Introduction to MATLAB

MATLAB, mathworks.com/products/matlab.html, (short for "MATrix LABoratory") is a software program developed and first marketed by a company called MathWorks as far back as 1984. MATLAB is particularly useful for performing numerical linear algebra calculations involving vectors and matrices. GNU Octave, octave.org, is an open-source alternative (clone) of MATLAB that can interpret and execute MATLAB commands. Octave Online, octave-online.net, is a free web user interface for GNU Octave. We introduce some basic MATLAB commands, all of which can be executed in Octave or Octave Online.

1 Basic Arithmetic Operations and Format

In MATLAB, commands are entered to the right of the `>>` prompt and are followed by pressing `<enter>`. In Octave Online, the prompt may look different, such as `octave:1>`. Calculations can be performed using basic arithmetic operations such as `+ - * / ^` along with the use of parentheses (round brackets) for order of operations. For example, to evaluate

$$\frac{2 \cdot 3 + 4}{5^6 - 7},$$

enter

```
>> (2*3+4)/(5^6-7)
ans = 6.4029e-04
```

which means the answer is (approximately) 6.4029×10^{-4}, or about 0.00064029. To express the answer as a fraction (rational number), we use the command `format rat`.

```
>> format rat
>> (2*3+4)/(5^6-7)
ans = 5/7809
```

For a decimal answer with many more significant figures (typically 16), we use `format long`.

```
>> format long
>> (2*3+4)/(5^6-7)
ans = 6.402868485081316e-04
```

The command `format short`, which is the default setting, will produce the original decimal answer having fewer significant figures.

After using the `format` command, all successive calculations will be displayed using the same format until a new format is selected.

The number π is entered `pi`. The default `format short` will display π using four decimal places, while `format rat` will approximate π as a fraction.

```
>> format short
>> pi
ans = 3.1416
>> format rat
>> pi
ans = 355/113
```

Numbers represented using scientific notation can be entered using `e`. For example, the numbers 3.45×10^5 and -7.9×10^{-4} are entered

```
>> 3.45e5
ans = 345000
>> -7.9e-4
ans = -7.9000e-04
```

2 Elementary Functions

The absolute value command is `abs`. For example,

```
>> abs(-13)
ans = 13
```

The factorial command is `factorial`. For example,

```
>> factorial(5)
ans = 120
```

To compute the square root of a number we use `sqrt`. To compute the n^{th} root of a number we use `nthroot` with a second argument being the root n. For example, to evaluate $\sqrt{36}$ and $\sqrt[5]{32}$ we enter

```
>> sqrt(36)
ans = 6
>> nthroot(32,5)
ans = 2
```

To evaluate an exponential function we use `exp` and to evaluate a natural logarithmic function we use `log`. For logarithms with base 10 we use `log10`. For example, here are the results of evaluating e^2, $\ln 3$ and $\log_{10} 100$.

```
>> exp(2)
ans = 7.3891
>> log(3)
ans = 1.0986
```

```
>> log10(100)
ans = 2
```

Trigonometric functions are evaluated using the commands `sin`, `cos`, `tan`, `csc`, `sec` and `cot`, assuming the angle is in radians. If the angle is in degrees, we append an extra `d` at the end, for example `sind` instead of `sin`. To evaluate inverse trigonometric functions we prefix the commands with an `a`. For example, to evaluate $\sin^{-1}$ (or arcsin) we use `asin` or `asind` depending on whether the answer is to be in radians or degrees, respectively. Consider the following examples of evaluating $\sin(\pi/6)$, $\sin(45^\circ)$, and $\sin^{-1}(1)$ with the answer in the latter case expressed in radians and then degrees.

```
>> sin(pi/6)
ans = 0.5000
>> sind(45)
ans = 0.7071
>> asin(1)
ans = 1.5708
>> asind(1)
ans = 90
```

