

# Using Computers For Scientific Work

©1996 C.T.J. Dodson

## 1 Introduction

### Objective and Prerequisites

The objective of this course is to acquire a practical working knowledge of how computers help find numerical solutions of statistical and mathematical problems and provide an environment in which to create high quality reports of scientific work. The software we shall need is a web browser, a spreadsheet package, Mathematica for computer algebra, calculus and graphics, and compilers for BASIC and C.

The course is for class or individual use, using a computer which by default is expected to be a PC equipped with the above software. The assignments are meant to be done at a computer in computer lab sessions with some human tutorial assistance; if you are working independently, then you will be expected to have to hand the manuals for the packages or learn how to use the on-line help provided with them. A bibliography of reference books that may be found useful is provided [1].

### Assignments

Your reports on the assignments and the on-line materials are your course notes and you may wish to refer to them in later courses, during project work or when in employment, so make them as useful as possible—by adding any clarifying notes on the programs, for example. You should include also in your reports any formulae that are used, to make them as self-contained as possible.

Mainly we shall work from examples which can be expanded and embellished to serve as models in other contexts. Remember, you should always treat with suspicion *any* mathematical or computational statements, until you have satisfied yourself that they make sense; this goes for lecture notes, assignments, and books as well!

### User Goals

By the end of the course, you should be able to

- use a spreadsheet to analyse data and plot graphs
- use Mathematica for algebra, calculus and geometric graphics
- read and make modifications to straightforward programs in BASIC and in C
- write scientific reports
- create simple hypertext documents for the World Wide Web.

### Updates!

If you are using these materials for learning or teaching, then please check periodically the date of the latest update (at the bottom of the page) as they are in an evolving state.

### Copyright Statement

Members of educational institutions are encouraged to make these materials available to others or distribute copies for educational use free of charge, but please acknowledge the source, include this front page and please also send the author any feedback. For any other purpose except educational, such as commercial, use of these materials is prohibited without prior written permission.

### Comments Please!

Please send me your comments on these course materials, and let me know of errors by email at: [ctdodson@manchester.ac.uk](mailto:ctdodson@manchester.ac.uk)

## 2 Using Computers

In technical work for courses and projects, as well as in subsequent working environments, several kinds of computing activities may be called for:

**Wordprocessing** On a PC, **WordPad**, **Word**, **WordPerfect** etc are standard wordprocessing packages which to varying degrees can incorporate scientific materials like mathematical formulae and graphics in documents for printing and you will need to study the documentation on-line for your computer. Often, files from one package may be imported to another, but sometimes problems arise between upgrades of a package so beware. The easiest way to create a scientific report on a PC is in Microsoft Word. You can import graphics from Mathematica as \*.wmf files; do this by selecting the graphic then under 'Edit' 'Save Selection As' followed by 'Windows Metafont' and save with extension .wmf. The problem with Microsoft Word is that it is very tedious to create mathematical equations.

The standard wordprocessing package used by most mathematicians and many other scientists is **L<sup>A</sup>T<sub>E</sub>X**, which is free, and this can generate PDF output with hyperlinks, include graphics and provide slideshows for presentations. For details of how to obtain the software and examples of typical documents see PDF **L<sup>A</sup>T<sub>E</sub>X** By Example Tutorial.

**Hypertext** For the generation of hypertext documents, such as webpages, we can write directly in **Hypertext Markup Language (HTML)** [3] or use authoring programs that can convert electronic documents into html versions with appropriate links. **L<sup>A</sup>T<sub>E</sub>X** has such a converter, **Latex2html**, for example and there is a beginner's guide at Creating HTML documents

A convenient method however, is to have a hyperlinked PDF document and for scientific work you can follow the procedure in PDF **L<sup>A</sup>T<sub>E</sub>X** By Example Tutorial. This was used to create the present PDF document.

**Spreadsheets** Packages like **Lotus**, **Excel** and **Quatro** allow you to process data and generate graphs, histograms, piecharts etc for interpretation of results, reports and presentations. Often, files from one package may be imported to another. You will need to familiarise yourself with the spreadsheet and graphing facilities on the computer you use—it is best to try several, just in case sometime you find that your usual one is not on the machine you must for a task in hand. Learn to use the on-line help menus!

**Computer Algebra** Packages like **Mathematica**, **Maple**, **Matlab**, **MathCAD** are standard software used by mathematicians, scientists and engineers for research and teaching. In this course we shall use Mathematica but we also provide materials for Maple. See Beginning Mathematica for beginning Mathematica.

To begin to learn how to use Mathematica download the file TRYMMA.nb from Introduction to Mathematica and try the functions there by selecting the inputs and keying **Shift-Enter**. We return to this in the next section.

**Numerical analysis** The common problems that arise in scientific work are summarised below. In each case we shall investigate a BASIC program to perform the work and to generate the output. The DOS **Quick Basic (QBASIC)** environment is convenient for writing developing and running basic programs. We shall rework these programs in the more efficient **C** programming language. You should look up in the standard mathematical texts or your notes to make sure you understand why the algorithms in the programs do what is wanted.

**Solution of algebraic equations** The standard two methods are **Newton's**, which is very efficient for functions which have non-zero derivatives, and that of **Bisection of intervals** for functions which are continuous but not necessarily having derivatives. All problems of this kind can be cast into the form:

Find  $x$  such that  $f(x) = 0$ .

Iterative procedures for solution depend on making an initial guess  $x_0$  at the solution, so it is usually best to plot a graph of  $f(x)$  over a wide range to find all regions likely to contain solutions. It is important always to know if the solution you find is unique or if others exist. From  $x_0$ , the program generates the next approximation  $x_1$ , and so on.

The bisection method involves bisecting subintervals in which the function  $f(x)$  changes sign, until the desired accuracy is obtained. If the solution lies in  $[a, b]$ , then test if there is a change in sign of

$f(x)$  on the first half or the second half of the interval. Then subdivide the interval on which the sign change occurred. Iterate this subdivision. This is a slow method but can work satisfactorily when Newton's method fails.

Newton's method uses the derivative to generate a line which points towards the solution and it is surprisingly efficient. The algorithm is given by:

$$x_{n+1} = x_n - f(x_n)/f'(x_n), \quad n = 0, 1, 2, \dots$$

and of course will work **only** when the derivative,  $f'(x_n)$  is not zero.

You can see the implementation of Newton's method in the BASIC program on page 6 or in the C program on page 9.

You can find out more about Newton by clicking about Newton

**Finding limits** Sometimes the solution of a problem is in the form of a limit of a function,

$$\lim_{x \rightarrow a} f(x),$$

or an asymptotic value for a function is needed. If it is possible to substitute for the desired value  $f(a)$ , then there is no problem but sometimes the form of  $f(x)$  is such that  $f(a)$  is indeterminate, like  $0/0$  for example. In this latter case we investigate the limiting value numerically.

You can see the implementation of a limit finding algorithm by clicking here for the BASIC program on page 6.

**Tabulation of functions** Real problems are notorious in generating solutions which are mathematically complicated; then we often want to tabulate them and graph the results.

You can see the implementation of a tabulation algorithm by clicking here for the BASIC program on page 6 and here for the C program on page 9.

