

The background is a dark blue gradient with a subtle pattern of small white dots. Overlaid on this are several faint, light blue circular and semi-circular lines. Some of these lines have tick marks and numerical labels, resembling a circular scale or a radar chart. The labels include 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, and 260. There are also arrows pointing in various directions, suggesting a sense of rotation or movement.

# ENGR 101 – Chapter 8

## Advanced Plotting and Data Visualization

Laura Alford, James Juett, Rick Niciejewski

9/26/2020

# 2D vs. 3D Data Visualization

- We used three dimensional arrays to represent images with rows, columns, and layers.



- We can also represent data with (x,y,z) values

X	1	3	5	7	9	11	13	15	17
Y	4	8	2	-1	18	5	9	10	-2
Z	40	37	34	31	28	25	22	19	16

# 3D Plotting

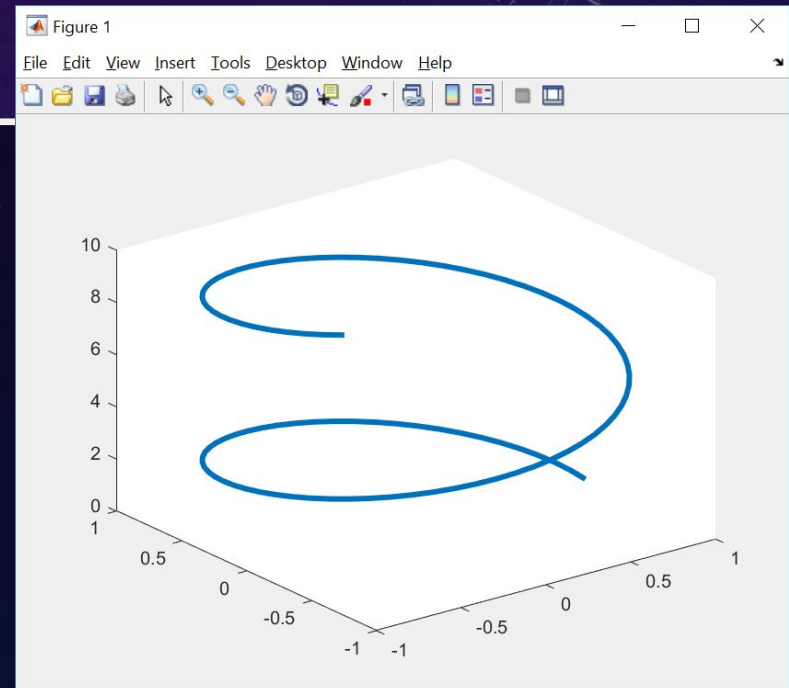
- Use the **plot3** function to create three dimensional plots.
- It works analogously to regular, 2D plotting.
  - Just provide a set of x, y, and z values.

```
% Our third axis is time  
% between 0 and 10 seconds  
t = linspace(0,10,100);
```

```
% x and y vary over time  
x = cos(t);  
y = sin(t);
```

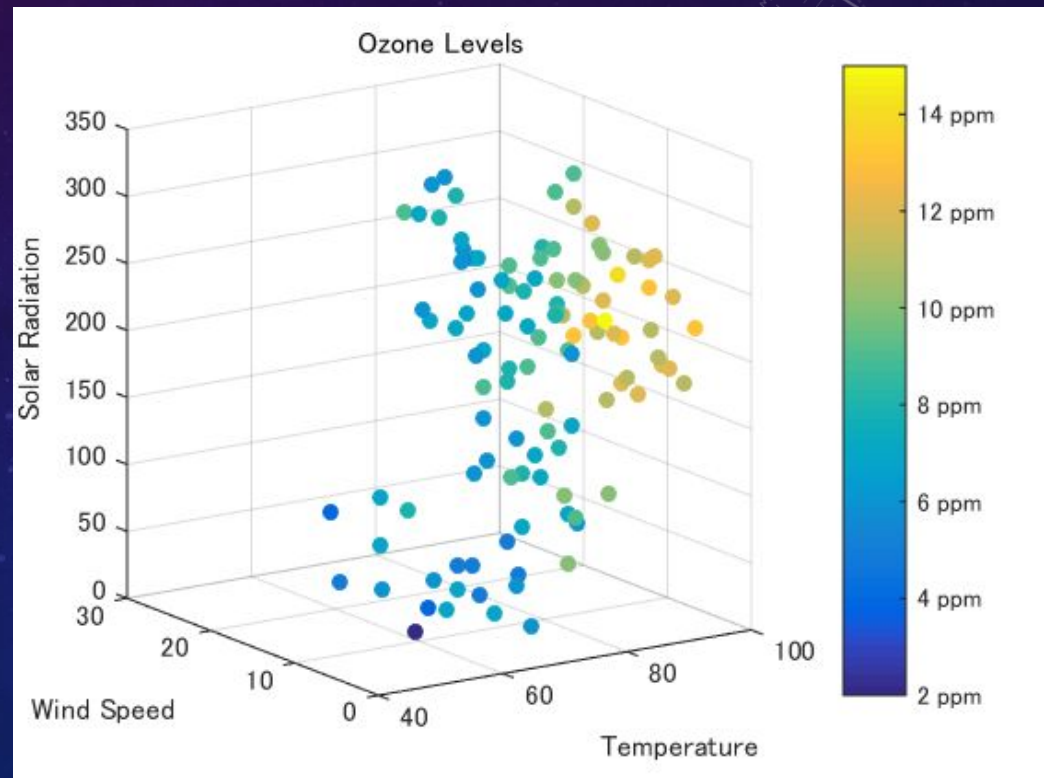
```
h = plot3(x, y, t);
```

```
h.LineWidth = 3; % wider than usual for projector
```



# scatter3

- Example: Measurements of ozone levels and other atmospheric conditions. See the MathWorks site linked below for more details.



