

## MATLAB Lecture 2. Making Plots with MATLAB

Plotting a function consists in two stages: first we must sample the function before rendering the plot. In this lecture we see how to plot curves given as  $y = f(x)$  in rectangular coordinates, or in polar coordinates as  $r = f(t)$ . In three dimensions we can visualize surfaces given as  $z = f(x, y)$  and space curves defined as  $(x(t), y(t), z(t))$ .

In sampling functions, we will encounter the dot before an arithmetic operator which indicates a componentwise rather than a matrix interpretation of the operator.

### 2.1 Basic Plotting

Suppose we want to plot the function  $y = \sin(x)$ , over the interval  $[-\pi, \pi]$ . First we have to sample points from this function, say at evenly spaced points at a distance of  $\pi/20$  apart from each other. Defining this range is done in the first command below:

```
>> x = -pi:pi/20:pi;    % range where we sample
>> s = sin(x);          % generate sample points
>> plot(x,s);           % render the plot
```

By default, MATLAB draws a smooth curve through those sample points. If we wish to plot the cosine function in the same figure, we must use the **hold on** directive:

```
>> hold on              % keep the current plot
>> c = cos(x);          % generate sample points for cosine
>> plot(x,c,'--');     % render the plot
```

The plot of the cosine function now appears as a dashed line. We can also work with colors, and plot only the sample points, instead of a smooth line. With the directive **hold off**, MATLAB will start with a new plot in the current figure.

```
>> hold off             % do not keep previous plots
>> plot(x,c,'r+');     % red + for cosine samples
>> hold on              % keep previous plot
>> plot(x,s,'gx');     % green x for cosine samples
```

We are not really quite happy with the default view window MATLAB offers us. We can change the default axis scaling and appearance to  $[-\pi, \pi]$  for  $x$  and  $[-2, 2]$  for  $y$  as follows:

```
>> axis([-pi,pi,-2,2]) % change axis scaling
```

With **clf** we clear the current figure. If we want to keep the current figure, and make new ones in a new window, you execute the **figure** command. Here is an example of such a session:

```
>> figure              % open new figure window
>> plot(x,c)           % plot cosine in new figure
>> figure(1)           % make figure one current
>> clf                 % clear figure one
```

Exporting or printing the plot goes via the toolbar of the figure window.

## 2.2 Polar Plots

Some plots are better drawn in polar coordinates. Instead of the  $(x, y)$  in rectangular coordinates, we use an angle  $t$  and radius  $r$  to define a point in the plane. The angle  $t$  is unique modulo  $2\pi$ , and we have the relations  $(x = r \cos(t), y = r \sin(t))$ .

As example, we plot Galileo's spiral, defined by  $r = at^2 - l$ . The spiral models the trajectory of a point falling near the earth's equator. We take  $a = 0.01$ ,  $l = 0.02$  and let  $t$  go from 0 to  $10\pi$ .

```
>> t = 0:0.1:10*pi;      % range to sample
>> r = 0.01*t.^2 - 0.02; % sample points
>> polar(t,r)           % make the polar plot
```

Observe that we use the  $\wedge$  operator in the definition of  $r$  to tell MATLAB we want the squaring to be done componentwise.

## 2.3 Three Dimensional Plots

Suppose we wish to plot the function  $z = \cos(xy)$ . As with two dimensional plots, we have to generate a grid of sample points to evaluate the function  $\cos(xy)$ . For example, if we are interested in the values  $(x, y) \in [-2, 2] \times [-2, 2]$ , we can evaluate  $\cos$  at the points

$$\begin{aligned} x_i &= -2 + 0.2(i - 1) & 1 \leq i \leq 21 \\ y_j &= -2 + 0.2(j - 1) & 1 \leq j \leq 21 \end{aligned}$$

The generation of the grid of points where we will sample the function is achieved by

```
>> xa = -2:0.2:2;      % range for x
>> ya = -2:0.2:2;      % range for y
>> [x,y]=meshgrid(xa,ya); % returns two matrices
```

Now that we have our grid, we still must be able to sample. An ordinary multiplication symbol will not do, we have to multiply the grid points componentwise. This is achieved with the  $\cdot^*$  operator:

```
>> z = cos(x .* y);    % evaluate at grid points
>> mesh(x,y,z)         % make the 3d plot
```

We can change the view point with the command **view**:

```
>> view([10,37])       % change horizontal rotation
>> view([-37.5,-20])  % change vertical rotation
>> view(3)            % back to default 3d view
```

The color coding for the surface depends on the height (the values of  $z$ ). The **mesh** command admits a fourth argument which allows the user to define the color coding of the surface. This color coding is often used to make plots of four dimensional objects.

## 2.4 Special Three Dimensional Plots

To model a spring, we can use the space curve with parametric equations  $(\cos(kt), \sin(kt), t)$ , for  $t$  going from 0 to  $2\pi$ . The constant  $k$  controls the period. For example, for  $k = 3$ , the spring completes three turns as  $t$  ranges from 0 to  $2\pi$ . The **plot3** is the 3-d analogue to plot:

```
>> t = 0:0.1:10*pi;           % range to sample
>> plot3(cos(3*t),sin(3*t),t); % make the plot
>> rotate3d                   % adjust view with mouse
```

The **rotate3d** allows us to change the view point: after clicking the mouse button down, with dragging, we can move the axes and set the view into the desired angle. After releasing the mouse button, the data is redrawn. Typing **rotate3d** again turns off this behavior. Alternatively, and perhaps more easily, is to click at the rightmost button of the toolbar of the figure window.

