8 PLOTTING

Contents

Initialization

```
% reduce needless whitespace
format compact
% reduce irritations
more off
% start a diary
%diary lectureN.txt
```

SAVING AND RELOADING

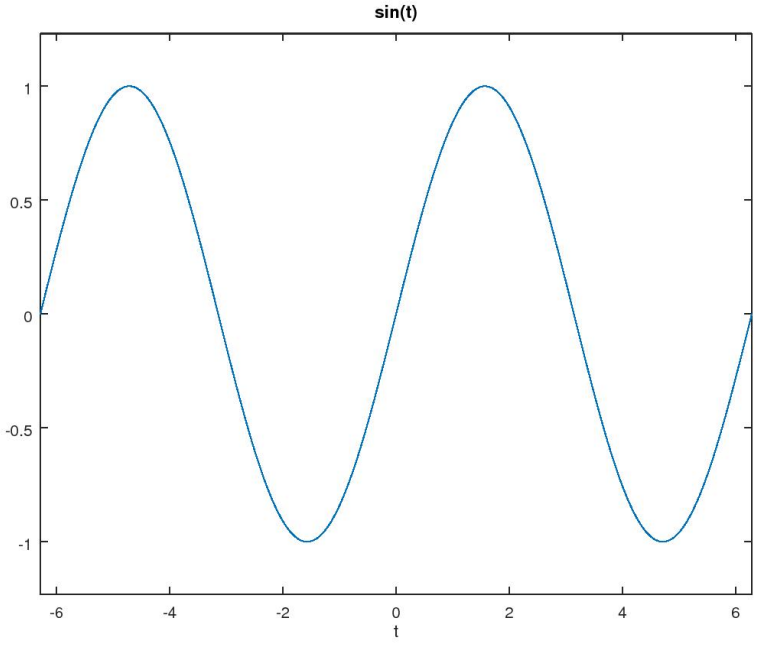
This section should be moved to an earlier lesson, like lesson3. (It has already been moved there.)
You can save all work space variables in a file NAME.mat using the `save NAME` command. Then next time, you can resume where you left off using the `load NAME` command. To save only a few variables, use the `save NAM VAR1 VAR2 ...` command.
Note: to read in data from an Excel spreadsheet, use `xlsread`. To write data to an Excel sheet, use `writetable` or `xlswrite`. Use "cell" arrays if not all data is numerical.

```
% see what variables are defined
who

% save them all in file lecture11.mat
save lecture11

% kill all variables in the work space
clear

% check that they are gone (no response)
who

% reload the variables from file lecture11.mat
load lecture11

% check that they are back.
who
```

PLOTTING ANALYTIC FUNCTIONS

Plotting a simple function

```
% just specify the function in ezplot
ezplot('sin(t)')
```

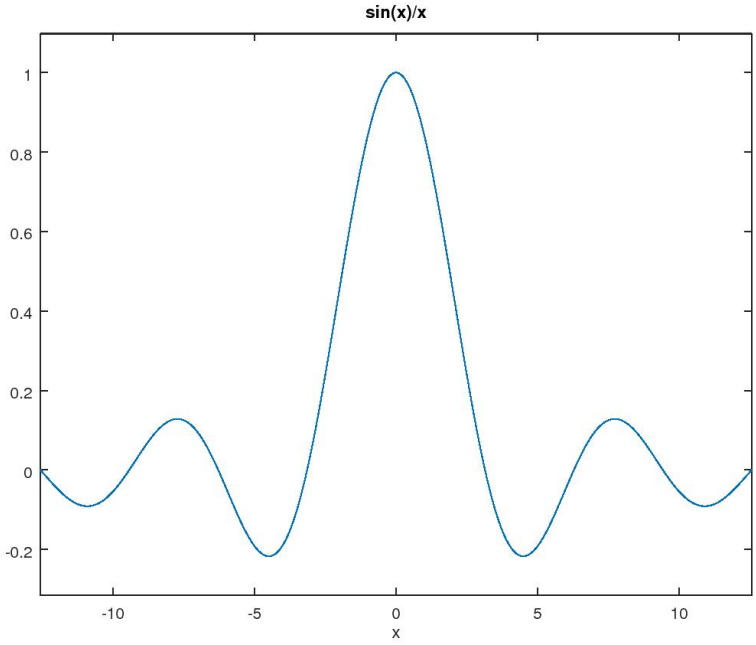


Plotting a simple function with specified limits

```
% change the axis size from the default [-2*pi 2*pi]
ezplot('sin(x)/x',[-4*pi 4*pi])
```

