients by minimising the sum of the squared errors of equation (9.8) at a number of points, for which *Excel Solver* is a very powerful tool.

A useful aspect of this method is that, instead of a table of computed function values, the end result is a table of coefficients such that we can evaluate a function at any point, which is especially useful for plotting purposes.

Example 9.5 We start with a problem where the solution is the same as the previous example, but where the problem is set up as a rather more difficult boundary value problem: consider the second-order differential equa-

tion

$$\frac{d^2y}{dt^2} + y = 0, \tag{9.9}$$

with boundary conditions $y(0) = 0$ and $y(\pi/2) = 1$, which has the solution $y = \sin t$.

We assume a fifth-degree polynomial in t:

$$\begin{aligned} y(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \\ \frac{d^2y}{dt^2} &= 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3. \end{aligned}$$

In fact, this special (linear) problem could be solved in a true spectral sense by requiring the error in the differential equation to be zero for all t and solving the resulting system of equations obtained by requiring the coefficient of each power of t to be zero, however the results are relatively poor (although it is easily shown that $a_0 = a_2 = a_4 = 0$).

Instead we prefer to develop a more general method. At the boundary points we write the two conditions in terms of errors ε_0 and ε_N:

$$\begin{aligned} \varepsilon_0 &= y(t_0) - 0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5, \qquad \text{and} \\ \varepsilon_N &= y(t_N) - 1. = a_N + a_1 t_N + a_2 t_N^2 + a_3 t_N^3 + a_4 t_N^4 + a_5 t_N^5 - 1. \end{aligned}$$

For N points t_i intermediate between the end points, substituting into the differential equation (9.9) gives an expression for the error at each point ε_i:

$$\varepsilon_i = (2a_2 + 6a_3 t_i + \ldots) + \left(a_0 + a_1 t_i + a_2 t_i^2 + \ldots\right).$$

Defining the total error of our approximation $E = \sum_{i=0}^{N} w_i \, \varepsilon_i^2$,where the w_i are weights which generally will be all equal to 1, but additional weight might be given to the boundary conditions to make sure that they are satisfied more accurately.

We use *Excel Solver* to minimise the value of the total error E by optimisation, varying the values of the coefficients a_j until an optimal solution is obtained. This is easily programmed, and is given in spreadsheet 5-DIFF-EQNS.XLS. In the present example, as we have a total of 6 unknown coefficients, it is necessary to use 6 or more computational points. We use $N = 7$, one more than necessary. The results are shown plotted in the figure, and it can be seen that the computed solution coincides with the exact solution to plotting accuracy, even with the relatively fewer computational points used.

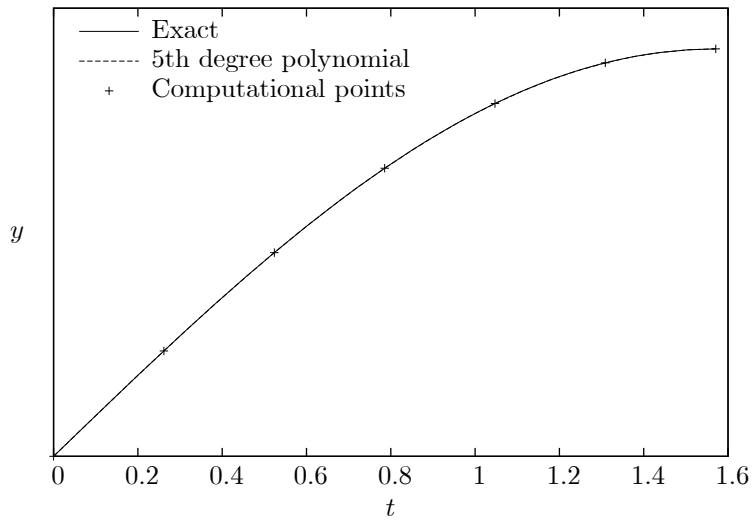


Figure 9-5. Results from solving boundary value ("two-point") problem with spectral method

There are some potential problems with this, but they are easily overcome. Particularly for civil and environmental

engineering problems, where the time or space values are likely to be large (building dimension in millimetres, road and river length in metres, time in minutes over many hours, *etc.*), it is very important to scale the independent variable (time or space) such that the actual computational domain is $[-1,1]$ or $[0,1]$,as was described in the sections on interpolation and approximation. An elegant away around some of the problems is to use orthogonal functions such as Chebyshev or trigonometric functions, but usually scaling to the domains shown is enough.

