SGN-84006 Introduction to Scientific Computing with Matlab

Lecture 4: Data Visualization

Alessandro Foi
Department of Signal Processing
alessandro.foi@tut.fi

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Outline

- Basics: figure, axes, handles, properties;
- Plotting univariate and multivariate data;
- Shading, Lighting, Alpha-blending, Transparency;
- Continuous drawing, Animations.

Handles and Properties

- Any figure as well as any of the figure's contents are univocally identified by numerical handles:

```
figure_handle=figure;
axis_handle=subplot(2,1,2);
plot_handle=plot(randn(1e5,1),randn(1e5,1),'.');
```

- Handles allow to inspect and modify any property of figures, axes, and various graphical entities.

Handles and Properties

- Handles of the current figure and axis are given by the commands:

```
>> gcf

ans =

     1

>> gca

ans =

    1.731507568359375e+002
```

- The figure handle typically coincides with the figure number shown in the title of the figure window.

Handles and Properties

- Available properties and their current values are accessed using the *get* command:

```
>> get(axis_handle)
ActivePositionProperty = position
ALim = [0 1]
ALimMode = auto
AmbientLightColor = [1 1 1]
Box = on
CameraPosition = [-2.5 0 17.3205]
CameraPositionMode = auto
CameraTarget = [-2.5 0 0]
CameraTargetMode = auto
CameraUpVector = [0 1 0]
CameraUpVectorMode = auto
CameraViewAngle = [6.60861]
CameraViewAngleMode = auto
CLim = [0 1]
CLimMode = auto
Color = [1 1 1]                                          ...
```

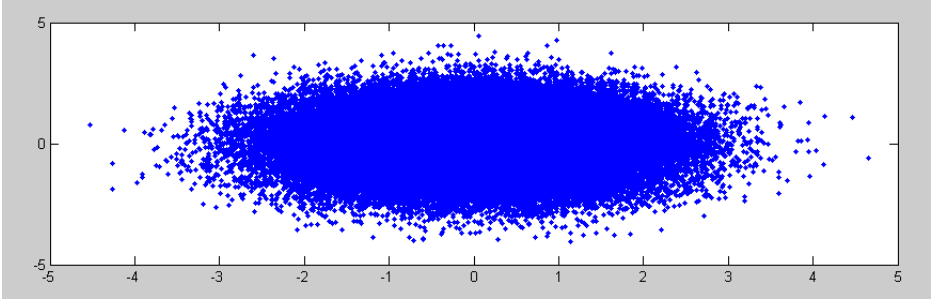
Handles and Properties

- Available properties and their current values are accessed using the *get* command:

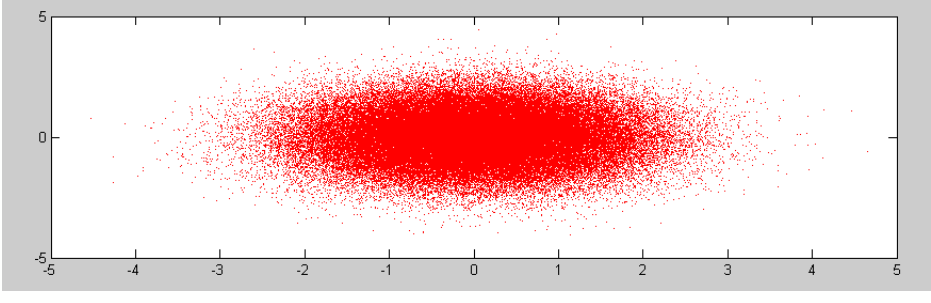
```
>> get(plot_handle)
           DisplayName: ''
            Annotation: [1x1 hg.Annotation]
                 Color: [0 0 1]
             LineStyle: 'none'
             LineWidth: 0.500000000000000
                Marker: '.'
            MarkerSize: 6
       MarkerEdgeColor: 'auto'
       MarkerFaceColor: 'none'
                 XData: [1x100000 double]
                 YData: [1x100000 double]
                 ZData: [1x0 double]
          BeingDeleted: 'off'
        ButtonDownFcn: []
              Children: [0x1 double]
              Clipping: 'on'                                    ...
```

Handles and Properties

- Values of properties can be modified using the *set* command.




```
>> set(plot_handle,'MarkerEdgeColor',[1 0 0],'MarkerSize',2)
```



