Lecture 2
Advanced MATLAB: Graphics

Matthew J. Zahr

CME 292
Advanced MATLAB for Scientific Computing
Stanford University

25th September 2014

Announcements

- Office hours are set for 5p - 7p in Durand 028 (or by drop-in/appointment)
- Homework 1 out today, due next Thursday (10/2)
- (Optional) Project

pack

`pack` frees up needed space by reorganizing information so that it only uses the minimum memory required. All variables from your base and global workspaces are preserved. Any persistent variables that are defined at the time are set to their default value (the empty matrix, []).

- Useful if you have a large numeric array that you *know* you have enough memory to store, but can't find enough *contiguous* memory
- Not useful if your array is too large to fit in memory

Outline

Overview

- Graphics objects
  - Basic drawing elements used by MATLAB to display data
  - Each object *instance* has unique identifier, *handle*
    - Stored as a double
  - Objects organized in *hierarchy*

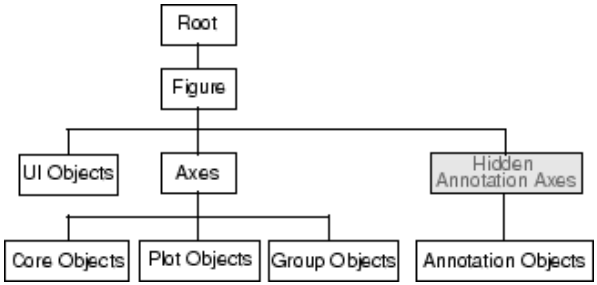


Figure : Organization of Graphics Objects (MathWorks `http://www.mathworks.com/help/matlab/creating_plots/organization-of-graphics-objects.html`)

Graphics Objects

Two basic types of graphics objects

- Core graphics object
  - `axes`, `image`, `light`, `line`, `patch`, `rectangle`, `surface`, `patch`
- Composite graphics object
  - Plot objects
    - `areaseries`, `barseries`, `contourgroup`, `errorbarseries`, `lineseries`, `quivergroup`, `scattergroup`, `staircase`, `stemseries`, `surfaceplot`
  - Annotation objects
    - arrow, doublearrow, ellipse, line, rectangle, textarrow, textbox
  - Group objects
    - `hggroup`, `hgtransform`
  - User Interface objects

Graphics Handle

- Similar to *pointers* in that they contain a *reference* to a particular graphics object
  - `h1 = figure(2); h2 = h1;`
  - Both `h1`, `h2` point to figure 2
- Best way to obtain graphics handle is *from the call that creates the graphics object*, i.e.
  - `figH = figure('pos',[141,258,869,523]);`
  - `axH = axes();`
  - `ax1H = subplot(2,2,3);`
  - `sinH = plot(sin(linspace(0,2*pi,100)))`
  - `[c,contH] = contour(peaks);`
- Alternatively, obtain graphics handle manually
  - Select figure/axes/object of interest with mouse
  - Use `gcf`, `gca`, `gco`
- Graphics handles stored as `double`

Handle stored as `double`

The value of the `double` really is the *only* identifier of the graphics object

```
>> format long
>> ax1 = gca % Copy/paste output to ax2
>> ax2 = 1.197609619140625e+03
>> ishandle(ax2)
ans =
     1
```

Specifying Figure or Axes to Use

Handles can be used to specify which figure or axes is used when new graphics objects generated

- Specify figure in which to create new axes object
  - `for i = 1:10, fHan(i)=figure(); end`
  - `ax = axes('Parent',fHan(4))`
- Specify axes in which to create new graphics object
  - Most, if not all, plotting commands accept an axes handle as the *first* argument
  - Graphics object generated in axes object corresponding to handle passed
  - If axes handle not specified, `gca` used
  - `[C,objHan] = contourf(ax,peaks)`
- By default, MATLAB uses `gcf` (handle of current figure) or `gca` (handle of current axes)

Exercise

- You are provided a fairly useless piece of code below (`which_plot_ex.m`)
- Your task is to alter the code below such that
  - `sin(k*x)` is plotted vs `x` for `k` even in a single figure
  - `sin(k*x)` is plotted vs `x` for `k` odd in a single figure (different figure from the one above)

```
figure;
axes(); hold on;

figure;
axes(); hold on;

x = linspace(0,2*pi,1000);
for k = 1:10
    plot(x,sin(k*x));
end
```

Working with Graphics Objects

Command	Description
`gca`	Return handle of current axes
`gcf`	Return handle of current figure
`gco`	Return handle of current object
`get`	Query values of object's properties
`ishandle`	True if value is valid object handle
`set`	Set values of an object's properties

Working with Graphics Objects

Command	Description
`allchild`	Return all children of objects
`ancestor`	Return ancestor of object
`copyobj`	Copy graphics object
`delete`	Delete an object
`findall`	Return all graphics objects
`findobj`	Return handles of objects with specified property

Query/Modify Graphics Object Properties

- `get` to query properties and values for any graphics handle
  - `get(han)`
    - Display all properties and values to screen
  - `get(han,'Property')`
    - Display `Property` value to screen
  - `V = get(han)`
    - Store all properties-value pairs in structure `V`
  - `V = get(han,'Property')`
    - Store `Property` value in `V`
- `set` to set properties for any graphics handle
  - `set(han,'Prop-1',Val-1,'Prop-2',Val-2...)`
    - Set `Prop-j`'s value to `Val-j`
  - `set(han,s)` `%s structure`
    - Set property-value pairs from `s`
  - `set(han,pn,pv)` `%pn, pv cell arrays`
    - Set value of property `pn{i}` to `pv{i}`

Properties Common to All Objects

Command	Description
`BeingDeleted`	`on` when object's `DeleteFcn` called
`BusyAction`	Control callback routine interruption
`ButtonDownFcn`	Callback routine that executes when button pressed
`Children`	Handles of all object's child objects
`Clipping`	Enables/disables clipping
`CreateFcn`	Callback routine that executes when object created
`DeleteFcn`	Callback routine that executes when object deleted