Of course, the methods used in this section can be used also for initial value problems, and might even be the recommended way to solve them.

10. Improved accuracy at almost no cost – Richardson & Aitken extrapolation

A simple way of obtaining results of a higher order of accuracy is to use Euler's method for two different time steps, and then to use Richardson extrapolation to the limit, where a simple numerical calculation gives a significantly more accurate solution, extrapolating the results to the limit of zero computational step. Two variants were developed by L. F. Richardson[2] and A. Aitken[3] in the early 20th century, the latter where the order of accuracy of the scheme is not known. They are beautifully simple procedures for obtaining more accurate solutions from computational schemes.

10.1 Richardson extrapolation

Consider the numerical value of the computational solution for some physical quantity ϕ obtained using a computational quantity δ, which might be a time or space step, such that we write $\phi(\delta)$. Let the computational scheme be of nth order such that the global error of the scheme is proportional to δ^n, then if Φ is the exact solution we are looking for, we can write

$$\Phi = \phi(\delta) + a_n\delta^n + \ldots$$

where the neglected terms vary like δ^{n+1}. If we obtain numerical values for two different values of δ, δ_1 and δ_2 then we write the equation twice expressing the as-yet-unknown exact solution Φ in terms of an unknown coefficient a_n:

$$\Phi = \phi(\delta_1) + a_n\delta_1^n + \ldots, \tag{10.1a}$$

$$\Phi = \phi(\delta_2) + a_n\delta_2^n + \ldots. \tag{10.1b}$$

These are two linear equations in the two unknowns Φ and a_n, whose numerical value is not important. Eliminating a_n between the two equations and neglecting the terms omitted, we obtain an approximation to the exact solution

$$\Phi \approx \frac{\phi(\delta_2) - r^n\phi(\delta_1)}{1-r^n}, \tag{10.2}$$

where $r = \delta_2/\delta_1$, the ratio of the δ. This is rather more accurate than either of the computed estimates, and the errors are now those of the neglected terms, proportional to δ^{n+1}, so that we have gained a higher-order scheme. Real higher-order schemes are often very complicated – here we have obtained higher-order results with a simple numerical calculation. This procedure, where n is known, is called *Richardson extrapolation to the limit*.

For simple Euler time-stepping solutions of ordinary differential equations, we have $n = 1$ so that say, for $r = 1/2$, equation (10.2) becomes

$$\Phi \approx 2\phi(\delta/2) - \phi(\delta), \tag{10.3}$$

which is very simply implemented, and we attach twice as much weight to the solution with the halved time step.

Example 10.1 Consider the application of Euler's method to the differential equation $dy/dx = y$, with $y(0) = 1$. This has the exact solution $y = e^x$, and we are going to estimate this by using Euler's method to compute to $x = 0$,giving an approximation to $e \approx 2.7183$.

Using two steps of the scheme $y(x+\delta) = y(x) + \delta\, dy/dx = y(x) + \delta\, y(x)$, gives $y(1) = 2.25$, with an error

[2] Lewis Fry Richardson (1881-1953), British physicist, meteorologist, and mathematician, the father of computational fluid mechanics, peace studies, and other fields.

[3] Alexander C. Aitken (1895-1967), New Zealand mathematician and calculating genius.

of 17%. Using four steps gives $y(1) = 2.4414$, with an error of 10%. As $n = 1$ for Euler's method, and we have $r = 0.25/0.5 = 1/2$, the formula (10.2) becomes

$$\begin{aligned} \Phi &\approx 2\phi(\delta_2) - \phi(\delta_1) \\ &= 2 \times 2.4414 - 2.25 = 2.6328, \end{aligned}$$

with an error of only 3%. With almost no effort we have reduced the error by a third. This works even better for smaller steps, and if we combine the last computed result with one from 8 steps with an error of 6% and a value of 2.5658, so we can see that the method is converging slowly, the result is accurate to 1%.

