8 PLOTS

Contents

```
% make sure the workspace is clear
if ~exist('____code____','var') ; clear ; end
% reduce needless whitespace
format compact
% reduce irritations (pausing and buffering)
more off
% start a diary (in the actual lecture)
%diary lectureN.txt
```

LESSON SUMMARY

This lesson is all about making plots of various kinds. That is an important thing for engineers to be able to do. It is used for understanding systems, representing data on systems, publicizing results, and advertising them.

1) The first main part of this lesson discusses how to easily plot functions whose analytical form is known. Covered functions include:

- `ezplot(FUN1V,[START END])` is an easy way to plot a function of one variable. Here `FUN1V` should be a quoted string describing the function to plot. `START` and `END` are the start and end values of the variable range that is plotted.
- `ezplot(FUN2V,[XMIN XMAX YMIN YMAX])` is an easy way to plot an implicitly given function (for example $x^2 + y^2 - 4 = 0$ for a circle of radius 2). Here `FUN2V` should be a quoted string describing the function of *both* the independent variable *and* the dependent one that must be *zero* (like `'x^2+y^2-4'` in the example). `XMIN`, `XMAX`, `YMIN`, and `YMAX` are the plot limits of the independent, respectively dependent variable.
- `ezplot(FUN1,FUN2,[START END])` is an easy way to plot a parametrically given curve of the form $x = f_1$ and $y = f_2$, where f_1 and f_2 are functions of a single parameter (often time). Here `FUN1` and `FUN2` should be quoted strings describing f_1, respectively f_2. `START` and `END` are the start and end values of the parameter range that is plotted.

- `ezplot3(FUN1,FUN2,FUN3,[START END])` plots a parametrically given curve in three dimensions instead of two.
- `ezmesh(FUN2V,[XMIN XMAX YMIN YMAX],POINTS)` is an easy way to plot a function of two variables as a surface in a three-dimensional graph. Here `FUN2V` should be a quoted string describing the function of two variables to plot. `XMIN`, `XMAX`, `YMIN`, and `YMAX` are the plot limits of the two variables. `POINTS` is the number of plot points to use in each direction. If `",POINTS"` is left away, POINTS is 60. For better plot quality, increase `POINTS` at your own peril.
- `ezsurf` works just like `ezmesh`, and generates a better looking surface. However, `ezsurf` is also much less user- and printer-friendly. Use at your own peril.
- `ezcontour(FUN2V,[XMIN XMAX YMIN YMAX],POINTS)` is an easy way to plot a function of two variables as contour lines in the plane of the two variables. Arguments are like those of `ezmesh`.
- `ezmeshc` and `ezsurfc` create combinations of surface and contour plots. They are used in the same way as the individual functions.

Note that if the functions to plot are a bit too complex to be easily described in a single string, you can instead provide the `ez...` functions a handle to the desired function, which can then be defined some other way.

2) The second main part of this lesson is using so-called "log-log" plots. In these two-dimensional plots, the plotted position along the two axes does not vary according to the value of the variables, but according to the value of their logarithm.

- Log-log plots are useful for dealing with variables that vary by orders of magnitude.
- Log-log plots are also useful for recognizing power relationships between variables, like in $y = Cx^p$. If the log-log plot looks like a straight line, there is a power relationship. You can get the parameters of the power relationship using the trick of taking the logarithms of the variables as new variables and then performing linear regression on that, like in the lesson on interpolation. In particular, if `xVals` and `yVals` are the given values of x and y, use:

```
Coefs=polyfit(log(xVals),log(yVals),1);
p=Coefs(1); C=exp(Coefs(2));
```

3) The third main part of this lesson is on how to plot a function of two variables, let's call it $f(x, y)$ here. Unlike in the first main part, now it is assumed that the values of f are measured or computed on a grid, rather than analytically known. Relevant functions include:

- `[xGrid yGrid] = meshgrid(xValues,yValues)` generates the x and y values of all the grid points of a rectangular mesh. Here `xValues` should be a one-dimensional array of n increasing x-values and `yValues` a one-dimensional array of m increasing y-values. The created `xGrid` and `yGrid` are two-dimensional arrays of size $m \times n$ describing all the x and y values of the complete two-dimensional grid. In particular `xGrid(i,j)` equals `xValues(j)` and `yGrid(i,j)` equals `yValues(i)`.

- `fGrid = SimplePoisson(xValues,yValues,forcing)` generates function values `fGrid` on the grid points above for typical simple real-life problems in physics and engineering. Here `SimplePoisson` is a function provided by the instructor. One-dimensional arrays `xValues` and `yValues` are as above. Array `forcing` must be an $m \times n$ two-dimensional array that describes the physics of what forces the function f to be nontrivial. For the interior grid points $2 \leq i \leq m-1$ and $2 \leq j \leq n-1$ the forcing is some "force" that the instructor will specify. For the four boundaries with $i=1$, $i=m$, $j=1$, respectively $j=n$, the forcing is simply the value of function f at the boundaries, which again the instructor will specify.

- `stem3(xGrid,yGrid,fGrid)` will show the individual function values on the grid above as "spikes" or "stems", starting from the corresponding grid point in the x, y-plane.

- `mesh(xGrid,yGrid,fGrid)` or `surf(xGrid,yGrid,fGrid)` will plot the function as a surface, similar to what `ezmesh` or `ezsurf` did for analytical functions.

- `contour(xGrid,yGrid,fGrid)` will plot contour lines of the function in the x, y-plane, similar to what `ezcontour` did for analytical functions.

4) The final part of this lesson shows how you can use polar, instead of Cartesian coordinates. There is one additional function needed to deal with polar coordinates:

- `[xGrid yGrid] = pol2cart(thetaGrid,rGrid)` takes the polar coordinates of your grid and converts them to Cartesian. That then allows you to use the Cartesian functions like `mesh`, `surf`, and `contour` to plot your function. Note the order θ, r in `pol2cart`.

Key areas of the online book

Before the lecture, in the online book do:

- 9.2 2D data plots I: skip example 9.2.1 at the end.
- 9.8 2D data plots II: all.
- 9.9 3D line plots: skip final question 3 in PA 9.9.1.

