Visualizing data in two and three dimensions/Graphing functions of 2 variables/Curves and Surfaces in 3-dimensions

We discuss the following MATLAB commands:
- `plot3(x,y,z)`
- `pcolor(A)` or `pcolor(X,Y,Z)`
- `contour(Z)` or `contour(X,Y,Z)`
- `surf(Z)` or `surf(X,Y,Z)`
- `surfc(Z)` or `surfc(X,Y,Z)`
- 
  ```matlab```
  ```markdown```
  ```
  [X,Y] = meshgrid(x,y)
  ```
  ```matlab```
  ```markdown```
- shading interp
- shading faceted
- shading flat
- colorbar
- colormap

Plotting curves in space:
Given one-dimensional arrays x,y,z, the command
- `plot3(x,y,z)`
plots the curve that connects, in order, the points
- \((x(1),y(1),z(1)),(x(2),y(2),z(2)),\ldots,(x(n),y(n),z(n))\).

This is the command used for plotting parametric curves, where the arrays x,y,z are each computed as functions of an array t.

Plotting the values of a matrix using `pcolor(A)`
Given a matrix A, the command `pcolor(A)` uses color to represent the values in the entries in A. The colors are assigned according to the colormap being used, with the lowest value in A corresponding to the lowest valued color and the highest value in A given the highest valued color, and the values in between determined by linear interpolation - this is what MATLAB means when it says that the values in A are scaled to the colormap. The elements in A are best thought of as corresponding to a rectangular set of gridpoints in the xy plane. The \((1,1)\) entry of A corresponds to the gridpoint \((x,y) = (1,1)\) and the \((m,n)\) entry of A corresponds to the gridpoint \((x,y) = (n,m)\); thus the \(i^{th}\) row of A corresponds to the \(i^{th}\) row of gridpoints (going up) and the \(j^{th}\) column of A corresponds to the \(j^{th}\) column of gridpoints. If you use pcolor without any modifications, then a general unit square in the plane is given the color associated with its lower left gridpoint and the values in the last row and last column of A are not used. As an example consider:
A=[1 2 3 4;5 6 7 8;9 10 11 12];
pcolor(A);colorbar

See (in the colorbar) how the largest value in A (the value 12) is associated with the highest numbered color (dark red) and the smallest value in A is associated with the lowest numbered color. The last row and column of A are not used in coloring the rectangles. This scheme of coloring (or shading, in MATLAB terminology) is called \textbf{faceted} shading - the colors are constant in a given rectangle and the gridlines between rectangles are shown in black. In so-called interpolated shading, each point is one of the rectangles above is given a color that is a weighted average of the colors associated with its four gridpoint neighbors, and the gridlines are not displayed. In that case, all of the elements of A are used in determining the colors in the plot. To employ this method, just write the command

\begin{verbatim}
shading interp;
\end{verbatim}

This gives a "smoothed" representation of the elements of A. This is actually more useful when the elements of A are the sampled values of a function - then interpolated shading gives an approximation of the "true" value (color) of each point in
the plane. Indeed, even in the case above, we can observe that the elements of A are
given by the formula $A(i,j) = 4i + j - 4$ and the colors above are a "picture" of the function
$f(x,y) = 4y + x - 4$.

**Color plots of functions of two variables using `pcolor(X,Y,Z)`**

Consider doing a colorplot of a function $f(x,y)$ in a rectangle $a \leq x \leq b$, $c \leq y \leq d$, where each $(x,y)$ point is to be colored according to some colormap. To do this in MATLAB you need to supply a two-dimensional array (named, for instance, X) of all the x-coordinates of the points, an array, say Y, of the y-coordinates, and then another array, say Z, of the function values. Then `pcolor(X,Y,Z)` will create the color plot.

MATLAB gives you an easy way of preparing the X and Y arrays, using the command `meshgrid` to create rectangular coordinate arrays corresponding to the desired grid points. We'll give an example, using $1 \leq x \leq 4$, $1 \leq y \leq 3$ and the function $f(x,y) = 4y + x - 4$. We'll use spacing of .5 in both the x and y directions. Here is how it goes, with the values in Z displayed in a color plot with interpolated shading.

```matlab
» [X,Y]=meshgrid(1:.5:4,1:.5:3);
» X
X =
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
» Y
Y =
1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
1.5000 1.5000 1.5000 1.5000 1.5000 1.5000 1.5000
2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
2.5000 2.5000 2.5000 2.5000 2.5000 2.5000 2.5000
3.0000 3.0000 3.0000 3.0000 3.0000 3.0000 3.0000
» Z=4*Y+X-4
Z =
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
3.0000 3.5000 4.0000 4.5000 5.0000 5.5000 6.0000
5.0000 5.5000 6.0000 6.5000 7.0000 7.5000 8.0000
7.0000 7.5000 8.0000 8.5000 9.0000 9.5000 10.0000
9.0000 9.5000 10.0000 10.5000 11.0000 11.5000 12.0000
» pcolor(X,Y,Z);colorbar;shading interp
```

The figure is exactly the same (because we chose it that way) as in our previous example of `pcolor(A)`. However, now you have the flexibility to use any rectangular domain in x and y, and we do not even need to restrict ourselves to rectangles, as we will see presently.