Plotting an implicit function

```
% plot the first (hyper)circle with ezplot
ezplot('x^2+y^2-1',[-1 1 -1 1])
% make sure the next ones go in the same plot
hold on
```

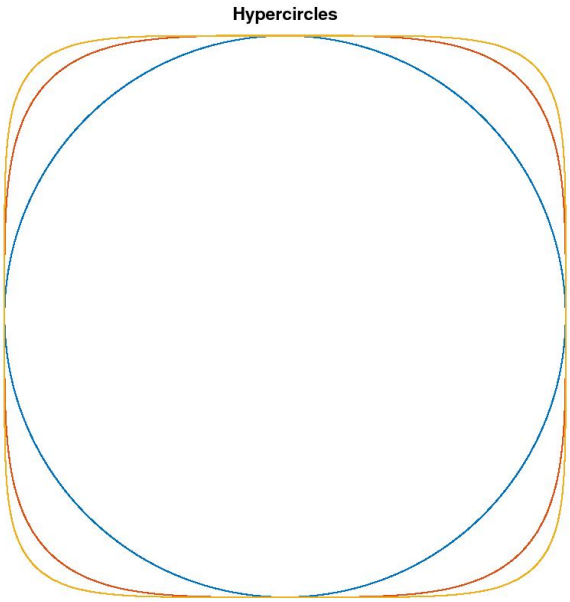


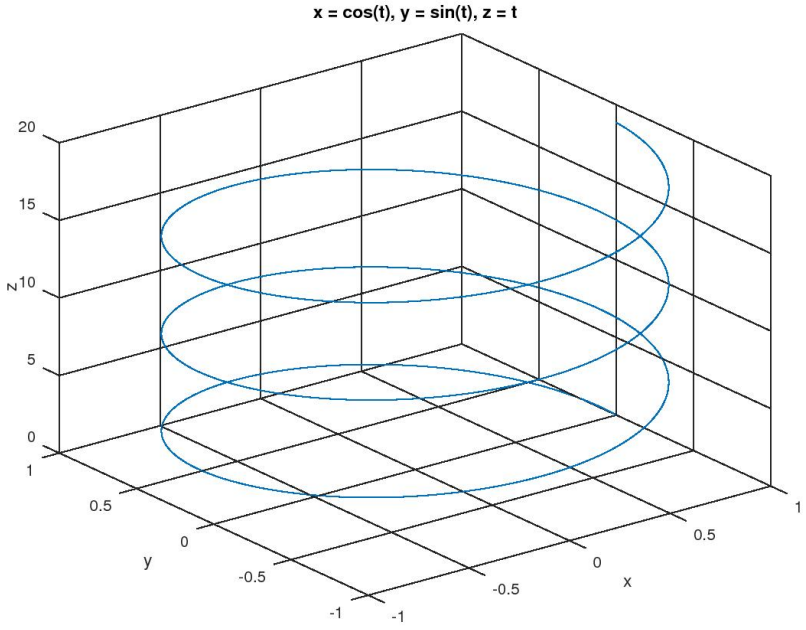

```
% plot the second hypercircle
ezplot('x^4+y^4-1',[-1 1 -1 1])
% plot the third hypercircle and set a better axis range
ezplot('x^6+y^6-1',[-1 1 -1 1])
% make the axes square, with an appropriate range
axis('square','off')
% change the title
title('Hypercircles')
% next plot erases the current one
hold off
```

