2.1 Lesson 1: A Minimum MATLAB Session

Goal: To learn how to log on, invoke MATLAB, do a few trivial calculations, quit MATLAB, and log off.

Time Estimates

Lesson: 10 minutes
Exercises: 30 minutes

What you are going to learn

- How to do simple arithmetic calculations. The arithmetic operators are

+	addition,
−	subtraction,
*	multiplication,
/	division, and
^	exponentiation.

- How to assign values to variables.
- How to suppress screen output.
- How to control the appearance of floating point numbers on the screen.
- How to quit MATLAB.

The MATLAB commands/operators used are

```
+, -, *, /, ^, ;
sin, cos, log
format
quit
```

In addition, if you do the exercises, you will learn more about arithmetic operations, exponentiation and logarithms, trigonometric functions, and complex numbers.

Method: Log on and launch MATLAB. Once the MATLAB command window is on the screen, you are ready to carry out the first lesson. Some commands and their output are shown in Fig. 2.1. Go ahead and reproduce the results.

Read the lessons and solve the questions marked at the end of each lesson

```
>> 2 + 2

ans =

     4

>> x = 2 + 2

x =

     4

>> y = 2^2 + log(pi)*sin(x);

>> y

y =

    3.1337

>> theta = acos(-1)

theta =

    3.1416

>> format short e
>> theta

theta =

   3.1416e+000

>> format long
>> theta

theta =

   3.141592653589793

>> quit
```

Enter `2+2` and hit the return/enter key. Note that the result of an unassigned expression is saved in the default variable *ans*.

You can also assign the value of an expression to a variable.

A semicolon at the end suppresses screen output. MATLAB remembers *y*, though. You can recall the value *y* by simply typing *y*.

MATLAB knows trigonometry. Here is arccosine of -1.

The floating point output display is controlled by the `format` command. Here are two examples. More information will be provided on this later.

Quit MATLAB. You can also quit by selecting Quit from the file menu on Macs and PCs.

Figure 2.1: Lesson 1: Some simple calculations in MATLAB.

EXERCISES

1. **Arithmetic operations:** Compute the following quantities:
   - $\frac{2^5}{2^5-1}$ and compare with $(1-\frac{1}{2^5})^{-1}$.
   - $3\frac{\sqrt{5}-1}{(\sqrt{5}+1)^2}-1$. The square root $\sqrt{x}$ can be calculated with the command `sqrt(x)` or `x^0.5`.
   - Area $=\pi r^2$ with $r=\pi^{\frac{1}{3}}-1$. (π is `pi` in MATLAB.)

2. **Exponential and logarithms:** The mathematical quantities e^x, $\ln x$, and $\log x$ are calculated with `exp(x)`, `log(x)`, and `log10(x)`, respectively. Calculate the following quantities:
   - e^3, $\ln(e^3)$, $\log_{10}(e^3)$, and $\log_{10}(10^5)$.
   - $e^{\pi\sqrt{163}}$.
   - Solve $3^x=17$ for x and check the result. (The solution is $x=\frac{\ln 17}{\ln 3}$. You can verify the result by direct substitution.)

3. **Trigonometry:** The basic MATLAB trigonometric functions are `sin`, `cos`, `tan`, `cot`, `sec`, and `csc`. The inverses, e.g., arcsin, arctan, etc., are calculated with `asin`, `atan`, etc. The same is true for hyperbolic functions. The inverse function `atan2` takes two arguments, y and x, and gives the four-quadrant inverse tangent. The argument of these functions must be in radians. Calculate the following quantities:
   - $\sin\frac{\pi}{6}$, $\cos\pi$, and $\tan\frac{\pi}{2}$.
   - $\sin^2\frac{\pi}{6}+\cos^2\frac{\pi}{6}$. (Typing `sin^2(x)` for $\sin^2 x$ will produce an error).
   - $y=\cosh^2 x-\sinh^2 x$, with $x=32\pi$.

4. **Complex numbers:** MATLAB recognizes the letters i and j as the imaginary number $\sqrt{-1}$. A complex number $2+5i$ may be input as `2+5i` or `2+5*i` in MATLAB. The former case is always interpreted as a complex number, whereas the latter case is taken as complex only if i has not been assigned any local value. The same is true for j. This kind of context dependence, for better or worse, pervades MATLAB. Compute the following quantities:
   - $\frac{1+3i}{1-3i}$. Can you check the result by hand calculation?
   - $e^{i\frac{\pi}{4}}$. Check the Euler's Formula $e^{ix}=\cos x+i\sin x$ by computing the right-hand side too, i.e., compute $\cos(\pi/4)+i\sin(\pi/4)$.
   - Execute the commands `exp(pi/2*i)` and `exp(pi/2i)`. Can you explain the difference between the two results?

2.2 Lesson 2: Creating and Working with Arrays of Numbers

Goal: To learn how to create arrays and vectors and how to perform arithmetic and trigonometric operations on them.

An *array* is a list of numbers or expressions arranged in horizontal rows and vertical columns. When an array has only one row or column, it is called a *vector*. An array with m rows and n columns is called a *matrix* of size $m \times n$. See Section 3.1 for more information.

