

Calculus, Johnston & Mathews, *MATLAB* Manual

Isom Herron, R.P.I.

Contents

1	Information for Instructors	1
1.1	Getting Started	2
1.1.1	Student Accessibility to MATLAB	2
1.1.2	Working in a MATLAB Environment	3
2	Plotting, Limits and the Derivative	5
2.1	Plotting simple functions with MATLAB	5
2.2	Taking limits with MATLAB	8
2.3	The Derivative with MATLAB	9
2.4	Solving Equations by MATLAB	10
2.5	Plotting Parametric Equations	11
2.6	Plotting Polar Equations	12
2.7	Approximate solutions of equations	12
2.8	Applications of the derivative	13
3	Integration	14
3.1	A Definite Integral	14
3.2	Integration by Parts; An Indefinite Integral	15
3.3	Approximation of Integrals	16
3.3.1	Trapezoidal Rule	16
3.3.2	Taylor Series	16

1 Information for Instructors

Many of the exercise sets in *Calculus* by Johnson and Mathews contain technology exercises, marked with the icon **T** which are intended to be

solved with the use of a computer algebra system (CAS) such as MATLAB. In addition, each chapter of the text contains at least one student project. These projects are designed to give students substantial experience in solving multistep problems, usually with the aid of a graphical calculator or a CAS. The purpose of this manual is to provide a brief introduction to the MATLAB computing environment, specifically geared to the textbook.

If you are considering having students use MATLAB for some of the Technology exercises or Student Projects, please spend a few minutes reading this section. It contains general advice and suggestions for the successful implementation of MATLAB in your course.

1.1 Getting Started

1.1.1 Student Accessibility to MATLAB

First you may need to determine where your students will have access to MATLAB. For instance, if you plan on having students do their work on campus in a computer lab, then it will be necessary that MATLAB is readily available in the lab or labs, where the students will be expected to work.

If possible, the labs should have the latest version of MATLAB (presently Version 6.1 at the time of writing). Up-to-date information about MATLAB can be found on the web at <http://www.mathworks.com>. If you or your school is planning on purchasing MATLAB for use in a computer lab, the information you may need can also be obtained from the Mathworks website. An academic site license can be obtained at a considerable discount. Once you purchase and obtain your site license for MATLAB, have your computer administrator install the program on the lab machines. For instance, on our campus, MATLAB is available on the computer network as soon as the students log-in.

If your school chooses not to obtain a site license, you may need to make arrangements to have your students purchase individual copies of MATLAB. The Student Edition of MATLAB is recommended. It is described at the URL

[http://www.mathworks.com/products/student version/](http://www.mathworks.com/products/student%20version/). You may want to check to see if the campus bookstore or computer store will stock the software for you.

You might want to spend some time working with MATLAB in the same environment as your students. If they will be working in an on-campus lab,

spend some time in the lab. Learn any special access procedures and how to launch MATLAB. Check to see if you can access materials in a course archive or on the WWW, save a workspace file, exit MATLAB, transfer a file via e-mail or onto a diskette, and log out of the computer.

1.1.2 Working in a MATLAB Environment

For a vast range of applications, academic and non-academic, MATLAB has become the technical computing environment of choice. To fully survey all of its possible applications would take us far beyond the scope of the textbook and that of this guidebook. However, MATLAB was designed to be user-friendly for mathematical calculations involving matrices and vectors. Its use for solving calculus problems has great power and appeal as well.

The code described in this manual was written specifically to solve the problems in the text. It was tested in an *X window system*, both locally and remotely. It was also run in the student edition of MATLAB on a Macintosh machine.

Once MATLAB is opened the window which appears is the *command window*. The *command line* is the input line next to the MATLAB prompt. Navigating the command line will depend on the system one uses, but most will permit editing lines since specific MATLAB syntax must be observed.

The MATLAB CAS commands are contained in the **Symbolic Math Toolbox**. MATLAB will generally produce a numerical answer to a calculation unless a symbolic answer is desired. The command *help symbolic* will give a list of the symbolic functions available. They range from Calculus, Linear Algebra, Solutions of Differential and Functional Equations, Integral Transforms, Conversions and Plots. Use the command *demo* to explore the wide range of applications available in MATLAB.