Plotting data

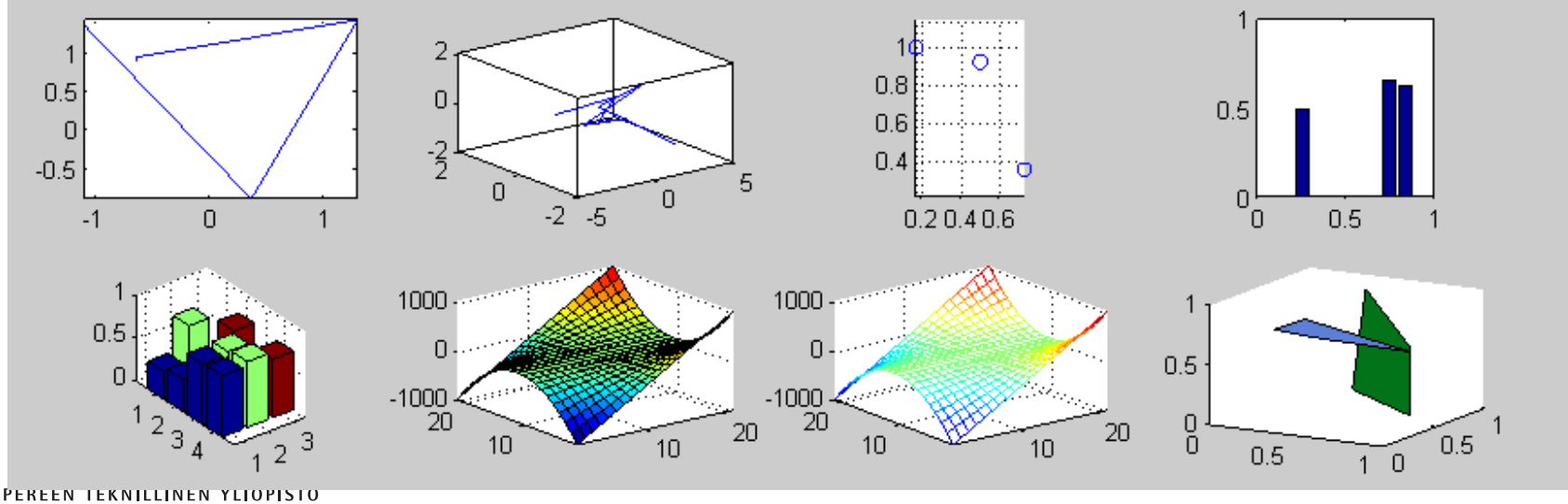
Matlab boasts a plethora of functions for plotting data: *plot*, *plot3*, *mesh*, *surf*, *surfl*, *scatter*, *scatter3*, *stem*, *bar*, *bar3*, *imagesc*, *patch*, etc.
They can be roughly divided in two classes:

- plots of univariate data (e.g., 1-D plots in 2-D, or plots of curves in 2-D or 3-D)
  - each data sample is "linked" with previous and next sample; data typically represented as a vector.
- plots of bivariate data (e.g., meshes, surfaces).
  - each data sample is "linked" with its "neighbours"; data is typically represented as a matrix.

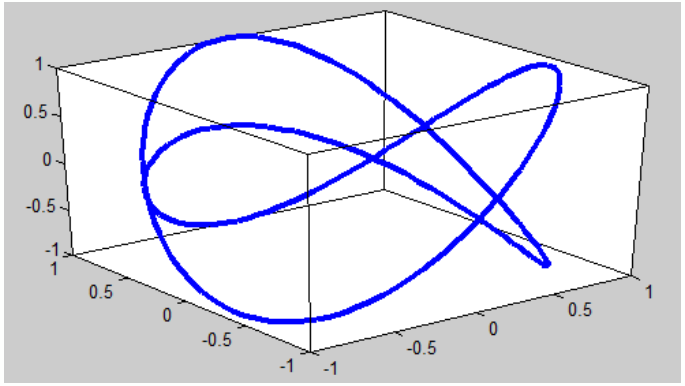
Axes are always 3-D, even when everything is 2-D.

Plotting data

```
subplot(2,4,1), plot(randn(1,5),randn(1,5)),axis tight
subplot(2,4,2), plot3(randn(1,8),randn(1,8),randn(1,8)),box on
subplot(2,4,3), scatter(rand(1,3),rand(1,3)),axis equal, grid on
subplot(2,4,4), bar(rand(1,3),rand(1,3)),axis square
subplot(2,4,5), bar3(rand(4,3))
subplot(2,4,6), surf(([-10:10]'.^2*[-10:10])),axis tight
subplot(2,4,7), mesh(([-10:10]'.^2*[-10:10])),axis tight
subplot(2,4,8), patch(rand(4,2),rand(4,2),rand(4,2),rand(1,2,3)),view(30,20)
```

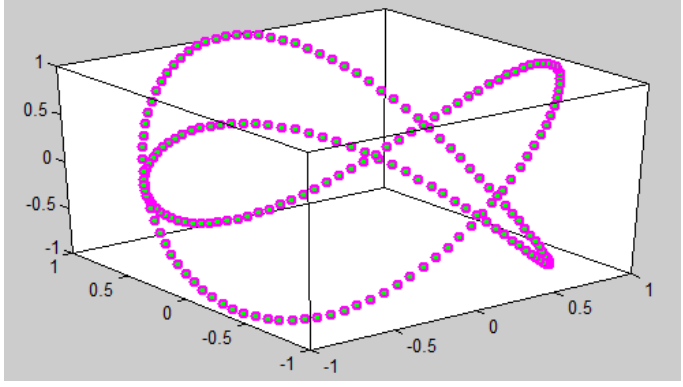


plot3: curve in 3-D




```
>> help plot3
 PLOT3  Plot lines and points in 3-D space.
    PLOT3(x,y,z), where x, y and z are three vectors of the same length,
    plots a line in 3-space through the points whose coordinates are the
    elements of x, y and z.

    PLOT3(X,Y,Z), where X, Y and Z are three matrices of the same size,
    plots several lines obtained from the columns of X, Y and Z.

>> plot_handle=plot3(sin(linspace(0,4*pi,200)),cos(linspace(0,4*pi,200)),cos(linspace
      (0,6*pi,200)))
>> set(plot_handle,'LineWidth',3),   box on,    camproj('persp'),
```

