Beginning Matlab Exercises

R. J. Braun
Department of Mathematical Sciences
University of Delaware

1 Introduction

This collection of exercises is intended to help you start learning MATLAB. MATLAB is a huge package with many capabilities, but it is easy to use on many levels.

The text indicated by the numbers will be MATLAB commands that you can input to the MATLAB prompt. ⟨text⟩ will indicate quantities that you need to replace with in a MATLAB command. Text to the right of a % symbol are comments and need not be typed at the prompt unless you wish to have the comment in a diary or other file.

In nearly all cases, no output will be shown; that's for you to find!

2 Getting Started

In the computer classroom, you can start MATLAB by opening a terminal window and then typing the appropriate command. Typing the command `matlab` followed by `<enter>` will start MATLAB using its window manager. In this mode, you can find helpful pull down menus across the top of the window (or desktop). The default configuration has one main command window, and two other windows that allow access to the workspace, command history, current directory and launch pad.

Typing the command `matlab -nojvm` runs MATLAB without the window manager. This can avoid some annoying bugs in the window manager that may appear (particularly in linux) on starting up. This will give only the command window; plots will appear in separate graphics windows in both modes.

To exit, use the pull down menu under File or type the command `exit` in the command window.

3 Scalar arithmetic

MATLAB can be used as a calculator. Some of the problems indicate the precedence of arithmetic operations.

1. `3*2^4`
2. `(3*2)^4` `% parentheses have highest priority`
3. `3-2^4`
4. `3^4-3`
5. `8/2^4`
6. `2^4\8` `% same as previous! two different divisions, \ and /`
7. `8^4/2`

Precedence of arithmetic operations is: (1) parentheses, (2) exponentiation, (3) multiplication and division, and (4) addition and subtraction. Calculations proceed from left to right on the line and equal precedence is settled in this way.

4 Vectors

It will typically be that our basic unit for computing will be a vector. We need to be able to create and manipulate them.

4.1 Creating vectors

1. `x = [3 4 7 11]   % create a row vector (spaces)`
2. `x = 3:8             % colon generates list; default stride 1`
3. `x = 8:-1:0          % ⟨start⟩ : ⟨stride⟩ : ⟨stop⟩ specifies list`
4. `xx = [ 8 7 6 5 4 3 2 1 0];     %  same as last; semicolon suppresses output`
5. `xx                    % display contents`
6. `x = linspace(0,1,11)  % generate vector automatically`
7. `x = 0:0.1:1           % same thing`
8. `y = linspace(0,1);    % note semicolon!`
9. `length(x)`
10. `length(y)`
11. `size(x)`
12. `size(y)`
13. `y(3)                  % access single element`
14. `y(1:12)               % access first twelve elements`
15. `y([3 6 9 12])         % access values specified in a vector!`
16. `x'                    % transpose`
17. `z = [ 1+2*i 4-3*i ]`
18. `z'`
19. `z.'                   % note difference in transposes!`
20. `3*[1 2 5]             % factor replicated, multiplies each element`

Matlab labels the arrays (vectors and matrices) beginning with 1; this will be an important programming detail on more than one occasion.

4.2 Help with vectors and other things

MATLAB has extensive online help; try these commands to get help at the command line. If the window manager is running in MATLAB, you can also get browser-based interactive help. Try both.

1. `help help           %  try Matlab's extensive help`
2. `help length`
3. `help size`
4. `help linspace`
5. `help logspace`
6. `help clc`
7. `help clear`
8. `help who`

9. `help whos`

Note that MATLAB help is also available in html format by using the menu bar at the top of the desktop. The help is more comprehensive in the browser-based help, but it is sometimes harder to find if you already know the command name.

5 Functions and plotting

5.1 Elementary functions in Matlab

1. help elfun
2. Use the following few commands (a *script*) to make a plot. The evaluation of $v = \cos(u)$ in Matlab creates a vector whose elements are $v(k) = \cos(u(k))$ where $k = 1, 2, ...n$.

```
n = 11;
u = linspace(0,2*pi,n);
v = cos(u);   % all function evaluations done at once!
plot(u,v)
```

