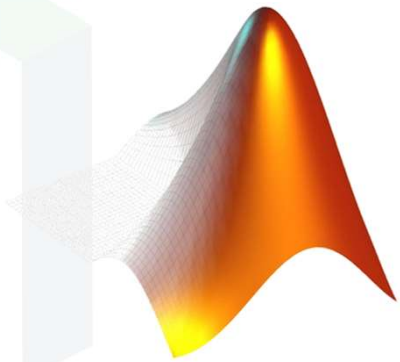
EMPossible

Computational Science:
Computational Methods in Engineering

Introduction to MATLAB & Graphics

Outline

- MATLAB
- Figures and handles
- 1D plots
- 2D graphics
- Creating movies in MATLAB
- String manipulation and text files
- Helpful tidbits



This lecture is NOT intended to teach the basics of MATLAB. Instead, it is intended to summarize specific skills required for this course to a student already familiar with MATLAB basics and programming.

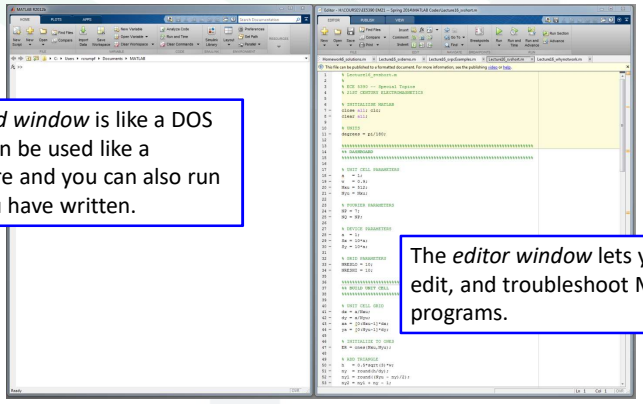
MATLAB Concepts

Key MATLAB Concepts to Learn

- MATLAB interface
  - Editor window vs. command window
  - Figure windows
- MATLAB programming
  - Scripts vs functions
  - Variables and arrays
  - Generating and manipulating arrays
  - Basic commands: for, while, if, switch
  - Basic graphics commands: figure, plot

MATLAB Interface

MATLAB has three main components: (1) command window, (2) m-file editor, and (3) Simulink. We will not use Simulink so we are only concerned with…



The *command window* is like a DOS prompt. It can be used like a calculator here and you can also run programs you have written.

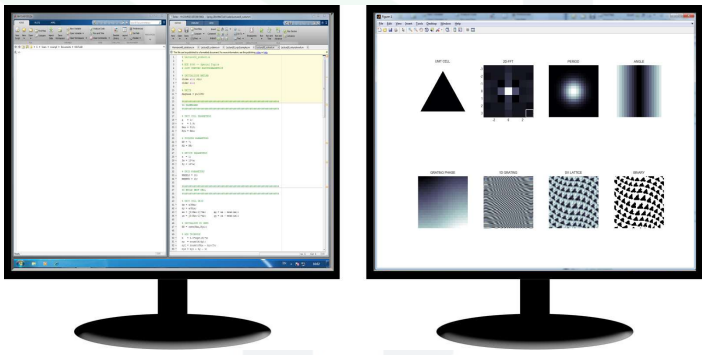
The *editor window* lets you write, edit, and troubleshoot MATLAB programs.

EMPossible

My Preferred Dual-Monitor Setup

On one monitor has my command and editor windows.

I create full-screen figure window on a second monitor for graphics and visualizations.



This way I can see all the information at the same time.

EMPossible

Scripts Vs. Functions

Script Files

Instead of typing all the commands at the command prompt, you can type them into a text file and then run the code when you are done. MATLAB behaves just like you typed the commands at the command prompt, but scripts let you go back and edit what you have done.

- Script files share memory and variables with the command window.
- Unless you know better, script files must be initialized.
- Variables are easily accessed at the command prompt for troubleshooting.

Functions

Programs can be written into functions so that they have defined inputs and outputs, like the function `y = cos(x)`. These do not share memory or variables with anything else, except for what is defined at input or output.

- Functions do not share memory or variables with the command window.
- As far as the function knows, memory is cleared when it is called except for the input variables.
- You cannot access variables inside functions for troubleshooting.
- **Do not overwrite input arguments!**

File Names

- File names cannot contain spaces.

  `double matrix.m` should be `double_matrix.m`

- Functions are called by their filename, not the name given in the code. It is best practice to save the file with the same name as the given function name in the code.

  `function y = dumbfunc(x)`
  should be named `dumbfunc.m`

How to Learn MATLAB

Tutorials

- Search the internet for different tutorials.

Be sure you know and can implement everything in this lecture.

Practice. Practice. Practice.

For Help in MATLAB

- The Mathworks website is very good!
  - http://www.mathworks.com/help/matlab/index.html
- Help at the command prompt
  - ">> help *command*"
- Dr. Rumpf's helpdesk
  - rcrumpf@utep.edu

Figures & Handles

Graphics Handles

Essentially every graphical entity in MATLAB has a handle associated with it. This handle points to all their properties and attributes which can be changed at any time after the graphical entity is generated.

```
h = figure;
h = plot(x,y);
h = line(x,y);
h = text(x,y,'hello');
h = imagesc(x,y,F);
h = pcolor(x,y,F);
h = surf(x,y,F);
  .
  .
  .
```

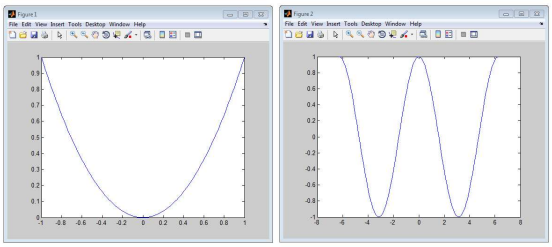
Here `h` is the handle returned by these graphics calls.

EMPossible

The Figure Window

All graphics are drawn to the active figure window.
There can be more than one figure window, but only one is active at a time..

```
fig1 = figure;
fig2 = figure;

figure(fig1);
plot(x1,y1);

figure(fig2);
plot(x2,y2);
```



This code opens two figure windows with handles `fig1` and `fig2`. It then plots `x1` vs. `y1` in the first figure window and `x2` vs. `y2` in the second figure window. It is possible to then go back to `fig1` and anything else.

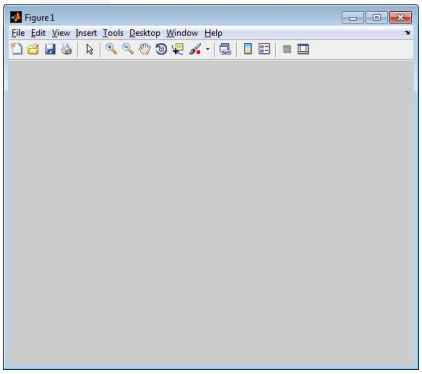
EMPossible

Slide 13

Investigating Graphics Properties (1 of 2)

To see all the properties associated with a graphics entity and their *current* values, type `get(h)` at the command prompt.

```
>> fig1 = figure;
>> get(fig1)
        Alphamap = [ (1 by 64) double array]
        CloseRequestFcn = closereq
        Color = [0.8 0.8 0.8]
        .
        .
        .
        Visible = on
```



To get the value of a single property:

```
>> c = get(fig1,'Color');
>> c

c =

    0.8000    0.8000    0.8000
```

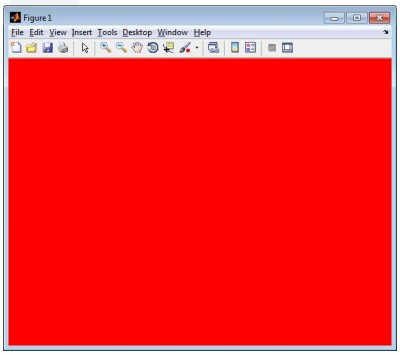
EMPossible

Slide 14

Investigating Graphics Properties (2 of 2)

To see all the properties associated with a graphics entity and their *possible* values, type `set(h)` at the command prompt.

```
>> fig1 = figure;
>> set(fig1)
       DockControls: [ {on} | off ]
       IntegerHandle: [ {on} | off ]
       InvertHardcopy: [ {on} | off ]
       .
       .
       .
       Visible: [ {on} | off ]
```



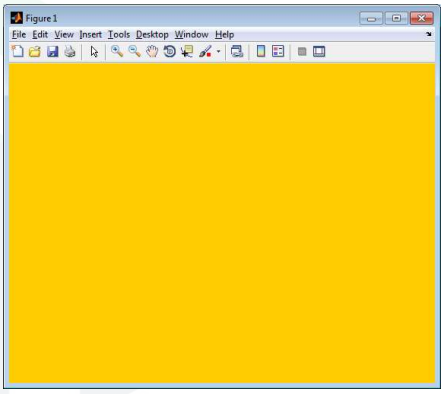
The `set()` command is what is used to change graphics properties.

```
>> set(fig1,'Color','r');
```


Changing the Background Color

red green blue

```
>> c = [1.0 0.8 0.0];
>> set(fig1,'Color',c);
```

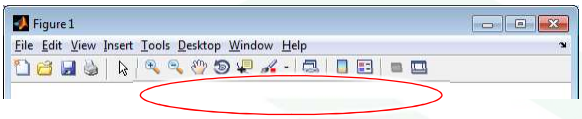


I almost exclusively use white as the background so that it is easier to paste the graphics in a paper/publication that has a white background.