In the case of the trapezoidal rule for numerical integration, $n = 2$, and we have from equation (10.2), for $r = 1/2$:

$$\Phi \approx \frac{\phi(\delta/2) - \frac{1}{4}\phi(\delta)}{1 - \frac{1}{4}} = \tfrac{1}{3}\left(4\phi(\delta/2) - \phi(\delta)\right), \tag{10.4}$$

which is called *Romberg Integration*, and which also works very well. In fact, it can be shown that a single application gives the same results as Simpson's rule.

Example 10.2 Calculate the area under a single lobe of the sine wave $y = \sin x$ between $x = 0$ and $x = \pi$. The result should be 2. In this case, it can be shown that the trapezoidal rule for computing the area is of second order accuracy such that $n = 2$. If we take just two steps, with only the central ordinate being non-zero, we have

$$\phi\left(\frac{\pi}{2}\right) \approx \frac{\pi}{4}\left(\sin 0 + 2\sin\frac{\pi}{2} + \sin\pi\right) = 1.5708,$$

while four steps gives

$$\phi\left(\frac{\pi}{2}\right) \approx \frac{2\pi}{8}\left(\sin\frac{\pi}{4} + \sin\frac{\pi}{2} + \sin\frac{3\pi}{4}\right) = 1.8961.$$

Applying the formula (10.4) gives

$$\Phi = \frac{4 \times 1.8961 - 1.5708}{3} = 2.0045,$$

and the calculation is accurate to 0.2%. In this second-order case the errors are now very small indeed.

10.2 Aitken's δ^2 method

The method known as Aitken's δ^2 method can be used where we do not know the value of n, the order of the scheme. As we have one more unknown, n, we have to do a computation with a third step so that we can write a third equation in addition to equations (10.1a) and (10.1b):

$$\Phi = \phi(\delta_3) + a_n\delta_3^n + \ldots.$$

The equations that we now have to solve are nonlinear (the n occurs in an exponent) but they can still be solved. It is easiest if we choose the ratios of the steps to be the same, $\delta_1/\delta_2 = \delta_2/\delta_3$, which can usually be arranged, the exponents are the same and we can obtain the solution

$$\Phi = \frac{\phi_1\phi_3 - \phi_2^2}{\phi_1 + \phi_3 - 2\phi_2}.$$

In this form the calculation can be poorly conditioned in the limit as all the ϕ go to the same value, when both top and bottom go to zero. It is better to re-write the solution as a correction to the most refined estimate ϕ_3:

$$\Phi \approx \phi_3 - \frac{(\phi_3 - \phi_2)^2}{\phi_1 + \phi_3 - 2\phi_2}, \qquad \text{for} \quad \frac{\delta_1}{\delta_2} = \frac{\delta_2}{\delta_3}. \tag{10.5}$$

and even though the top and bottom of the fraction go to zero as the process converges, the top goes more quickly and the expression, which is a small correction anyway, is less-susceptible to error.

Example 10.3 Re-calculate both examples 10.1 and 10.2 using results from three steps and without assuming a knowledge of n.

Example 10.1 – for the solution of the differential equation we have from equation (10.5):

$$\Phi \approx 2.5658 - \frac{(2.5658 - 2.4414)^2}{2.25 + 2.5658 - 2 \times 2.4414} = 2.796\,6,$$

which is 3% in error.

Example 10.4 Example 10.2 shows the problem solved by Romberg integration solved by Aitken's δ^2 method:– for the trapezoidal rule evaluation of area we have for a time step of $\pi/2$, $\phi_1 = 1.570\,8$ and for $\pi/4$, $\phi_2 = 1.896\,1$. Another calculation using 8 steps is necessary, giving

$$\phi_3 = \frac{\pi}{8}\left(\sin\frac{\pi}{8} + \sin\frac{\pi}{4} + \sin\frac{3\pi}{8} + \sin\frac{\pi}{2} + \sin\frac{5\pi}{8} + \sin\frac{3\pi}{4} + \sin\frac{7\pi}{8}\right) = 1.974\,2,$$

and substitution into equation (10.5) gives

$$\Phi \approx 1.974\,2 - \frac{(1.974\,2 - 1.896\,1)^2}{1.570\,8 + 1.974\,2 - 2 \times 1.896\,1} = 1.998\,9$$

and the estimate is very close to the exact value of 2.