**Integration** Solution of problems like evaluation of work done in a process, cost of an operation, total time elapsed, total mass transferred, etc, involve the integration of the quantity-density over the domain of interest. Thus

$$W(a, b) = \int_a^b F(x) dx$$

could represent the work done in a process where the force  $F(x)$  acts along the  $x$ -axis from  $x = a$  to  $x = b$ . Clearly, the answer is a function of  $a$  and  $b$  and geometrically it is the area under the graph of  $F(x)$  from  $x = a$  to  $x = b$ . If  $F(x)$  is not a standard integral, then we can estimate this area by a limiting process of sums of areas over small subintervals of width  $\Delta x$ . The simplest way is to use the rectangles of height equal to the value of  $F(x)$  at the centre points  $x_i$  :

$$\int_a^b F(x) dx \approx \sum_{i=1}^{i=N} F(x_i) \Delta x$$

However, a much more efficient algorithm for doing this is that using **Simpson's Rule** and we shall employ this in our program:

$$\int_a^b F(x) dx \approx \frac{\Delta}{3} (F(x_0) + 4F(x_1) + 2F(x_2) + 4F(x_3) + 2F(x_4) + \dots + 2F(x_{N-2}) + 4F(x_{N-1}) + 2F(x_N))$$

Clearly, accuracy is improved by using small subintervals. Find out more about Simpson by clicking about Simpson

You can see the implementation of Simpson's Rule in the BASIC program on page 7 and in the C program on page 11.

**Solving ordinary differential equations** It is a fact of scientific life that real processes evolve in ways which depend not only on the values of quantities but also on their rates of change. Thus, mathematical models lead to **ordinary differential equations** to describe real phenomena [1]. So, we often need to solve **initial value problems**: given an initial value for a variable of interest, and a function which represents its rate of change, find the form of the function over some finite interval of time. For simple expressions this can be done by a standard integral, but for more complex problems, typically

nonlinear models, we need to resort to numerical analysis. We shall use the procedure known as **improved Euler's method** which uses an averaged linear approximation to the incremental changes in  $y$ . Find out more about Euler by clicking about Euler

Consider the differential equation

$$y' = F(x, y)$$

with initial conditions

$$y(x_0) = y_0$$

We start with  $y(x_0) = y_0$  and step  $x$  to  $x_1 = x_0 + h$ , then approximate the increment to  $y(x_1)$  by the linear average:

$$y(x_1) = y(x_0) + \frac{h}{2}(F(x_0, y(x_0)) + F(x_1, u_1))$$

where

$$u_1 = y(x_0) + hF(x_0, y(x_0))$$

The algorithm iterates this procedure.

You can see the implementation of Euler's Method by clicking here for the BASIC program on page 8 and here for the C program on page 11.

It is a theorem in calculus due to Cauchy that such an initial value problem actually *has a unique solution*. You can find out more about Cauchy by clicking about Cauchy

**Solving partial differential equations** For phenomena that depend on rates of change of more than one variable, we need to use models that involve **partial differential equations**. Examples of such models are the heat equation for the evolution of the temperature distribution in a body, the wave equation for the motion of a wavefront, and the flow equation for the flow of fluids. In such cases we need to have not only the initial conditions, but also **boundary conditions** for the region in which the model applies; thus we have to solve **boundary value problems**.

The solution of partial differential equations is in general difficult analytically, because most functions cannot be integrated in terms of elementary functions. However, numerical procedures involving iterative application of **finite difference equations** [7] as approximations to derivatives can be very effective in providing solutions sufficiently good for practical applications. We shall take three examples of explicit difference equations to solve the partial differential equations for heat flow, fluid flow and waves. The programs for these cases will be written in a straightforward style which can be understood easily and can be used as templates for other cases.

#### Difference equation for the heat equation

We illustrate the method of deriving a difference equation approximation following Smith [7] for the case of the heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

which models the temperature  $u(t, x)$  in a rod which lies along the  $x$ -axis. Thus,  $u$  is a function of time  $t$  and position  $x$  and so we view the region of solution as the  $x - t$  plane. Imagine that the  $x$ -axis is subdivided into intervals of length  $h$ , and the  $t$ -axis is subdivided into intervals of length  $k$ , such that

$$x_m = mh \quad \text{and} \quad t_n = nk, \quad \text{for } m, n = 0, 1, 2, \dots$$

We seek to approximate the solution  $u(t, x)$  by  $u_{m,n}$  at the grid point represented by  $m, n$ .

We approximate the left hand side of the heat equation by

$$\frac{\partial u}{\partial t} = \frac{u_{m,n+1} - u_{m,n}}{k}$$

and applying a similar procedure twice along the  $x$  direction, the right hand side becomes

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{m+1,n} - 2u_{m,n} - u_{m-1,n}}{k^2}$$

Assembling the approximation to the full heat equation we get an iterative formula for solution values in terms of earlier values; so we can step forward in time from given initial and boundary values with an *explicit* equation:

$$u_{m,n+1} = ru_{m-1,n} + (1 - 2r)u_{m,n} + ru_{m+1,n} \quad \text{with } r = \frac{k}{h^2}$$

This procedure is encoded in the C program on page 13 below, for a particular choice of boundary conditions.

This explicit method can be shown to be valid for increments giving  $0 < r \leq \frac{1}{2}$  [7], and the same book provides alternative implicit difference equations with wider applicability.

A C program implementation of the difference equation method for numerical solution of the heat equation can be found on page 13.

#### **Difference equation for the flow equation**

In simple fluid flow problems, the conservation of flow of a quantity  $u(t, x)$ , can be modelled, for 1-dimensional situations, by an equation of type

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0$$

A difference equation approximating this is [7]

$$u_{m,n+1} = \frac{1}{2}pu_{m-1,n} + (1 - p^2)u_{m,n} - \frac{1}{2}p(1 - p)u_{m+1,n} \quad \text{with } p = \frac{k}{h}$$

This procedure is encoded in the C program on page 14, for a particular choice of boundary conditions.

This explicit method can be shown to be valid for increments giving  $0 < p \leq 1$  [7], and the same book provides alternative implicit difference equations with wider applicability.

A C program implementation of the difference equation method for numerical solution of the flow equation can be found on page 14.

#### **Difference equation for the wave equation**

A 1-dimensional wave  $u(t, x)$  can be modelled by

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

The difference equation approximating this is [7]

$$u_{m,n+1} = r^2u_{m-1,n} + 2(1 - r^2)u_{m,n} + r^2u_{m+1,n} - u_{m,n-1} \quad \text{with } r = \frac{k}{h}$$

A C program implementation of the difference equation method for numerical solution of the wave equation can be found on page 15.

### **3 Computer algebra**

You may encounter the main computer algebra packages, **Mathematica**, **Matlab** and **Maple**, for all of which student editions are available. This software can do all of the usual calculus, like differentiation and integration, as well as more advanced mathematical and statistical operations, and provides excellent graphics generation, including animation. See Beginning Mathematica for getting started.

Download and save the Mathematica notebooks **TRYMMA.nb** and **StatsIntro.nb** from Mathematica Notebooks. Open these in Mathematica to try out the various operations. You will find that Mathematica can do most of the mathematical, statistical and numerical work that you need in your coursework, as well as plot high quality 3-dimensional colour graphics and animate them.

For Mathematica notebooks concerning calculations and graphics on curves and surfaces consult the webpages **Curves and Surfaces**

Mathematica has a function **DSolve** for attempting to solve ordinary differential equations analytically and another called **NDSolve** which does solve them numerically. For more extensive functions and notebooks concerning the solution of differential equations consult the webpages **Ordinary Differential Equations**.

## 4 BASIC programs

The BASIC programs described here are quite simple and easily understood. You can find many introductions to BASIC programming, and a language reference book usually arrives with your documentation for DOS. In each case below, try to ascertain the sequence of mathematical operations that are being done, which will be easier if you look up first in a calculus book what the individual methods are doing. More advanced programs of relevance to scientific work can be found in, for example, the book by Cordingley and Chamund [2].

