MATH 3670 - SCIENTIFIC COMPUTATION I
Fall 2015

Week 5: Applications in Numerical Analysis - cont. (Chapter 9). Advanced Graphics in MATLAB (Chapter 10)

Content

Part I
- Numeric integration
- Solving Ordinary Differential Equations (ODEs)

Part II
- 3D Plots
- Movies
- Graphics Handles
- Graphical User Interface (GUI)

• **Homework # 5** (Due Fri, Oct 2):
Chapter 9, page 318–320 # 27, 35, 39
Chapter 10, pages 341–346 # 11, 19, **21***

This is a continuation of the previous lab on the applications of MATLAB in Numerical Analysis, specifically for numerical integration of functions and for solving ordinary differential equations.

1. Numerical Integration

Numerical integration of a function of one variable can be performed using several different methods. MATLAB has two distinct ways to perform this task. One method is with using the built-in function `quad` (and relatives of it, such as `quadl`) which integrates functions given either as anonymous functions or function files. The other way is using the `trapz` command, which integrates discrete data points using the Trapezoidal Rule. More sophisticated tools exist as well; see the MATLAB help for other integrating commands.

1.1 The `quad` and `dblquad` command

The `quad` command refers to "adaptive quadrature". To integrate

$$\int_a^b f(x)\,dx$$

we use the command:

$$\mathrm{Q} = \texttt{quad(fun,a,b)};$$

where `fun` is the name (handle) of the function to be integrated, which needs to have already been defined. If there is a user-defined tolerance (error bound) `Tol` for the value of the approximate integral, then one uses

$$\mathrm{Q} = \texttt{quad(fun,a,b,Tol)};$$

The default tolerance is 10^{-6}.

As with the `fzero` command, one needs to define and pass in the function $f(x)$. There are several options, depending on the complexity of your definition of f(x). Below we give several examples:

Example 1:

$$\int_0^2 \frac{1}{x^3 - 2x + 5}\, dx$$

which in MATLAB would be, using functions defined as a string (or inline functions):

```
>> syms x
>> f1=inline(1./(x.^3-2*x+5))
>> Q=quad(f1,0,2);

>> f2=@(x) 1./(x.^3-2*x+5);
>> Q=quad(f2,0,2);
```

We could also define the function using an M-file. Here is a sample, which we will save as myfun.m:

```
function y=myfun(x)
  y=1./(x.^3-2*x+5);
```

NOTE the element by element multiplication, division, exponentiation in all of these. It is VERY important that you understand why.
We could also define the function as an anonymous function
In the MATLAB command window, we would type:

```
>> Q=quad(@myfun,0,2)
```

All these different methods should give the same answer, which is

```
Q =
    0.4213
```

Similarly, if we are integrating a built-in function like:

$$\int_0^\pi \sin(x)\, dx$$

we would type:

```
>> Q=quad(@sin, 0, pi);
```

Example 2. Suppose we now wish to integrate a double integral:

$$\int_1^2 \int_0^3 x^2 y \, dxdy.$$

NOTE the EXACT ordering of the limits of integration.

When working with an anonymous function handle:

```
>> f2=@(x,y)x.^2.*y;
>> Q=dblquad(f2,0,3,1,2);
```

When working with a function defined in a separate M-file, first type and save the M-file (here, we call it myfun.m again):

```
function z=myfun(x,y)
z=x.^2.*y;
```

Then call it in MATLAB using

```
>> Q=dblquad(@myfun,0,3,1,2);
```

The double quadrature formula only works on rectangular regions in the plane. How would we integrate over non-square regions? It depends on the complexity of the region. Here are two options- The first is where the region is somewhat simple (a circle in the plane), and the second is more general.

