MATH2221
Mathematics Laboratory II

Lecture 7:
Miscellaneous Visualization Topics and Review for Test 1

Gary Choi

February 25, 2025

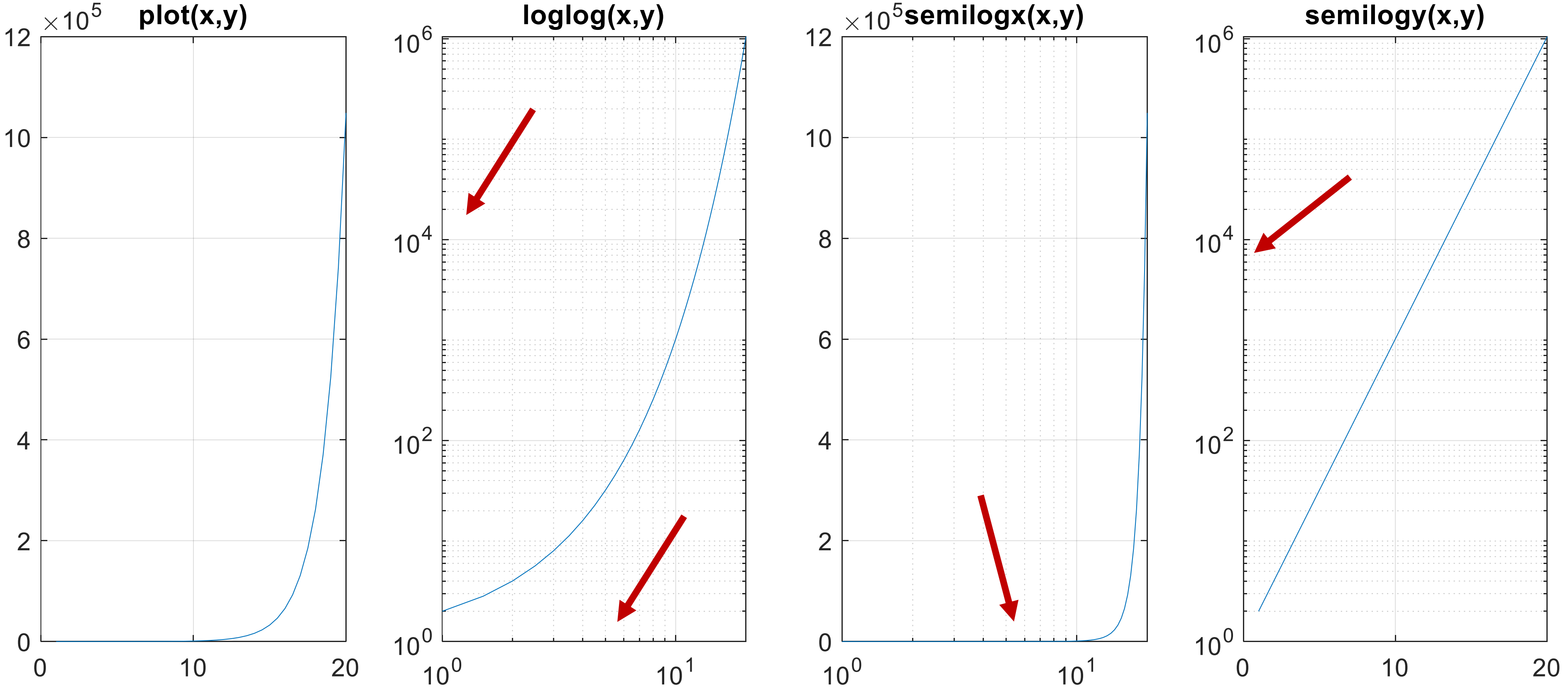
Recall: Some specialized 2D plots in MATLAB

- **Log-log plots**
  - loglog(x,y): plots x- and y-coordinates using a base-10 logarithmic scale on the x-axis and the y-axis

- **Semi-log plots**
  - semilogx(x,y): only use a base-10 logarithmic scale on the x-axis
  - semilogy(x,y): only use a base-10 logarithmic scale on the y-axis

Example:
plotting $y = 2^x$

```
x = 1:0.5:20;
y = 2.^x;
```

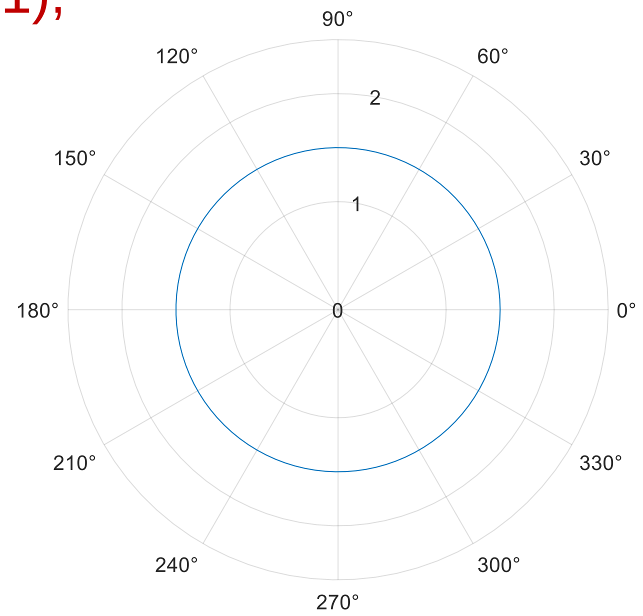


Recall: Some specialized 2D plots in MATLAB

- **Polar plots**
  - polarplot(theta,rho):
    Plot a curve $r(\theta)$ in the polar coordinate system,
    with θ indicating the angle in radians and
    r indicating the radius value for each point

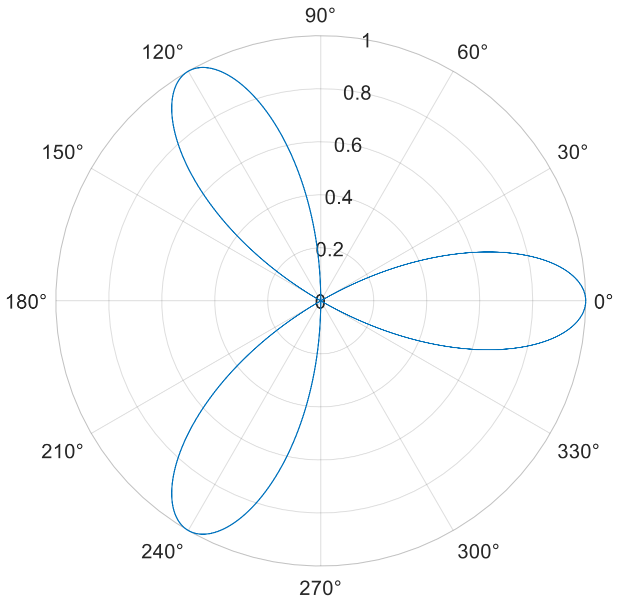
Example:

```
theta = linspace(0,2*pi,100);
rho = 1.5*ones(100,1);
figure;
polarplot(theta,rho)
```



Example:

```
theta = 0:0.01:2*pi;
rho = cos(3*theta);
figure;
polarplot(theta,rho)
```

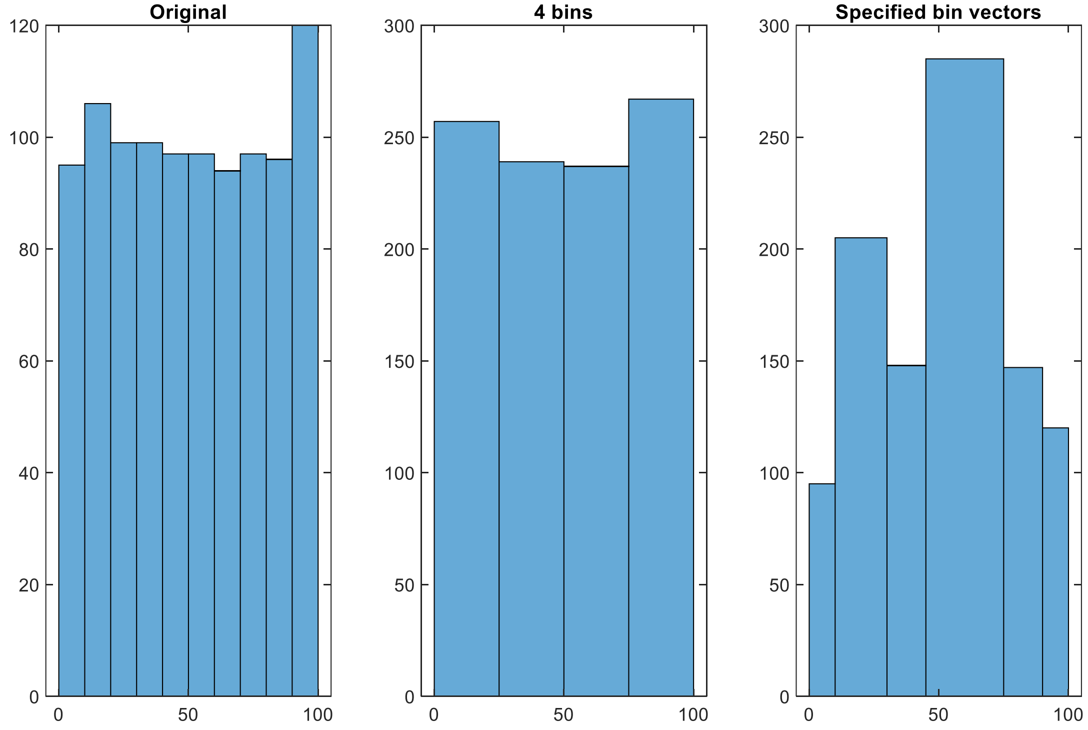


Recall: Some specialized 2D plots in MATLAB

- **Histogram plots**
  - histogram(X): create a histogram plot of X with automatic binning
  - histogram(X, nbins): specify the number of bins
  - histogram(X, edges): sort X into bins with bin edges specified in a vector