Properties Common to All Objects

Command	Description
`HandleVisibility`	Allows control over object handle's visibility (command line and callbacks)
`HitTest`	Determines if object selectable via mouse click
`Interruptible`	Determines whether callback can be interrupted by subsequently called callback
`Parent`	The object's parent
`Selected`	Indicates whether object is selected

Properties Common to All Objects

Command	Description
`SelectionHighlight`	Specifies whether object visually indicates selection state
`Tag`	User-specified object label
`Type`	The type of object
`UserData`	Any data user associates with object
`Visible`	Determines whether object is visible

Figure, Axes, and Plot Objects

- Figure window
  - `get(gcf)` to see all properties of Figure object and defaults
  - `Colormap`, `Position`, `PaperPositionMode`
- Axes Object
  - Axes objects contain the lines, surfaces, and other objects that represent the data visualized in a graph
  - `get(gca)` to see all properties of Figure object and defaults
  - `XLim`, `YLim`, `ZLim`, `CLim`, `XGrid`, `YGrid`, `ZGrid`, `XTick`, `XTickLabel`, `YTick`, `YTickLabel`, `ZTick`, `ZTickLabel`, `XScale`, `YScale`, `ZScale`
- Plot Objects
  - Plot objects are composite graphics objects composed of one or more core objects in a group
  - `XData`, `YData`, `ZData`, `Color`, `LineStyle`, `LineWidth`

The Figure Window

- `get(gcf)` to see all properties of Figure object and defaults
  - `Colormap`
    - Defines colors used for plots
    - Must be $m \times 3$ array of m RGB values
  - `PaperOrientation`, `PaperPosition`, `PaperPositionMode`, `PaperSize`
    - Relevant for printing
  - `Position`
    - Position and size figure: `[x, y, w, h]`
    - `x`, `y` - (x, y) coordinates of lower left corner of figure
    - `w`, `h` - width, height of figure
  - `NextPlot`
    - Behavior when multiple axes object added to figure

The Axes Object

Axes objects contain the lines, surfaces, and other objects that represent the data visualized in a graph

- `get(gca)` to see all properties of Axes object and defaults
  - `XLim`, `YLim`, `ZLim`, `CLim`
    - Set plot limits in each dimension (including color)
    - More information on `CLim` here
  - `XGrid`, `XMinorGrid`, `YGrid`, `YMinorGrid`, `ZGrid`, `ZMinorGrid`
    - Toggle major and minor grid lines in each dimension
  - `XTick`, `XTickLabel`, `YTick`, `YTickLabel`, `ZTick`, `ZTickLabel`
    - Control tick locations and labels in each dimension
  - `XScale`, `YScale`, `ZScale`
    - Toggle between linear and log scale in each dimension
  - Camera, Fonts, Line style options

Colormap

Colormaps enable control over how MATLAB maps data values to colors in surfaces, patches, images, and plotting functions

- `C = colormap(jet(128));`
  - Sets colormap of current figure to `jet` with `128` colors
  - `autumn`, `bone`, `colorcube`, `cool`, `copper`, `flag`, `gray`, `hot`, `hsv`, `jet`, `lines`, `pink`, `prism`, `spring`, `summer`, `white`, `winter`
- Alternatively

```
>> fig = figure();
>> ax = axes('Parent',fig);
>> load spine; image(X);
>> colormap(ax,bone);
```

  - This is a bit strange as `Colormap` is a property of the figure (not axes), but the axes handle is passed to `colormap`
    - Access to figure handle (`get(ax,'Parent')`)

Plot Objects

- Plot objects are composite graphics objects composed of one or more core objects in a group
- Most common plot objects: `lineseries`, `contourgroup`
- `Lineseries`
  - `XData`, `YData`, `ZData`
    - Control x, y, z data used to plot line
  - `Color`, `LineStyle`, `LineWidth`
    - Control appearance of line
  - `Marker`, `MarkerSize`, `MarkerEdgeColor`, `MarkerFaceColor`
    - Control appearance of markers on line
- `Contourgroup`
  - `XData`, `YData`, `ZData`
    - Control x, y, z data used to plot line
  - `LineStyle`, `LineWidth`, `LineColor`
  - `Fill`, `LevelStep`

Legend

- Probably familiar with basic legend syntax
  - `legend('First plotted','Second ... plotted','Location','Northwest')`
- What if legend based on order of objects plotted is not sufficient?
  - Use handles for fine grained control
  - `legend(h,'h(1)label','h(2)label')`
- Legend handle
  - Get handle by `leg = legend( )`
  - Use handle to control size/location (more control that `'Location'`)
  - Font size/style, interpreter, line style, etc

Callback Routines

- Function associated with graphics handle that gets called in response to a specific action applied to the associated graphics object
  - Object creation, deletion
  - Mouse motion, mouse press, mouse release, scroll wheel
  - Key press, key release
  - More here
- All callback routines automatically passed two inputs
  - Handle of component whose callback is being executed
  - Event data
- Callback routines specified in many possible forms
  - String
    - Expression evaluated in *base* workspace
  - Function handle
  - Cell arrays to pass additional arguments to callback routine

Demo & In-Class Assignment

`graphics_obj_han_ex.m`

Outline

Line plots

Command	Description
`plot`	2D line plot
`plotyy`	2D line plot, y-axes both sides
`plot3`	3D line plot
`loglog`	2D line plot: x- and y-axis log scale
`semilogx`	2D line plot, x-axis log, y-axis linear
`semilogy`	2D line plot, x-axis linear, y-axis log
`errorbar`	Error bars along 2D line plot
`fplot`	Plot function between specified limits
`ezplot`	Function plotter
`ezplot3`	2D parametric curve plotter

Examples: `plotyy`, `errorbar`

Figure : `plotyy` Plot

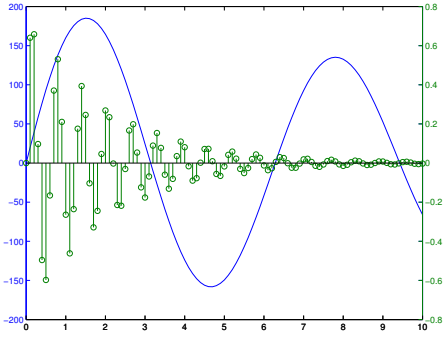
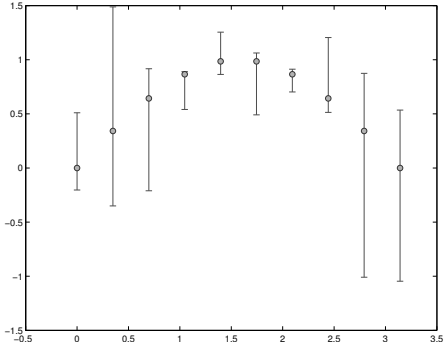