In the approach of this section, it can be seen that for a finite increase in computational effort, possibly doubling or trebling, one still has the advantage of simple schemes but with results that are more accurate by one or more orders, a handsome gain indeed.

11. Piecewise polynomial interpolation – cubic splines

It can be shown that the error of global polynomial approximation increases towards the ends of the interval. There is actually a famous example in numerical analysis (Runge's example) to illustrate the point that *global* polynomial approximation may not very accurate. Consider the function $f(x) = 1/(1 + x^2)$ over the interval $[-1, +1]$ interpolated by a 30-degree polynomial interpolation, as shown in figure 11-1. It can be seen that it approximates well in the middle, but at the ends it is terrible. This is due to the fact that the approximation that has to work so hard to describe the crest, works too hard at the outside ends.

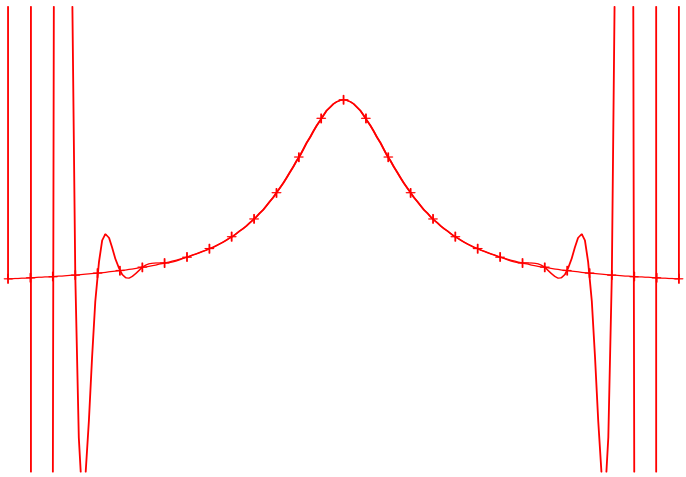


Figure 11-1. Interpolation of the function $1/(1 + 25x^2)$ by global approximation: a 30th degree polynomial over 31 equally-spaced points.

Global polynomial approximation and piecewise-continuous interpolation differ in important respects. If the data to be interpolated is rapidly varying somewhere, the polynomial interpolant is apt to be poor over a finite range. There are no such difficulties in piecewise interpolation. Also, as the number of interpolation points *n* increases, the amount of computation for global approximation goes up like n^2, whereas for piecewise interpolation it increases like n.

The most familiar method for piecewise interpolation is to use cubic splines, which is described in many books (for example, Conte and de Boor, 1980, and de Boor, 1978) and included in software packages. In almost all of those, with the exception of de Boor (1978) the method suffers from a major disadvantage, and it is the intention here to describe that and to recommend that de Boor's "not-a-knot" condition be used.

The physical interpretation and the name of cubic splines is familiar to civil engineers, for it comes from a draughtsperson's flexible strip or "spline" which can be used to fair smooth curves between points. If the strip is held in position at various points by pins, then between any two of those pins there are no lateral forces acting so that the shear force in the strip is constant, the bending moment varies at most linearly, and hence by beam theory (for sufficiently small deflections) the strip takes on a cubic variation between the two points. As the variation of moment is different between other points, other cubics will apply there. However, because the shear force and bending moment are continuous across each pin, then the first and second derivatives are continuous across the pins, or interpolation points. With four unknown coefficients for the cubic in between each pair of points, and the requirement that each of the two cubics, to left and right of each interpolation point, must interpolate at that point and must have the same first and second derivatives, almost enough equations are obtained for all the four coefficients of each cubic. It is necessary, however, to specify two more conditions. This may be by specifying the slope at the two end points, as in Conte and de Boor (1980, Section 6.7), or by arbitrarily specifying that the second derivative at both the end points is zero. This "moment-free" end condition gives the so-called "natural spline" approximation, which is the method usually adopted. In general, however, there is no particular reason at all why the second derivative of the interpolating spline should be zero at the ends, and we disagree with thuse of the term "natural".