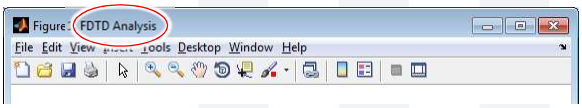
```
>> c = [1 1 1];
>> set(fig1,'Color',c);
```

Changing the Figure Name

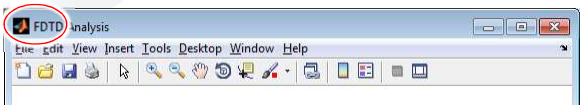
```
>> fig1 = figure('Color','w');
```




```
>> set(fig1,'Name','FDTD Analysis');
```



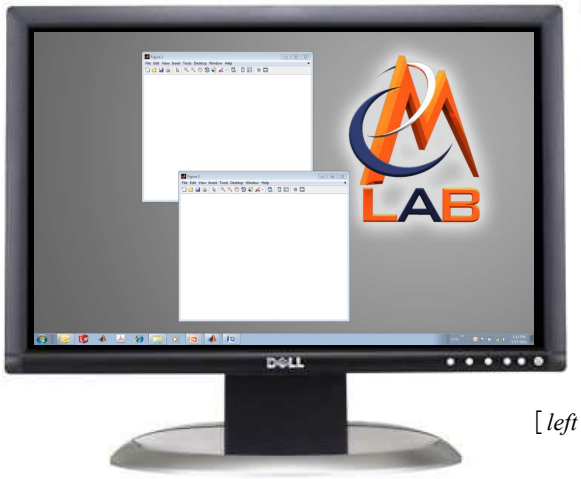

```
>> set(fig1,'NumberTitle','off');
```



EMPossible

Changing the Figure Position

```
>> fig1 = figure('Color','w','Position',[371 488 560 420]);
>> fig2 = figure('Color','w','Position',[494 87 560 420]);
```



$[left\ \ bottom\ \ width\ \ height]$

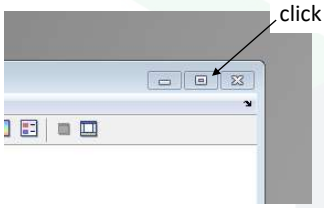
EMPossible

Full Screen Figure Window

Step 1: Open a figure window.

```
>> fig1 = figure;
```

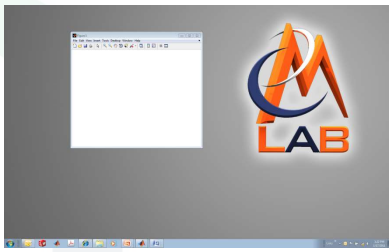
Step 2: Maximize figure window



Step 3: Use `get(fig1)` to copy figure position

```
>> get(fig1)
        …
        Position = [1 41 1680 940]
        …
```

copy this



Step 4: Paste into command in code.

```
fig1 = figure('Color','w',…
'Position',[1 41 1680 940]);
```

EMPossible

Auto Full Screen Window

Using “normalized units,” we can easily open a figure window to be full screen.

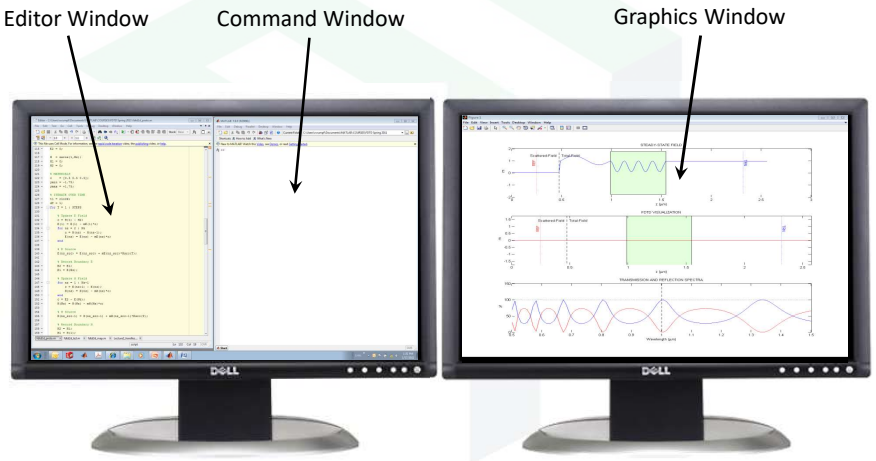
```
figure('units','normalized','outerposition',[0 0 1 1]);
```

We can do the same to open a full screen window on a second monitor.

```
figure('units','normalized','outerposition',[1 0 1 1]);
```

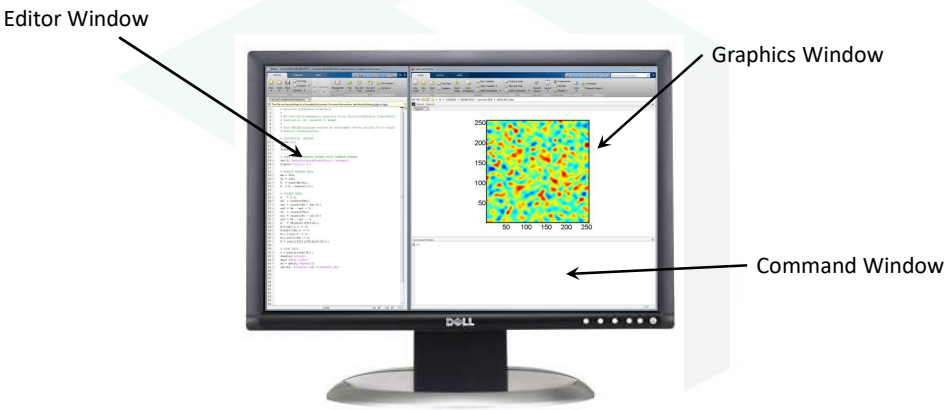
EMPossible

How I Like to Arrange My Windows



EMPossible

MATLAB Setup for a Single Monitor




```
% OPEN FIGURE WINDOW DOCKED WITH COMMAND WINDOW
set(0,'DefaultFigureWindowStyle','docked');
figure('Color','w');
```

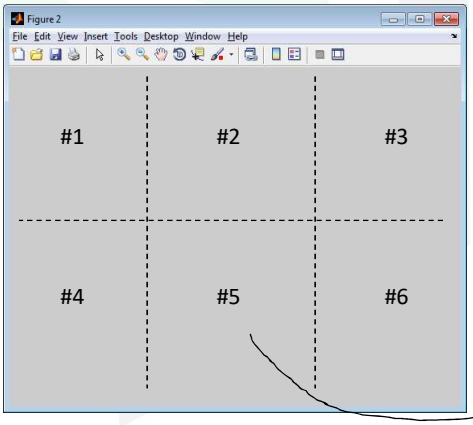
EMPossible

Subplots

MATLAB can show more than one diagram in a single figure window.

```
subplot(M,N,p);
```

(# rows, # columns, subplot #)



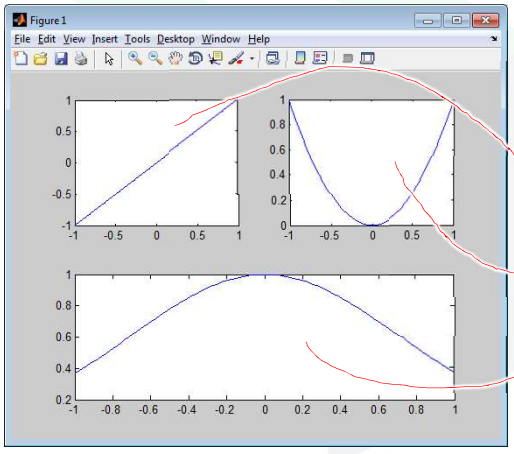
2 rows
3 columns

```
subplot(2,3,5);
```

EMPossible

Non-Uniform Partitioning

Figure windows can be partitioned non-uniformly.

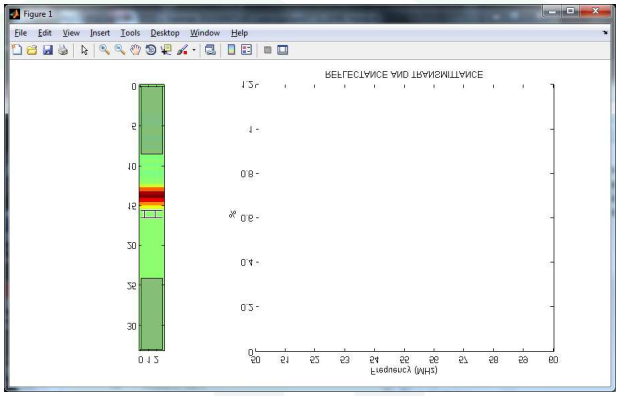



```
x  = linspace(-1,1,100);
y1 = x;
y2 = x.^2;
y3 = exp(-x.^2);

subplot(2,2,1);
plot(x,y1);

subplot(2,2,2);
plot(x,y2);

subplot(2,2,[3:4]);
plot(x,y3);
```

EMPossible

A Problem with Graphics Rendering

Some versions of MATLAB have a known problem with some ATI graphics devices.



One solution is to switch to the OpenGL renderer by: `opengl('software')`

This also makes graphics rendering much faster! ☺

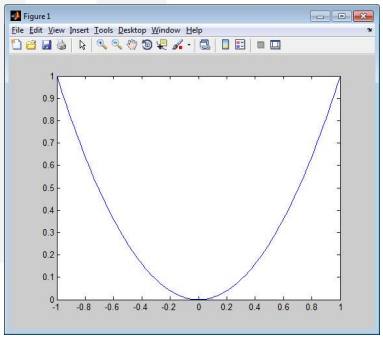
EMPossible

1D Plots

The Default MATLAB Plot

```
x = linspace(-1,1,100);
y = x.^2;
plot(x,y);
```

Things I don't like:
- Background doesn't work well.
- Lines are too thin
- Fonts are too small
- Axes are not labeled.