Time Estimates
Lesson: 15 minutes
Exercises: 45 minutes

What you are going to learn

- How to create row and column vectors.
- How to create a vector of n numbers linearly (equally) spaced between two given numbers a and b.
- How to do simple arithmetic operations on vectors.
- How to do *array operations*:

`.*`	term-by-term multiplication,
`./`	term-by-term division, and
`.^`	term-by-term exponentiation.

- How to use trigonometric functions with array arguments.
- How to use elementary math functions such as square root, exponentials, and logarithms with array arguments.

This lesson deals primarily with 1-D arrays, i.e., vectors. One of the exercises introduces you to 2-D arrays, i.e., matrices. There are many mathematical concepts associated with vectors and matrices that we do not mention here. If you have some background in linear algebra, you will find that MATLAB is set up to do almost any matrix computation (e.g., inverse, determinant, rank).

Method: You already know how to launch MATLAB. Go ahead and try the commands shown in Fig. 2.2. Once again, you are going to reproduce the results shown.

```
>> x = [1 2 3]

x =
     1     2     3

>> y = [2; 1; 5]

y =
     2
     1
     5

>> z = [2 1 0];
>> a = x + z

a =
     3     3     3

>> b = x + y

??? Error using ==> plus
Matrix dimensions must agree.

>> a = x.*z

a =
     2     2     0

>> b = 2*a

b =
     4     4     0

>> x = linspace(0,10,5)

x =
         0    2.5000    5.0000    7.5000   10.0000

>> y = sin(x);

>> z = sqrt(x).*y

z =
         0    0.9463   -2.1442    2.5688   -1.7203
```

x is a row vector with three elements.

y is a column vector with three elements.

You can add (or subtract) two vectors of the same size.

But you cannot add (or subtract) a row vector to a column vector.

You can multiply (or divide) the elements of two same-sized vectors term by term with the *array operator* `.*` (or `./`).

But multiplying a vector with a scalar does not need any special operation (no dot before the `*`).

Create a vector *x* with 5 elements *lin*early *space*d between 0 and 10.

Trigonometric functions `sin`, `cos`, etc., as well as elementary math functions `sqrt`, `exp`, `log`, etc., operate on vectors term by term.

Figure 2.2: Lesson 2: Some simple calculations with vectors.

EXERCISES

1. **Equation of a straight line:** The equation of a straight line is $y = mx + c$, where m and c are constants. Compute the y-coordinates of a line with slope $m = 0.5$ and the intercept $c = -2$ at the following x-coordinates:

$$x = 0, \quad 1.5, \quad 3, \quad 4, \quad 5, \quad 7, \quad 9, \text{ and } 10.$$

   [Note: Your command should not involve any array operators because your calculation involves multiplication of a vector with a scalar m and then addition of another scalar c.]

2. **Multiply, divide, and exponentiate vectors:** Create a vector t with 10 elements: 1, 2, 3, ..., 10. Now compute the following quantities:
   - $x = t \sin(t)$.
   - $y = \frac{t-1}{t+1}$.
   - $z = \frac{\sin(t^2)}{t^2}$.

3. **Points on a circle:** All points with coordinates $x = r\cos\theta$ and $y = r\sin\theta$, where r is a constant, lie on a circle with radius r, i.e., they satisfy the equation $x^2 + y^2 = r^2$. Create a column vector for θ with the values 0, $\pi/4$, $\pi/2$, $3\pi/4$, π, and $5\pi/4$. Take $r = 2$ and compute the column vectors x and y. Now check that x and y indeed satisfy the equation of a circle, by computing the radius $r = \sqrt{(x^2 + y^2)}$. [To calculate r you will need the array operator `.^` for squaring x and y. Of course, you could compute x^2 by `x.*x` also.]

4. **The geometric series:** This is funky! You know how to compute x^n element by element for a vector x and a scalar exponent n. How about computing n^x, and what does it mean? The result, again, is a vector with elements n^{x_1}, n^{x_2}, n^{x_3}, etc. The sum of a geometric series $1 + r + r^2 + r^3 + \cdots + r^n$ approaches the limit $\frac{1}{1-r}$ for $r < 1$ as $n \to \infty$. Create a vector n of 11 elements from 0 to 10. Take $r = 0.5$ and create another vector $x = [r^0 \quad r^1 \quad r^2 \quad \cdots \quad r^n]$ with the `x=r.^n` command. Now take the sum of this vector with the command `s=sum(x)` (s is the sum of the actual series). Calculate the limit $\frac{1}{1-r}$ and compare the computed sum s. Repeat the procedure taking n from 0 to 50 and then from 0 to 100.

5. **Matrices and vectors:** Go to Fig. 3.1 on page 67 and reproduce the results. Now create a vector and a matrix with the following commands: `v=0:0.2:12;` and `M=[sin(v); cos(v)];` (see Section 3.1.4 on page 72 for use of ":" in creating vectors). Find the sizes of v and M using the **`size`** command. Extract the first 10 elements of each row of the matrix and display them as column vectors.

2.3 Lesson 3: Creating and Printing Simple Plots

Goal: To learn how to make a simple 2-D plot in MATLAB and print it out.