MATLAB is truly a language, and a sequence of commands used over and over again can be composed in a program called an *M-file*. There are two types of *M-files*, *scripts* and *functions*. The use of an *M-file* may become necessary when MATLAB, on a rare occasion, does not have a built-in command or function to solve a problem in the text.

The commands used in this manual are listed below:

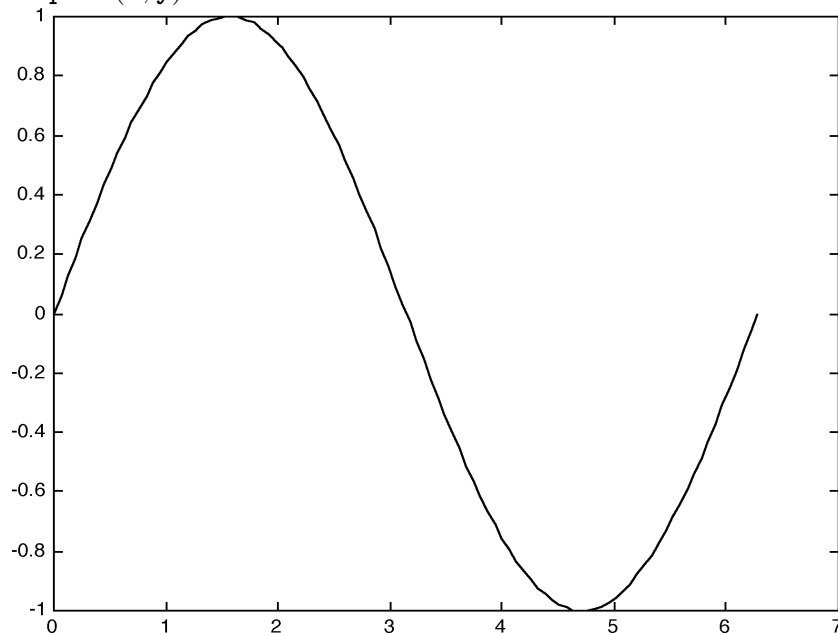
Command	Function
syms	Short cut for constructing symbolic objects
ezplot	Easy-to-use function plotter
limit	Limit of a symbolic expression
solve	Symbolic solution of functional equations
double	Convert symbolic expression to numerical
plot	Linear plot
linspace	Linearly spaced vector
subs	Symbolic substitution
diff	Differentiate
int	Integrate
pretty	Pretty print a symbolic expression
taylor	Taylor series
polar	Plot in polar coordinates
trapz	Trapezoidal integration

2 Plotting, Limits and the Derivative

2.1 Plotting simple functions with MATLAB

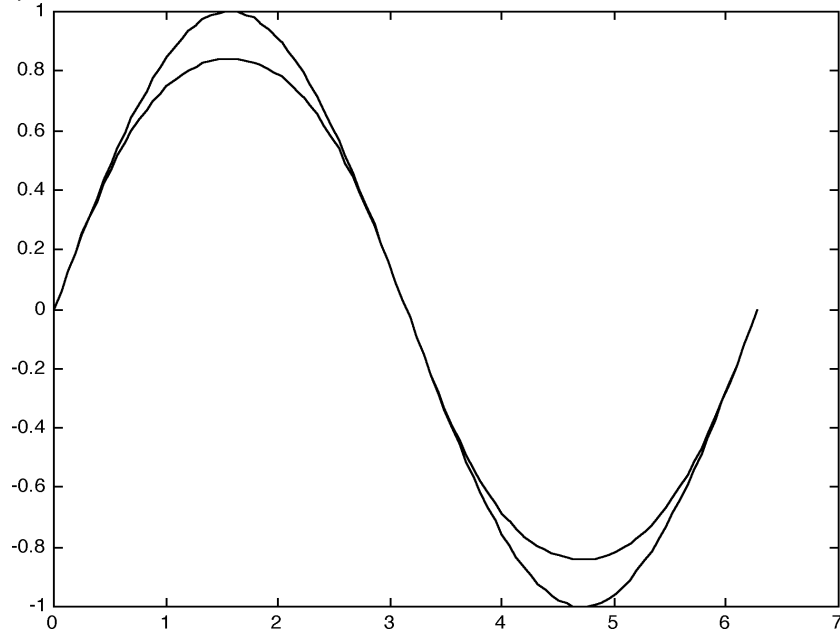
One of the most useful benefits of a program such as MATLAB is in plotting functions. There are several ways in which this may be carried out. The most common command for two-dimensional graphs is *plot*. Data arrays are formed and plotted on the horizontal and vertical axes, and the points are connected by line segments. If there are not enough line segments, the curve may appear choppy. The command to segment the interval is *linspace*, which is actually an internal function, since it is made up of more than one command. You may adjust the number of data points as needed. However there is a default number of 100, which is often adequate. To perform a plot, the process is illustrated on a **T** exercise from Section 1.2, p.19, #39.

```
>> syms x y y1 y2
>> x = linspace(0, 2 * pi);
>> y = sin(x);
>> y1 = sin(y);
>> y2 = sin(y1);
>> plot(x, y)
```