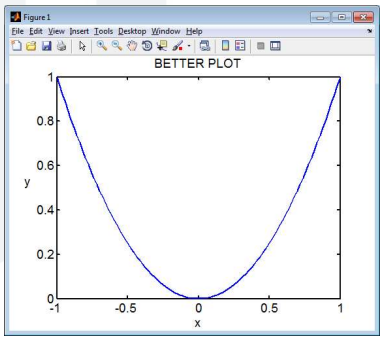


EMPossible

Revised Code for Better Plots

```
x = linspace(-1,1,100);
y = x.^2;

figure('Color','w');
h = plot(x,y,'-b','LineWidth',2);
h2 = get(h,'Parent');
set(h2,'FontSize',14,'LineWidth',2);
xlabel('x');
ylabel('y   ','Rotation',0);
title('BETTER PLOT');
```

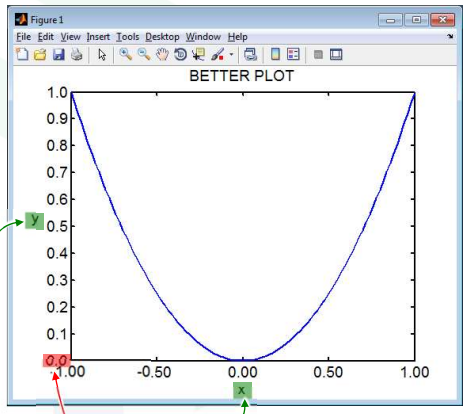
Things I still don't like:
- Uneven number of digits for axis tick labels.
- Too coarse tick marks along *x* axis.



EMPossible

Improving the Tick Marking

```
% Plot Function
fig = figure('Color','w');
h = plot(x,y,'-b','LineWidth',2);

% Set Graphics View
h2 = get(h,'Parent');
set(h2,'FontSize',14,'LineWidth',2);
xlabel('x');
ylabel('y   ','Rotation',0);
title('BETTER PLOT');

% Set Tick Markings
xm = [-1:0.5:+1];
xt = {};
for m = 1 : length(xm)
    xt{m} = num2str(xm(m),'%3.2f');
end
set(h2,'XTick',xm,'XTickLabel',xt);

ym = [0:0.1:+1];
yt = {};
for m = 1 : length(ym)
    yt{m} = num2str(ym(m),'%2.1f');
end
set(h2,'YTick',ym,'YTickLabel',yt);
```



Variables should be italicized.

This should be just "0."

Since it overlaps with the "-1.00," it may be best to just skip the "0."

EMPossible

Summary of Format String

String Constant

`%s`

Decimal Number

`%d`

Double Fixed-Point Number

`%f` `%m.nf`

m = total number of characters
n = number of digits after decimal.

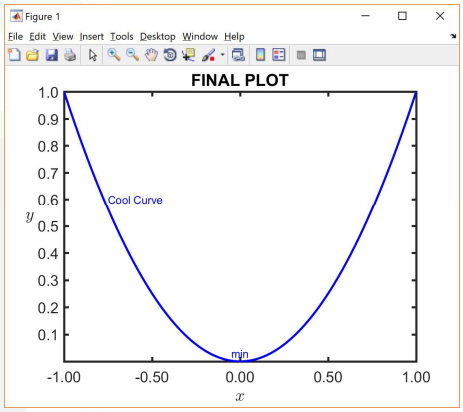
Examples

```
sprintf('%0.5g',(1+sqrt(5))/2)       % 1.618
sprintf('%0.5g',1/eps)               % 4.5036e+15
sprintf('%15.5f',1/eps)              % 4503599627370496.00000
sprintf('%d',round(pi))              % 3
sprintf('%s','hello')                % hello
sprintf('The array is %dx%d.',2,3)   % The array is 2x3.
```

EMPossible

Final Plot

```
% Plot Function
fig = figure('Color','w');
h = plot(x,y,'-b','LineWidth',2);

% Set Graphics View
h2 = get(h,'Parent');
set(h2,'FontSize',14,'LineWidth',2);
xlabel('$x$','Interpreter','LaTex');
ylabel('$y$','Interpreter','Latex',...
       'Rotation',0,...
       'HorizontalAlignment','right');
title('FINAL PLOT');

% Set Tick Markings
xm = [-1:0.5:+1];
xt = {};
for m = 1 : length(xm)
    xt{m} = num2str(xm(m),'%3.2f');
end
set(h2,'XTick',xm,'XTickLabel',xt);

ym = [0.1:0.1:+1];
yt = {};
for m = 1 : length(ym)
    yt{m} = num2str(ym(m),'%2.1f');
end
set(h2,'YTick',ym,'YTickLabel',yt);

% Label Minimum
text(-0.75,0.6,'Cool Curve','Color','b','HorizontalAlignment','left');
text(0,0.03,'min','Color','b','HorizontalAlignment','center');
```



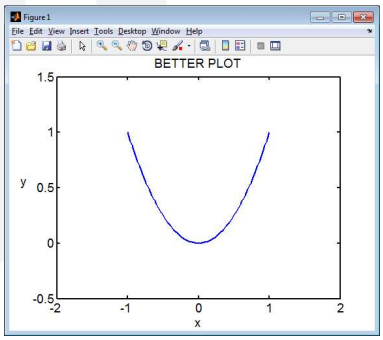
EMPossible

Setting the Axis Limits

Sometimes MATLAB will generate plots with strange axis limits.
Never depend on the MATLAB defaults for the axis limits.

```
plot(x,y,'-b','LineWidth',2);

axis([-2 2 -0.5 1.5]);

   axis([x1 x2 y1 y2]);

xlim([-2 2]);
ylim([-0.5 1.5]);

   xlim([x1 x2]);
   ylim([y1 y2]);
```

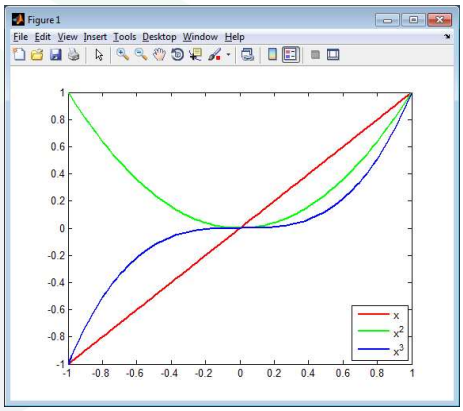
Does the same thing



EMPossible

Superimposed Plots

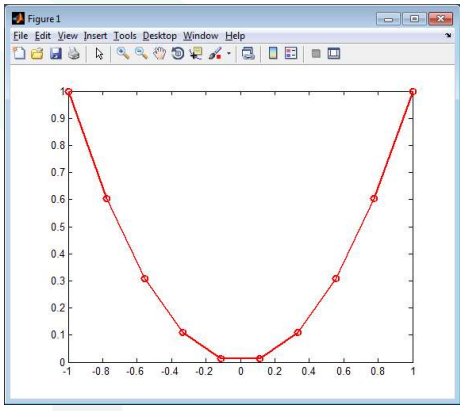
```
% Calculate Functions
x  = linspace(-1,1,100);
y1 = x.^1;
y2 = x.^2;
y3 = x.^3;

% Plot Functions
fig = figure('Color','w');
plot(x,y1,'-r','LineWidth',2);
hold on;
plot(x,y2,'-g','LineWidth',2);
plot(x,y3,'-b','LineWidth',2);
hold off;

% Add Legend
legend('x','x^2','x^3',…
       'Location','SouthEast');
```



EMPossible

Showing Where the Data Points Are

```
% Calculate Function
x = linspace(-1,1,10);
y = x.^2;

% Plot Function
fig = figure('Color','w');
plot(x,y,'o-r','LineWidth',2);
```

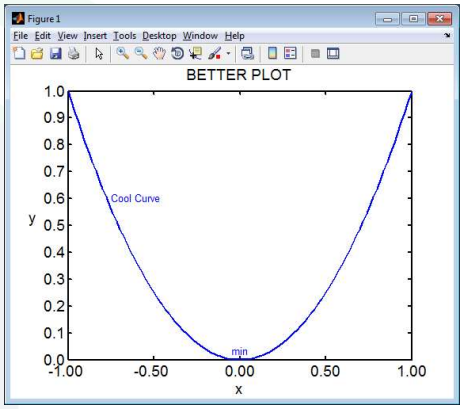


This should be standard practice when displaying measured data, or whenever only sparse data has been obtained. If at all feasible, avoid sparse data!

EMPossible

Annotating the Plot

```
% Plot Function
fig = figure('Color','w');
h = plot(x,y,'-b','LineWidth',2);

% Set Graphics View
h2 = get(h,'Parent');
set(h2,'FontSize',14,'LineWidth',2);
xlabel('x');
ylabel('y   ','Rotation',0);
title('BETTER PLOT');

% Set Tick Markings
xm = [-1:0.5:+1];
xt = {};
for m = 1 : length(xm)
    xt{m} = num2str(xm(m),'%3.2f');
end
set(h2,'XTick',xm,'XTickLabel',xt);

ym = [0:0.1:+1];
yt = {};
for m = 1 : length(ym)
    yt{m} = num2str(ym(m),'%2.1f');
end
set(h2,'YTick',ym,'YTickLabel',yt);

% Label Minimum
text(-0.75,0.6,'Cool Curve','Color','b','HorizontalAlignment','left');
text(0,0.03,'min','Color','b','HorizontalAlignment','center');
```



EMPossible

Advanced Labels

Subscripts

```
xlabel('123x_123');
xlabel('123x_{12}3');
```

$123x_{1}23$

$123x_{12}3$

Superscripts

```
xlabel('123x^123');
xlabel('123x^{12}3');
```

$123x^{1}23$

$123x^{12}3$

Special Symbols

TEX markup

LATEX markup

For more information see
http://www.mathworks.com/help/techdoc/creating_plots/f0-4741.html#f0-28104

```
xlabel('x (\mum)');
```

x (μm)