Time Estimates

Lesson: 10 minutes
Exercises: 40 minutes

What you are going to learn

- How to generate x- and y-coordinates of 100 equidistant points on a unit circle.
- How to plot x vs. y and thus create the circle.
- How to set the scale of the x-axis and the y-axis to be the same, so that the circle looks like a circle and not an ellipse.
- How to label the axes with text strings.
- How to title the graph with a text string.
- How to get a hard copy of the graph.

The MATLAB commands used are

`plot`	creates a 2-D line plot,
`axis`	changes the aspect ratio of the x-axis and the y-axis,
`xlabel`	annotates the x-axis,
`ylabel`	annotates the y-axis,
`title`	puts a title on the plot, and
`print`	prints a hard copy of the plot.

This lesson teaches you the most basic graphics commands. The exercises take you through various types of plots, overlay plots, and more involved graphics.

Method: You are going to draw a circle of unit radius. To do this, first generate the data (x- and y-coordinates of, say, 100 points on the circle), then plot the data, and finally print the graph. For generating data, use the parametric equation of a unit circle:

$$x = \cos\theta, \quad y = \sin\theta, \quad 0 \leq \theta \leq 2\pi.$$

In the sample session shown in Fig. 2.3, only the commands are listed. You should see the output on your screen.

```
>> theta = linspace(0,2*pi,100);
```
Create a linearly spaced 100-elements-long vector θ.

```
>> x = cos(theta);
```
Calculate x- and y-coordinates.

```
>> y = sin(theta);
```

```
>> plot(x,y)
```
Plot x vs. y (see Section 6.1).

```
>> axis('equal');
```
Set the length scales of the two axes to be the same.

```
>> xlabel('x')
```
Label the x-axis with `x`.

```
>> ylabel('y')
```
Label the y-axis with `y`.

```
>> title('Circle of unit radius')
```
Put a title on the plot.

```
>> print
```
Print on the default printer.

Figure 2.3: Lesson 3: Plotting and printing a simple graph.

Comments:

- After you enter the command `plot(x,y)`, you should see an ellipse in the figure window. MATLAB draws an ellipse rather than a circle because of its default rectangular axes. The command `axis('equal')` directs MATLAB to use the same scale on both axes, so that a circle appears as a circle. You can also use `axis('square')` to override the default rectangular axes.
- The arguments of the `axis`, `xlabel`, `ylabel`, and `title` commands are text strings. Text strings are entered within single right-quote (') characters. For more information on text strings, see Section 3.3 on page 77.
- The `print` command sends the current plot to the printer connected to your computer.

EXERCISES

1. **A simple sine plot:** Plot $y = \sin x, \quad 0 \leq x \leq 2\pi$, taking 100 linearly spaced points in the given interval. Label the axes and put "Plot created by *yourname*" in the title.
2. **Line styles:** Make the same plot as in Exercise 1, but rather than displaying the graph as a curve, show the unconnected data points. To display the data points with small circles, use `plot(x,y,'o')`. [Hint: You may look into Section 6.1 on page 175 if you wish.] Now combine the two plots with the command `plot(x,y,x,y,'o')` to show the line through the data points as well as the distinct data points.
3. **An exponentially decaying sine plot:** Plot $y = e^{-0.4x} \sin x, \quad 0 \leq x \leq 4\pi$, taking 10, 50, and 100 points in the interval. [Be careful about computing y. You need array multiplication between `exp(-0.4*x)` and `sin(x)`. See Section 3.2.1 on page 73 for more discussion on array operations.]
4. **Space curve:** Use the command `plot3(x,y,z)` to plot the circular helix $x(t) = \sin t, \quad y(t) = \cos t, \quad z(t) = t, \quad 0 \leq t \leq 20$.
5. **On-line help:** Type `help plot` on the MATLAB prompt and hit return. If too much text flashes by the screen, type `more on`, hit return, and then type `help plot` again. This should give you paged screen output. Read through the on-line help. To move to the next page of the screen output, simply press the spacebar.
6. **Log-scale plots:** The plot commands `semilogx`, `semilogy`, and `loglog` plot the x-values, the y-values, and both x- and y-values on a $\log_{10}$ scale, respectively. Create a vector `x=0:10:1000`. Plot x vs. x^3 using the three log-scale plot commands. [Hint: First, compute `y=x.^3` and then use `semilogx(x,y)`, etc.]
7. **Overlay plots:** Plot $y = \cos x$ and $z = 1 - \frac{x^2}{2} + \frac{x^4}{24}$ for $0 \leq x \leq \pi$ on the same plot. You might like to read Section 6.1.5 on page 179 to learn how to plot multiple curves on the same graph. [Hint: You can use `plot(x,y,x,z,'--')` or you can plot the first curve, use the `hold on` command, and then plot the second curve on top of the first one.]
8. **Fancy plots:** Go to Section 6.1.6 and look at the examples of specialized 2-D plots given there. Reproduce any of the plots you like.
9. **A very difficult plot:** Use your knowledge of *splines* and *interpolation* to draw a lizard (just kidding).

2.4 Lesson 4: Creating, Saving, and Executing a Script File

Goal: To learn how to create script files and execute them in MATLAB.