For three dimensional surfaces, we can make contour plots. For example, consider the surface  $z = e^{x+y}$ , for  $x$  and  $y$  ranging from  $-2$  to  $+2$ .

```
>> xr = -2:.1:2; yr = -2:.1:2; % define ranges
>> [x,y]=meshgrid(xr,yr);      % generate grid
>> z = x.*exp(-x.^2 - y.^2);   % sample points
>> contour(z)                  % make contour plot
```

With the command **contour** we plot level curves for  $z$ . Besides  $z$ , we can add the number of contour lines we want to see. Interactively, labels with heights can be added to the level curves:

```
>> h = contour(z,20);         % 20 level curves, h is handle
>> clabel(h,'manual');       % add labels manually
```

## 2.4 Subplots

For the three dimensional surface we created above, we will create a new figure with four subplots with different views:

```
>> figure                     % create new figure
>> subplot(2,2,1)            % set axes for upper left
>> mesh(x,y,z)               % plot with default 3d view
>> subplot(2,2,2)            % activate second subplot
>> mesh(x,y,z)               % plot with default 3d view
>> view([10,10])             % change view
>> subplot(2,2,3)            % activate third subplot
>> mesh(x,y,z)               % plot with default 3d view
>> view([10,70])             % change view
>> subplot(2,2,4)            % activate fourth subplot
>> mesh(x,y,z)               % plot with default 3d view
>> view([10,-30])            % change view
```

The two numbers in the argument of view as used above are two angles (called azimuth and elevation) which define the viewer's eye position. Changing only the first angle makes the plot rotate around the vertical coordinate axis. To see the plot from above or below, we must change the second angle.

## 2.5 Assignments

1. Do  $\mathbf{x} = [1 \ 2 \ 3]$ ;  $\mathbf{y} = [3 \ 2 \ 1]$ ; followed by  $\mathbf{x}.*\mathbf{y}$  and  $\mathbf{x}*\mathbf{y}'$ . Explain the difference between  $\mathbf{x}.*\mathbf{y}$  and  $\mathbf{x}*\mathbf{y}'$ , describing in your own words what happened in  $\mathbf{x}.*\mathbf{y}$  and  $\mathbf{x}*\mathbf{y}'$ .
2. Make a plot of the function  $f(x) = e^{-x^2} \sin(\pi x^3)$  over the interval  $[-2, 2]$ .
3. Plot the 10-th Chebyshev polynomial defined by  $y = \cos(10 \arccos(x))$ . The inverse cosine is implemented by the function `acos`, and is of course only defined for  $x \in [-1, 1]$ . Because the polynomial is oscillating, make sure to take the sampling range for  $x$  fine enough.
4. The curves  $r = R \sin(kt)$  are called *roses*. Take  $R = 1$  and make several plots for different values of  $k$ . Make a table of two rows and three columns for the respective values  $k = 4, 5, 6$  in the top row and  $k = \frac{2}{3}, \frac{4}{3}, \frac{5}{3}$  in the bottom row.
5. The *inverse Galileo spiral* for  $l = 0$  is defined by  $r = \frac{a}{t^2}$ . Take  $a = 100$  and choose a suitable range for  $t$ . Make several subplots for different ranges of  $t$  to illustrate that the trajectory modeled by this spiral is the inverse of the one we have seen.
6. The twisted cubic is a space curve defined as the intersection of two cylinders:  $y = x^2$  and  $z = x^3$ . Make a window with three subplots, and
  - (a) in the first subplot, plot the space curve;
  - (b) in the second subplot, plot the two cylinders; and
  - (c) plot the space curve AND the two cylinders in the third subplot.

Adjust the viewpoints so that you get a feeling of how the twist in the curve appears as a consequence of turning the defining cylinders.

7. Make a plot of the epicycloid (see [1]) defined by

$$\begin{cases} x(t) = (a + b) \cos(t) - b \cos((a/b + 1)t) \\ y(t) = (a + b) \sin(t) - b \sin((a/b + 1)t) \end{cases} \quad \text{for } a = 12, b = 5, t \in [0, 10\pi].$$

8. Consider the surface  $z = x^3 - 3xy^2$ .
  - (a) Make a plot over the grid  $[-1, 1] \times [-1, 1]$ .
  - (b) Make four subplots for different view points.
9. Closed curves on a sphere are defined by  $x = \cos(mt) \cos(nt)$ ,  $y = \sin(mt) \sin(nt)$ ,  $z = \sin(nt)$ , for  $t \in [-\pi, \pi]$ . Make a table of plots, for  $m = 2, 5, 7$  and  $n = 5, 8, 10$ . Make sure to take enough samples to see smooth curves.

## References

- [1] D.J. Higham and N.J. Higham. *MATLAB Guide*. SIAM, 2000.