Example 3. The volume of a unit hemisphere $x^2 + y^2 + z^2 \leq 1$, $z \geq 0$:

$$\int_{-1}^1 \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \sqrt{1-(x^2+y^2)} \, dydx$$

First, solve for **z** then determine appropriate limits of integration.
In this case, the integrand is somewhat simple, and we can extend the definition of the integrand so that it is zero outside the area of interest. Mathematically speaking,

$$\int_{-1}^1 \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \sqrt{1-x^2-y^2} \, dydx = \int_{-1}^1 \int_{-1}^1 G(x,y) \, dydx$$

where

$$G(x,y) = \begin{cases} \sqrt{1-x^2-y^2} & \text{if } x^2+y^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

In Matlab, the expression `x.^2+y.^2<=1` is a logical statement. It returns '1' if the expression is TRUE, '0' if FALSE. Therefore, to find the volume, we could type:

```
>> f3=@(x,y)sqrt(1-(x.^2+y.^2)).*(x.^2+y.^2<=1);
>> Q=dblquad(f3,-1,1,-1,1);
```

Example 4. Alternatively (or more generally), given:

$$\int_a^b \int_{y=g_1(x)}^{y=g_2(x)} F(x,y)\,dydx = \int_a^b \left(\int_{g_1(x)}^{g_2(x)} F(x,y)\,dy \right) dx = \int_a^b G(x)\,dx$$

where, given a numerical value of x, we would write the function

$$G(x) = \int_{g_1(x)}^{g_2(x)} F(x,y)dy$$

Here's a specific example:

$$\int_0^2 \int_{x^2}^{2x} x^2 + y^2\,dydx$$

We write the M-file for the inside integral:

```
function z=myfun(x)
     n=length(x);
     z=zeros(size(x));
     for j=1:n
           a=x(j)^2;
           b=2*x(j);
           h=@(y)(a+y.^2);
           z(j)=quad(h,a,b);
     end
```

In MATLAB, type:

```
>> Q=quad(@myfun,0,2);


Q =
    6.1714
```

1.2 The `trapz` command

We illustrate the integration of function given by discrete data points on the following example: The exact value of $\int_0^\pi \sin x\,dx$ is 2.
To approximate this numerically on a uniformly spaced grid, use

```
>> X = 0:pi/100:pi;
>> Y = sin(X);
```

Then both

```
>> Z = trapz(X,Y)
```

and

```
>> Z = pi/100*trapz(Y)
```

produce

```
Z =
      1.9998
```

A nonuniformly spaced example is generated by

```
>> X = sort(rand(1,101)*pi);
>> Y = sin(X);
>> Z = trapz(X,Y);
```

The result is not as accurate as the uniformly spaced grid. One such random sample produced

```
Z =
        1.9984
```

2. Numerical Solutions of ODEs.

Many problems that arise in science and engineering require a knowledge of a function $y = y(t)$ that satisfies the first order differential equation

$$y' = f(t, y)$$

and the initial condition $y(t_0) = y_0$, where t_0 and y_0 are given real numbers and f is a function (of two variables) that satisfies certain smoothness conditions. This is also refered to as the initial value problem (since the initial data is known or given).

The general initial value problem is formulated as follows. Given function f of n variables, find $y = y(t)$ that satisfies the nth order ordinary differential equation

$$y^{(n)} = f(t, y, y', ..., y^{(n-1)})$$

together with the initial conditions $y(t_0) = y_0, y'(t_0) = y_1, ..., y^{(n-1)}(t_0) = y_{n-1}$. The latter problem is often transformed into the problem of solving a system of the first order differential equations. To this end a term "ordinary differential equations" will be abbreviated as ODEs.

Solving the initial value problems using MATLAB built-in functions MATLAB has several functions for computing a numerical solution of the initial value problems for the ODEs. They are listed in the following table

Function	Application	Method used
ode23	Nonstiff ODEs	Explicit Runge-Kutta (2, 3) formula
ode45	Nonstiff ODEs	Explicit Runge-Kutta (4, 5) formula
ode113	Nonstiff ODEs	Adams-Bashforth-Moulton solver
ode15s	Stiff ODEs	Solver based on the numerical differentiation formulas
ode23s	Stiff ODEs	Solver based on a modified Rosenbrock formula of order 2

The simplest form of the syntax for the MATLAB ODE solvers is

$$[\texttt{t,y}] = \text{solver(fun, tspan, y0)},$$

where **solver** is one of the above functions, `fun` is a string containing name of the ODE m-file that describes the differential equation, `tspan` is the interval of integration, and `y0` is the vector holding the initial value(s). If `tspan` has more than two elements, then solver returns computed values of `y` at these points. The output parameters `t` and `y` are the vectors holding the points of evaluation and the computed values of `y` at these points.