- 9.10 Rectangular data grids: skip ndgrid, PA 9.10.6 question 3, PA 9.10.8 all, and everything behind the surface plot.
- 9.12 3D mesh and surface graphs: all.
- 18.2 Curve fitting - Least squares: PA 18.2.4. Try to read through the material leading up to PA 18.2.4. If it leaves you clueless, as I think it will, just solve PA 18.2.4 by trial and error. Also complete whatever material in this section from lesson3 you might not have done yet.
- 21.5 Plots: all.

PLOTTING ANALYTICAL FUNCTIONS

In many previous lessons we have seen how you can use the Matlab `plot` function to plot functions given a set of function values. However, Matlab also provides a series of `ez...` functions (where "ez" stands for "easy") that can make plots of functions given as an analytical expression.

The analytical function can usually most simply be described in a quoted character string. However, you can instead give the `ez...` functions a handle to the function to plot if you want.

Here we will look at a few important examples.

Plotting a function of a single variable simply

Function `ezplot` will easily plot analytical functions of a single variable. For example, the *absolute* simplest way to plot, say, $\sin(t)$ versus t in Matlab is:

```
% use ezplot to plot sin(t) agains t
ezplot('sin(t)')
```

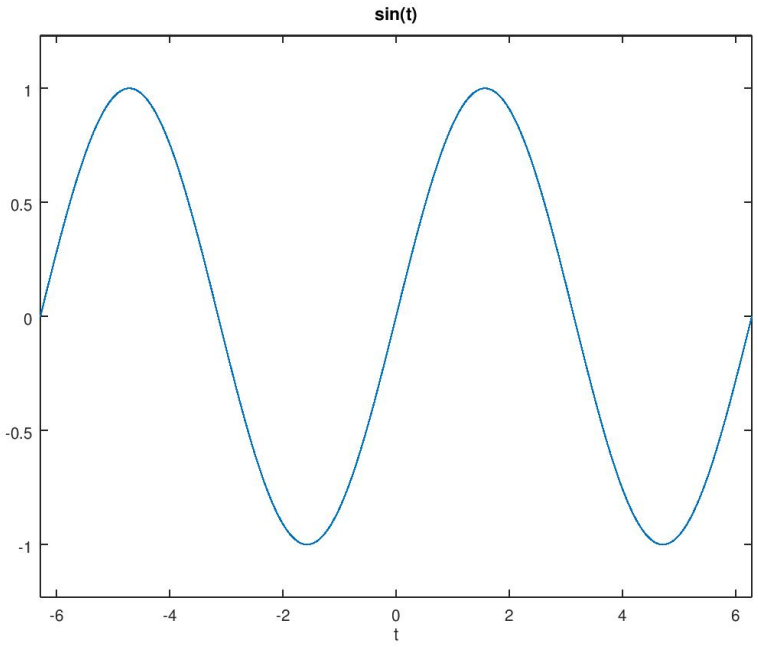
Note that `ezplot` did all the work for us, including title and axis labels!

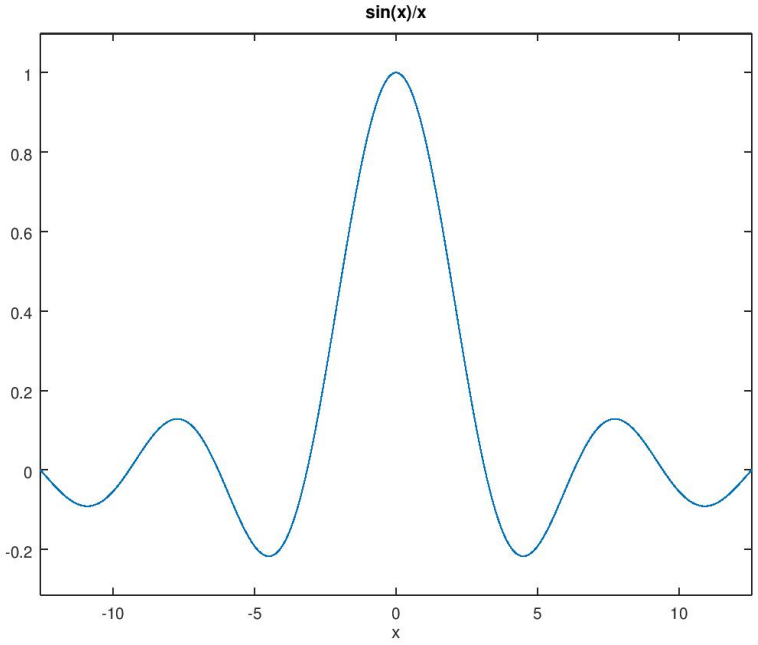
But in real life, you must specify the limits

By default, `ezplot` takes the independent variable to go from -2π to 2π. Of course, this is very unlikely to be what you really want. So normally you must specify the range of the dependent variable to plot as a second argument. For example, if you want to plot $\sin(x)/x$ from -4π to 4π, you can do it as:

```
% plot sin(x)/x with specified limits on x
ezplot('sin(x)/x',[-4*pi 4*pi])
```

Function $\sin(x)/x$ is known as the "sinc" function. As noted in the lesson on symbolic math, its antiderivative is "Si", the "sine integral".


sin(t)
1
0.5
0
-0.5
-1
-6
-4
-2
0
2
4
6
t


sin(x)/x
1
0.8
0.6
0.4
0.2
0
-0.2
-10
-5
0
5
10
x

Plotting an implicit function

Often, you do not know the dependent variable as a given function of the independent one, but you know only some equation relating the two variables. That is called an implicitly given function. For example, unit hypercircles are given by the implicit equation

$$x^{2n} + y^{2n} = 1$$

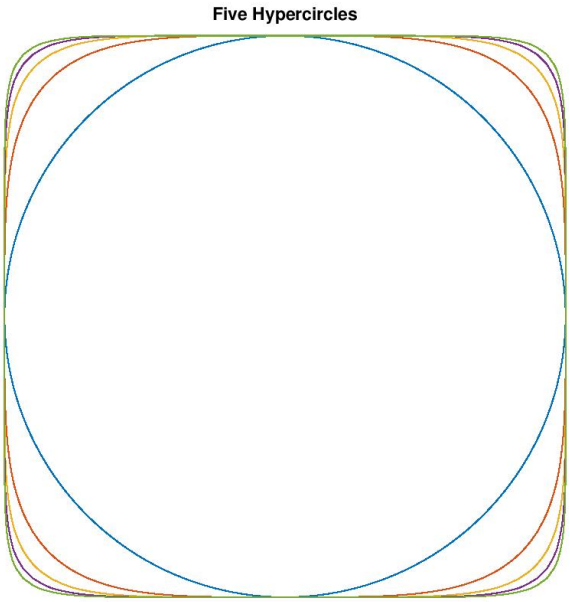
where n is a positive integer. You can easily see that if $n = 1$, the equation above is just the equation for a normal unit circle around the origin. You do not have y explicitly as a given function of x here, so this is an implicitly given function.