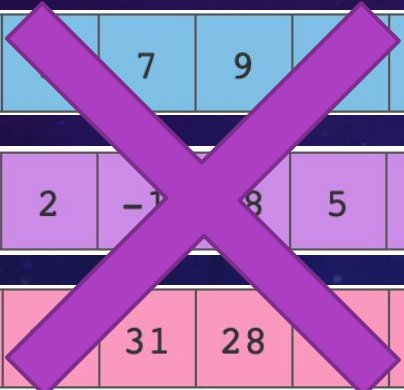


# Plotting Functions of Two Variables in 3D


- Consider the mathematical function:

$$z = 2 - (x^2 + y^2)$$

- How could we plot this in MATLAB?



X	1	3		7	9		13	15	17
Y	4	8	2	-1	8	5	9	10	-2
Z	40	37		31	28		22	19	16

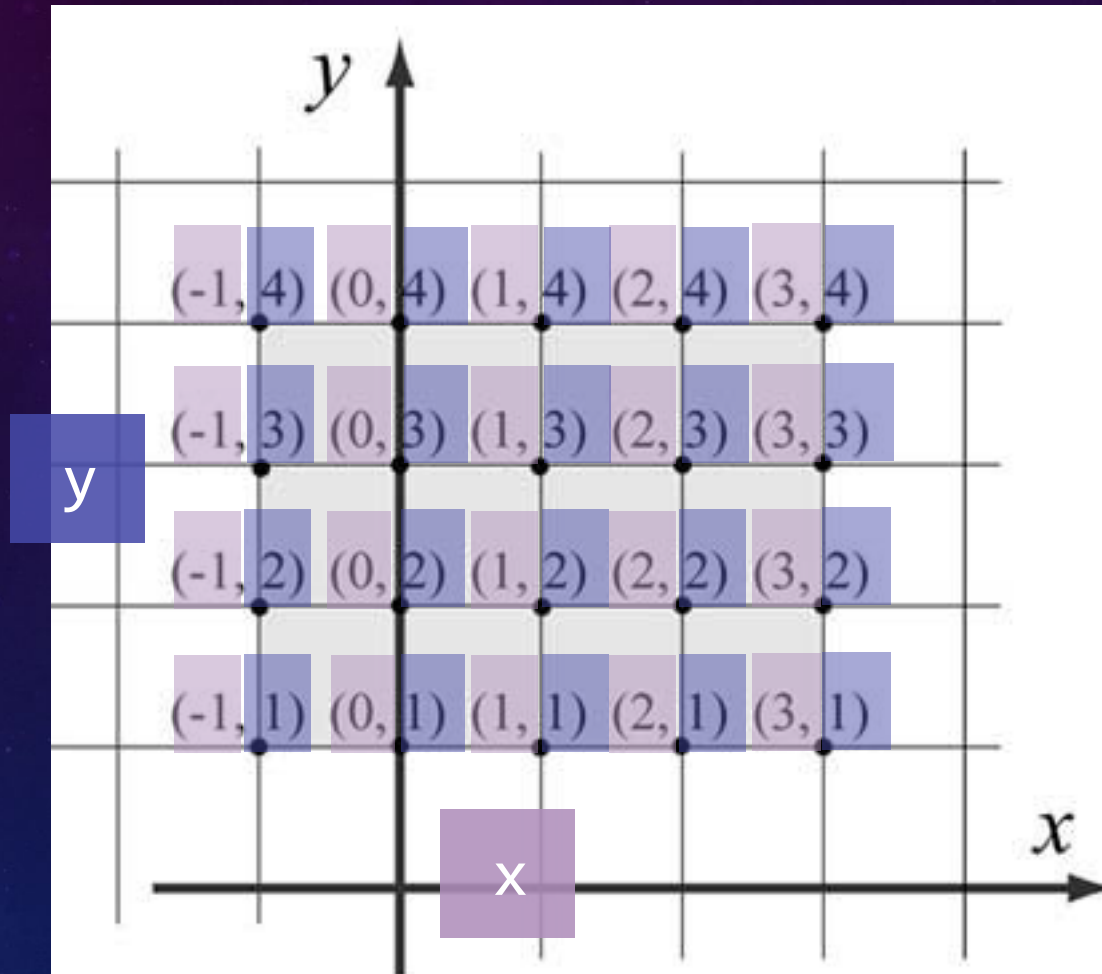


-6	-3	-2	-3	-6
-3	0	1	0	-3
-2	1	2	1	-2
-3	0	1	0	-3
-6	-3	-2	-3	-6

Z

- Remember, MATLAB does NOT plot math functions – it plots x, y, and z data points. We need to calculate a matrix for z from **all the combinations of x and y coordinates**.

# x and y coordinates



# meshgrid

x	-2	-1	0	1	2
y	-2	-1	0	1	2

- The **meshgrid** function is quite useful for calculating functions of two variables.

```
x = -2:1:2;  
y = -2:1:2;  
[X,Y] = meshgrid(x,y);  
Z = X + Y;
```

-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2

X

-2	-2	-2	-2	-2
-1	-1	-1	-1	-1
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2