In the following example we will seek a numerical solution y at $t = 0, .25, .5, .75, 1$ to the following initial value problem

$$y' = -2ty^2,$$

with the initial condition $y(0) = 1$. We will use both the `ode23` and the `ode45` solvers. The exact solution to this problem is $y(t) = 1/(1 + t^2)$ [Convince yourself that this is indeed the solution!] The ODE m-file needed in these computations is named eq1.m

```
function dy = eq1(t,y)
% The m-file for the ODE y' = -2ty^2.
      dy = -2*t.*y.^2;
```

Now we type in the command window:

```
>> format long
>> tspan = [0 .25 .5 .75 1]; y0 = 1;
>> [t1 y1] = ode23(@eq1, tspan, y0);
>> [t2 y2] = ode45(@eq1, tspan, y0);
```

To compare the results obtained from both methods, let us create a three-column table holding the points of evaluation and the y-values obtained with the aid of the ode23 and the ode45 solvers

```
>> [t1 y1 y2]


ans =
    0 1.00000000000000 1.00000000000000
    0.25000000000000 0.94118221525751 0.94117646765650
    0.50000000000000 0.80002280597122 0.79999999678380
    0.75000000000000 0.64001788410487 0.63999998775736
    1.00000000000000 0.49999658522366 0.50000000471194
```

Example 5. Next example deals with the system of the first order ODEs

$$y_1'(t) = y_1(t) - 4y_2(t),$$
$$y_2'(t) = -y_1(t) + y_2(t),$$

with initial data $y_1(0) = 1; y_2(0) = 0$. Instead of writing the ODE m file for this system, we will use the anonymous function

```
dy = @(t,y) [1 -4;-1 1]*y;
```

The interval over which numerical solution is computed and the initial values are stored in the vectors `tspan` and `y0`, respectively

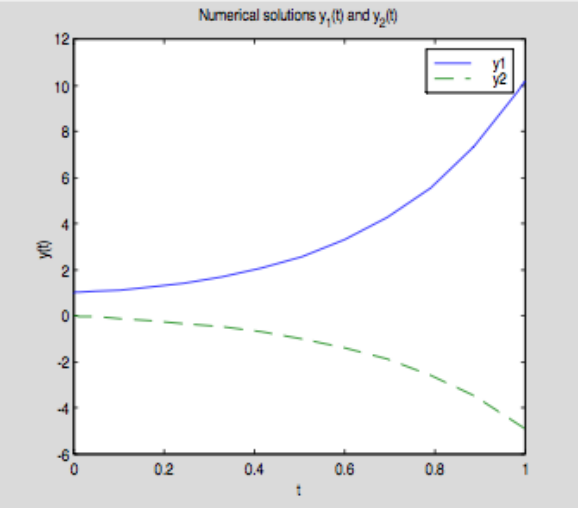
```
tspan = [0 1]; y0 = [1 0];
```

MAKE SURE you understand why these vectors are defined as they are... notice the coefficients of the above ODEs and the fact there are two of them.
Numerical solution to this system is obtained using the ode23 function

```
[t,y] = ode23(dy, tspan, y0);
```

Graphs of $y_1(t)$ (solid line) and $y_2(t)$ (dashed line) are shown below

```
plot(t,y(:,1),t,y(:,2),'--'),
legend('y1','y2'),
xlabel('t'),
ylabel('y(t)'),
title('Numerical solutions y_1(t) and y_2(t)')
```



The exact solution $(y_1(t), y_2(t))$ to this system is

$$y_1(t) = 1/2e^{-t} + 1/2e^{3*t}$$
$$y_2(t) = -1/4e^{3*t} + 1/4e^{-t}$$

The functions y_1 and y_2 given above were found using command `dsolve` which is available in the Symbolic Math Toolbox.