To plot implicitly given functions with `ezplot`, first take all terms to the left hand side. For the above equation, we need to take the 1 in the right hand side to the left hand side. Now give the *left hand side,* a function of both x and y, to `ezplot`. Ezplot will then draw the curve on which this function is zero. You will also need to specify the plot limits on x and y as a vector `[xMin xMax yMin yMax]`:

```
% plot the first (hyper)circle, for n=1, with ezplot
ezplot('x^2+y^2-1',[-1 1 -1 1])

% make sure the next hypercircles go in the same plot
hold on

% add hypercircle for n=2, 3, 4, and 5 to the plot
ezplot('x^4+y^4-1',[-1 1 -1 1])
ezplot('x^6+y^6-1',[-1 1 -1 1])
ezplot('x^8+y^8-1',[-1 1 -1 1])
ezplot('x^10+y^10-1',[-1 1 -1 1])

% change the title
title('Five Hypercircles')
% equally scaled axes tightly around the curves, hidden
axis('equal','tight','off')

% allow the next plot to erase the current one again
hold off
```

Plotting a parametric curve

Sometimes a function is described in terms of some additional parameter, like, say, time, which is not plotted. That is called a parametrically given curve.



For example, consider a bicycle wheel of unit radius that rolls with unit angular velocity. The point on the tire thread of this wheel that is at the ground at time zero moves according along the parametric equations

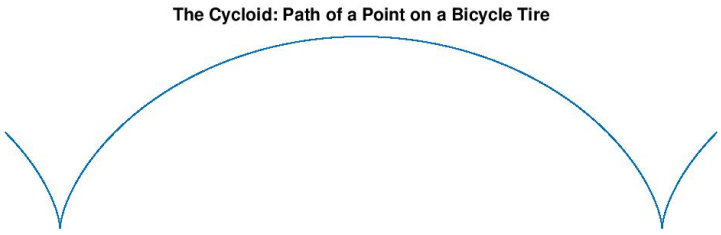
$$x = t - \sin(t) \qquad y = 1 - \cos(t)$$

The curve in the x, y-plane that this point traces out is called the "cycloid". It can be easily drawn using `ezplot`. Just specify both functions. And, of course, the range of the parameter to plot.

```
% draw the cycloid over a bit more than a full revolution
ezplot('t-sin(t)','1-cos(t)',[-.5*pi 2.5*pi])

% equally scaled axes tightly around the curve, hidden
axis('equal','tight','off')
% change the title
title('The Cycloid: Path of a Point on a Bicycle Tire')
```

Plotting a parametric curve in three dimensions

The previous subsection used `ezplot` to draw a parametrically given curve in a plane. Parametrically given curves in three dimensions also occur a lot. They



can be drawn using `ezplot3`.

For example, the parametric equations

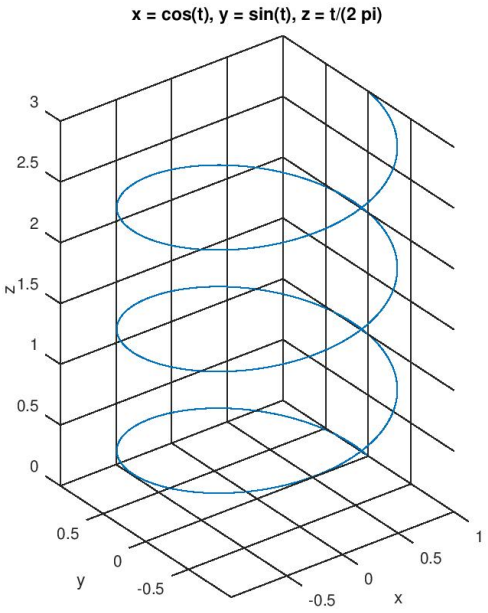
$$x = \cos(t) \qquad y = \sin(t) \qquad z = \frac{t}{2\pi}$$

describe a spiral in three dimensional x, y, z space. The spiral can be plotted as:

```
% plot the spiral in three dimensions for 3 turns
ezplot3('cos(t)','sin(t)','t/(2*pi)',[0 6*pi])

% equally scaled axes tightly around the curve, hidden
axis('equal','tight')
```

Plotting a function of two variables as a mesh surface

Suppose that you want to plot a function of two variables, call it $f(x, y)$ for now. Then the most straightforward way is to make a three-dimensional plot in which two coordinates are the independent variables, x and y here, and the third coordinate is the function value, f here. You will then get a surface, in this



three-dimensional space, in which the height of the surface above the x, y-plane at any x and y gives the function value at that x and y.

In Matlab you can plot this surface easily using `ezmesh`; that will show the surface as a "mesh" of little quadrilaterals.

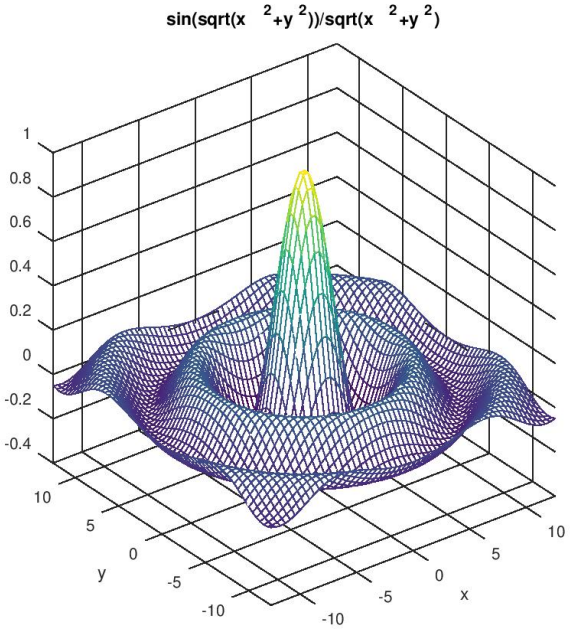
For example, let's plot a generalization of our previous example, "sinc" function $\sin(x)/x$, to two dimensions:

$$f = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}}$$

We will use a plot range from -4π to 4π for both x and y.

```
% plot the function using ezmesh
ezmesh('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)', ...
       [-4*pi 4*pi -4*pi 4*pi])

% use axes of the same length to avoid distortion
axis('square')
```

To understand the function better, in an interactive session grab hold of the graph with the mouse and move it around! (You may need to click the rotate button first.)