EMPossible

Common TeX Symbols

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
\alpha	α	\upsilon	υ	\sim	~	\leftrightarrow	↔
\beta	β	\phi	φ	\leq	≤	\leftarrow	←
\gamma	γ	\chi	χ	\infty	∞	\uparrow	↑
\delta	δ	\psi	ψ	\clubsuit	♣	\rightarrow	→
\epsilon	ε	\omega	ω	\diamondsuit	♦	\downarrow	↓
\zeta	ζ	\Gamma	Γ	\heartsuit	♥	\circ	∘
\eta	η	\Delta	Δ	\spadesuit	♠	\pm	±
\theta	θ	\Theta	Θ	\leftrightarrow	↔	\geq	≥
\vartheta	υ	\Lambda	Λ	\leftarrow	←	\propto	∝
\iota	ι	\Xi	Ξ	\uparrow	↑	\partial	∂
\kappa	κ	\Pi	Π	\rightarrow	→	\bullet	•
\lambda	λ	\Sigma	Σ	\downarrow	↓	\div	÷
\mu	μ	\Upsilon	Υ	\circ	∘	\neq	≠
\nu	ν	\Phi	Φ	\pm	±	\aleph	ℵ
\xi	ξ	\Psi	Ψ	\geq	≥	\wp	℘
\pi	π	\Omega	Ω	\propto	∝	\oslash	∅
\rho	ρ	\forall	∀	\partial	∂	\supseteq	⊇
\sigma	σ	\exists	∃	\bullet	•	\subset	⊂
\varsigma	ς	\ni	∋	\div	÷	\o	o
\tau	τ	\cong	≅	\neq	≠	\nabla	∇
\equiv	≡	\approx	≈	\aleph	ℵ	\ldots	…
\Im	ℑ	\Re	ℜ	\wp	℘	\prime	′
\otimes	⊗	\oplus	⊕	\oslash	∅	\0	∅
\cap	∩	\cup	∪	\supseteq	⊇	\mid	|
\supset	⊃	\subseteq	⊆	\subset	⊂	\copyright	©
\int	∫	\in	∈	\o	o		
\lfloor	⌊	\lceil	⌈	\nabla	∇		
\wedge	∧	\times	x	\ldots	…		
\rceil	⌉	\surd	√	\prime	′		
\vee	∨	\varpi	ϖ	\0	∅		
\langle	〈	\rangle	〉	\mid	|		

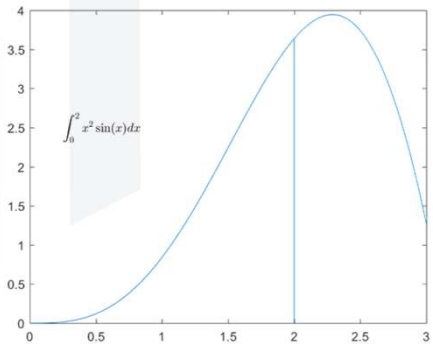
For more information:
http://www.math.jhu.edu/~shiffman/370/help/techdoc/ref/text_props.html

EMPossible

LaTeX in MATLAB

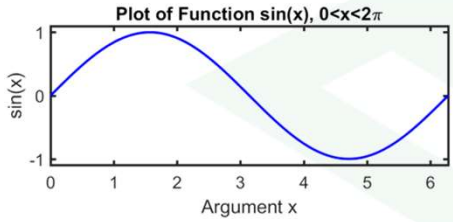
Instead of regular text, MATLAB can interpret LaTeX.

```
plot(x,y)
line([2,2],[0,2^2*sin(2)])
str = '$$ \int_{0}^{2} x^2\sin(x) dx $$';
text(0.25,2.5,str,'Interpreter','latex')
```



EMPossible

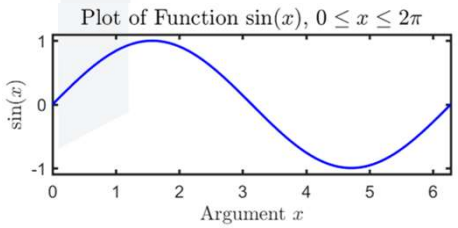
Labeling Plots with LaTex



Ordinary Axis Labels

- Improper formatting of math functions and variables.
- Less professional

```
xlabel('Argument x');
ylabel('sin(x)');
title('Plot of Function sin(x), 0<x<2\pi');
```



LaTex Axis Labels

- Proper formatting of math functions and variables.
- More professional.

```
xlabel('$ \textrm{Argument } x $',...
       'Interpreter','LaTex','FontSize',16);
ylabel('$ \textrm{sin}(x) $',...
       'Interpreter','LaTex','FontSize',16);
title('$ \textrm{Plot of Function sin}(x) \textrm{, } 0 \leq x \leq 2\pi $',...
      'Interpreter','LaTex','FontSize',18);
```

EMPossible

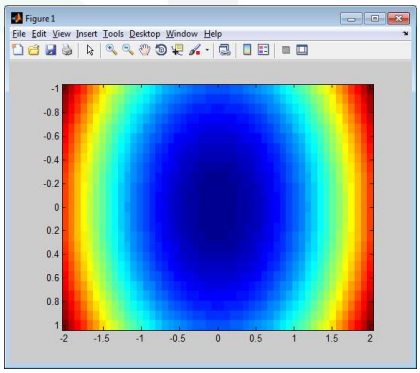
2D Graphics

imagesc() (1 of 3)

The `imagesc()` command displays a 2D array of data as an image to the screen. It automatically scales the coloring to match the scale of the data.

```
xa = linspace(-2,2,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
imagesc(xa,ya,D');
```

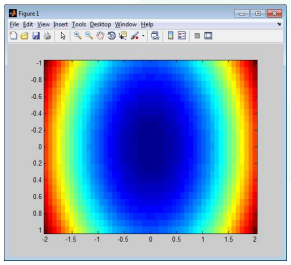
I use this function to display "digital looking" data from arrays.



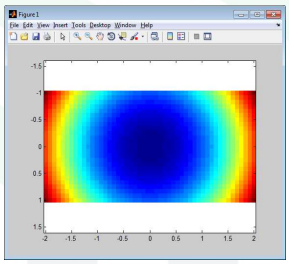
EMPossible

imagesc() (2 of 3)

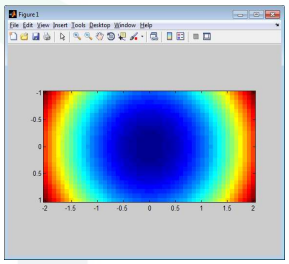
Scaling can be off. Use the `axis` command to correct this.



No `axis` command.



`axis equal;`



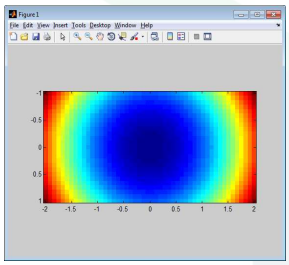
`axis equal tight;`

```
xa = linspace(-2,2,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
imagesc(xa,ya,D');
axis equal tight;
```

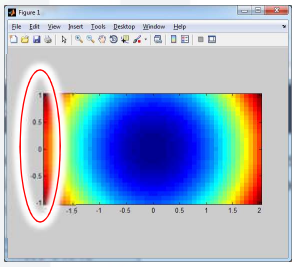
EMPossible

imagesc() (3 of 3)

Notice the orientation of the vertical axis using `imagesc()`. MATLAB assumes it is drawing a matrix so the numbers increase going downward.



```
imagesc(xa,ya,D');
```



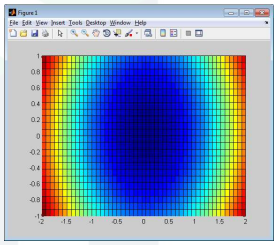
```
h = imagesc(xa,ya,D');
h2 = get(h,'Parent');
set(h2,'YDir','normal');
```

EMPossible

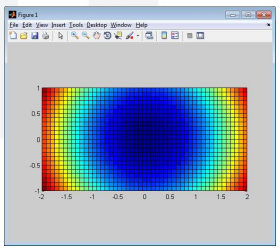
pcolor() (1 of 3)

`pcolor()` is a bit like `imagesc()`, but is better for displaying functions and smooth data because it has more options for this.

```
xa = linspace(-1,1,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
pcolor(xa,ya,D);
```



```
axis equal tight;
```

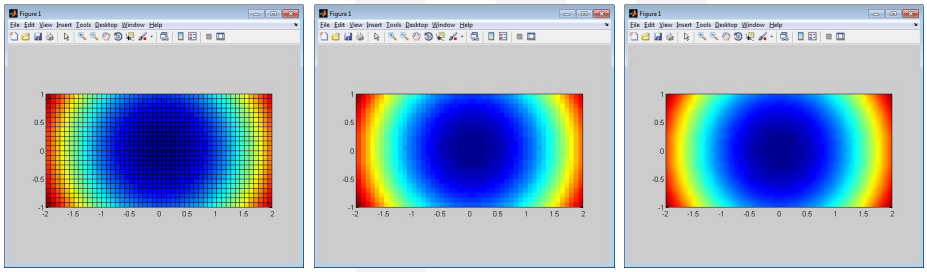


EMPossible

pcolor() (2 of 3)

```
xa = linspace(-2,2,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
pcolor(xa,ya,D');
shading interp;
axis equal tight;
```

Here are the main options for shading.