Figure : `errorbar` Plot



- Code: `advanced_plotting_ex.m`

Line plots: Examples

- Multiple y-axes
  - `[ax,h1,h2]=plotyy(X1,Y1,X2,Y2)`
    - Plot `X1`, `Y1` using left axis and `X2`, `Y2` using right axis
  - `[ax,h1,h2]=plotyy(X1,Y1,X2,Y2,'function')`
    - Plot `X1`, `Y1` using left axis and `X2`, `Y2` using right axis with plotting function defined by string `'function'`
  - `[ax,h1,h2]=plotyy(X1,Y1,X2,Y2,'f1','f2')`
    - Plot `X1`, `Y1` using left axis with plotting function `'f1'` and `X2`, `Y2` using right axis with plotting function `'f2'`
- Error plots
  - `h = errorbar(X,Y,E)`
    - Create 2D line plot from data `X`, `Y` with symmetric error bars defined by `E`
  - `h = errorbar(X,Y,L,U)`
    - Create 2D line plot from data `X`, `Y` with upper error bar defined by `U` and lower error bar defined by `L`

Pie Charts, Bar Plots, and Histograms

Command	Description
`bar`, `barh`	Vertical, horizontal bar graph
`bar3`, `bar3h`	Vertical, horizontal 3D bar graph
`hist`	Histogram
`histc`	Histogram bin count (no plot)
`rose`	Angle histogram
`pareto`	Pareto chart
`area`	Filled area 2D plot
`pie`, `pie3`	2D, 3D pie chart

Examples: `hist`, `bar`, `barh`, `pie3`

Figure : `hist/bar/barh` Plot

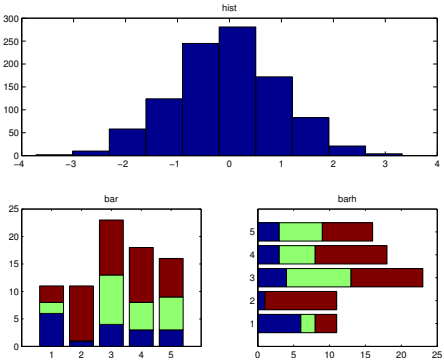
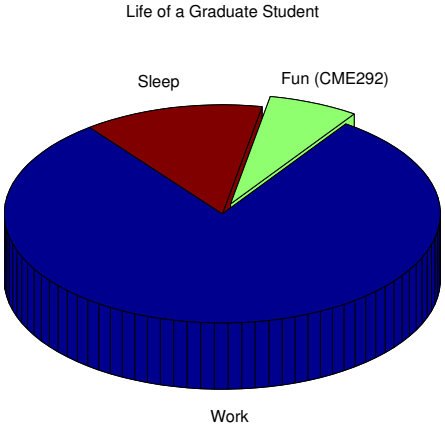


Figure : `pie3` Plot



- Code: `advanced_plotting_ex.m`

Discrete Data Plots

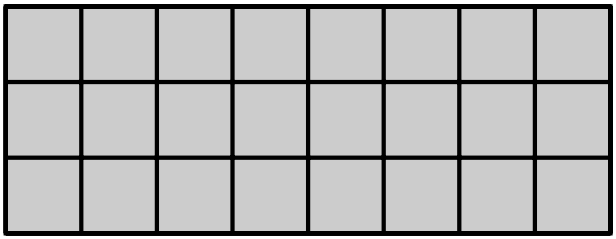
Command	Description
`stem`, `stem3`	Plot 2D, 3D discrete sequence data
`stair`	Stairstep graph
`scatter`, `scatter3`	2D, 3D scatter plot

Polar Plots

Command	Description
`polar`	Polar coordinates plot
`rose`	Angle histogram plot
`compass`	Plot arrows emanating from origin
`ezpolar`	Polar coordinate plotter

Generating Grid Data

- MATLAB graphics commands work primarily in terms of N-D grids



- Use `meshgrid` to define grid compatible with 2D, 3D MATLAB plotting commands from discretization in each dimension
  - `[X,Y] = meshgrid(x,y)`
  - `[X,Y,Z] = meshgrid(x,y,z)`

meshgrid

- Generate 2D grid: `[X,Y] = meshgrid(x,y)`
  - Relationships
    - `X(i,:)` = `x` for all i
    - `Y(:,j)` = `y` for all j
    - `X(:,i)` = `x(i)` for all i
    - `Y(j,:)` = `y(j)` for all j
- Generate 3D grid: `[X,Y,Z] = meshgrid(x,y,z)`
  - Relationships
    - `X(i,:,k)` = `x` for all i, k
    - `Y(:,j,k)` = `y` for all j, k
    - `Z(i,j,:)` = `z` for all i, j
    - `X(:,i,:)` = `x(i)` for all i,
    - `Y(j,:,:)` = `y(j)` for all j
    - `Z(:,:,k)` = `z(k)` for all k

Implication of `meshgrid` ordering

Consider the implication of `meshgrid` in the context of the function $\mathbf{F}(x, y) = \sin(x)\cos(y)$

- `s = linspace(0,2*pi,100)`
- `[X,Y] = meshgrid(s,s)`
- `F = sin(X).*cos(Y)`
- `F(i,j)== sin(X(i,j))*cos(Y(i,j))`
  `== sin(s(?))*cos(s(?))`
  `== sin(s(j))*cos(s(i))`

`meshgrid` and Plotting Functions

- In MATLAB Help documentation, grid or domain data inputs/outputs usually refer to output of `meshgrid` or `meshgrid` or `ndgrid`

surfnorm

R2014a

Compute and display 3-D surface normals

expand all in page

Syntax

```
surfnorm(Z)
surfnorm(X,Y,Z)
surfnorm(axes_handle, __ )
surfnorm( __ ,Name,Value)
[Nx,Ny,Nz] = surfnorm( __ )
```

Description

`surfnorm(Z)` plots a surface of the matrix `Z` with `surf` and displays its surface normals as radiating vectors.