Y

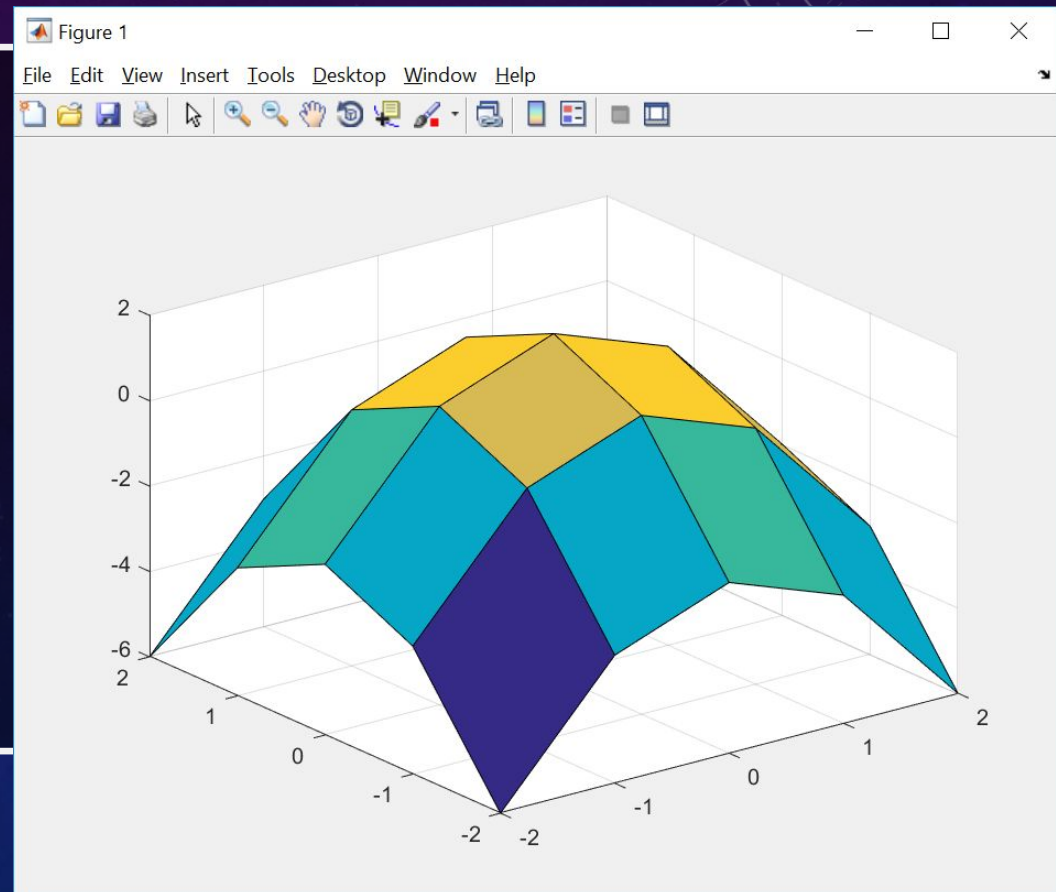
-4	-3	-2	-1	0
-3	-2	-1	0	1
-2	-1	0	1	2
-1	0	1	2	3
0	1	2	3	4

Z

# 3D Surface Plots

□ Use the **surf** function to create 3D surface plots.

```
x = -2:1:2;  
y = -2:1:2;  
[X,Y] = meshgrid(x,y);  
  
Z = 2 - (X.^2 + Y.^2);  
  
surf(X,Y,Z);
```

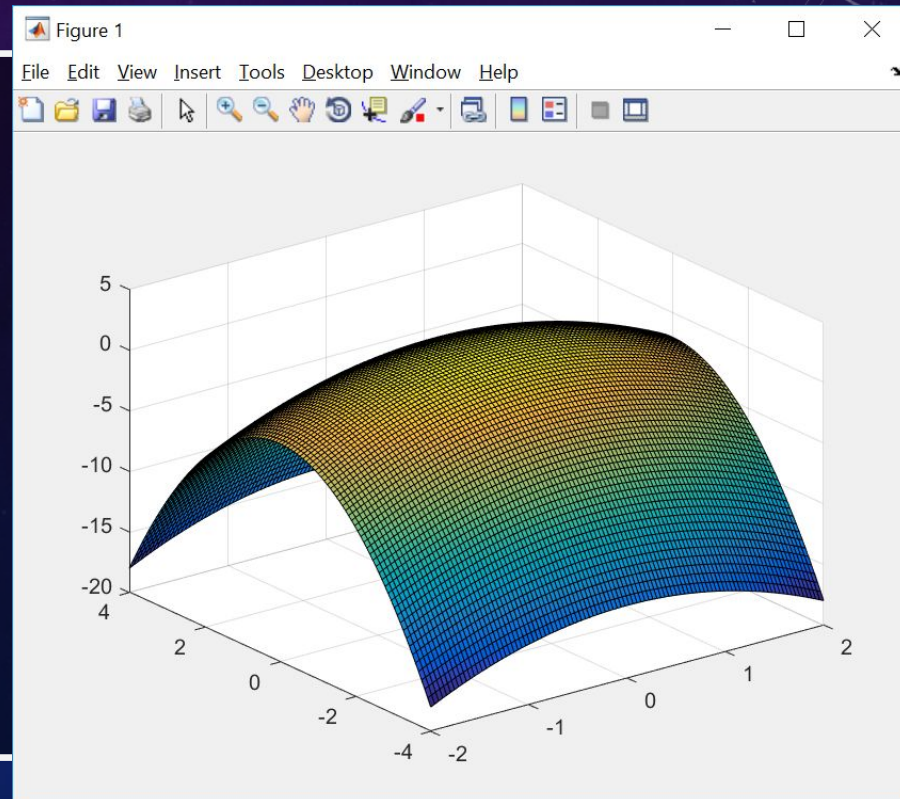




# 3D Surface Plots

- Use the **linspace** function for higher resolution x and y.
- x and y don't have to be the same.