A *script file* is a user-created file with a sequence of MATLAB commands in it. The file must be saved with a .m extension to its name, thereby, making it an *M-file*. A script file is executed by typing its name (without the .m extension) at the command prompt. For more information, see Section 4.1 on page 99.

Time Estimates

Lesson: 20 minutes
Exercises: 30 minutes

What you are going to learn

- How to create, write, and save a script file.
- How to execute the script file in MATLAB.

Unfortunately, creating, editing, and saving files are somewhat system-dependent tasks. The commands needed to accomplish these tasks depend on the operating system and the text editor you use. It is not possible to provide an introduction to these topics here. So, we assume that

- You know how to use a text editor on your UNIX system (for example, *vi* or *emacs*), or that you're using the built-in MATLAB editor on a Mac or a PC.
- You know how to open, edit, and save a file.
- You know which directory your file is saved in.

Method: Write a script file to draw the unit circle of Lesson 3. You are essentially going to write the commands shown in Fig. 2.3 in a file, save it, name it, and execute it in MATLAB. Follow the directions below.

1. Create a new file:
   - **On PCs and Macs:** Select File→New→Blank M-File from the File menu. A new edit window should appear.
   - **On UNIX workstations:** Type `!vi circle.m` or `!emacs circle.m` at the MATLAB prompt to open an edit window in *vi* or *emacs*.
2. Type the following lines into this file. Lines starting with a % sign are interpreted as comment lines by MATLAB and are ignored.

```
% CIRCLE - A script file to draw a unit circle
% File written by Rudra Pratap. Last modified 5/28/98
% ------------------------
theta=linspace(0,2*pi,100);      % create vector theta
x=cos(theta);                    % generate x-coordinates
y=sin(theta);                    % generate y-coordinates
plot(x,y);                       % plot the circle
axis('equal');                   % set equal scale on axes
title('Circle of unit radius')   % put a title
```

3. Write and save the file under the name circle.m:

   - **On PCs:** Select Save As... from the File menu. A dialog box should appear. Type `circle.m` as the name of the document. Make sure the file is being saved in the folder you want it to be in (the current working folder/directory of MATLAB). Click Save to save the file.

   - **On UNIX workstations:** You can use your favorite editor to write and save the file. After writing the file, quit the editor to get back to MATLAB.

4. Now get back to MATLAB and type the commands shown in Fig. 2.4 in the command window to execute the script file.

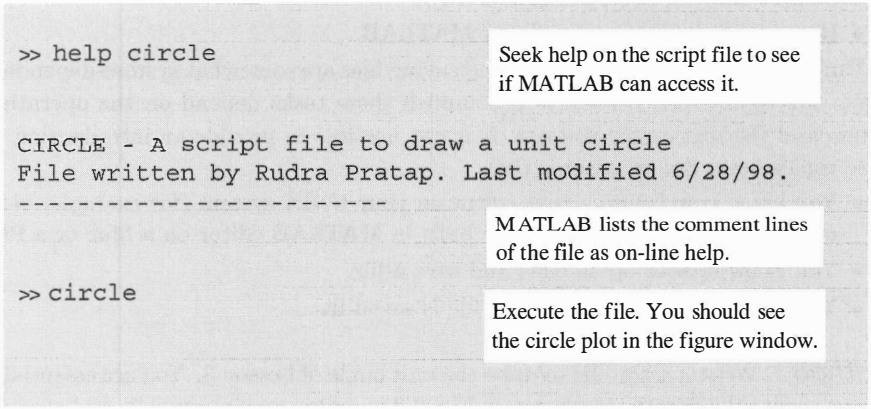


Figure 2.4: Executing a script file.

5. If you have the script file open in the MATLAB editor window, you can also execute the file by pressing the Run *file* icon (the little green arrowhead over a white page in the edit window nemu bar) or the F5 function key (see Debug menu).

EXERCISES

1. **Show the center of the circle:** Modify the script file circle.m to show the center of the circle on the plot, too. Show the center point with a "+". (Hint: See Exercises 2 and 7 of Lesson 3.)
2. **Change the radius of the circle:** Modify the script file circle.m to draw a circle of arbitrary radius r as follows:
   - Include the following command in the script file before the first executable line (`theta=...`) to ask the user to input (r) on the screen:
     ```
     r = input('Enter the radius of the circle:  ')
     ```
   - Modify the x- and y-coordinate calculations appropriately.
   - Save and execute the file. When asked, enter a value for the radius and press return.
3. **Variables in the workspace:** All variables created by a script file are left in the global workspace. You can get information about them and access them, too:
   - Type `who` to see the variables present in your workspace. You should see the variables `r, theta, x`, and `y` in the list.
   - Type `whos` to get more information about the variables and the workspace.
   - Type `[theta' x' y']` to see the values of θ, x, and y listed as three columns. All three variables are row vectors. Typing a single right quote (') after their names transposes them and makes them column vectors.
4. **Contents of the file:** You can see the contents of an M-file without opening the file with an editor. The contents are displayed by the `type` command. To see the contents of circle.m, type `type circle.m`.
5. **H1 line:** The first commented line before any executable statement in a script file is called the *H1 line*. It is this line that is searched by the `lookfor` command. Because the `lookfor` command is used to look for M-files with keywords in their description, you should put keywords in the H1 line of all M-files you create. Type `lookfor unit` to see what MATLAB comes up with. Does it list the script file you just created?
6. **Just for fun:** Write a script file that, when executed, greets you, displays the date and time, and curses your favorite TA or professor. [The commands you need are `disp, date, clock`, and possibly `fix`. See the on-line help on these commands before using them.]

