Chapter 4

A.p.D.Hana Hassan Kareem

Root Approximations

This chapter is an introduction to Newton's and bisection methods for approximating roots of linear and non–linear equations. Several examples are presented to illustrate practical solutions using MATLAB and Excel spreadsheets.

4.1 Newton's Method for Root Approximation

Newton's (or Newton–Raphson) method can be used to approximate the roots of any linear or non–linear equation of any degree. This is an iterative (repetitive procedure) method and it is derived with the aid of Figure 4.1.

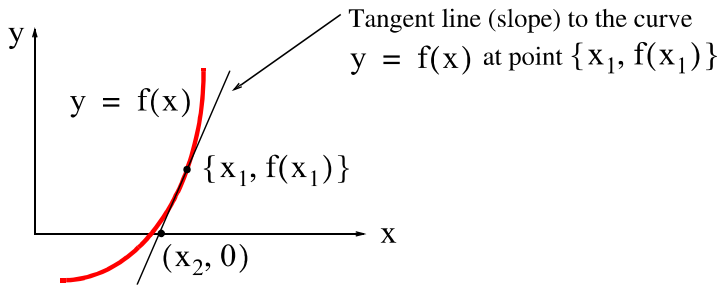


Figure 2.1. Newton's method for approximating real roots of a function

We assume that the slope is neither zero nor infinite. Then, the slope (first derivative) at $x = x_1$ is

$$f'(x_1) = \frac{y - f(x_1)}{x - x_1}$$

$$y - f(x_1) = f'(x_1)(x - x_1) \tag{4.1}$$

The slope crosses the $x-$axis at $x = x_2$ and $y = 0$. Since this point $[x_2, f(x_2)] = (x_2, 0)$ lies on the slope line, it satisfies (4.1). By substitution,

$$0 - f(x_1) = f'(x_1)(x_2 - x_1)$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{4.2}$$

and in general,

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}} \tag{4.3}$$

Example 4.1

Use Newton's method to approximate the positive root of

$$f(x) = x^2 - 5 \tag{4.4}$$

to four decimal places.

Solution:

As a first step, we plot the curve of (4.4) to find out where it crosses the $x-$axis. This can be done easily with a simple plot using MATLAB or a spreadsheet. We start with MATLAB and will discuss the steps for using a spreadsheet afterwards.

We will now introduce some new MATLAB functions and review some which are discussed in Chapter 1.

input('string'): It displays the text **string**, and waits for an input from the user. We must enclose the text in single quotation marks.

We recall that the **polyder(p)** function displays the row vector whose values are the coefficients of the first derivative of the polynomial **p**. The **polyval(p,x)** function evaluates the polynomial **p** at some value **x**. Therefore, we can compute the next iteration for approximating a root with Newton's method using these functions. Knowing the polynomial **p** and the first approximation x_0, we can use the following script for the next approximation x_1.

```
q=polyder(p)
x1=x0–polyval(p,x0)/polyval(q,x0)
```

We've used the **fprintf** command in Chapter 1; we will use it many more times. Therefore, let us review it again.

The following description was extracted from the **help fprintf** function.

It formats the data in the real part of matrix A (and in any additional matrix arguments), under control of the specified format string, and writes it to the file associated with file identifier **fid** *and contains C language conversion specifications. These specifications involve the character* **%**, *optional flags, optional width and precision fields, optional subtype specifier, and conversion characters* **d, i, o, u, x, X, f, e, E, g, G, c**, *and* **s**. *See the Language Reference Guide or a C manual for complete details. The special formats* **\n,\r,\t,\b,\f** *can be used to produce linefeed, carriage return, tab, backspace, and formfeed characters respectively. Use* \\ *to produce a backslash character and* **%%** *to produce the percent character.*

To apply Newton's method, we must start with a reasonable approximation of the root value. In all cases, this can best be done by plotting $f(x)$ versus x with the familiar statements below. The following two lines of script will display the graph of the given equation in the interval $-4 \le x \le 4$.

```
x=linspace(–4, 4, 100);        % Specifies 100 values between -4 and 4
y=x .^ 2 – 5; plot(x,y); grid  % The dot exponentiation is a must
```

We chose this interval because the given equation asks for the square root of 5 ; we expect this value to be a value between 2 and 3 . For other functions, where the interval may not be so obvious, we can choose a larger interval, observe the x-axis crossings, and then redefine the interval. The plot is shown in Figure 4.2.

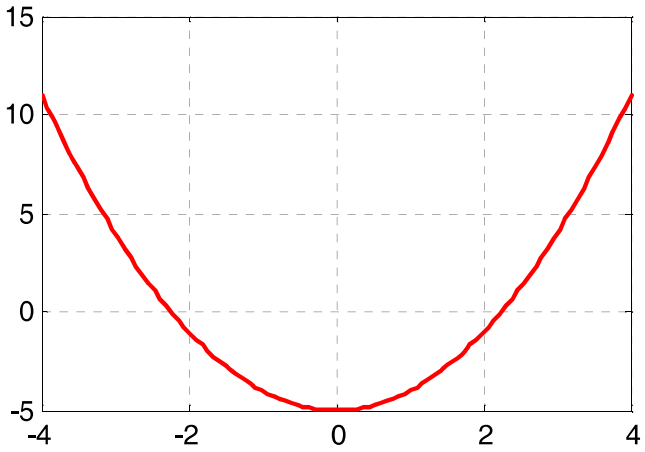


Figure 4 .2. Plot for the curve of Example 2.1

As expected, the curve shows one crossing between $x = 2$ and $x = 3$, so we take $x_0 = 2$ as our first approximation, and we compute the next value x_1 as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{(2)^2 - 5}{2(2)} = 2 - \frac{(-1)}{4} = 2.25 \tag{4.5}$$

The second approximation yields

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 2.25 - \frac{(2.25)^2 - 5}{2(2.25)} = 2.25 - \frac{0.0625}{4.5} = 2.2361 \tag{4.6}$$

We will use the following MATLAB script to verify (4.5) and (4.6).

```
%  Approximation of a root of a polynomial function p(x)
%  Do not forget to enclose the coefficients in brackets [ ]
p=input('Enter coefficients of p(x) in descending order: ');
x0=input('Enter starting value: ');
q=polyder(p);                          %  Calculates the derivative of p(x)
x1=x0–polyval(p,x0)/polyval(q,x0);
fprintf('\n');                         % Inserts a blank line
%
%  The next function displays the value of x1 in decimal format as indicated
%  by the specifier %9.6f, i.e., with 9 digits where 6 of these digits
%  are to the right of the decimal point such as xxx.xxxxxx, and
```

```
%  \n prints a blank line before printing x1
fprintf('The next approximation is: %9.6f \n', x1)
fprintf('\n');                          % Inserts another blank line
%
fprintf('Rerun the program using this value as your next....
approximation \n');
```

The following lines show MATLAB's inquiries and our responses (inputs) for the first two approximations.

```
Enter coefficients of P(x) in descending order:
[1  0 –5]
Enter starting value: 2
The next approximation is:  2.250000
Rerun the program using this value as your
next approximation
Enter polynomial coefficients in
descending order: [1  0 –5]
Enter starting value: 2.25
The next approximation is:  2.236111
```

We observe that this approximation is in close agreement with (4.6).

In Chapter 1 we discussed *script files* and *function files*. We recall that a function file is a user–defined function using MATLAB. We use function files for repetitive tasks. The first line of a function file must contain the word *function* followed by the output argument, the equal sign (=), and the input argument enclosed in parentheses. The function name and file name must be the same but the file name must have the extension ***.m***. For example, the function file consisting of the two lines below

```
function y = myfunction(x)
y=x .^ 3 + cos(3 .* x)
```

is a function file and must be saved as ***myfunction.m***

We will use the **while end** loop, whose general form is

while *expression*
    commands ...
end

where the *commands ...* in the second line are executed as long as all elements in *expression* of the first line are true.

We will also be using the following commands:

disp(x): Displays the array *x* without printing the array name. If *x* is a string, the text is displayed. For example, if `v = 12`, **disp(v)** displays `12`, and **disp('volts')** displays `volts`.

sprintf(format,A): Formats the data in the real part of matrix **A** under control of the specified *format* string. For example,

```
sprintf('%d',round(pi))

ans =
3
```

where the format script **%d** specifies an integer. Likewise,

```
sprintf('%4.3f',pi)

ans =
3.142
```

where the format script **%4.3f** specifies a fixed format of 4 digits where 3 of these digits are allocated to the fractional part.

Example 4.2

Approximate one real root of the non–linear equation

$$f(x) = x^2 + 4x + 3 + \sin x - x\cos x \tag{4.7}$$

to four decimal places using Newton's method.

Solution:

As a first step, we sketch the curve to find out where the curve crosses the $x-axis$. We generate the plot with the script below.

```
x=linspace(–pi, pi, 100); y=x .^ 2 + 4 .* x + 3 + sin(x) – x  .* cos(x); plot(x,y); grid
```

The plot is shown in Figure 4.3.

The plot shows that one real root is approximately at $x = -1$, so we will use this value as our first approximation.