3 Complex Numbers

The imaginary number i is entered `i`. Here are some examples of complex number calculations beginning with evaluating i^2 and then adding, multiplying and dividing the complex numbers $2 + 4i$ and $3 + 7i$.

```
>> format rat
>> i^2
ans = -1
>> (2+4i)+(3+7i)
ans =   5 + 11i
>> (2+4i)*(3+7i)
ans = -22 + 26i
>> (2+4i)/(3+7i)
ans = 17/29 - 1/29i
```

The commands `real` and `imag` will extract the real and imaginary parts of a complex number, while `conj` will return its conjugate.

```
>> real(8-5i)
ans = 8
>> imag(8-5i)
ans = -5
>> conj(8-5i)
ans =  8 + 5i
```

Applying `abs` to a complex number z will evaluate its modulus (magnitude) $|z|$. Meanwhile, the command `angle` will give its argument (phase angle) $\arg(z)$ in radians. For example,

```
>> abs(5+12i)
ans = 13
>> angle(5+12i)
ans = 1.1760
```

Thus, in polar form $5 + 12i \approx 13e^{1.1760i}$.

4 Assignment and Output

Variables can be assigned values using `=` and used in subsequent calculations. For example, the following code defines $a = 2$ and $b = 3$ and then adds them together.

```
>> a=2
a = 2
>> b=3
b = 3
>> a+b
ans = 5
```

The use of a semicolon following a command will suppress its output.

```
>> a=2;
>> b=3;
>> a+b
ans = 5
```

Multiple commands can be entered on the same line if separated by commas or, if their output is to be suppressed, semicolons. For example, the following code defines $x = 7$ and $y = -5$ and then computes and outputs the values of both $x^2 + y^2$ and $2xy$.

```
>> x=7;y=-5;x^2+y^2,2*x*y
ans = 74
ans = -70
```

If no variable is assigned to an output, then the output is assigned the variable `ans`. For example,

```
>> 2+3
ans = 5
>> ans^2
ans = 25
```

5 Vectors

Vectors are created using square brackets. To create a row vector we separate its components using either spaces or commas. To create a column vector we separate its components using semicolons. As before, we can assign names to vectors by using `=`. Consider the following examples. The first two commands are equivalent and they define the row vector **u** while the third command defines the column vector **v**.

```
>> u=[1 2 3]
u =
   1   2   3
>> u=[1,2,3]
u =
   1   2   3
>> v=[1;2;3]
v =
   1
   2
   3
```

Vectors can be added or subtracted using `+` and `-`, while scalar multiplication and division of vectors are done using `*` and `/`. For example,

```
>> u=[1 2 3];
>> v=[4 5 6];
>> u+v
ans =
   5   7   9
>> 7*u
ans =
    7   14   21
>> v/2
ans =
   2.0000   2.5000   3.0000
>> 8*u-3*v
ans =
  -4   1   6
```

The length (magnitude) of a vector is found using the command `norm`. For example,

```
>> u=[1 2 3];
>> norm(u)
ans = 3.7417
```

which is a decimal approximation of the exact value of $\sqrt{14}$.

The dot product of two vectors and the cross product of two 3-dimensional vectors are found using the `dot` and `cross` commands. For example,

```
>> u=[1 2 3];
>> v=[4 5 6];
>> dot(u,v)
ans = 32
>> cross(u,v)
ans =
  -3   6  -3
```

The following code computes the triple scalar product of three 3-dimensional vectors.

```
>> u=[1 2 3];v=[4 5 6];w=[1 5 7];
>> dot(u,cross(v,w))
ans = 6
```

6 Matrices

Matrices are defined like vectors with semicolons used to separate rows. For example, the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ is entered

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

Entries in each row can be separated by spaces, as above, or commas.

The size $m \times n$ of matrix A is found using the `size` command.

```
>> size(A)
ans =
   2   3
```

This confirms that A is a 2×3 matrix, having 2 rows and 3 columns.