- Example:

```
% 1000 random integers between 1 and 100
Y = randi([1,100],1000,1);
figure;
subplot(1,3,1);
histogram(Y); % automatic binning
title('Original')
subplot(1,3,2);
histogram(Y,4); % specify to be 4 bins only
title('4 bins')
subplot(1,3,3);
histogram(Y,[0,10,30,45,75,90,100]); % specify the bin edges
title('Specified bin vectors')
```

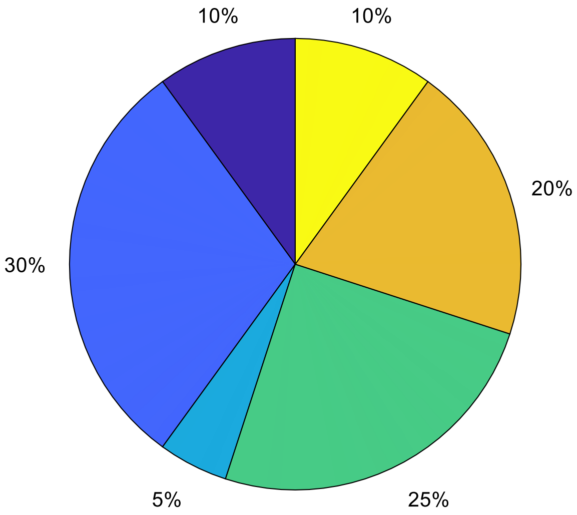


Recall: Some specialized 2D plots in MATLAB

- **Pie charts**
  - pie(X): draws a pie chart using the data in X
  - pie3(X): draws a three-dimensional pie chart using the data in X

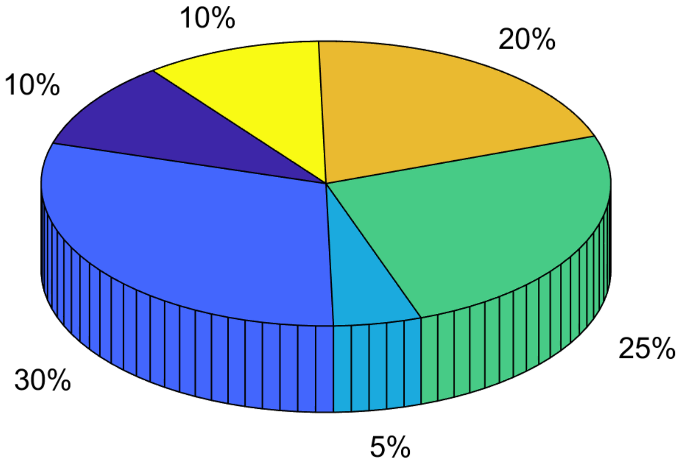
Example:

```
X = [1, 3, 0.5, 2.5, 2, 1];
figure;
pie(X)
```



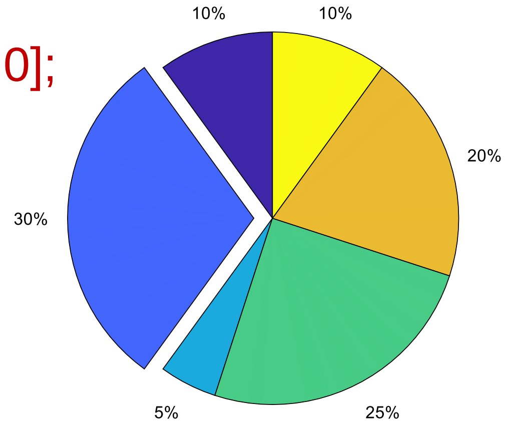
Example:

```
X = [1, 3, 0.5, 2.5, 2, 1];
figure;
pie3(X)
```



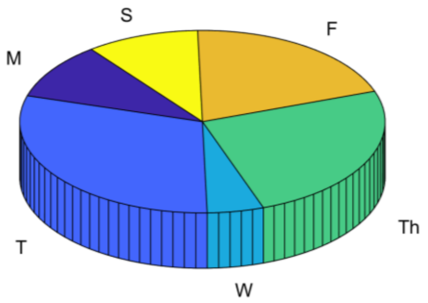
Example:

```
X = [1, 3, 0.5, 2.5, 2, 1];
explode = [0, 1, 0, 0, 0, 0];
figure;
pie(X,explode)
```



Example:

```
X = [1, 3, 0.5, 2.5, 2, 1];
labels = {'M','T','W','Th','F','S'};
figure;
pie3(X,labels)
```



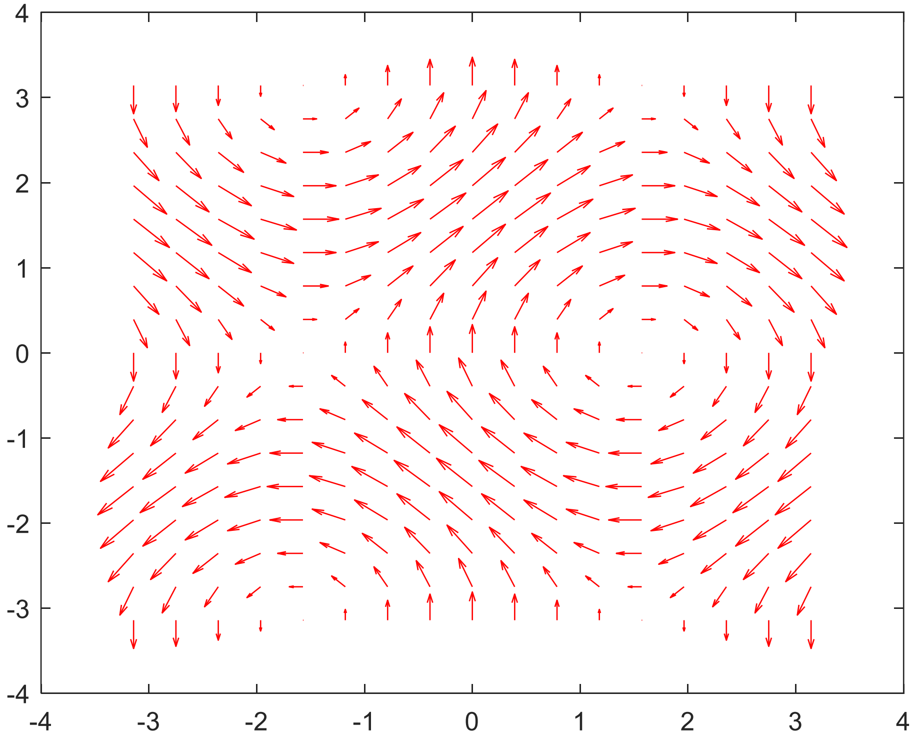
Recall: Some specialized 2D plots in MATLAB

- **Vector plots**
  - quiver(X,Y,U,V): plots arrows with directional components U and V at the Cartesian coordinates specified by X and Y

- A relevant function
  - [X,Y] = meshgrid(x,y): generate 2-D grid coordinates based on the coordinates contained in vectors x and y

Example:

```
[X,Y] = meshgrid(-pi:pi/8:pi,-pi:pi/8:pi);
U = sin(Y); % the directions
V = cos(X); % the directions
figure;
quiver(X,Y,U,V,'r')
```