`surfnorm(X,Y,Z)` plots a surface and its surface normals from the vectors or matrices `X`, `Y`, and matrix `Z`. `X`, `Y`, and `Z` must be the same size.

`surfnorm(axes_handle, __ )` plots into `axes_handle` instead of `gca` and it can include any of the input arguments in previous syntaxes.

`surfnorm( __ ,Name,Value)` can be used to set the value of the specified `Surface Properties`.

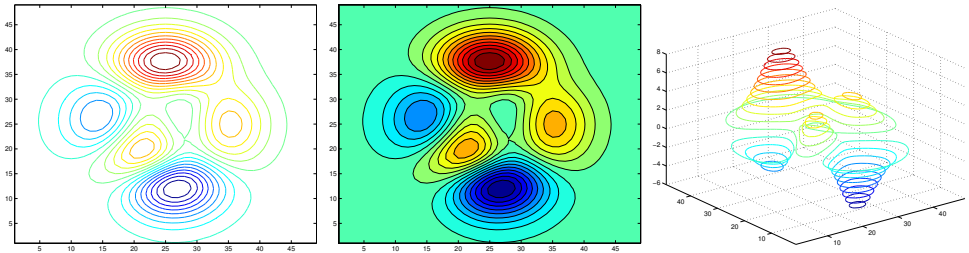
`[Nx,Ny,Nz] = surfnorm( __ )` returns the components of the 3-D surface normals for the surface without plotting the surface or surface normals.

Contour Plots

- Plot scalar-valued function of two variables as lines of constant value.
  - Visualize $f(x, y) \in \mathbb{R}$ by displaying lines where $f(x, y) = c$ for various values of c

Command	Description
`contour`	Contour plot
`contourf`	Filled contour plot
`contourc`	Contour plot computation (no plot)
`contour3`	3D contour plot
`contourslice`	Draw contours in volume slice planes
`ezcontour`	Contour plotter
`ezcontourf`	Filled contour plotter

Contour Plots



(a) `contour` (b) `contourf` (c) `contour3`

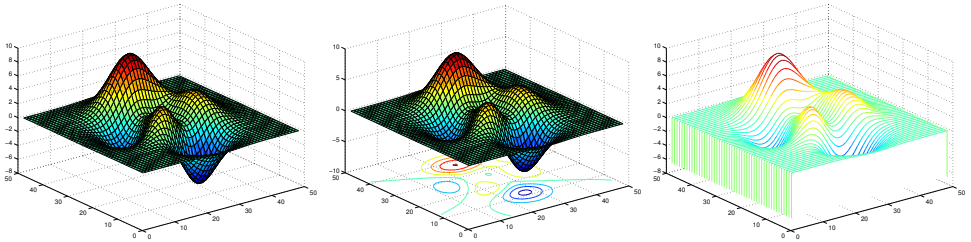
Code for plots generated in the remainder of the section:
`advanced_plotting_ex.m` or `lec_figs.m`

Surface and Mesh Plots

- Plot scalar-valued function of two variables $f(x, y) \in \mathbb{R}$

Command	Description
`surf`	3D shaded surface plot
`surfc`	Contour plot under `surf` plot
`surfl`	Surface plot with colormap lighting
`surfnorm`	Compute/plot 3D surface normals
`mesh`	Mesh plot
`meshc`	Contour plot under `mesh` plot
`waterfall`	Waterfall plot
`ribbon`	Ribbon plot
`ezsurf`, `ezsurfc`	Colored surface plotters
`ezmesh`,`ezmeshc`	Mesh plotters

Suface/Mesh Plots



(a) `surf` (b) `surfc` (c) `waterfall`

(d) `mesh` (e) `meshc`

Contour/Surface/Mesh Plots

- `[C,h] = contour_func(Z)`
  - Contour plot of matrix `Z`
- `[C,h] = contour_func(Z,n)`
  - Contour plot of matrix `Z` with `n` contour levels
- `[C,h] = contour_func(Z,v)`
  - Contour plot of matrix `Z` with contour lines corresponding to the values in `v`
- `[C,h] = contour_func(X,Y,Z)`
  - Contour plot of matrix `Z` over domain `X`, `Y`
- `[C,h] = contour_func(X,Y,Z,n)`
  - Contour plot of matrix `Z` over domain `X`, `Y` with `n` levels
- `[C,h] = contour_func(X,Y,Z,v)`
  - Contour plot of matrix `Z` over domain `X`, `Y` with contour lines corresponding to the values in `v`
- Similar for surface/mesh plots

Vector Fields

- Visualize vector-valued function of two or three variables $\mathbf{F}(x, y) \in \mathbb{R}^2$ or $\mathbf{F}(x, y, z) \in \mathbb{R}^3$

Command	Description
`feather`	Plot velocity vectors along horizontal
`quiver, quiver3`	Plot 2D, 3D velocity vectors from specified points
`compass`	Plot arrows emanating from origin
`streamslice`	Plot streamlines in slice planes
`streamline`	Plot streamlines of 2D, 3D vector data