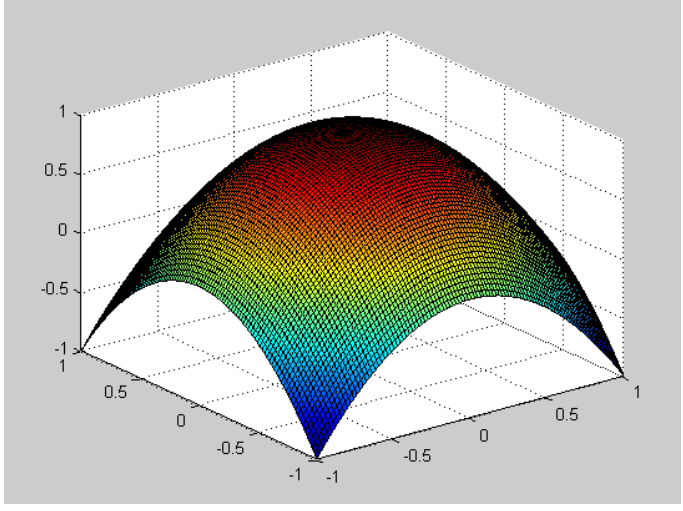
Using *plot3* for scatterplots in 3-D




```
>> set(plot_handle,'LineStyle','none','Marker','o','MarkerFaceColor',[0 1 0],'
      MarkerEdgeColor',[1 0 1],'MarkerSize',5,'LineWidth',1.5)
```

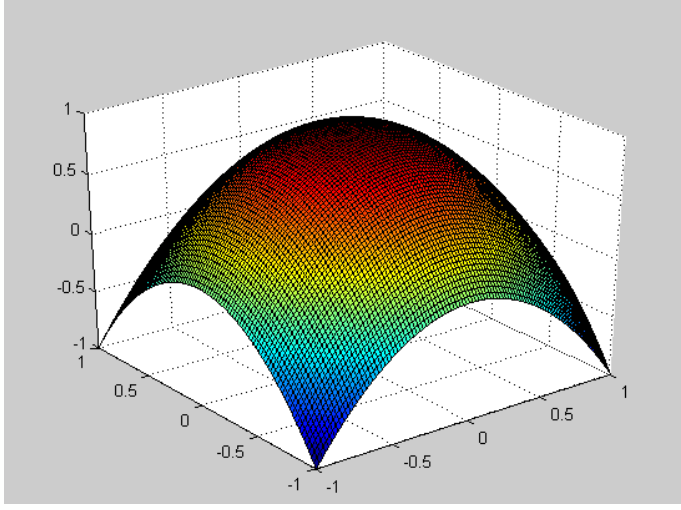
.

surf: default



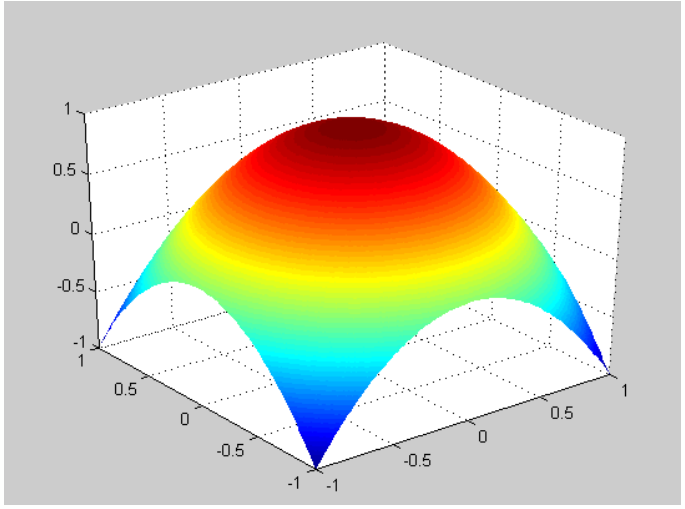

```
[X, Y]=meshgrid(linspace(-1,1,100),linspace(-1,1,100));  % create X and Y coordinates
Z = 1 - X.^2 - Y.^2;    % parabolic surface
surf_handle=surf(X,Y,Z);     % draw surface
```

Enable perspective rendering



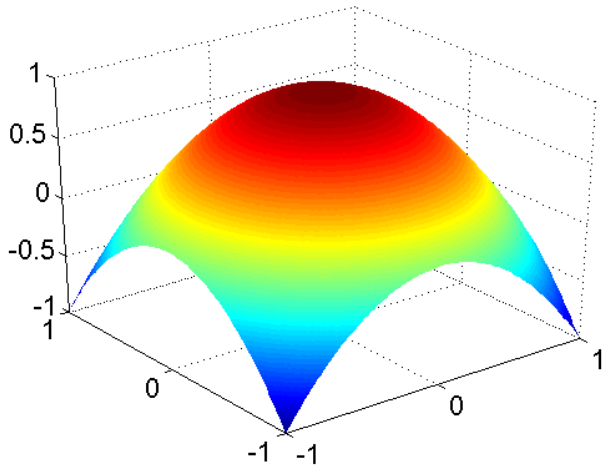

```
>> camproj('persp')  % switch from orthographic to perspective projection
```

surf: improve rendering



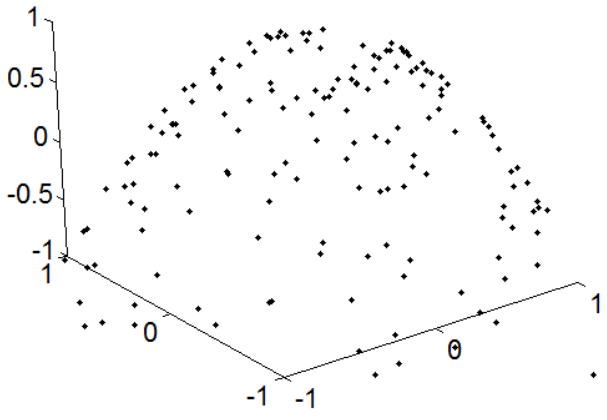

```
>> shading interp     % changes color shading

>> lighting phong     % better rendering quality
```