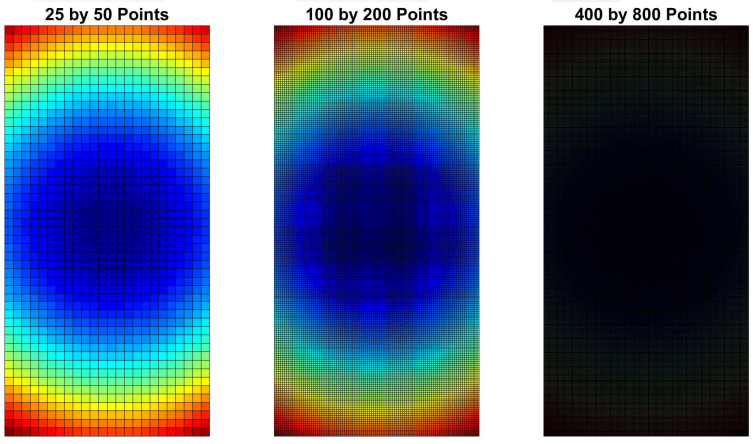


shading faceted; shading flat; shading interp;

EMPossible

pcolor() (3 of 3)

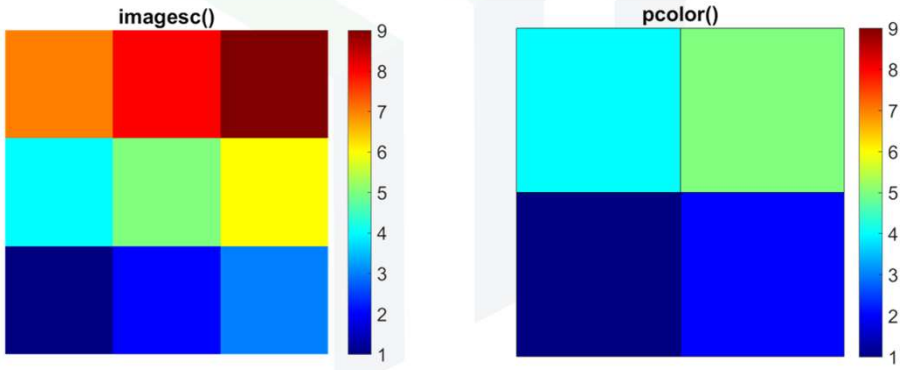
CAUTION!! When large arrays are visualized and `faceted` shading is selected (it is default), your image will appear completely black.



EMPossible

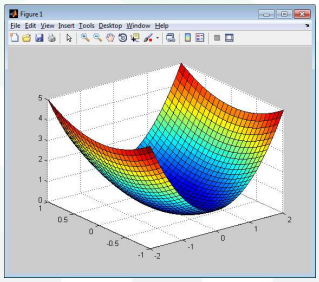
imagesc() vs. pcolor()

```
A = [ 1 2 3 ; ...
      4 5 6 ; ...
      7 8 9];
```

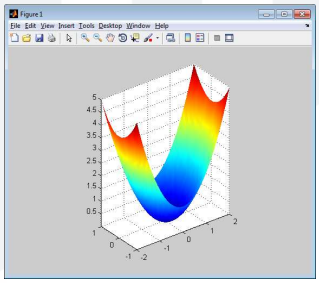


surf() (1 of 2)

surf() is like pcolor(), but shows a 3D representation.



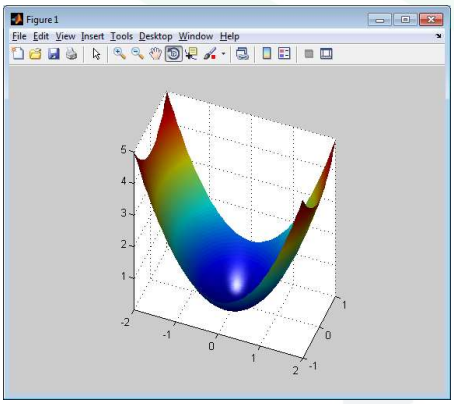
```
xa = linspace(-2,2,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
surf(xa,ya,D');
```



```
xa = linspace(-2,2,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
surf(xa,ya,D');

axis equal tight;
shading interp;
```

surf() (2 of 2)

The `surf()` command generates a 3D entity so it has all the properties and features of a 3D graph. I recommend orbiting to find the best view as well as playing with the lighting.



`view(az,el)`

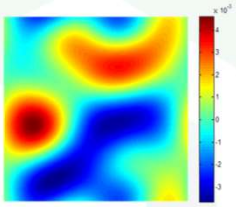
```
xa = linspace(-2,2,50);
ya = linspace(-1,1,25);
[Y,X] = meshgrid(ya,xa);

D = X.^2 + Y.^2;
surf(xa,ya,D');

axis equal tight;
shading interp;

camlight; lighting phong;
view(25,45);
```

EMPossible

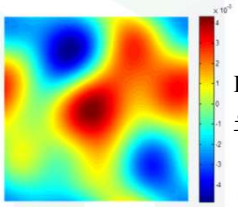
Plotting Complex Functions

Suppose we wish to plot a complex function $F(x,y)$. MATLAB won't let us plot a complex function so we are forced to plot only the real part, imaginary part, magnitude, phase, etc.

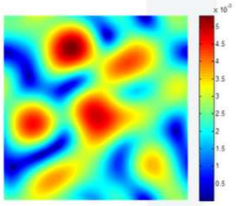
$\mathrm{Re}\left[F(x,y)\right]$ `real(F)`



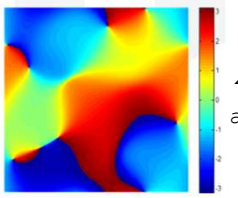
$\mathrm{Im}\left[F(x,y)\right]$ `imag(F)`



$\left|F(x,y)\right|$ `abs(F)`



$\angle F(x,y)$ `angle(F)`

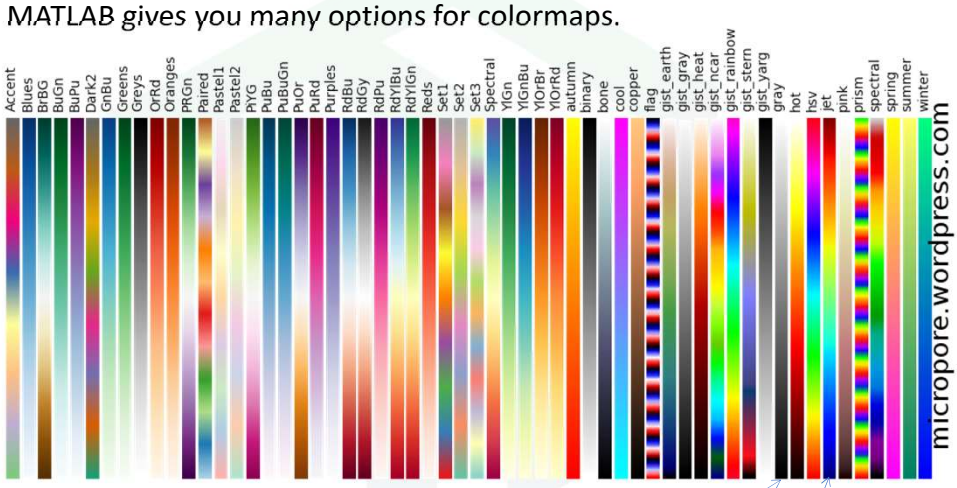


EMPossible

Caution: expect crazy results when your plotted function is a constant.

Colormaps

MATLAB gives you many options for colormaps.



Use 'gray' for black and white printouts.

```
colormap('gray')
```

'jet' is the default in 2014a and older versions of MATLAB.

EMPossible

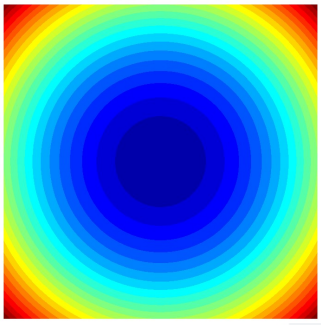
Obtaining Smoother Color Shading

Colormaps define a range of colors, but contain only discrete color levels.

Smoother colors are obtained by using more color levels.

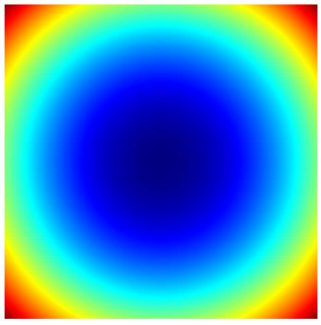
64 levels is the default.

24 Color Levels



```
colormap(jet(24));
```

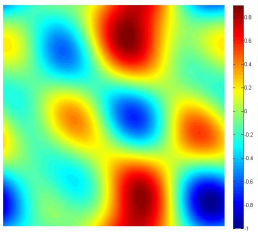
1024 Color Levels



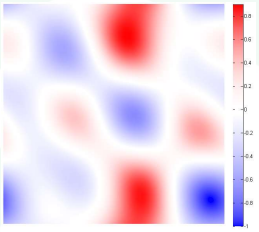
```
colormap(jet(1024));
```

EMPossible

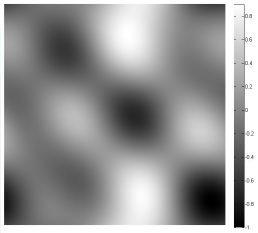
Simple Colormaps for Negative/Positive



```
colormap('jet');
```



```
% DEFINE CUSTOM COLORMAP
CMAP = zeros(256,3);
c1   = [0 0 1];   %blue
c2   = [1 1 1];   %white
c3   = [1 0 0];   %red
for nc = 1 : 128
    f = (nc - 1)/128;
    c = (1 - sqrt(f))*c1 + sqrt(f)*c2;
    CMAP(nc,:) = c;
    c = (1 - f^2)*c2 + f^2*c3;
    CMAP(128+nc,:) = c;
end

colormap(CMAP);
```

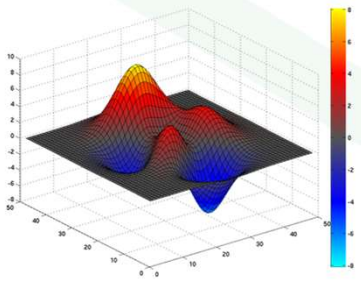


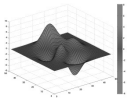
```
colormap('gray');
```


Other Colormaps in MATLAB Central

Bipolar Colormap

This is an excellent colormap when the sign of information is important.

