Using MATLAB for Linear Algebra

This is a collection of basic information and techniques for using MATLAB to explore matrix properties, manipulations, and identities. These only address "numerical" (`x` is a particular number or vector of numbers or matrix of numbers) rather than "symbolic" (`x` is a variable without any particular value) functions of MATLAB.

1. **Overview: General Things to Know**

   a. π is `pi` and $\sqrt{-1}$ is either `i` or `j`.

   b. MATLAB versions of common functions (these are applied *element-by-element* if the argument is a vector or matrix):

Function	MATLAB	Example
x^n	`x^n`	$x^2 \longrightarrow$ `x.^2`
e^x	`exp(x)`	$e^{-5} \longrightarrow$ `exp(-5)`
$\ln x$	`log(x)`	$\ln \pi \longrightarrow$ `log(pi)`
$\sinh x, \cosh x$	`sinh(x)`, `cosh(x)`	$\sinh 1 \longrightarrow$ `sinh(1)`
$\sin x, \cos x$	`sin(x)`, `cos(x)`	$\cos \pi \longrightarrow$ `cos(pi)`
$\sin^{-1} x$ or $\arcsin x$	`asin(x)`	$\arcsin 0.5 \longrightarrow$ `asin(0.5)`
$\sqrt{x}$	`sqrt(x)`	$\sqrt{2.5} \longrightarrow$ `sqrt(2.5)`

   There are matrix versions of `exp`, `log`, `sqrt` called `expm`, `logm`, `sqrtm`, see below.

   c. You can use the $\uparrow$ and $\downarrow$ keys to move back and forth through the list of past commands ("history"). If you bring back a previous command, you can change it (use the $\leftarrow$ and $\rightarrow$ keys to move to parts you want to delete or replace or add something) and rerun the commands (by pressing "Enter").

   d. Getting help: To find help for any function (e.g., `exp`), type `help exp` (or whatever the function is) at the `>>` command prompt. Or, bring up the "Help Browser" using F1 (or "MATLAB Help" under the "Help" menu in the main MATLAB window).

   e. Use the `clear` statement to set all variables to zero (to avoid conflicts with previous definitions).

2. **Complex Numbers**

   a. **Cartesian form.** You can use either `i` or `j` for $\sqrt{-1}$ (since we're in a physics class, we'll use `i` exclusively). If `i` appears in a numerical expression, MATLAB will simply interpret the number as a complex number. For example, if we want to set z equal to $3 + 4i$, simply type it in this way:

```
>> z = 3 + 4*i
```

   We could even omit the `*` in this case, but it is best to leave it in, since we need it if we are using a variable such as y:

```
>> x = 1/sqrt(2);
>> y = -1/sqrt(3);
>> z = x + i*y
```

We can use any of the standard functions (e.g., exponentials or trigonometric functions) directly. For example,

```
>> sin(1-sqrt(2)*i)
ans = 1.8329 - 1.0455i
```

Note that the answer comes back in standard $x + yi$ form with decimals (as opposed to fractions or explicit π's).

b. **Polar form.** If we want to enter a number in the form $re^{i\theta}$, we just use the `exp` function:

```
>> r = 1;
>> theta = pi/6;
>> z = r*exp(i*theta)
z = 0.8660 + 0.5000i
```

Note that the answer comes back in Cartesian form.

c. **Arithmetic with complex numbers.** All of the standard operations (`+ - * /`) will work as expected (i.e., correctly) with complex numbers.

d. **Complex conjugate.** The function `conj` takes the complex conjugate:

```
>> conj(2-3*i)
ans = 2.0000 + 3.0000i
```

e. **Real and imaginary parts.** The functions `real` and `imag` do the trick:

```
>> z = (1 + i)/(2 - i);
>> real(z)
ans = 0.2000
>> imag(z)
ans = 0.6000
```

Note that you always end up with the decimal version of numbers.

f. **Modulus and angle.** You can find r and θ using the `abs` and `angle` functions:

```
>> z = 2 * exp(i*pi/4)
z = 1.4142 + 1.4142i
>> abs(z)
ans = 2
>> angle(z)   % answer should be pi/5 = 0.7854
ans = 0.7854
```

3. **Creating Vectors and Matrices and Accessing Elements**

Both vectors and matrices are specified by entries between `[ ]`'s with semicolons `;` used to separate rows. So

$$A = \begin{bmatrix} 1 & -2 & 1 \\ 2 & -5 & 4 \\ -1 & 3 & -2 \end{bmatrix} \qquad B = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} \qquad C = \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$$

are entered as

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

>> B = [3; 4; 5]
B =
     3
     4
     5

>> C = [3 4 5]
C =
     3     4     5
```

To get the "ij" matrix element of A, use `A(i,j)`. So