2.5 Lesson 5: Creating and Executing a Function File

Goal: To learn how to write and execute a function file and to learn the difference between a script file and a function file.

A *function file* is also an M-file, just like a script file, except it has a function definition line on the top that defines the input and output explicitly. For more information, see Section 4.2.

Time Estimates
Lesson: 15 minutes
Exercises: 60 minutes

What you are going to learn

- How to open and edit an existing M-file.
- How to define and execute a function file.

Method: Write a function file to draw a circle of a specified radius, with the radius as the input to the function. You can either write the function file from scratch or modify the script file of Lesson 4. We advise you to select the latter option.

1. Open the script file circle.m:
   - **On PCs:** Select File→Open... from the File menu. Navigate and select the file circle.m from the Open dialog box. Double-click to open the file. The contents of the file should appear in an edit window.
   - **On UNIX workstations:** Type `!vi circle.m` or `!emacs circle.m` on the MATLAB prompt to open the file in a *vi* or *emacs* window.
2. Edit the file circle.m from Lesson 4 to look like the following:

```
function [x,y] = circlefn(r);
% CIRCLEFN - Function to draw a circle of radius r.
% File written by Rudra Pratap on 9/17/94. Last modified 7/1/98
% Call syntax:   [x,y] = circlefn(r);  or just:  circlefn(r);
% Input:      r = specified radius
% Output: [x,y] = the x- and y-coordinates of data points
theta=linspace(0,2*pi,100);     % create vector theta
x = r*cos(theta);               % generate x-coordinates
y = r*sin(theta);               % generate y-coordinates
plot(x,y);                      % plot the circle
axis('equal');                  % set equal scale on axes
title(['Circle of radius r =',num2str(r)])
                                % put a title with the value of r
```

Alternatively, you could select File→New→Function M-File and type all the lines in the new file.

3. Now write and save the file under the name circlefn.m as follows:

   - **On PCs:** Select Save As... from the File menu. A dialog box should appear. Type `circlefn.m` as the name of the document (usually, MATLAB automatically writes the name of the function in the document name). Make sure the file is saved in the folder you want (the current working folder/directory of MATLAB). Click Save to save the file.

   - **On UNIX workstations:** You are on your own to write and save the file using the editor of your choice. After writing the file, quit the editor to get back to MATLAB.

4. Figure 2.5 shows a sample session that executes the function `circlefn` in three different ways. Try it out.

```
>> R = 5;
>> [x,y] = circlefn(R);

>> [cx,cy] = circlefn(2.5);

>> circlefn(1);

>> circlefn(R^2/(R+5*sin(R)));
```

Specify the input and execute the function with an explicit output list.

You can also specify the value of the input directly.

If you don't need the output, you don't have to specify it.

Of course, the input can also be a valid MATLAB expression.

Figure 2.5: Lesson 5: Executing a function file.

Comments:

- Note that a function file (see previous page) must begin with a function definition line. To learn more about function files, refer to Section 4.2 on page 102.
- The argument of the `title` command in this function file is slightly complicated. It consists of two character strings. The first one is a simple character string, `'Circle of radius r ='`. The second one, `num2str(r)`, is a function that converts the numeric value of r to a string (and hence the name of the function). The square brackets create an array of the two strings by concatenating them. You will learn more about this in Section 3.3 on page 77.

EXERCISES

1. **On-line help:** Type `help function` to get on-line help on `function`. Read through the help file.

2. **Convert temperature:** Write a function that outputs a conversion table for Celsius and Fahrenheit temperatures. The input of the function should be two numbers: T_i and T_f, specifying the lower and upper range of the table in Celsius. The output should be a two-column matrix: the first column showing the temperature in Celsius from T_i to T_f in the increments of 1°C and the second column showing the corresponding temperatures in Fahrenheit. To do this, (i) create a column vector C from T_i to T_f with the command `C=[Ti:Tf]'`, (ii) calculate the corresponding numbers in Fahrenheit using the formula $[F = \frac{9}{5}C + 32]$, and (iii) make the final matrix with the command `temp=[C F];`. Note that your output variable will be named *temp*.

3. **Calculate factorials:** Write a function `factorial` to compute the factorial $n!$ for any integer n. The input should be the number n and the output should be $n!$. You might have to use a *for* loop or a *while* loop to do the calculation. See Section 4.3.4 on page 114 for a quick description of these loops. (You can use the built-in function `prod` to calculate factorials. For example, $n!$ is `prod(1:n)`. In this exercise, however, do not use this function.)