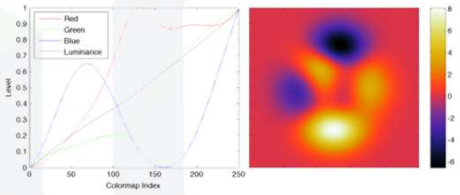


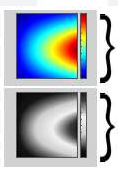


This colormap does not work when printed in grayscale.

CMR Colormap

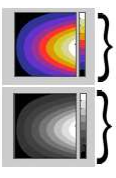
This is a color colormap, but also looks good when printed in grayscale.





Jet in full color. ☺

Jet in full grayscale. ☹

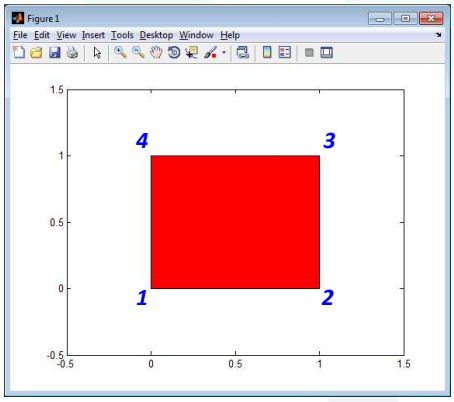


CMR in full color. ☺

CMR in full grayscale. ☺

fill(x,y,c) (1 of 3)

You can fill a polygon using the `fill(x,y,c)` command.



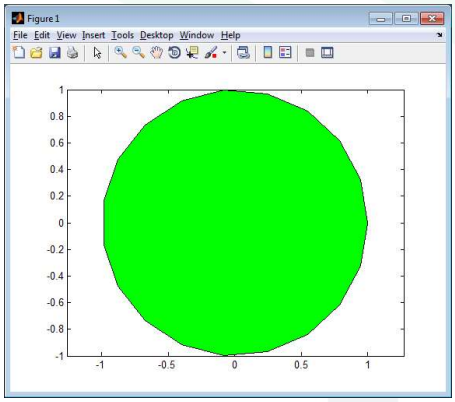

```
x = [ 0 1 1 0 0 ];
y = [ 0 0 1 1 0 ];
fill(x,y,'r');
axis([-0.5 1.5 -0.5 1.5]);
```

$x = [x_1\ x_2\ x_3\ x_4\ x_1];$
$y = [y_1\ y_2\ y_3\ y_4\ y_1];$

EMPossible

fill(x,y,c) (2 of 3)

You can make circles too!



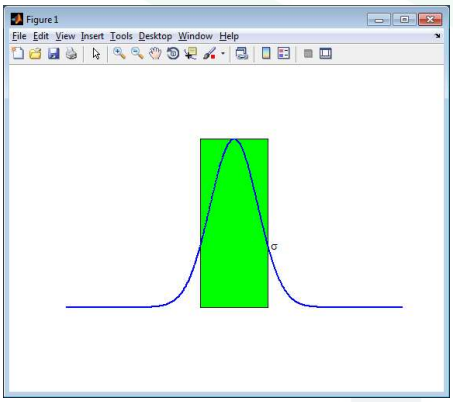
The more points you use, the smoother your circle will look.

```
phi = linspace(0,2*pi,20);
x = cos(phi);
y = sin(phi);
fill(x,y,'g');
axis([-1 +1 -1 +1]);
axis equal;
```

EMPossible

fill(x,y,c) (3 of 3)

These polygons can be superimposed onto each other and onto other graphical entitites using the `hold` command.

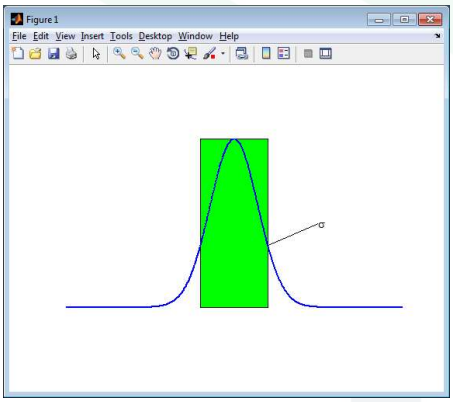



```
sigma = 0.2;
x = linspace(-1,1,1000);
y = exp(-(x/sigma).^2);

xx = (1*sigma) * [ -1 1 1 -1 -1 ];
yy = [ 0 0 1 1 0 ];
fill(xx,yy,'g');

hold on;
plot(x,y,'-b','LineWidth',2);
hold off;

text(1.1*sigma,1/exp(1),'\sigma');

axis equal tight off;
```

EMPossible

line(x,y)

The `line()` command behaves like `fill()`, but only draws a single line.

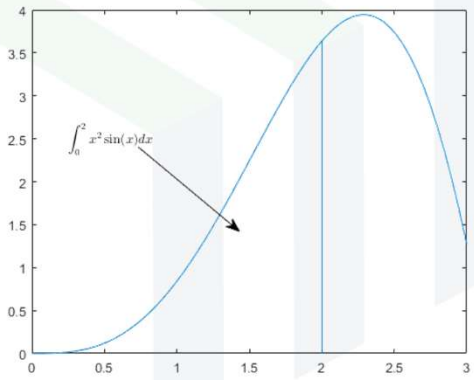



```
sigma = 0.2;
x = linspace(-1,1,1000);
y = exp(-(x/sigma).^2);

xx = (1*sigma) * [ -1 1 1 -1 -1 ];
yy = [ 0 0 1 1 0 ];
fill(xx,yy,'g');

hold on;
    plot(x,y,'-b','LineWidth',2);

    xx = [sigma 0.5];
    yy = [1/exp(1) 0.5];
    line(xx,yy,'Color','k');
hold off;

text(0.5,0.5,'\sigma');

axis equal tight off;
```

EMPossible

Annotation Arrows

```
annotation('arrow','X',[0.32,0.5],'Y',[0.6,0.4])
```



EMPossible

General Tips for Good Graphics

- Ensure lines are thick enough to be easily seen, but not too thick to be awkward.
- Ensure fonts are large enough to be easily read, but not too large to be awkward.
- All axes should be properly labeled with units.
- Figures should be made as small as possible so that everything is still easily observed and pleasing to the eye.
- Provide labels and/or legends to identify everything in the figure.
- It is sometimes good practice to not include much formatting for graphics that will be updated many times during the execution of a code.

EMPossible

Creating Movies with MATLAB

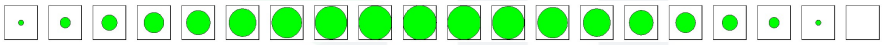
61

Basic Flow of Movie Code

Step 1: Open the Movie

```
movie_name = 'dumb_movie.mp4';
vidObj = VideoWriter(movie_name,'MPEG-4');
open(vidObj);
```

Step 2: Add frames to the movie



```
% Draw Frame
clf;
…
drawnow;

% Add Frame to AVI
F = getframe(fig);
writeVideo(vidObj,F);
```

This is repeated over however many frames you wish to add to the movie.

There is a MATLAB bug where you may have to replace `clf` with `close all;`

Step 3: Close the movie

```
close(vidObj);
```

EMPossible

Slide 62

getframe() command

You can capture the entire figure window to include all the subplots.

```
F = getframe(fig);
```

or

```
F = getframe(gcf);
```

Note: you can only capture frames from Monitor 1.

Or you can capture a specific subplot only.

```
subplot(???);
F = getframe(gca);
```

Aside #1: You can capture a frame and convert it to an image.

```
F = getframe(fig);
B = frame2im(F);
imwrite(B,'dumb_pic.jpg','JPEG');
```

Aside #2: You can load an image from file and add it as a frame.

```
B = imread('dumb_pic.jpg','JPEG');
F = im2frame(B);
writeVideo(vidObj,F);
```

EMPossible

Trick: Be Able to "Turn Off" Movie Making

Often times, you will need to play with your code to fix problems or tweak the graphics in your frames.

It is best to be creating a movie as you tweak your code and graphics.

Add a feature to your code to turn the movie making on or off.

```
MAKE_MOVIE = 0;
movie_name = 'mymovie.mp4';

% INITIALIZE MOVIE
if MAKE_MOVIE
    vidObj = VideoWriter(movie_name,'MPEG-4');
    open(vidObj);
end

%
% CREATE FRAMES
%
for nframe = 1 : NFRAMES

    % Draw Frame
    …

    % Add Frame to AVI
    if MAKE_MOVIE
        F = getframe(fig);
        writeVideo(vidObj,F);
    end
end

% CLOSE THE MOVIE
if MAKE_MOVIE
    close(vidObj);
end
```

EMPossible

Adjusting the Movie Parameters

It is possible to adjust properties of the video including quality, frame rate, video format, etc.

```
% INITIALIZE MOVIE
if MAKE_MOVIE
    vidObj =
VideoWriter(movie_name);

    vidObj.FrameRate = 20;
    vidObj.Quality   = 75;

    open(vidObj);
end
```

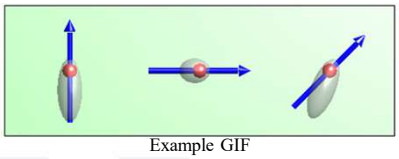
Parameters must be set after the video object is created and before it is opened.

Type `>> help VideoWriter` at the command prompt to see a full list of options for videos.