Example 6. Last example in this section deals with the so-called 'stiff' ODE. Consider

$$y'(t) = -1000(y - \log(1+t)) + \frac{1}{1+t}$$

with initial condition $y(0) = 1$.

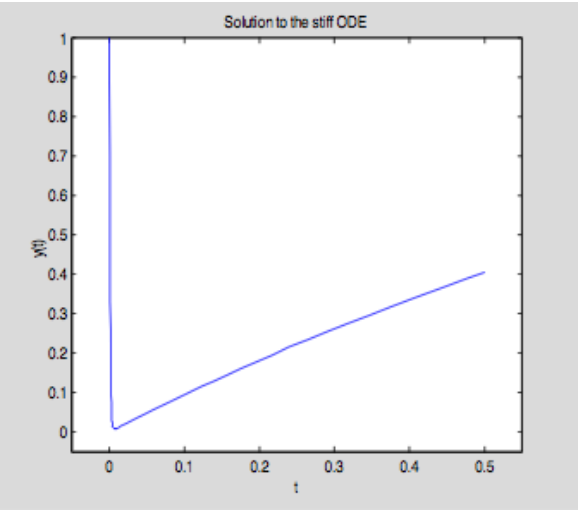
```
>> dy = @(t,y) -1000*(y  log(1 + t)) + 1/(1 + t);
```

Using the `ode23s` function (specially designed for the stiff ODEs) on the interval [0 0.5] we obtain

```
>> [t, y] = ode23s(dy,  [0 0.5] , 1);
```

To illustrate the effect of stiffness of the differential equation in question, let us plot the graph of the computed solution

```
plot(t, y), axis([-.05 .55 -.05 1] ),
xlabel('t'),
ylabel('y(t)'),
title('Solution to the stiff ODE')
```



The exact solution to this problem is $y(t) = log(1 + t) + e^{-1000*t}$. Try to plot this function on the interval [-0.05, 0.5]. The 'stiffness' of the ODE comes from the behavior of the solutions, such as depicted in this figure. An ordinary solver (such as `ode23` or `ode45` would spend a lot of iterations (hence computer time) trying to approximate the first part of the solution. Convince yourself of this!

SUMMARY: Things to remember so far

While symbolic integration of functions and of ordinary differential equations might seem like an easier route to follow, it turns out that for most applications, the symbolic computations give no explicit solutions. So numerical methods, such as those implemented for the integration of functions or integration of ordinary differential equations are the remaining viable option. In fact there are many efficient numerical methods that are already implemented in MATLAB as built in functions, and even if they are not, can easily be implemented (with some more advanced knowledge of numerical analysis). A course in Numerical Analysis is the natural path for any student wanting to get more proficient in this area get the most of a computer software such as MATLAB.

3. 3D Plots

Making 3D plots (surface plots, contour plots, and vector field plots) in MATLAB is slightly more complicated than making simple line graphs, although it follows the same ideas. Below are some examples that, with simple modifications, should enable you to create most of the pictures that you will need.

Recall that for 2D plots, we set up an array of values for x, then evaluated the function at those values and plotted the 2D coordinates:

```
x = linspace(0,1,50)
y=x.^2
plot(x,y)
```

What would be the analog for 3D? Well, it depends on the type of object being plotted.