Plotting a curve in three dimensions

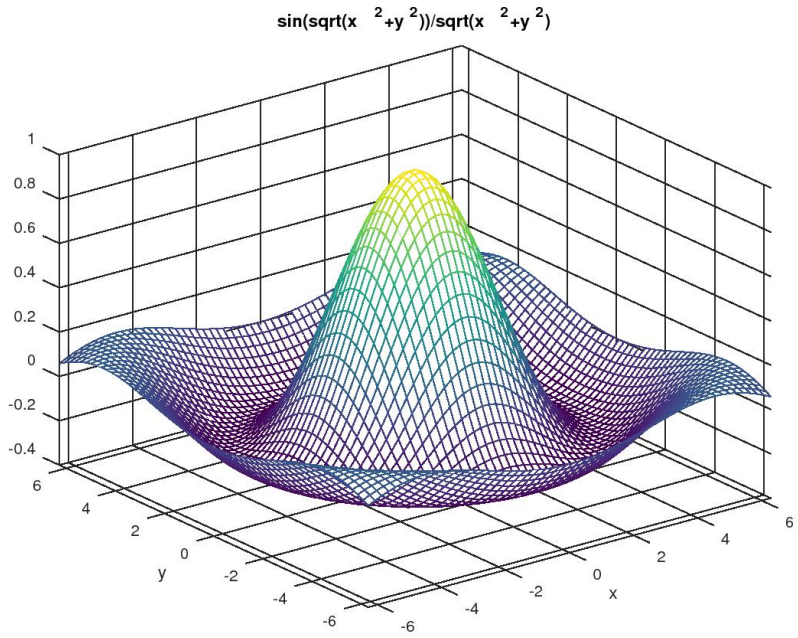
```
% use ezplot3
ezplot3('cos(t)','sin(t)','t',[0 6*pi])
```

Plotting a function of two variables as a mesh


Hypercircles


x = cos(t), y = sin(t), z = t
20
15
z 10
5
0
1
0.5
0
y
-0.5
-1
-1
-0.5
0
x
0.5
1

```
% use ezmesh
ezmesh('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)')
```



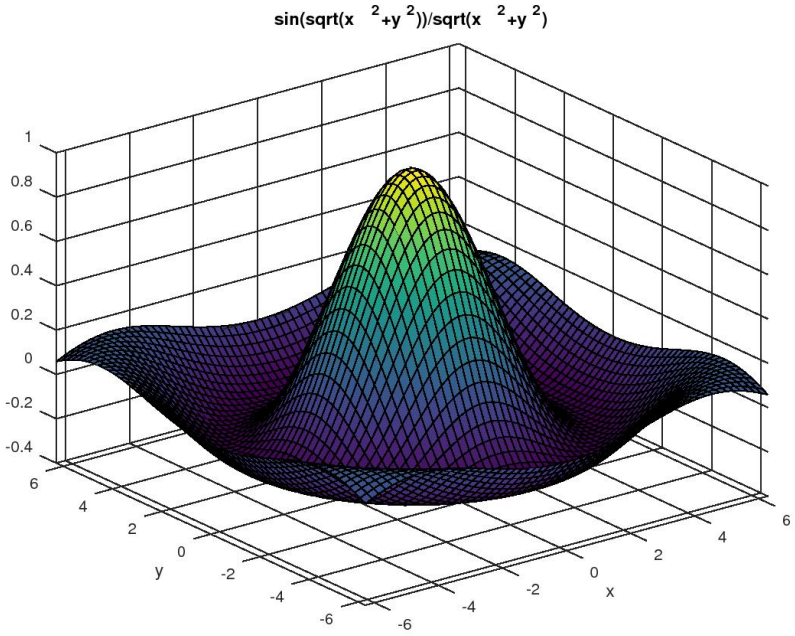
Plotting a function of two variables as a surface

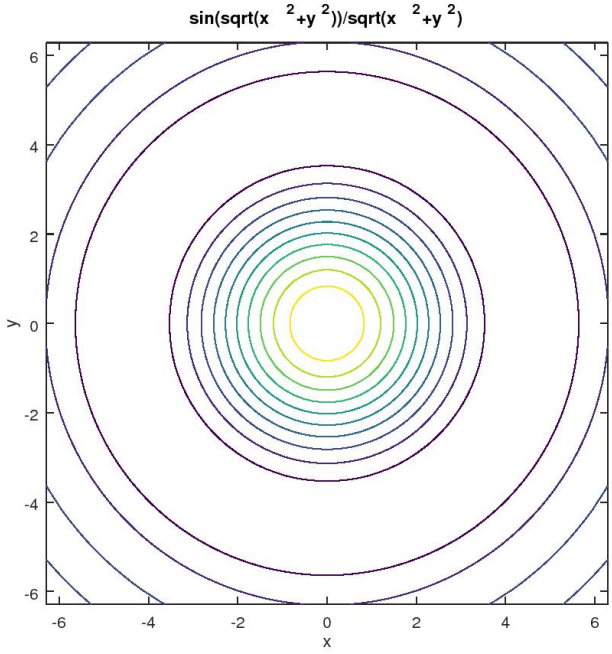
This may not be a good idea for hard copy publication

```
% use ezsurf
ezsurf('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)')
```