### 4.1 Newton's method for solving equations

```
10 REM NEWTON'S METHOD FOR SOLVING F(X)=0
110 REM THE FUNCTION F(X) AND ITS DERIVATIVE G(X) MUST BE INPUT
120 REM THE INITIAL GUESS X0 MUST BE INPUT
200 INPUT "INITIAL GUESS X0"; X
210 N = 0
220 PRINT " N", " X": PRINT
300 REM NEWTON'S ITERATION LOOP
310
320 PRINT N, X
330 F(X) = X ^ 2 - 2
340 G(X) = 2 * X
350 XNEW = X - F(X) / G(X)
360 X = XNEW: N = N + 1
370 IF INKEY$ = "" THEN 370
380 GOTO 320
390 REM PRESS CTRL-BREAK TO STOP
400 END
```

### 4.2 Finding limits

```
10 REM ESTIMATE THE LIMIT OF A FUNCTION F(X) AS X->A
100 REM INPUT THE DEFINITION OF THE FUNCTION IN LINE 230
150 DEFDBL A, F, H, X
160 INPUT "POINT AT WHICH LIMIT WANTED A="; A
170 INPUT "INITIAL DISTANCE X-A="; H
180 INPUT "NUMBER OF STEPS="; N
190 D$ = "###.##### ###.#####"
200 REM ITERATION LOOP
210 FOR I = 0 TO N
220 X = A + H
230 F = SIN(X) / X
240 PRINT USING D$; X; F
250 H = H / 10
260 NEXT I
300 END
```

### 4.3 Tabulation of functions

The first program tabulates the function and prints it on the screen, after you enter the range and step size. The second program creates a file called **TAB.OUT** and prints the tabulation in that.

#### Printing to screen

```
10 REM TABULATE A FUNCTION F(X) FROM X=A TO X=B IN STEPS DX
100 REM ENTER THE FUNCTION F(X) IN THE NEXT LINE
110 DEF FN F(X)=X^2
```

```

120 :
130 INPUT "FROM X= ";A
140 INPUT " TO X= ";B
150 INPUT "IN STEPS OF ";DX
200 :
210 FOR X=A TO B STEP DX
220   Y=FN F(X)
230   PRINT X,Y
240 NEXT X
300 :
310 END

```

**Printing to a file** This program outputs to a file **TAB.OUT**, which is created if it does not already exist, and is overwritten with each run.

```

10  REM TABULATE FUNCTION F(X)
100 :
110 DEF FNF (X) = X ^ 2
120 :
130 INPUT "FROM X= "; A
140 INPUT " TO X= "; B
150 INPUT "IN STEPS OF "; DX
200 OPEN "TAB.OUT" FOR OUTPUT AS #1
205 PRINT #1, " X           F(X)"
210 FOR X = A TO B STEP DX
220   Y = FNF(X)
230   PRINT #1, X, Y
240 NEXT X
245 CLOSE #1
300 :
310 END

```

#### 4.4 Simpson's Rule for integration

```

10  REM SIMPSON'S RULE FOR INTEGRAL OF F(X) OVER interval [A,B]
100 REM ENTER THE FUNCTION F(X) IN THE NEXT LINE
110 DEF FN F(X) = SQR(1 - X ^ 2)
120 :
130 INPUT "FROM X= "; A
140 INPUT " TO X= "; B
150 INPUT "NUMBER OF SUBINTERVALS (EVEN) "; N
160 :
170 H = (B - A) / N
200 :
210 SUM = FNF(A): C = 2
220 FOR K = 1 TO N - 1
230   C = 6 - C: REM GENERATES COEFFICIENTS 4,2,4,2,...
240   XK = A + K * H
250   SUM = SUM + C * FNF(XK)
260 NEXT K
270 SUM = SUM + FNF(B)
280 :
290 INTEGRAL = (H / 3) * SUM
300 :
310 PRINT "NUMERICAL INTEGRATION OF F(X) OVER [A,B] = "; INTEGRAL
400 END

```

## 4.5 Euler's method for solving initial value problems

This program outputs to a file **IVP.OUT**, which is created if it does not already exist, and is overwritten with each run.

```
10 REM EULER'S METHOD FOR SOLVING Y'=F(X,Y) WITH Y(0)=X0
110 REM YOU ENTER THE FUNCTION F(X,Y) IN THE NEXT LINE
200 DEF FNF (X, Y) = X + Y
210 INPUT "INITIAL VALUES X0, Y(0)"; X, Y
220 INPUT "ITERATION STEP SIZE H"; H
230 INPUT "NUMBER OF STEPS"; N
240 INPUT "NUMBER OF STEPS BETWEEN PRINTS P"; P
250 REM EULER'S ITERATION LOOP
255 OPEN "0", #1, "IVP.OUT"
260 PRINT #1, " POINT X      VALUE Y"
280 PRINT #1, X, Y
300 FOR I = 1 TO N
310     U = Y + H * FNF(X, Y)
320     Y = Y + (H / 2) * (FNF(X, Y) + FNF(X + H, U))
330     X = X + H
340     IF INT(I / P) = I / P THEN PRINT#1, X, Y
350 NEXT I
355 CLOSE #1
400 END
```

## 5 C Programs

The C programming language is extremely efficient, but somewhat harder to learn than BASIC. The standard reference work, written by the authors of the original C language is Kerninghan and Ritchie [5]. Two other books that illustrate the wide range of existing subroutines are Press et al. [6] for Numerical Recipes, and Jamsa [4] for tips on C and C++. C++ is a language which has developed out of C, and which contains C as a subset.

### Using C Programs

To use a program written in C, you need to follow these steps:

**Finalise** Use an editor to put in the definitions of the functions to which you wish to apply the algorithm, then save the program, say as prog.c. We call **prog.c** the **source file** or **source code** for the program. It needs to be pre-processed to create an executable file.

**Compile** In Borland C, for example, select the menu item **Compile** from the **Project** menu. On unix workstations you do the compiling from a command line with something like

```
cc prog.c -o prog -lm or gcc prog.c -o prog -lm
```

which will compile the program, linking in the mathematics library of functions and create an executable file called **prog**.

If the source file **prog.c** contains syntax errors, then you will be given error messages. The errors need to be corrected before compiling is possible. Warning messages are indicative of possible shortcomings in your program, but may not stop it compiling.

**Run** Run the program and view the output.

The examples that follow will illustrate some straightforward procedures in C, to perform similar tasks to those you have seen above using BASIC programs. You are expected to be able to read, understand and modify the C programs to use them for different numerical examples. Typical modifications will be to change output between screen and a file; examples of both are given.

## 5.1 Tabulation of Functions

Here is an example of source file for a C program which you should save as **fahr2c.c**; it computes and prints a table of conversion between Fahrenheit and Celsius temperatures:

```
/* This is how to put in a remark */
/* This C program should be saved as fahr2c.c */
/* To compile it run cc fahr2c.c or select compile */

#include <stdio.h> /* This inputs standard in out procedures */
#define lower 0 /* lower limit */
#define upper 300 /* upper limit */
#define step 20 /* step size */

void main(void)
{
int fahr; /* Declares integer variable name */
printf("deg F deg C\n"); /* Prints table header */
for (fahr = lower; fahr <= upper; fahr = fahr + step) /* Do loop */
printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32)); /* Print table */
} /* End */
```

Note that this program outputs its results to the screen.