Plotting a function of two variables as a true surface

The surface created by `ezMesh` would definitely look better if the quadrilaterals were filled up. Using `ezsurf` instead of `ezmesh` will do that for you.

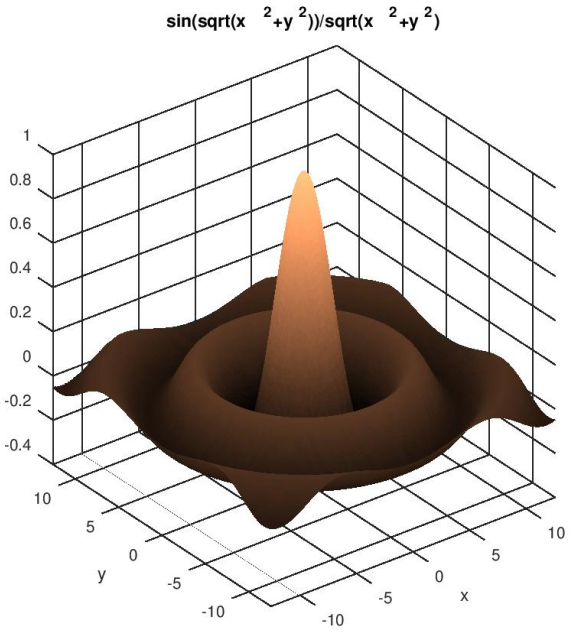
But there are some problems with that. For one thing, it takes Matlab a lot of time to do it. In particular, it can be very slow to publish.

In addition, if you print it out, all these colored pixels will take a lot of expensive toner. And if your laser printer is not in the best state of maintenance, it may also make a mess of these solid areas.

Anyway, here it goes:

```
% use ezsurf
ezsurf('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)', ...
       [-4*pi 4*pi -4*pi 4*pi],120)

% use axes of the same length to avoid distortion
axis('square')

% needed for a good looking surface
shading interp
```

```
% change color to copper tones
colormap copper
```



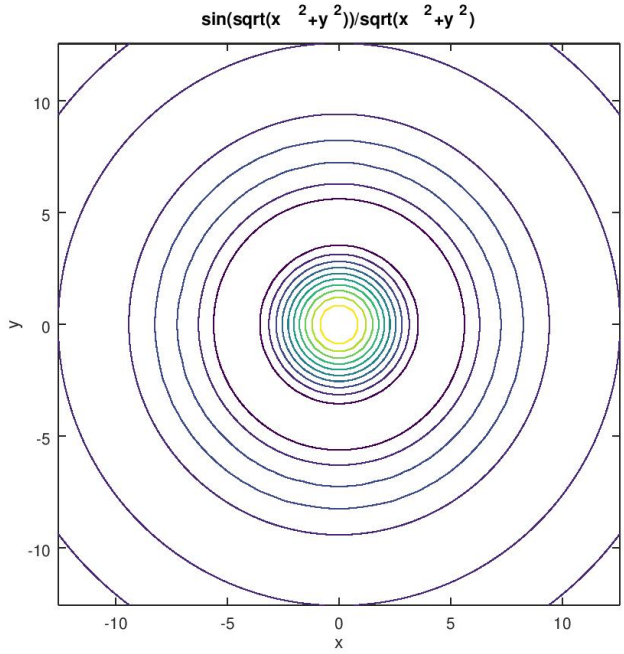
The final 120 in the `ezsurf` command tells Matlab to subdivide the x and y ranges into 120 intervals. The default is 60, but 120 makes the peak look better. It does take more time to publish (10 seconds more on my Windows 7 PC).

Plotting contour lines for a function of two variables

Of course, the previous "three dimensional" graphs were really fake. In reality, they were pictures on the two dimensional surface of a piece of paper or computer screen. They were not really three-dimensional.
In many cases it is a better idea to simply show the function in the two-dimensional x, y-plane of the two independent variables. You can do that by drawing lines on which the function has a constant value. Such lines of constant function value are called "contour lines." On weather maps, thermo graphs, and such, if the function is pressure, contour lines are also called "isobars"; if the function is temperature, "isotherms". In general, "iso" is Greek for "equal".
To plot contour lines easily, use `ezcontour`:

```
% use ezcontour on our beloved sinc function.
ezcontour('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)', ...
```

```
        [-4*pi 4*pi -4*pi 4*pi])

% use axes of the same length to avoid distortion
axis('square')
```



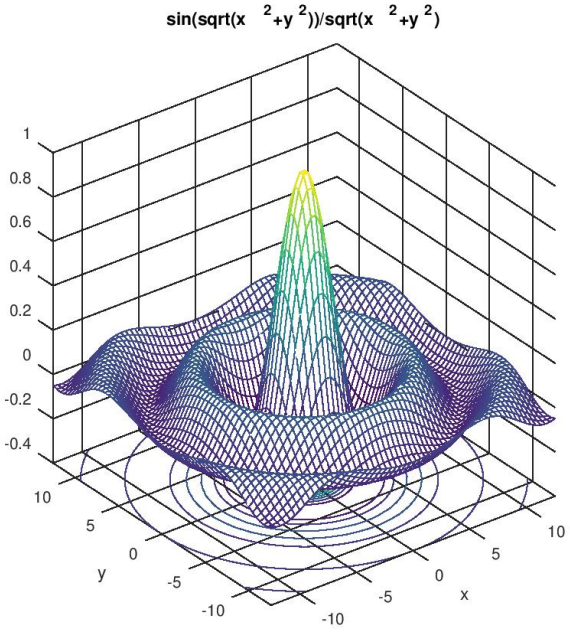
There does not seem to be a way to influence *what* values of the function are plotted. For that, you would need function `contour`, as shown in the third part.