Plotting contour lines for a function of two variables

```
% use ezcontour
ezcontour('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)')
axis('square')
```

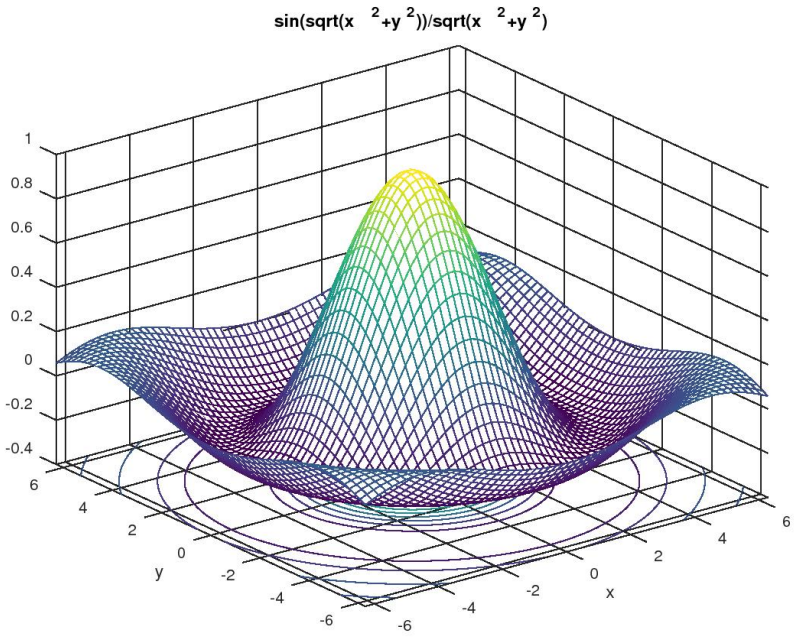

sin(sqrt(x 2+y 2))/sqrt(x 2+y 2)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
6
4
2
0
-2
-4
-6
y
-6
-4
-2
0
2
4
6
x


sin(sqrt(x 2+y 2))/sqrt(x 2+y 2)
6
4
2
0
-2
-4
-6
y
-6
-4
-2
0
2
4
6
x

Plotting both mesh and contour lines

```
% use ezmeshc
ezmeshc('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)')
disp('Try rotating the graph!')
```

```
Try rotating the graph!
```



PLOTTING NON-ANALYTIC FUNCTIONS

Using log-log plots

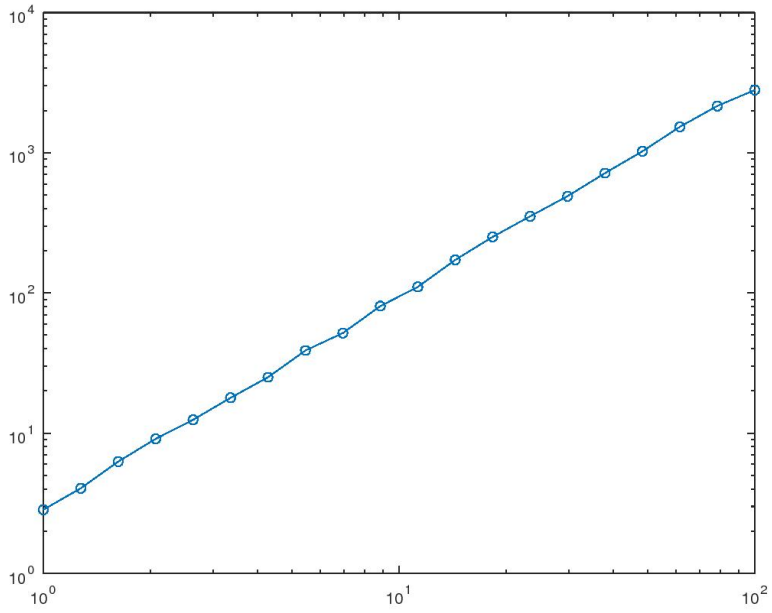
If you have measured data points, say y-values corresponding to x-values, you often want to see whether the relationship between the two can be approximated by a power relationship, like in

$$y = Cx^p$$

where p is some power and C is some other constant.
You can test whether the above relationship is a good one by plotting y versus x in a `loglog` plot. If the above power relationship is accurate, the graph in the `loglog` plot will be a straight line. And the slope of the straight line will tell you the value of the power p.