Next, we generate the function **funcnewt01** and we save it as an *m–file*. To save it, from the *File* menu of the command window, we choose *New* and click on *M–File*. This takes us to the *Editor Window* where we type the following three lines and we save it as ***funcnewt01.m***.

```
function y=funcnewt01(x)
% Approximating roots with Newton's method
y=x .^ 2 + 4 .* x + 3 + sin(x) – x .* cos(x);
```

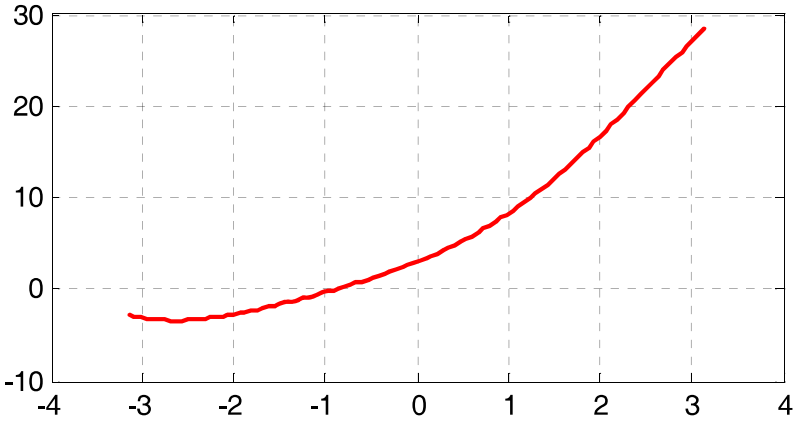


Figure 2.3. Plot for the equation of Example 2.2

We also need the first derivative of y; This is $y' = 2x + 4 + x\sin x$

The computation of the derivative for this example was a simple task; however, we can let MATLAB do the differentiation, just as a check, and to introduce the **diff(s)** function. This function performs differentiation of symbolic expressions. The **syms** function is used to define one or more symbolic expressions.

```
syms x
y = x^2+4*x+3+sin(x)–x*cos(x);      % Dot operations are not necessary with
                                    % symbolic expressions, but correct
                                    % answer will be displayed if they are used.
y1=diff(y)                          % Find the derivative of y
```

```
y1 =
2*x+4+x*sin(x)
```

Now, we generate the function **funcnewt02,** and we save it as m–file. To save it, from the *File* menu of the command window, we choose *New* and click on M–*File*. This takes us to the *Editor Window* where we type these two lines and we save it as ***funcnewt02.m.***

```
function y=funcnewt02(x)
%  Finding roots by Newton's method
%  The following is the first derivative of the function defined as funcnewt02
y=2 .* x + 4 + x .* sin(x);
```

Our script for finding the next approximation with Newton's method follows.

```
x = input('Enter starting value: ');
fx = funcnewt01(x);
fprimex = funcnewt02(x);
xnext = x–fx/fprimex;
   x = xnext;
```

```
   fx = funcnewt01(x);
   fprimex = funcnewt02(x);
disp(sprintf('First approximation is x =  %9.6f \n', x))
while input('Next approximation? (<enter>=no,1=yes)');
   xnext=x–fx/fprimex;
   x=xnext;
   fx=funcnewt01(x);
   fprimex=funcnewt02(x);
disp(sprintf('Next approximation is x =  %9.6f \n', x))
end;
disp(sprintf('%9.6f \n', x))
```

MATLAB produces the following result with –1 as a starting value.

```
Enter starting value: –1
First approximation is: -0.894010
Next approximation? (<enter>=no,1=yes)1
-0.895225
Next approximation? (<enter>=no,1=yes) <enter>
```

We can also use the **fzero(f,x)** function. It was introduced in Chapter 1. This function tries to find a zero of a function of one variable. The string **f** contains the name of a real–valued function of a single real variable. As we recall, MATLAB searches for a value near a point where the function **f** changes sign and returns that value, or returns `NaN` if the search fails.

4.2 The Bisection Method for Root Approximation

The *Bisection* (or *interval halving*) method is an algorithm* for locating the real roots of a function.

* *This is a step–by–step problem–solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.*

The objective is to find two values of x, say x_1 and x_2, so that $f(x_1)$ and $f(x_2)$ have opposite signs, that is, either $f(x_1) > 0$ and $f(x_2) < 0$, or $f(x_1) < 0$ and $f(x_2) > 0$. If any of these two conditions is satisfied, we can compute the midpoint x_m of the interval $x_1 \le x \le x_2$ with

$$x_m = \frac{x_1 + x_2}{2} \tag{4.18}$$

Knowing x_m, we can find $f(x_m)$. Then, the following decisions are made:

1. If $f(x_m)$ and $f(x_1)$ have the same sign, their product will be positive, that is, $f(x_m) \cdot f(x_1) > 0$. This indicates that x_m and x_1 are on the left side of the *x–axis* crossing as shown in Figure 2.11. In this case, we replace x_1 with x_m.

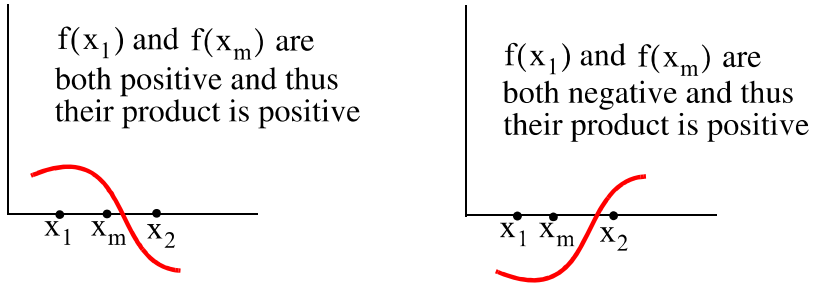


Figure 2.11. Sketches to illustrate the bisection method when $f(x_1)$ *and* $f(x_m)$ *have same sign*

2. If $f(x_m)$ and $f(x_1)$ have opposite signs, their product will be negative, that is, $f(x_m) \cdot f(x_1) < 0$. This indicates that x_m and x_2 are on the right side of the *x–axis* crossing as in Figure 2.12. In this case, we replace x_2 with x_m.

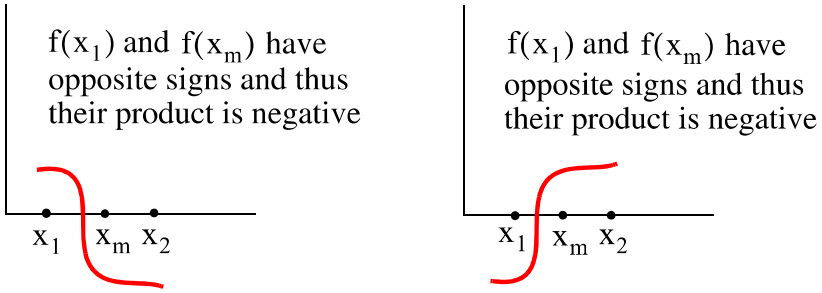


Figure 2.12. Sketches to illustrate the bisection method when $f(x_1)$ *and* $f(x_m)$ *have opposite signs*

After making the appropriate substitution, the above process is repeated until the root we are seeking has a specified tolerance. To terminate the iterations, we either:

a. specify a number of iterations

b. specify a tolerance on the error of $f(x)$

We will illustrate the *Bisection Method* with examples using both MATLAB and Excel.

Example 4.7

Use the Bisection Method with MATLAB to approximate one of the roots of by

$$y = f(x) = 3x^5 - 2x^3 + 6x - 8 \tag{4.19}$$

a. by specifying *16* iterations, and using a **for end** loop MATLAB program

b. by specifying *0.00001* tolerance for $f(x)$, and using a **while end** loop MATLAB program

Solution:

This is the same polynomial as in Example 2.4.

a. The **for end** loop allows a group of functions to be repeated a fixed and predetermined number of times. The syntax is:

```
for x = array
commands...
end
```

Before we write the program script, we must define a function assigned to the given polynomial and save it as a *function m–file*. We will define this function as ***funcbisect01*** and will save it as ***funcbisect01.m***.

```
function y= funcbisect01(x);
y = 3 .* x .^ 5 – 2 .* x .^ 3 + 6 .* x – 8;
%  We must not forget to type the semicolon at the end of the line above;
%  otherwise our script will fill the screen with values of y
```

On the script below, the statement `for k = 1:16` says for $k = 1, k = 2, \ldots, k = 16$, evaluate all commands down to the **end** command. After the $k = 16$ iteration, the loop ends and any commands after the end are computed and displayed as commanded.

Let us also review the meaning of the `fprintf('%9.6f %13.6f \n', xm,fm)` line. Here, `%9.6f` and `%13.6f` are referred to as *format specifiers* or *format scripts*; the first specifies that the value of `xm` must be expressed in *decimal format* also called *fixed point format*, with a total of 9 digits, 6 of which will be to the right of the decimal point. Likewise, `fm` must be expressed in *decimal format* with a total of 13 digits, 6 of which will be to the right of the decimal point. Some other specifiers are `%e` for scientific format, `%s` for string format, and `%d` for integer format. For more information, we can type `help fprintf`. The special format `\n` specifies a linefeed, that is, it prints everything specified up to that point and starts a new line. We will discuss other special formats as they appear in subsequent examples.

The script for the first part of Example 2.7 is given below.

```
x1=1;  x2=2;                    % We know this interval from Example 2.4, Figure 2.6
disp('   xm          fm')       % xm is the average of x1 and x2, fm is f(xm)
disp('-------------------------') % insert line under xm and fm
for k=1:16;
     f1=funcbisect01(x1); f2=funcbisect01(x2);
xm=(x1+x2) / 2; fm=funcbisect01(xm);
fprintf('%9.6f %13.6f \n', xm,fm) % Prints xm and fm on same line;
if (f1*fm<0)
  x2=xm;
else
    x1=xm;
    end
end
```

When this program is executed, MATLAB displays the following:

```
   xm                fm
-------------------------
 1.500000     17.031250
 1.250000      4.749023
 1.125000      1.308441
 1.062500      0.038318
 1.031250     -0.506944
 1.046875     -0.241184
 1.054688     -0.103195
 1.058594     -0.032885
 1.060547      0.002604
 1.059570     -0.015168
 1.060059     -0.006289
 1.060303     -0.001844
 1.060425      0.000380
 1.060364     -0.000732
 1.060394     -0.000176
 1.060410      0.000102
```

We observe that the values are displayed with 6 decimal places as we specified, but for the integer part unnecessary leading zeros are not displayed.

b. The **while end** loop evaluates a group of commands an indefinite number of times. The syntax is:

while expression
    commands...

end

The commands between **while** and **end** are executed as long as all elements in expression are ***true***. The script should be written so that eventually a ***false*** condition is reached and the loop then terminates.

