Plotting review

MATLAB’s `plot` function can be used for simple "join the dots" x-y plots.

```matlab
x = [1.5 2.2 3.1 4.6 5.7 6.3 9.4];
y = [2.3 3.9 4.3 7.2 4.5 3.8 1.1];
plot(x, y, 'r*--')
```
Plotting review

Options for the `plot` command

<table>
<thead>
<tr>
<th>Color</th>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>○ Circle</td>
</tr>
<tr>
<td>g</td>
<td>∗ Asterisk</td>
</tr>
<tr>
<td>b</td>
<td>· Point</td>
</tr>
<tr>
<td>c</td>
<td>⊕ Plus</td>
</tr>
<tr>
<td>m</td>
<td>× Cross</td>
</tr>
<tr>
<td>y</td>
<td>s Square</td>
</tr>
<tr>
<td>k</td>
<td>d Diamond</td>
</tr>
<tr>
<td>w</td>
<td>^ Upward triangle</td>
</tr>
<tr>
<td></td>
<td>v Downward triangle</td>
</tr>
<tr>
<td></td>
<td>&gt; Right triangle</td>
</tr>
<tr>
<td></td>
<td>&lt; Left triangle</td>
</tr>
<tr>
<td></td>
<td>p Five-point star</td>
</tr>
<tr>
<td></td>
<td>h Six-point star</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>-.</td>
</tr>
</tbody>
</table>
Plotting review

You can add a title and label the axes using the commands
\texttt{title, xlabel, ylabel}.

\begin{verbatim}
x = [1.5 2.2 3.1 4.6 5.7 6.3 9.4] ;
y = [2.3 3.9 4.3 7.2 4.5 3.8 1.1] ;
plot(x,y, 'r*--')
xlabel('x')
ylabel('y')
title('Title of the plot')
\end{verbatim}
# 2D plotting functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>plot</code></td>
<td>Simple x-y plot</td>
</tr>
<tr>
<td><code>loglog</code></td>
<td>Plot with logarithmically scaled axes</td>
</tr>
<tr>
<td><code>semilogx</code></td>
<td>Plot with logarithmically scale x-axis</td>
</tr>
<tr>
<td><code>semilogy</code></td>
<td>Plot with logarithmically scale y-axis</td>
</tr>
<tr>
<td><code>plotyy</code></td>
<td>x-y plot with the y-axes on the left and the right</td>
</tr>
<tr>
<td><code>polar</code></td>
<td>Plot in the polar coordinates</td>
</tr>
<tr>
<td><code>fplot</code></td>
<td>Automatic function plot</td>
</tr>
<tr>
<td><code>ezplot</code></td>
<td>Easy-to-use version of fplot</td>
</tr>
<tr>
<td><code>ezploar</code></td>
<td>Easy-to-use version of polar</td>
</tr>
<tr>
<td><code>fill</code></td>
<td>Polygon fill</td>
</tr>
<tr>
<td><code>area</code></td>
<td>Filled area graph</td>
</tr>
<tr>
<td><code>bar</code></td>
<td>Bar graph</td>
</tr>
<tr>
<td><code>barh</code></td>
<td>Horizontal bar graph</td>
</tr>
<tr>
<td><code>hist</code></td>
<td>Histogram</td>
</tr>
<tr>
<td><code>pie</code></td>
<td>Pie chart</td>
</tr>
<tr>
<td><code>errorbar</code></td>
<td>Error bar plot</td>
</tr>
<tr>
<td><code>quiver</code></td>
<td>Velocity vector plot</td>
</tr>
<tr>
<td><code>scatter</code></td>
<td>Scatter plot</td>
</tr>
</tbody>
</table>
Log-scale plots are useful when the data span a very wide range.

```matlab
x = 0:20;
y = factorial(x);
plot(x,y,'--x')  % badly scaled
semilogy(x,y,'--x')  % easier to read
```
2D plotting functions

Double-axis plots:

```matlab
x = linspace(0, 10, 101);
y1 = (1+x).^(-2);
y2 = 100*exp(-x/25);
plot(x, [y1' y2']) % badly scaled
plotyy(x, y1, x, y2) % easier to read
```
3D plotting functions

Commonly used 3D plotting functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot3</td>
<td>Simple x-y-z plot</td>
</tr>
<tr>
<td>contour</td>
<td>Contour plot</td>
</tr>
<tr>
<td>contourf</td>
<td>Filled contour plot</td>
</tr>
<tr>
<td>contour3</td>
<td>3D contour plot</td>
</tr>
<tr>
<td>mesh</td>
<td>Wireframe surface</td>
</tr>
<tr>
<td>meshc</td>
<td>Wireframe surface plus contours</td>
</tr>
<tr>
<td>meshz</td>
<td>Wireframe surface with curtain</td>
</tr>
<tr>
<td>surf</td>
<td>Solid surface</td>
</tr>
<tr>
<td>surfc</td>
<td>Solid surface plus contours</td>
</tr>
<tr>
<td>waterfall</td>
<td>Unidirectional wireframe</td>
</tr>
<tr>
<td>bar3</td>
<td>3D bar graph</td>
</tr>
<tr>
<td>bar3h</td>
<td>3D horizontal bar graph</td>
</tr>
<tr>
<td>pie3</td>
<td>3D pie chart</td>
</tr>
<tr>
<td>fill3</td>
<td>Polygon fill</td>
</tr>
<tr>
<td>scatter3</td>
<td>3D scatter plot</td>
</tr>
</tbody>
</table>
3D plotting functions

- 3D plots are necessary for visualizing functions of two variables: \( z = f(x, y) \)
- Need to pass in \( x, y, z \) instead of \( x, y \)
- Line plot:

```matlab
t = linspace(0, 8*pi, 401);
x = cos(t);
y = sin(t);
z = exp(-t/4);
plot3(x, y, z)
```
3D plotting functions