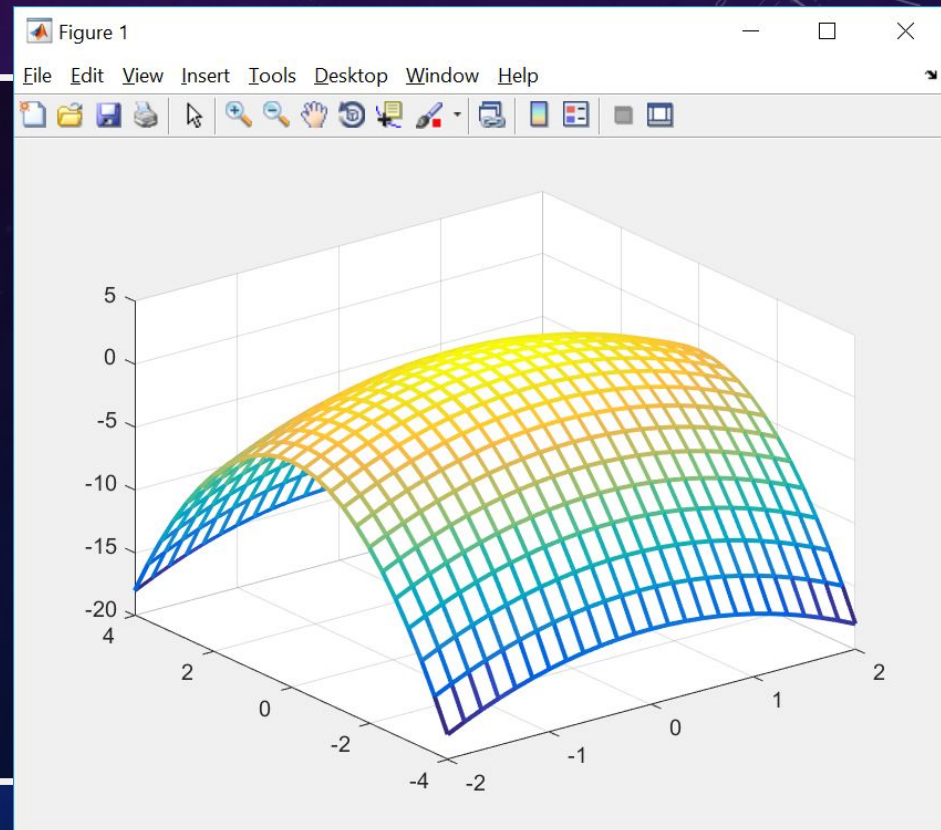
```
x = linspace(-2,2,100);  
y = linspace(-4,4,100);  
[X,Y] = meshgrid(x,y);  
  
Z = 2 - (X.^2 + Y.^2);  
  
surf(X,Y,Z);
```



# 3D Mesh Plots

- The mesh function works similarly to surf, but does not color in sections of the surface.

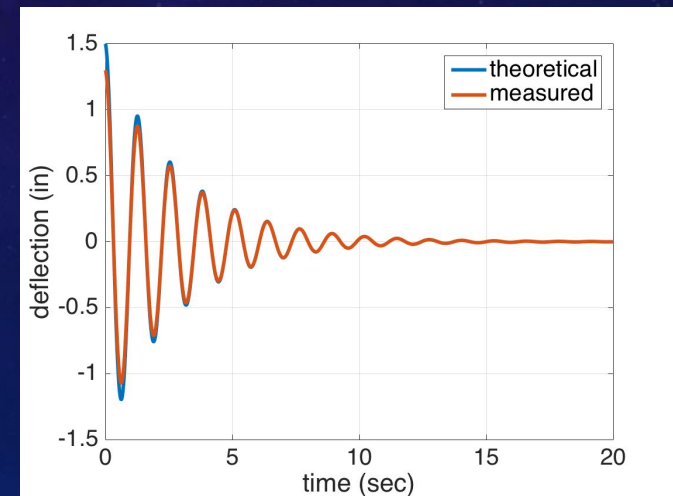
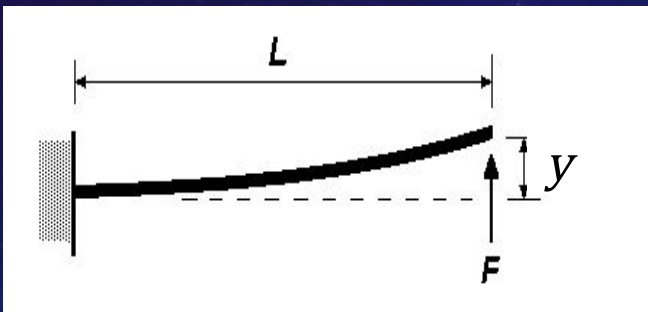
```
x = linspace(-2,2,25);  
y = linspace(-4,4,25);  
[X,Y] = meshgrid(x,y);  
  
Z = 2 - (X.^2 + Y.^2);  
  
h = mesh(X,Y,Z);  
h.LineWidth = 2;
```





## Exercise: Optimize Beam Characteristics

- Let's investigate different damping ratios and natural frequencies to see if our original guess was a good one.
- Complete the `OptimizeBeamCharacteristics.m` script on the google drive.
- Use a damping ratio range of 0.05 to 0.15
- Use a natural frequency range of 4.5 to 5.5