EMPossible

Making Animated GIFs

```
MAKE_GIF = 0;
gif_name = 'mygif.gif';
dt       = 0;

%
% CREATE FRAMES
%
for nframe = 1 : NFRAMES

    % Draw Frame
    …

    % Add Frame to GIF
    if MAKE_GIF
        pause(0.5);
        F = getframe(gca);
        F = frame2im(F);
        [ind,cmap] = rgb2ind(F,256,'nodither');
        if nframe == 1
            imwrite(ind,cmap,gif_name,'gif','DelayTime',dt,'Loopcount',inf);
        else

imwrite(ind,cmap,gif_name,'gif','DelayTime',dt,'WriteMode','append');
        end
    end
end
```



Example GIF

EMPossible

String Manipulation & Text Files

Parsing Strings

A line of text can be parsed into separate words using `sscanf()`.

```
S = 'Hello Class';
T = sscanf(S,'%s %s');
```

```
T =
HelloClass
```

```
S = 'Hello Class';
T = sscanf(S,'%*s %s');
```

skip string

```
T =
Class
```

You can also extract numbers from a string using `sscanf()`.

```
S = 'I am 25 years old and 6 feet tall.';
N = sscanf(S,'%*s %*s %f %*s %*s %*s %f %*s %*s');
```

```
N =
    25
     6
```

EMPossible

Checking for Specific Words in Strings (1 of 3)

You can perform an exact comparison between two strings using `strcmp()`.

```
s1 = 'cat';
s2 = 'dog';
c = strcmp(s1,s2);
```

```
c =
     0
```

```
s1 = 'cat';
s2 = 'Cat';
c = strcmp(s1,s2);
```

```
c =
     0
```

```
s1 = 'cat';
s2 = 'cat';
c = strcmp(s1,s2);
```

```
c =
     1
```


Checking for Specific Words in Strings (2 of 3)

You can do the same comparison, but case insensitive using `strcmpi()`.

```
s1 = 'cat';
s2 = 'dog';
c = strcmpi(s1,s2);
```

```
c =
     0
```

```
s1 = 'cat';
s2 = 'Cat';
c = strcmpi(s1,s2);
```

```
c =
     1
```

```
s1 = 'cat';
s2 = 'cat';
c = strcmpi(s1,s2);
```

```
c =
     1
```

Checking for Specific Words in Strings (3 of 3)

You can find the occurrence of one string inside another using `strfind()`.

```
s1 = 'University of Texas at El Paso';
s2 = 'Hawaii';
ind = strfind(s1,s2);
```

```
ind =
     []
```

```
s1 = 'University of Texas at El Paso';
s2 = 'Texas';
ind = strfind(s1,s2);
```

```
ind =
    15
```


Converting Between Strings and Numbers

You can convert a string to a number using `str2num()`.

```
S = '534';
N = str2num(S);
```

```
N =
534
```

```
S = '2.74E-10';
N = str2num(S);
```

```
N =
2.74E-10
```

Similarly, you can convert numbers to strings using `num2str()`.

```
N = 1234;
S = num2str(N);
```

```
S =
1234
```

```
N = 1.23456789;
S = num2str(N);
```

```
S =
1.2346
```

```
N = 1.23456789;
S = num2str(N,'%1.6f');
```

```
S =
1.234568
```

Opening and Closing Files

A file is opened in MATLAB as read-only with the following command.

```
% OPEN ASCII STL FILE
fid = fopen('pyramid.STL','r');       %'r' is read-only for safety
if fid==-1
    error('Error opening file.');
end
```

Use 'w' to write files.

A file is closed in MATLAB with the following command.

```
% CLOSE FILE
fclose(fid);
```

Open and closing the file is always the first and last thing you do.
WARNING! Always close open files!

EMPossible

Reading a Line from the Text File

A line of text is read from the text file using the following command.

```
% READ A LINE OF TEXT
L = fgetl(fid);
```

You can read and display an entire text file with the following code.

```
% READ AND DISPLAY AN ENTIRE TEXT FILE
while feof(fid)==0

    % Get Next Line from File
    L = fgetl(fid);

    % Display the Line of Text
    disp(L);

end
```

EMPossible

Writing a Line to the Text File

A line of text is written to the text file using `fprintf()`.

```
% WRITE A LINE OF TEXT
fprintf(fid,'solid pyramid\r\n');
```

This writes "`solid pyramid`" to the file followed by carriage line return.

Numbers can also be written to the file.

```
% Write Facet Normal to File
N = [ 1.234567 68.76543 1.592745 ];
L = '   facet normal %8.6f %8.6f %8.6f\r\n';
fprintf(fid,L,N);
```

This writes " `facet normal 1.234567e0 6.876543e1 1.592745e0`" to the file followed by carriage line return.

EMPossible

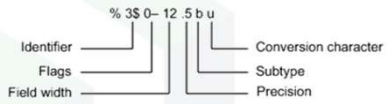
ANSI Formatting -- Summary

ESCAPE CHARACTERS

''	Single quotation mark
%%	Percent character
\\	Backslash
\a	Alarm
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\xN	Hexadecimal number, *N*
\N	Octal number, *N*

For most cases, \n is sufficient for a single line break.
However, if you are creating a file for use with Microsoft Notepad, specify a combination of \r\n to move to a new line.

CONVERSION CHARACTERS



Value Type	Conversion	Details
Integer, signed	%d or %i	Base 10 values
Integer, unsigned	%u	Base 10
Floating-point number	%f	Fixed-point notation
	%e	Exponential notation, such as 3.141593e+00
	%E	Same as %e, but uppercase, such as 3.141593E+00
	%g	The more compact of %e or %f, with no trailing zeros
	%G	The more compact of %E or %f, with no trailing zeros
Characters	%c	Single character
	%s	String of characters

EMPossible

ANSI Formatting – Conversion Characters

Value Type	Conversion	Details
Integer, signed	%d or %i	Base 10 values
	%ld or %li	64-bit base 10 values
	%hd or %hi	16-bit base 10 values
Integer, unsigned	%u	Base 10
	%o	Base 8 (octal)
	%x	Base 16 (hexadecimal), lowercase letters a–f
	%X	Same as %x, uppercase letters A–F
	%lu %lo %lx or %lX	64-bit values, base 10, 8, or 16
	%hu %ho %hx or %hX	16-bit values, base 10, 8, or 16
Floating-point number	%f	Fixed-point notation
	%e	Exponential notation, such as 3.141593e+00
	%E	Same as %e, but uppercase, such as 3.141593E+00
	%g	The more compact of %e or %f, with no trailing zeros
	%G	The more compact of %E or %f, with no trailing zeros
	%bx or %bX %bo %bu	Double-precision hexadecimal, octal, or decimal value Example: %bx prints pi as 400921fb54442d18
	%tx or %tX %to %tu	Single-precision hexadecimal, octal, or decimal value Example: %tx prints pi as 40490fdb
Characters	%c	Single character
	%s	String of characters

EMPossible

77

ANSI Formatting – Flags

Action	Flag	Example
Left-justify.	'-'	%-5.2f
Print sign character (+ or −).	'+'	%+5.2f
Insert a space before the value.	' '	% 5.2f
Pad with zeros.	'0'	%05.2f
Modify selected numeric conversions: • For %o, %x, or %X, print 0, 0x, or 0X prefix. • For %f, %e, or %E, print decimal point even when precision is 0. • For %g or %G, do not remove trailing zeros or decimal point.	'#'	%#5.0f

EMPossible

78

Helpful Tidbits

Initializing MATLAB

I like to initialize MATLAB this way…

```
% INITIALIZE MATLAB
close all;          %closes all figure windows
clc;                %erases command window
clear all;          %clears all variables from memory
```

Initializing Arrays

A 10×10 array can be initialized to all zeros.

```
A = zeros(10,10);
```

A 10×10 array can be initialized to all ones.

```
A = ones(10,10);
```

A 10×10 array can be initialized to all random numbers.

```
A = rand(10,10);
```

Numbers can also be put in manually. Commas separate numbers along a row while semicolons separate columns.

```
A = [ 1 2 3 4 5 6 7 8 9 ]

A = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
```

```
A = [ 1 2 3 4 5 6 7 8 9 ]

A = [ 1 2 3
      4 5 6
      7 8 9 ]
```

EMPossible

`break` Command

The `break` command is used to break out of a `for` or `while` loop, but execution continues after the loop.

```
a = 1;
while 1
    a = a + 1;
    if a > 5
        break;
    end
end
a
```

```
a =

     6

>>
```

Note: In old versions of MATLAB, `break` could be used to stop execution of a program. Today, you must use the `return` command for this.

EMPossible

log() Vs. log10()

Be careful, the `log()` command is the natural logarithm*!*

$\ln(2) = 0.6931$ → `log(2)` →

```
ans =
    0.6931

>>
```

The base-10 logarithm is `log10()`.

$\log_{10}(2) = 0.3010$ → `log10(2)` →

```
ans =
    0.3010

>>
```

EMPossible

find() Command

The `find()` command is used to find the array indices of specific values in an array. Examples for 1D arrays are:

```
A = [ 0.2 0.4 0.1 0.6 ]
ind = find(A==0.1)
```

→

```
A =
    0.2   0.4   0.1   0.6
ind =
     3

>>
```

```
A = [ 0.2 0.4 0.1 0.6 ]
ind = find(A>=0.4)
```

→

```
A =
    0.2   0.4   0.1   0.6
ind =
     2     4

>>
```

This command also works for multi-dimensional arrays*!* ☺

EMPossible

' vs. .'

The apostrophe ' operator performs a complex transpose (Hermitian) operation.

