Appendix D

Introduction to MATLAB Programming

Contents

The following is a brief introduction to using MATLAB. This is not intended to provide a listing of various MATLAB commands, but illustrate how simple MATLAB programs are written, and the output displayed and printed. The goal is to get you started, and to also provide enough information that you can use the very good help facility that comes with MATLAB (most people who use MATLAB make frequent use of the help pages). It is also assumed that you have a basic, but perhaps imperfect, understanding of coding. This means that you are aware that there are such things as for-loops and if/then/else type statements, although unclear of their exact formatting. Finally, for many of the commands or actions that are considered, there are likely other ways to accomplish the same thing. Some of them possibly simpler, or maybe more clever, than what is shown here.

Caveat: The examples are worked out using a Mac implementation of MATLAB. It's assumed similar responses are obtained using other implementations.

D.1 Getting Started

The tool bar and window arrangement when you start MATLAB are shown in Figure D.1. The worksheet area, what is called the *Command Window*, is seen and it contains the ≫ prompt. Commands can be typed directly into the worksheet, including comment lines. Of more interest here are when the commands are contained in an m-file, and the file run as a program. How this is done is explained shortly, but it is first necessary to tell MATLAB where the files are located. As you can see in Figure D.1, right above the *Command Window*, the current folder is identified. In this case it is the Desktop folder. Just to the left of the *Command Window* the contents of the current folder are listed. The m-files needed for this Appendix are contained in the folder Appendix4/m files, and this is selected to run the various examples used here.

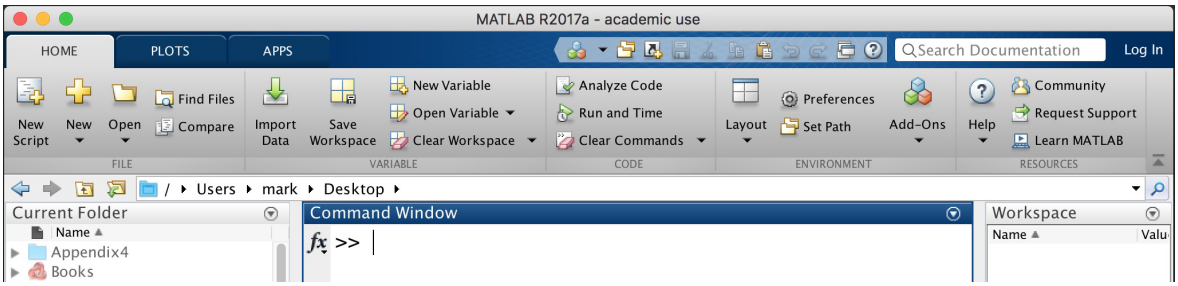


Figure D.1: Typical (default) MATLAB window.

After changing the current folder to Appendix4/m files, there is a list of the m-files on the left (similar to the list shown in Figure D.1). The first example to be discussed is in the m-file `ss.m`. Clicking on this, the result is shown in Figure D.2 (only the fist line of `ss.m` is shown). As you see, when the m-file is opened, its contents are shown in the *Editor* window. The *Command Window* is below this window (and is not shown in Figure D.2). Note the little red box, which contains an inverted triangle. This triangle lets you separate a window from the others (this is useful depending on the size of your computer screen).

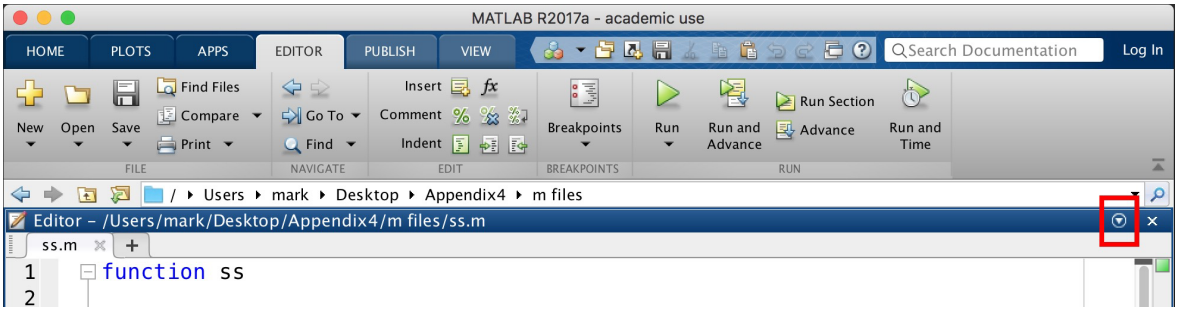


Figure D.2: The Editor window, and the first line of the m-file `ss.m`.