```
>> A(1,2)
ans = -2
>> A(2,2)
ans = -5
```

The n^{th} row is `A(n,:)` and the m^{th} column is `A(:,m)`. So the 2nd row and 3rd columns are:

```
>> A(2,:)
ans =
     2    -5     4
>> A(:,3)
ans =
     1
     4
    -2
```

4. **Special Matrices**

   a. For a 3×3 unit matrix, use `eye(3)` while for $N \times N$ use `eye(N)`.

   b. `zeros(N)` is an $N \times N$ matrix of zeros while `zeros(1,N)` is an N-dimensional row vector of zeros and `zeros(N,1)` is an N-dimensional column vector of zeros.

   c. `ones(N)` is an $N \times N$ matrix of ones while `ones(1,N)` is an N-dimensional row vector of ones and `ones(N,1)` is an N-dimensional column vector of ones.

   d. **Random matrices.** Use `rand(N)` to generate an $N \times N$ matrix whose entries are random numbers uniformly distributed between 0 and 1. E.g.,

```
>> M = rand(3)
M =
    0.1239    0.4238    0.0785
    0.7745    0.1592    0.7084
    0.1123    0.2949    0.0181
```

   The numbers are really "pseudo-random" numbers. See `help rand` for more info. To generate a uniform distribution of random numbers on a specified interval [a,b], multiply the output of rand by (b-a), then add a. For example, to generate a 5-by-5 array of uniformly distributed random numbers on the interval [10,50],

```
>> a = 10; b = 50;
>> x = a + (b-a) * rand(5);
```

   e. **Random complex matrix.** We can combine a real and an imaginary random matrix to get a complex one:

```
>> C = rand(3) + i * rand(3)
C =
   0.8189 + 0.3162i   0.2035 + 0.3700i   0.3652 + 0.0847i
   0.4283 + 0.5119i   0.5217 + 0.2280i   0.9393 + 0.6571i
   0.3677 + 0.3355i   0.6054 + 0.9477i   0.4161 + 0.5234i
```

   f. **Normally distributed random matrices.** Use `randn(N)` to generate an $N \times N$ matrix whose entries are random numbers distributed according to a normal distribution (i.e., a bell-shaped curve) with mean zero and standard deviation one. E.g.,

```
>> M = randn(3)
M =
   -0.0956   -1.3362   -0.6918
   -0.8323    0.7143    0.8580
    0.2944    1.6236    1.2540
```

5. **Matrix Operations**

   a. **Matrix multiplication.** Ordinary matrix multiplication is performed by using `*`. In contrast, `.*` is used for element-by-element operations (e.g., `A*B` is matrix multiplication while `A.*B` multiplies each element in A by the corresponding one in B).

   b. **Inverse of a matrix.** The inverse of the square matrix A is designated A^{-1} and is defined by $AA^{-1} = A^{-1}A = I$, where I is the identity matrix. We can find the inverse of a matrix either by raising A to the -1 power, i.e., `A^(-1)`, or with the `inv(A)` function.

   c. **Determinant of a matrix.** The `det` function returns the determinant of a square matrix. That is, `det(A)` gives the determinant of the matrix A.

   d. **Exponential of a matrix.** The `expm` function returns the exponential of a matrix, as defined by its Taylor series. So `expm(A)` gives e^A. [Note: if you use `exp(A)` by mistake (no `m` at the end of the name), you'll get a matrix whose elements are each the exponential of the corresponding matrix element in A.]

   e. **General matrix functions.** To calculate the cosine, sine, or logarithm or a matrix A, use `funm(A,'cos')`, `funm(A,'sin')`, or `funm(A,'log')`.

   f. **Trace of a matrix.** The sum of the diagonal matrix element of matrix A is `trace(A)`.

   g. **Adjoint and transpose of a matrix.** The adjoint of matrix A (designated $A^\dagger$), which is the complex conjugate of the transpose, is found from `A'` or the function `ctranspose(A)`. If you want just the transpose A^T of A and not the complex conjugate, use `A.'` or `transpose(A)`.

   h. **Eigenvalues and eigenvectors of a matrix.** `E = eig(A)` gives a vector with the eigenvalues of the matrix A. `[V,D] = eig(A)` gives a diagonal matrix D of eigenvalues and a matrix V whose *columns* are the corresponding eigenvectors.

   The n^{th} row of M is `M(n,:)` and the m^{th} column is `M(:,m)`. So the eigenvector v_1 and eigenvalue λ_1 are (using random normally distributed matrix M from above.)