3.1. 3D space curves: the `plot3` command

We can plot space curves parameterized by a single parameter `t`. Parametrically, a space curve in 3D can be written for some t as

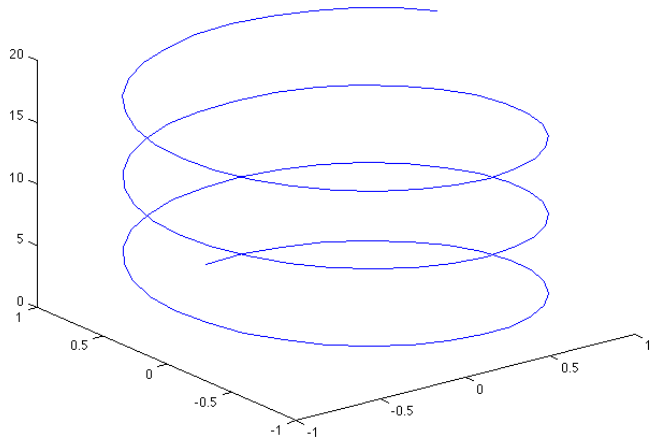
$$x = f(t), \quad y = g(t), \quad z = h(t).$$

For example, we show how to plot a helix parametrically by having $x = \sin(t), y = \cos(t)$ and $z = t$. Just as with 2D, we must set the domain. Let us say we wish to plot this helix for $0 \leq t \leq 20$. First we partition the interval $[0, 10]$ into equal subintervals:

```
>> t = linspace(0,20);
```

(default for `linspace`is 100 partition points)) To plot a space curve in 3D space we use the plot3 function. The idea is this is the 3D analog of plot, which recall plots a curve in 2D space. It takes the `x, y,`and `z` coordinates of the parameterization as arguments:

```
plot3(sin(t),cos(t),t);
```



One of the advantages to plotting in MATLAB is that we can now play with our 3D object. Go to the figure window and click on the icon that looks like a counterclockwise arrow wrapped around a cube (it is called Rotate 3D when you move your mouse on it). Then go to the plot, click and hold your mouse, and move the mouse around. You'll see that you can rotate the plot and look at if from any angle you desire. Alternately, one can use the `view` command to choose the point in space from which the plot is viewed. To view a plot from above, you can use the command

```
view(2)
```

or to view it from a different point, use `view([x,`y, z]). One can also use `view(az,`el) to specify the angles of elevation and azimuth (spherical coordinates).

Here is another example of a space curve:

$$\begin{aligned} x &= \cos(t)(e^{\cos(t)} - 2\cos(4t) - \sin(5t/12)), \\ y &= \sin(t)(e^{\cos(t)} - 2\cos(4t) - \sin(5t/12)), \\ z &= t \end{aligned}$$

for $0 \leq t \leq 20$. Try to plot it yourself, then view it from above. PLEASE understand why element by element multiplication is done here!

```
x=@(t) cos(t) .* (exp(cos(t) - 2.*cos(4.*t) - sin(5.*t/12)));
y=@(t) sin(t) .* (exp(cos(t) - 2.*cos(4.*t) - sin(5.*t/12)));
z=t;
plot3(x(t),y(t),z);
figure
plot(x(t), y(t));
```

Note the difference between the first 3D plot and the 2nd 2D plot.

3.2. 3D surface plot: the `surf` command

Suppose we want to plot the function

$$z = x^2 - 2xy - y^2 + 2.$$

over the region $-4 \leq x \leq 4$ and $-3 \leq y \leq 3$. We have to make a grid of points over which we want the heights of the surface. First partition the `x` and `y` ranges as follows (the choice of the step size is ours, keeping in mind that the resulting surface will look smooth if we choose more grid points, e.g. approximately 50×50)

```
>> x = -4:.1:4; y = -3:.1:3;
```

NOTE: in the workspace, **x** is a 1x81 vector and **y** is a 1x61 vector.
MATLAB provides a command for creating the grid

```
>> [X,Y] = meshgrid(x,y);
```

(X and Y will hold the rows and columns of this grid; we could have used any letters for these variables.). ALSO, note, in the workspace, that both **X, Y** are 2D arrays of size 61x81; i.e. **numel(x)** by **numel(y)**.
Note that we have placed a semicolon (;) after each command; this suppresses the long list of components which MATLAB generates as it creates these vectors.
As an aside: some authors state that "real" Matlab programming usually requires usage of Matlab's **(meshgrid)** function.
We create `Z` from the variables `X` and `Y` as follows (PLEASE understand why element by element multiplication is done here!):