Plotting both mesh and contour lines

For a combination of a surface and contour lines, use `ezmeshc` or `ezsurfc`:

```
% use ezmeshc
ezmeshc('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)', ...
        [-4*pi 4*pi -4*pi 4*pi])

% use axes of the same length to avoid distortion
axis('square')
```

To see better how contour lines relate to the function surface, grab again hold of the figure and move it around.



Warning: do not use on interpolated data

Unlike, say, `plot`, the `ez...` functions above do not handle sets of discrete function values. Do not try to get around this by interpolating the function values and giving the `ez...` functions the function handle of the interpolated function. Interpolated functions are only a guess at the true function. So, if you show people interpolated data, you should *also* show them the discrete functions values that you actually measured or computed, as symbols. Anything else is grossly misleading and completely unacceptable.

LOG-LOG PLOTS

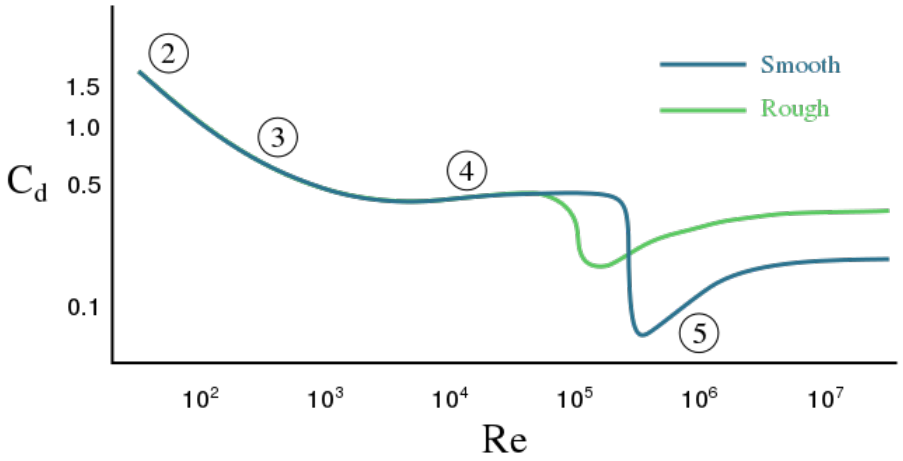
In "log-log" plots, the distances along the two axes do not vary according to the values of the variables themselves, but according to the values of their logarithms. In Matlab you can make log-log plots by using function `loglog` instead of `plot`.

Application to strongly varying data

Log-log plots are very useful for dealing with variables that vary by orders of magnitude. The logarithm of a variable varies only slowly when the variable grows by a large factor. So you get a decent looking graph. If in the lesson

on ODE, you looked up the drag coefficient C_d from a sphere on Wikipedia as suggested, you saw an example.

From Wikipedia:



If the horizontal axis would not have been logarithmic, the range from 10^2 to 10^3 would have been far smaller than a *single pixel* compared to the range from 10^6 to 10^7. With a logarithmic axis however, you can see all orders of magnitude of the Reynolds number Re properly.

Note also that on logarithmic axes, the tick marks are no longer equally spaced. That is because the tick marks still indicate values of the variables themselves, not those of their logarithms.

You might wonder whether the vertical, C_d axis would really have to be logarithmic. But if the plot had included small Reynolds numbers, you would have wondered no more; the drag coefficient becomes infinite when the Reynold number goes to zero. In the log-log plot, the curve becomes an oblique straight line growing to infinity going towards the left. You can see that much better at the following two, equivalent, links:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4648403/figure/f1/
https://www.nature.com/articles/srep12304/figures/1

Application to power relationships

Log-log plots are also very useful for recognizing "power relationships" between variables; relationships in which the dependent variable is proportional to some power of the independent variable. Assuming for now that the independent variable is called x and the dependent one y, a power relationship looks like:

$$y = Cx^p$$

where the power p is some constant and the constant of proportionality C is some other constant.
Do power relationships occur a lot in science and engineering? Yes! Basically because in physical relationships the units must match up. When you learn dimensional analysis in fluid mechanics you will better understand why. For now just note that since the radius of a circle has units of length, and its area units of length squared, the area of a circle *must* be proportional to the radius to the power 2. That is a power relationship. In this elementary example, the power p is 2 and the constant of proportionality C is π: $A = \pi r^2$. (The drag coefficient of a sphere at small Reynolds numbers is also a power relationship, with p=−1 and C=24.)
So how do log-log plots help in recognizing these relationships? Well, simple: if the log-log plot looks like a straight line, there is a power relationship.
To show this, we will make up an arbitrary example, in which we take y to be equal to $3x^{3/2}$. We will also add some random "measurement" errors to make it look "real", and then plot the result in a log-log graph.

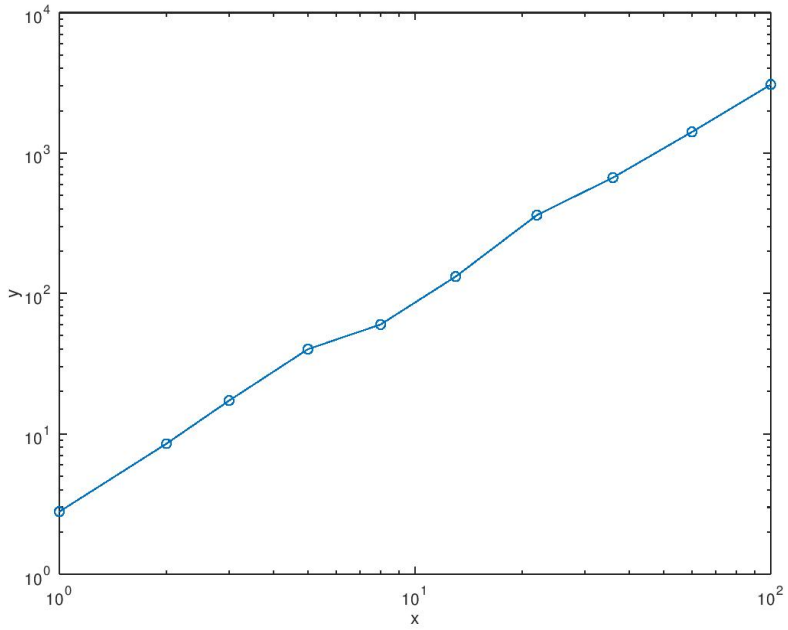
```
% choose supposed values of x of the "measurements"
xVals=[1 2 3 5 8 13 22 36 60 100];

% create the supposed "measured" values of y
yVals=3*xVals.^1.5;

% add about 10% experimental error to make it look "real"
%rng('default') % Matlab version
randn('seed',9) % Octave version
yVals=yVals.*(1+0.10*randn(size(xVals)));

% plot on a log-log scale
loglog(xVals,yVals,'o-');
xlabel('x')
ylabel('y')
```

Note that indeed this pretty much looks like a straight line, allowing for the random errors.