```
>> [V D] = eig(M)
V =
    0.5736    0.4791   -0.4836
    0.6074   -0.5096    0.5685
   -0.5495    0.7147    0.6655
D =
   -0.8479         0         0
         0    0.2937         0
         0         0    2.4269
>> v1 = V(:,1)
v1 =
    0.5736
```

```
    0.6074
   -0.5495
>> lambda1 = D(1,1)
lambda1 = -0.8479
```

We can verify that $Mv_1 = \lambda_1 v_1$:

```
>> M*v1
ans =
   -0.4863
   -0.5150
    0.4660
>> lambda1*v1
ans =
   -0.4863
   -0.5150
    0.4660
```

i. **Powers of Matrices.** To get A^3, just use `A^3`, and so on.

j. **Bra's and ket's.** We associate the "ket" $|V\rangle$ (or $|1\rangle$ or whatever) with a column vector. The "bra" $\langle V|$ is the adjoint of $|V\rangle$, i.e., $\langle V| = (|V\rangle)^\dagger$. Then $\langle V|W\rangle$ is simply the inner product, which is a generalized dot product. Note that $\langle W|V\rangle = \langle V|W\rangle^*$. Examples:

```
>> Vket = [2; 3i]
Vket =
   2.0000
        0 + 3.0000i
>> Vbra = Vket'    % note that the i's change sign
Vbra =
   2.0000                  0 - 3.0000i
>> Wket = [i; -1]
Wket =
        0 + 1.0000i
  -1.0000
>> Vbra*Wket       % just row vector times column vector
ans = 0 + 5.0000i
>> Wket'*Vket      % in this order, we get the complex conjugate
ans = 0 - 5.0000i
```

k. **Unit Vectors.** Given $|V\rangle$, its magnitude is $\sqrt{\langle V|V\rangle}$, so the unit vector is $|V\rangle/\sqrt{\langle V|V\rangle}$. In MATLAB, we can use the `norm` function for the magnitude. E.g.,

```
>> Vket_unit = Vket/sqrt(Vbra*Vket)   % the basic definition
Vket_unit =
```

```
   0.5547
        0 + 0.8321i
>> Vket_unit = Vket/norm(Vket)        % an easier way
Vket_unit =
   0.5547
        0 + 0.8321i
```

l. **Isolating the Diagonal Elements.** If M is a square matrix, then `diag(M)` is a vector with the matrix elements on the main diagonal. E.g., for the normally distributed random matrix M:

```
>> diag(M)
ans =
   -0.0956
    0.7143
    1.2540
```

To cube each diagonal element, use `diag(M).^3` (note the "`.`" before the `^`). To get a square matrix of the same size as M but with just it's diagonal elements and zeros elsewere, use `diag(diag(M))`.

m. **Random Hermitian matrices.** Generate a random complex matrix A and then a random hermitian matrix by $H = (A + A^\dagger)/2$.

n. **Random Unitary matrices.** Generate a random Hermitian matrix H as above and then $U = e^{iH}$ is unitary (so $UU^\dagger = I$). Use the MATLAB matrix exponentiation function `expm`.

6. **Timing Matrix Operations**

The functions `tic` and `toc` can be used to time one or more MATLAB operations (not just matrix functions). A stopwatch is started with `tic` and stopped with `toc`, which then displays the elapsed time. For example, to time how long it takes to calculate the determinant of a 100×100 matrix:

```
>> M = rand(100);
>> tic; det(M); toc
Elapsed time is 0.130749 seconds.
```

7. **Condition Number**

To find the condition number of a matrix, use `cond(M)`. A number near one is good; if the number is large the matrix is ill-conditioned. See the help for `svd` and `gsvd` to learn about (generalized) singular value decomposition.

8. **Solving Matrix Equations**

Suppose we want to solve the simultaneous equations

$$\begin{aligned} 3x - 2y &= 17 \\ 5x + 3y &= 3 \end{aligned}$$

We write this in the form $MX = B$, with M a matrix and B a column vector, then find the desired column vector X from $X = M^{-1}B$ using the MATLAB `inv` function:

```
>> M = [3 -2; 5 3]  % separate the rows of the matrix with ;'s
M =
     3    -2
     5     3

>> B = [17; 3]      % note the ; to make it a column vector (two rows)
B =
    17
     3

>> X = inv(M)*B     % regular matrix multiplication uses * (not .*)
X =
    3.0000
   -4.0000
```

That's the answer: $x = 3$ and $y = -4$. *Note:* If you had defined the vector B as a row vector instead of a column vector, you would have received an error like this:

```
>> B_row = [17 3]       % there is no ";" so this is a row vector
B_row =
  17     3
>> X = inv(M)*B_row
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

An alternative way to using `inv` to solve the equation is:

```
>> X = M\B   % The "\" means "matrix left division"
```

which is actually preferred because it solves the problem much more efficiently.