There is no need to create another function *m–file*; we will use the same as in part a. Now we type and execute the following **while end** loop program.

```
x1=1; x2=2; tol=0.00001;
disp('    xm          fm'); disp('------------------------')
while (abs(x1-x2)>2*tol);
  f1=funcbisect01(x1); f2=funcbisect01(x2); xm=(x1+x2)/2;
  fm=funcbisect01(xm);
  fprintf('%9.6f %13.6f \n', xm,fm);
  if (f1*fm<0);
    x2=xm;
  else
   x1=xm;
  end
end
```

When this program is executed, MATLAB displays the following:

```
   xm              fm
-------------------------
 1.500000     17.031250
 1.250000      4.749023
 1.125000      1.308441
 1.062500      0.038318
 1.031250     -0.506944
 1.046875     -0.241184
 1.054688     -0.103195
 1.058594     -0.032885
 1.060547      0.002604
 1.059570     -0.015168
 1.060059     -0.006289
 1.060303     -0.001844
 1.060425      0.000380
 1.060364     -0.000732
 1.060394     -0.000176
 1.060410      0.000102
 1.060402     -0.000037
 1.060406      0.000032
 1.060404     -0.000003
```

Next, we will use an Excel spreadsheet to construct a template that approximates a real root of a function with the bisection method. This requires repeated use of the IF function which has the following syntax.

=IF(logical_test,value_if_true,value_if_false)

where

logical_test: any value or expression that can be evaluated to true or false.

value_if_true: the value that is returned if logical_test is true.

If logical_test is true and value_if_true is omitted, true is returned. Value_if_true can be another formula.

value_if_false is the value that is returned if logical_test is false. If *logical_test* is false and value_if_false is omitted, false is returned. Value_if_false can be another formula.

These statements may be clarified with the following examples.

=IF(C11>=1500,A15, B15):If the value in C11 is greater than or equal to *1500*, use the value in A15; otherwise use the value in B15.

=IF(D22<E22, 800, 1200):If the value in D22 is less than the value of E22, assign the number *800*; otherwise assign the number *1200*.

=IF(M8<>N17, K7*12, L8/24):If the value in *M8* is not equal to the value in *N17*, use the value in *K7* multiplied by *12*; otherwise use the value in *L8* divided by 24.

Example 4.8

Use the bisection method with an Excel spreadsheet to approximate the value of $\sqrt{5}$ within *0.00001* accuracy.

Solution:

Finding the square root of 5 is equivalent to finding the roots of $x^2 - 5 = 0$. We expect the positive root to be in the $2 < x < 3$ interval so we assign $x_1 = 2$ and $x_2 = 3$. The average of these values is $x_m = 2.5$. We will create a template as we did in Example 2.6 so we can use it with any polynomial equation. We start with a blank spreadsheet and we make the entries in rows 1 through 12 as shown in Figure 2.13.

Now, we make the following entries in rows 13 and 14.

A13: 2
B13: 3
C13: =(A13+B13)/2

	A	B	C	D	E	F	G	H
1	Spreadsheet for finding approximations of the real roots							
2	of polynomials using the Bisection method							
3								
4	Equation:	$y = f(x) = x^2 - 5 = 0$						
5								
6	Powers of x and corresponding coefficients of given polynomial f(x)							
7	Enter coefficients of f(x) in Row 9							
8	x^7	x^6	x^5	x^4	x^3	x^2	x	Constant
9	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0	-5
10								
11	x_1	x_2	x_m	$f(x_1)$	$f(x_m)$	$f(x_1)f(x_m)$		
12			$(x_1+x_2)/2$					

Figure 2.13. Partial spreadsheet for Example 2.8

```
D13: =$A$9*A13^7+$B$9*A13^6+$C$9*A13^5+$D$9*A13^4
      +$E$9*A13^3+$F$9*A13^2+$G$9*A13^1+$H$9*A13^0
E13: =$A$9*C13^7+$B$9*C13^6+$C$9*C13^5+$D$9*C13^4
       +$E$9*C13^3+$F$9*C13^2+$G$9*C13^1+$H$9*C13^0
F13: =D13*E13
A14: =IF(A14=A13, C13, B13)
B14: =IF(A14=A13, C13, B13)
```

We copy C13 into C14 and we verify that C14: =(A14+B14)/2

Next, we highlight D13:F13 and on the *Edit* menu we *click* on *Copy*. We place the cursor on D14 and from the *Edit* menu we *click* on *Paste*. We verify that the numbers on D14:F14 are as shown on the spreadsheet of Figure 2.14. Finally, we highlight A14:F14, from the *Edit* menu we *click* on *Copy*, we place the cursor on A15, and holding the mouse left button, we highlight the range A15:A30. Then, from the *Edit* menu, we click on *Paste* and we observe the values in A15:F30.

The square root of 5 accurate to six decimal places is shown on C30 in the spreadsheet of Figure 2.14.

	A	B	C	D	E	F	G	H
1	Spreadsheet for finding approximations of the real roots							
2	of polynomials using the Bisection method							
3								
4	Equation:	$y = f(x) = x^2 - 5 = 0$						
5								
6	Powers of x and corresponding coefficients of given polynomial f(x)							
7	Enter coefficients of f(x) in Row 9							
8	x^7	x^6	x^5	x^4	x^3	x^2	x	Constant
9	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0	-5
10								
11	x_1	x_2	x_m	$f(x_1)$	$f(x_m)$	$f(x_1)f(x_m)$		
12			$(x_1+x_2)/2$					
13	2.00000	3.00000	2.50000	-1.00000	1.25000	-1.25000		
14	2.00000	2.50000	2.25000	-1.00000	0.06250	-0.06250		
15	2.00000	2.25000	2.12500	-1.00000	-0.48438	0.48438		
16	2.12500	2.25000	2.18750	-0.48438	-0.21484	0.10406		
17	2.18750	2.25000	2.21875	-0.21484	-0.07715	0.01657		
18	2.21875	2.25000	2.23438	-0.07715	-0.00757	0.00058		
19	2.23438	2.25000	2.24219	-0.00757	0.02740	-0.00021		
20	2.23438	2.24219	2.23828	-0.00757	0.00990	-0.00007		
21	2.23438	2.23828	2.23633	-0.00757	0.00116	-0.00001		
22	2.23438	2.23633	2.23535	-0.00757	-0.00320	0.00002		
23	2.23535	2.23633	2.23584	-0.00320	-0.00102	0.00000		
24	2.23584	2.23633	2.23608	-0.00102	0.00007	0.00000		
25	2.23584	2.23608	2.23596	-0.00102	-0.00047	0.00000		
26	2.23596	2.23608	2.23602	-0.00047	-0.00020	0.00000		
27	2.23602	2.23608	2.23605	-0.00020	-0.00006	0.00000		
28	2.23605	2.23608	2.23607	-0.00006	0.00000	0.00000		
29	2.23605	2.23607	2.23606	-0.00006	-0.00003	0.00000		
30	2.23606	2.23607	2.23606	-0.00003	-0.00001	0.00000		

Figure 4.14. Entire spreadsheet for Example 2.8

4.5 Exercises

1. Use MATLAB to sketch the graph $y = f(x)$ for each of the following functions, and verify from the graph that $f(a)$ and $f(b)$, where a and b defined below, have opposite signs. Then, use Newton's method to estimate the root of $f(x) = 0$ that lies between a and b.

   a. $f_1(x) = x^4 + x - 3 \qquad a = 1 \qquad b = 2$

   b. $f_2(x) = \sqrt{2x+1} - \sqrt{x+4} \qquad a = 2 \qquad b = 4$

   Hint: Start with $x_0 = (a+b)/2$

2. Repeat Exercise 1 above using the Bisection method.

3. Repeat Example 2.5 using MATLAB.

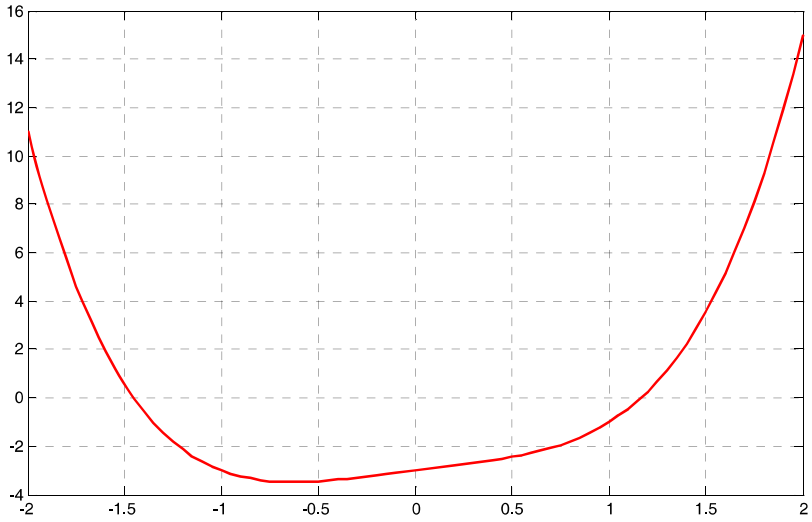
   Hint: Use the procedure of Example 2.2

4.6 Solutions to End-of-Chapter Exercises

1.

a.

```
x=–2:0.05:2; f1x=x.^4+x–3; plot(x,f1x); grid
```



From the plot above we see that the positive root lies between $x = 1$ and $x = 1.25$ so we choose $a = 1$ and $b = 1.25$ so we take $x_0 = 1.1$ as our first approximation. We compute the next value x_1 as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1.1 - \frac{(1.1)^4 + 1.1 - 3}{4(1.1)^3 + 1} = 1.1 - \frac{(-0.436)}{6.324} = 1.169$$

The second approximation yields

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.169 - \frac{(1.169)^4 + 1.169 - 3}{4(1.169)^3 + 1} = 1.169 - \frac{0.0365}{7.39} = 1.164$$

Check with MATLAB:

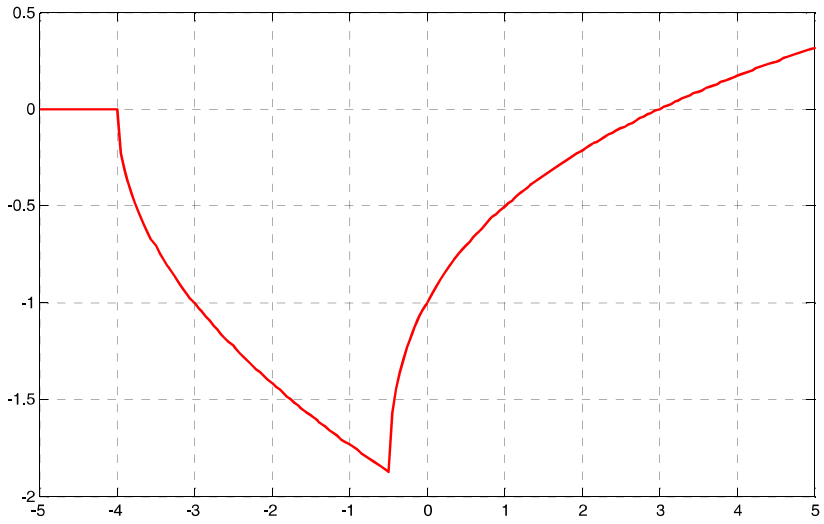
```
pa=[1 0 0 1 –3]; roots(pa)
```

```
ans =

  -1.4526
   0.1443 + 1.3241i
   0.1443 - 1.3241i
   1.1640
```

b.

```
x=–5:0.05:5; f2x=sqrt(2.*x+1)–sqrt(x+4); plot(x,f2x); grid
Warning: Imaginary parts of complex X and/or Y arguments ignored.
```