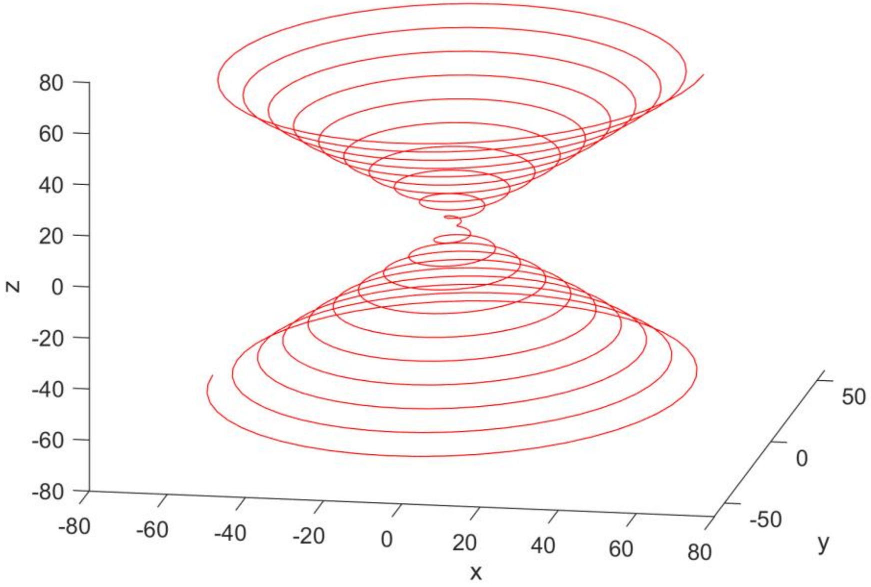
Recall: 3D plot for points and curves

- Basic syntax: plot3(x,y,z)

- Plotting style options: similar to the ones for plot
  - Changing line color, line style, marker style, line width etc. by adding specifications in the plot3 function
  - Plotting multiple curves on the same plot: use hold on
  - Changing the camera angle: view(a,b)

- Example:

```
t = linspace(-20*pi,20*pi,1000);
figure;
plot3(t.*cos(t),t.*sin(t),t, 'r-');
xlabel('x')
ylabel('y')
zlabel('z')
view(10,20);
```

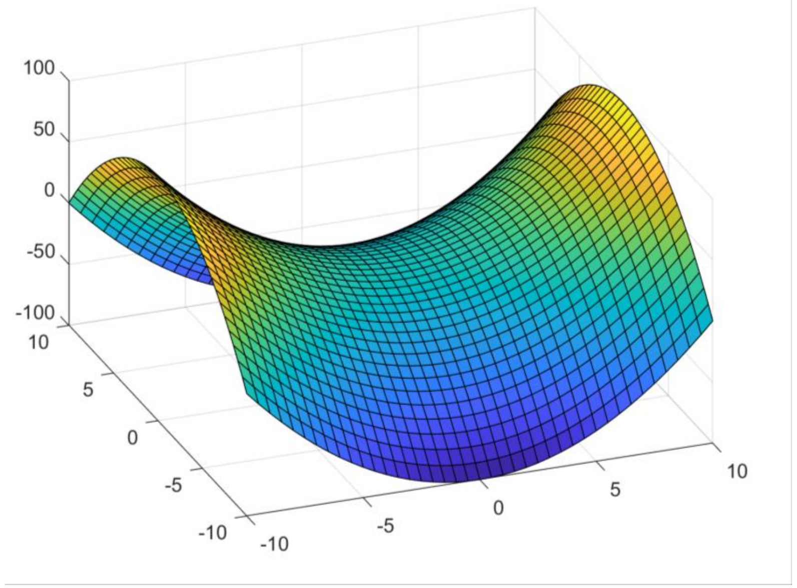


Recall: 3D plot for surfaces

- Basic syntax: surf(X,Y,Z)
  - creates a three-dimensional surface plot based on the 2-D arrays X, Y, Z (usually generated from meshgrid)

Example: hyperbolic paraboloid

```
x = -10:0.5:10;
y = -10:0.5:10;
[X,Y] = meshgrid(x,y);
Z = X.^2-Y.^2;
figure;
surf(X,Y,Z);
```

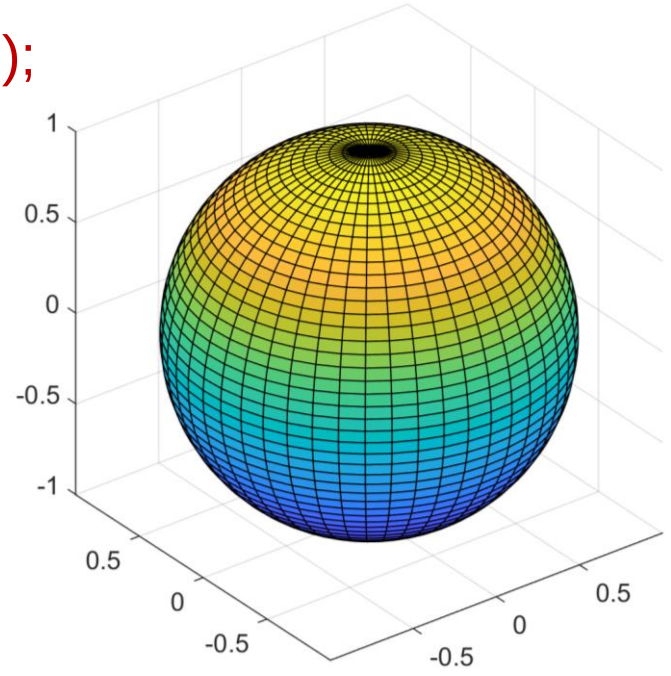


Example (parametric surface): unit sphere

$(X(u,v),Y(u,v),Z(u,v)) = (\cos u \cos v\,, \cos u \sin v\,, \sin u)$

with $-\frac{\pi}{2} \leq u \leq \frac{\pi}{2}, -\pi \leq v \leq \pi$

```
u = linspace(-pi/2,pi/2,50);
v = linspace(-pi,pi,50);
[U, V] = meshgrid(u,v);
X=cos(U).*cos(V);
Y=cos(U).*sin(V);
Z=sin(U);
figure;
surf(X,Y,Z);
axis equal
```

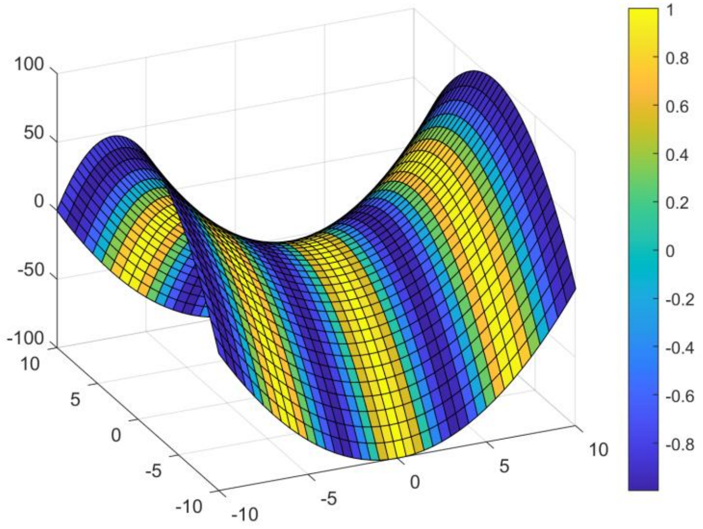


Recall: Surface plot style options

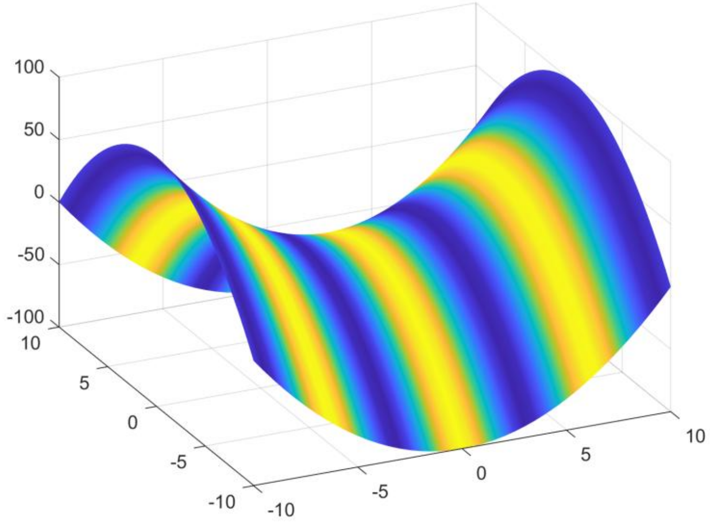
- More style options:
  - surf(X,Y,Z,C): creates a three-dimensional surface plot using X, Y, Z, where the face color is based on the C value
  - colorbar: show the color bar to see the corresponding values
  - colorbar off: close the color bar
  - EdgeColor: 'k','r', 'none' (no color) etc.
  - LineStyle: '-', '--' etc.
  - FaceColor: 'flat', 'interp' (interpolated coloring) etc.
  - FaceAlpha: a scalar in range [0,1] for face transparency

Example:

```
x = -10:0.5:10;
y = -10:0.5:10;
[X,Y] = meshgrid(x,y);
Z = X.^2-Y.^2;
C = cos(X);
figure;
surf(X,Y,Z,C);
colorbar
```