### Exercise

Try these two C programs which perform similar tasks with different modes of output. Note the way C represents powers using the function **pow**, which is different from BASIC. The sequence being tabulated is actually convergent to the transcendental number  $e$ .

- ```
1. #include <stdio.h>
#include <math.h>

void main()
{
float N;
printf("\t N \t(1+1/N)^N\n");
for (N = 1000; N <= 10000; N = N + 1000)
printf("\t%5.0f %1.9f\n", N, pow((1+1/N),N)); /* pow for raising to power */
}
```
- ```
2. #include <stdio.h>
#include <math.h>

void main()
{
float N;
FILE *fp;
fp = fopen("e.out", "w");
fprintf(fp, "\t N \t(1+1/N)^N\n");
for (N = 1000; N <= 10000; N = N + 1000)
fprintf(fp, "\t%5.0f %1.9f\n", N, pow((1+1/N),N));
fclose(fp);
}
```

## 5.2 Solving Equations

This C code, **newton.c**, applies Newton's method to solve the equation  $f(x) = 0$ .

```
# include <stdio.h>
# include <math.h> /* This inputs mathematical functions */
```

```

double f(double x)    /* Definition of f(x) using double precision numbers */
{
    double function;
    function = x*x-2;
    return(function);
}

double df(double x)   /* Definition of f'(x) = df(x) */
{
    double derivative;
    derivative = 2*x;
    return(derivative);
}

void main(void)       /* Newton's method for solving f(x)=0 */
{
    int    n=10, k=0;    /* Number of steps n = 10 */
    double x0=1;        /* Initial guess at solution, x0 */
    double x;
    FILE *fp;           /* Declares a pointer to a file */
    fp=fopen("newton.out","w"); /*Opens the file for output of data */
                                /* Next line prints to the file */
    fprintf(fp, "\nTable of Newton's iteration to solve f(x)=0\n");

        x=x0;
        fprintf(fp, "\n\nStep \t\t Next x \t\t\t\t\t f(x)\n");

    if (df(x) == 0)     /* Checks for division by zero */
        printf("Method fails: zero derivative for this x value!");
    else
        while (k<=n && df(x)!=0) /* Steps through the iteration */
        {
            /* Next line prints each result */
            fprintf(fp, "%2d \t\t\t%.10f \t\t\t%.10f\n", k, x, f(x));

            x = x-(f(x)/df(x)); /* Updates x using Newton */
            k++;                /* Proceeds to next step */
        }
        fclose(fp);           /* Closes the output file */
}
                                /* End of program */

```

This program outputs its results to the file **newton.out**, which for this example should look like:

Table of Newton's iteration to solve  $f(x)=0$

Step	Next x	f(x)
0	+1.0000000000	-1.0000000000
1	+1.5000000000	+0.2500000000
2	+1.4166666667	+0.0069444444
3	+1.4142156863	+0.0000060073
4	+1.4142135624	+0.0000000000
5	+1.4142135624	+0.0000000000
6	+1.4142135624	+0.0000000000
7	+1.4142135624	+0.0000000000
8	+1.4142135624	+0.0000000000
9	+1.4142135624	+0.0000000000
10	+1.4142135624	+0.0000000000

## 5.3 Numerical Integration

This C code, **simpson.c**, uses Simpson's Rule to obtain a numerical integration of a function  $f(x)$  over the interval  $[a, b]$ .

```
# include <stdio.h>
# include <math.h>

double f(double x)          /* Definition of f(x) */
{
    double function;
    function = x*x*x*x;
    return(function);      /* What the function gives when it is called */
}

void main(void)             /* Simpson's Rule for integral f(x) over [a,b] */
{
    int n=20;                /* Maximum number of steps (even) n */
    int c,k=1;              /* Counters in the algorithm */
    double a=0,x;           /* Lower limit x=a */
    double b=1;             /* Upper limit x=b */
    double h,SUM;
    FILE *fp; /* Declares a pointer to a file */
    fp=fopen("simpson.out","w"); /*Opens the file for output of data */
    /* Next 2 lines print title and table head to file */

    fprintf(fp, "\nSimpson's Rule for numerical integration of f(x)\n");
    fprintf(fp, "\n From a=%f to b=%f", a,b);
    fprintf(fp, "\n\nIntervals \t\tWidth \t\tSimpson Sum\n");

    SUM=f(a); /* Initial function value */
    c=2;
    h=(b-a)/n; /* Step size h=(b-a)/n */
    while (k <= n-1) /* Steps through the iteration */
    {
        c=6-c; /* gives the 4,2,4,2,... */
        SUM = SUM + c*f(a+k*h); /* Adds on the next area */
        k++; /* Increases k value by +1 */
    }
    fprintf(fp, "%2d \t\t%.10f \t\t%.10f\n", n, h, (SUM + f(b))*h/3);

    fclose(fp); /* Closes the output file */
} /* End of program */
```

Identify the function  $f(x)$  in the program and observe that the output is printed into a file called **simpson.out**, which for this example should look like:

Simpson's Rule for numerical integration

From a=0.000000 to b=1.000000

Intervals	Width	Simpson Sum
20	+0.0500000000	+0.2000008333

## 5.4 Solving ordinary differential equations

This C code, **euler.c**, is a simple application of Euler's method to solve the differential equation

$$y' = F(x, y)$$

with initial conditions

$$y(p) = q.$$

```
# include <stdio.h>
# include <math.h>

double f(double x, double y)      /* Definition of f(x) */
{
    double function;
    function = x+y;
    return(function);
}

void main(void)      /* Euler's method for y'= f(x,y) */
{
    int n=100;          /* Maximum number of steps (even) n */
    int s=5;           /* Step size for printing values */
    int k=0;           /* Counter in the algorithm */
    double p=0,q=1;    /* Initial values: x=p, y(p)=q */
    double h=0.05;     /* Step size */
    double x,y,u;      /* Variables */
    FILE *fp;          /* Declares a pointer to a file */
    fp=fopen("euler.out","w"); /*Opens the file for output of data */
                                /* Next 2 lines print title and table head to file */
    fprintf(fp, "\nEuler's method for solving y'= f(x,y) with step size %f\n",h);
    fprintf(fp, "\n\nStep \t\t x \t\t\t\t y(x)\n");
        x=p;
        y=q;
        while (k <= n)
            {
                if (k%s==0) /* Sets the steps between printout to multiples of s */
                {fprintf(fp, "%3d \t\t\t%.10f \t\t\t%.10f\n", k, x, y);}
                u=y+h*f(x,y); /* Linear approximation to next y */
                y=y+(h/2)*(f(x,y)+f(x+h,u)); /* Average of u and y from next x */
                k++; /* Increments counter k by 1 */
                x=x+h; /* Increments x by the step size h */
            }
        fclose(fp); /* Closes the output file */
} /* End of program */
```

The output file, **euler.out**, should look like this:

Euler's method for solving y'= f(x,y) with step size 0.050000

Step	x	y(x)
0	+0.0000000000	+1.0000000000
5	+0.2500000000	+1.3177931720
10	+0.5000000000	+1.7967808871
15	+0.7500000000	+2.4827257257
20	+1.0000000000	+3.4343821087
25	+1.2500000000	+4.7271846364
30	+1.5000000000	+6.4579835345
35	+1.7500000000	+8.7511244774
40	+2.0000000000	+11.7662544517
45	+2.2500000000	+15.7083436786
50	+2.5000000000	+20.8405527251
55	+2.7500000000	+27.5007525451
60	+3.0000000000	+36.1227345026
65	+3.2500000000	+47.2634418488
70	+3.5000000000	+61.6379321227

75	+3.7500000000	+80.1642652574
80	+4.0000000000	+104.0211352667
85	+4.2500000000	+134.7218633706
90	+4.5000000000	+174.2093975175
95	+4.7500000000	+224.9782819446
100	+5.0000000000	+290.2312534822