D.2 Basic m-file

To begin, in the textbook, in Example 1 in Section 1.1, the sum

$$S(n) = \sum_{j=1}^{n} \frac{1}{j} \tag{D.1}$$

was considered. A MATLAB m-file named `ss.m` was created to compute this for $n = 10, 10^2, 10^3$, and then print the results. The file is shown in Figure D.3. Note that these are simple text files that are named so they have a `.m` ending. As such, they can be created and then edited using a text editor (as illustrated in the top of Figure D.3), or created and edited using MATLAB (as illustrated in the bottom of Figure D.3). To create it in MATLAB, you first click on the *New Script* icon in the tool bar (far left in Figure D.1). This will open a blank work area in the *Editor*, and you then simply type in the commands as they appear in Figure D.3. Once finished, you save it by either clicking on the *Save* icon (see Figure D.2), or use the usual keystrokes for saving a file. Note that the file will be saved in the current folder (see Figure D.1).

ss.m

```
function ss

for ic=1:3
    n=10^ic;

    % sum series
    S=0;
    for j=1:n
        S=S+1/j;
    end

    fprintf('\n n = %i  S(n) = %8.3e \n',n,S)

end
```

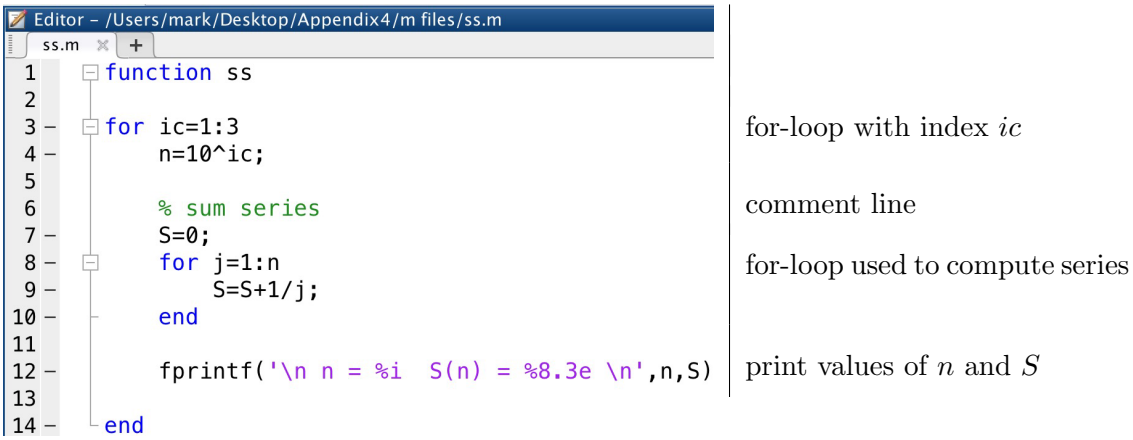
Editor – /Users/mark/Desktop/Appendix4/m files/ss.m

ss.m

```
1   function ss
2
3   for ic=1:3
4       n=10^ic;
5
6       % sum series
7       S=0;
8       for j=1:n
9           S=S+1/j;
10      end
11
12      fprintf('\n n = %i  S(n) = %8.3e \n',n,S)
13
14  end
```

Figure D.3: The m-file `ss.m` used to compute $S(n)$ in (D.1). Top: file when opened using a text editor. Bottom: file when opened using MATLAB.

A synopsis of the commands in `ss.m` is given below.



A couple of additional comments related to the commands used.

***for-loops*:** The for-loop starting on line 3 uses the values $1, 2, 3$ (in that order). If you wanted the values to increase by, say, 4 each time, then $ic = 1{:}3$ is replaced with $ic = 1{:}4{:}13$. With this, the values for ic are $1, 5, 9, 13$. Similarly, suppose you wanted to use, in order, $7, 5, 3, 1$. In this case $ic = 1{:}3$ is replaced with $ic = 7{:}{-}2{:}1$. Note that there are also while-loops, which are very useful, and an example is given in Section D.5.