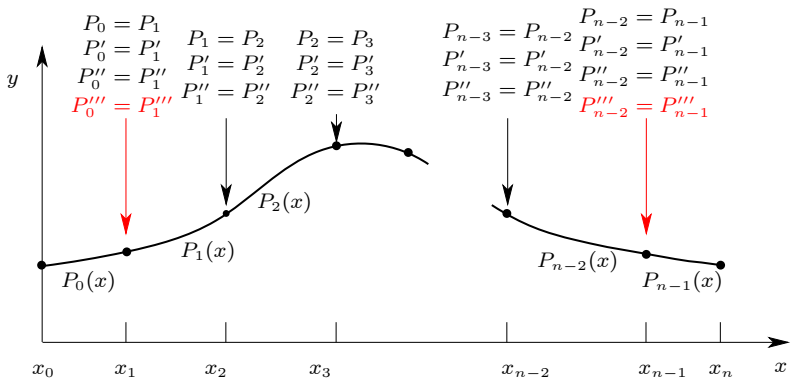


Figure 11-2. Piecewise-continuous interpolation by cubic splines with not-a-knot conditions shown in red

A superior means of interpolation is to use the "not-a-knot" condition at the first and last *interior* points (de Boor 1978), where it is required that in addition to the first and second derivatives agreeing to left and right of the interpolation point, that also the third derivatives agree, to give the two extra equations necessary, as shown in figure 11-2. The physical significance of this is simply that a single cubic interpolates over the first two intervals and another over the last two intervals. In the notation of figure 11-2 this means that $P_0(x) \equiv P_1(x)$ and $P_{n-2}(x) \equiv P_{n-1}(x)$. No arbitrary assumptions have been introduced, and it can be shown that the error is much less than for "natural" splines. It is a source of amazement to the lecturer that the "not-a-knot" conditions have not been adopted as the standard means of implementing spline interpolation if no gradient information is known.

The manner in which the fluctuations of the interpolating polynomial are held down has made cubic splines popular as a means of interpolating and obtaining derivatives numerically. FORTRAN programs are given in de Boor (1978), and are available as shareware.

There is a spreadsheet available that contains the necessary subroutines that the lecturer has coded in Visual Basic. If one downloads the spreadsheets **URL:** *http://johndfenton.com/Lectures/Numerical-Methods/Splines.xls* and **URL:** *http://johndfenton.com/Lectures/Numerical-Methods/7-Splines.xls* into the same directory, and opens the latter, then the former, which contains the necessary programs, is automatically opened. Students might like to play with their own data, as the latter is easily modified.

Figure 11-3 shows interpolation of Runge's example using cubic splines with 9 and 31 points, the latter being the same as for Figure 11-1. With 9 points some differences between the function and the interpolant are clear, but very much less than for global polynomial approximation. With 31 points the interpolant everywhere is obscured by the function, even near the sharp crest. The superiority over global polynomial methods is obvious. Nevertheless, the

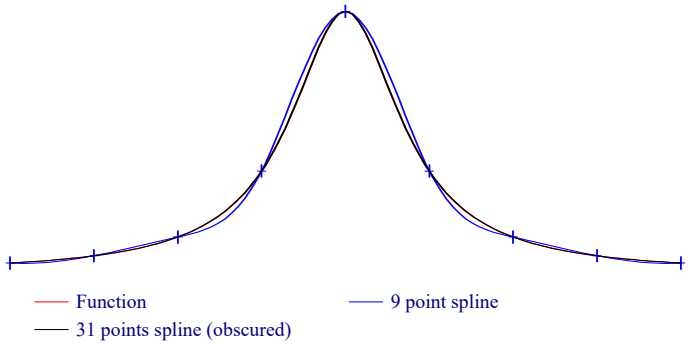


Figure 11-3. Interpolation of the function $1/(1+25x^2)$ by a cubic spline over 9 and 31 equally-spaced points

global polynomial methods described above are simple in concept and in application, and for relatively low-degree interpolation might generally find favour.

12. Piecewise polynomial approximation – cubic splines

The lecturer has developed a package for the use of splines to perform approximation of data rather than interpolation. The files and a document describing the method can be found here: **URL:** *http://johndfenton.com/Approximating-splines/index.html*. It overcomes some of the problems of polynomial approximation as described above.