## 5.5 Solving partial differential equations

### Heat equation

This C program solves the heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

for boundary values

$$u(t, 0) = u(t, 1) = 0$$

and initial conditions

$$u(0, x) = 2x \quad 0 \leq x \leq \frac{1}{2}$$

$$u(0, x) = 2(1 - x) \quad \frac{1}{2} \leq x \leq 1$$

```
#include <stdio.h>
#include <math.h>

void main(void)          /* Numerical solution of heat equation */
{                        /* CTJ Dodson */
  int x,t;               /* Counters in array for solution */
  double u[21][11];
  FILE *fp;             /* File pointer for output */

  for (t=1;t<=20;t++)   /* Boundary values: u(t,0)=u(t,1)=0 */
    u[t][0]=u[t][10]=0;

  for (x=0;x<=5;x++)    /* Initial values */
    u[0][x]=0.2*x;
  for (x=6;x<=10;x++)  /* Initial values */
    u[0][x]=2-0.2*x;

  for (t=0;t<=19;t++)   /* Explicit difference equation approximation */
    for (x=1;x<=9;x++)
      u[t+1][x]=0.1*u[t][x-1]+0.8*u[t][x]+0.1*u[t][x+1]; /* dt=0.001, dx=0.1 */

  fp=fopen("heat.out","w"); /* Open output file */
  fprintf(fp, "\n\tNumerical solution u(t,x) to heat equation du/dt=d^2u/dx^2 for rod\n");
  fprintf(fp, "Time step dt=0.001 s \t Space step dx=0.1\n");
  fprintf(fp, "t sec \t Position x->");
  for (t=0;t<=20;t++)
    fprintf(fp,          /* You can break a line, but not inside a format list*/
            "\n%1.3f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f",0.001*t,
            u[t][0],u[t][1],u[t][2],u[t][3],u[t][4],u[t][5],u[t][6],u[t][7],u[t][8],u[t][9],u[t][10]);
    fclose(fp);          /* Close output file */
}
```

The output file for this program should look something like this:

```
Numerical solution u(t,x) to heat equation du/dt=d^2u/dx^2 for rod
Time step dt=0.001 s      Space step dx=0.1
t sec      Position x->
0.000 0.0000 0.2000 0.4000 0.6000 0.8000 1.0000 0.8000 0.6000 0.4000 0.2000 0.0000
```

```

0.001 0.0000 0.2000 0.4000 0.6000 0.8000 0.9600 0.8000 0.6000 0.4000 0.2000 0.0000
0.002 0.0000 0.2000 0.4000 0.6000 0.7960 0.9280 0.7960 0.6000 0.4000 0.2000 0.0000
0.003 0.0000 0.2000 0.4000 0.5996 0.7896 0.9016 0.7896 0.5996 0.4000 0.2000 0.0000
0.004 0.0000 0.2000 0.4000 0.5986 0.7818 0.8792 0.7818 0.5986 0.4000 0.2000 0.0000
0.005 0.0000 0.2000 0.3998 0.5971 0.7732 0.8597 0.7732 0.5971 0.3998 0.2000 0.0000
0.006 0.0000 0.2000 0.3996 0.5950 0.7643 0.8424 0.7643 0.5950 0.3996 0.2000 0.0000
0.007 0.0000 0.1999 0.3992 0.5924 0.7551 0.8268 0.7551 0.5924 0.3992 0.1999 0.0000
0.008 0.0000 0.1999 0.3986 0.5893 0.7460 0.8125 0.7460 0.5893 0.3986 0.1999 0.0000
0.009 0.0000 0.1998 0.3978 0.5859 0.7370 0.7992 0.7370 0.5859 0.3978 0.1998 0.0000
0.010 0.0000 0.1996 0.3968 0.5822 0.7281 0.7867 0.7281 0.5822 0.3968 0.1996 0.0000
0.011 0.0000 0.1993 0.3956 0.5783 0.7194 0.7750 0.7194 0.5783 0.3956 0.1993 0.0000
0.012 0.0000 0.1990 0.3942 0.5741 0.7108 0.7639 0.7108 0.5741 0.3942 0.1990 0.0000
0.013 0.0000 0.1986 0.3927 0.5698 0.7025 0.7533 0.7025 0.5698 0.3927 0.1986 0.0000
0.014 0.0000 0.1982 0.3910 0.5653 0.6943 0.7431 0.6943 0.5653 0.3910 0.1982 0.0000
0.015 0.0000 0.1977 0.3892 0.5608 0.6863 0.7333 0.6863 0.5608 0.3892 0.1977 0.0000
0.016 0.0000 0.1970 0.3872 0.5562 0.6784 0.7239 0.6784 0.5562 0.3872 0.1970 0.0000
0.017 0.0000 0.1963 0.3851 0.5515 0.6708 0.7148 0.6708 0.5515 0.3851 0.1963 0.0000
0.018 0.0000 0.1956 0.3828 0.5468 0.6632 0.7060 0.6632 0.5468 0.3828 0.1956 0.0000
0.019 0.0000 0.1948 0.3805 0.5420 0.6559 0.6975 0.6559 0.5420 0.3805 0.1948 0.0000
0.020 0.0000 0.1939 0.3781 0.5373 0.6487 0.6891 0.6487 0.5373 0.3781 0.1939 0.0000

```

### Flow equation

This C program solves the flow equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0$$

for boundary values

$$u(t, 0) = 2t$$

and initial conditions

$$u(0, x) = x(x - 2) \quad 0 \leq x \leq 2$$

$$u(0, x) = 2(x - 2) \quad 2 \leq x \leq 1$$

```

#include <stdio.h>
#include <math.h>

void main(void)          /* Numerical solution of flow equation */
{
    int x,t;              /* CTJ Dodson */
    double u[11][10];    /* Counters in solution array */
    FILE *fp;            /* Array to hold solution values u(t,x) */
    for (x=0;x<=2;x++)   /* Initial values u(0,x) */
        u[0][x]=0.25*x*(0.25*x-2);
    for (x=3;x<=9;x++)   /* Initial values u(0,x) */
        u[0][x]=2*(0.25*x-2);

    for (t=0;t<=10;t++)  /* Boundary values u(t,0)=2t */
        u[t][0]=0.25*t;

    for (t=0;t<=9;t++)   /* Explicit difference equation approximation */
        for (x=1;x<=9;x++)

            u[t+1][x]=0.375*u[t][x-1]+0.75*u[t][x]-0.125*u[t][x+1];

    fp=fopen("flow.out","w"); /* Open output file */
    fprintf(fp, "\n\tNumerical solution u(t,x) to fluid flow equation: du/dt+du/dx=0\n");
    fprintf(fp, "\tBoundary u(t,0)=2t\n Time step dt=0.125 s\t Space step dx=0.25\n");
    fprintf(fp, "t sec\tPosition x->");
    for (t=0;t<=10;t++)   /* Print values u(t,x) */
        fprintf(fp, /* You can break a line, but not inside a format list*/
            "\n%2.3f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f",
            t*0.125, u[t][0],u[t][1],u[t][2],u[t][3],u[t][4],u[t][5],u[t][6],u[t][7],u[t][8],u[t][9]);
}