***fprintf*:** This command contains the formatting to be used, with n written in integer format and S in exponential format. By default, the output appears in the *Command Window*. To find out about the format options, consult the help page for *fprintf*.

To run the commands in `ss.m`, with the file open in the *Editor* window, click on the green *Run* icon (see Figure D.2). The output appears in the *Command Window*, and what is computed is shown in Figure D.4. The green comment line was typed directly into the *Command Window* after the program was run (this is useful for annotating the output when it is to be submitted as homework).

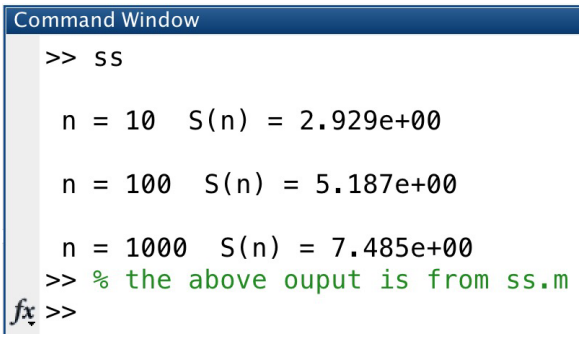


Figure D.4: Output from the `ss.m` file, along with a comment added afterward.

D.2.1 Printing

What is considered next are ways to print MATLAB files as well as the output. The easiest way is to use the Publish command, but there are other possibilities for specific tasks and these are described as well.

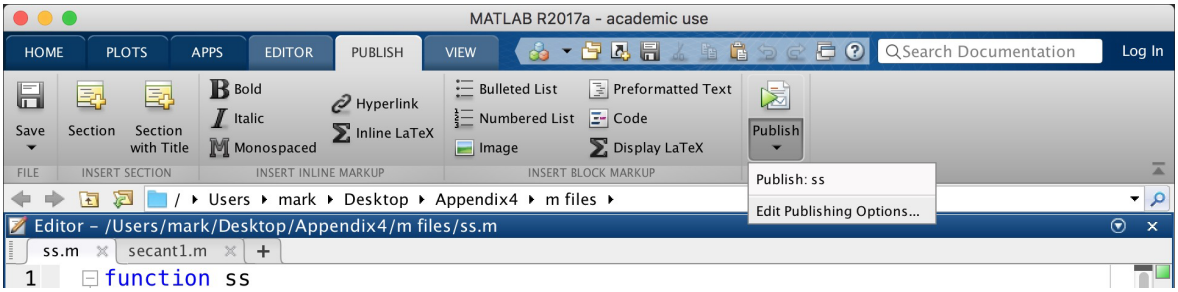


Figure D.5: Publish tab used to print MATLAB files.

Publish Command

With the m-file open, the Publish tab is selected (see Figure D.5). You simply click on the green triangle on the right to run the Publish command. What you get depends on what you have selected for the Publishing Options. Using the default settings, you will get a web-page (html file) which includes a copy of the m-file as well as the output, and this is easily printed. However, there are other possible outcomes, and some of the more useful are listed below.

Only *m-file*: Change option "Evaluate code:true" to "Evaluate code:false".

Only output: Change option "Include code:true" to "Include code:false".

PDF file: Change "Output file format:html" to "Output file format:pdf".

WORTH MENTIONING: There is an auto-formatting command that makes reading an m-file much easier. This is done in the *Editor* window by selecting the text of the file and then clicking on the green page icon just to the right of the word Indent on the tool bar (see Figure D.2).

LIMITATION: The Publish command prints the file that is currently open, and will not print other files that might be used by the code. If you want them printed then they must be done separately or else included in the open file as local functions (these are explained in Section D.5).

Non-Publish Methods

***m-file*:** The contents of an m-file can be printed using LaTeX. There is a LaTeX option for Publish, but a better approach is to use the *matlab-prettifier* package. An example is given in the figure below, which contains the LaTeX commands to print `ss.m`. Note that when the document is created, the TeX previewer reads the m-file, which is located by pathname (line 4). Moreover, it's possible to include text and mathematical expressions in the document, which is useful for producing homework write-ups and other technical documents.

```
1 \documentclass[11pt]{article}
2 \usepackage[numbered,framed]{matlab-prettifier}
3 \begin{document}
4 \lstinputlisting[style=Matlab-editor]{"m files/ss.m"}
5 \end{document}
```

Finally, it is possible to print an m-file by simply using the keystrokes usually used for printing a file (on a Mac this is Command-P). This is a few milliseconds faster than using the Publish method, but what you get in this case is a full page, large format, printout of the file.