Prepare figure for print output




```
>> set(gca,'FontSize',16)       % prepare for print output (larger fonts)
>> set(gcf,'Color',[1 1 1])     % prepare for print output (white background)
```

Surfaces from irregularly sampled data

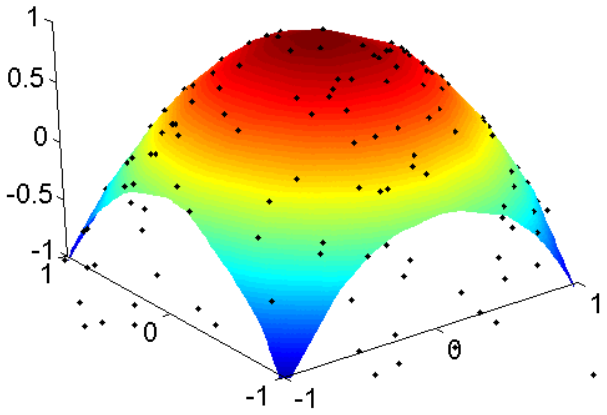



```
X=randn(2e2,1);  Y=randn(2e2,1);    % random sampling points
Z = 1 - X.^2 - Y.^2;                % parabolic surface sampled at (X,Y)
plot_handle=plot3(X,Y,Z);           % draw scatterplot
set(plot_handle,'Marker','.','LineStyle','none','Color',[0 0 0])
set(gca,'xlim',[-1 1],'ylim',[-1 1],'zlim',[-1 1])   % set limits of axes
camproj('persp')                % switch from orthographic to perspective projection
set(gca,'FontSize',16)          % prepare for print output (larger fonts)
set(gcf,'Color',[1 1 1])        % prepare for print output (white background)
```

TriScatteredInterp: interpolation the easy way

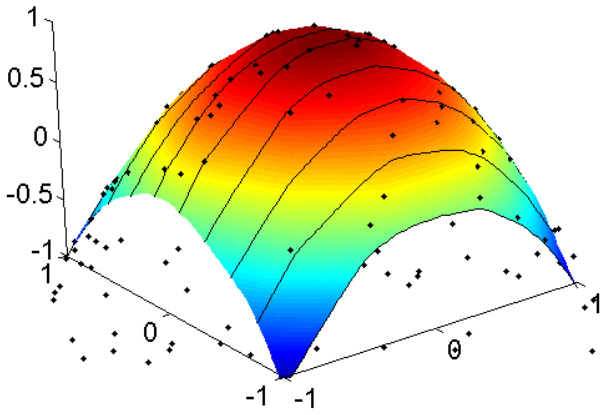
```
>> help TriScatteredInterp
  TriScatteredInterp   Scattered data interpolant
     TriScatteredInterp is used to perform interpolation on a scattered
     dataset that resides in 2D/3D space. A scattered data set defined by
     locations X and corresponding values V can be interpolated using a
     Delaunay triangulation of X. This produces a surface of the form V = F(X).
     The surface can be evaluated at any query location QX, using QV = F(QX),
     where QX lies within the convex hull of X. The interpolant F always
     goes through the data points specified by the sample.

     F = TriScatteredInterp(X, V) Creates an interpolant that fits a surface
     of the form V = F(X) to the scattered data in (X, V). X is a matrix
     of size mpts-by-ndim, where mpts is the number of points and ndim is
     the dimension of the space where the points reside, ndim >= 2. V is a
     column vector that defines the values at X, where the length of V
     equals mpts.

     F = TriScatteredInterp(X, Y, V) and F = TriScatteredInterp(X, Y, Z, V)
     allow the data point locations to be specified in alternative column
     vector format when working in 2D and 3D.
```

Surfaces from irregularly sampled data



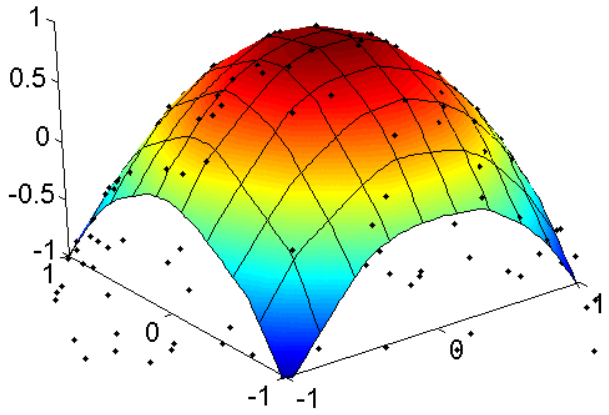

```
F=TriScatteredInterp(X(:),Y(:),Z(:));   % create interpolant function

[Xgrid, Ygrid]=meshgrid(linspace(-1,1,100),linspace(-1,1,100));  % create grids of X
      and Y coordinates
hold on         %  holds the current plot
surf_handle=surf(Xgrid,Ygrid,F(Xgrid,Ygrid));   % draw surface
shading interp    % changes color shading quality
```