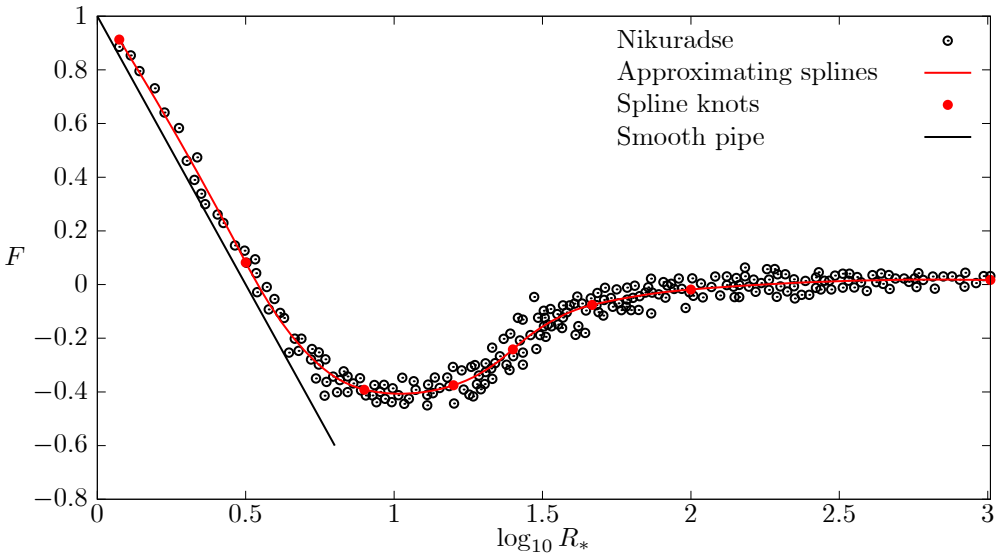


Figure 12-1. Nikuradse's results for F as a function of the logarithm of the grain Reynolds number

Figure 12-1 shows Nikuradse's results for resistance in pipes approximated by splines.

13. A programming language – Visual Basic and Excel Macros

Visual Basic (VBA) is a typical high-level programming language, such as Fortran, C++, or Pascal. It can be entered easily *via* the use of Excel Macros, and this is where we start.

13.1 Macros

Introduction: Macros are used in Excel to automate tasks that are performed regularly, or for problems which require "tailoring" to specific needs. The Visual Basic language is used to write scripts, and these scrips are run as

"Macros". It is possible to record a Macro in Excel, or you can write it from scratch in Visual Basic code. Often it is best to record a section of the Macro, and then change the code to suit your particular needs.

Recording a Macro: When a Macro is recorded, Excel follows the steps you take in Excel, and converts each of these commands into a Visual Basic script. When the Macro is 'run', the Visual Basic script is run, and this tells Excel to execute the commands you have chosen.

- Go to `Tools/Macro/Record New Macro`
- The dialogue box will ask for a name for the Macro and you also have the option of selecting a shortcut key. If you enter a key in this field, you are able to press Control+(the key you select) to run the Macro. This prevents having to use the menu every time you wish to run a Macro. For a Macro that graphs data, you may want to call it "graph" and give it the shortcut key "g".
- You also have the choice of specifying a Worksheet where your Macro will be stored. It is probably best to choose Personal Worksheet, which runs every time you run Excel. Otherwise the worksheet which contains the Macro has to be open if it is to be available.
- Once you are satisfied, click "OK" and then perform the steps (as you would normally in Excel) that you wish to automate. It is a good idea to have the steps clear in your mind and maybe even run through them before you record your Macro. Excel will convert these steps into Visual Basic commands. When you are finished, go `Tools/Macro/Stop Recording`.

How to run a Macro:

- If you open an Excel file that has a Macro embedded in it, a warning box may appear. You must click "Enable Macros" to allow the Macros to run.
- If you wish to run a Macro:
  - The simplest is to press Control+(the shortcut key you selected), or,
  - Go to `Tools/Macro/Macros`..(which brings up the Macro dialogue box), select the Macro you wish to run from the list, and click "Run", or
  - Press "Alt+F8" (which brings up the Macro dialogue box), select the Macro you wish to run from the list and click "Run".