# Solution: Optimize Beam Characteristics

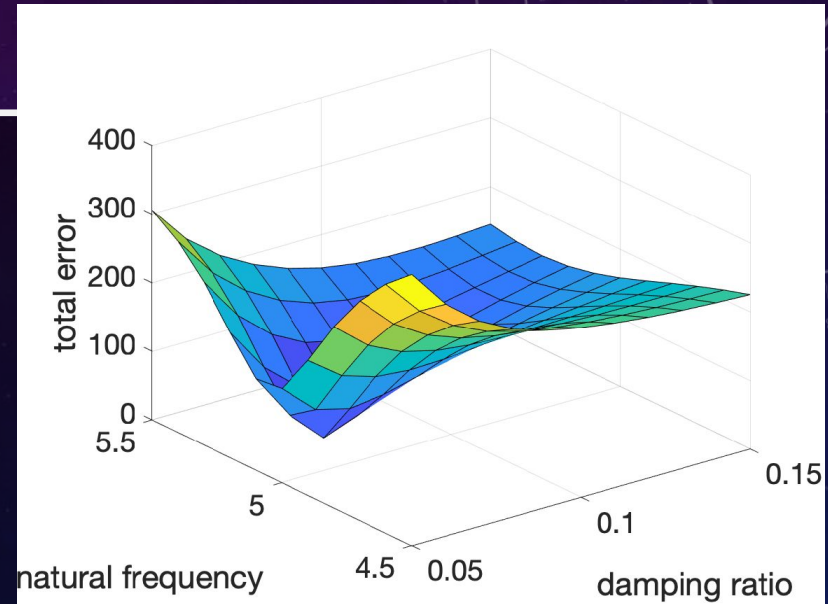
```
% Damping ratio range
d = 0.05:0.01:0.15;

% Natural frequency range
w_n = 4.5:0.1:5.5;

% create combos of d and w_n
[D, W_N] = meshgrid(d, w_n);

% get the error for each (D, W_N) combo
solutionErrors = vibrationError(D, W_N);

% create 3D surface plot
surf(D, W_N, solutionErrors);
```

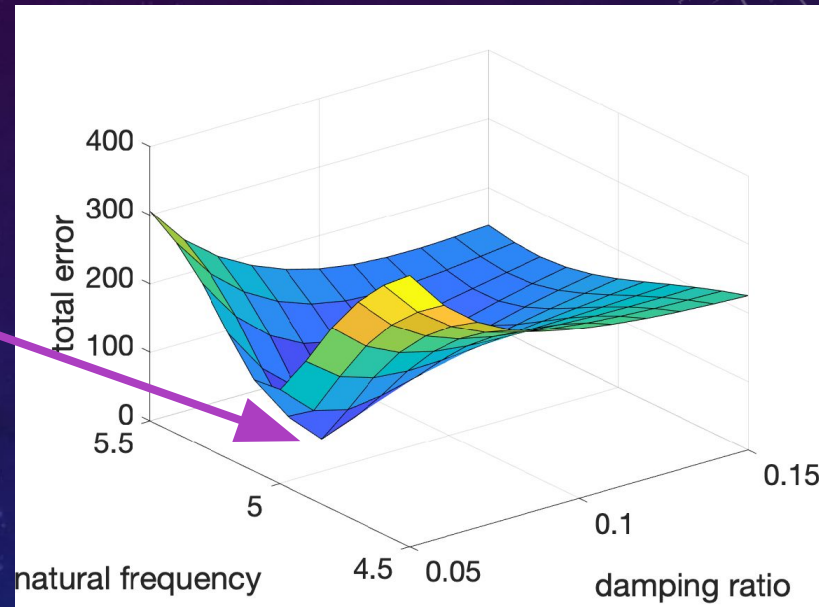




# Interpretation: Optimize Beam Characteristics

This “lowest point” corresponds to the minimum error.

The natural frequency and damping ratio at this point are our best guesses based on the experimental data.



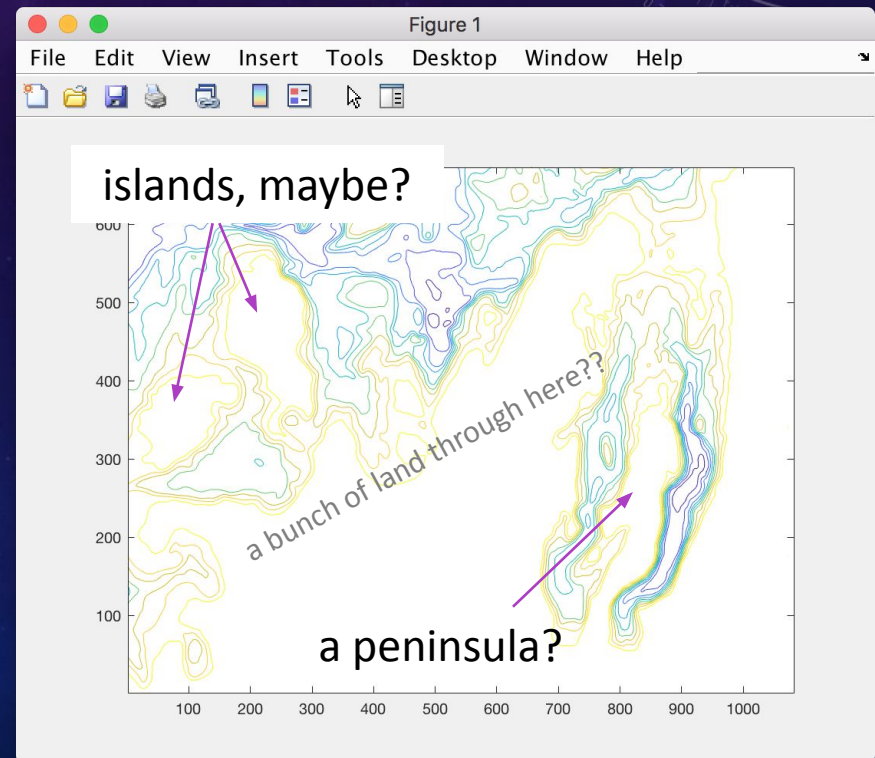
# Making Contour Maps

- Use the **contour** function to create a contour map.
- Each line represents locations with equal values.

```
load bathymetryData;  
contour(bathymetryData);
```

This is a map of the water depth in part of northern Lake Michigan. The white areas correspond to **NaNs** in the data file – meaning land in this case.

