Matrices and vectors can be thought of as being shorthand for collections of data—notational or bookkeeping conveniences. When using matlab (or octave\(^1\)), it is absolutely imperative that you think in terms of matrices and vectors. Traditional programming languages pick apart matrices and vectors to handle the elements on a term-by-term basis with looping structures (such as a for-loop in C or a do-loop in FORTRAN). In matlab/octave you have the ability to treat matrices and vectors as objects which need not be picked apart (i.e., matlab will do the underlying dirty work for you). You will be required to use matlab at several points in your studies and it behooves you to develop a good understanding of how to use it. The help documentation provided with matlab is an excellent resource. Use it!

Before covering some matlab basics, a review of vectors and matrices is in order. There are rules which specify how these objects combine or can be manipulated. For example, consider matrices \(A\) and \(B\) and vectors \(v_1\) and \(v_2\) given by

\[
A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix}, \quad v_1 = [x \ y \ z], \quad v_2 = \begin{bmatrix} a \\ b \end{bmatrix}. \tag{1}
\]

When describing the size of an object it is convention to give the number of rows followed by the number columns. Thus \(A\) is a \(2 \times 3\) matrix while \(B\) is \(3 \times 2\). One could describe \(v_1\) as a \(1 \times 3\) matrix and \(v_2\) as being \(2 \times 1\), but it would be more common to refer to \(v_1\) as a three-element row vector and and \(v_2\) as a two-element column vector.

Matrices can have any number of dimensions. Matrices \(A\) and \(B\) are both two-dimensional, but three-dimensional matrices are often encountered in some disciplines (and higher dimensional matrices certainly are not uncommon). Each element of a matrix is specified by the a set of indices, one index for each dimension. Vectors can be thought of one-dimensional matrices so that only a single index specifies an element. In some applications the index corresponds to an offset from the first element and in that case the counting starts from zero. In matlab the counting starts from one.

We will employ that convention here. The indices are sometimes written as a subscript or enclosed in parentheses. For a two-dimensional matrix the first index corresponds to the row and the second index to the column. For example,

\[
A_{2,3} = A(2, 3) = 6 \quad v_1(2) = y. \tag{2}
\]

---

\(^1\)Octave is the free open-source equivalent of matlab. See www.octave.org for details.
Thus, in general, \( A(i, j) \) corresponds to the element from the \( i \)th row and \( j \)th column of \( A \). We also note that the term dimension is sometimes used interchangeably with size. The context should make clear the intended quantity.

If a matrix or vector is multiplied by a scalar value (i.e., a single number), each element of the matrix or vector is scaled by that number. For example:

\[
7A = \begin{bmatrix}
7 & 14 & 21 \\
28 & 35 & 42
\end{bmatrix}.
\]

Matrices can be added to each other as can vectors, but only if they have the same size. So, none of the objects given in (1) can be added to a different object since no two of them have the same size. However, any of them could be added to itself. For example,

\[
B + B = \begin{bmatrix}
20 & 40 \\
60 & 80 \\
100 & 120
\end{bmatrix}.
\]

And, of course, \( B + B \) is simply \( 2B \).

Multiplication is also defined for matrices and vectors. The number of columns of the first object in the multiplication must be the same as the number of rows of the second object. We say that the sizes must be conformal. For example, \( B \) can be multiplied by \( v_2 \). A matrix times a vector yields a vector (you can think of the matrix as a kind of transformation operator—it maps vectors to different vectors). Each element of the result is the sum of products: the products are formed from an element from the column of the first object and the corresponding element from the row of the second object. Since you should have seen this before, an example should suffice to illustrate this:

\[
Bv_2 = \begin{bmatrix}
10 & 20 \\
30 & 40 \\
50 & 60
\end{bmatrix} \begin{bmatrix}
a \\
b
\end{bmatrix} = \begin{bmatrix}
10a + 20b \\
30a + 40b \\
50a + 60b
\end{bmatrix}.
\]

The result is a three-element column vector. Note that in general one cannot commute multiplication of vectors and matrices. Although we can multiply \( B \) by \( v_2 \) (technically we are post-multiplying \( B \) by \( v_2 \)), we cannot multiply \( v_2 \) by \( B \) (this would be called pre-multiplying \( B \) by \( v_2 \)). We can, however, pre-multiply \( B \) by \( v_1 \) since the number of columns in \( v_1 \) is the same as the number of rows in \( B \):

\[
v_1 B = \begin{bmatrix}
x & y & z
\end{bmatrix} \begin{bmatrix}
10 & 20 \\
30 & 40 \\
50 & 60
\end{bmatrix} = [10x + 30y + 50z, 20x + 40y + 60z].
\]
It is important to note that here, because of the arrangement and structure of the objects, the result is a row vector (which happens to have two elements in this case). Contrast this to (5) where the result was a column vector.