Finding the approximate power relationship

For the made up example of the previous problem, suppose these measurements were all you knew about the variables in question. Then from the fact that the log-log graph is well described as a straight line, you can reasonably conclude that there is some power relationship between the variables. But how to get the power p and the constant of proportionality C in that relationship?

Well, there is a trick. First note what you get if you take a logarithm of both sides of the power relationship:



$$y = Cx^p \qquad \Longrightarrow \qquad \ln(y) = p\ln(x) + \ln(C)$$

So if you take $\ln(x)$ to be a *new* independent variable X, and similarly $\ln(y)$ to be a *new* dependent variable Y, you get

$$Y = C_1 X + C_2 \qquad C_1 \equiv p \quad C_2 \equiv \ln(C)$$

Note first that according to the above, the relationship between Y and X is linear. That confirms mathematically that a power relationship becomes a straight line when plotted on a log-log scale.

And then remember from the lesson on interpolation that for a linear relationship like the one between Y and X, you can easily get the coefficients C_1 and C_2 using polyfit. And C_1 is the power p and C_2 is the logarithm of the constant of proportionality C, so you can find these too.

Let's try that for our example (remember that Matlab uses `log` for ln:

```
% the coefficients of the line between log(x) and log(y)
Coefs=polyfit(log(xVals),log(yVals),1);

% find p and C
p=Coefs(1)
C=exp(Coefs(2))
```

```
p =   1.5079
C =   3.0183
```

Note that despite the errors, the found values are close to the exact values $p = 1.5$ and $C = 3$.

We might also want to plot the approximate power relation versus the measurements to see graphically how well it stands up.

And we might guess that the power should really be 1.5 exactly. (There is normally no way to guess what the coefficient C really is.) So we could try plotting that too.

```
% evaluate the found approximate power relationship
yPower=C*xVals.^p;
yPower2=C*xVals.^1.5;

% plot it and the measured points
loglog(xVals,yVals,'o',...
       xVals,yPower2,'-b',xVals,yPower,'--b');
title('Verification of the Power Relationship')
legend('Measured',...
       [num2str(C,3), ' x^{', num2str(p,3), '}'],...
       [num2str(C,3), ' x^{1.5}'])
legend('location','southeast')
```

Additional remarks

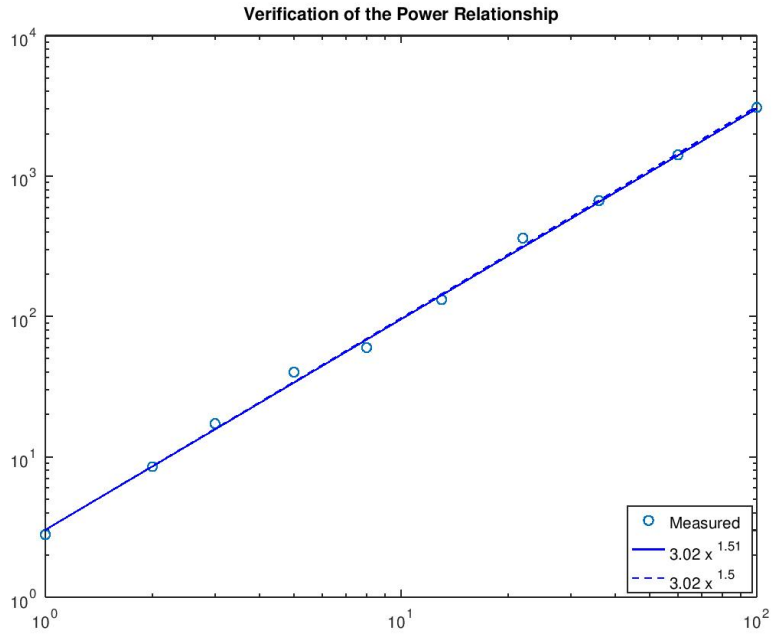
Be sure to make appropriate name changes if your variables are not called x and y.

Because of the taking of the logarithms, function `polyval` does not make the average *absolute* errors in y as small as possible, but the *relative* ones. However, that is typically exactly what you want in these relationships.

Matlab also has functions `semilogx` and `semilogy`, in which only the horizontal coordinate, respectively only the vertical one, is logarithmically spaced. Use these functions if only one of your variables changes greatly in magnitude.

Function `semilogx` can also be used to check for a logarithmic dependence of y on x, and `semilogy` for an exponential one.

In particular, in the lesson on interpolation we could have checked for exponential behavior of the temperature by plotting it using `semilogy`. And the trick for getting the constants of the exponential relationship was already explained in the lesson; we just skipped it. So look it up there if you ever need it.



PLOTTING THREE DIMENSIONAL (3D) DATA

In the first main part of this lesson we plotted analytical functions of two variables using Matlab functions `ezmesh`, `ezsurf`, and `ezcontour`.

But suppose you knew the function only at discrete points? Well, if that happened for a function of one variable, you would use `plot` instead of `ezplot`. Similarly, for two variables, use `mesh` instead of `ezmesh`, `surf` instead of `ezsurf`, and `contour` instead of `ezcontour`.

The example problem

Recall that in a homework, you looked at the sag of power lines. You solved a system of equations to get the height of the power line, call it h, at a number of points. In that example, you could plot the height of the power line versus the horizontal position coordinate x in a simple two-dimensional plot.