Here’s another example of `pcolor`, using the function $f(x,y) = x^2 - 2y^2$:
```
>> [X,Y]=meshgrid(-2:.1:2,-1:.1:1);
>> Z=X.^2-2*Y.^2;
>> pcolor(X,Y,Z)
In the plots above, the contour lines, i.e. the lines of constant values of $z = f(x,y)$ are readily apparent (we know them to be hyperbolas but MATLAB doesn’t know that). If you want to simply extract the contour lines of $Z$, you can use `contour(X,Y,Z)`, or you can specify a certain number of contours, say 10, with `contour(X,Y,Z,10)`.

**Curvilinear coordinates and grids:**

When we use `pcolor(X,Y,Z)`, the arrays $X,Y$ do not have to come from a rectangular grid, but they do need to be rectangular arrays. In particular, they can come from a “curvilinear” recangular grid or, in fact, they can be any arrays, except that the quadrilaterals formed from the coordinates in $X,Y$ should not overlap. Here are some "hand-constructed" examples, where we plot the gridpoints.
You can now define any array $Z$ you want of the same size as $X$ and $Y$. We'll just take a random $3 \times 5$ matrix and plot.

```matlab
Z = rand(3,5);
pcolor(X,Y,Z);pause
shading interp
```

Now curvilinear rectangular grids arise naturally when using other coordinate systems, e.g. polar coordinates. Here we'll do a colorplot of the function $f(x,y) = x^2 - 2y^2$ in the region $1 \leq r \leq 2$, $\pi/4 \leq \theta \leq \pi$. Note how we first prepare a rectangular grid in polar coordinates:

```matlab
X = [-1 2 3 7 10; -2 2 4 6 9; 1 3 4 7 8];
Y = [0 2 1 3 5; 1 4 4 5 6; 7 8 7 10 10];
figure; hold on; for i = 1:3, plot(X(:,i),Y(:,i),X(:,i),Y(:,i),'o'); end
for i = 1:5, plot(X(:,i),Y(:,i),X(:,i),Y(:,i),'o'); end
```
coordinates and then calculate the X and Y arrays using the transformation
\[ x = r \cos \theta, \quad y = r \sin \theta. \]

» [R,T]=meshgrid(1:.1:2,pi/4:pi/16:pi);
» X=R.*cos(T); Y=R.*sin(T);
» Z=X.^2-2*Y.^2;
» pcolor(X,Y,Z);
» axis equal; pause
» shading interp;

Plotting in 3-dimensions:
Instead of using Z to determine color at the points specified in the X,Y arrays, we can plot the points defined by X,Y,Z in three dimensions. This produces a curvilinear surface in space, which, most generally, is allowed to intersect itself. Of course if \( z = f(x,y) \) the surface cannot intersect itself, but it is possible for more general arrays. The main command here is surf(X,Y,Z). Here is an example:

>> [X,Y]=meshgrid(-2:.1:2,-1:.1:1);
>> Z=X.^2-2*Y.^2;
>> surf(X,Y,Z)
>> pause
>> shading interp
The shading here is faceted, by default, but the command
\texttt{shading interp}
will employ interpolated colors and get rid of the grid lines. Each rectangle is colored, by default, according to the value of \( Z \). The surface can be rotated with the plotting tools. If we look straight down at the surface, we will see the plot produced by \texttt{pcolor(X,Y,Z)} because the colors are determined by \( Z \) in both plots. It is certainly possible that you would want other coloring schemes to use for the points, since color represents a 4th dimension (actually not just one, but three additional dimensions!) in which information can be conveyed. In that case the command
\texttt{surf(X,Y,Z,C)}
where \( C \) is an array of the same size as \( X,Y \), and \( Z \) and whose values are used to choose a color from the current colormap. Examples of \( C \) might be the distance of \((x,y,z)\) from the origin, in which case \( C=(X.^2+Y.^2+Z.^2).^(1/2) \) would be used, or perhaps the curvature of the surface at \((x,y,z)\) computed using a mathematical formula.

The command \texttt{surf(X,Y,Z)} will also include a contour plot of the surface (curves of constant \( Z \)) along with the surface itself.
\begin{verbatim}
>> C=X+Y+Z;
>> surf(X,Y,Z,C)
\end{verbatim}

The arrays \( X,Y,Z \) may be defined by parametric equations, with two parameters. Consider for instance the parameters \( \rho \) and \( \theta \) from spherical coordinates, with the spherical coordinate \( \phi \) given by \( \phi = \frac{\pi}{2} \left[ 1 - \frac{1}{2} \sin \left( \frac{\theta}{2} \right) \right] \). Our parameter grid will be \( 0 \leq \theta \leq 4\pi \), \( 0 \leq \rho \leq 1 \). We use the transformation equations
\begin{align*}
x &= \rho \sin \phi \cos \theta \\
y &= \rho \sin \phi \sin \theta \\
z &= \rho \cos \phi
\end{align*}
This is what it looks like in MATLAB, with \( t=\theta \) and \( \rho=\rho \) and \( \phi=\phi \) our variable names in MATLAB. We provide two views of the surface to show the self-intersection.
\[ \begin{align*}
[t, \rho] &= \text{meshgrid}(0:\pi/32:4\pi, 0:.1:1); \\
\phi &= (\pi/2) \times (1-.5\sin(t/2)); \\
X &= \rho \times \sin(\phi) \times \cos(t); \\
Y &= \rho \times \sin(\phi) \times \sin(t); \\
Z &= \rho \times \cos(\phi); \\
surf(X,Y,Z); 
\end{align*} \]