Vector Fields

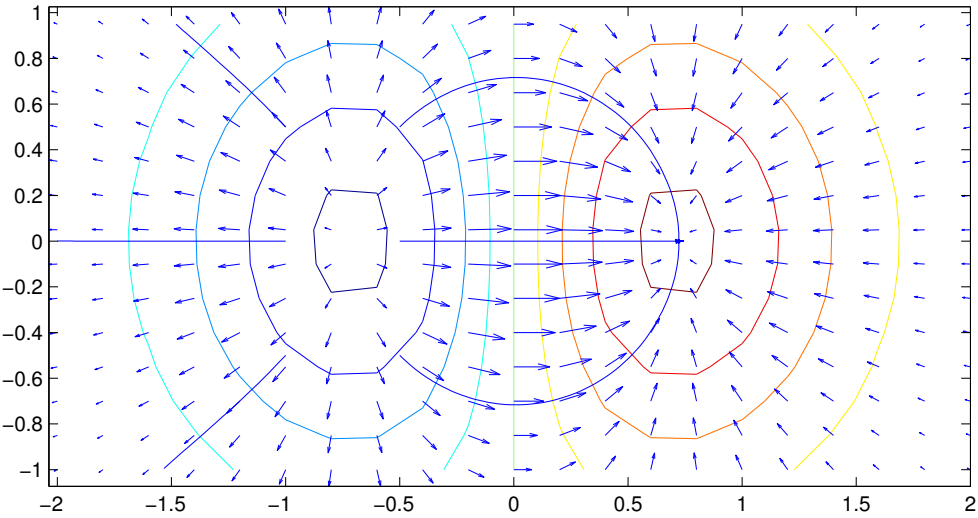


Figure : `contour`, `quiver`, `streamline`

Vector fields: `quiver`, `feather`, `compass`

- Quiver plots
  - `h = quiver(X,Y,U,V)`
    - Displays velocity vectors as arrows with components `(u,v)` at the point `(x,y)`
    - `X,Y` generated with meshgrid
    - Additional call syntaxes to control display
  - `h = quiver3(X,Y,Z,U,V,W)`
    - Displays velocity vectors as arrows with components `(u,v,w)` at the point `(x,y,z)`
    - `X,Y,Z` generated with meshgrid
    - Additional call syntaxes to control display
  - `Quivergroup Properties`
- `feather`, `compass` similar, but simpler (don't require `X`, `Y`)

Streamline-type plots

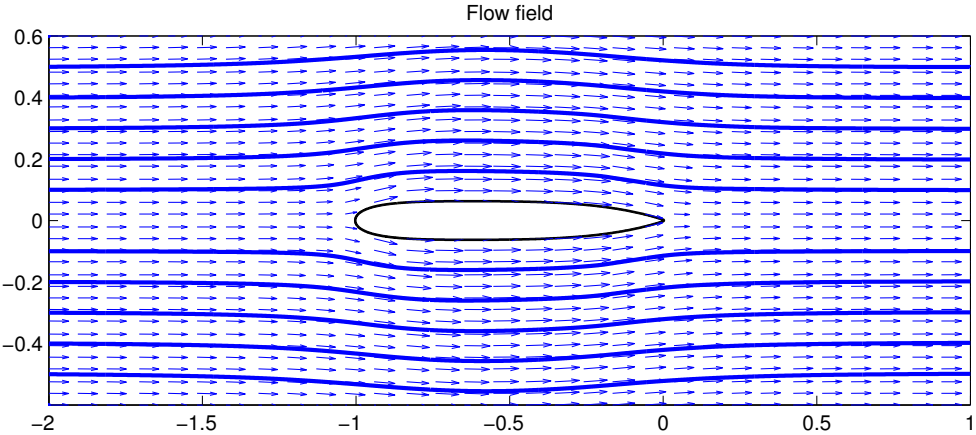


Figure : `quiver`, `streamline`, `fill` plots

Streamline-type plots

- `streamline`, `stream2`, `stream3`
- Relevant for vector-valued functions of 2 or 3 variables ($\mathbf{F}(x, y)$ or $\mathbf{F}(x, y, z)$)
- Requires points to initialize streamlines
- Plot the trajectory of a particle through a vector field that was placed at a given position
  - `han=streamline(X,Y,Z,F1,F2,F3,StX,StY,StZ)`
    - `X, Y, Z` - grid generated with `meshgrid`
    - `F1,F2,F3` - vector components of $\mathbf{F}$ over grid
    - `StX, StY, StZ` - vectors (of the same size) specifying the starting location of the particles to trace

Assignment

- Define `s = linspace(0,2*pi,100)`
- Plot $f(x, y) = \sin(xy)$ for $x, y \in [0, 2\pi]$ using any contour function
  - Make sure there are contour lines at $[-1.0, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1.0]$
  - Use any colormap except `jet` (the default)
    - `autumn`, `bone`, `colorcube`, `cool`, `copper`, `flag`, `gray`, `hot`, `hsv`, `jet`, `lines`, `pink`, `prism`, `spring`, `summer`, `white`, `winter`
  - Use a colorbar
- Numerically compute $\nabla f(x, y)$ as `[Fx,Fy] = gradient(F)`
  - Make a `quiver` plot of $\nabla f(x, y)$
  - Plot streamline of $\nabla f(x, y)$ vector field, beginning at the point $(2, 2)$

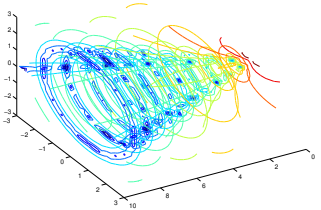
Volume Visualization - Scalar Data

- Visualize scalar-valued function of two or three variables $f(x, y, z) \in \mathbb{R}$

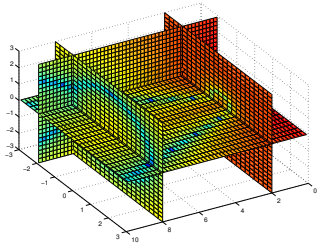
Command	Description
`contourslice`	Draw contours in volume slice planes
`flow`	Simple function of three variables
`isocaps`	Compute isosurface end-cap geometry
`isocolors`	Compute isosurface and patch colors
`isonormals`	Compute normals of isosurface vertices
`isosurface`	Extract isosurface data from volume data
`slice`	Volumetric slice plot

Volume Visualization - Scalar Data

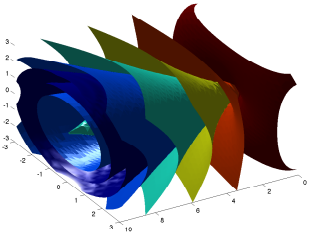
- Visualize scalar data defined over a volume.