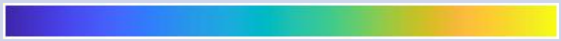
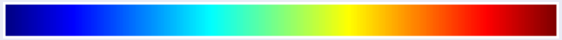
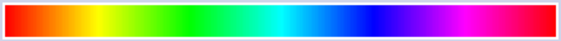
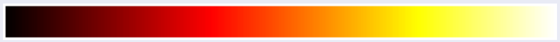
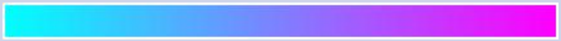
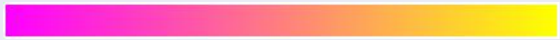
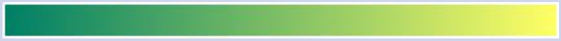
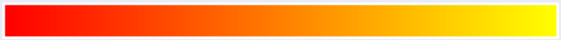
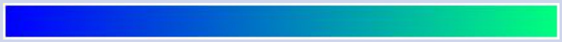
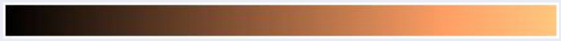
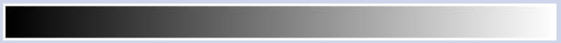


surf(X,Y,Z,C,'EdgeColor','none','FaceColor','interp');



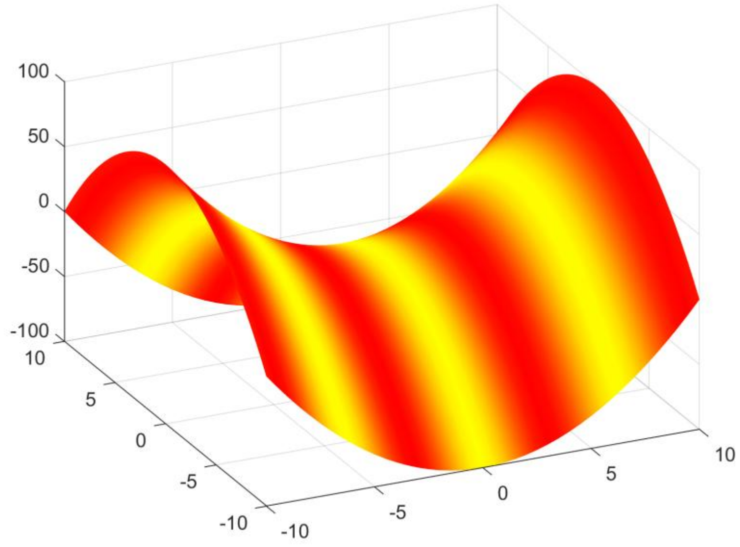
Recall: Colormap

- Built-in color schemes in MATLAB:

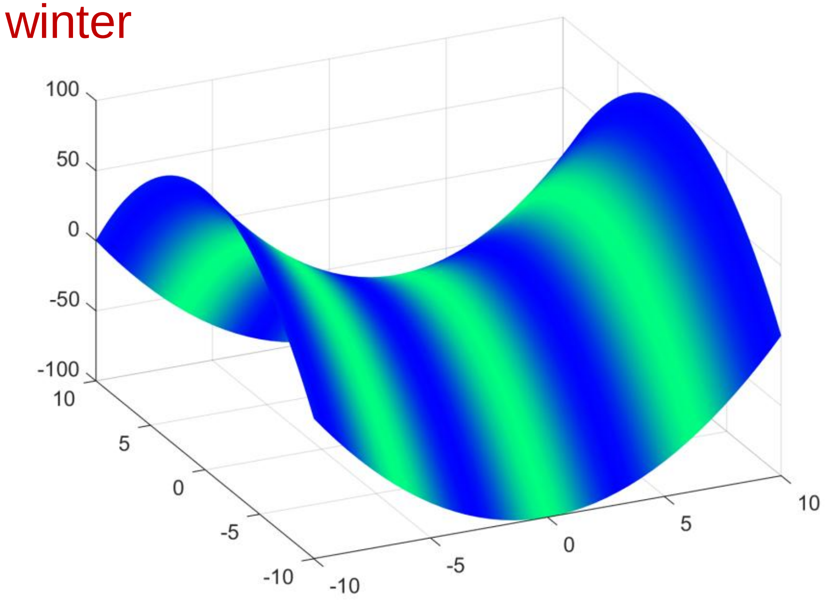
Name	Color scale
parula	
jet	
hsv	
hot	
cool	
spring	
summer	
autumn	
winter	
copper	
gray	

… and a lot more!

```
surf(X,Y,Z,C,'EdgeColor','none','FaceColor','interp');
colormap autumn
```



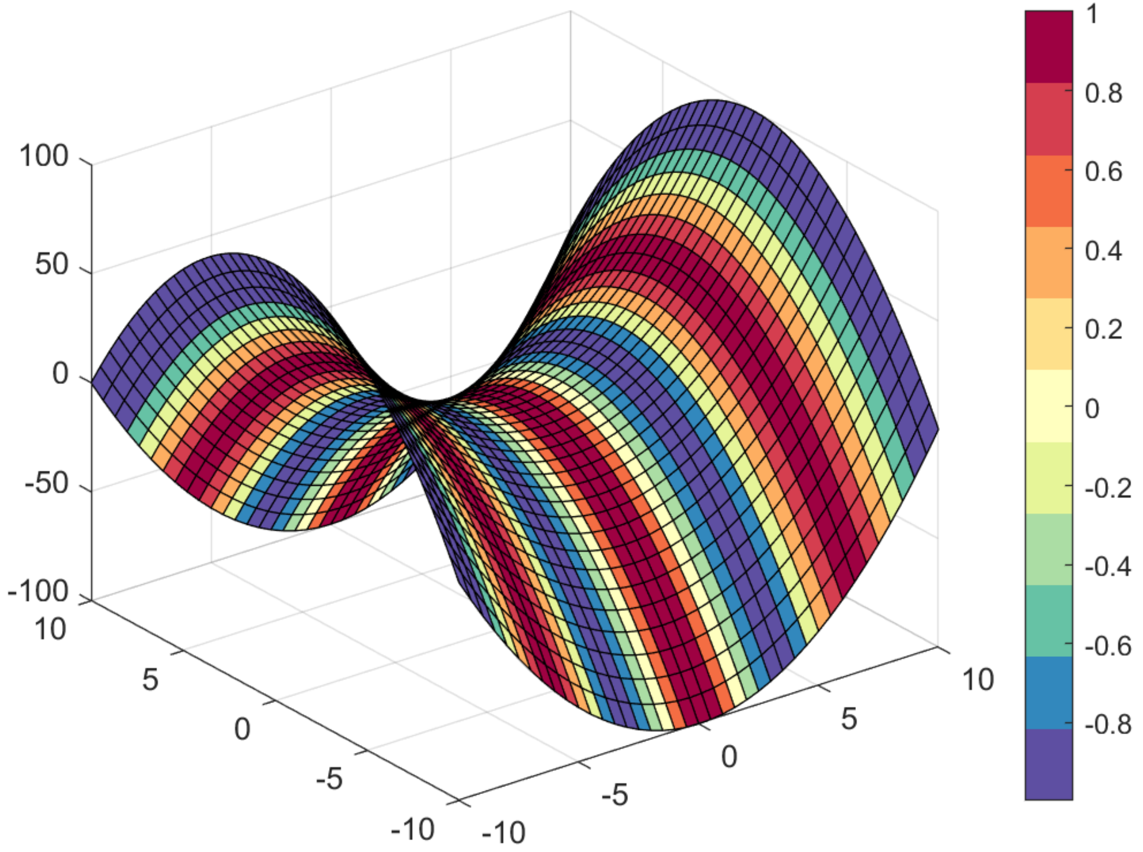
```
colormap winter
```



Miscellaneous visualization topics

- **How to create a custom colormap?**
  - Define a 3-column matrix M representing different RGB values
  - Then use colormap(M)

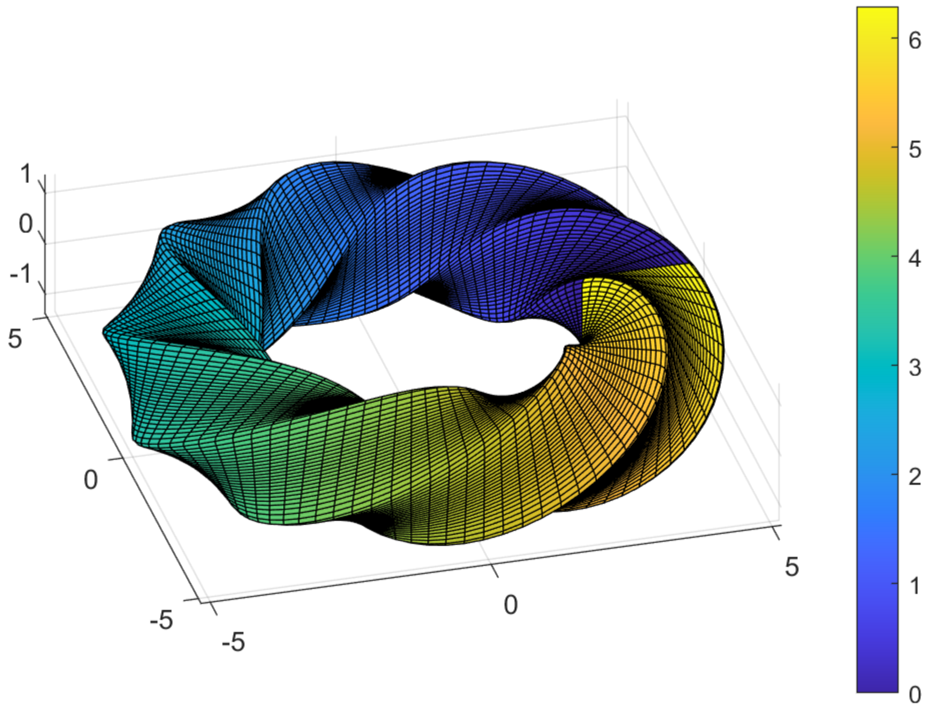
```
M = [ ...
    94    79   162;
    50   136   189;
   102   194   165;
   171   221   164;
   230   245   152;
   255   255   191;
   254   224   139;
   253   174    97;
   244   109    67;
   213    62    79;
   158     1    66  ] / 255;
colormap(M)
```