The entry in row i and column j of matrix A can be found by entering `A(i,j)`. Meanwhile, we can extract the entire i^{th} row of A or j^{th} column of A using `A(i,:)` and `A(:,j)`, respectively. For example,

```
>> A(2,1)
ans = 4
>> A(2,:)
ans =
   4   5   6
>> A(:,1)
ans =
   1
   4
```

The transpose of matrix A is found using an apostrophe `'`.

```
>> A'
ans =
   1   4
   2   5
   3   6
```

Transpose can also be used to convert a row vector into a column vector and vice versa. For example,

```
>> v=[1 2 3]'
v =
   1
   2
   3
```

Addition, subtraction, scalar multiplication and scalar division of matrices are done the same as with vectors using `+ - * /`. For example,

```
>> A=[1 2 3; 4 5 6]
A =
   1   2   3
   4   5   6
>> B=[3 2 1; 6 5 4]
B =
   3   2   1
   6   5   4
>> A+B
ans =
    4    4    4
   10   10   10
>> 2*A
ans =
    2    4    6
    8   10   12
>> B/3
ans =
   1.0000   0.6667   0.3333
   2.0000   1.6667   1.3333
```

Matrices are multiplied, assuming their sizes are compatible, by using `*`. In the following example, the product AB is defined, but BA is undefined

```
>> A=[1 2; 3 4]
A =
   1   2
   3   4
>> B=[5 6 7; 8 9 10]
B =
    5    6    7
    8    9   10
>> A*B
ans =
   21   24   27
   47   54   61
```

```
>> B*A
error...
```

Positive integer powers of matrices can be computed using `^`. For example, for the matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, the power $A^3 = AAA$ is found using

```
>> A^3
ans =
    37    54
    81   118
```

We can find the inverse of an invertible matrix by using the `inv` command. For example, using fractions, if $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, then $B = A^{-1}$ can be found as follows.

```
>> format rat
>> A=[1 2; 3 4]
A =
           1           2
           3           4
>> B=inv(A)
B =
          -2           1
         3/2        -1/2
```

If we multiply A and B together to verify they are inverses, we get

```
>> A*B
ans =
           1           0
           *           1
```

Note the presence of `*` in the matrix AB. When using `format rat`, `*` indicates the number is too small or too large to be properly displayed using integers or rational numbers. In this case the entry is zero. However, because MATLAB performs calculations numerically, there is some possible roundoff error that could indicate the entry is not exactly zero. These are the results we get if we display the answer using `format short` or `format long`.

```
>> format short
>> A*B
ans =
   1.0000         0
   0.0000    1.0000
>> format long
>> A*B
ans =

   1.000000000000000                   0
   0.000000000000001   1.000000000000000
```

While * can often be interpreted as zero, at other times it's a sign that the number is too large to be displayed using `format rat`. Consider the following example where we evaluate A^9.

```
>> A=[1 2; 3 4]
A =
   1   2
   3   4
>> format rat
>> A^9
ans =
     890461           *
          *           *
>> format long
>> A^9
ans =
    890461   1297782
   1946673   2837134
```

7 More on Matrices

An $m \times n$ zero matrix can be created using the `zeros` command, while an $n \times n$ identity matrix can be created using the `eye` command. For example,

```
>> A=zeros(2,3)
A =
   0   0   0
   0   0   0
>> B=eye(4)
B =
Diagonal Matrix
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

A diagonal matrix can be constructed using the `diag` command by specifying the diagonal entries using a vector. For example,

```
>> C=diag([1 2 3 4])
C =
Diagonal Matrix
   1   0   0   0
   0   2   0   0
   0   0   3   0
   0   0   0   4
```

The reduced row echelon form (RREF) of a matrix can be found using the `rref` command. For example,

```
>> A=[1 2 3 4; 5 6 7 8; 9 10 11 12]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
>> rref(A)
ans =
    1    0   -1   -2
    0    1    2    3
    0    0    0    0
```

The rank of a matrix is found using `rank`. The rank of the previous matrix is 2, as confirmed by the command

```
>> rank(A)
ans = 2
```

For a square matrix, its determinant, trace, and set of eigenvalues are found using `det`, `trace`, and `eig`, respectively. For example,