4. **Compute the cross product:** Write a function file `crossprod` to compute the cross product of two vectors $\mathbf{u}$ and $\mathbf{v}$, given $\mathbf{u} = (u_1, u_2, u_3)$, $\mathbf{v} = (v_1, v_2, v_3)$, and $\mathbf{u} \times \mathbf{v} = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$. Check your function by taking cross products of pairs of unit vectors: $(\mathbf{i}, \mathbf{j})$, $(\mathbf{j}, \mathbf{k})$, etc. $[\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$, $\mathbf{k} = (0, 0, 1)]$. (Do not use the built-in function `cross` here.)

5. **Sum a geometric series:** Write a function to compute the sum of a geometric series $1 + r + r^2 + r^3 + \cdots + r^n$ for a given r and n. Thus the input to the function must be r and n and the output must be the sum of the series. [See Exercise 4 of Lesson 2.]

6. **Calculate the interest on your money:** The interest you get at the end of n years, at a flat annual rate of r%, depends on how the interest is compounded. If the interest is added to your account k times a year, and the principal amount you invested is X_0, then at the end of n years you would have $X = X_0\left(1 + \frac{r}{k}\right)^{k\,n}$ amount of money in your account. Write a function to compute the interest $(X - X_0)$ on your account for a given X, n, r, and k. Use the function to find the difference between the interest paid on \$1000 at the rate of 6% a year at the end of five years if the interest is compounded (i) quarterly ($k = 4$) and (ii) daily ($k = 365$). For screen output, use `format bank` (see Section 1.6.3, page 9, for a description of various formats).

2.6 Lesson 6: Working with Arrays and Matrices

Goal: To become familiar with 2-D arrays, indexing, matrix manipulation and simple computations with arrays and matrices.

Time Estimates

Lesson: 20 minutes
Exercises: 30 minutes

What you are going to learn

- How to enter a matrix.
- How to access an element, a row, a column, or a submatrix of a matrix.
- How to multiply a matrix with a vector.
- How to distinguish between array operation and matrix operation.

Method: We will take a simple 3×3 matrix and use its row and columns indices to manipulate the matrix, extract a vector, compute inner and outer product of vectors, multiply the matrix with a vector, exponentiate the matrix, and exponentiating each element of the matrix using an *array operation.*

Comments: Arrays are the backbone of MATLAB computation. The more familiar you are with arrays, their manipulation and usage (Figs. 2.6 and 2.7), the better off you are in the world of MATLAB. You should look out for understanding the following two concepts and acquiring associated computational skills.

- A 2-D array is a list of numbers arranged in rows and columns. If you form an array by writing numbers in rows, all rows must have the same number of entries. Same is true for columns. An array with m rows and n columns is called an $m \times n$ array and it has a total of $m \cdot n$ entries. An element of the array is recognized by its location—its row number and column number. These row and column identifiers are called indices of the matrix. Thus $A(i, j)$ refers to a specific element of matrix A located in the ith row and jth column. These locations are unique. In MATLAB, you can also use a list of numbers for the row and column indices to access more than one element of the matrix at a time. This concept of using indices carries over to higher dimensional arrays (discussed in Section 4.4) as well.
- You can use arrays for carrying out element-by-element operations (also called *array operations*, see Section 3.2 for details) as well as matrix operations. If A is a square array, then $B = [A]*[A]$ or $B = A^2$ is very different from $B = [a_{ij}^2]$. The first one is the square of matrix A or the product of the matrix A with itself, whereas the second one is an array made of squares of corresponding elements of array A.

```
>> A=[1 2 3; 4 5 6; 7 8 8]

A =

     1     2     3
     4     5     6
     7     8     8

>> A(2,3)

ans =

     6

>> A(3,3) = 9

A =

     1     2     3
     4     5     6
     7     8     9

>> B = A(2:3,1:3)

B =

     4     5     6
     7     8     9

>> B = A(2:3,:)

B =

     4     5     6
     7     8     9

>> B(:,2)=[]

B =

     4     6
     7     9
```

Matrices are entered row-wise. Rows are separated by semicolons and columns are separated by spaces or commas.

Element A_{ij} of matrix A is accessed as `A(i,j)`.

Correcting any entry is easy through indexing.

Any submatrix of A is obtained by using range specifiers for row and column indices.

The colon by itself as a row or column index specifies all rows or columns of the matrix.

A row or a column of a matrix is deleted by setting it to a null vector `[]`.

Figure 2.6: Lesson 6: Examples of matrix input and matrix manipulation.