Draw coarse mesh over finely sampled surface



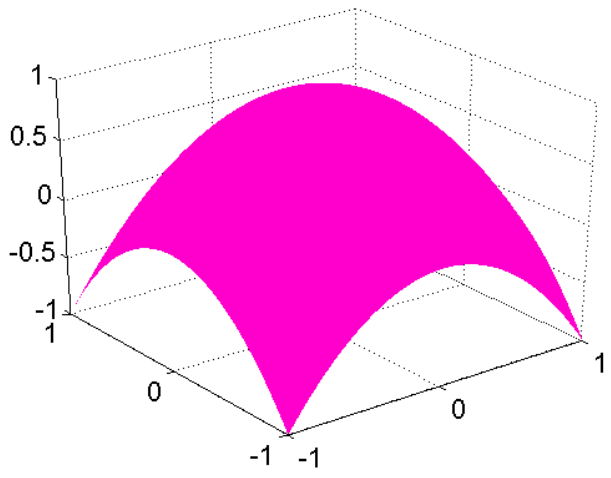

```
hold on

for y=linspace(-1,1,10)
    x=linspace(-1,1,100);
    plot_handle=plot3(x,y(ones(1,100)),F(x,y(ones(1,100))));
    set(plot_handle,'color',[0 0 0])
end
```

Draw coarse mesh over finely sampled surface



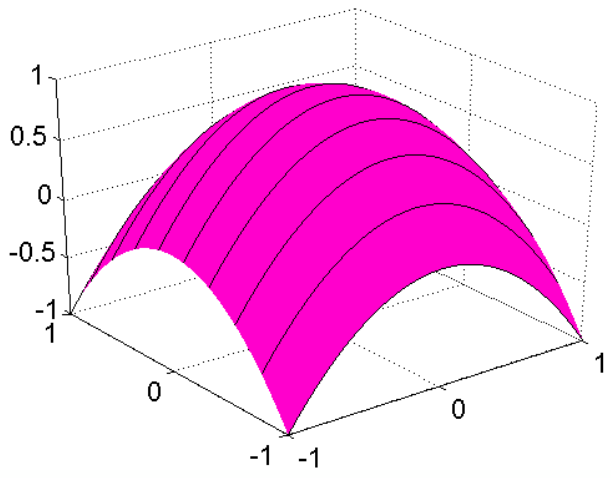

```
for x=linspace(-1,1,10)
    y=linspace(-1,1,100);
    plot_handle=plot3(x(ones(1,100)),y,F(x(ones(1,100)),y));
    set(plot_handle,'color',[0 0 0])
end
```

surf: uniform surfaces are ambiguous



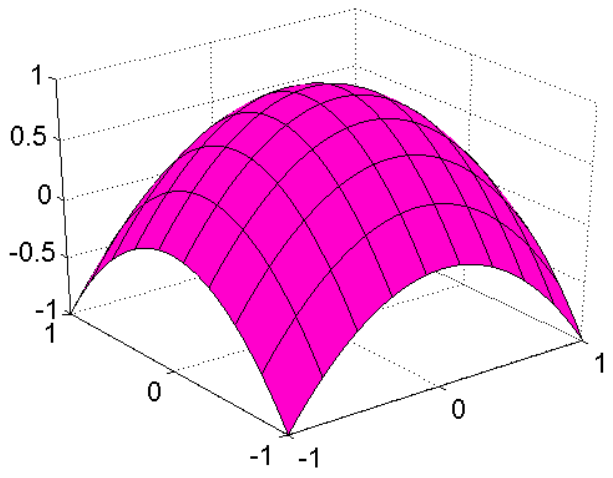

```
[X, Y]=meshgrid(linspace(-1,1,100),linspace(-1,1,100));  % create X and Y coordinates
Z = 1 - X.^2 - Y.^2;        % parabolic surface
surf_handle=surf(X,Y,Z);    % draw surface
set(surf_handle,'FaceColor',[1 0.0 0.8],'EdgeColor','none')  % set single color and
     no mesh
set(gca,'FontSize',16,'Color',[1 1 1])
set(gcf,'Color',[1 1 1])
camproj('persp')  % switch from orthographic to perspective projection
```

surf: disambiguation by coarse mesh



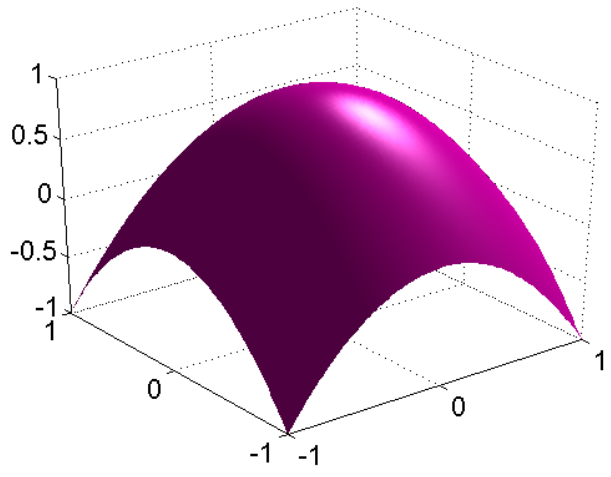

```
F=TriScatteredInterp(X(:),Y(:),Z(:));   % create interpolant function
% draw coarse mesh
hold on
x=linspace(-1,1,100);
for y=linspace(-1,1,10)
    plot_handle=plot3(x,y(ones(1,100)),F(x,y(ones(1,100))));
    set(plot_handle,'color',[0 0 0])
end
```