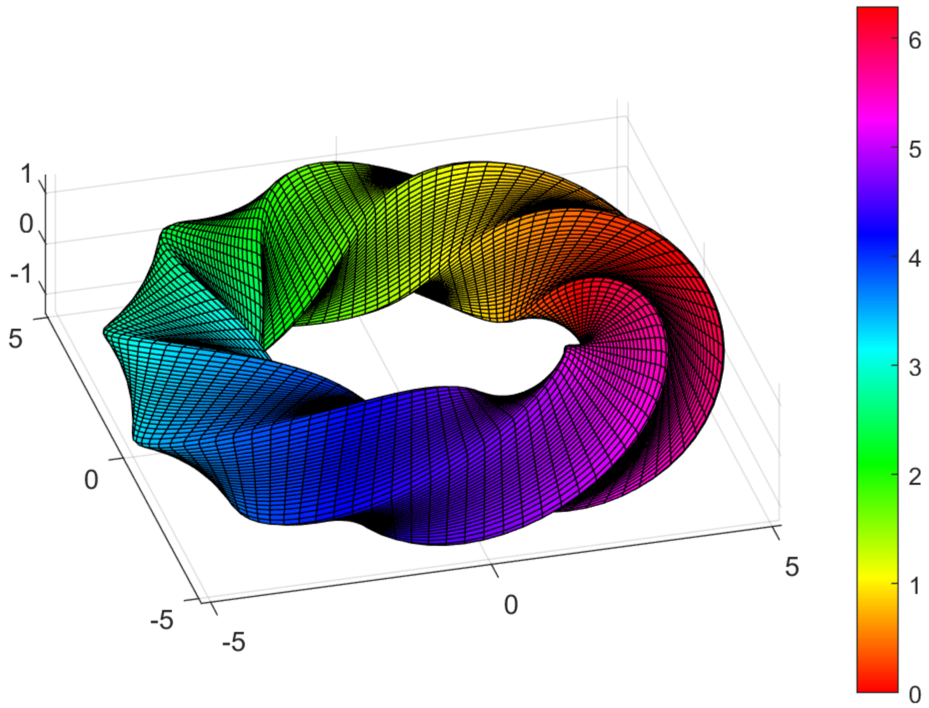


Practical considerations in choosing colormap

- Is your color data periodic?

- If so, it may be better to choose a periodic colormap
  (e.g. hsv or a custom colormap) instead of the default parula / jet

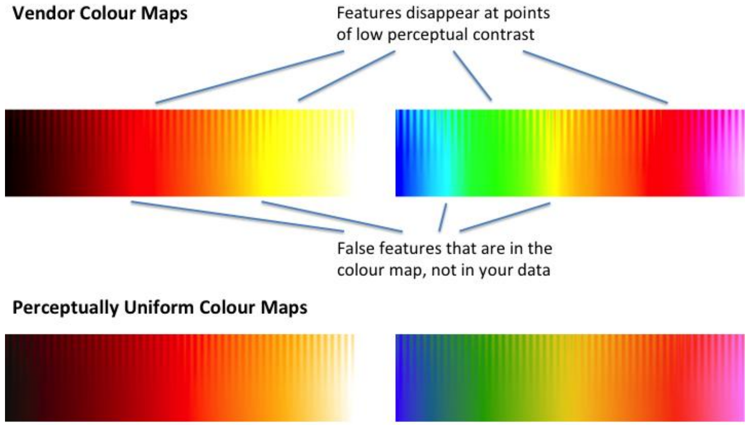


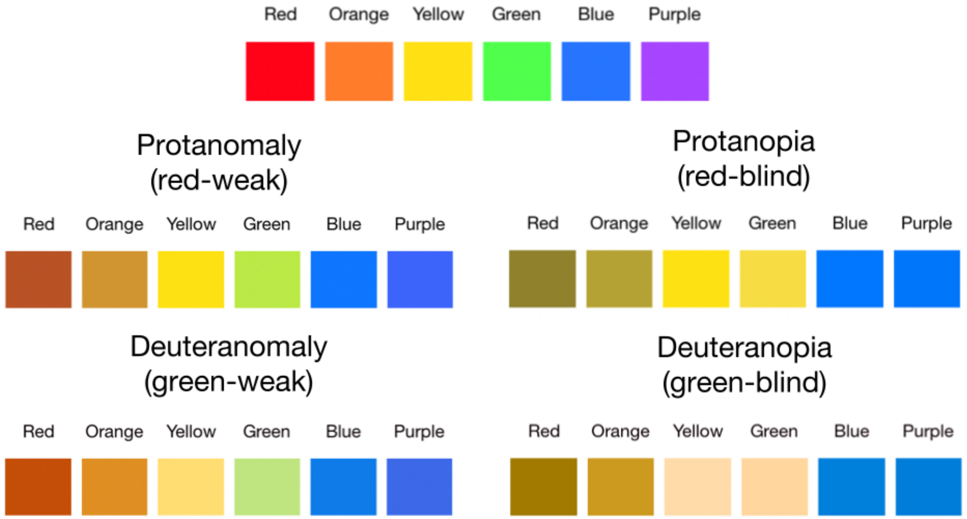


Practical considerations in choosing colormap

- Is your colormap color blind friendly?

- Is your colormap perceptually uniform?
  https://colorcet.com/


Vendor Colour Maps
Features disappear at points of low perceptual contrast
False features that are in the colour map, not in your data
Perceptually Uniform Colour Maps



- Some further discussions (not written for MATLAB, but a good general guideline)
  - Choosing Colormaps
    https://matplotlib.org/stable/users/explain/colors/colormaps.html
  - How Bad Is Your Colormap? https://jakevdp.github.io/blog/2014/10/16/how-bad-is-your-colormap/

Making your plot more three-dimensional by adding light

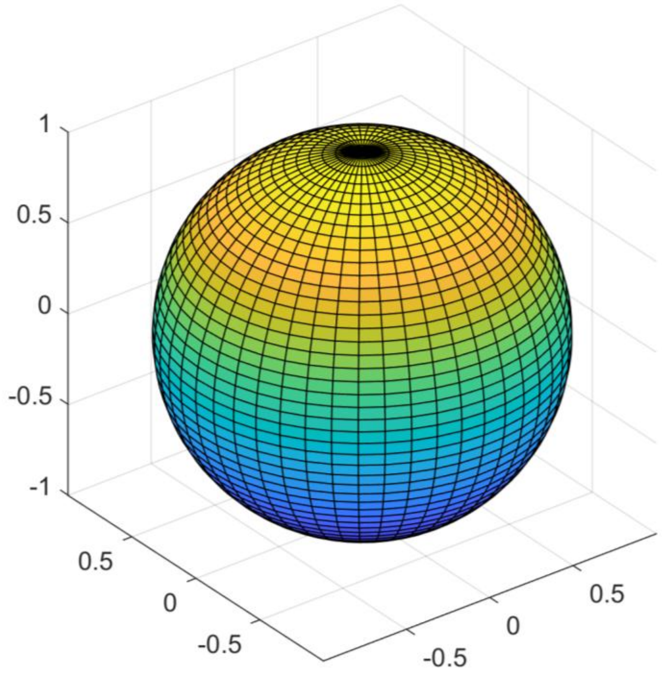
- camlight: creates a light to the right and up from the camera position.