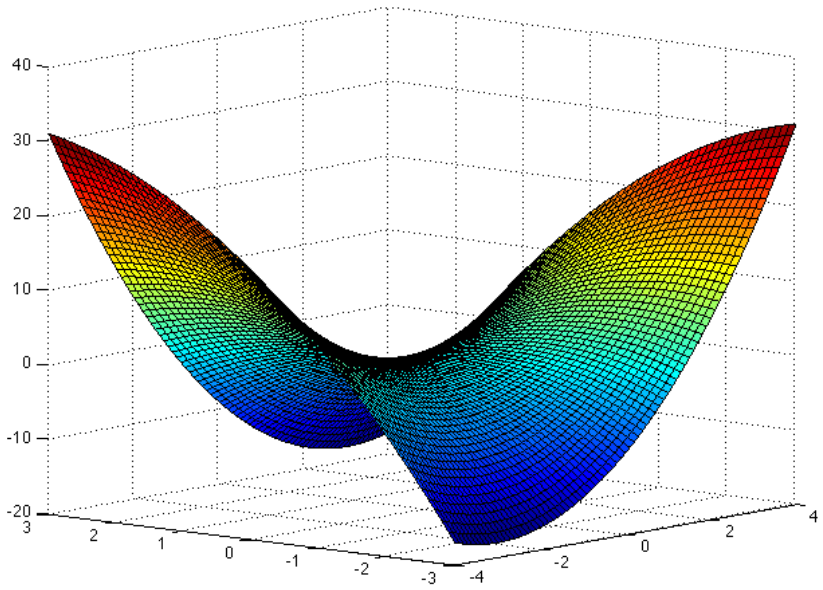
```
>> Z = X.^2 - 2*(X.*Y) - Y.^2 + 2;
```

Please understand why **Z** is a 2D matrix of the same dimensions as **X** or **Y**.
Now we are ready to plot. The simplest way of getting a surface plot is the MATLAB command

```
>> surf(X,Y,Z)
```

This gives a fixed view with standard colors. There are other palettes of colors available; they are chosen using the colormap command. A good all-purpose choice is `colormap(jet)`, which is also good for contour plots. Other color schemes are: `hsv,`hot, cool, pink, gray, bone, copper, prism, and flag. One can choose a different view point

```
>> view(-50, 10)
```



It is also possible to rotate the plot in 3D to get different views. In order to do this, issue the command `rotate3d` on before doing any plotting. Now, once you plot the surface, you can rotate it by dragging with the mouse: try it. You can also choose how your axes are drawn. Estimating the values of the function over the grid of `x` and `y` values shows that its values lie roughly between -15 and 40, so we set up our axes as follows:

```
>> axis([-4 4 -3 3 -15 40]).
```

Note that we don't need commas, but we have to use brackets and parentheses: ([ ]). This is because Matlab's **axis** function takes a 1x6 vector of values and interprets the 1st two values as max, min of x-axis, 2nd two as max, min of y-axis, 3rd two as max,min of z-axis.

Advice on Exporting/Saving a figure

The process of getting a figure out of MATLAB and into another program varies across each platform (Linux, Mac, Windows). If you would like to save a figure for later use in a different program, here is how to save it. Once you have plotted something, go to the figure window, then go to File $->$ Save As... . You should be able to pick a location to save the file. By default, MATLAB should try to save your figure as a .fig. This is a native MATLAB format and will be able to be opened by MATLAB ONLY!!! You can change the format of the image file in the save window to whatever you would like (such as .jpg, .png). Then type the name of the file you want to save. If you decide to use MATLAB for more than an occasional picture or two, it is recommended to save two copies of a figure. One in native MATLAB .fig format and another as whatever type you prefer (such as JPEG). The reason for this is sometime it may take you an hour or so to go through all of the work to obtain a nice plot. If you save the plot as a JPEG, then decide later on your wish to change the plot (for example, maybe your title was Plot of two functions and you decided it should be Plot of 2 Functions) then you CANNOT change the JPEG file easily. You can, however, open up a figure window in MATLAB, tell it to open your .fig file, then change the title and re-save the JPEG with the new title.