surf: disambiguation by coarse mesh



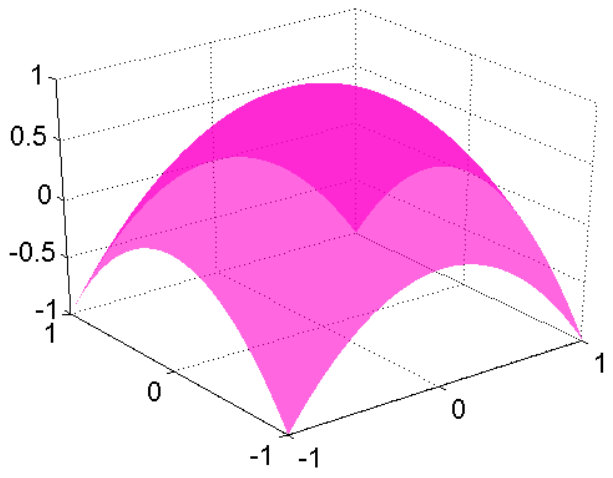

```
% and now along the other direction:


y = linspace(-1,1,100);
for x = linspace(-1,1,10)
    plot_handle = plot3( x(ones(1,100)) , y , F(x(ones(1,100)),y) );
    set(plot_handle,'color',[0 0 0])
end
```

surf: disambiguation by lighting



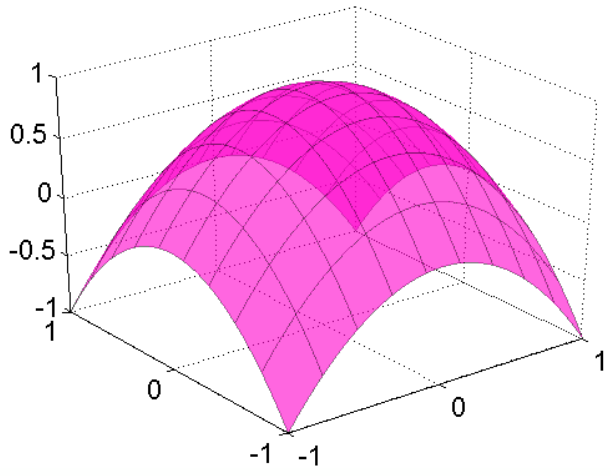

```
lighting phong     % better rendering quality


light              % adds a light to the scene
% camlight         % adds a light to the scene
```

surf: disambiguation by transparency



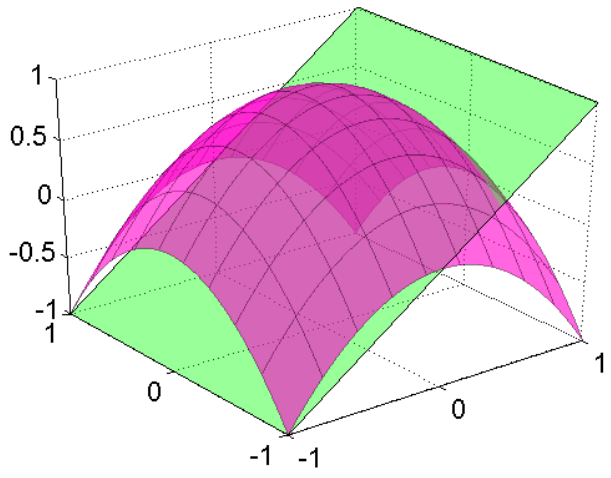

```
set(surf_handle,'FaceAlpha',0.6)    % alpha blending


% small alpha makes surfaces and patches very transparent
% alpha = 1 makes them fully opaque
```

Alpha blending is only for patches or surfaces



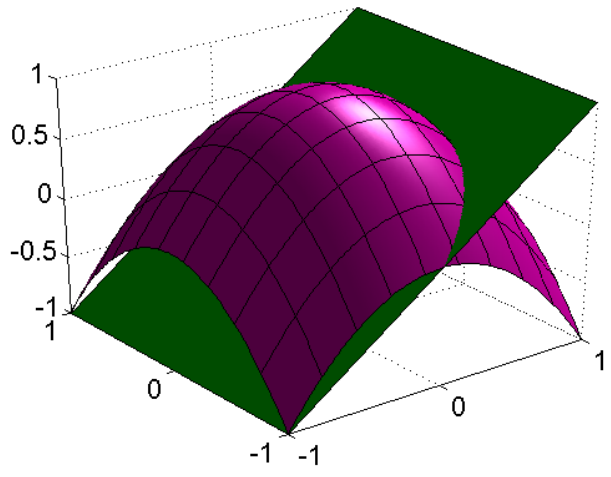

```
% HOW TO DRAW ALPHA-BLENDING PLOTS??
% draw coarse mesh using surf instead of plot to exploit alpha-blending
x=linspace(-1,1,100);
for y=linspace(-1,1,10)
    my_handle=surf([x;x],y(ones(2,100)),F([x;x],y(ones(2,100))));
    set(my_handle,'EdgeColor',[0 0 0],'FaceColor','none','EdgeAlpha',0.3)
end
                                                ... and similar for the other direction
```