(a) `contourslice`



(b) `slice`



(c) `isosurface`

Slice-type plots

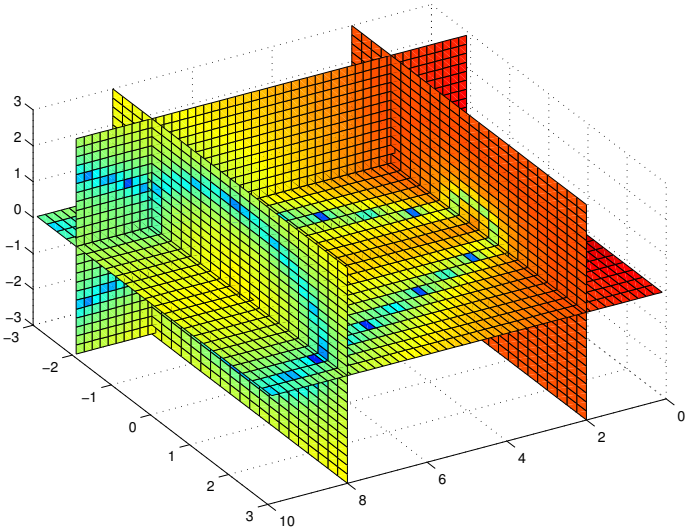


Figure : `slice`

Slice-type plots

- `slice`, `contourslice`, `streamslice`
- Relevant for scalar- or vector-valued volume functions ($f(x, y, z)$ or $\mathbf{F}(x, y, z)$)
- Plot information in planar slices of the volumetric domain
  - `han = slice(X,Y,Z,F,Sx,Sy,Sz)`
    - `X`, `Y`, `Z` - grid generated by meshgrid
    - `F` = $\mathbf{F}$(`X`,`Y`,`Z`)
    - `Sx`, `Sy`, `Sz` vectors specifying location of slice planes in the $y - z$, $x - z$, and $x - y$ planes
  - `han = slice(X,Y,Z,F,XI,YI,ZI)`
    - `XI`, `YI`, `ZI` define *surface* (i.e. that could be plotted with surf) on which to plot `F`

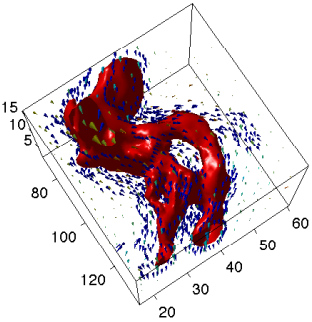
Volume Visualization - Vector Data

- Visualize vector-valued function of three variables $\mathbf{F}(x, y, z) \in \mathbb{R}^3$

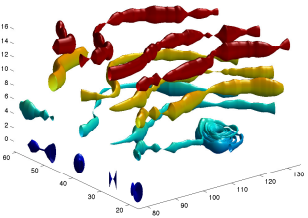
Command	Description
`coneplot`	Plot velocity vectors as cone
`interpstreamspeed`	Interpolate stream-line vertices from flow speed
`stream2`, `stream3`	Compute 2D, 3D streamline data
`streamline`	Plot streamlines of 2D, 3D vector data
`streamparticles`	Plot stream particles
`streamribbon`	3D stream ribbon plot
`streamslice`	Plot streamlines in slice planes
`streamtube`	Create 3D stream tube plot

Volume Visualization - Vector Data

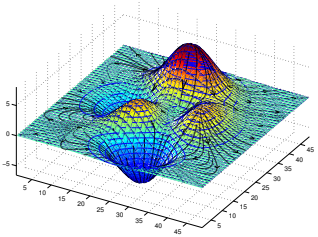
- Visualize vector data defined over a volume.



(a) `coneplot`



(b) `streamtube`



(c) `streamslice`

Polygons

Command	Description
`fill, fill3`	Fill 2D, 3D polygon
`patch`	Create filled polygons
`surf2patch`	Convert surface data to patch data

- Patch graphics object
  - Core graphics object
  - `Patch Properties`

Polygons

- `h = fill(X,Y,C)`
- `h = fill3(X,Y,Z,C)`
- `h = patch(X,Y,Z,C)`
  - For $m \times n$ matrices `X`, `Y`, `Z` draws n polygons with vertices defined by each column
  - Color of each patch determined by `C`
    - If `C` is a string (`'r','w','y','b','k',...`), all polygons filled with specified color
    - If `C` is a $1 \times n$ vector, each polygon face is flat colored by `C(j)`
    - If `C` is a $1 \times n \times 3$ matrix, each polygon face colored by RGB value
    - If `C` is a $m \times n \times 3$ matrix, each vertex colored by RGB value and face color determined by interpolation

Outline

MATLAB File Exchange

The MATLAB File Exchange is a very useful forum for find solutions to many MATLAB-related problems

- 3D Visualization
- Data Analysis
- Data Import/Export
- Desktop Tools and Develepment Environment
- External Interfaces
- GUI Development
- Graphics
- Mathematics
- Object-Oriented Programming
- Programming and Data Types

MATLAB File Exchange

The MATLAB File Exchange is a very useful forum for sharing solutions to many MATLAB-related problems

- Clean integration of MATLAB figures in LaTeX documents
  - `matlabfrag`, `figuremaker`, `export_fig`, `mcode`, `matrix2latex`, `matlab2tikz`
- Plot formatting and manipulation
  - `xticklabel_rotate`, `tight_subplot`
- Interfacing to iPhone, iPad, Android, Kinect devices
- Interfacing to Google Earth and Maps
- Much more