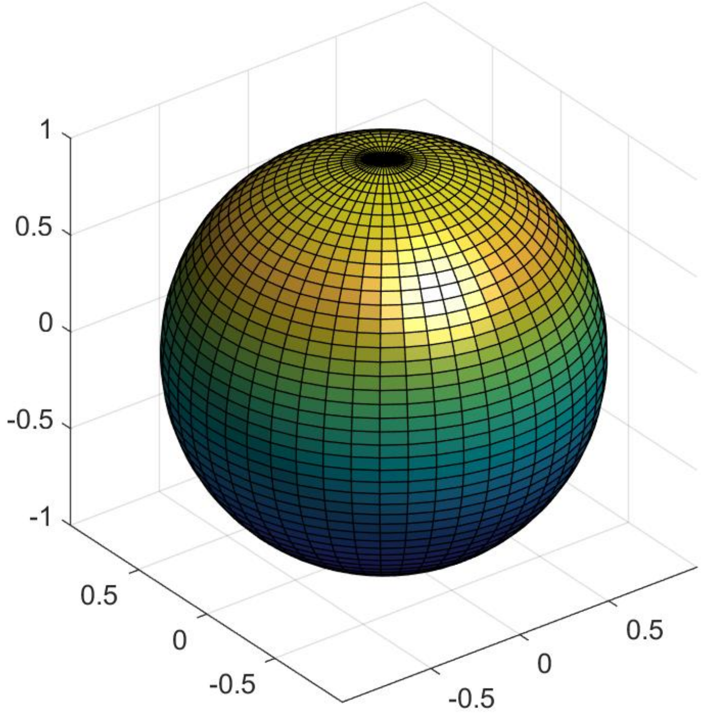
- camlight(az,el): creates a light at the specified azimuth angle az and elevation angle el

```
u = linspace(-pi/2,pi/2,50);
v = linspace(-pi,pi,50);
[U, V] = meshgrid(u,v);

% sphere equation
X=cos(U).*cos(V);
Y=cos(U).*sin(V);
Z=sin(U);

figure;
surf(X,Y,Z);
axis equal
camlight
```



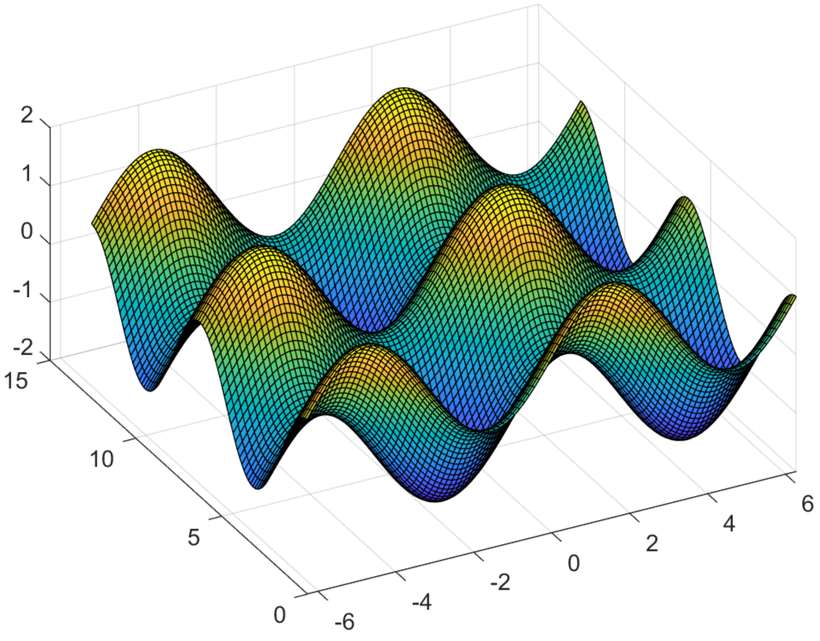


Other 3D plot functions

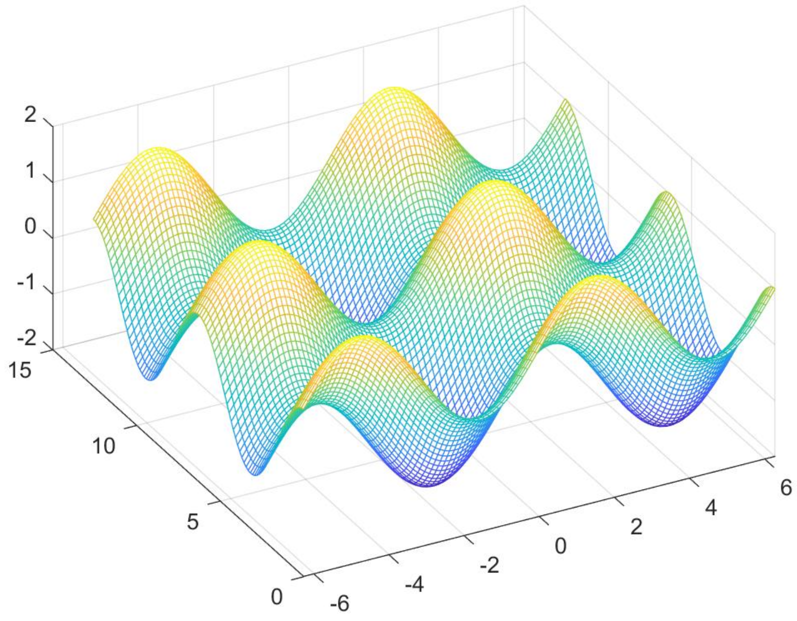
- mesh(X,Y,Z): mesh plot, with solid edge colors and no face colors
- contour(X,Y,Z): contour lines

```
x = linspace(-2*pi,2*pi);
y = linspace(0,4*pi);
[X,Y] = meshgrid(x,y);
Z = sin(X)+cos(Y);
```

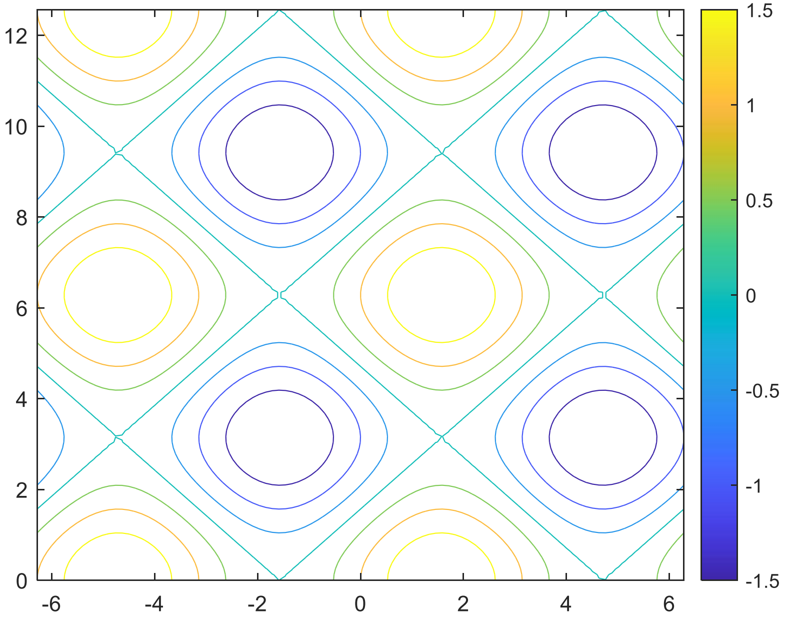
surf(X, Y, Z)



mesh(X, Y, Z)



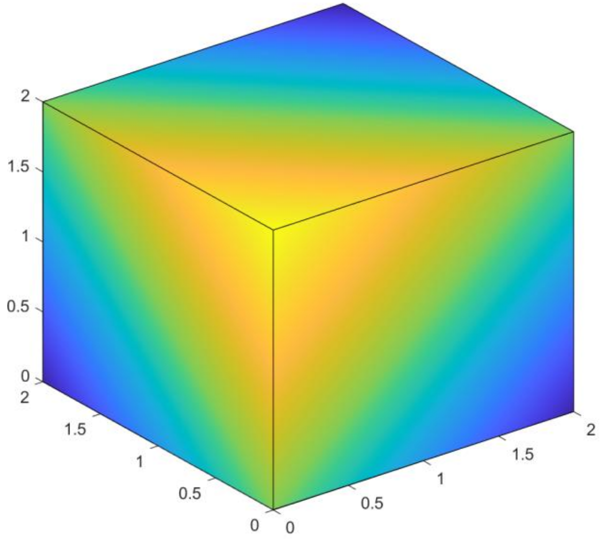
contour(X, Y, Z)



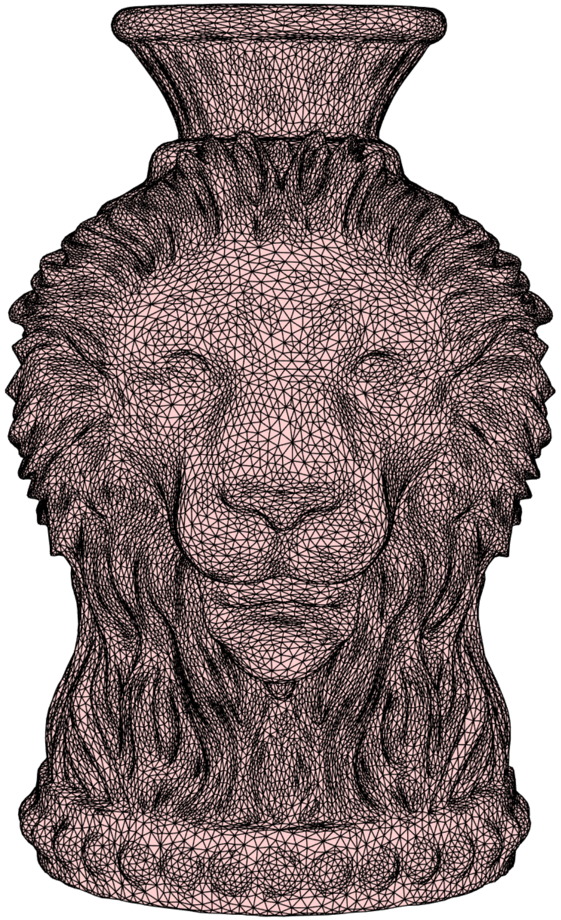
Other 3D plot functions

- Plotting more general polygons (or collection of polygons):
  - fill
  - fill3
  - patch

fill3(x,y,z,c)



patch('Faces',f,'Vertices',v);



patch('Faces',f,'Vertices',v);
camlight;