3. Once you have typed these to the MATLAB prompt, you can recall and edit them one at a time, or you can save time by not having to retype them in the following way. You can save these commands in a file and execute the file. Click on the "blank sheet" icon at the upper left in the MATLAB desktop; using the MATLAB editor, you can type the commands or copy them to the editor and then save it to ⟨filename⟩.m, where filename is the name you supply. (Avoid reserved words in MATLAB, e.g., "plot;" see below for more info.) You can execute this newly created file by typing the filename without extension (that is, without the .m) to the MATLAB prompt. This collections of commands is essentially a small MATLAB program, and this kind of file is called a *script file*.
4. The last plot had a crude "curve" representing the cosine function; repeat the last example with n=101 to improve the curve.
5. Add these commands to clean up the plot by adding axis labels and a title. The text inside the single quotes is a *string* which we intend to be the labels, etc.

```
xlabel('u'); ylabel('v'); title('v = cos(u)');
```

Further improvements in plotting appearance will be discussed later; the command `help plot` will be very helpful.

5.2 Reserved words in Matlab

The following words are already used in MATLAB and they must be avoided: `for, end, if, while, function, return, elseif, case, otherwise, switch, continue, else, try, catch, global, persistent, break`. (You can use similar words by capitalizing one or more letters if your system is case sensitive.) the following variable names are special in MATLAB and you should avoid changing them unless you are very careful about what you are doing: `ans, beep, pi, eps, inf, NaN, nan, i, j, nargin, nargout, realmin, realmax, bitmax, varargin, varargout`.

Try the following to see the distinction between the two cases just described.

1. `end = 1   % you should get an error`
2. `End = 1   % could get error, depending on system`
3. `pi = 3    % forgetting thousands of years of progress!`
4. `pi`
5. `clear pi`
6. `pi       % back to the present!`

6 Diary files

We can record what was typed and the text-based output during a MATLAB session in a file called a *diary file*. This recording of a session will be useful for homework and projects. It is not a program and the MATLAB commands can't be run in the desktop like a script file can.

1. `diary ThisClassIsGreat   % puts diary in file called ThisClassIsGreat`
2. `g*h`
3. `e*f`
4. `diary off                % diary file closed`
5. `diary ThisClassIsGreat   % reopening it will append additional work`
6. `f*e`
7. `diary                    % diary toggles with no additional specification`

7 Output

We will have the need to produce legible output; let's see how to do that. First, we will do some very basic output commands; then, we need to see how to repeat commands in MATLAB. Finally, we will do a little nicer output.

7.1 Basic output

1. `disp('The disp command writes text.')   % inside quotes: verbatim`
2. `t = logspace(-3,6,10);`
3. `x = 0.1234567890123456*t;`
4. `format short`
5. `t`
6. `x`
7. `format long`
8. `x`
9. `format long e`
10. `x`
11. `format short e`
12. `x`
13. `format short g`
14. `x`
15. `format long g`
16. `x`
17. `disp(sprintf('The 10th element of x is x(10) = %6.3f',x(10)))`

18. `disp(sprintf('The 10th element of x is x(10) = %10.3f',x(10)))`
19. `disp(sprintf('The 10th element of x is x(10) = %12.3f',x(10)))`
20. `disp(sprintf('The 10th element of x is x(10) = %8.3e',x(10)))`
21. `disp(sprintf('The 10th element of x is x(10) = %8.3e',x(10)))`
22. `disp(sprintf('The 10th element of x is x(10) = %8.12e',x(10)))`
23. `disp(sprintf('The 10th element of x is x(10) = %24.15e',x(10)))`
24. `disp(sprintf('The 10th element of x is x(10) = %1.4g',x(10)))`
25. `disp(sprintf('The 10th element of x is x(10) = %16.12g',x(10)))`
26. `disp(sprintf('The 10th element of x is x(10) = %24.15g',x(10)))`

7.2 Repeating Commands I

The `for` loop repeats the commands that are inside of it. The basic structure is

```
for <list-of-values>
  <statements-to-be-done>
end
```

Use this structure to repeat output, or do calculations

1. % for loop now

   ```
   for k = 1:5
     disp(sprintf('%8.5e',x(k)))
   end
   ```

2. % for loop doing a calculation

   ```
   total = 0
   for k = 1:10
     total = total + x(k);
     disp(sprintf('Total for k=%2.0f is %15.8g',k,sum))
   end
   ```

3. `sum(x)  % built in matlab command sums components of vector`

Here the command `sum` can replace the loop; it adds up the elements of a vector.