4. Movies with MATLAB

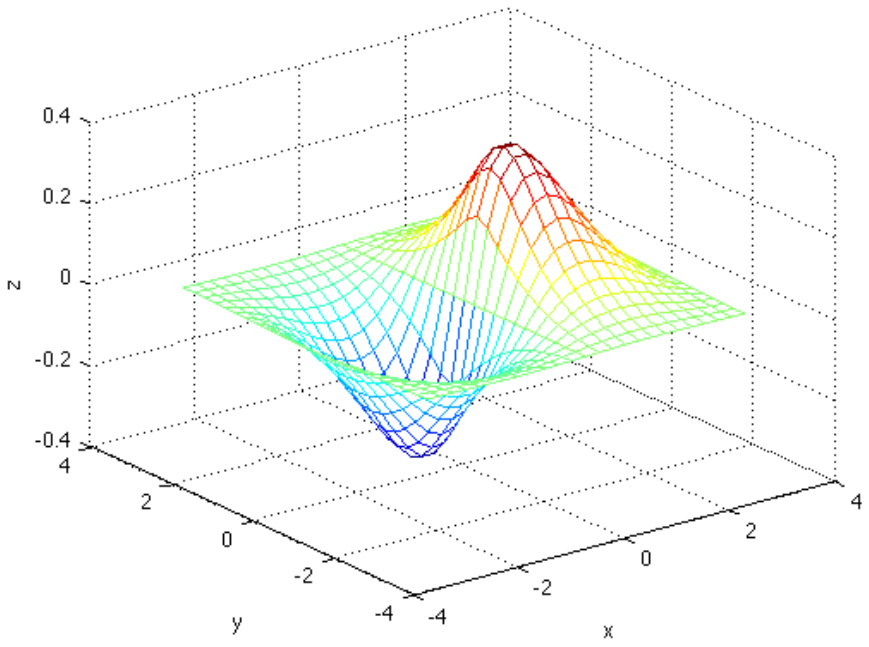
One nice feature if MATLAB is the possibility of creating animations and exporting them as movies files. Below is an example of such animation.
Let's say we want to animate the surface

$$z = 1.8^{-1.5\sqrt{x^2+y^2}} \cos(0.5y)\sin(x)$$

which can be obtained with the `mesh` command:

```
x=-3:0.25:3;
y=-3:0.25:3;
[X,Y]=meshgrid(x,y);

Z=1.8.^(-1.5*sqrt(X.^2+Y.^2)).*cos(0.5*Y).*sin(X);
mesh(X,Y,Z);
xlabel('x'); ylabel('y');
zlabel('z');
```



First we create the frames of the movie.

```
nframes=40;
Wave=moviein(nframes);

t=linspace(1,-1,nframes);

for i=1:nframes
Z=t(i)*1.8.^(-1.5*sqrt(X.^2+Y.^2)).*cos(0.5*Y).*sin(X);
mesh(X,Y,Z);
axis([-3 3 -3 3 -0.5 0.5]);
Wave(:,i)=getframe
end
```

Then one can play the movie at any specified speed (fps)

```
fps=30; % frames per second
>> movie(Wave, fps)
```

or it can be converted to .avi format.

```
>> movie2avi(Wave, 'mymovie.avi')
```

5. Graphics Handles. Modifying Plots

Each part of a MATLAB plot has some set of properties that can be changed to change its appearance. You can create a "handle" for each part of the plot which basically lets you "grab" that part of the plot and change some of its properties. The two commands `get` and `set`

together allow you to find out the current properties of everything in your plot, and change just about anything about them.