***Output*:** As illustrated in Figures D.3 and D.4, the *fprintf* command can be used to generate formatted output in the *Command Window*. It's possible to write it to a text file instead (which are easily printable as well as editable). Assuming the name of the text file is *data.txt*, then the commands shown in the figure below will work. Specifically, the file is created (line 3), written into (line 13), and then closed (line 16).

Editor – /Users/mark/Desktop/Appendix4/m files/ssO.m

ss.m | ssO.m

```
1  function ssO
2
3  fileID = fopen('data.txt','w');
4  for ic=1:3
5      n=10^ic;
6
7      % sum series
8      S=0;
9      for j=1:n
10         S=S+1/j;
11     end
12
13     fprintf(fileID,'\n n = %i  S(n) = %8.3e \n',n,S);
14
15 end
16 fclose(fileID);
```

D.3 Basic Plotting

A basic m-file, named `p.m`, that plots $y = cos(x)$ for $0 \leq x \leq 2\pi$ is shown in the top row in Figure D.6. To compute the curve, 100 points are used along the x-axis. The fact is that the resulting plot is rather poor, and unacceptable for publication. For example, the text is almost unreadable, and the curve is very light. This is easily fixed, and the modified file, named `pp.m`, is shown in second row of Figure D.6. What was done was to change the line width for the curve (line 14), and to change the font properties used for labeling (line 20). There is considerable flexibility to modify the plot parameters, and the help pages for the *plot* command should be consulted. Also, in terms of the font size used in a plot, it is worth knowing that in publishing it is generally expected that the font size in the plot approximately matches the size used in the text.