Note: there are also `semilogx` and `semilogy` plots for if you want to check for an exponential relationship instead of a power one.

```
% create the supposed values of x of the "measurements"
x=logspace(0,2,20)';

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

% add some experimental error to make it "real"
randn('seed',1) % 987654321 for similar results in Matlab
y=y.*(1+0.05*randn(size(x)));

% plot
loglog(x,y,'o-');
```



Adding the power relationship

If the measured y-values can be approximated by

$$y = Cx^p$$

then taking logarithm of both sides

$$\ln y = p \ln x + \ln C$$

Defining for now $Y \equiv \ln y$, $p \equiv C_1$, $X \equiv \ln x$ $\ln C \equiv C_2$, this takes the form

$$Y = C_1 X + C_2$$

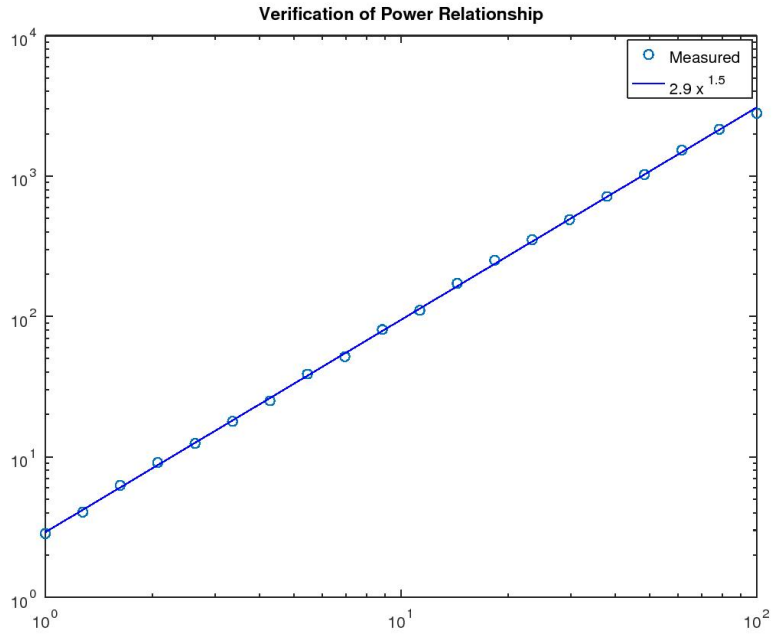
We saw in lesson3 that the best values of $C_1 = p$ and $C_2 = \ln C$ can then be found using `polyfit` with $n = 1$.

```
% create the X and Y values
X=log(x);
Y=log(y);
% find the coefficients of the straight line
coefs=polyfit(X,Y,1);
% find p and C
p=coefs(1)
C=exp(coefs(2))

% create the power relation
yPower=C*x.^p;

% plot power relation and measured points now
loglog(x,y,'o',x,yPower,'b-');
title('Verification of Power Relationship')
legend('Measured',...
       [num2str(C,2), ' x^{', num2str(p,2), '}'])
```

```
p =  1.5131
C =  2.9044
```

Plotting three dimensional (3D) data

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

% create the x- and y-values of the m x n points now
[xGrid yGrid]=meshgrid(xValues,yValues)
```