- More common: plot over a whole range of values $[x_0, x_1] \times [y_0, y_1]$
- Need something like linspace, but for 2 dimensions:

  ```
  x = linspace(-3, 3, 101);
  y = linspace(-1, 1, 51);
  [X, Y] = meshgrid(x, y);
  ```

- Creates matrices $X, Y$ such that $Z = f(X, Y)$ has $i,j$th entry $f(x_i, y_j)$
Exercise

- Draw the contours and surface for the function 
  \( \sin(3y - x^2 + 1) + \cos(2y^2 - 2x) \) over the range 
  \(-2 \leq x \leq 2 \) and \(-1 \leq y \leq 1 \)

\[ x = \ldots \]
\[ y = \ldots \]
\[ [X,Y] = \text{meshgrid}(x,y); \]
\[ Z = \ldots \]
\[ \text{contour}(\ldots) \]
\[ \text{surf}(\ldots) \]
A function handle is a callable association to a MATLAB function. It contains an association to that function that enables you to invoke the function regardless of where you call it from.

With function handles, you can:

- Pass a function to another function
- Capture data values for later use by a function
- Call functions outside of their normal scope
- Save the handle in a MAT-file to be used in a later MATLAB session
Creating function handles

You can construct a handle for a specific function by preceding the function name with an @ sign. The syntax is:

\[ h = \texttt{@functionname} \]

Once you create a handle for a function, you can invoke the function by means of the handle instead of using the function name.
Calling a function using its handle

The syntax for calling a function using a function handle is the same as that used when calling the function directly.

\[
h = @\text{plot}; \quad % \ h \text{ is a function handle for plot}
\]
\[
x = \text{linspace}(-\pi, \pi, 100);
\]
\[
h(x, \sin(x)) \ ; \ % \text{ Equivalent to plot}(x, \sin(x))
\]

Because the handle contains the absolute path to its function, you can invoke the function from any location that MATLAB is able to reach.
Passing a function as an argument

Other cases when we’d need to pass a function as an argument to another function:

- Plotting a function
- Finding the roots of a function
- Finding the maximum/minimum of a function
- Integrating a function
Plotting using `ezplot`:

- Can pass the function as a string:
  
  ```matlab
ezplot('x^2-1', [-1 1])
  ```

- For more complicated functions, pass in a function handle:

  ```matlab
  function y = myfun(x)
  y = x.^2 - 1;
  end
  ezplot(@myfun, [-1 1])
  ```
Calling a function using its handle

The syntax for calling a function using a function handle is the same as that used when calling the function directly:

```matlab
h = @sin;
x = linspace(0, 2*pi, 101);
plot(x, h(x));
```

A function handle contains the absolute path to its function, so you can call it from any directory. Could come in handy if your code is split across many directories.
Anonymous functions

Anonymous functions are a simple way to define one-line functions without creating a separate .m file.

```matlab
f = @(x) (abs(x+1));
ezplot(f, -2, 2);
```

Or, even more simply:

```matlab
ezplot(@(x) (abs(x+1)), -2, 2);
```
More easy-plotting

Make parametric plots by defining a function of two variables $x, y$:

```matlab
ezplot( @sin, [0 2*pi] )
ezplot( @(x,y) x.^4+y.^4-1, [-1 1] ), axis equal
x= @(t) exp( t ).*cos(8*t);
y= @(t) exp( t ).*sin(8*t);
ezplot(x,y,[0 3])
```

Line plots in 3D with `ezplot3`:

```matlab
ezplot3(@cos, @sin, @(t) exp( t /8), [0 40], 'animate'
```

Surface plots with `ezmesh`, `ezsurf`, `ezcontour`:

```matlab
ezsurf(@(x,y) exp(-x.^2-2*y.^2))
```
Finding the roots of function

\[ x = \text{fzero}(\text{fun}, x_0) \] tries to find a zero of \( \text{fun} \) near \( x_0 \), if \( x_0 \) is a scalar. \( \text{fun} \) is a function handle. The value \( x \) returned by \( \text{fzero} \) is near a point where \( \text{fun} \) changes sign, or \( \text{NaN} \) if the search fails.

Let’s find the zero of \( f(x) = x^3 - 4x^2 + 3x + 2 \) near 1.

\[
\begin{align*}
\text{myfun} &= @(x) (x^3 - 4\times x^2 + 3\times x + 2) ; \\
\text{fzero} & (\text{myfun}, 1) ; \\
\text{ans} &= 2.0000
\end{align*}
\]
Finding the minimum of a function

\texttt{fminbnd}(f,a,b) minimizes the (univariate) function $f$ over the interval $[a, b]$:

\begin{verbatim}
f = @(x) -sin(x+pi/4);
ezplot(f,0,1);
xstar = fminbnd(f,0,1)
xstar = 0.7854
\end{verbatim}

- Can minimize functions of multiple variables with \texttt{fminunc}, \texttt{fmincon}, \texttt{fminsearch}
- Maximize function by minimizing $-f(x)$
Numerical integration

\[ \text{quad}(f, a, b) \text{ computes } \int_{a}^{b} f(x) \, dx \text{ via numerical approximation:} \]

\[
f = @(x) -\sin(x+\pi/4);
\]
\[
\text{quad}(f, 0, 1)
\]
\[
\text{ans} =
\]
\[
-0.9201
\]

- Double and triple integrals can be computed with \text{dblquad} and \text{triplequad}, beyond that things get complicated.

- Side note: numerical differentiation is “easier” than integration, so the function \text{diff} is simple: \text{diff}(y) just returns \[ [y(2)-y(1) \ldots y(n)-y(n-1)] \].