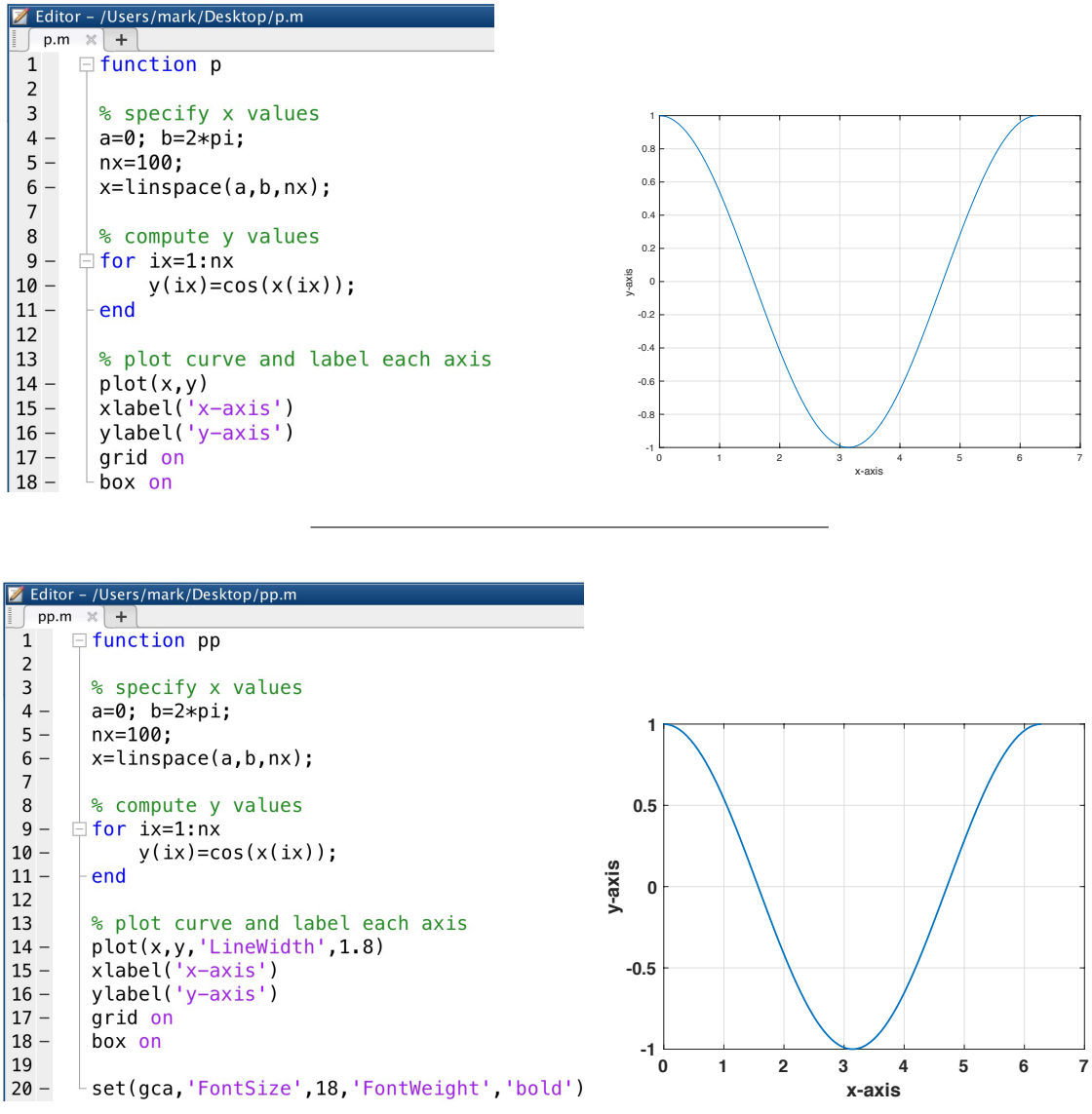


Figure D.6: The commands on the left produce the plot on the right.

D.3.1 Legends

One additional situation worth illustrating concerns plots with multiple curves, and the associated legend required in the plot. An example of this is given in Figure D.7, where $y = \cos(x)$ and $Y = e^{-x/10} + x^{1/5}$ are plotted for $0 \leq x \leq 2\pi$. The file in this case is named `ppp.m`. What has been required is the *hold* command (line 16), the new *plot* command (line 17), and the *legend* command (line 22). For readability, the font properties used in the legend have been modified (line 25).

Editor - /Users/mark/Desktop/ppp.m

ppp.m

```
1   function ppp
2
3   % specify x values
4 - a=0; b=2*pi;
5 - nx=100;
6 - x=linspace(a,b,nx);
7
8   % compute y and Y values
9 - for ix=1:nx
10 -     y(ix)=cos(x(ix));
11 -     Y(ix)=exp(-x(ix)/10)-x(ix)^(1/5);
12 - end
13
14  % plot curve and label each axis
15 - plot(x,y,'LineWidth',1.8)
16 - hold on
17 - plot(x,Y,'--','LineWidth',1.8)
18 - xlabel('x-axis')
19 - ylabel('y-axis')
20 - grid on
21 - box on
22 - legend({' cos(x)',' e^{-x/10} - x^{1/5}'},'Location','north')
23
24 - set(gca,'FontSize',18,'FontWeight','bold')
25 - set(findobj(gcf,'tag','legend'),'FontSize',16)
```

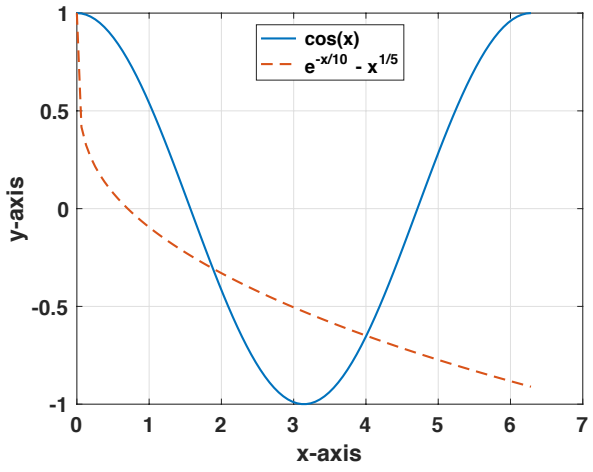


Figure D.7: The commands on the top produce the plot on the bottom.