Outline

Motivation

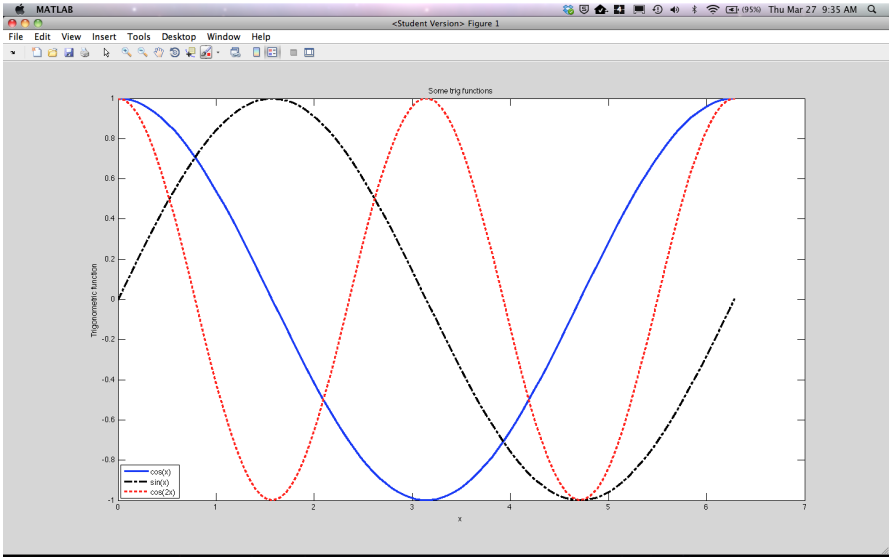
- Generating publication quality plots in MATLAB is not a trivial task
  - Plot annotation to match font size/style of document
  - Esthetic - dependent on type of publication
  - Legends can be difficult to work with
  - MATLAB figures not WYSIWYG by default
- Three fundamental approaches to generate plots for publications using MATLAB
  - Generate *plots* in MATLAB and import into document
    - Graphics handles to deal with esthetics
    - MATLAB File Exchange to integrate figures with LaTeX
  - Generate *data* in MATLAB and *plot* in document
    - TikZ/PGF popular choice for LaTeX
  - Hybrid (`matlab2tikz`)
- `high_quality_ex.m`

WYSIWYG

MATLAB is not *What You See Is What You Get* (WYSIWYG) by default, when it comes to plotting

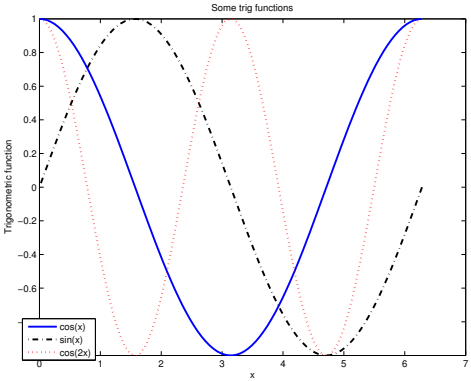
- Spend time making plot look exactly as you want
- Doesn't look the same when saved to file
  - Legend particularly annoying
  - Issues amplified when figure resized
- Very frustrating
- Force WYSIWYG
  - `set(gcf, 'PaperPositionMode', 'auto');`

WYSIWYG Example

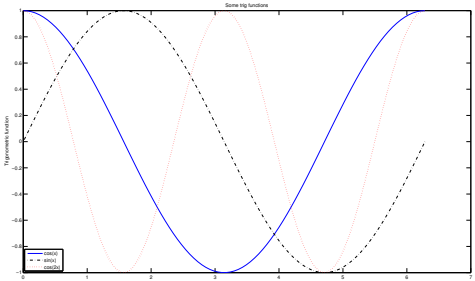


WYSIWYG Example

(a) Without WYSIWYG



(b) With WYSIWYG



High-Quality Graphics

This information is based on websites here and here.

- Generate plot with all lines/labels/annotations/legends/etc
- Set properties (graphics handles or interactively)
  - Figure width/height
  - Axes line width, object line width, marker size
  - Font sizes
- Save to figure to file
  - WYSIWYG
    - `set(gcf, 'PaperPositionMode', 'auto');`
  - Print to file for inclusion in document
    - `print(gcf,'-depsc2',filename)`
    - `matlabfrag(filename)`
    - `matlab2tikz(filename)`
- Fixing EPS file
  - Esthetics of dashed and dotted lines
  - `fixPSlinestyle`

Important Properties - Figure

- Figure properties
  - `InvertHardCopy`
    - Change hardcopy to black objects on white background
  - `PaperPositionMode`
    - Forces the figure's size and location on the printed page to directly reflect the figure's size on the screen
  - `PaperOrientation`, `PaperPosition`, `PaperUnits`, `Position`, `Units`
    - Allows manual mapping from figure to paper

Important Properties - Axes

- `Color`
  - Background color of plot
- `XLabel`, `YLabel`, `ZLabel`
- `XScale`, `YScale`, `ZScale`
- `XLim`, `YLim`, `ZLim`, `CLim`
- `XTick`, `XTickLabel`, `YTick`, `YTickLabel`, `ZTick`, `ZTickLabel`
  - Locations and labels of tick marks
- `XDir`, `YDir`, `ZDir`
  - Direction of axis ticks (ascending/descending)
- `XAxisLocation`, `YAxisLocation`
  - Place axis at left/right or top/bottom of axes
- `NextPlot`
  - Behavior when multiple objects added to axes

Important Properties - Other

- Line object
  - `LineWidth`
  - `Marker`
  - `MarkerSize`
  - `MarkerEdgeColor`
  - `MarkerFaceColor`
- Legend object
  - `Position`, `Interpreter`
  - `LineStyle`
    - Type of line used to box legend
- Text object
  - `Position`, `Interpreter`

Show Mcode

Two options to modify appearance of figure

- Interactively via MATLAB Figure GUI
  - Simplest and most popular
  - Not repeatable/automated
- Command line control via graphics handles
  - Less intuitive than interactive approach
  - Highly automated
  - Annoying trial/error when it comes to positioning/sizing

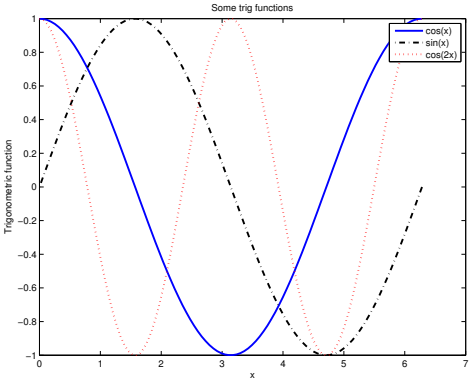
A hybrid approach that combines the above options is available

- Use GUI to interactively modify appearance of figure
- `Show Mcode` option to print underlying graphics handle operations to file
- Copy/paste into script for repeatability
- Demo: `show_mcode_ex.m`