From the plot above we see that the positive root is very close to $x = 3$ and so we take $x_0 = 3$ as our first approximation. To compute the next value x_1 we first need to find the first derivative of $f_2(x)$. We rewrite it as

$$f_2(x) = \sqrt{2x+1} - \sqrt{x+4} = (2x+1)^{1/2} - (x+4)^{1/2}$$

Then,

$$\frac{d}{dx} \cdot f_2(x) = \frac{1}{2} \cdot (2x+1)^{-1/2} \cdot 2 - \frac{1}{2} \cdot (x+4)^{-1/2} \cdot 1 = \frac{1}{\sqrt{2x+1}} - \frac{1}{2\sqrt{x+4}}$$

and

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 3 - \frac{\sqrt{2\times 3+1} - \sqrt{3+4}}{1/\sqrt{7} - 1/(2\sqrt{7})} = 3 - \frac{0}{1/(2\sqrt{7})} = 3$$

Thus, the real root is exactly $x = 3$. We also observe that since $f(x_0) = \sqrt{7} - \sqrt{7} = 0$, there was no need to find the first derivative $f'(x_0)$.

Check with MATLAB:

```
syms x; f2x=sqrt(2.*x+1)–sqrt(x+4); solve(f2x)
ans =
   3
```

2.

a. We will use the **for end** loop MATLAB program and specify 12 iterations. Before we write the program script, we must define a function assigned to the given polynomial and save it as a function m–file. We will define this function as exercise2 and will save it as exercise2.m

```
function y= exercise2(x);
y = x .^ 4 +x – 3;
```

After saving this file as exercise2.m, we execute the following program:

```
x1=1;  x2=2;                    % x1=a and x2=b
disp('   xm          fm')       % xm is the average of x1 and x2, fm is f(xm)
disp('------------------------') % insert line under xm and fm
for k=1:12;
     f1=exercise2(x1); f2=exercise2(x2);
xm=(x1+x2) / 2; fm=exercise2(xm);
fprintf('%9.6f %13.6f \n', xm,fm)% Prints xm and fm on same line;
if (f1*fm<0)
   x2=xm;
  else
     x1=xm;
   end
end
```

MATLAB displays the following:

```
    xm                fm
-------------------------
1.500000       3.562500
1.250000       0.691406
1.125000      -0.273193
1.187500       0.176041
1.156250      -0.056411
1.171875       0.057803
1.164063       0.000200
1.160156      -0.028229
1.162109      -0.014045
1.163086      -0.006930
1.163574      -0.003367
1.163818      -0.001584
```

b. We will use the **while end** loop MATLAB program and specify a tolerance of 0.00001.

We need to redefine the function *m–file* because the function in part (b) is not the same as in part a.

```
function y= exercise2(x);
y = sqrt(2.*x+1)–sqrt(x+4);
```

After saving this file as exercise2.m, we execute the following program:

```
x1=2.1; x2=4.3; tol=0.00001;      %  If we specify x1=a=2 and x2=b=4, the program
% will not display any values because xm=(x1+x2)/2 = 3 = answer
disp('    xm           fm'); disp('------------------------')
while (abs(x1-x2)>2*tol);
  f1=exercise2(x1); f2=exercise2(x2); xm=(x1+x2)/2;
  fm=exercise2(xm);
  fprintf('%9.6f %13.6f \n', xm,fm);
  if (f1*fm<0);
x2=xm;
  else
    x1=xm;
  end
end
```

When this program is executed, MATLAB displays the following:

```
    xm              fm
------------------------
3.200000       0.037013
2.650000      -0.068779
2.925000      -0.014289
3.062500       0.011733
2.993750      -0.001182
3.028125       0.005299
3.010938       0.002065
3.002344       0.000443
2.998047      -0.000369
3.000195       0.000037
2.999121      -0.000166
2.999658      -0.000065
2.999927      -0.000014
3.000061       0.000012
2.999994      -0.000001
3.000027       0.000005
3.000011       0.000002
```

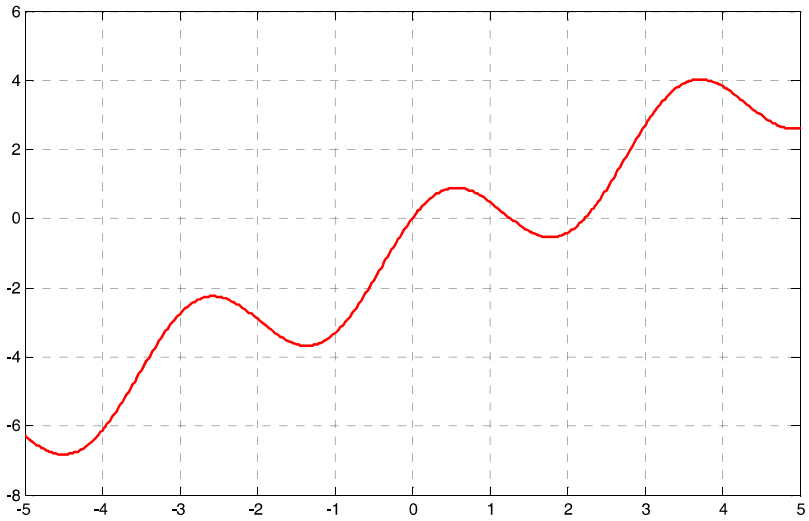
3.

From Example 2.5,

$$y = f(x) = \cos 2x + \sin 2x + x - 1$$

We use the following script to plot this function.

```
x=–5:0.02:5; y=cos(2.*x)+sin(2.*x)+x–1; plot(x,y); grid
```



Let us find out what a symbolic solution gives.

```
syms x; y=cos(2*x)+sin(2*x)+x–1; solve(y)
```

```
ans =
   [0]
   [2]
```

The first value (0) is correct as it can be seen from the plot above and also verified by substitution of $x = 0$ into the given function. The second value (2) is not exactly correct as we can see from the plot. This is because when solving equations of periodic functions, there are an infinite number of solutions and MATLAB restricts its search for solutions to a limited range near zero and returns a non–unique subset of solutions.

To find a good approximation of the second root that lies between $x = 2$ and $x = 3$, we write and save the function files ***exercise3*** and ***exercise3der*** as defined below.

```
function y=exercise3(x)
% Finding roots by Newton's method using MATLAB
y=cos(2.*x)+sin(2.*x)+x–1;

function y=exercise3der(x)
```

```
% Finding roots by Newton's method
% The following is the first derivative of
% the function defined as exercise3
y=–2.*sin(2.*x)+2.*cos(2.*x)+1;
```

Now, we write and execute the following program and we find that the second root is $x = 2.2295$ and this is consistent with the value shown on the plot.

```
x = input('Enter starting value: ');
fx = exercise3(x);
fprimex = exercise3der(x);
xnext = x–fx/fprimex;
   x = xnext;
   fx = exercise3(x);
   fprimex = exercise3der(x);
disp(sprintf('First approximation is x =  %9.6f \n', x))
while input('Next approximation? (<enter>=no,1=yes)');
   xnext=x–fx/fprimex;
   x=xnext;
   fx=exercise3(x);
   fprimex=exercise3der(x);
disp(sprintf('Next approximation is x =  %9.6f \n', x))
end;
disp(sprintf('%9.6f \n', x))
```

```
Enter starting value: 3

First approximation is x = 2.229485
```

Introduction

Numerical methods are commonly used for solving mathematical problems that are formulated in science and engineering where it is difficult or impossible to obtain exact solutions. MATLAB has a large library of functions for numerically solving a wide variety of mathematical problems. This chapter explains a number of the most frequently used of these functions. It should be pointed out here that the purpose of this book is to show users how to use MATLAB.

4.1 Solving an equation with one variable

In general, a function can have zero, one, several, or an infinite number of solutions.

In MATLAB a zero of a function can be determined with the command (built-in function) `fzero` with the form:

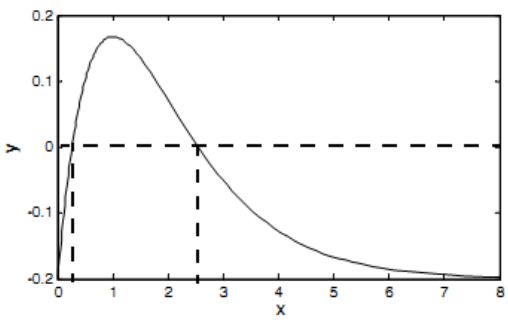
```
x = fzero(function,x0)
```

Solution — The function to be solved. — A value of x close to where the function crosses the axis.

EX: Determine the solution of the equation $xe^{-x} = 0.2$.

Solution

The equation is first written in the form of a function: $f(x) = xe^{-x} - 0.2$. A plot of the function, shown on the right, shows that the function has one solution between 0 and 1 and another solution between 2 and 3. The plot is obtained by typing




```
>> fplot('x*exp(-x)-0.2',[0 8])
```

```
>> x1=fzero('x*exp(-x)-0.2',0.7)
x1 =
    0.2592
>> F=@(x)x*exp(-x)-0.2
F =
    @(x)x*exp(-x)-0.2
>> fzero(F,2.8)
ans =
    2.5426
```

The function is entered as a string expression.

The first solution is 0.2592.

Creating an anonymous function.

Using the name of the anonymous function in `fzero`.

The second solution is 2.5426.

EX: Find the root of $xe^{-x} - 0.3 = 0$.