```
% plot it
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
```

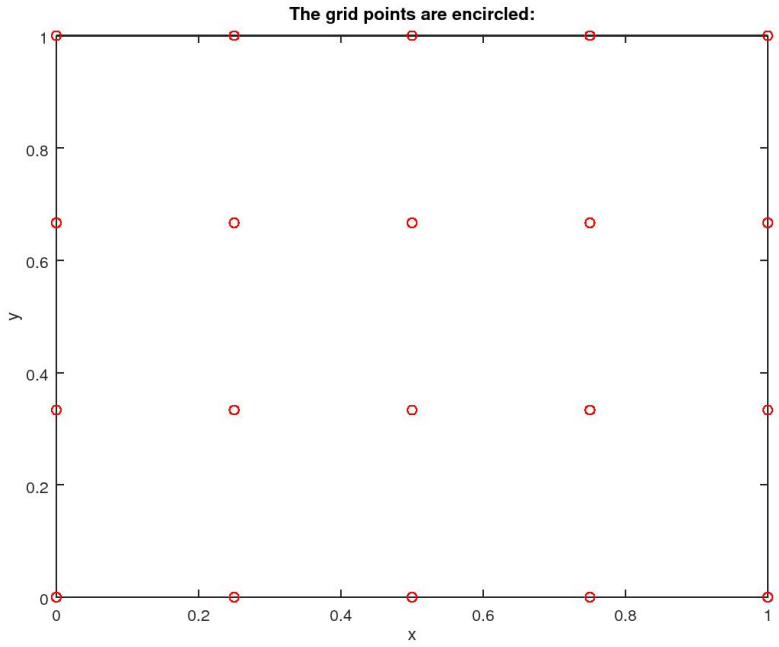
Finding the height of the membrane

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 `data` with data on the desired solution.

At any interior point of the grid, this array should contain the value of the "forcing" 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, which we take to be constant and 2 for simplicity.

At any boundary point, (i=1, i=m, j=1, or j=n), array `data` should contain the desired value of the solution at that point. We assume that the membrane is attached to the drum at a constant height 1.

```
% the interior point forcing
forcing=2

% the attachment height of the membrane
heightBoundary=1
```




```
% initialize the data array to all forcing
data=forcing*ones(size(xGrid));

% change it to heightBoundary on the four boundaries
data(1,:)=heightBoundary;
data(m,:)=heightBoundary;
data(:,1)=heightBoundary;
data(:,n)=heightBoundary;

% let simplePoisson find the heights at the grid points
height=simplePoisson(xGrid,yGrid,data)
```

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

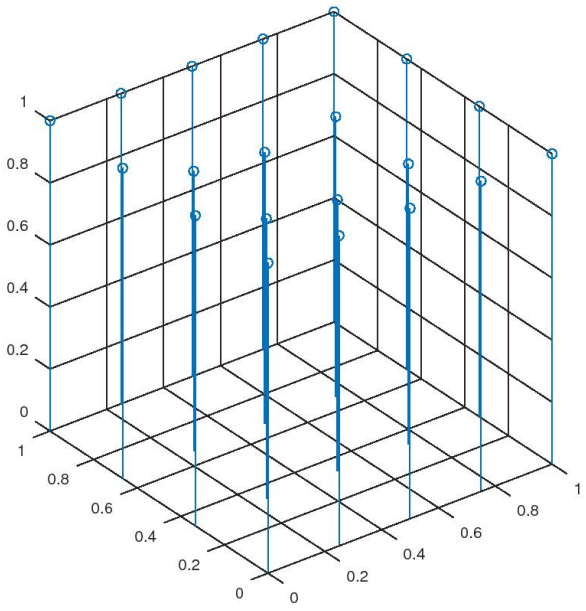
```
1.00000   0.90248   0.87511   0.90248   1.00000
1.00000   1.00000   1.00000   1.00000   1.00000
```

Let's have a look at the raw data

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

```
stem3(xGrid,yGrid,height)
axis('square')
disp('Rotate the graph!')
```

```
Rotate the graph!
```

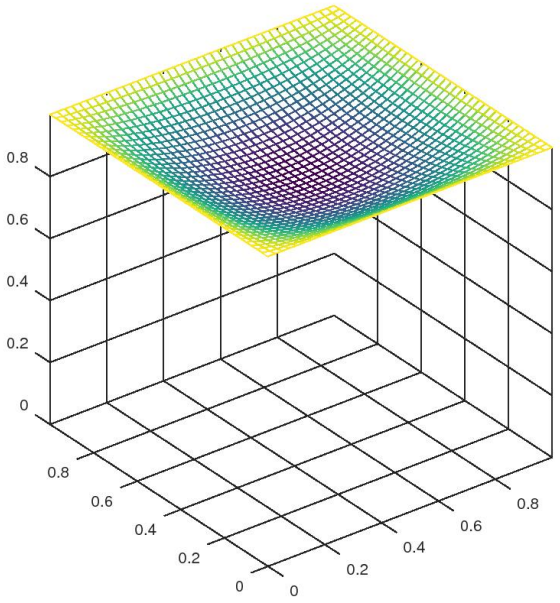


Plot as a surface