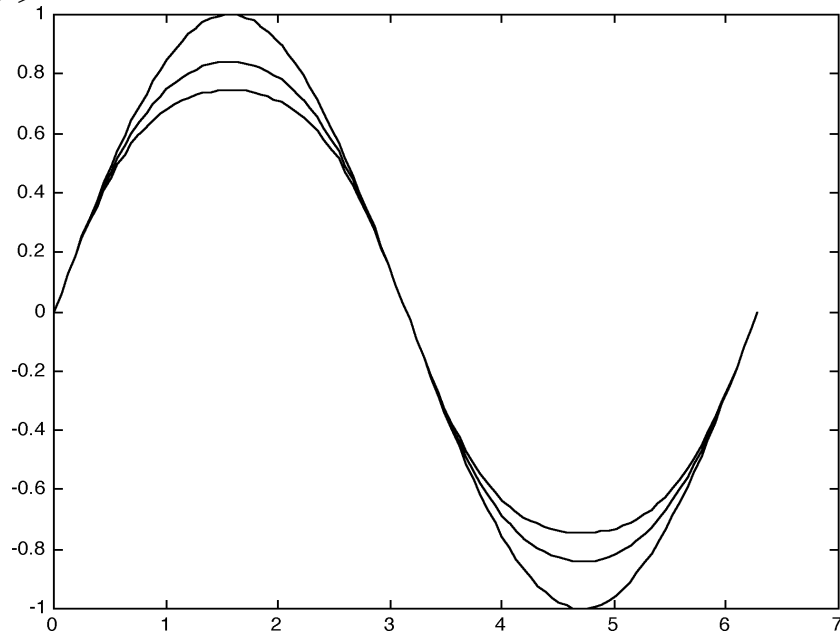
```
>> plot(x, y, x, y1)
```

```
>>
```



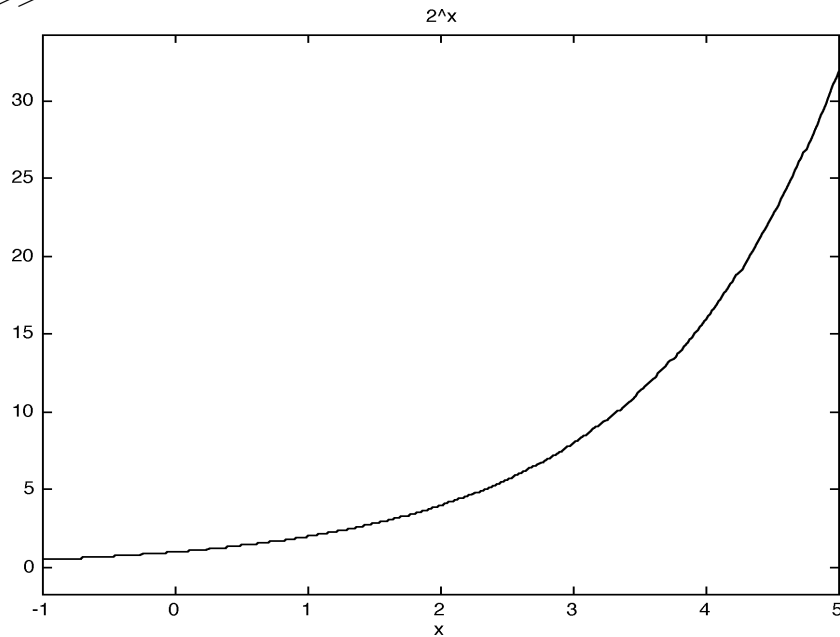
```
>> plot(x, y, x, y1, x, y2)
```

```
>>
```