7.3 Creating a table

1. Try this script to make a neat table.

   ```
   disp(' ')    %  a blank line
   disp(' k        t           x       ')
   disp('----------------------------')
   for k=1:10
     disp(sprintf('%2.0f    %7.3g    %7.7e',k,t(k),x(k)))
   end
   ```

8 More about working with vectors and matrices

We need more tools to efficiently manipulate vectors and matrices in MATLAB.

8.1 Vector and matrix arithmetic

We'll now explore making matrices and doing arithmetic with them. There are some new arithmetic operations that you did not see before in linear algebra, but they will be useful nonetheless.

1. `g = [1; 2; 3; 4]`
2. `g = [1 2 3 4; 5 6 7 8]`
3. `g - 2`
4. `2*g-1`
5. `g = [1 2 3 4; 5 6 7 8; 9 10 11 12]`
6. `h = [1 1 1 1; 2 2 2 2; 3 3 3 3]`
7. `g-h`
8. `g*h % What happened?`
9. `h*g`
10. `g.*h % NOT the usual matrix multiplication! Elementwise!`
11. `g./h % elementwise division; denominator below slash`
12. `h.\g % Last two same!`
13. `g*h'`
14. `g'*h`
15. `e = [ 1 2 3; 4 5 6; 7 8 0]`
16. `f = [9 8 7; 6 5 4; 3 2 1]`
17. `e*f`
18. `f*e % Not same!`
19. `q = rand(3) % matrix with elements uniformly on [0,1]`
20. `A^2 % raise matrix to a power`
21. `A^2 - A*A % verify with definition`
22. `A.^2 % square each element`
23. `A.^2 - A*A % not the same!`

8.2 Standard arrays

Some matrices occur so often that MATLAB has standard ways to produce them; this makes programming and use easier.

1. `ones(3) % All ones`
2. `zeros(2,5) % All zeros`
3. `size(zeros(2,5))`
4. `size(g)`
5. `ones(size(g))`

6. `eye(4)          % Identity matrix`
7. `eye(2,4)`
8. `eye(4,2)`
9. `rand(3)         % Random elements uniformly distributed between 0 and 1`
10. `b = eye(3)`
11. `rand(size(b))`
12. `a = 1:4`
13. `diag(a)`
14. `diag(a,1)`
15. `diag(a,-2)`

8.3 More about creating arrays

1. `d = exp(1)`
2. `d*ones(3,4)         % slowest`
3. `d+zeros(3,4)        % slow`
4. `d(ones(3,4))        % faster`
5. `repmat(d,3,4)       % fastest`

8.4 Selecting and changing parts of matrices

1. `g = [1 2 3 4; 5 6 7 8; 9 10 11 12]`
2. `g(3,2)                % single element`
3. `g(:,2)                % all rows, second column`
4. `g(2,:)                % second row, all columns`
5. `g(1:2,:)`
6. `g(2,:) = [1 1 1 1]    % replace row`
7. `g(:,2) = [2; 2; 2]    % replace column`

9 A start at matrix algebra

In linear algebra, there was a need to solve systems, find determinants and inverses, and to construct special matrices; we now scratch the surface of these capabilities in MATLAB.

1. `A = rand(5)     % matrix of coefficients from Ax=b`
2. `b = rand(5,1)   % right hand side`
3. `det(A)          % determinant of A`
4. `inv(A)          % inverse of A`
5. `x = inv(A)*b    % solves system, NOT recommended!`

6. `size(x), size(b), size(A)`
7. `x2 = A\b        % better way to solve systems`
8. `spy(A)          % sparsity plot: show nonzero elements`
9. `D = diag(1:5)`
10. `spy(D)`
11. `L = tril(A)`
12. `U = triu(A)`
13. `size(U), spy(U)`

10 More vectors and plotting

10.1 Vectorizing function evaluations

We try a different rational function than in your textbook. The ideas behind rational functions approximations are discussed in *Numerical Analysis* by Burden and Faires (Section 8.4), which is on reserve for this course. (Maple can compute the rational function approximation with the `ratpoly` command.) Consider the rational function

$$y = \frac{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3}{1 + \frac{2}{5}x + \frac{1}{20}x^2}, \tag{1}$$

which approximates e^{-x} on the interval $x \in [0, 1]$. We want to implement this function for plotting and to compare with the exponential function.