D.3.2 Saving and Printing Plots

It is recommended that before printing that you first save the plot. On the plot toolbar, under *File*, there is a *Save as* option. After selecting this, you will be given the opportunity to select the format, as well as the name and location of the plot file. It is strongly recommended that you pick the EPS (*.eps) format. The reason is that it uses vector graphics, which means the image is scalable and capable of producing high resolution plots no matter what size you need. As an illustration of this, the plot in Figure D.6 was saved using four different formats: eps, png, tiff, and jpg. Each saved image was then opened, magnified, and the term $x^{1/5}$ in the legend was isolated and printed. The results are shown in Figure D.8. As expected the eps format is (clearly) superior to the others. Note that the Portable Document Format (*.pdf) format, also available, has the same high resolution capability as eps. The drawback to this is that MATLAB uses an entire page for the plot, which limits embedding it into a document or text file (as was done in Figures D.6 and Figures D.7).

As a tip, before saving the plot you might consider what relative dimensions you want (wider versus taller, etc). You can resize the plot window with the mouse, and the resized version is the one that will be saved.

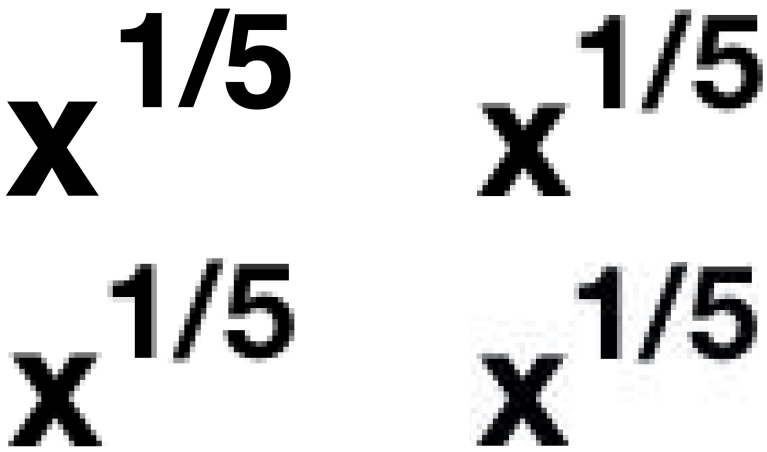
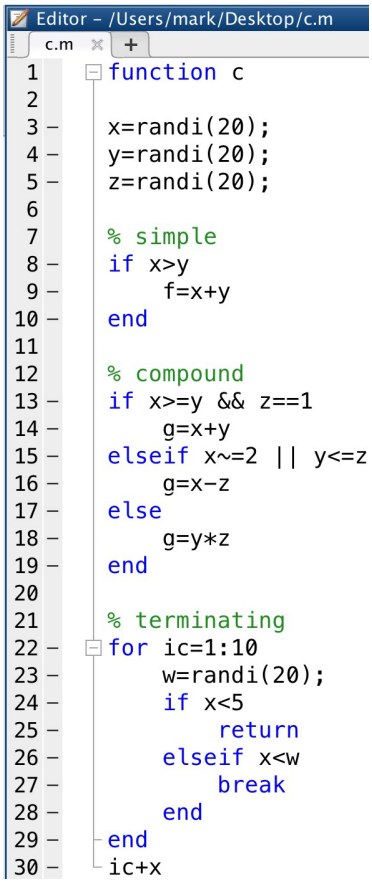


Figure D.8: The $x^{1/5}$ term from Figure D.6 using eps (upper left), png (upper right), tiff (lower left), and jpg (lower right) format.

D.4 If/Else/Break/Return

Conditionals are often needed, and can involve a wide variety of conditions or outcomes. This requires knowing how relational and logical operators are written. Examples of some of the more often used possibilities are illustrated in the figure below. For more information about these, look at *relational operations* and *logical operations* in MATLAB's help pages.



Generate some random integers

Compute f if $x > y$

Compute g if $x \geq y$ and $z = 1$

Otherwise, compute g if $x \neq 2$ or $y \leq z$

Otherwise, use this formula for g

If $x < 5$ stop computing and leave function

Otherwise, if $x < w$ then leave the for-loop and go to next command (line 30)

A few additional comments are in order. First, in the compound version (as in lines 13-19) it is possible to have multiple *elseif* conditions, and to not have an *else* condition. It is also possible to have more than two tests for an *if* or *elseif* condition. If this is done, it's important to group the tests based on what is being required. For example, the tests

$$x \geq y \text{ or } (x * y > 50 \text{ and } z > 8)$$

can produce a different result from

$$(x \geq y \text{ or } x * y > 50) \text{ and } z > 8.$$

Finally, as the latter tests indicate, it is possible to have arithmetic expressions as part of a test.