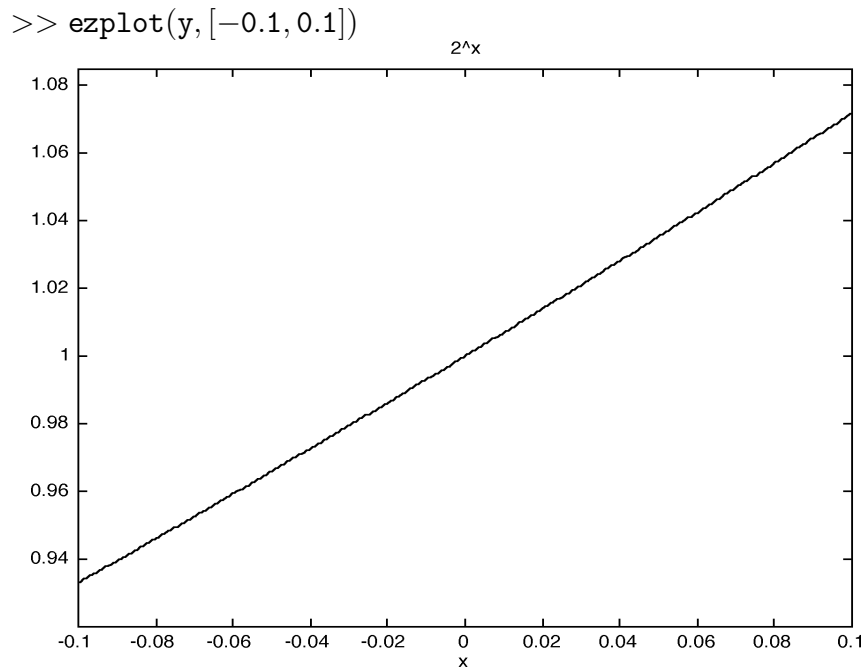


Alternatively, the command *ezplot* may be used to perform a plot. This is particularly recommended when only one function over one interval is needed. This is illustrated on a **T** exercise from Section 1.4, p.35, #13.

```
>> syms x y
>> y = 2^x;
>> ezplot(y, [-1 5])
>>
```



This shows the graph of $y = 2^x$ for $-1 < x < 5$.



This shows zooming in near the point $(0, 1)$.

2.2 Taking limits with MATLAB

Just about all of the important processes of calculus may be performed using MATLAB. The first one to be shown is that of taking limits. This is illustrated on a **T** exercise in section 1.5, p.47, #13 of the text. If no limit point is given, it is automatically taken to be 0.

```
>> syms h
>> limit(sin(h)/h)
ans =
1
```

Another possibility is that the limit does not exist. MATLAB identifies this difficulty, which is also on p.47 in the text.

```
>> syms x
>> limit(tan(x), pi/2)
```

```
ans =
NaN
>>help NaN
NaN Not-a-Number.
NaN is the IEEE arithmetic representation for Not-a-Number.
A NaN is obtained as a result of mathematically undefined
operations like 0.0/0.0 and inf-inf.
```

A more complicated example of a limit that does exist is the following T exercise from section 1.6, p.63, #17.

```
>> limit((tan(x) - 1)/(x - pi/4), pi/4)
ans =
2
```

It may also be necessary to take a **one-sided limit**. MATLAB will also evaluate such a limit as is illustrated on an example from Section 1.6, p.63. The first place after the function is the variable v which is to approach the limit c from the left, given in single quotes.

```
>> syms v c
>> limit(sqrt(1 - (v/c)^2), v, c, 'left')
ans =
0
```

2.3 The Derivative with MATLAB

The definition of the derivative is based on taking a limit. Hence it is very natural to apply the techniques just used. The derivation of the derivative of $\cos(x)$ follows easily from its definition in Example 1 on p.123:

$$\frac{d}{dx} \cos(x) = \lim_{h \rightarrow 0} \frac{\cos(x+h) - \cos(x)}{h}.$$

The MATLAB derivation is as follows.

```
>> limit((cos(x+h) - cos(x))/h, h, 0)

ans =
```

```
-sin(x)
```

The derivatives of other functions such as b^x may also be derived. This is motivated on p.134 in the text.

```
>>syms h b
>>limit((b^h-1)/h,h,0)

ans =

log(b)

>> % The log is the natural logarithm ln.
```