```

```
fclose(fp);
/* Close output file */
}
```

The output file for this program should look something like this:

```

Numerical solution u(t,x) to fluid flow equation: du/dt+du/dx=0
Boundary u(t,0)=2t
Time step dt=0.125 s   Space step dx=0.25
t sec   Position x->
0.000 +0.0000 -0.4375 -0.7500 -2.5000 -2.0000 -1.5000 -1.0000 -0.5000 +0.0000 +0.5000
0.125 +0.2500 -0.2344 -0.4141 -1.9062 -2.2500 -1.7500 -1.2500 -0.7500 -0.2500 +0.3438
0.250 +0.5000 -0.0303 -0.1602 -1.3037 -2.1836 -2.0000 -1.5000 -1.0000 -0.5117 +0.1016
0.375 +0.7500 +0.1848 +0.0315 -0.7649 -1.8766 -2.1313 -1.7500 -1.2485 -0.7715 -0.2095
0.500 +1.0000 +0.4159 +0.1885 -0.3273 -1.4279 -2.0835 -1.9557 -1.4962 -1.0206 -0.5714
0.625 +1.2500 +0.6634 +0.3383 +0.0037 -0.9332 -1.8536 -2.0610 -1.7280 -1.2551 -0.9675
0.750 +1.5000 +0.9240 +0.5020 +0.2463 -0.4668 -1.4825 -2.0249 -1.9120 -1.4684 -1.3838
0.875 +1.7500 +1.1927 +0.6922 +0.4313 -0.0724 -1.0338 -1.8356 -2.0098 -1.6453 -1.8073
1.000 +2.0000 +1.4643 +0.9125 +0.5921 +0.2367 -0.5731 -1.5132 -1.9900 -1.7617 -2.2224
1.125 +2.2500 +1.7341 +1.1595 +0.7567 +0.4712 -0.1519 -1.1010 -1.8397 -1.7897 -2.6087
1.250 +2.5000 +1.9994 +1.4253 +0.9434 +0.6561 +0.2004 -0.6528 -1.5690 -1.7061 -2.9402

```

## Wave equation

This C program solves the wave equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

with boundary values

$$u(t,0) = u(t,1) = 0$$

and initial conditions

$$\frac{\partial u}{\partial t} \Big|_{t=0} = 0 \quad u(0,x) = \frac{1}{8} \sin \pi x \quad 0 \leq x \leq 1$$

```

#include <stdio.h>
#include <math.h>

void main(void)          /* Numerical solution of wave equation */
{
    /* CTJ Dodson */
    int x,t;             /* Counters in solution array */
    double u[21][11];   /* Array to hold solution values u(t,x) */
    FILE *fp;

    for (t=1;t<=20;t++) /* Boundary values u(t,0)=u(t,1)=0 */
        u[t][0]=u[t][10]=0;

    for (x=0;x<=10;x++) /* Initial values u(0,x) */
        u[0][x]=0.125*sin(0.1*x*3.14159);
    for (x=0;x<=8;x++)
        u[1][x+1]=0.5*(u[0][x]+u[0][x+2]); /* Initial flow: du/dt = 0 at t=0 for all x */

    for (t=1;t<=19;t++) /* Explicit difference equation approximation */
        for (x=1;x<=9;x++)
            u[t+1][x]=u[t][x-1]+u[t][x+1]-u[t-1][x];

    fp=fopen("wave.out","w"); /* Open output file */
    fprintf(fp, "\n\tNumerical solution u(t,x) to wave equation: d^2u/dt^2=d^2u/dx^2\n");
    fprintf(fp, "\tInitial rate: du/dt = 0 at t=0 for all x\n");
    fprintf(fp, "Time step dt=0.1 s\t Space step dx=0.1\n");
    fprintf(fp, "t sec\tPosition x->");

```

```

for (t=0;t<=20;t++)          /* Print values u(t,x) */
fprintf(fp,                 /* You can break a line, but not inside a format list*/
"\n%1.2f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f %2.4f",
t*0.1,u[t][0],u[t][1],u[t][2],u[t][3],u[t][4],u[t][5],u[t][6],u[t][7],u[t][8],u[t][9],u[t][10]);
fclose(fp);
/* Close output file */
}

```

The output file for this program should look something like this:

```

Numerical solution u(t,x) to wave equation: d^2u/dt^2=d^2u/dx^2
Initial rate: du/dt = 0 at t=0 for all x
Time step dt=0.1 s   Space step dx=0.1
t sec   Position x->
0.00 +0.0000 +0.0386 +0.0735 +0.1011 +0.1189 +0.1250 +0.1189 +0.1011 +0.0735+0.0386 +0.0000
0.10 +0.0000 +0.0367 +0.0699 +0.0962 +0.1131 +0.1189 +0.1131 +0.0962 +0.0699+0.0367 +0.0000
0.20 +0.0000 +0.0312 +0.0594 +0.0818 +0.0962 +0.1011 +0.0962 +0.0818 +0.0594+0.0312 +0.0000
0.30 +0.0000 +0.0227 +0.0432 +0.0594 +0.0699 +0.0735 +0.0699 +0.0594 +0.0432+0.0227 +0.0000
0.40 +0.0000 +0.0119 +0.0227 +0.0313 +0.0367 +0.0386 +0.0367 +0.0312 +0.0227+0.0119 +0.0000
0.50 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000+0.0000 +0.0000
0.60 +0.0000 -0.0119 -0.0227 -0.0312 -0.0367 -0.0386 -0.0367 -0.0312 -0.0227-0.0119 +0.0000
0.70 +0.0000 -0.0227 -0.0432 -0.0594 -0.0699 -0.0735 -0.0699 -0.0594 -0.0432-0.0227 +0.0000
0.80 +0.0000 -0.0312 -0.0594 -0.0818 -0.0962 -0.1011 -0.0962 -0.0818 -0.0594-0.0312 +0.0000
0.90 +0.0000 -0.0367 -0.0699 -0.0962 -0.1131 -0.1189 -0.1131 -0.0962 -0.0699-0.0367 +0.0000
1.00 +0.0000 -0.0386 -0.0735 -0.1011 -0.1189 -0.1250 -0.1189 -0.1011 -0.0735-0.0386 +0.0000
1.10 +0.0000 -0.0367 -0.0699 -0.0962 -0.1131 -0.1189 -0.1131 -0.0962 -0.0699-0.0367 +0.0000
1.20 +0.0000 -0.0312 -0.0594 -0.0818 -0.0962 -0.1011 -0.0962 -0.0818 -0.0594-0.0312 +0.0000
1.30 +0.0000 -0.0227 -0.0432 -0.0594 -0.0699 -0.0735 -0.0699 -0.0594 -0.0432-0.0227 +0.0000
1.40 +0.0000 -0.0119 -0.0227 -0.0312 -0.0367 -0.0386 -0.0367 -0.0312 -0.0227-0.0119 +0.0000
1.50 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000 +0.0000+0.0000 +0.0000
1.60 +0.0000 +0.0119 +0.0227 +0.0313 +0.0367 +0.0386 +0.0367 +0.0312 +0.0227+0.0119 +0.0000
1.70 +0.0000 +0.0227 +0.0432 +0.0594 +0.0699 +0.0735 +0.0699 +0.0594 +0.0432+0.0227 +0.0000
1.80 +0.0000 +0.0312 +0.0594 +0.0818 +0.0962 +0.1011 +0.0962 +0.0818 +0.0594+0.0312 +0.0000
1.90 +0.0000 +0.0367 +0.0699 +0.0962 +0.1131 +0.1189 +0.1131 +0.0962 +0.0699+0.0367 +0.0000
2.00 +0.0000 +0.0386 +0.0735 +0.1011 +0.1189 +0.1250 +0.1189 +0.1011 +0.0735+0.0386 +0.0000

```

## 6 Assignments

The course is for a 1-semester class or individual use, in a computer lab. The assignments are meant to be done in the computer lab sessions. Your reports on these assignments and the on-line materials are your course notes and you may wish to refer to them in later courses, projects or work so make them as useful as possible, by adding any clarifying notes on the programs for example. You should include also in your report any formulae that are used.