D.5 Function Programs

The first line of the m-file `ss.m`, shown in Figure D.3, is *function ss*. The other m-files that have been considered begin in a similar manner. The reason for doing this is that by declaring it a function program, all of the variables that are defined in the program are kept separate from other calculations that have been, or will be, done by MATLAB in the *Command Window*. What is considered now is how to pass information between function programs, and to consider some of the reasons why you might want to do this.

Consider the secant method to solve $f(x) = 0$, which according to (2.24), gives rise to the equation

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}, \text{ for } i = 1, 2, \cdots .$$

A function program for this is shown in the figure below (the algorithm is a modified version of the one appearing in Table 2.6). It is not necessary to understand the various steps in

Editor - /Users/mark/Desktop/Appendix4/m files/secant1.m

secant1.m

```
1   function secant1
2 -  xa=0; xb=0.5;
3 -  tol = 0.00001; err = 1;
4 -  fa = f(xa); fb = f(xb);
5 -  while err > tol
6 -      xc = xb - fb*(xb - xa)/(fb - fa);
7 -      err = abs(xc - xb);
8 -      xa=xb; fa=fb;
9 -      xb=xc; fb=f(xb);
10 -  end
11 -  fprintf('\n x = %8.3e \n',xc)
```

this program, other than it requires (in lines 4 and 9) the evaluation of $f(x)$. This is going to be done using another function program, and we want the value to be accessible to our secant program. As an example, suppose $f(x) = 3\cos(2\pi x) - 4x$. The corresponding function program is shown below. The only remaining question is, where does this second function

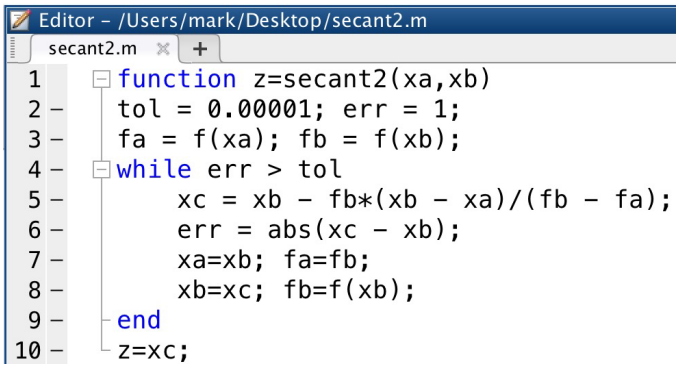
Editor - /Users/mark/Desktop/f.m

f.m

```
1   function g = f(x)
2 -  g = 3*cos(2*pi*x)-4*x;
```

go? Some like one function per m-file, and so they would put it in a file named `f.m`. It is also possible to include it at the end of the m-file that contains the *secant1* function. The latter method is used with many of the m-files included with the textbook because it results in a self-contained file. In doing this, the first function is the *main function*, and is the one that MATLAB associates with the file name. Subsequent functions in the m-file are called *local functions* and they are only available to the other functions within that file.

To illustrate another possibly, suppose you want to have a general purpose secant method solver. The idea is that the user would provide the `f.m` file and the two starting values xa and xb, and then would find the solution by issuing the command *secant2(xa,xb)*. This requires a few small adjustments to `secant1.m`, and the result is shown in the figure below (the modified file is named `secant2.m`). If one takes $xa = 0$ and $xb = 0.5$, then in the *Command Window*



they would enter the command: *x=secant2(0,0.5)*. In this way the *xa* and *xb* are passed to the secant2 function, and the computed value for the solution is passed back to the *Command Window*.

For more information about functions and what can be done with them, see the MATLAB help pages *function* and *Create Functions in Files*.

D.6 Entering Matrices and Vectors

Generally, the problems solved numerically are large enough that entering matrices and vectors by hand is impractical. Instead, for-loops or specialized commands are employed. What is described below are some of the ways to do this.