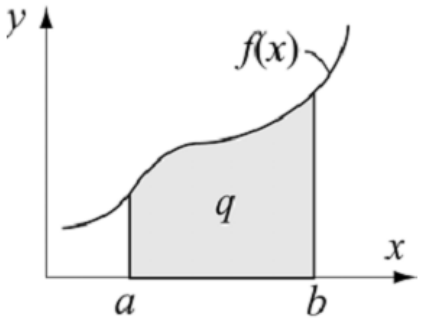
Solution:

```
>> x=fzero('x*exp(-x)-0.3',0.5)
x =
    0.4894
```

4.2 Numerical integration

It is assumed in the presentation below that the reader has knowledge of integrals and integration. A definite integral of a function $f(x)$ from a to b has the form:

$$q = \int_a^b f(x)dx$$



The function $f(x)$ is called the integrand, and the numbers a and b are the limits of integration. Graphically, the value of the integral q is the area between the graph of the function, the x axis, and the limits a and b (the shaded area in the figure).

The following discussion describes how to use the three MATLAB built-in integration functions `quad`, `quadl`, and `trapz`. The `quad` and `quadl` commands are used for integration when $f(x)$ is a function, and `trapz` is used when $f(x)$ is given by data points.

The `quad` command:

The form of the `quad` command, which uses the adaptive Simpson method of integration, is:

```
q = quad(function,a,b)
```

The value of the integral. The function to be integrated. The integration limits.

EX: Use numerical integration to calculate the following integral:

$$\int_0^8 (xe^{-x^{0.8}} + 0.2)dx$$

The use of the `quad` command in the Command Window, with the function to be integrated typed in as a string, is shown below. Note that the function is typed with element-by-element operations.

```
>> quad('x.*exp(-x.^0.8)+0.2',0,8)
ans =
    3.1604
```

The `trapz` command:

The `trapz` command can be used for integrating a function that is given as data points. It uses the numerical trapezoidal method of integration. The form of the command is

```
q = trapz(x,y)
```

where `x` and `y` are vectors with the *x* and *y* coordinates of the points, respectively. The two vectors must be of the same length.

EX: find the area under the curve for the data :

```
x=(0.9 1.5 3 4 6 8 9.5)
y=(0.9 1.5 2.5 5.1 4.5 4.9 6.3)
```

Solution:

```
x=[0.9 1.5 3 4 6 8 9.5];
y=[0.9 1.5 2.5 5.1 4.5 4.9 6.3];
area=trapz(x,y)
```

answer

area =

34.9200

4.1 Differentiation

The derivative of a function y = f(x) is written as $\frac{dy}{dx}f(x)$ or `f'(x)`, and is defined as the rate of change of the dependent variable y with respect to x. The derivative is the slope of the line tangent to the function at a given point.

MATLAB has a function **polyder**, which will find the derivative of a polynomial. For example, for the polynomial $x^3 + 2x^2 - 4x + 3$, which would be represented by the vector `[1 2 -4 3]`, the derivative is found by:

```
>> origp = [1 2 -4 3];
>> diffp = polyder(origp)
diffp =

     3     4    -4
```

which shows that the derivative is the polynomial $3x^2 + 4x - 4$. The function **polyval** can then be used to find the derivative for certain values of x, such as for x = 1, 2, and 3:

```
>> polyval(diffp, 1:3)
ans =
     3    16    35
```

The derivative can be written as the limit

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

and can be approximated by a difference equation.

Recall that MATLAB has a built-in function, **diff**, which returns the differences between consecutive elements in a vector. For a function `y = f(x)` where x is a vector, the values of `f'(x)` can be approximated as **diff(y)** divided by **diff(x)**. For example, the equation $x^3 + 2x^2 - 4x + 3$ can be written as an anonymous

function. It can be seen that the approximate derivative is close to the values found using **polyder** and **polyval**.

```
>> f = @ (x) x .^ 3 + 2 * x .^ 2 - 4 * x + 3;
>> x = 0.5 : 3.5
x =
   0.5000      1.5000      2.5000      3.5000
>> y = f(x)
y =
   1.6250      4.8750     21.1250     56.3750
>> diff(y)
ans =
      3.2500     16.2500     35.2500
>> diff(x)
ans =
     1           1           1
>> diff(y) ./ diff(x)
ans =
     3.2500     16.2500     35.2500
```

Chapter 4

Integration by Numerical Methods

This chapter is an introduction to numerical methods for integrating functions which are very difficult or impossible to integrate using analytical means. We will discuss the trapezoidal rule that computes a function $f(x)$ with a set of linear functions, and Simpson's rule that computes a function $f(x)$ with a set of quadratic functions.

4.8 The Trapezoidal Rule

Consider the function $y = f(x)$ for the interval $a \le x \le b$, shown in Figure 4.11.

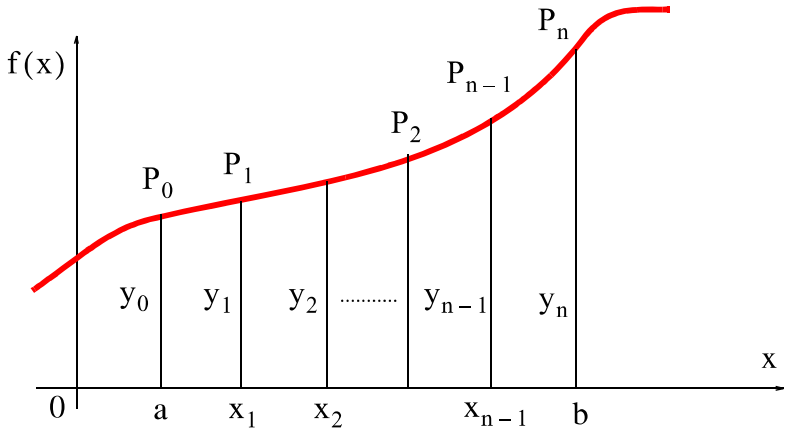


Figure 4.11. Integration by the trapezoidal rule

To evaluate the definite integral $\int_a^b f(x)dx$, we divide the interval $a \le x \le b$ into n subintervals each of length $\Delta x = \frac{b-a}{n}$. Then, the number of points between $x_0 = a$ and $x_n = b$ is $x_1 = a + \Delta x$, $x_2 = a + 2\Delta x$, …, $x_{n-1} = a + (n-1)\Delta x$. Therefore, the integral from *a* to *b* is the sum of the integrals from a to x_1, from x_1 to x_2, and so on, and finally from x_{n-1} to b. The total area is

$$\int_a^b f(x)dx = \int_a^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \ldots + \int_{x_{n-1}}^{b} f(x)dx = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x)dx$$

The integral over the first subinterval, can now be approximated by the area of the trapezoid

$aP_0P_1x_1$ that is equal to $\frac{1}{2}(y_0 + y_1)\Delta x$ plus the area of the trapezoid $x_1P_1P_2x_2$ that is equal to $\frac{1}{2}(y_1 + y_2)\Delta x$, and so on. Then, the trapezoidal approximation becomes

$$T = \frac{1}{2}(y_0 + y_1)\Delta x + \frac{1}{2}(y_1 + y_2)\Delta x + \ldots + \frac{1}{2}(y_{n-1} + y_n)\Delta x$$

or

$$\boxed{T = \left(\frac{1}{2}y_0 + y_1 + y_2 + \ldots + y_{n-1} + \frac{1}{2}y_n\right)\Delta x}$$

Trapezoidal Rule (4.11)

Example 4.11

Using the trapezoidal rule with $n = 4$, estimate the value of the definite integral

$$\int_1^2 x^2 dx \tag{4.12}$$

Compare with the exact value, and compute the percent error.

Solution:

The exact value of this integral is

$$\int_1^2 x^2 dx = \frac{x^3}{3}\Big|_1^2 = \frac{8}{3} - \frac{1}{3} = \frac{7}{3} = 2.33333 \tag{4.13}$$

For the trapezoidal rule approximation we have

$$x_0 = a = 1$$
$$x_n = b = 2$$
$$n = 4$$
$$\Delta x = \frac{b-a}{n} = \frac{2-1}{4} = \frac{1}{4}$$
$$y = f(x) = x^2$$

Then,

$$x_0 = a = 1 \qquad y_0 = f(x_0) = 1^2 = \frac{16}{16}$$

$$x_1 = a + \Delta x = \frac{5}{4} \qquad y_1 = f(x_1) = \left(\frac{5}{4}\right)^2 = \frac{25}{16}$$

$$x_2 = a + 2\Delta x = \frac{6}{4} \qquad y_2 = f(x_2) = \left(\frac{6}{4}\right)^2 = \frac{36}{16}$$

$$x_3 = a + 3\Delta x = \frac{7}{4} \qquad y_3 = f(x_3) = \left(\frac{7}{4}\right)^2 = \frac{49}{16}$$

$$x_4 = b = 2 \qquad y_4 = f(x_4) = \left(\frac{8}{4}\right)^2 = \frac{64}{16}$$

and by substitution into (4.11),

$$T = \left(\frac{1}{2}\times\frac{16}{16} + \frac{25}{16} + \frac{36}{16} + \frac{49}{16} + \frac{1}{2}\times\frac{64}{16}\right)\times\frac{1}{4} = \frac{150}{16}\times\frac{1}{4} = \frac{75}{32} = 2.34375 \tag{10.4}$$

From (4.13) and (4.14), we find that the percent error is

$$\%\ Error = \frac{2.34375 - 2.33333}{2.33333}\times 100 = 0.45\ \% \tag{4.15}$$

The MATLAB function **trapz(x,y,n)** where **y** is the integral with respect to **x**, approximates the integral of a function $y = f(x)$ using the trapezoidal rule, and **n** (optional) performs integration along dimension n.