```
>> A=[1 2; 3 4]
A =
    1    2
    3    4
>> det(A)
ans = -2
>> trace(A)
ans = 5
>> eig(A)
ans =
   -0.3723
    5.3723
```

The exact eigenvalues of $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ are $\lambda_1 = \dfrac{5-\sqrt{33}}{2} \approx -0.3723$ and $\lambda_2 = \dfrac{5+\sqrt{33}}{2} \approx 5.3723$. For such a diagonalizable matrix A, the command `[P,D]=eig(A)` will produce a diagonal matrix D, whose diagonal entries are the eigenvalues of A, along with a matrix P whose column vectors are normalized (i.e. unit) eigenvectors corresponding to the eigenvalues in the same order, whereby $P^{-1}AP = D$. For example, using the same matrix A we get

```
>> [P,D]=eig(A)
P =
   -0.8246   -0.4160
    0.5658   -0.9094
D =
Diagonal Matrix
```

```
   -0.3723         0
         0    5.3723
```

This shows that $\mathbf{x}_1 = \begin{bmatrix} -0.8246 \\ 0.5658 \end{bmatrix}$ is an eigenvector associated with λ_1 and $\mathbf{x}_2 = \begin{bmatrix} -0.4160 \\ -0.9094 \end{bmatrix}$ is an eigenvector associated with λ_2.

8 Solving Polynomial Equations

The `roots` command can be used to numerically solve polynomial equations by specifying the coefficients using a vector. For example, to solve the quadratic equation $x^2 - 2x - 8 = 0$, we enter

```
>> roots([1 -2 -8])
ans =
    4
   -2
```

We can use `roots` to find roots of a real or complex number. For example, to find the fourth roots of $-16i$, we note that this is equivalent to solving the equation $z^4 + 16i = 0$ and so we get the following.

```
>> roots([1 0 0 0 16i])
ans =
   -1.8478 + 0.7654i
   -0.7654 - 1.8478i
    0.7654 + 1.8478i
    1.8478 - 0.7654i
```

Given a square matrix, the command `poly` will produce a vector consisting of the coefficients of the characteristic polynomial $\det(\lambda I - A)$. For example,

```
>> A=[1 2; 3 4]
A =
    1    2
    3    4
>> poly(A)
ans =
    1   -5   -2
```

From this we conclude that if $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, then $\det(\lambda I - A) = \lambda^2 - 5\lambda - 2$, whose roots are the eigenvalues of A. Using the `roots` command we get the same answer as we got previously with the `eig` command.

```
>> roots(poly(A))
ans =
    5.3723
   -0.3723
```

Recall that if A is $n \times n$, then $\det(A - \lambda I) = (-1)^n \det(\lambda I - A)$, meaning that if n is odd, then the coefficients of $\det(A - \lambda I)$ will be the negative of those found using `poly`.

9 Solving Linear Systems

Consider the problem of solving a system of linear equations $A\mathbf{x} = \mathbf{b}$. If A is invertible, then $\mathbf{x} = A^{-1}\mathbf{b}$ can be found by either entering `inv(A)*b` or the shortcut `A\b`. For example, to solve the system

$$\begin{cases} x + 2y - z = 2 \\ 3x + 7y - 5z = 5 \\ -x - 2y = 1 \end{cases}$$

we enter

```
>> A=[1 2 -1; 3 7 -5; -1 -2 0]
A =
   1   2  -1
   3   7  -5
  -1  -2   0
>> b=[2;5;1]
b =
   2
   5
   1
>> x=inv(A)*b
x =
   13
   -7
   -3
>> x=A\b
x =
   13
   -7
   -3
```

and so $x = 13$, $y = -7$ and $z = -3$. We could also form the augmented matrix $\left[A \mid \mathbf{b}\right]$ using the code `[A b]` and row reduce it, as follows, to extract the answer.

```
>> [A b]
ans =
   1   2  -1   2
   3   7  -5   5
  -1  -2   0   1
>> rref(ans)
ans =
```

```
1     0     0    13
0     1     0    -7
0     0     1    -3
```

Note that vertical bars are not displayed in augmented matrices.

* * *