The most versatile way to enter a matrix is using nested for-loops. To illustrate, for the $n \times m$ Vardermonde matrix $\mathbf{V}$ in Section 8.3, $V_{ij} = (1/i)^{j-1}$. In MATLAB, assuming n and m have been specified, this can be entered using the commands shown in the figure below.

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1/2 & (1/2)^2 & \dots & (1/2)^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 1/n & (1/n)^2 & \dots & (1/n)^{m-1} \end{pmatrix}$$

Editor – /Users/mark/Desktop/v.m

v.m

```
1 - for i=1:n
2 -     for j=1:m
3 -         V(i,j)=(1/i)^(j-1);
4 -     end
5 - end
```

This works for all matrices, with line 3 changed accordingly. Vectors can be entered similarly, using a single for-loop. What needs to be done is to indicate whether it is a column or row vector. To illustrate, suppose the ith entry in a vector $\mathbf{x}$ is $x_i = 1/i$. How to enter this as a column or row vector is shown in the figures below (it is assumed n has been specified). For other vectors, line 3 would change accordingly. Also, to switch between row and column format just use the MATLAB's transpose command: x'.

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1/2 \\ \vdots \\ 1/n \end{pmatrix}$$

Editor – /Users/mark/Desktop/cols.m

cols.m

```
1 - x=ones(n,1);
2 - for i=1:n
3 -     x(i)=1/i;
4 - end
```

$$\mathbf{x} = \begin{pmatrix} 1 & 1/2 & \dots & 1/n \end{pmatrix}$$

Editor – /Users/mark/Desktop/rows.m

rows.m

```
1 - x=ones(1,n);
2 - for i=1:n
3 -     x(i)=1/i;
4 - end
```

For a matrix with particular patterns in its entries, it is possible to use some of the specialized commands available in MATLAB. Of particular interest are the following matrix generating commands:

***ones(n,m)*:** This is an $n \times m$ matrix consisting of all ones.

***zeros(n,m)*:** This is an $n \times m$ matrix consisting of all zeros.

***eye(n,m)*:** This is an $n \times m$ matrix with ones along the main diagonal, and zeros elsewhere.

***diag(v,k)*:** This creates a zero matrix except that the vector **v** is placed on the kth diagonal (with $k = 0$ for main diagonal, $k = 1$ for the superdiagonal, $k = -1$ for the subdiagonal, etc). Note that the size of the resulting matrix depends on the size of **v** and the value of k.

An an example, a tridiagonal matrix can be written as follows (it is assumed n is specified):

$$\begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & & & 1 \\ & & & 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 0 & & & \\ 0 & 2 & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & & & 0 \\ & & & 0 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & & & \\ 1 & 0 & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & & & 0 \\ & & & 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & & & \\ 0 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & & & 1 \\ & & & 0 & 0 \end{pmatrix}$$

$$= \quad 2 * eye(n, n) \; + \; diag(ones(n-1, 1), -1) \; + \; diag(ones(n-1, 1), 1)$$

The above formula might qualify for "advanced MATLAB," in the sense that most casual users would not know to use it. However, there is an alternative that most anyone understands. This involves using a single for-loop, and the commands are given in the figure below.

Editor - /Users/mark/Desktop/triG.m

triG.m

```
1 - for i=1:n
2 -     A(i,i)=2;
3 -     if i>1 A(i,i-1)=1; end
4 -     if i<n A(i,i+1)=1; end
5 - end
```

D.6.1 Row and Column Manipulation

An often useful command uses the colon (:) to access any particular row or column of a matrix. It is used in basically the same way that it is used in a for-loop, and in the figure below some of the possibilities are illustrated. The big difference is when the colon is used by itself (as in lines 4 and 6), which means that all possible index values are used.

Editor - /Users/mark/Desktop/colonN.m

colonN.m

1 -	`n=5; m=6;`	
2 -	`A=rand(n,m)`	Generate a random $n \times m$ matrix
3		
4 -	`x=A(:,3)`	Vector x is formed using the third column of A
5		
6 -	`A(2,:)=ones(1,m)`	Set the second row of A to all ones
7		
8 -	`A(2:4,5)=zeros(3,1)`	In the fifth column of A, set rows 2, 3, 4 to zero
9		
10 -	`y=A(4,1:2:m)`	The vector y is formed from the fourth row of A, using column entries, 1, 3, 5