Seeing through a transparent *patch*



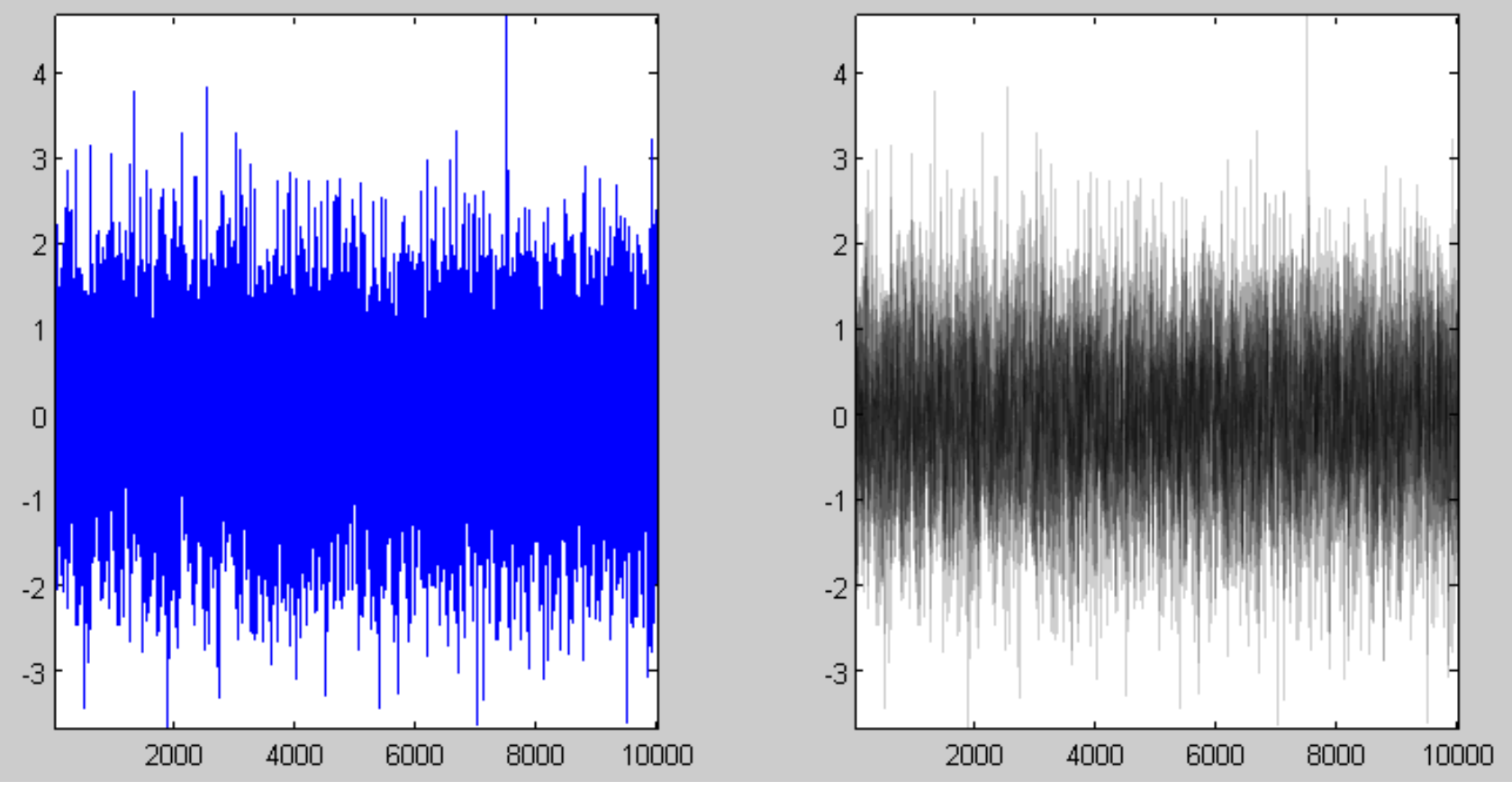

```
y=linspace(-1,1,100);
for x=linspace(-1,1,10)
    my_handle=surf(x(ones(2,100)),[y;y],F(x(ones(2,100)),[y;y]));
    set(my_handle,'EdgeColor',[0 0 0],'FaceColor','none','EdgeAlpha',0.3)
end

patch_handle=patch([-1 -1 1 1 -1],[-1 1 1 -1 -1],[-1 -1 1 1 -1],[0 1 0]);
set(patch_handle,'faceAlpha',0.4)
```

Opaque objects are easier to understand




```
[X, Y]=meshgrid(linspace(-1,1,100),linspace(-1,1,100));  % create X and Y coordinates
Z = 1 - X.^2 - Y.^2;        % parabolic surface
surf_handle=surf(X,Y,Z);    % draw surface
set(surf_handle,'FaceColor',[1 0.0 0.8],'EdgeColor','none')
set(gca,'FontSize',16,'Color',[1 1 1]),   set(gcf,'Color',[1 1 1])
camproj('persp')  % switch from orthographic to perspective projection
lighting phong    % better rendering quality
light             % adds a light to the scene
```