2.4 Solving Equations by MATLAB

The command `solve('f(x) = 0')` in MATLAB solves the equation $f(x) = 0$ for the desired values of x . As the need arises this command will be greatly used. A first illustration is from section 2.7, a **T** exercise on p.152, #43. The command gives nine roots of the equation $2^x = x^8$, in terms of the built-in function `lambertw`, which at this stage is not too useful to us. MATLAB, the genie that it is, desires to please us by obtaining as exact an answer as possible.

```
>>[x] =solve('2^x=x^8')

x =

[-8 * lambertw(-1/8 * log(2))/log(2)]
[-8 * lambertw(-1, -1/8 * log(2))/log(2)]
[-8 * lambertw(-1/8 * log(2) * (1/2 * 2^(1/2) + 1/2 * i * 2^(1/2)))/log(2)]
[-8 * lambertw(-1/8 * i * log(2))/log(2)]
[-8 * lambertw(-1/8 * log(2) * (-1/2 * 2^(1/2) + 1/2 * i * 2^(1/2)))/log(2)]
[-8 * lambertw(1/8 * log(2))/log(2)]
[-8 * lambertw(-1/8 * log(2) * (-1/2 * 2^(1/2) - 1/2 * i * 2^(1/2)))/log(2)]
[-8 * lambertw(1/8 * i * log(2))/log(2)]
[-8 * lambertw(-1/8 * log(2) * (1/2 * 2^(1/2) - 1/2 * i * 2^(1/2)))/log(2)]
```

However, MATLAB, being the practical tool it is, has another command, `double` which converts a symbolic expression to a numerical one.

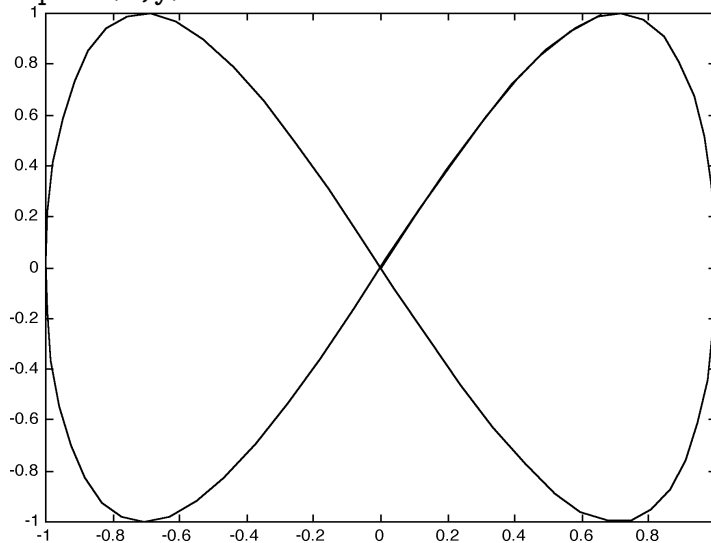
```
>> double(x)
ans =
 1.1000
43.5593
0.6972 + 0.8014i
-0.0850 + 0.9890i
-0.7007 + 0.6283i
-0.9231
-0.7007 - 0.6283i
-0.0850 - 0.9890i
0.6972 - 0.8014i
```

Only the positive real values of x are of interest to us. Thus for $0 < x < 1.1$, $x^8 < 2^x$, while for $1.1 < x < 43.56$, $2^x < x^8$, but for $x > 43.56$, $2^x > x^8$.

2.5 Plotting Parametric Equations

The need may arise to plot a curve given in parametric form. The plot command is capable of handling this, as illustrated on a **T** exercise from section 3.4, p.232, #79.