But what if you want to look at the sag of a drum membrane under its own weight? Then there are *two* horizontal position coordinates, call them x and y. For every point (x, y) in the horizontal plane, there is a corresponding height of the membrane at that location. Now a three-dimensional plot is needed to see the sag. Mathematically the height h is some function $h(x, y)$ of x and y. We will now address ways that you can plot such a function.

Defining a grid

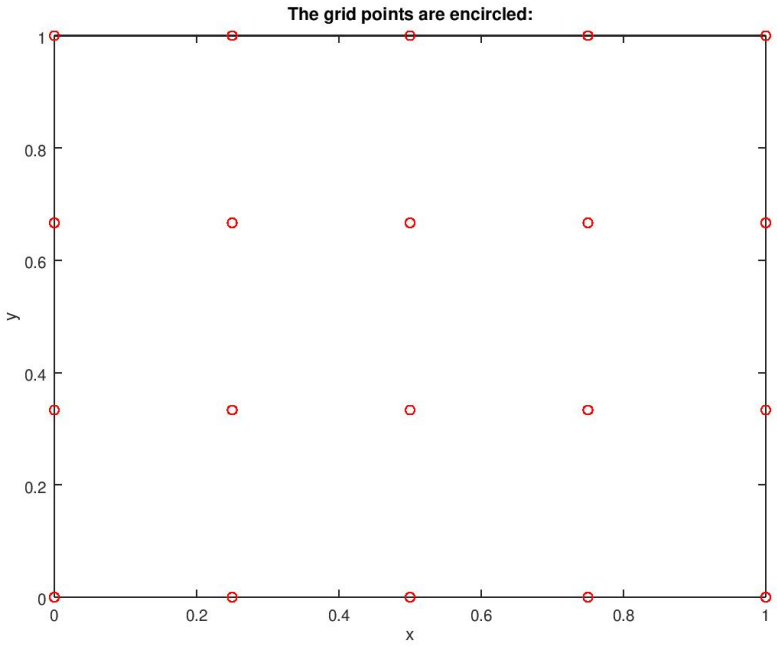
For simplicity, we will assume that the drum membrane is square, with sides of unit length. So the relevant x and y values form a unit square. Unfortunately, a square contains infinitely many points, and that is too many. We must restrict the number of points to a finite number. We can do so by selecting a finite number, call it n, of x-values with `linspace`, and similarly a finite number, call it m, of y-values. Then we restrict the points in the square to only the $m \times n$ points that have those x and y values. Such a set of points is called a "mesh" or a "grid".

Function `meshgrid` gives the x-and y values of the grid points, as $m \times n$ arrays.

```
% number of x-values we will use for now
n=5
% get the n x-values themselves from linspace
xValues=linspace(0,1,n)
% number of y-values we will use for now
m=4
% get the m y-values themselves from linspace
yValues=linspace(0,1,m)

% create a [m n] grid of all combinations of these values
[xGrid yGrid]=meshgrid(xValues,yValues)

% plot the grid
plot(xGrid,yGrid,'or')
xlabel('x')
ylabel('y')
title('The grid points are encircled:')
```

```
n =  5
xValues =
    0.00000    0.25000    0.50000    0.75000    1.00000
m =  4
yValues =
    0.00000    0.33333    0.66667    1.00000
xGrid =
    0.00000    0.25000    0.50000    0.75000    1.00000
    0.00000    0.25000    0.50000    0.75000    1.00000
    0.00000    0.25000    0.50000    0.75000    1.00000
    0.00000    0.25000    0.50000    0.75000    1.00000
yGrid =
    0.00000    0.00000    0.00000    0.00000    0.00000
    0.33333    0.33333    0.33333    0.33333    0.33333
    0.66667    0.66667    0.66667    0.66667    0.66667
```

```
    1.00000    1.00000    1.00000    1.00000    1.00000
```



Note that in the graph, y increases going upwards, as usual. However, in the way the arrays are printed, y increases going downwards.

Finding the height of the membrane using SimplePoisson

Unfortunately, finding the height of the membrane requires the solution of what is called a "Partial Differential Equation". And solving such equations is far, far, beyond the scope of this class. That is true even for one of the simplest of such equations, the so-called "Poisson" equation, that governs the height of the membrance. So I have created a function, `SimplePoisson`, that finds the solution for you. The only thing you need to do is create an array `forcing` with information on the desired solution.

At any *interior* point of the grid, array `forcing` should contain the value of the "force" at the interior point, i.e. whatever wants to make the solution nontrivial. For the membrane, that is the scaled weight of the membrane per unit area. You will always be given the value of the "force" in this class. In particular, for the membrane here we take the "force" to be constant and equal to 2 for simplicity.

At any boundary point, (i=1, i=m, j=1, or j=n), array `forcing` should contain the desired value of the solution at that point. (In a sense, nontrivial boundary

values make the solution nontrivial at the boundary.) You should again be given that. In particular, here we assume that the membrane is attached to the drum at a constant height 1 at all boundaries.

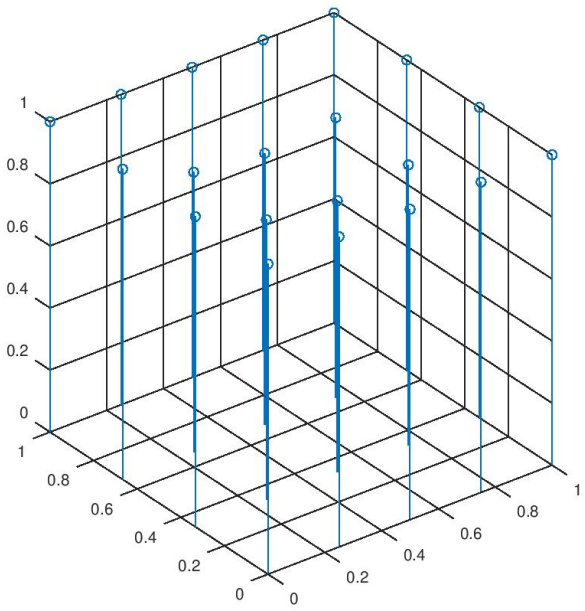
```
% the interior point forcing
force=2

% the attachment height of the membrane
heightBoundary=1

% initialize the forcing array to the interior force
forcing=force*ones(size(xGrid));

% now change it to heightBoundary on the four boundaries
forcing(1,:)=heightBoundary; % boundary y = 0
forcing(m,:)=heightBoundary; % boundary y = 1
forcing(:,1)=heightBoundary; % boundary x = 0
forcing(:,n)=heightBoundary; % boundary x = 1

% let SimplePoisson find the heights at the grid points
height=SimplePoisson(xValues,yValues,forcing)
```

```
force =  2
heightBoundary =  1
n =  5
m =  4
N =  20
solRelerrDueToMatlab =    2.3583e-14
height =
   1.00000   1.00000   1.00000   1.00000   1.00000
   1.00000   0.90248   0.87511   0.90248   1.00000
   1.00000   0.90248   0.87511   0.90248   1.00000
   1.00000   1.00000   1.00000   1.00000   1.00000
```