- This plot is a little hard to understand though...



# Making Contour Maps

- **contourf** works like contour, but fills in regions with color.
- Each line represents locations with equal values.

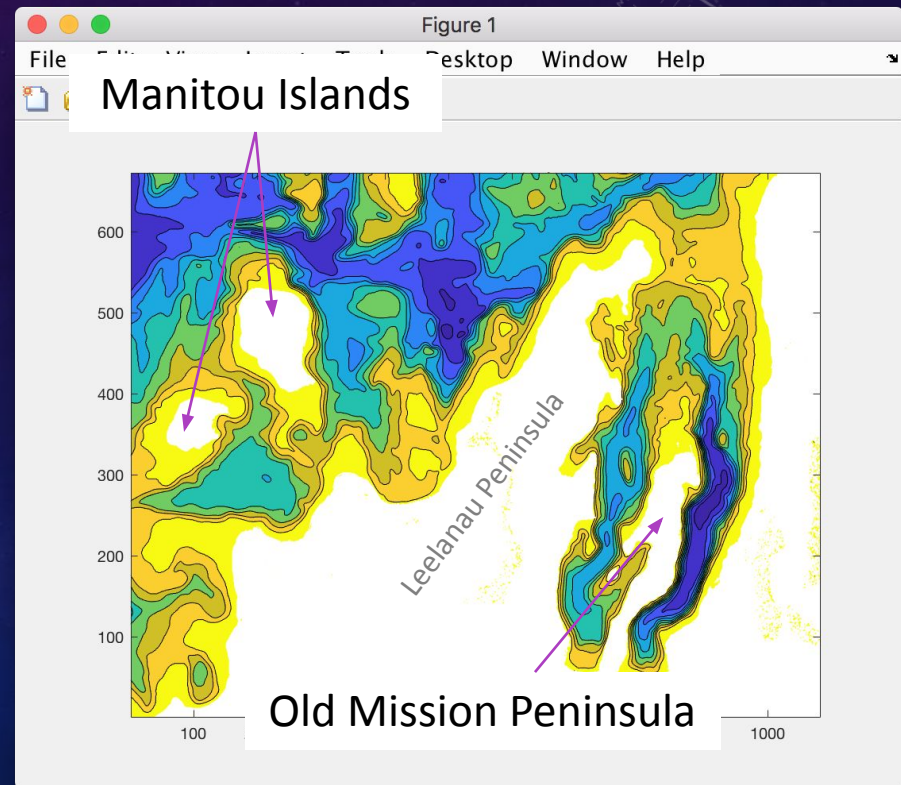
```
load bathymetryData;  
contourf(bathymetryData);
```

- It's easier to see the water depth (and the land!) in this version of the plot.

Data source:

<https://maps.ngdc.noaa.gov/viewers/wcs-client/>

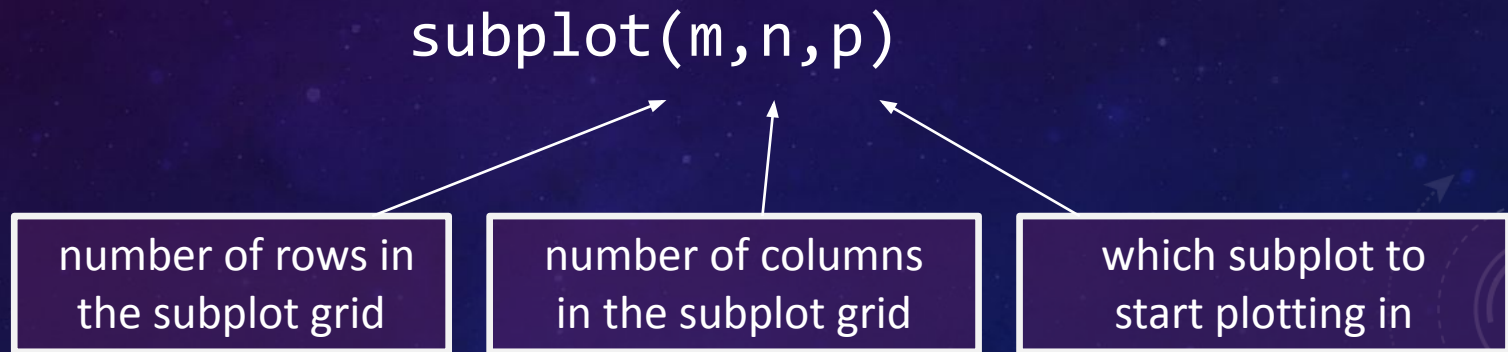
1. Choose a Layer – Great Lakes Bathymetry
2. Zoom in to Lake Michigan; draw box around what you want data for
3. Choose ArcGIS ASCII Grid output format
4. Click to download





# subplot

- The **subplot** function allows multiple axes per figure.
- The axes are arranged in a grid-like configuration.