A standard transpose is performed by a dot-apostrophe operator .'

```
A = [ 0.1+0.1i , 0.2+0.2i ; …
      0.3-0.3i , 0.4-0.4i ]
A'
A.'
```

```
A =
   0.1000 + 0.1000i   0.2000 + 0.2000i
   0.3000 - 0.3000i   0.4000 - 0.4000i

A' =
   0.1000 - 0.1000i   0.3000 + 0.3000i
   0.2000 - 0.2000i   0.4000 + 0.4000i

A.' =
   0.1000 + 0.1000i   0.3000 - 0.3000i
   0.2000 + 0.2000i   0.4000 - 0.4000i

>>
```


Anonymous Functions

An anonymous function is a simple function defined within a single MATLAB statement. The syntax is

```
f = @(arglist)expression
```

Example 1 – Square Function

```
>> sqr = @(x) x.^2;
>> a = sqr(8)
a =
    64
```

Example 2 – Sum of Two Numbers

```
>> addthem = @(x,y) x+y;
>> addthem(8,10)
ans =
    18
```

Example 3 – Nested Anonymous Functions

```
>> addsqr = @(x,y) sqr(x)+sqr(y);
>> addsqr(8,10)
ans =
   164
```

Function Handle

A function handle stores an association to a function so that the function can be called by another name.

```
f = @thefunction
```

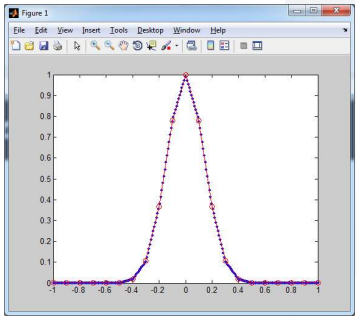
Example 1 – Cosine Function

```
>> f = @cos;
>> f(0.1)
ans =
    0.9950
```

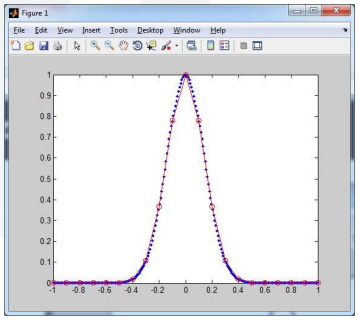
EMPossible

interp1() Command

```
% GRID
xa1 = linspace(-1,1,21);
xa2 = linspace(-1,1,250);

% FUNCTION
f1 = exp(-xa1.^2/0.2^2);

% INTERPOLATE
f2 = interp1(xa1,f1,xa2);

plot(xa2,f2,'.b'); hold on;
plot(xa1,f1,'o-r'); hold off;
```



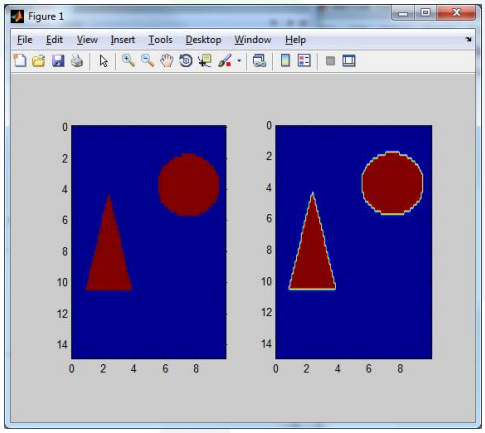
```
% GRID
xa1 = linspace(-1,1,21);
xa2 = linspace(-1,1,250);

% FUNCTION
f1 = exp(-xa1.^2/0.2^2);

% INTERPOLATE
f2 = interp1(xa1,f1,xa2,'cubic');

plot(xa2,f2,'.b'); hold on;
plot(xa1,f1,'o-r'); hold off;
```



EMPossible

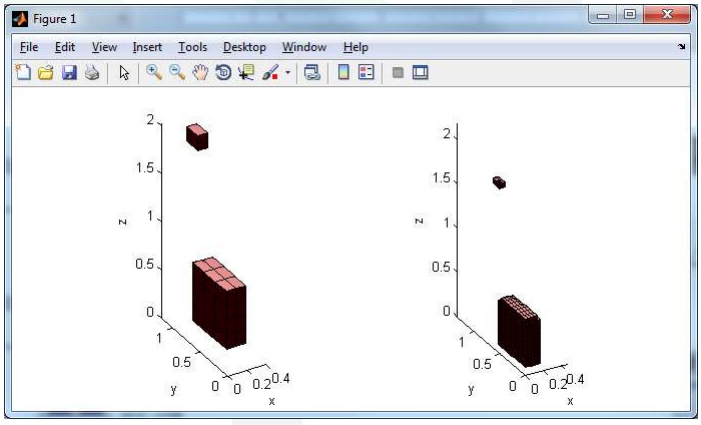
interp2() Command

```
% USE INTERP2
ER2 = interp2(ya,xa',ER,ya2,xa2');
```



interp3() Command

```
% USE INTERP3
ER2 = interp3(ya,xa',za,ER,ya2,xa2',za2,'linear');
```



Timing Code

Method 1: `tic & toc`

```
% START TIMER
tic

% CODE TO MEASURE
…

% STOP TIMER
toc
```

The time will be reported at the command prompt.

```
Elapsed time is 0.127685 seconds.
>>
```

You cannot nest `tic/toc` statements.

Method 2: `etime & clock`

```
% START TIMER
t1 = clock;

% CODE TO MEASURE
…

% STOP TIMER
t2 = clock;
t  = etime(t2,t1);
disp(['Elapsed time is ' num2str(t) …
      ' seconds.']);
```

You must manually report the time.

```
Elapsed time is 0.043 seconds.
>>
```

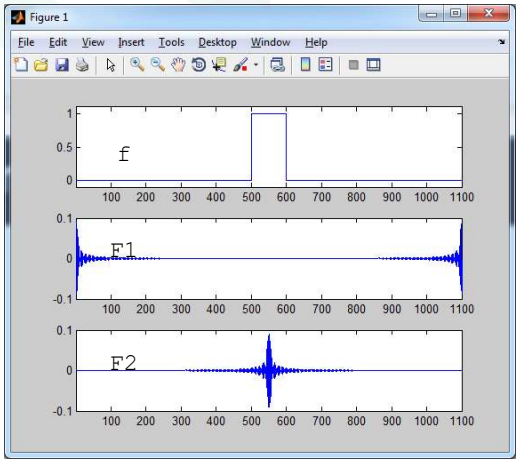
These commands can be nested.

EMPossible

FFT and FFTSHIFT

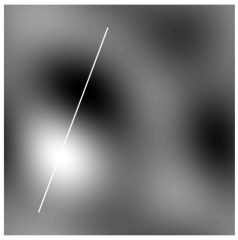
```
% CALCULATE DATA
f  = [ zeros(1,500) ones(1,100) zeros(1,500) ];
F1 = fft(f)/length(f);
F2 = fftshift(F1);
```

`fftshift()` centers your spectrum. →



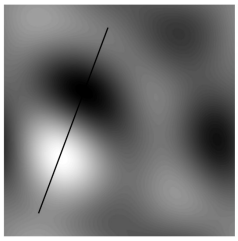
EMPossible

Drawing Lines Across Busy Backgrounds



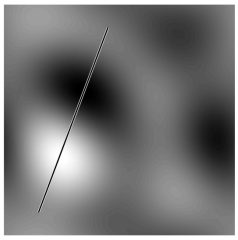
Light lines are not visible against light backgrounds.

```
line(x,y,'Color','w','LineWidth',3);
```



Dark lines are not visible against dark backgrounds.

```
line(x,y,'Color','k','LineWidth',3);
```



A solution…

Plot the same line twice to give it an outline.

```
line(x,y,'Color','w','LineWidth',6);
line(x,y,'Color','k','LineWidth',3);
```

EMPossible

Text Strings with Numbers

You can convert numbers to text strings using the `num2str()` function in MATLAB.

```
>> num2str(32.456)
ans =
32.456
```

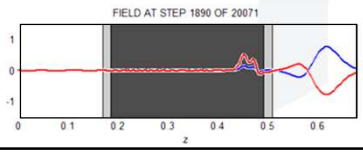
```
>> num2str(32.456,'%7.4f')
ans =
32.4560
```

It is possible to construct text strings with numbers within the string.

```
>> ['There are ' num2str(12) ' eggs in a dozen.']
ans =
There are 12 eggs in a dozen.
```

In FDTD, it is sometimes helpful to report the iteration information in the title of a figure.

```
title(['FIELD AT STEP ' num2str(T) ' OF ' num2str(STEPS)]);
```



EMPossible

Getting Date/Time From Internet

You can get the exact date and time from NIST's website using MATLAB.

```
% GET DATE & TIME FROM INTERNET
% Time code is GMT in microseconds since 1 Jan 1970
try
    raw_data  = webread('http://nist.time.gov/actualtime.cgi?lzbc=siqm9b');
catch
    error('No internet connection is available.');
end
ind       = strfind(raw_data,'"');
time_usec = str2num(raw_data(ind(1)+1:ind(2)-1));
time_sec  = time_usec/1000000;
time_days = time_sec/86400;
ML_time   = datenum(1970,1,1) + time_days;
```

Even more, you can follow this to check an expiration date.

```
% CHECK IF CODE HAS EXPIRED
% datenum(year, month, day)
expiration_datetime = datenum(2019,6,14);
if expiration_datetime < ML_time
    T = ['This MATLAB code expired on ' datestr(expiration_datetime) '.'];
    error(T);
end
```

EMPossible

parfor

The `parfor` loop is perhaps the easiest way to parallelize your code.

Before you can use the `parfor` loop, you must initialize your processors. This code must be placed at the start of your MATLAB program.

```
% INITIALIZE PARALLEL PROCESSING
if isempty(gcp)
    pool = parpool;
end
```

The `parfor` loop is used exactly like a regular `for` loop, but the code inside the for loop is sent out to your different processors.

```
% PERFORM PARALLEL PROCESSING
parfor m = 1 : M
    …
end
```

WARNING! The code inside your loop must not depend on the results from any other iterations of the loop. They must be completely independent.

EMPossible