### 6.1 Assignment 1: Wordprocessing and Spreadsheet

1. Using one of the wordprocessing packages, create a curriculum vitae for yourself.
2. Look through the BASIC programs and the C programs and prepare a list of at least ten corresponding terms for these two languages and their meanings. For example:

```

BASIC:   FOR I = 0 TO N
C       :   for (N = 1000; N <= 10000; N = N + 1000)
Meaning: step through the values of the indicated variable

```

3. The **variance**,  $Var(x)$ , of a random variable  $x$  is the difference between the mean square value,  $\bar{x}^2$ , and the square of the mean value,  $\bar{x}^2$ . However, for a finite sample of  $n$  datapoints, it is normal to estimate the variance of the parent population as

$$Var^{est}(x) = \frac{n}{n-1}[\bar{x}^2 - \bar{x}^2].$$

The square root of the variance is the **standard deviation** and the ratio of the standard deviation to the mean is the **coefficient of variation**. The variance, standard deviation and coefficient of variation are all used as measures of ‘spread’ in a collection of data.

Using a spreadsheet package, first correct the obvious errors in the following table of grouped data. Then find the mean  $\bar{x}$ , variance  $Var^{est}(x)$ , standard deviation  $sd(x)$  and coefficient of variation  $cv(x)$ .

Class range	Class centre	Frequency
40-42	51	12
42-44	43	29
44-46	46	34
46-48	47	27
48-50	49	15
50-52	51	11

Plot a histogram of this data and include this as an Appendix to a 1-page document which contains the data and the formulae you used, as well as the values you found. The document should be titled **Spreadsheet Exercise 1** and have your name and date.

## 6.2 Assignment 2: Mathematica for algebra, calculus and graphics

Read first the file at Beginning Mathematica to get started. Download the file TRYMMA.nb from Mathematica Notebooks and try the functions there by selecting the inputs, modifying them and keying **Shift-Enter**.

Open a new NoteBook and do the following.

### Exercise

Investigate the function

$$f(x) = x^{1/x}$$

by plotting its graph for  $x \geq 0$  over successively smaller intervals to estimate

1. the limit of  $f(x)$  as  $x \rightarrow 0$
2. the values of  $x$  at which  $f(x)$  has maximum and minimum values and what these values are to 4 decimal places

Use Mathematica operations to solve analytically the above problems.

Try investigating

$$g(x) = x^{\sin x}$$

for  $x \geq 0$  in a similar way.

Give your NoteBook a title and your name using the Style item under Format.

## 6.3 Assignment 3: BASIC Programs for Numerical Analysis

1. Use Newton’s method 2 to find

$$\sqrt{2}, \sqrt{3}, 2^{1/23}$$

to 5 decimal places. Try doing this using a spreadsheet and by using the previously given BASIC program 4.1 with appropriate modifications (first try the program as it stands to clarify what it is doing):

```

10  REM NEWTON'S METHOD FOR SOLVING F(X)=0
110 REM THE FUNCTION F(X) AND ITS DERIVATIVE G(X) MUST BE INPUT
120 REM THE INITIAL GUESS XO MUST BE INPUT
200  INPUT "INITIAL GUESS XO"; X
210  N = 0
220  PRINT " N", "      X": PRINT
300 REM NEWTON'S ITERATION LOOP
310
320  PRINT N, X
330  F(X) = X ^ 2 - 2
340  G(X) = 2 * X
350  XNEW = X - F(X) / G(X)
360  X = XNEW: N = N + 1
370  IF INKEY$ = "" THEN 370
380  GOTO 320
390 REM PRESS CTRL-BREAK TO STOP
400 END

```

2. In a unit circle, the area of the inscribed n-sided regular polygon is

$$A_n = \frac{n}{2} \sin(2\pi/n)$$

and the area of the exscribed n-sided regular polygon is

$$B_n = n \tan(\pi/n)$$

Tabulate these functions for  $n = 3, 30, 300, 3000$ , and compare how closely the two polygonal areas approach the true area of the unit circle.

3. Tabulate the function

$$f(x) = x^{1/x} \text{ for } 0 < x \leq 10,$$

and try to identify its maximum, using Excel and using one of the previously given BASIC programs 4.3 with appropriate modifications (first try each program as it stands to clarify what it is doing):

1 Output to a file:

```

10  REM TABULATE FUNCTION F(X) TO A FILE TAB.OUT
100  :
110  DEF FNF (X) = X ^ 2
120  :
130  INPUT "FROM X= "; A
140  INPUT "  TO X= "; B
150  INPUT "IN STEPS OF "; DX
200  OPEN "TAB.OUT" FOR OUTPUT AS #1
205  PRINT #1, " X          F(X)"
210  FOR X = A TO B STEP DX
220    Y = FNF(X)
230    PRINT #1, X, Y
240  NEXT X
245  CLOSE #1
300  :
310  END

```

And

2 Output to screen:

```

10  REM TABULATE F(X) FROM X=A TO X=B IN STEPS DX TO SCREEN
100 REM ENTER THE FUNCTION F(X) IN THE NEXT LINE
110 DEF FN F(X)=X^2
120 :
130 INPUT "FROM X= ";A
140 INPUT " TO X= ";B
150 INPUT "IN STEPS OF ";DX
200 :
210 FOR X=A TO B STEP DX
220   Y=FN F(X)
230   PRINT X,Y
240 NEXT X
300 :
310 END

```

4. Write a BASIC program to tabulate the function of two variables,

$$f(x, y) = \sin xy,$$

over the grid of values

$$0 \leq x \leq \pi, \quad 0 \leq y \leq \pi, \quad \text{in steps of } \pi/4.$$

5. In your report on this assignment, include the programs you have used with their various modifications as an Appendix, referring to these programs when you describe the results in the main body of the report. Add any further notes of your own to clarify the programs—remember, these reports and the on-line materials are your course notes.

## 6.4 Assignment 4: BASIC Programs for Integration; Web Authoring

1. Plot a graph of  $e^{-x^2}$  over the interval  $[-3, 3]$ . This function is used in the Normal, or Gaussian distribution, commonly encountered for representing distributions of student marks and other data. It has been chosen also because it has no indefinite integral in terms of elementary functions.
2. By inspection, roughly estimate the areas  $A(x)$  under the graph over the subintervals  $[0, x]$ , for:

$$x = 0.2, \quad x = 0.4, \quad x = 0.6, \quad \dots, \quad x = 3.0$$

and plot these areas. This gives us an idea of what the numerical estimates should be like.

3. Read again about numerical integration at 2. Use the BASIC program for Simpson's Rule 4.4 to obtain better numerical estimates of the areas  $A(x)$  in the previous part, and plot these on the same graph. You might note that the true value of the area under the curve  $y = e^{-x^2}$  is  $\sqrt{\pi}$ , when taken over the whole  $x$ -axis.
4. Download the source code for

`<a href="wwwpage.html">wwwpage.html</a>`

and use it to create a web page of your own. You should scan in a suitable small photo or graphic as a **mypic.gif** file to replace the **ankle.gif** example and incorporate that in your html file, together with suitable text.

Note that in DOS the file extension **\*.html** will be abbreviated to **\*.htm**.

## 6.5 Assignment 5: C Programs

1. Consider the C program

```

#include <stdio.h>
#include <math.h>

void main()
{
    float N;
        printf("\t\t N \t(1+1/N)^N\n");
    for (N = 1000; N <= 10000; N = N + 1000)
        printf("\t\t%5.0f \t%1.9f\n", N, pow((1+1/N),N));
}

```

- (i) Explain what each line of this C program does.
- (ii) Report the output that the program generates. Hint: Copy it from the web, compile and run it!
- (iii): Write a short BASIC program to perform the same task as the above C program. Hint: You should test it and give the output!.