1. We could plot the function as follows; I recommend opening the editor in matlab and typing this into the editor and saving the file. The resulting script file can be run in MATLAB by typing the filename without the extension .m to the command window prompt. The first approach is:

   ```
   n = 150;
   x = linspace(0,1,n);
   y = zeros(1,n);
   for k = 1:n
     y(k) = (1-(3/5)*x(k)+(3/20)*x(k)^2 -(x(k)/60)*x(k)^2)/...
            (1+(2/5)*x(k)+(1/20)*x(k)^2)
   end
   plot(x,y)
   xlabel('x'), ylabel('y');
   title('Rational function approximation to e^{-x}')
   ```

   This approach makes no use of vectorization; the `for` loop computes each element one at a time. Note that x^2 is computed several times.

2. Now make use of vectorization and try to minimize the amount of calculation of powers of x; you may type this in by modifying your previous script file, or download this script from the course web page. In this case, we

   ```
   % Script file: RatFunPlot
   % Compare rational function approximation to exp(-x) on [0,1]
   % f(x) = (1-3x/3-3x^2/20-x^3/60)/(1+2x/5+x^2/20)
   %
   n = 150;
   ```

```
x = linspace(0,1,n);
xsqd = x.^2;
num = 1-(3/5)*x+(3/20)*xsqd -(x/60).*xsqd
den = 1+(2/5)*x+(1/20)*xsqd
y = num./den;
plot(x,y,x,exp(-x))
xlabel('x');
title('Rational function approximation to e^{-x}')
legend('Rational Function','e^{-x}',0)
```

In this case, the calculation of x^2 over all entries is computed in a single line; this is a vectorized calculation. The numerator and denominator are computed similarly, using the result of the squaring operation `xsqd`. The quotient is also a vectorized calculation.

The legend is located by the number of the corner of the plot when counting starting from the upper right and proceeding counterclockwise (for values 1 through 4); a value of 0 attempts to find the least conflict with data and -1 locates the legend to the right of the plot. More details about the legend can be found in section 1.7.5 of Van Loan or in MATLAB help.

10.2 More about plotting

1. We can control the number of figures and add to figures that are open. Try this script:

```
x = linspace{0,1};
y = 1 - x + x.^2/2 - x.^3/6;
exact = exp(-x);
figure            % create new figure window for plotting y
plot(x,y)
figure            % create new figure window
plot(x,exact)
hold on           % causes subsequent curves to be added to current figure
plot(x,y,'r-.')
hold off          % returns to normal plotting mode
```

   We can use `hold on` and `hold off` to control when to add curves to figures.

2. `close all` `% will close all graphics windows`

11 Conditional Execution and Repeating Commands II

11.1 Values and Variables in Matlab

Values that variables may take in MATLAB include floating point (double precision, e.g.), character and logical. Floating point numbers are the standard numerical values in MATLAB; character values are messages and labels containing letters and numbers. The logical values are True and False, which are indicated in MATLAB with a 1 and a 0, respectively. Variables with these values are not arithmetic and are used differently that numerical array to access parts of arrays; more about this in class.

11.2 Relational operators

Relational operators carry out comparisons or tests and the result is to return logical values True (1) or False (0). The relational operators that are available (and their MATLAB form) are: less than (<), less than or equal to (<=), greater than (<), greater than or equal to (>=), equal to (==), and not equal to ($\sim$=). Note that == is a test between two quantities, while = assigns a value to a variable. We can use these to compare scalars and arrays; try out the following.

1. `a = 4; b = -1; c = 1;`
2. `a > b         % note that the output is a logical value`
3. `a < b`
4. `a >= b`
5. `mytest = (abs(b) == c)   % parentheses optional but clearer`

Things get a little more complicated with arrays. Using the `g` and `h` from above, try out the following.

1. `g = [1 2 3 4; 5 6 7 8; 9 10 11 12]`
2. `h = [3 3 4 4; 5 5 6 6; 7 7 8 8]`
3. `h >= g`
4. `g == h          % note that the output is a logical value`
5. `bigger = (g >= h)`
6. `g(bigger)`
7. `g([0 0 0 1; 1 1 1 1; 1 1 1 1])   % array argument not logical!`

We will use some of this kind of thing programming later, but I don't want to elaborate now.

11.3 Logical operators

We can string relational operators together and manipulate logical variables using logical operators. They are (MATLAB in parentheses): logical AND (&), logical OR (|) and logical NOT (∼). The NOT operator just changes the value of the logical variable to the other value. A truth table shows what the AND and OR operators do; it will be given in class. The upshot is that the both parts from an AND operator must be True to return a True; only one of the inputs for the OR operator need be True for it to return True. Continuing with the last set of MATLAB commands, try these out.