Example 10.2

Use the MATLAB function **trapz(x,y)** to approximate the value of the integral

$$\int_1^2 \frac{1}{x}dx \tag{4.16}$$

and by comparison with the exact value, compute the percent error when $n = 5$ and $n = 10$

Solution:

The exact value is found from

$$\int_1^2 \frac{1}{x}dx = \ln x\Big|_1^2 = \ln 2 - \ln 1 = 0.6931 - 0.0000 = 0.6931$$

For the approximation using the trapezoidal rule, we let x_5 represent the row vector with $n = 5$,

and x_{10} the vector with $n = 10$, that is, $\Delta x = 1/5$ and $\Delta x = 1/10$ respectively. The corresponding values of y are denoted as y_5 and y_{10}, and the areas under the curve as area5 and area10 respectively. We use the following MATLAB script.

```
x5=linspace(1,2,5); x10=linspace(1,2,10);
y5=1./x5; y10=1./x10;
area5=trapz(x5,y5), area10=trapz(x10,y10)
```

```
area5 =
    0.6970

area10 =
    0.6939
```

The percent error when $\Delta x = 1/5$ is used is

$$\% \text{ Error } = \frac{0.6970 - 0.6931}{0.6931} \times 100 = 0.56\ \%$$

and the percent error when $\Delta x = 1/10$ is used is

$$\% \text{ Error } = \frac{0.6939 - 0.6931}{0.6931} \times 100 = 0.12\ \%$$

Example 4.13

The integral

$$f(t) = \int_0^t e^{-\tau^2} d\tau \tag{4.17}$$

where τ is a dummy variable of integration, is called *the error function** and it is used extensively in communications theory. Use the MATLAB **trapz(x,y)** function to find the area under this integral with $n = 10$ when the upper limit of integration is $t = 2$.

Solution:

We use the same procedure as in the previous example. The MATLAB script for this example is

```
t=linspace(0,2,10); y=exp(–t.^2); area=trapz(t,y)
```

MATLAB displays the following result.

* *The formal definition of the error function is* $\text{erf}(u) = \frac{2}{\sqrt{\pi}}\int_0^u e^{-\tau^2} d\tau$

```
area =
    0.8818
```

Example 4.14

The $i-v$ (current–voltage) relation of a non–linear electrical device is given by

$$i(t) = 0.1(e^{0.2v(t)} - 1) \tag{4.18}$$

where $v(t) = \sin 3t$.

By any means, find

a. The instantaneous power $p(t)$

b. The energy $W(t_0, t_1)$ dissipated in this device from $t_0 = 0$ to $t_1 = 10$ s.

Solution:

a. The instantaneous power is

$$p(t) = v(t)i(t) = 0.1\sin 3t(e^{0.2\sin 3t} - 1) \tag{10.9}$$

b. The energy is the integral of the instantaneous power, that is,

$$W(t_0, t_1) = \int_{t_0}^{t_1} p(t)dt = 0.1\int_{0}^{10\ s} \sin 3t(e^{0.2\sin 3t} - 1)dt \tag{10.10}$$

An analytical solution of the last integral is possible using integration by parts, but it is not easy. We can try the MATLAB **int(f,a,b)** function where **f** is a symbolic expression, and **a** and **b** are the lower and upper limits of integration respectively.

When MATLAB cannot find a solution, it returns a warning. For this example, MATLAB returns the following message when integration is attempted with the symbolic expression of (10.10).

```
t=sym('t');
s=int(0.1*sin(3*t)*(exp(0.2*sin(3*t))–1),0,10)
```

When this script is executed, MATLAB displays the following message.

```
Warning: Explicit integral could not be found.
```

Next, we will find and sketch the power and energy by the trapezoidal rule using the MATLAB **trapz(x,y)** function. For this example, we choose $n = 100$, so that $\Delta x = 1/100$. The MATLAB script below will compute and plot the power.

```
t=linspace(0,10,100);
v=sin(3.*t); i=0.1.*(exp(0.2.*v)–1); p=v.*i;
```

```
plot(t,p); grid; title('Power vs Time'); xlabel('seconds'); ylabel('watts')
```

The power varies in a uniform fashion as shown by the plot of Figure 10.2.

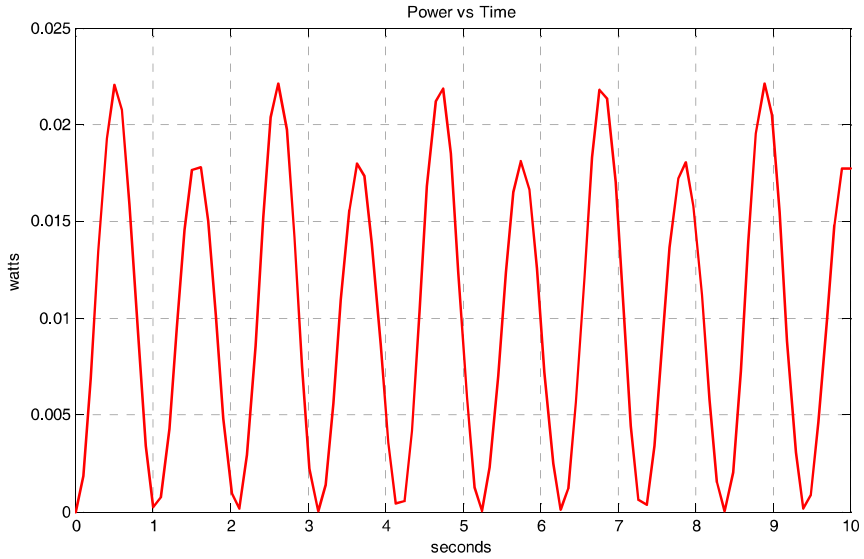


Figure 10.2. Plot for the power variation in Example 10.4

The plot of Figure 10.2 shows that the power is uniform for all time, and thus we expect the energy to be constant.

The MATLAB script below computes and plots the energy.

```
energy=trapz(t,p), plot(t,energy, '+'); grid; title('Energy vs Time');...
xlabel('seconds'); ylabel('joules')
```

```
energy =
    0.1013
```

Thus, the value of the energy is 0.1013 joule. The energy is shown in Figure 10.3.

4- 9 Simpson's Rule

The trapezoidal and Simpson's rules are special cases of the *Newton–Cote rules* which use higher degree functions for numerical integration.

Let the curve of Figure 10.4 be represented by the parabola

$$y = \alpha x^2 + \beta x + \gamma \qquad (10.11)$$

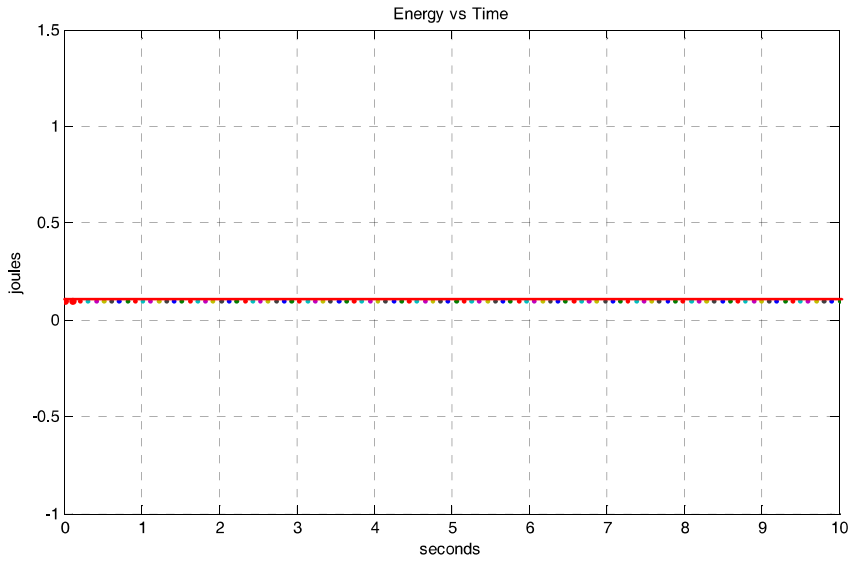


Figure 10.3. Plot for the energy of Example 10.4

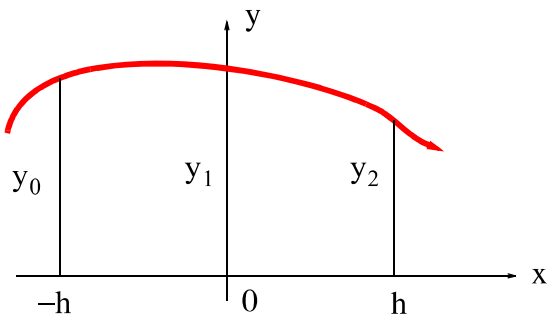


Figure 10.4. Simpson's rule of integration

The area under this curve for the interval $-h \le x \le h$ is

$$\begin{aligned} \text{Area}\Big|_{-h}^{h} &= \int_{-h}^{h} (\alpha x^2 + \beta x + \gamma)dx = \frac{\alpha x^3}{3} + \frac{\beta x^2}{2} + \gamma x \Big|_{-h}^{h} \\ &= \frac{\alpha h^3}{3} + \frac{\beta h^2}{2} + \gamma h - \left(-\frac{\alpha h^3}{3} + \frac{\beta h^2}{2} - \gamma h\right) = \frac{2\alpha h^3}{3} + 2\gamma h \\ &= \frac{1}{3} h(2\alpha h^3 + 6\gamma) \end{aligned} \tag{10.12}$$

The curve passes through the three points $(-h, y_0)$, $(0, y_1)$, and (h, y_2). Then, by (10.11) we have:

$$y_0 = \alpha h^2 - \beta h + \gamma \qquad (a)$$
$$y_1 = \gamma \qquad (b) \qquad (10.13)$$
$$y_2 = \alpha h^2 + \beta h + \gamma \qquad (c)$$

We can now evaluate the coefficients α, β, γ and express (10.12) in terms of h, y_0, y_1 and y_2. This is done with the following procedure.

By substitution of (b) of (10.13) into (a) and (c) and rearranging we obtain

$$\alpha h^2 - \beta h = y_0 - y_1 \qquad (10.14)$$

$$\alpha h^2 + \beta h = y_2 - y_1 \qquad (10.15)$$

Addition of (10.14) with (10.15) yields

$$2\alpha h^2 = y_0 - 2y_1 + y_2 \qquad (10.16)$$

and by substitution into (10.12) we obtain

$$\text{Area}\Big|_{-h}^{h} = \frac{1}{3}h(2\alpha h^3 + 6\gamma) = \frac{1}{3}h[(y_0 - 2y_1 + y_2) + 6y_1] \qquad (10.17)$$

or

$$\text{Area}\Big|_{-h}^{h} = \frac{1}{3}h(y_0 + 4y_1 + y_2) \qquad (10.18)$$

Now, we can apply (10.18) to successive segments of any curve $y = f(x)$ in the interval $a \le x \le b$ as shown on the curve of Figure 10.5.