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> x = A(1,:)'
x =
     1
     2
     3
>> x'*x
ans =
    14
>> x*x'
ans =
     1     2     3
     2     4     6
     3     6     9
>> A*x
ans =
    14
    32
    50
>> A^2
ans =
    30    36    42
    66    81    96
   102   126   150
>> A.^2
ans =
     1     4     9
    16    25    36
    49    64    81
```

Matrices are transposed using the single right-quote character ('). Here x is the transpose of the first row of A.

Matrix or vector products are well-defined between compatible pairs. A row vector (x') times a column vector (x) of the same length gives the inner product, which is a scalar, but a column vector times a row vector of the same length gives the outer product, which is a matrix.

Look how easy it is to multiply a vector with a matrix, compared with Fortran or Pascal.

You can even exponentiate a matrix if it is a square matrix. A^2 is simply A*A.

When a dot precedes the arithmetic operators *, ^, and /, MATLAB performs array operation (element-by-element operation). So, A.^2 produces a matrix with elements $(a_{ij})^2$.

Figure 2.7: Lesson 6: Examples of matrix transpose, matrix multiplication, matrix exponentiation, and array exponentiation.

EXERCISES

1. **Entering matrices:** Enter the following three matrices.

$$A = \begin{bmatrix} 2 & 6 \\ 3 & 9 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad C = \begin{bmatrix} -5 & 5 \\ 5 & 3 \end{bmatrix}.$$

2. **Check some linear algebra rules:**
   - **Is matrix addition commutative?** Compute `A+B` and then `B+A`. Are the results the same?
   - **Is matrix addition associative?** Compute `(A+B)+C` and then `A+(B+C)` in the order prescribed. Are the results the same?
   - **Is multiplication with a scalar distributive?** Compute α`(A+B)` and α`A+`α`B`, taking $\alpha = 5$, and show that the results are the same.
   - **Is multiplication with a matrix distributive?** Compute `A*(B+C)` and compare with `A*B+A*C`.
   - **Matrices are different from scalars!** For scalars, $ab = ac$ implies that $b = c$ if $a \neq 0$. Is that true for matrices? Check by computing `A*B` and `A*C` for the matrices given in Exercise 1. Also, show that `A*B` $\neq$ `B*A`.
3. **Create matrices with `zeros`, `eye`, and `ones`:** Create the following matrices with the help of the matrix generation functions `zeros`, `eye`, and `ones`. See the on-line help on these functions, if required (e.g., `help eye`).

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad E = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}, \qquad F = \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}.$$

4. **Create a big matrix with submatrices:** The following matrix G is created by putting matrices A, B, and C, given previously, on its diagonal. In how many ways can you create this matrix using submatrices A, B, and C (that is, you are not allowed to enter the nonzero numbers explicitly)?

$$G = \begin{bmatrix} 2 & 6 & 0 & 0 & 0 & 0 \\ 3 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 3 \end{bmatrix}.$$

5. **Manipulate a matrix:** Do the following operations on matrix G created in Exercise 4.
   - Delete the last row and last column of the matrix.
   - Extract the first 4×4 submatrix from G.
   - Replace $G(5,5)$ with 4.
   - What do you get if you type `G(13)` and hit return? Can you explain how MATLAB got that answer?
   - What happens if you type `G(12,1)=1` and hit return?

2.7 Lesson 7: Working with Anonymous Functions

Goal: To learn how to define and use anonymous functions in command-line computations.

An anonymous function is a function of one or more variables that you create on the command line for subsequent evaluation (Figs 2.8 and 2.9). Such a function is especially useful if you need to evaluate the function several times (with different input) during a single MATLAB session and you do not care to code it in a function file and save for later use. For example, let us say that you have $f(x) = x^3 - 32x^2 + (x-22)x + 100$, and you need to evaluate this function at different values of x, plot it over certain range of x, or find the zeros of $f(x)$. For such computations, you can define $f(x)$ as an anonymous function and evaluate it at any value(s) of x or use this function as an input to other functions that need such input.

Time Estimates

Lesson: 25 minutes
Exercises: 40 minutes

What you are going to learn

- How to define an anonymous function of a single variable.
- How to evaluate an anonymous function with scalar or array arguments.
- How to define an anonymous function of several variables.
- How to use anonymous functions as input to other functions.

Method:

The key to anonymous functions is the syntax of its creation:

fn_name = @(*list of input variables*) *function_expression*

Here, the key element is the symbol @ that assigns a *function handle* to the defined function. A *function handle* is a name given to a function by which you can call it wherever you need it. In the anonymous function definition line, *fn_name* is the name of the function or the *handle* of the function. The syntax @(*list of input variables*) is what tells MATLAB that you are defining an anonymous function here.

Comments:

- Anonymous functions are defined on the command line. They live in the MATLAB workspace and are alive as long as the MATLAB session lasts.
- You can define an anonymous function with any number of input variables.
- You must use a vectorized expression (using array operators) for the function if you intend to use an array as an input variable.
- You can use anonymous functions as input to other functions where appropriate.

```
>> f = @(x) x^3-3*x^2 +x*log(x-1)+100

f =

    @(x)x^3-3*x^2+x*log(x-1)+100

>> f(0)

ans =

   100

>> f(1)

ans =

  -Inf

>> values = [f(0) f(1) f(2) f(10)]

values =

  100.0000       -Inf    96.0000   821.9722

>> x=[0 1 2 10];
>> f(x)
???? Error using ==> mpower
Matrix must be square.