1. `h >= g`
2. `p = ~(h >= g)`
3. `~p`
4. `(g == h) & (g > h)`
5. `(g == h) | (g > h)`
6. `g >= h`

11.4 The `while` loop

Sometimes, we want to repeat commands until a condition is satisfied. If we don't know how many repetitions are required in advance, a `while` loop is preferable over a `for` loop. The general structure is

```
while <expression>
    <commands>
end
```

The ⟨`commands`⟩ inside the while loop will be repeated as long as the ⟨`expression`⟩ returns a value of True, provided that ⟨`expression`⟩ returns a scalar. If the ⟨`expression`⟩ evaluates to an array, then *all* of the values that result from it must be True for ⟨`commands`⟩ to be evaluated. An example follows; it could be typed into a script file or at the command prompt. In the example, we halve a number until it is so small that the computer can't detect it when added to 1; the last number before this occurs is called *machine epsilon* and we'll talk about it more in class.

```
num = 0; myeps = 1;
while (1+myeps) > 1
   myeps = myeps/2;
   num = num + 1;
end
num
myeps = 2*myeps
eps                % matlab's preset variable for machine epsilon
```

The result is that `myeps` = `eps` = 2^{-52}.

11.5 The `if-else-end` construct

Sometimes we want to compute one set of commands or another depending on the result of a relational test. There are several forms that this sort of construct can take.

11.5.1 Simplest

The simplest is:

```
if <expression>
   <commands evaluated if True>
end
```

An example follows; given a positive number of pens, a cost is computed and displayed.

```
if (pens >= 0)
     cost = pens*1.99;
     disp(sprintf('Cost of %3.0f pens is $ %5.2f',pens,cost))
end
```

11.5.2 Intermediate

The next most complicated is:

```
if <expression>
      <commands evaluated if True>
   else
      <commands evaluated if False>
end
```

The example now prints a warning for negative numbers of pens.

```
if (pens >= 0)
      cost = pens*1.99;
      disp(sprintf('Cost of %3.0f pens is $ %5.2f',pens,cost))
   else
      disp('An invalid number of pens was specified.')
end
```

11.5.3 Most general

The most general form is:

```
if <expression1>
      <commands evaluated if expression1 is True>
   elseif <expression2>
      <commands evaluated if expression2 is True>
```

```
   elseif <expression3>
      <commands evaluated if expression3 is True>
   elseif <expression4>
      <commands evaluated if expression4 is True>
   else
      <commands evaluated if all other expressions are False>
end
```

We aren't limited to 4 expressions here; we could add more. We could use this to price pens according to quantity, as follows.

```
sale = false;
if (pens >= 0) & (pens < 20)
     cost = pens;
     sale = true;
   elseif (pens >= 20) & (pens < 40)
     cost = pens*0.95;
     sale = true;
   elseif (pens >= 40) & (pens <= 100)
     cost = pens*0.90;
     sale = true;
   else
     disp('not a valid number of pens')
end
if sale
     disp(sprintf('Cost of %3.0f pens is $ %5.2f',pens,cost))
end
```

Note that leaving off the last part (`else` and the related commands following it) will result in no action being taken if none of the previous expressions result in True values.

References

[1] D. Hanselman and B. Littlefield, *Mastering Matlab 6*, (Prentice Hall, Upper Saddle River, 2001).

[2] D. Hanselman and B. Littlefield, *Mastering Matlab 7*, (Prentice Hall, Upper Saddle River, 2004).

[3] D.J. Higham and N.J. Higham, *Matlab Guide*, (SIAM, Philadelphia, 2005, 2nd edn.). (Matlab 6 version available.)

[4] W.J. Palm III, *Introduction to Matlab 7 for Engineers*, (McGraw-Hill, New York, 2005). (Matlab 6 version available.)

[5] R. Pratap, *Getting Started with Matlab 7*, (University Press, Oxford, 2005).

[6] A. Gilat, *Matlab: An Introduction with Applications*, (Wiley, New York, 2003). This book is based on Matlab 6, but is still a friendly introduction.

[7] C. Van Loan, *Introduction to Scientific Computing*, (Prentice Hall, Upper Saddle River, 2000). The opening two chapters are a great introduction to MATLAB.