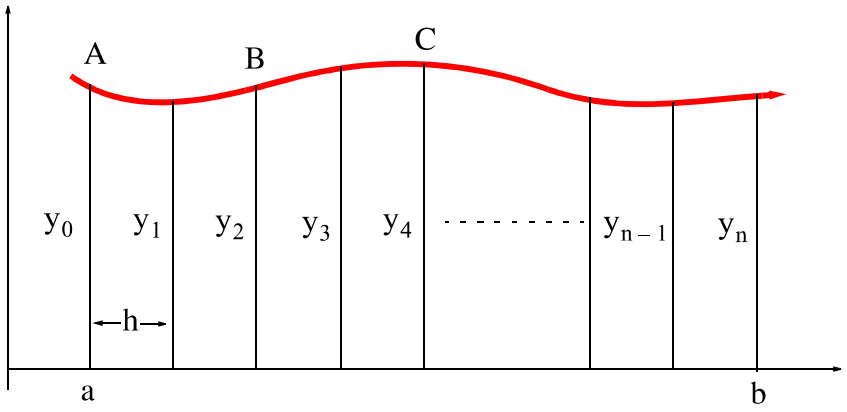


Figure 10.5. Simpson's rule of integration by successive segments

From Figure 10.5, we observe that each segment of width $2h$ of the curve can be approximated by a parabola through its ends and its midpoint. Thus, the area under segment AB is

$$\text{Area}|_{AB} = \frac{1}{3}h(y_0 + 4y_1 + y_2) \tag{10.19}$$

Likewise, the area under segment BC is

$$\text{Area}|_{BC} = \frac{1}{3}h(y_2 + 4y_3 + y_4) \tag{10.20}$$

and so on. When the areas under each segment are added, we obtain

$$\text{Area} = \frac{1}{3}h(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots + 2y_{n-2} + 4y_{n-1} + y_n) \tag{10.21}$$

Simpson's Rule of Numerical Integration

Since each segment has width $2h$, *to apply Simpson's rule of numerical integration, the number* n *of subdivisions must be even*. This restriction does not apply to the trapezoidal rule of numerical integration. The value of h for (10.21) is found from

$$h = \frac{b-a}{n} \quad n = \text{even} \tag{10.22}$$

Example 10.5

Using Simpson's rule with 4 subdivisions $(n = 4)$, compute the approximate value of

$$\int_1^2 \frac{1}{x}dx \tag{10.23}$$

Solution:

This is the same integral as that of Example 10.2 where we found that the analytical value of this definite integral is $\ln = 0.6931$. We can also find the analytical value with MATLAB's **int(f,a,b)** function where **f** is a symbolic expression, and **a** and **b** are the lower and upper limits of integration respectively. For this example,

```
syms x
Area=int(1/x,1,2)
```

```
Area =
log(2)
```

We recall that log(x) in MATLAB is the natural logarithm.

To use Simpson's rule, for convenience, we construct the following table using the spreadsheet of Figure 10.6.

	A	B	C	D	E
1	**Example 10.5**				
2	*∫(1/x)dx evaluated from a = 1 to b = 2 with n = 4*				
3	Numerical integration by Simpson's method follows				
4	Given	a=	1		
5		b=	2		
6		n=	4		
7	Then,	h = (b-a)/n =	0.2500		
8				Multiplier	Products
9		x_0=a=	1.00000		
10		y_0=1/x_0=	1.00000	1	1.00000
11		x_1=a+h=	1.25000		
12		y_1=1/x_1=	0.80000	4	3.20000
13		x_2=a+2h=	1.50000		
14		y_2=1/x_2=	0.66667	2	1.33333
15		x_3=a+3h=	1.75000		
16		y_3=1/x_3=	0.57143	4	2.28571
17		x_4=b=	2.00000		
18		y_4=1/x_4=	0.50000	1	0.50000
19			Sum of Products =		8.31905
20	Area = (h/3)*(Sum of Products) = (1/12)*8.31905 =				**0.69325**

Figure 10.6. Spreadsheet for numerical integration of (10.23)

By comparison of the numerical with the exact value, we observe that the error is very small when Simpson's method is applied.

MATLAB has two quadrature functions for performing numerical integration, the **quad** and **quad8**. The description of these can be seen by typing **help quad** or **help quad8**. Both of these functions use *adaptive quadrature methods*; this means that these methods can handle irregularities such as singularities. When such irregularities occur, MATLAB displays a warning message but still provides an answer.

The **quad** function uses an adaptive form of Simpson's rule, while the **quad8** function uses the so–called *Newton–Cotes 8–panel rule*. The **quad8** function detects and handles irregularities more efficiently.

Both functions have the same syntax, that is, **q=quad('f',a,b,tol)**, and integrate to a relative error **tol** which we must specify. If **tol** is omitted, it is understood to be the *standard tolerance* of 10^{-3}. The string **'f'** is the name of a user defined function, and **a** and **b** are the lower and upper limits of integration respectively.

Example 10.6

Given the definite integral

$$y = f(x) = \int_0^2 e^{-x^2} dx \tag{10.24}$$

a. Use MATLAB's symbolic **int** function to obtain the value of this integral

b. Obtain the value of this integral with the **q=quad('f',a,b)** function

c. Obtain the value of this integral with the **q=quad('f',a,b,tol)** function where tol = 10^{-10}

d. Obtain the value of this integral with the **q=quad8('f',a,b)** function

e. Obtain the value of this integral with the **q=quad8('f',a,b,tol)** function where tol = 10^{-10}

Solution:

a.

```
syms x; y=int(exp(–x^2),0,2)  % Define symbolic variable x and integrate

y =
1/2*erf(2)*pi^(1/2)
pretty(y)
                                1/2
                    1/2 erf(2) pi
```

erf is an acronym for the error function and we can obtain its definition with help erf

b. First, we need to create and save a function m–file. We name it errorfcn1.m as shown below. We will use format long to display the values with 15 digits.

```
function y = errorfcn1(x)
y = exp(–x.^2);
```

With this file saved as errorfcn1.m, we write and execute the following MATLAB script.

```
format long
y_std=quad('errorfcn1',0,2)
```

We obtain the answer in standard tolerance form as

```
y_std =
      0.88211275610253
```

c. With the specified tolerance, the script and the answer are as follows:

```
y_tol=quad('errorfcn1',0,2,10^–10)

y_tol =
     0.88208139076242
```

d. With the standard tolerance,

```
y_std8=quad8('errorfcn1',0,2)

y_std8 =
    0.88208139076194
```

e. With the specified tolerance,

```
y_tol8=quad8('errorfcn1',0,2,10^–10)

y_tol8 =
    0.88208139076242
```

We observe that with the 10^{-10} tolerance, both **quad** and **quad8** produce the same result.

Example 10.7

Using the **quad** and **quad8** functions with standard tolerance, evaluate the integral

$$y = f(x) = \int_a^b \sqrt{x}\,dx \tag{10.25}$$

at $((a, b) = (0.2, 0.8), (1.4, 2.3))$, and $(3,8)$. Use the **fprintf** function to display first the analytical values, then, the numerical values produced by the **quad** and **quad8** functions for each set of data.

Solution:

Evaluating the given integral, we obtain

$$y = \int_a^b x^{1/2}dx = \frac{x^{3/2}}{3/2}\Big|_a^b = \frac{2}{3}(b^{3/2} - a^{3/2}) \tag{10.26}$$

where a and b are non–negative values. Substitution of the values of the given values of a and b will be included in the MATLAB script below.

The **sqrt** function in a built–in function and therefore, we need not write a user defined m–file. We will include the **input** function in the script. The script is then saved as *Example_10_7*.

```
% This script displays the approximations obtained with the quad and quad8 functions
% with the analytical results for the integration of the square root of x over the
% interval (a,b) where a and b are non–negative.
%
fprintf(' \n'); % Insert line
a=input('Enter first point  "a" (non–negative): ');
b=input('Enter second point "b" (non–negative): ');
```

```
k=2/3.*(b.^(1.5)–a.^(1.5));
kq=quad('sqrt',a,b);
kq8=quad8('sqrt',a,b);
fprintf(' \n');... % Insert line
fprintf(' Analytical: %f \n Numerical quad, quad8: %f  %f \n',k,kq,kq8);...
fprintf(' \n'); fprintf(' \n') % Insert two lines
```

Now, we execute this saved file by typing its name, that is,

Example_10_7

```
Enter first point  "a" (non-negative): 0.2
Enter second point "b" (non-negative): 0.8

Analytical: 0.417399

Numerical quad, quad8: 0.417396  0.417399
```

Example_10_7

```
Enter first point  "a" (non-negative): 1.4
Enter second point "b" (non-negative): 2.3

Analytical: 1.221080

Numerical quad, quad8: 1.221080  1.221080
```

Example_10_7

```
Enter first point  "a" (non-negative): 3
Enter second point "b" (non-negative): 8

Analytical: 11.620843

Numerical quad, quad8: 11.620825  11.620843
```

4.4 Exercises

1. Use the trapezoidal approximation to compute the values the following definite integrals and compare your results with the analytical values. Verify your answers with the MATLAB **trapz(x,y,n)** function.

   a. $\int_0^2 x\,dx \quad n = 4$

   b. $\int_0^2 x^3\,dx \quad n = 4$

   c. $\int_0^2 x^4\,dx \quad n = 4$

   d. $\int_1^2 \frac{1}{x^2}\,dx \quad n = 4$

2. Use Simpson's rule to approximate the following definite integrals and compare your results with the analytical values. Verify your answers with the MATLAB **quad('f',a,b)** function.

   a. $\int_0^2 x^2\,dx \quad n = 4$

   b. $\int_0^\pi \sin x\,dx \quad n = 4$

   c. $\int_0^1 \frac{1}{x^2+1}\,dx \quad n = 4$

4.5 Solution to End–of–Chapter Exercises

1.