Review: Basic operations for creating scalars/vectors/matrices

- Scalars:
  - a = 1;
  - b = a*2.5 + 3;

- Vectors:
  - Row vector: u = [1, pi, -2.34];
  - Column vector: w = [2; 3; 5; 8];
  - v = 1:0.5:10;
  - t = linspace(3,10,200);

- Matrices:
  - A = [1, 2, 3; 4, 5, 6];
  - B = zeros(3,4);
  - C = ones(2,2);
  - D = eye(3);

Review: Some common MATLAB commands

- Information/management:
  - Checking MATLAB documentation: help, demo, …
  - Management: clear x, clear all, clc, close all, …
  - Suppressing output: “ ; ”
  - Separating commands into multiple lines: “ … ”
  - Adding comments: “ % ”
  - disp
  - input
  - tic; …; toc

- Mathematical functions/constants:
  - Special numbers: pi, 1i, Inf, NaN (relevant functions: isfinite, isnan)
  - Basic functions: sqrt, exp, log, log10, abs, sin, cos, tan, mod, rand, randi, max, min, mean, median, …

Review: Vector and matrix operations

- Empty matrix: A = [ ]; (relevant function: isempty(A))

- Extracting matrix entries:
  - Extracting an entry: A(1,2)
  - Extracting a submatrix: A(1:2, 3:5), A(2:end, 1), A(:, 1:2:end), …

- Modifying an existing matrix:
  - Changing some entries: B(1,3) = 0, B([1,3],[2,4]) = [0.5, pi; -1, 3], …
  - Deleting some rows/columns: B(2,:) = [ ], B(:, [1, 3, 4]) = [ ]

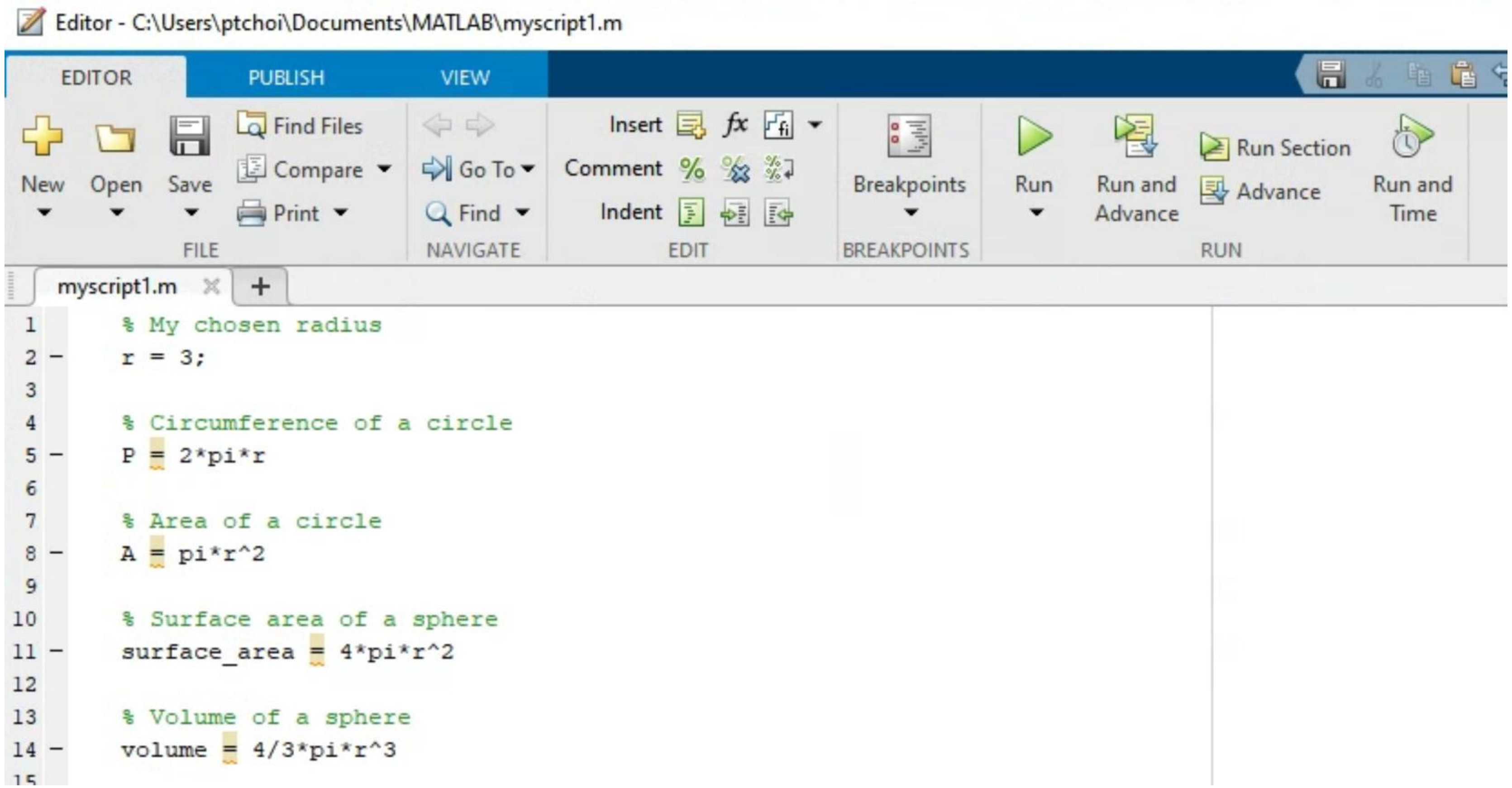
- Creating a new matrix:
  - Direct input: A = [1, 2, 3; 4, 5, 6]
  - Special matrix commands: zeros, ones, eye, …
  - Concatenation: A = [B; C], D = [zero(2,2), ones(2,1)], …

Review: Vector and matrix operations

- Getting the length or size: length(v), size(A)
- Transpose/flipping: A', flipud(A), fliplr(A)
- Reshaping a matrix: reshape(A,m,n)
- Finding certain indices and values: find(A==0.5), ismember(A,B)
- Extracting diagonal/triangular parts: diag, triu, tril
- Solving matrix equation (backslash operator): x = A\b
- Sum of entries of A along columns or rows: sum(A,1) (columns) , sum(A,2) (rows)
- Entrywise operations: A.*B, A./B, A.^B, …

Review: Writing MATLAB scripts

- Writing a MATLAB script:
  - A file with a .m extension
  - Can execute a series of MATLAB statements
  - Save, edit and debug easily



```
% My chosen radius
r = 3;

% Circumference of a circle
P = 2*pi*r

% Area of a circle
A = pi*r^2

% Surface area of a sphere
surface_area = 4*pi*r^2

% Volume of a sphere
volume = 4/3*pi*r^3
```

Review: Writing MATLAB functions

- Writing a MATLAB function:
  - A .m file in a specific format:

    ```
    function [y1,...,yn] = myfun(x1,...,xm)
            …
            …
    end
    ```

  - Can handle complicated tasks inside the function
  - Only the output variables y1,...,yn will be stored in the workspace
  - Can be reused in other functions and scripts

- Creating a function handle using @
  - e.g. `f = @(x,y) 2*x+3*y;`
  - Simpler creation procedure
  - Only defined and used within the current workspace

Review: Relational and logical operators

- The following operators return a logical value or an array of logical values with
  - Logical value 1 (true)
  - Logical value 0 (false)

Description	Command
Equal to (The = character is for assignment, whereas the == character is for comparing the elements in two arrays.)	==
Not equal to	~=
Greater than	>
Greater than or equal to	>=
Less than	<
Less than or equal to	<=

Description	Command
AND	&
OR	|
NOT	~
Short-circuit AND For A && B, MATLAB does not evaluate condition B at all if condition A is false	&&
Short-circuit OR For A || B, MATLAB does not evaluate condition B at all if condition A is true	||

Review: *if/elseif/else* statements

- **if-end statement:**

```
if expression
    statements                  % do this if the expression is true
end
```

- **if-else-end statement:**

```
if expression
   statements1                  % do this if the expression is true
else
   statements2                  % do this if the expression is false
end
```