5.1 The `gca` and `gcf` commands.

You can always get the handle for two different parts of the plot: the axis, and the figure. The properties of the figure describe the general characteristics of your entire figure window. The axis has properties that describe the characteristics of your axes.
To list all properties of the figure, you can type

```
>> get(gcf)  % stands for get current figure
```

To list all properties of the axis, you can type

```
>> get(gca)  % stands for get current axis.
```

The figure and the axis each have "children." What this means is there are objects in the plot which are a part of them. Each set of axes is in your figure window is a "child" of gcf. (If you have only one set of axes, the child of `gcf` is automatically `gca` .) Each thing plotted on a set of axes (gca) is a child of that gca. One of the properties that gca and gcf have is 'Children', which gives you a handle to its children so you can modify things like lines you've plotted, too.

5.2 The `gco` command

The other useful handle you can get is gco (get current object.) If you click on something in the matlab window (a particular piece of text, a particular line you've plotted, etc), and then type get(gco), you will get the properties of the thing you have just clicked on.

5.3 Syntax and Examples

To start with a pointer to the thing you want to do, you can whichever is appropriate of

```
>> h = get(gco)
>> h = get(gca)
>> h = get(gcf)
```

For example, if I want to get a handle for the xlabel of my plot, click on the text there, and then type

```
>> h = get(gco)
```

If I want to get a handle for the line I've plotted on my set of axes (remember, this is the child of gca), I can click on the line and do the same thing I did with the xlabel, *or* I can type

```
>> h = get(gca,'Children')
```

Once you have a handle, you can list all the properties of that handle with

```
>> get(h)
```

or you can list a specific property, if you know which one to look for, with

```
>> get(h,'PropertyName').
```

One useful property, for a plot of some data, is XData. If you have a handle for the line part of the plot (from get(gca,'Children'), you can use

```
>> x = get(h,'XData');
>> y = get(h,'YData');
```

to get back the data you've plotted and modify it somehow.

To reset a property to a different value, you use the set command. If you simply type

```
>> set(h,'PropertyName')
```

you will get back a list of the options you can set that property to. For something like 'XData', you will be told that there is no fixed set of property values (obviously, you can use anything for the x data of a plot). For something like 'XScale', which tells you the scale of the x axis, if you type

```
>> set(gca,'XScale')
```

you will get back [linear — log], which indicates that the two options are linear or log, and it is currently linear.

To change the value of a property, and actually modify the plot, you would do

```
>> set(handle,'PropertyName',value)
```

In the example above, to change the scale of the current x axis to log, you would type

```
>> set(gca,'XScale','log')
```

You can use these features to make just about any modification you might want to do to a graph (including all the ones in the problem set...), if you find the right property to change. The basic approach is to get a list of the properties until you find the right one, get a handle to the object that has that property, and then set the property to a new value using the handle.

For more details about the Graphics Handles, see the article by Roger A. Green *"Getting a handle on MATLAB graphics"*.

6. GUIs in MATLAB

On exciting addition to MATLAB has been the ability to easily create Graphical User Interfaces (GUIs). These are programs that allow the user to interact with MATLAB codes in a customized, interactive fashion.

The starting point in designing a GUI is the command

```
>> guide
```

which allows you to build your interface, the GUI components etc. Unfortunately, this lab cannot cover the entire process of building a GUI from scratch. Instead, we will analyze in class the anatomy of a GUI based on a simple example.

An excellent resource for learning how to build GUIs in Matlab is " Learning to Program with MATLAB. Building GUI Tools." by C. Lent, Wiley, 2013.

A FINAL THOUGHT:

Though Matlab is a very popular computer algebra software environment used in both academia and industry, it is even more important to be familiar with some generic capabilities of such software. It is better to be able to state that one knows how to use this type of software to solve problems than to state familiarity with a particular implementation such as Matlab. That being said, many in both academia and industry have found Matlab to be an efficient way to prototype an idea - first with command-line commands, then scripts/functions. After some familiarity, one can quickly work out "proof of principle" ideas in Matlab. Afterwards, one may then decide to make the Matlab code more robust (check many possible error scenarios, user friendliness, etc.) or, as is sometimes decided, using this prototype to fully develop a solution in a more traditional, compiled type computer language/environment like C++, Java, etc.