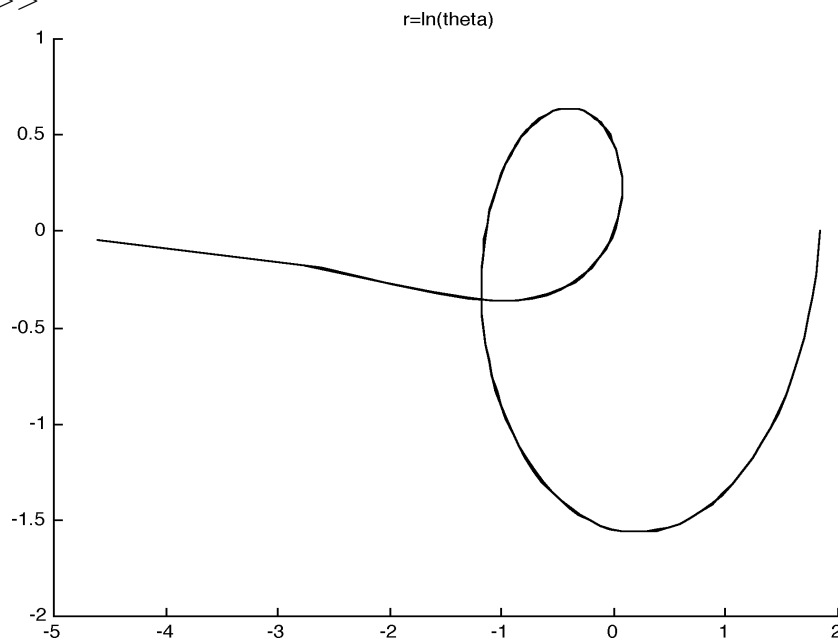
```
>>t=0:0.1:7;
>>x=sin(t);
>>y=sin(2*t);
>>plot(x,y)
```



2.6 Plotting Polar Equations

Plots in polar coordinates can be done using the command `polar(t,r)`, where t is the angle in radians and r is the radius vector. This is illustrated on a **T** exercise from section 6.3, p.471, #39.

```
>> % Graph the curve described by r=ln(theta), 0<theta<2*pi
>>t=linspace(0,2*pi);
>>r=log(t);
Warning: Log of zero.
>>polar(t,r)
>>title('r=ln(theta)')
>>
>>
```



2.7 Approximate solutions of equations

Newton's method is fast and accurate in finding the solution of the equation $f(x) = 0$ when a reasonably good initial guess is known. The `solve` command will give the desired answers. This is illustrated on a **T** exercise in Section 4.2, on p.275, #35.

```
>>solve('x^4-5*x^3+21*x^2+13*x+49=0');
```

```
>>double(ans)
ans =
 2.9956- 4.0003i
 2.9956+ 4.0003i
-0.4956+ 1.3101i
-0.4956- 1.3101i
```

2.8 Applications of the derivative

The command *diff* performs the symbolic differentiation of a function. This technique is illustrated on an interesting but complicated **T** exercise from section 4.7, p.319, #23.

```
>>syms A r h
>>A=pi^r^2/2-r^2*asin(1-h/r)-(r-h)*sqrt(h*(2*r-h))

A =

1/2*(pi^r)^2-r^2*asin(1-h/r)-(r-h)*(h*(2*r-h))^(1/2)

>>Ah=diff(A,h)

Ah=
r/(1-(1-h/r)^2)^(1/2)+(h*(2*r-h))^(1/2)
-1/2*(r-h)/(h*(2*r-h))^(1/2)*(2*r-2*h)

>>Ah1=subs(Ah,r,1.5)

Ah1=
3/2/(1-(1-2/3*h)^2)^(1/2)+(h*(3-h))^(1/2)
-1/2*(3/2-h)/(h*(3-h))^(1/2)*(3-2*h)

>>Ah2=subs(Ah1,h,2)

Ah2 =

3/16*8^(1/2)*9^(1/2)+7/8*2^(1/2)

>>hrate=0.5/(6*Ah2)
```

```

hrate =

1/2/(9/8*8^(1/2)*9^(1/2)+21/4*2^(1/2))

>>double(hrate)
ans =
    0.0295
>>

```

3 Integration

3.1 A Definite Integral

The command *int* may be used to obtain both indefinite and definite integrals. Whenever possible, MATLAB attempts to provide an exact answer. This is illustrated on a **T** example from section 5.6, p.393, #35. This example is somewhat involved and the code is developed next.

```

>>% First, find the area of the region removed. It is determined
by the points where the circle and the parabola meet.
>>syms x
>>solve('4-sqrt(16-x^2)=-0.4*x^2+3.2')

ans =

[ -2.4356475028411506249717241735830]
[ 2.4356475028411506249717241735830]

>>% The area of the removed section is then obtained as a definite
integral between those limits.
>>A=int(-0.4*x^2+3.2-(4-sqrt(16-x^2)),x,-2.4356475,2.4356475)

A =

-1037004934973087065929065648105004366319763483/133804471
191183738849214309635890169035882496+4113442940027205/31691