$$T = \left(\frac{1}{2}y_0 + y_1 + y_2 + \ldots + y_{n-1} + \frac{1}{2}y_n\right)\Delta x$$

a. The exact value is

$$\int_0^2 x\,dx = \left.\frac{x^2}{2}\right|_0^2 = 2$$

For the trapezoidal rule approximation we have

$$x_0 = a = 0$$
$$x_n = b = 2$$
$$n = 4$$
$$\Delta x = \frac{b-a}{n} = \frac{2-0}{4} = \frac{1}{2}$$
$$y = f(x) = x$$

$$x_0 = a = 0 \qquad y_0 = f(x_0) = 0$$
$$x_1 = a + \Delta x = \frac{1}{2} \qquad y_1 = f(x_1) = \frac{1}{2}$$
$$x_2 = a + 2\Delta x = 1 \qquad y_2 = f(x_2) = 1$$
$$x_3 = a + 3\Delta x = \frac{3}{2} \qquad y_3 = f(x_3) = \frac{3}{2}$$
$$x_4 = b = 2 \qquad y_4 = f(x_4) = 2$$

$$T = \left(\frac{1}{2}\times 0 + \frac{1}{2} + 1 + \frac{3}{2} + \frac{1}{2}\times 2\right)\times\frac{1}{2} = 4\times\frac{1}{2} = 2$$

```
x=linspace(0,2,4); y=x; area=trapz(x,y)
```

```
area =
     2
```

b. The exact value is

$$\int_0^2 x^3\,dx = \left.\frac{x^4}{4}\right|_0^2 = 4$$

For the trapezoidal rule approximation we have

$$x_0 = a = 0$$
$$x_n = b = 2$$
$$n = 4$$
$$\Delta x = \frac{b-a}{n} = \frac{2-0}{4} = \frac{1}{2}$$
$$y = f(x) = x^3$$

$$x_0 = a = 0 \qquad y_0 = f(x_0) = 0$$
$$x_1 = a + \Delta x = \frac{1}{2} \qquad y_1 = f(x_1) = \frac{1}{8}$$
$$x_2 = a + 2\Delta x = 1 \qquad y_2 = f(x_2) = 1$$
$$x_3 = a + 3\Delta x = \frac{3}{2} \qquad y_3 = f(x_3) = \frac{27}{8}$$
$$x_4 = b = 2 \qquad y_4 = f(x_4) = 8$$

$$T = \left(\frac{1}{2}\times 0 + \frac{1}{8} + 1 + \frac{27}{8} + \frac{1}{2}\times 8\right)\times\frac{1}{2} = \left(5 + \frac{7}{2}\right)\times\frac{1}{2} = 4.25$$

```
x=linspace(0,2,4); y=x.^3; area=trapz(x,y)

area =
    4.4444
```

The deviations from the exact value are due to the small number of divisions n we chose.

c. The exact value is

$$\int_0^2 x^4 dx = \frac{x^5}{5}\Big|_0^2 = \frac{32}{5} = 6.4$$

For the trapezoidal rule approximation we have

$$x_0 = a = 0$$
$$x_n = b = 2$$
$$n = 4$$
$$\Delta x = \frac{b-a}{n} = \frac{2-0}{4} = \frac{1}{2}$$
$$y = f(x) = x^4$$

$$x_0 = a = 0 \qquad y_0 = f(x_0) = 0$$
$$x_1 = a + \Delta x = \frac{1}{2} \qquad y_1 = f(x_1) = \frac{1}{16}$$
$$x_2 = a + 2\Delta x = 1 \qquad y_2 = f(x_2) = 1$$
$$x_3 = a + 3\Delta x = \frac{3}{2} \qquad y_3 = f(x_3) = \frac{81}{8}$$
$$x_4 = b = 2 \qquad y_4 = f(x_4) = 16$$

$$T = \left(\frac{1}{2}\times 0 + \frac{1}{16} + 1 + \frac{81}{16} + \frac{1}{2}\times 16\right)\times\frac{1}{2} = \left(9 + \frac{41}{8}\right)\times\frac{1}{2} = 7.0625$$

```
x=linspace(0,2,4); y=x.^4; area=trapz(x,y)
area =
    7.5720
```

d. The exact value is

$$\int_1^2 \frac{1}{x^2}dx = -\frac{1}{x}\Big|_1^2 = \frac{1}{2}$$

For the trapezoidal rule approximation we have

$$x_0 = a = 1$$
$$x_n = b = 2$$
$$n = 4$$
$$\Delta x = \frac{b-a}{n} = \frac{2-1}{4} = \frac{1}{4}$$
$$y = f(x) = 1/x^2$$

$$x_0 = a = 1 \qquad y_0 = f(x_0) = 1$$
$$x_1 = a + \Delta x = \frac{5}{4} \qquad y_1 = f(x_1) = \frac{16}{25}$$
$$x_2 = a + 2\Delta x = \frac{3}{2} \qquad y_2 = f(x_2) = \frac{4}{9}$$
$$x_3 = a + 3\Delta x = \frac{7}{4} \qquad y_3 = f(x_3) = \frac{16}{49}$$
$$x_4 = b = 2 \qquad y_4 = f(x_4) = \frac{1}{4}$$

$$T = \left(\frac{1}{2}\times 1 + \frac{16}{25} + \frac{4}{9} + \frac{16}{49} + \frac{1}{2}\times\frac{1}{4}\right)\times\frac{1}{4} = \left(\frac{3905}{1918}\right)\times\frac{1}{4} = 0.5090$$

```
x=linspace(1,2,4); y=1./x.^2; area=trapz(x,y)

area =
    0.5158
```

2.

$$\text{Area} = \frac{1}{3}h(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots + 2y_{n-2} + 4y_{n-1} + y_n)$$

a. The exact value is

$$\int_0^2 x^2 dx = \frac{x^3}{3}\Big|_0^2 = \frac{8}{3} = 2.6667$$

To use Simpson's rule we construct the following table using a spreadsheet.

	A	B	C	D	E
1	**Exercise 10.2.a**				
2	*$\int x^2 dx$ evaluated from a = 0 to b = 2 with n = 4*				
3	Numerical integration by Simpson's method follows				
4	Given	a=	0		
5		b=	2		
6		n=	4		
7	Then,	h = (b-a)/n =	0.5000		
8				Multiplier	Products
9		x_0=a=	0.00000		
10		$y_0=x_0^2$=	0.00000	1	0.0000
11		x_1=a+h=	0.50000		
12		$y_1=x_1^2$=	0.25000	4	1.0000
13		x_2=a+2h=	1.00000		
14		$y_2=x_2^2$=	1.00000	2	2.0000
15		x_3=a+3h=	1.50000		
16		$y_3=x_3^2$=	2.25000	4	9.0000
17		x_4=b=	2.00000		
18		$y_4=x_4^2$=	4.00000	1	4.0000
19			Sum of Products =		16.0000
20	Area = (h/3)*(Sum of Products) = (1/12)*8.31905 =				**2.6667**

We create and save a function m–file. We name it exer_10_2_a.m as shown below.

```
function y = exer_10_2_a(x)
y = x.^2;
```

We write and execute the following MATLAB script:

```
y_std=quad('exer_10_2_a',0,2)

y_std =
    2.6667
```

b. The exact value is

$$\int_0^{\pi} \sin x\,dx = -\cos x\Big|_0^{\pi} = -(-1-1) = 2$$

To use Simpson's rule we construct the following table using a spreadsheet.

	A	B	C	D	E
1	**Exercise 10.2.b**				
2	*∫sinxdx evaluated from a = 0 to b = π with n = 4*				
3	Numerical integration by Simpson's method follows				
4	Given	a=	0		
5		b=	3.14159		
6		n=	4		
7	Then,	h = (b-a)/n =	0.7854		
8				Multiplier	Products
9		x_0=a=	0.00000		
10		y_0=sinx_0=	0.00000	1	0.0000
11		x_1=a+h=	0.78540		
12		y_1=sinx_1=	0.70711	4	2.8284
13		x_2=a+2h=	1.57080		
14		y_2=sinx_2=	1.00000	2	2.0000
15		x_3=a+3h=	2.35619		
16		y_3=sinx_3=	0.70711	4	2.8284
17		x_4=b=	3.14159		
18		y_4=sinx_4=	0.00000	1	0.0000
19			Sum of Products =		7.6569
20	Area = (h/3)*(Sum of Products) = (1/12)*8.31905 =				**2.0046**

We create and save a function m–file. We name it exer_10_2_b.m as shown below.

```
function y = exer_10_2_b(x)
y = sin(x);
```

We write and execute the following MATLAB script:

```
y_std=quad('exer_10_2_b',0,pi)
```

```
y_std =
    2.0000
```

c. The exact value is

$$\int_0^1 \frac{1}{x^2+1}dx = \tan^{-1}x\Big|_0^1 = \frac{\pi}{4} = 0.7854$$

To use Simpson's rule we construct the following table using a spreadsheet.

	A	B	C	D	E
1	**Exercise 10.2.c**				
2	$\int(1/(x^2+1))dx$ evaluated from a = 0 to b = 1 with n = 4				
3	Numerical integration by Simpson's method follows				
4	Given	a=	0		
5		b=	1		
6		n=	4		
7	Then,	h = (b-a)/n =	0.2500		
8				Multiplier	Products
9		x_0=a=	0.00000		
10		$y_0=1/(x_0^2+1)$=	1.00000	1	1.0000
11		x_1=a+h=	0.25000		
12		$y_1=1/(x_1^2+1)$=	0.94118	4	3.7647
13		x_2=a+2h=	0.50000		
14		$y_2=1/(x_2^2+1)$=	0.80000	2	1.6000
15		x_3=a+3h=	0.75000		
16		$y_3=1/(x_3^2+1)$=	0.64000	4	2.5600
17		x_4=b=	1.00000		
18		$y_4=1/(x_4^2+1)$=	0.50000	1	0.5000
19			Sum of Products =		9.4247
20	Area = (h/3)*(Sum of Products) = (1/12)*8.31905 =				**0.7854**

We create and save a function m–file. We name it exer_10_2_c.m as shown below.

```
function y = exer_10_2_c(x)
y = 1./(x.^2+1);
```

We write and execute the following MATLAB script:

```
y_std=quad('exer_10_2_c',0,1)
```

```
y_std =
    0.7854
```