# subplot Schematic Examples

`subplot(2,1,1)`

`subplot(2,1,2)`

`subplot(2,2,1)`

`subplot(2,2,2)`

`subplot(2,2,3)`

`subplot(2,2,4)`

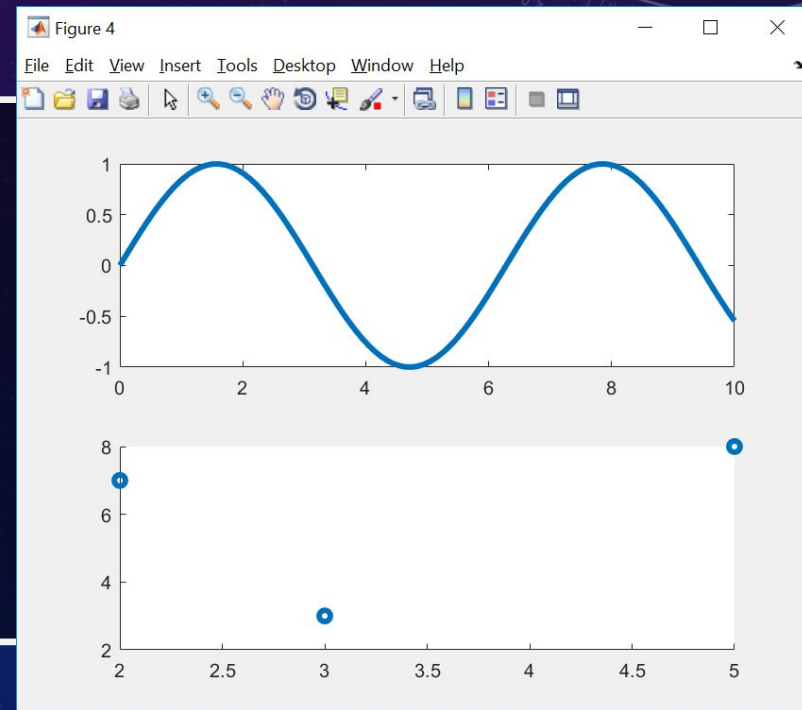
# subplot

- The **subplot** function allows multiple axes per figure.
- The axes are arranged in a grid-like configuration.
- Example:

```
figure(); % create a figure

% select the first plot in a 2x1 grid
subplot(2,1,1);
x = linspace(0,10,100);
plot(x, sin(x));

% select the second plot in the 2x1 grid
subplot(2,1,2);
scatter([2,3,5], [7,3,8]);
```





3 min

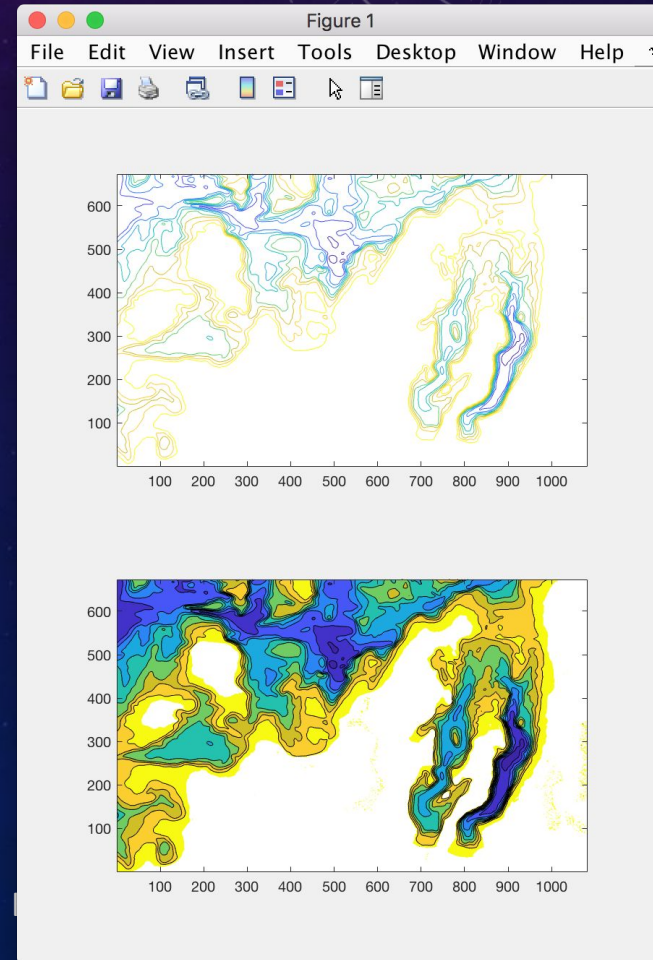
# Exercise: Compare Contour Plots

- Complete the UpNorth.m script to plot both contour plots in the same figure.

```
load BathymetryData; % load the data
figure(); % create a figure

% select the first plot in a 2x1 grid
axis equal; % keeps the aspect ratio correct

% select the second plot in the 2x1 grid
axis equal; % keeps the aspect ratio correct
```