2. You have a short BASIC program to print to a file a table of values of

$$f(x, y) = \sin xy,$$

over the grid of values

$$0 \leq x \leq \pi, \quad 0 \leq y \leq \pi,$$

in steps of  $\pi/4$ . Write a short C program to do the same task; report both programs and their outputs.

## 6.6 Assignment 6: Initial Value Problems

This assignment involves the study of a differential equation which represents the competition between growth and decay of a quantity,  $y(x)$ , which could represent the fraction of a region occupied by an organism, or the fraction of a chemical present in a reaction, at time  $x$ . We obtain numerical solutions using the Euler method 2, by means of the two programs: using BASIC 4.5 and using C 5.4. As always with computer experiments in mathematics, it is important to have an idea of what a solution should look like before we use numerical programs; this is the first part of the assignment.

The differential equation is

$$\frac{dy}{dx} = \frac{1-y}{a} - \frac{y}{b} \quad \text{with } 0 \leq y \leq 1$$

where  $a$  is the rate constant for creation of the quantity represented by  $y$  and  $b$  is the rate constant for decay of this quantity. This differential equation becomes an **initial value problem** when we specify the value of  $y(0)$ ; that is, the initial value of  $y$ .

1. Take the case  $a = 1000$  and  $b = 1$ , and write down an approximate analytic solution to the initial value problem with  $y(0) = 0.8$ .

*Hint: If  $a = 1000 \times b$ , then the differential equation becomes approximately*

$$\frac{dy}{dx} = -\frac{y}{1}$$

and so

$$\frac{dy}{y} = -dx$$

2. Take the case  $a = 1$  and  $b = 1000$ , and write down an approximate analytic solution to the initial value problem with  $y(0) = 0.8$ .
3. Take the case  $a = 1$  and  $b = 2$ , and use the BASIC program 4.5 for Euler's method to obtain solutions for the four cases:

$$y(0) = 0.2, \quad 0.4, \quad 0.6, \quad 0.8$$

using time steps of size 0.2 for  $0 \leq x \leq 10$ . Plot your four solutions on the same graph. Include in your report the modified BASIC program that you use for the case  $y(0) = 0.2$ , and append the output from this program.

What is the final steady state of the quantity  $y$  as  $x$  becomes large?

4. Modify the C program given previously on page 5.4 to solve the case for  $y(0) = 0.2$ ; include in your report this program and append the output.

## 6.7 Assignment 7: Practise Problems

1. You have a short BASIC program to print to a file a table of values of

$$f(x, y) = \sin xy,$$

over the grid of values

$$0 \leq x \leq \pi, \quad 0 \leq y \leq \pi,$$

in steps of  $\pi/4$ . Write a short C program to do the same task.

2. Repeat the above using Mathematica to create the table of values.
3. Write a BASIC program to perform the same task as the C program given on page 5.1 to convert temperature values.
4. Create a Mathematica notebook with functions to convert temperature values from Centigrade to Fahrenheit and conversely. Use the functions to create tables and graphs for these conversions.
5. Repeat the above using Mathematica.
6. Find the mean and standard deviation for the following grouped data:

Class range	Class centre	Frequency
30-32	31	10
32-34	33	37
34-36	35	44
36-38	37	34
38-40	39	25
40-42	41	12

7. (i) Explain what each line of this BASIC program does. (ii) Write down the output that the program generates:

```

10 OPEN "E.OUT" FOR OUTPUT AS #1
20 PRINT #1, " N          (1+1/N)^N"
30 FOR N = 1000 TO 10000 STEP 1000
40 PRINT #1, N, (1+1/N)^N
50 NEXT N
60 CLOSE #1
70 END

```

8. Write a short C program to perform the same task as the above BASIC program.
9. Explain what each line of this html file does.

```

<HTML>
<HEAD>
<TITLE> Fred's page </TITLE>
</HEAD>
<BODY>
<H1> Fred's page </H1>
Hello! Welcome to the homepage of Fred!
<HR>
<P>
Here is a recent picture of Fred <IMG SRC = "ankle.gif" ALIGN="MIDDLE">
<HR>
I enjoy writing web pages, and also:
<UL>
  <LI> Playing Doom
  <LI> Playing Quake
  <LI> Writing C programs
</UL>
<HR>
<P>
The source file for this page available via the menu bar.
<P>
You can contact Fred by email at:
<a href="mailto:ctdodson@manchester.ac.uk">ctdodson@manchester.ac.uk</a></H4>

```

10. Using Mathematica:

- Find the derivative of  $(\sin x)^x$ .
- Plot  $(\sin x)^x$  over the interval  $[0, \pi]$ . Estimate to 4 decimal places the value of  $x$  at the local minimum near  $x = 0.4$ .
- Find to 4 decimal places the area under the graph of  $(\sin x)^x$  over the interval  $[0, \pi]$ .

11. (i) Explain what each line of this C program does.  
(ii) Write down the output that the program generates [Hint: copy it, compile and run it!]:

```

#include <stdio.h>
#include <math.h>

void main()
{
  float N;
  printf("\t  N \t(1+1/N)^N\n");
  for (N = 1000; N <= 10000; N = N + 1000)
    printf("\t%5.0f \t%1.9f\n", N, pow((1+1/N),N));
}

```

12. Write a short BASIC program to perform the same task as the above C program [you should test it!].

13. Correct the obvious error in the following table of grouped data, then find the mean and standard deviation; state the formulae that you use:

Class range	Class centre	Frequency
80-82	81	100
82-84	83	370
84-86	84	440
86-88	87	340
88-90	89	250
90-92	91	120

14. Using HTML, write down the necessary commands to produce a webpage containing:
- (a) A title.
  - (b) A list of three items (short lines of text).
  - (c) A vertical sequence of three graphics, called `1.tif`, `2.tif`, `3.tif`.
  - (d) A hyperlink to your email address.

## 6.8 Final Test

This could take the following form:

1. Use a spreadsheet to compute statistics of given data.
2. Use Mathematica for calculus, algebra, numerical work and graphics.
3. Read a simple BASIC program and annotate it.
4. Make a simple modification to the BASIC program, run and report results.
5. Read a simple C program and annotate it.
6. Make a simple modification to the C program, run and report results.

It could be an open book test allowing use of any materials you wish to bring or can find on the computer.

## References

- [1] V.I. Arnol'd. **Ordinary Differential Equations** Springer-Verlag, Berlin-Heidelberg-New York, 1992.
- [2] B.V. Cordingley and D.J. Chamund. **Advanced BASIC scientific subroutines** Macmillan, Basingstoke, 1988.
- [3] I.A. Graham. **HTML Sourcebook** John Wiley, New York 1995.
- [4] K. Jamsa. **Jamsa's 1001 C/C++ tips** Jamsa Press, Las Vegas, 1993.
- [5] B.W. Kerninghan and D.M. Ritchie. **The C programming language** Prentice Hall, Englewood Cliffs NJ, 1988.
- [6] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling. **Numerical Recipes in C: The Art of Scientific Computing** Cambridge University Press, Cambridge, 1986.
- [7] G.D. Smith. **Numerical Solution of Partial Differential Equations: Finite Difference Methods** 3<sup>rd</sup> Edition, Oxford University Press, Oxford, 1985.
- [8] S. Wolfram. **The Mathematica Book** Cambridge University Press, Cambridge 1996.