  This will execute the script. Once the script has been executed, the original data will have been changed or rearranged (depending on the purpose of the Macro) and therefore running the Macro again will have no effect. It is necessary to reset the data to the original format to see the Macro run again. The "undo" function will not reset the data. If you have more than one Excel workbook open at the same time, you can run Macros from one workbook on a sheet from the other. For example if there is a Macro stored in "Book1" called "Macro1", and you wish to use it , in "Book2", open both books,then from Book2 go to Tools/Macro/Macros...the Macro will be named and by the book it is stored in, followed by the "!" symbol, and then the Macro name, for example, "Book1.xls!Macro1".

A warning: Personally I have found Macros very useful, but there are some unsatisfactory aspects. In particular is the use of absolute cell referencing when one records a Macro. For example if you want to record a Macro to copy two cells three places to the right, Excel records the absolute reference and only uses that. For example, if you record in C13 and C14 and wish to copy to C16 and then use it anywhere else in the Workbook, it will only return to C13 and do it there. There is supposed to be an alternative form of Relative referencing, that you can turn on with some difficulty, using `Tools/Options/General/R1C1` and a blank Worksheet, but I have found that it still records Macros in the Absolute referencing mode.

How to view a Visual Basic script: The Macros for a particular workbook are saved with the Excel file they have been recorded/written for, and it is necessary to open the workbook in order to view the script. Once the Workbook containing a Macro is open, go to `Tools/Macro/Macros...` , select the name of the Macro you wish to view and then click "Edit". This will open up the Visual Basic Editor, and your script will be displayed. The "Project" window in the top left hand corner of the Visual Basic Editor lists the "VBAProject" for each workbook currently open, as well as the "Sheets" contained in the workbook and the "Modules". The Modules are where the scripts for the Macros are stored. The different scripts for the different Macros can be navigated by opening the

desired Modules.

Writing your own Macro: Recording a Macro is a useful skill to have, however for most engineering applications, the recorded Macro alone does not deliver a suitable result. It is often necessary to tailor a Macro the specific needs of the user, and this requires working with the Visual Basic programming language. NOTE: Visual Basic is a user-friendly language, and you do not need an intricate knowledge of it to use macros.....do not be afraid!!!

The language is actually quite harmless, and most of the commands are English (or slightly modified) words, like "MsgBox" to display a message box and "Select" to select a cell. In this way, it is possible to read the code and get the general idea of what is going on.

Macros can be written from scratch if you are a Visual Basic whizz. If you don't know this language thoroughly, or you want to speed up the process of writing a Macro, it is possible to record a Macro that contains the vital elements you desire, and then edit it to suit your needs.

The best way to learn to write Macros is to carefully examine other people's, and determine the elements you can extract to apply to your problem. It is recommended that you run through the following example Worksheets to see what the Macros can do, and how they are written. The selection of examples covers common engineering related problems, so you may like to modify these Macros to suit your needs.

Errors in Macro scripts: If you try to run a Macro and there is an error in the script, an error message will appear:

If you click "Debug", the Visual Basic Editor will open and the error in the script will he highlighted in yellow, where you may be able to fix the problem.

If you click "End", the normal Excel screen will appear, and the bug in the script will remain as it is.

Examples: One that I use is a routine which takes a cell and copies it down until it meets another non-zero cell or the end of a data block.

```
Sub Copydown()
    Selection.Copy
    Range(Selection, Selection.End(xlDown)).Select
    ActiveSheet.Paste
End Sub
```

You can download the example spreadsheets and then open the Visual Basic Editor to examine the scripts. The scripts are heavily annotated (comments in green after the ') and describe each element and its purpose.

13.2 Visual Basic

13.2.1 What Is Visual Basic?

Visual Basic (VB, or VBA) is a high-level language, in that it contains many of the characteristics of other high-level languages such as Fortran, Pascal, C++. It evolved from the earlier DOS version called BASIC (Beginners' Allpurpose Symbolic Instruction Code). It is a fairly easy programming language to learn.