- **if-elseif-else-end statement:**

```
if expression1
   statements1                  % do this if expression1 is true
elseif expression2
   statements2                  % do this if expression1 is false but expression2 is true
elseif expression3
   statement3                   % do this if expression1 and expression 2 are false but expression3 is true
else
   statementsN                   % do this if all the above expressions are false
end
```

Review: *switch/case/otherwise* statements

- **switch/case/otherwise statement:**

```
switch switch_expression

  case case_expression1
    statement1

  case case_expression2
    statement2
     ...
  otherwise
    statementN

end
```

```
% Similar to if-elseif-else-end:

% if switch_expression == case_expression1 (for scalar)
% (or if strcmp(switch_expression,case_expression1) (for string))
%       do statement1

% elseif switch_expression == case_expression2 (for scalar)
% (or elseif strcmp(switch_expression,case_expression2) (for string))
%       do statement2

% else
%         statementN
%
% end
```

Review: *for* statements (*for* loop)

- ***for* loop: loop a set of statements repeatedly**

```
for i = (a given vector)
      statements;                    % Do this for every element i in the vector
      …
end
```

- Usually used when you need to repeat some operations for a fixed number of times, e.g.
  - Construct a matrix with some given size
  - Do some update/sampling for 100 times
  - …

Review: *while* statements (*while* loop)

- ***while* loop: repeat the statements until the condition is NOT satisfied**

```
while condition
    statements          % repeating the statements while the condition is true (1)
end
```

- Usually used when you don't know how many times you need to repeat but have a known stopping criterion, e.g.
  - Find the smallest n/smallest number of steps such that …
  - Repeat the operations until the result converges (with error less than a threshold)
  - …

Review: More on conditional statements

- Nested loops

```
for i = 1:m
      statementA;                          % For every i, statementA will be run once
      for j = 1:n
            while condition
                    statementC;          % For every combination of (i,j), statementC will be run multiple times until “condition” is not
            end
      end
end
```

- Efficiency issue: loops vs vectorization

- The *break* command: terminates the execution of a *for* or *while* loop
  - Statements in the loop after the break statement do not execute.
  - In nested loops, *break* only break one loop

- The *continue* command: forces the program to skip the remaining part of the current *for/while* loop and continue with the next iteration

Review: Recursion

- Idea: Create a function that calls itself during its own execution

- The function should consist of:
  - A base case for the final step/lowest level (otherwise it may loop endlessly)
  - Some way to proceed to the next level

- Example: Compute $1^1 \cdot 2^{1/2} \cdot 3^{1/3} \cdot \dots \cdot 100^{1/100}$

Method 1 (*for* loop):

```
s = 1;
for n = 1:100
        s = s*n^(1/n);
end
s
```

Method 3 (vectorized):

```
v = 1:100;
s = prod(v.^(1./v))
```

Method 2 (*while* loop):

```
s = 1;
n = 1;
while n <= 100
        s = s*n^(1/n);
        n = n+1;
end
s
```

Method 4 (recursion):
First create a function:

```
function s = myproduct(n)
if n <= 1
        s = 1;
else
        s = n^(1/n) * myproduct(n-1);
end
end
```

Then we can run the following command:

```
>> myproduct(100)
```

Review: 2D visualization using the plot function

- Basic command:
  - figure;
    plot(x,y,…)

- Different plot styles:
  - Changing the marker style ('o', 's', '^',…) and line style ('-', '--',…)
  - Changing the line color, line width, marker edge color, marker face color etc.

- Figure management:
  - hold on, hold off
  - subplot(m,n,p)
  - axis (on, off, equal, tight)
  - xlim, ylim, axis([xmin, ymin, xmax, ymax])

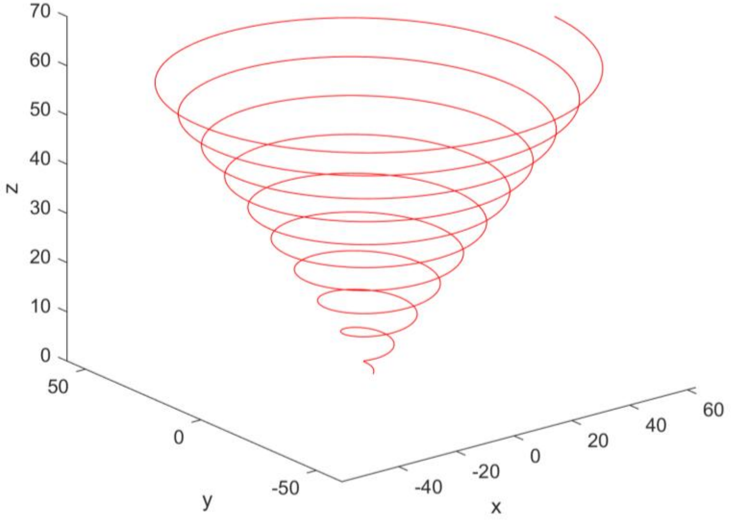
- Annotation/label:
  - xlabel, ylabel, title, legend, grid (on/off), text, …

Review: Other 2D plots

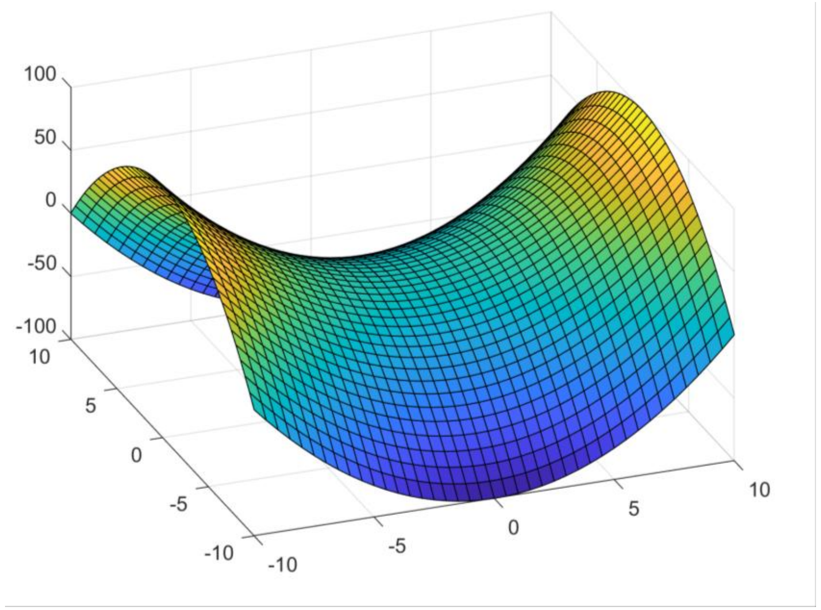
- Log-log plot: loglog(x,y)
- Semi-log plot: semilogx(x,y), semilogy(x,y)
- Polar plot: polarplot(theta,rho)
- Histogram plot: histogram(X), histogram(X,nbins), histogram(X,edges)…
- Bar chart: bar(X), bar3(X), …
- Vector plot: quiver(X,Y,U,V)
- (Many other functions not covered: fill, contour, …)

Review: 3D visualization

- Plotting points and curves in 3D
  - plot3(x,y,z)
  - x,y,z: vectors
  - Plotting style options: similar to the ones for plot
  - Changing the camera angle: view(a,b)



- Plotting surfaces in 3D:
  - surf(X,Y,Z)
  - X, Y, Z: 2D arrays, usually generated with the aid of the meshgrid function
  - Plotting style options:
    - surf(X,Y,Z,C,…)
    - Changing edge color, edge width etc.
    - Changing colormap



- (Many other functions not covered: custom colormap, camlight, …)

Reminder: Test 1 this week

January

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	[28]	[29]	[30]	[31]	

February

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						[1]
[2]	[3]	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	1

March

Sun	Mon	Tue	Wed	Thu	Fri	Sat
2	[3]	[4]	[5]	[6]	[7]	[8]
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

April

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17		

🟧	**Lecture 1- Lecture 13**
🟨	**Lab 1 - Lab 10 (40%)**
🟩	**Test 1 (30%)** **Test 2 (30%)**

Reminder: Test 1 this week

- **90-minute test**, take place during the usual lab session
  - (Class A) 10:30 - 12:00
  - (Class B) 12:30 - 14:00
  - (Class C) 14:30 - 16:00
  - You must attend your registered session but not the other MATH2221 sessions. Attending the wrong session will result in 0 marks for the assessment.
- Please arrive 10 minutes early.
- Writing part + Coding part
- Additional 10 minutes for uploading all files to Blackboard after the test
- Coverage: Lecture 1-6 and Lab Assignment 1-5

Reminder: Test 1 this week

- **Open-note** in the sense that you may access our lecture notes and lab assignment solutions, which will be available in a designated folder on the desktop computer.
- Other reference books/materials are NOT allowed.
- Other printed notes/tablets/phones/calculators are NOT allowed.
- The test is NOT open-internet.
- NO discussions are allowed during the test.

Thank you!

Next time:

- Advanced linear algebra functions in MATLAB