Matrices can also be multiplied by other matrices. Each element of the result corresponds to the sum of the product of elements from a row of the first matrix and a column of the second matrix. Again, since you should have seen this before, an example should suffice to illustrate. The product of $A$ and $B$ is given by

$$A B = \begin{bmatrix} 1 \cdot 10 + 2 \cdot 30 + 3 \cdot 50 & 1 \cdot 20 + 2 \cdot 40 + 3 \cdot 60 \\ 4 \cdot 10 + 5 \cdot 30 + 6 \cdot 50 & 4 \cdot 20 + 5 \cdot 40 + 6 \cdot 60 \end{bmatrix} = \begin{bmatrix} 220 \\ 490 \end{bmatrix}.$$  \hfill (7)

Note that product $BA$ is also defined since the size of the matrices is conformal for this operation too (but don’t expect this to hold in general—this is a consequence of the rows and columns of $A$ and $B$ being duals of each other). Even when both products can be defined, in general $BA$ is not equal to $AB$. The result is

$$B A = \begin{bmatrix} 90 & 120 & 150 \\ 190 & 260 & 330 \\ 290 & 400 & 510 \end{bmatrix}.$$  \hfill (8)

The transpose of an object swaps the rows and columns. The operation of transposition is often indicated with a prime (‘). Thus, the transpose of $B$ and $v_1$ are

$$B' = \begin{bmatrix} 10 & 30 & 50 \\ 20 & 40 & 60 \end{bmatrix}, \quad v_1' = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$  \hfill (9)

Although the product $Av_1$ is not defined, the product $Av_1'$ is:

$$A v_1' = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + 2y + 3z \\ 4x + 5y + 6z \end{bmatrix}.$$  \hfill (10)

Although addition (and, by extension subtraction) and multiplication are defined for matrices and vectors. Division is not. *matlab* lets you define division on a term-by-term basis, but that is another beast altogether. So, never, ever, ever, try to divide a matrix or vector by another matrix or vector! (You can divide by a scalar since that is just multiplication by the inverse of the scalar.)

*matlab*

The command-line prompt in *matlab* is two greater-than signs. *matlab* will ignore everything on a line that appears after a percent sign (so that comments can be added to lines).
Matrices or vectors are enclosed in square brackets. When entering matrices, columns are separated by commas or spaces. Rows are separated by semicolons or by a new line (i.e., the `<enter>` character). Thus the following are all equivalent:

```matlab
>> A=[1 2 3; 4 5 6] % columns separated by spaces, rows by semicolon
>> A=[1,2,3; 4,5,6] % columns separated by commas, rows by semicolon
>> A=[1 2 3
    4 5 6] % columns separated by spaces, rows by newline
```

After entering these values, by default `matlab` will echo the result back to you. However the output can be suppressed by putting a semicolon at the end of a statement. Thus we would get

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

However, if we entered “A=[1 2 3; 4 5 6];”, the matrix A would be defined, but nothing would be echoed back to us. To see the contents of the matrix A, just enter A at the prompt and hit return. Multiple commands can be put on a single line if they are separated by semicolons.

To pick out the element of a vector or matrix, we specify the indices enclosed in parentheses. Thus

```matlab
>> A(2,3)
```
```
ans =
    6
```

Addition, subtraction, and multiplication are represented with the usual keyboard symbols: +, -, and *. Unlike `mathematica`, `matlab` is not a symbolic manipulation package. One cannot manipulate equations with undefined variables. With that in mind, here is a transcript of a `matlab` session that illustrates some of the basic vector and matrix operation discussed above.
>> % define the matrices A and B and row vector v1 and column vector v2
>> A=[1 2 3; 4 5 6]; B=[10 20; 30 40; 50 60]; v1=[7 8 9]; v2=[11; 12];
>> A*B % product of A and B

ans =

    220    280
    490    640

>> B*A % product of B and A

ans =

    90    120    150
   190    260    330
   290    400    510

>> A*v1 % multiplication of A and v1 -- not conformal! so we get error
??? Error using ==> *
   Inner matrix dimensions must agree.