Let's have a look at the raw data

Matlab function `stem3` will show you the raw data graphically: For each grid point x, y, a vertical line segment is plotted whose length represents the height of the membrane at that grid point.

```
% show the function stems
stem3(xGrid,yGrid,height)

% avoid distortion
axis('square')
```



Try rotating the graph to see more clearly how it looks.

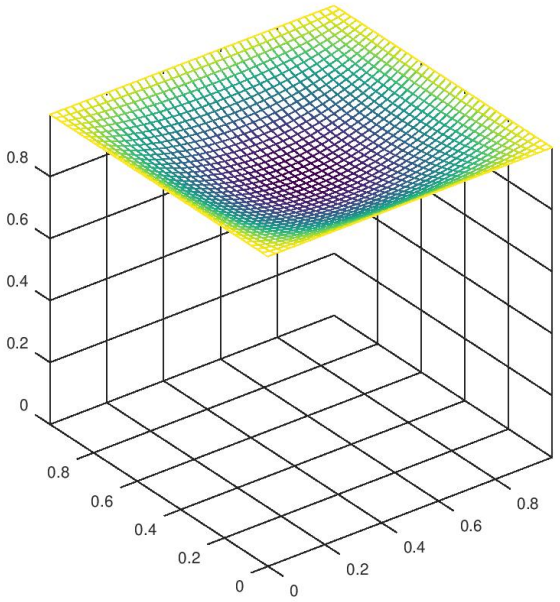
Plot as a surface

Functions `mesh` or `surf` will show our function values as a surface, similar to the way `ezmesh` and `ezsurf` do it for continuously defined functions.

```
% let's have some more points for a better appearance
force=2;
heightBoundary=1;
n=40
xValues=linspace(0,1,n);
m=40
yValues=linspace(0,1,m);
[xGrid yGrid]=meshgrid(xValues,yValues);
forcing=force*ones(size(xGrid));
forcing(1,:)=heightBoundary;
forcing(m,:)=heightBoundary;
forcing(:,1)=heightBoundary;
forcing(:,n)=heightBoundary;
height=SimplePoisson(xValues,yValues,forcing);

% use function mesh to plot
mesh(xGrid,yGrid,height)
```

```
% avoid distortion
axis([0 1 0 1 0 1],'square')
```

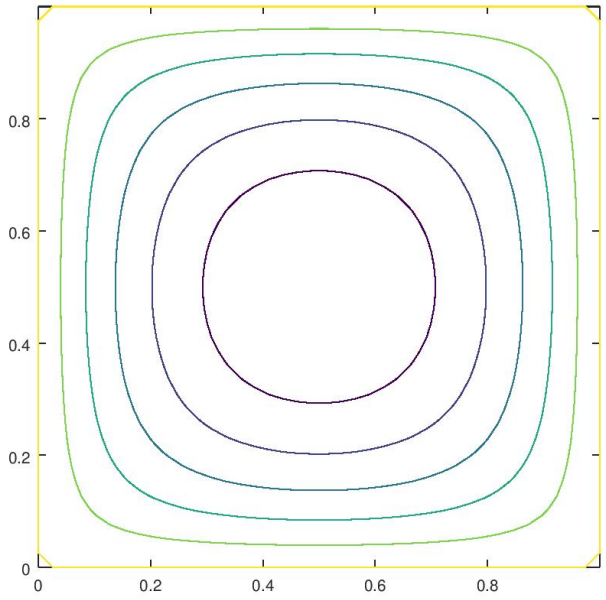
```
n =  40
m =  40
n =  40
m =  40
N =  1600
Using an estimated L1 condition number.
solRelerrDueToMatlab =     2.9006e-12
```



Plot contour lines

Function `contour` will draw contour lines for our function values as a surface, similar to the way `ezcontour` does it for continuously defined functions.
Note that if the membrane get rained upon, the edge of the water surface will be a contour line: it has everywhere the same height. That is why contour lines are sometimes referred to as water lines. (But of course, the weight of the water would add to that of the membrane, and the shape will change.)
We will draw the contour lines corresponding to heights 1, 0.975, 0.95, ...

```
% use function contour to plot the desired heights
contour(xGrid,yGrid,height,[1:-.025:.85]);

% avoid distortion
axis('square')
```



Note that since the 0.85 contour line is not plotted, the minimum height is in between 0.85 and 0.875.

POLAR COORDINATES

If your problem is best described in polar coordinates, things gets a bit messier. You will need to convert the polar coordinates to Cartesian ones using function `pol2cart` before plotting. Remember that this stupid function uses the order (θ, r) instead of (r, θ).

As an example, here we will plot our beloved sinc function in polar coordinates. In polar coordinates the function is $\sin(r)/r$, where $r = \sqrt{x^2 + y^2}$.

```
% create a set of r values from 0 to, say, 2 pi
rValues=linspace(0,2*pi,40);
```

```
% create a set of theta values from 0 to, of course, 2 pi
thetaValues=linspace(0,2*pi,40);

% create a mesh of all combinations of these values
[rGrid thetaGrid]=meshgrid(rValues,thetaValues);

% the discrete function values on the grid equal sin(r)/r
fGrid=sin(rGrid)./rGrid;

% find the Cartesian coordinates of the points
[xGrid yGrid]=pol2cart(thetaGrid,rGrid);
% note the stupid order theta, r

% plot as a mesh
mesh(xGrid,yGrid,fGrid)
% set square axes of a suitable size
axis([-2*pi 2*pi -2*pi 2*pi -Inf Inf],'square')
% does this seem like a better name than sinc?
title('The "Sombrero Function"')
```