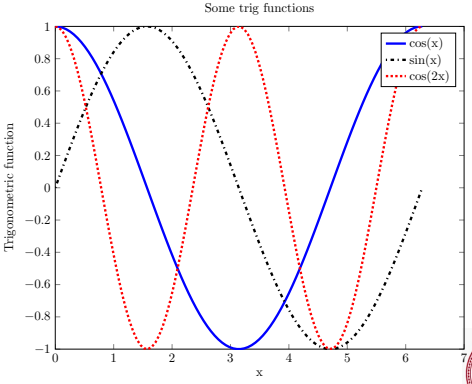
matlab2tikz

- `matlab2tikz(FileName,...)`
  - Save figure in native LaTeX (TikZ/Pgfplots).
  - Import file in LaTeX (`\input` or `\includegraphics`)

(a) without `matlab2tikz`



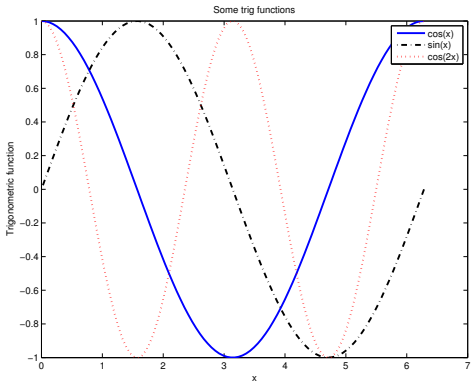
(b) with `matlab2tikz`



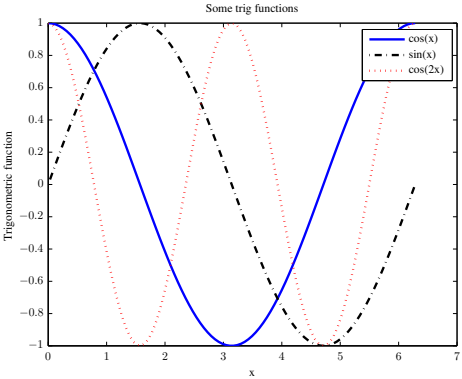
matlabfrag

- `matlabfrag(FileName,OPTIONS)`
  - Exports a matlab figure to an .eps file and a .tex file for use with psfrag in LaTeX.
  - Doesn't seem to work well with beamer

(a) without `matlabfrag`

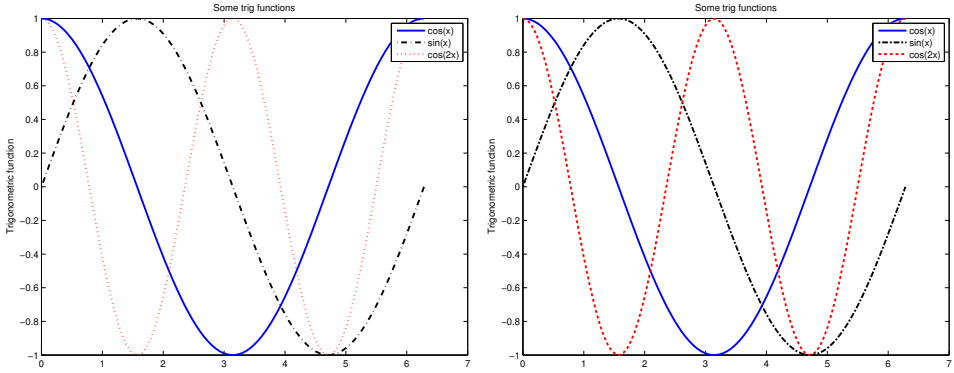


(b) with `matlabfrag`



fixPSlinestyle

(a) without `fixPSlinestyle` (b) with `fixPSlinestyle`



`fixPSlinestyle` syntax

- `fixPSlinestyle(fname)`
- `fixPSlinestyle(old_fname,new_fname)`

Outline

Animation

Two main types of animation

- Interactive animation
  - Generate and display animation during execution of code
- Animation movies
  - Save animation in movie format

Interactive Animation

- Generated by calling plot commands inside a loop with new data generated at each iteration
  - Before entering loop
    - Create figure and axes
    - Modify using handles to achieve desired appearance
    - Use command `set(gca,'nextplot','replacechildren')` to ensure only *children* of axes object will be replaced upon next plot command (will not modify axes properties)
  - During loop
    - Plotting command to generate data on plot
    - Modify object using handle to achieve desired appearance
    - Use command `drawnow` to draw object, otherwise will not be drawn until execution is complete (MATLAB optimization as plotting is expensive)
- Alternatively, modify `XData`, `YData`, `ZData` properties of initial plot object

Interactive Animation

- Additionally, save sequence of plotting command as frames (`getframe`) and play back from MATLAB window (`movie`)
- `animate_ex.m`

Approach 1

```
fig=figure();
ax=axes();
obj=plot(..);
set(ax,'nextplot',...
    'replacechildren');
for j = 1:N
   obj=plot(..);
   drawnow;
end
```

Approach 2

```
fig=figure();
ax=axes();
obj=plot(..);
set(ax,'xlim',..,'ylim',..);
for j = 1:N
    set(obj,'xdata',..);
    set(obj,'ydata',..);
    drawnow;
end
```

Animation Movies

Saving animations as movie files can be accomplished using `VideoWriter` class (`video_writer_ex.m`)

- `VideoWriter` enables creation of video files from MATLAB figures, still images, or MATLAB movies

```
writerObj = VideoWriter('my_movie.avi');  %Video obj
set(writerObj, 'FrameRate',10);  % Set the FPS
open(writerObj); % Open the video object
% Prepare the movie
figure; set(gca,'NextPlot','replaceChildren')
th = linspace(0,2*pi,100);
for j = th
    plot(sin(th),cos(th),'k-');  hold on;
    plot(sin(j),cos(j),'ro');
    writeVideo( writerObj, getframe );
end
close(writerObj); % Close the video object
```

VideoWriter

- List of `VideoWriter` properties
  - Here on MathWorks website
  - `FrameRate` - rate of playback (cannot change after `open`)
  - `Quality` - integer between 0, 100
- `VideoWriter` methods
  - `open` - Open file for writing video data
  - `writeVideo` - Write video data to file
  - `close` - Close file after writing video data