```

```

2650057057350374175801344*354506281325390153376730559031
^(1/2)+16*asin(1371147646675735/2251799813685248)
>>double(A)
ans =
    10.4530
>> % The remaining area, and the mass, are given in the following
calculations.
>>Ar=16*pi-10.4530
Ar =
    39.8125
>>M=Ar*0.5*8920*1e-6
M =
    0.17756
>>

```

3.2 Integration by Parts; An Indefinite Integral

MATLAB will quickly evaluate indefinite or definite integrals which, if done by hand, would require lengthy algebraic or trigonometric calculations. For instance, here is the MATLAB calculation for Example 5 in section 5.7:

```

>>syms t
>>A=int(exp(t)*(65+10*cos(pi*t/12)-5*cos(pi*t/6)-15*sin(pi*t/12)),t)

A =

    65*exp(t)+10/(1+1/144*pi^2)*exp(t)*cos(1/12*pi*t)
+5/6*pi/(1+1/144*pi^2)*exp(t)*sin(1/12*pi*t)
-5/(1+1/36*pi^2)*exp(t)*cos(1/6*pi*t)
-5/6*pi/(1+1/36*pi^2)*exp(t)*sin(1/6*pi*t)
+5/4*pi/(1+1/144*pi^2)*exp(t)*cos(1/12*pi*t)
-15/(1+1/144*pi^2)*exp(t)*sin(1/12*pi*t)
>>
>>% The answer is easier to read if we pretty it up.
>>pretty(A)

```

$$65 \exp(t) + 10 \frac{\exp(t) \cos(1/12 \pi t)}{1 + 1/144 \pi^2} + 5/6 \frac{\pi \exp(t) \sin(1/12 \pi t)}{1 + 1/144 \pi^2}$$

$$\begin{array}{r}
\frac{\exp(t) \cos(1/6 \pi t)}{1 + 1/36 \pi} \\
- 5 \\
\frac{\pi \exp(t) \cos(1/12 \pi t)}{1 + 1/144 \pi} \\
+ 5/4
\end{array}
\qquad
\begin{array}{r}
\frac{\pi \exp(t) \sin(1/6 \pi t)}{1 + 1/36 \pi} \\
- 5/6 \\
\frac{\exp(t) \sin(1/12 \pi t)}{1 + 1/144 \pi} \\
- 15
\end{array}$$

3.3 Approximation of Integrals

3.3.1 Trapezoidal Rule

There are three built-in MATLAB functions for numerically evaluating an integral over a finite range. They are *trapz*, *quad* and *quad8*. The trapezoid rule is employed in the first of those commands. It is illustrated in solving a **T** exercise from section 6.4 on p.479, #27. The purpose is to evaluate $\int_0^{\pi/2} \sqrt{(a^2 - b^2) \sin^2 x + b^2} dx$. Since the default number in *linspace* is 100, the effect is to compute T_{100} .

```

>>x=linspace(0,pi/2);
>>a=17.9435;
>>b=4.5511;
>>
>>f=sqrt((a^2-b^2)*(sin(x).^2)+b^2);
>>trapz(x,f)
ans =
19.2710

```

3.3.2 Taylor Series

MATLAB can be used to find the Taylor series of a function. The command *taylor(f)* gives (by default) the fifth order Maclaurin polynomial approximation to f , that is, the Taylor series about $x = 0$. Here we employ *taylor(f,n)* which is the $(n-1)$ -st order Maclaurin polynomial. Several other parameters can be specified and a complete description may be found under *help taylor*. For very smooth functions, the Taylor series method will

give an accurate approximation to a definite integral, using few terms when the interval of integration is short. The method is developed in an example at the end of section 7.7 on p.621 in the text. In what follows the method is illustrated on a **T** exercise, p.624, #21.

```
>>syms f x
>>f=taylor(sin(x^2),15)

f =

x^2-1/6*x^6+1/120*x^10-1/5040*x^14

>>I=int(f,x,0,pi/2)

I =

1/24*pi^3-1/5376*pi^7+1/2703360*pi^11-1/2477260800*pi^15

>>double(I)
ans =
0.8274
```