Most conveniently for us, it sits behind Excel, and that provides a convenient way into it, *via* Macros at first, and then one can use it alone to do more complicated programming operations. As Excel is on most personal computers, VB is available to most of us, whereas with other high-level languages separate compilers are necessary.

The current version wick works behind Excel is VB 6. After this version Microsoft has called it VB.NET.

13.2.2 Beginners Guides

There are many of these on the Internet. There are *many* books available. Personally, the lecturer has always found using the software to be a better way than many books.

13.2.3 Entering via Excel Tool/Macro/Macros

Pressing Alt-F11 or Tool/Macro/Macros opens up the VB window.

To insert a new Procedure, go Insert/Procedure, and choose probably to call it a "Sub", which is short for Subroutine, a particular program which can do a variety of things. The other details do not matter at this stage.

If you have to return a numerical value for a Worksheet, for example, then call it a Function.

It is a good idea to press Ctl-G to bring up the "Immediate" window, which shows results which can be cut and pasted, deleted, *etc*.

13.2.4 Features of the language and interface

As an example, let us type the following, noting that anything after an apostrophe is treated just as a Comment.

```
Sub Print_Example()
String1 = "See results in the Immediate pane" 'A String, enclosed in quotes
String2 = "Hello" 'Another string
Debug.Print String1 'This is quite a clumsy way to print here
For i = 1 To 5 'A "For" loop, which enables repetitive operations
  Debug.Print String2, i ' We print out the value of i as it repeats
Next i' Tells the computer to increment i and loop again until satisfied
End Sub
```

Debugging: Now, with the cursor somewhere in that program, we could press F5 and the program would run. However, *almost always*, we will have made some error, and we will instead "Debug" it by pressing F8, then we can step through the program line by line, checking our output as we go, by repeatedly pressing F8. To leap to a point further down, move the cursor to the desired line and then press Ctl-F8. If there is anything wrong, the program will tell us. In this case, all results appeared in the Immediate Window as we stepped through.

Writing to files: Now we consider a rather more powerful approach, where we can write to a file directly. For example, we want to print out values of $\sin x$ for x from 0 to π in 10 steps. Firstly we have to Open a file and give it a Stream number, 3 in this case, and after writing is complete we should close it:

```
Open "C:/Testfile.txt" For Output Shared As #3
.............
Close #3
```

Repeated operations, Looping: Now we can set the value of π and step through the values, printing out to Stream 3, and then closing the stream and file at the end.

```
Pi = 3.141592653
For i = 0 To 10
   Print #3, i, Sin(i*Pi/10)
Next i
```

Variable names must begin with a letter as the first character. You cannot use a space, full stop (.), exclamation mark (!), or the characters @, &, $, # in the name. Names can't exceed 255 characters in length.

If...Then...Else...End If Statements: These allow huge developments, enabling branching of the program execution depending on the value of an expression.

If <condition> Then [statements] [Else elsestatements]

For example if we have a number x, and we want to take the logarithm, we might guard against zero or negative values by:

```
If x > 0 Then
   y = Log(x)
Else
   Debug.Print "Attempted to take Log of 0 or negative number"
End If
```

Various logical or Boolean operators can be used: < (Less than), <= (Less than or equal to), > (Greater than), >= (Greater than or equal to), = (Equal to), <> (Not equal to).

Watching values of quantities: So that we don't have to write out print statements for values of quantities, at the Debug stage we can use the Watch Window (`View/Watch Window`) and by right-clicking on the Expression column, we can add a Watch and monitor the value of any variable. This is very helpful.

Finally: there is a huge amount in VB, however it is possible to use it at a relatively simple level – one learns by going from the known to the unknown. Good luck!

References

Conte, S. D. & de Boor, C. (1980), *Elementary Numerical Analysis*, third edn, McGaw-Hill Kogakusha, Tokyo.

de Boor, C. (1978), *A Practical Guide to Splines*, Springer, New York.

Fenton, J. D. (1994), Interpolation and numerical differentiation in civil engineering problems, *Civ. Engng Trans, Inst. Engnrs Austral.* **CE36**, 331–337. http://johndfenton.com/Papers/Fenton94b-Interpolation-and-numerical-differentiation-in-civil-engineering-problems.pdf.