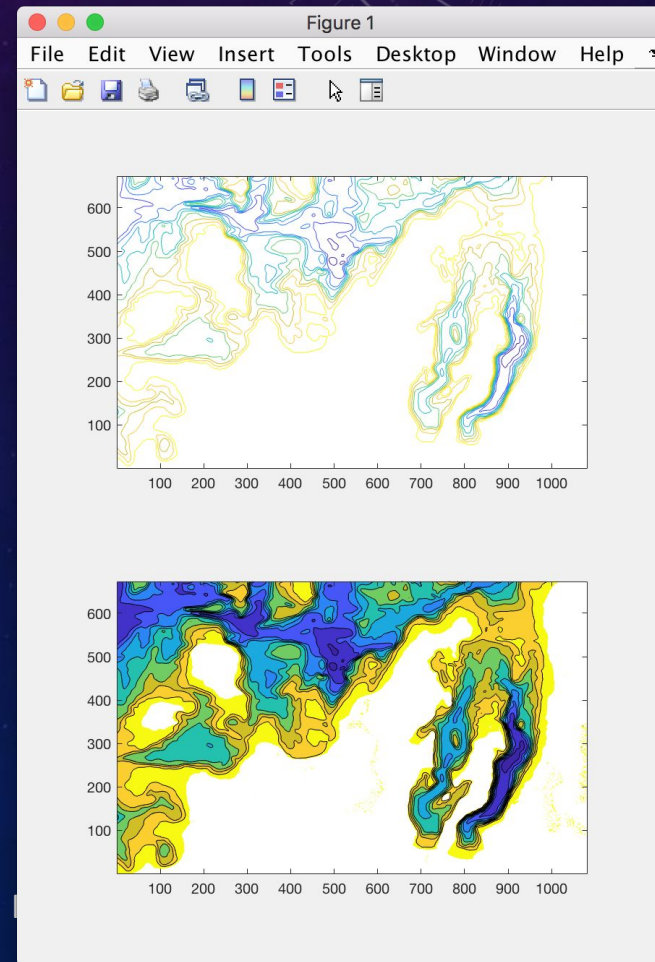
# Solution: Compare Contour Plots

- Complete the UpNorth.m script to plot both contour plots in the same figure

```
load BathymetryData; % load the data
figure(); % create a figure

% select the first plot in a 2x1 grid
subplot(2,1,1);
contour(bathymetryData);
axis equal; % keeps the aspect ratio correct

% select the second plot in the 2x1 grid
subplot(2,1,2);
contourf(bathymetryData);
axis equal; % keeps the aspect ratio correct
```





# Exercise: Projectile Motion - Golf



This is the 17<sup>th</sup> hole at the TPC Sawgrass golf course.

It is 120 meters from the tee box to the hole on the island green. But if you hit the ball in the water, you lose a stroke and have to hit again, so that's bad.

**How fast and at what angle** should you hit the ball so it lands on the green instead of in the water?



4 min

# Exercise: Projectile Motion - Golf

dist2green.m

LandOnGreen.m

- Write a function `dist2green` that calculates the distance a golf ball will travel before it hits the ground<sup>1</sup>:
  - `t = 2 .* speed .* sin(angle) ./ 9.8 % g = 9.8 m/s^2`
  - `distance = speed .* cos(angle) .* t`
- Use `meshgrid` to create matrices that correspond to the different speed and angle combinations.
- Calculate the distance traveled for each speed/angle combination by calling `dist2green` and passing the matrices created by `meshgrid`.
- Make two plots to interpret your simulation.
  - A `contourf` plot of speed, angle, and distance
  - A `contourf` plot of the viable combinations of speed and angle

<sup>1</sup>This is one of the solutions to the  $y = v_0 \sin(\theta) t - \frac{1}{2} g t^2$  equation we saw in Lecture 10; in this case,  $y = 0$ , and the other solution is  $t = 0$  (the ball is hit).

# Solution: Projectile Motion - Golf

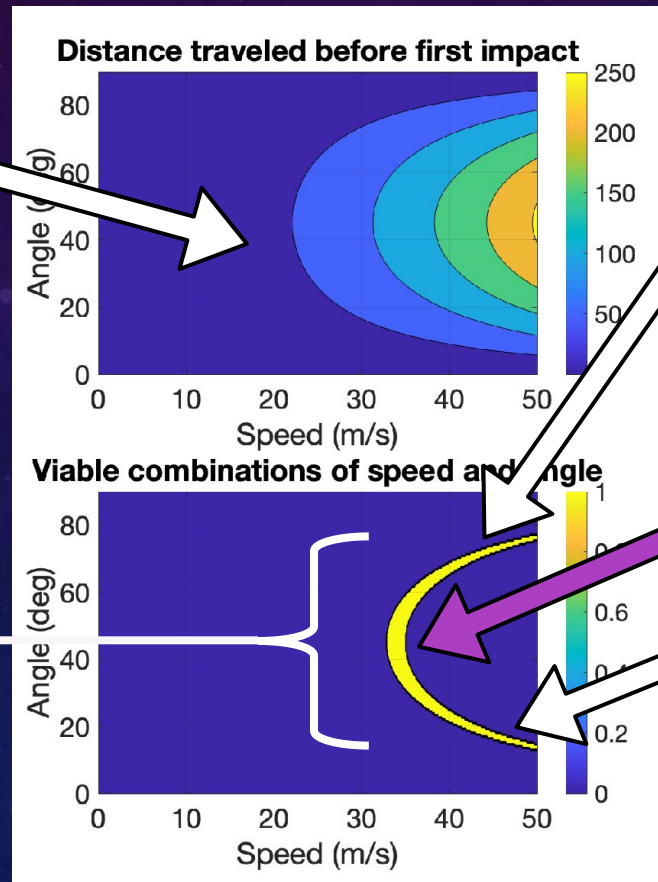
```
function [distance] = dist2green(speed,angle)
    % determine time in the air, neglecting air drag, etc.
    t = 2.* speed.* sin(angle) ./ 9.8;
    % determine horizontal motion for this length of time
    distance = speed .* cos(angle) .* t;
end
```

```
...
% use meshgrid to get matrices of the initial conditions
[V, T] = meshgrid(v0,theta);
% determine distance traveled (before first hit!)
distance = dist2green(V,T);
...
% code that plots things...
...
% locations where distance is within desired range
loc = distMin < distance & distance < distMax;
...
% code that plots things...
```



# Interpretation: Projectile Motion - Golf

This graph just shows all the combinations of speed, angle, and distance, but it's good to see what the overall range of the data is



This yellow arc shows all the combinations of speed and angle that land the ball in our desired range of distances...

But hitting the ball with a high angle and a high speed is really difficult, even for professionals.

This is the “sweet spot,” the combinations that are likely to result in success. Remember, just because math gave you an answer doesn't mean it's a useful answer!

And hitting the ball with a low angle and a high speed will likely cause the ball to go bouncing right through the green and into the water.