```
% let's have some more points for a better appearance
n=40
xValues=linspace(0,1,n);
m=40
```

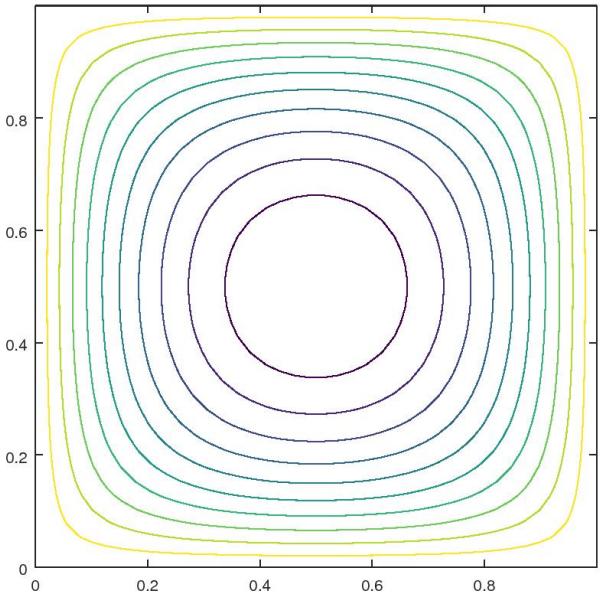
```
yValues=linspace(0,1,m);
[xGrid yGrid]=meshgrid(xValues,yValues);
data=forcing*ones(size(xGrid));
data(1,:)=heightBoundary;
data(m,:)=heightBoundary;
data(:,1)=heightBoundary;
data(:,n)=heightBoundary;
height=simplePoisson(xGrid,yGrid,data);

% use mesh
mesh(xGrid,yGrid,height)
axis([0 1 0 1 0 1],'square')
```

```
n =  40
m =  40
m =  40
n =  40
N =  1600
relErrorMatlab =     9.2407e-12
```



Plot contour lines

```
% use contour
contour(xGrid,yGrid,height)
axis('square')
```

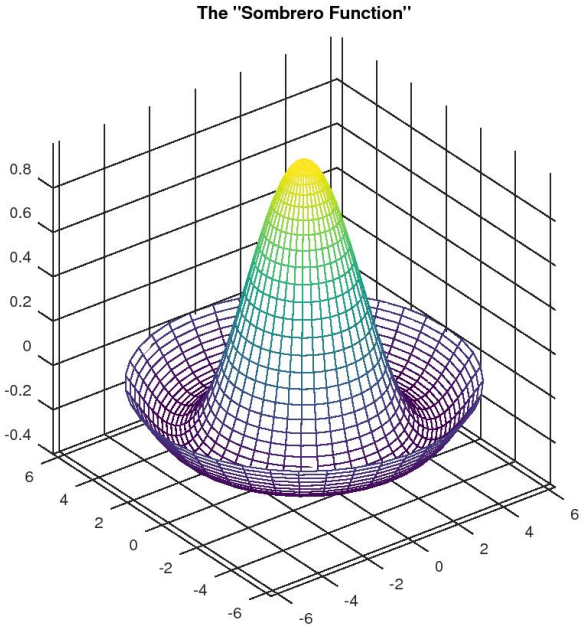


Try polar coordinates

Polar coordinates can be converted to Cartesian using function `pol2cart` before plotting. Note that idiot function `pol2cart` uses the order "theta,r".

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

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

% assume our f values are the beloved sin(r)/r
fGrid=sin(rGrid)./rGrid;
```

```
% find the Cartesian coordinates of the points
[xGrid yGrid]=pol2cart(thetaGrid,rGrid);

% plot using mesh
mesh(xGrid,yGrid,fGrid)
title('The "Sombrero Function"')
axis([-2*pi 2*pi -2*pi 2*pi -Inf Inf],'square')
```



End lesson 8