Error in ==> @(x)x^3-3*x^2+x*log(x-1)+100

>> f = @(x) x.^3-3*x.^2 +x.*log(x-1)+100;
f(x)

ans =

  100.0000       -Inf    96.0000   821.9722

>> x = linspace(-10,10);
>> plot(x,f(x))

Warning: Imaginary parts of complex X and/or Y
arguments ignored
```

Create a function $f(x) = x^3 - 3x^2 + x\log(x-1) + 100$

Evaluate the function at $x = 0$, i.e., find $f(0)$.

Evaluate the function at $x = 1$. Note that f is singular at $x = 1$.

You can use f in an array also.

Using an array as the input to f causes an error. This is because the expression for f is not vectorized.

Redefine f by vectorizing the expression (use array operators). Now use it with an array argument.

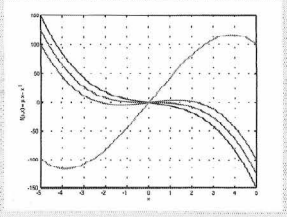
You can also use f as input to other functions where appropriate.

Figure 2.8: Lesson 7: Creating and evaluating anonymous functions in one variable.

```
>> f = @(mu,x) mu*x - x.^3;
>> x = linspace(-5,5)';
>> f_many = [f(-5,x) f(0,x) f(5,x) f(45,x)];
>> plot(x,f_many)
>> grid
>> xlabel('x')
>> ylabel('f(\mu,x)=\mu x - x^3')
```

Define a function of two variables $f(\mu,x) = \mu\, x - x^3$ and evaluate over a range of x for different values of μ.

Output



```
>> g = @(a,x) a(4)*x^3 + a(3)*x^2 + a(2)*x + a(1);
>> a = [1 2 -1 0];
>> g(a,5)
ans =
    -14

>> g([1 2 -1 0],5)
ans =
    -14
```

Define a function $g(a,x)$ of two variables and use a, one of the variables, as an array of parameters. Here, $a = [a_1\ a_2\ a_3\ a_4]$ and $g(x) = a_4x^3 + a_3x^2 + a_2x + a_1$. This is useful for all polynomial type of functions.

```
>> g = @(a,x) a(4)*x.^3 + a(3)*x.^2 + a(2)*x + a(1);
>> a = [-1 2 1 -1];
>> fplot(g(a),[-2 2])

??? Input argument "x" is undefined.
Error in ==> @(a,x)a(4)*x.^3+a(3)*x.^2+a(2)*x+a(1)
```

Be careful about passing functions of two variables as input variable to functions that expect a function of a single variable as input.

```
>> g = @(x) a(4)*x.^3 + a(3)*x.^2 + a(2)*x + a(1);
>> fplot(g,[-2 2])
>> g_integral = quad(g,0,1)

g_integral =
    0.0833
```

You can first define the array a in the workspace and then define the function g with single input x. Now you can use it as input to functions that require the function to be dependent on a single variable. Here we compute $\int_0^1 g(x)dx$ with `quad`.

Figure 2.9: Lesson 7: More explorations with anonymous functions.

EXERCISES

1. **Creation and evaluation of anonymous functions:** Create the function: $f(x) = x^2 - \sin(x) + \frac{1}{x}$.
   (a) Find $f(0), f(1)$, and $f(\pi/2)$.
   (b) Vectorize f and evaluate $f(x)$ where $x = [\ 0 \quad 1 \quad \pi/2 \quad \pi\ ]$.
   (c) Create `x = linspace(-1,1)`, evaluate $f(x)$, and plot x vs $f(x)$.
   (d) Combine the following three commands into a single command to produce the plot that you will get at the end of the third command.

```
x = linspace(-1, 1);    y = f(x);    plot(x,y);
```

   (e) Use `fplot` to graph $f(x)$ over x from $-\pi$ to π.

2. **Computation with multiple anonymous functions:** Create three anonymous functions corresponding to the following expressions:

$$\begin{aligned} f(x) &= x^4 - 8x^3 + 17x^2 - 4x - 20 \\ g(x) &= x^2 - 4x + 4 \\ h(x) &= x^2 - 4x - 5. \end{aligned}$$

   (a) Evaluate $f(x) - g(x)h(x)$ at $x = 3$.
   (b) Evaluate $f(x) - g(x)h(x)$ at $x = [1\ 2\ 3\ 4\ 5]$.
   (c) Evaluate $\frac{f(x)}{g(x)} - h(x)$ for any x.
   (d) Plot $f(x)$ and $\frac{f(x)}{g(x)h(x)}$ over $x \in [-5, 5]$.

3. **Anonymous functions as the input to other functions:** Use the same function $f(x)$ defined previously, i.e., $f(x) = x^4 - 8x^3 + 17x^2 - 4x - 20$ for the following tasks.
   (a) Plot the function using `fplot` over an appropriate interval of x and locate the zeros of the function on the graph (that is, find all x for which $f(x) = 0$).
   (b) Learn about the built-in function `fzero` that finds the zero of any given function by typing `help fzero`. Now use `fzero` to find the zeros of f accurately, making approximate initial guesses based on the plot you made above.
   (c) Use function `quad` to integrate $f(x)$ from $x = 0$ to $x = 1$ and verify the results by direct integration.

4. **Anonymous functions of several variables:** The formula for computing compound interest on an investment is given by

$$x = x_0 \left(1 + \frac{r}{100}\right)^n$$

   where x = accumulated amount, x_0 = initial investment, r = rate of annual interest in percentage, and n = number of years. Define an anonymous function to compute x with (x_0, r, n) as the input. Using this function, compare the growth of a \$1000 investment over a period of eight years earning an interest of 9% with that over a period of nine years earning an interest of 8%.