9. **Scripts and Functions in MATLAB**

Here is a very brief introduction to scripts and functions using the `my_area.m` script and the `circle_area.m` function as examples. They are run by typing the name (without the `.m` ending) at the command prompt, e.g., `>> my_area` will run `my_area.m`.

a. `%` is used to mark a comment to help explain what the program is doing (or supposed to do!). Everything on a line after a `%` is ignored. Examples:

```
% This is a comment line only; no MATLAB instructions.
height = 5  %  This comment might say that height is in meters
```

b. A semicolon `;` is used at the end of a line to suppress extra output from MATLAB, which might be distracting.

c. `input` is used to get a value from the user for a variable. The general form is:

```
variable_name = input('message to be displayed')
```

For example,

```
radius = input('Enter the radius of a circle: ');
```

d. `disp` is used for output of a message (in the form of text enclosed in single quotes) or the value of a variable. For example,

```
disp('The area is: ')
disp(area)
```

e. A MATLAB function is like a script but it starts with a `function` declaration with a list of outputs between `[]`'s, then an `=`, then the name of the function (same as the filename), then the list of inputs in `()`'s. Example:

```
function [area] = circle_area (radius)
```

which takes `radius` as input and returns `area`.

10. **Numerical Two-Dimensional Plotting**

Here we outline how to make two-dimensional graphs using `plot`, `line`, and `loglog`, which take arrays of numerical values as input.

a. To plot $\sin x$ from $-\pi$ to $+\pi$, we first define a set of points to plot, in the form of a vector (which is considered a $1 \times N$ matrix by MATLAB). We can specify a set of evenly spaced points with spacing 0.1 by:

```
x = -pi:0.1:pi
```

To avoid seeing this vector printed out, add a semicolon to the end:

```
x = -pi:0.1:pi;
```

If instead we want a specified number of points (say 100), then

```
x = linspace(-pi,pi,100);
```

will do ("linspace" is short for "linear space").

b. Then we give the x vector and the function to plot as the y vector to the `plot` command:

```
plot(x,sin(x))
```

which should pop up a figure window or make a new plot in an existing window (which may be hidden by other windows). This function makes a regular linear-linear plot.

c. To add a second curve to the same graph (e.g., of $\cos x$ in the same range), use:

```
line(x,cos(x))
```

d. To make a log-log plot instead, we will typically want to space the points *logarithmically* rather than linearly. To set x equal to 150 points from 10^{-5} to 10^{2}, we can use

```
x = logspace(-5,2,150);
```

To plot $x^2 e^{-x}$ from 10^{-10} to 10^{1} with 200 points:

```
x = logspace(-10,1,200);
loglog(x,x.^2.*exp(-x))
```

Note that we use `.^` and `.*` rather than `^` rather than `*`. The "`.`" means to exponentiate or multiply (or divide, etc.) each term in the vector x, rather than trying to operate on entire vectors or matrices. If you forget the "`.`" you'll get an error message.

11. **Plotting Complex Functions**

We can use plotting commands with functions of a complex variable z if we take the modulus of the function using the `abs` command. For example, suppose we want to plot the function $f(z) = 1/(z^4 + 1)$ in the complex plane for the real and imaginary parts of z varying from -2 to $+2$. Here's an excerpt from an M-file to do this.

```
% Set a grid X Y with the desired range (and 20 points on each axis)
[X Y] = meshgrid( linspace(-2,2,20), linspace(-2,2,20) );

% Define z = x + i*y for each point in the grid
Z = X + i*Y;

% Evaluate the function we want to study.
%  Note the use of "." before operations like "^" and "/".
f = abs( 1 ./ (Z.^4 + 1) );  % use "abs" to take the modulus

figure(1);                   % figure 1 will be a surface plot
surf(X,Y,f); colorbar;       % make the plot and add a color bar
xlabel('x-axis');  ylabel('y-axis');   % add labels x and y axes

figure(2);                                % figure 2 will be a contour plot
num_contours = 10;                        % use num_contours contour lines
contourf(X,Y,f,num_contours); colorbar;  % make the plot and add a color bar
xlabel('x-axis');  ylabel('y-axis');     % add labels
```