>> A*v1' % multiplication of A and transpose v1 -- conformal, so okay

ans =

    50
    122

>> v1*B % pre-multiply B by v1 and get a row vector

ans =

    760    1000

>> B*v2 % post-multiply B by v2 and get a column vector

ans =

    350
    810
There are actually several ways in which a vector or matrix can be created (such as with the `ones` or `zeros` commands). One way in which vectors can be created is with colon notation. If one writes

\[ v=\text{first}:\text{increment}:\text{last} \]

that tells MATLAB to create a vector \( v \) where the first element has a value “first”, then subsequent elements should have a value of \( \text{first} + (n-1)\times\text{increment} \) where \( n \) is the index of the element. This continues until the value exceeds the \( \text{last} \) value. Thus the following command

\[ v=3:.25:4.4 \]

will create a vector with elements

\[
3.0000 \quad 3.2500 \quad 3.5000 \quad 3.7500 \quad 4.0000 \quad 4.2500
\]

An alternative form of the colon notation is

\[ v=\text{first}:\text{last} \]

In this case with just two terms, the increment is assumed to be one. If you want a fixed number of equally spaced points between the first and last value (and you want to ensure the last value is included in the resulting vector), you can use the `linspace` command.

Many MATLAB commands can be distributed over a matrix or vector, i.e., they can operate on the matrix on a term-by-term basis. For example, assume the vector `tea` has been defined as follows:

\[
\text{tea}=0:.05:7;
\]

Thus `tea` would be a row vector with 141 elements, the elements being 0.00, 0.05, 0.10, \ldots, 7.00.

If we want a vector which contains samples of the function \( v_g(t) = 6 \sin(12t) \) at times \( 0 \leq t \leq 7 \), that could be accomplished in MATLAB using a command such as

\[
\text{vg} = 6*\sin(12*\text{tea});
\]
The resulting vector \( v_g \) will also have 141 elements corresponding to six times the sine of 12 times the elements of \( \text{tea} \). Note that the colon notation could be used directly in the construction of the \( v_g \) vector (thus eliminating the intermediate vector \( \text{tea} \)). This would be done as

\[
v_g = 6 \times \sin(12 \times (0:0.05:7));
\]

(Here the parentheses around the colon notation term are important. If they are not there \textit{matlab} would first multiple 12 by zero and then evaluate the sine of \( 0:0.05:7 \).

Finally, it is worth noting some of the ways in which to get help within \textit{matlab}:

\texttt{help} Typing \texttt{help command} where "command" is the command for which you seek help, will provide information about that command. For example, typing \texttt{help ones} produces the following output

\begin{verbatim}
ONES   Ones array.
       ONES(N) is an N-by-N matrix of ones.
       ONES(M,N) or ONES([M,N]) is an M-by-N matrix of ones.
       ONES(M,N,P,...) or ONES([M N P ...]) is an M-by-N-by-P-by-...
       array of ones.
       ONES(SIZE(A)) is the same size as A and all ones.

See also ZEROS.
\end{verbatim}

\texttt{lookfor} Entering \texttt{lookfor foo} will search for occurrences of the string "foo" in the first comment-line in all the m-files. m-files specify the implementation of commands. The first comment-line gives a summary of the command. \texttt{lookfor foo} returns the name of the commands corresponding to the m-files and the comment line itself (note that the command name will appear in upper case but the command itself would typically be invoked in lower case—\textit{matlab} is case sensitive).

For example, if we wanted to find which commands use the string \texttt{Fourier} in their first comment line, we could type \texttt{lookfor Fourier} which produces the following output

\begin{verbatim}
FFT   Discrete Fourier transform.
FFT2  Two-dimensional discrete Fourier Transform.
FFTN  N-dimensional discrete Fourier Transform.
IFFT  Inverse discrete Fourier transform.
IFFT2 Two-dimensional inverse discrete Fourier transform.
\end{verbatim}
IFFTN N-dimensional inverse discrete Fourier transform.
XFOURIER Graphics demo of Fourier series expansion.
DFTMTX Discrete Fourier transform matrix.

Note that the command for the discrete Fourier transform is really *fft* and not *FFT*.

doc Typing `doc` by itself will launch the help desk (HTML format). Typing `doc command` will display the HTML documentation for the specified command.