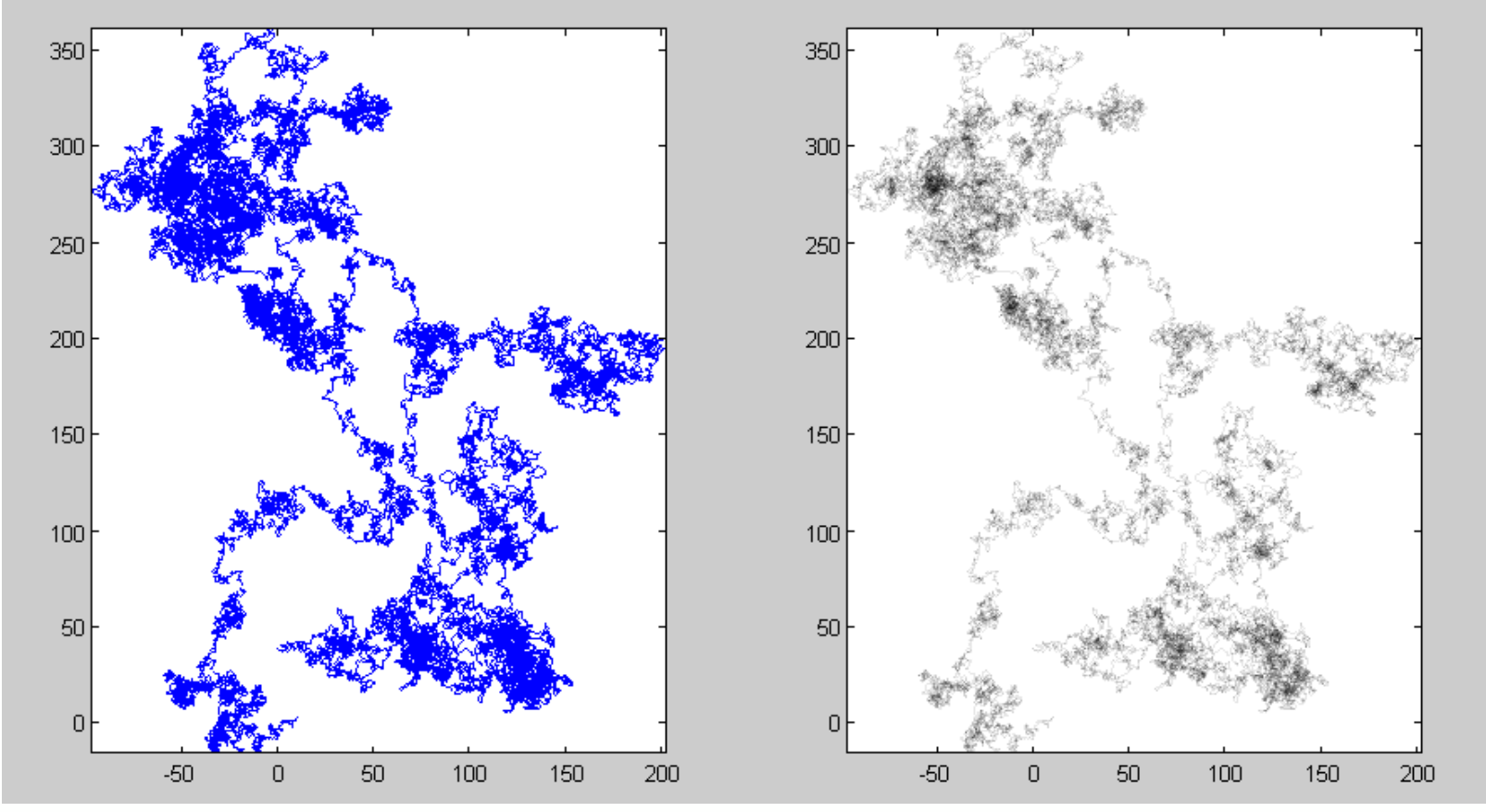
Using alpha-blending to battle crowding and to highlight trends in data: Noise



Using alpha-blending to battle crowding and to highlight trends in data: Noise

```
% Noise
close all
y=randn(1,1e4);  % random noise
figure
subplot(1,2,1)
plot(y)              % plot noise
axis tight       % set margins of axis
subplot(1,2,2)
surf_handle=surf([1:size(y,2);1:size(y,2)],[y;y],zeros(2,size(y,2)));  % plot curve
     as surface
set(surf_handle,'EdgeAlpha',0.05)           % enable alpha blending
view([0 90])           % set view to x-y plane
axis tight
grid off               % disable grid
box on                 % enable box (frame around axis)
```

Using alpha-blending to battle crowding and to highlight trends in data: Brownian motion



Using alpha-blending to battle crowding and to highlight trends in data: Brownian motion

```
% Brownian motion
close all
delta_x=randn(1,5e4);  % random shifts in x
delta_y=randn(1,5e4);  % random shifts in y
x=cumsum(delta_x);     % path in x
y=cumsum(delta_y);     % path in y
figure
subplot(1,2,1)
plot(x,y)              % plot path
axis equal tight       % set margin and scale of axis
subplot(1,2,2)
surf_handle=surf([x;x],[y;y],zeros(2,size(x,2)));  % plot path curve as surface
set(surf_handle,'EdgeAlpha',0.05)           % enable alpha blending
view([0 90])           % set view to x-y plane
axis equal tight       % set margin and scale of axis
grid off               % disable grid
box on                 % enable box (frame around axis)
```

Continuous drawings and Animations

Continuous drawing is usually managed through a timed loop:

```
plotEveryNseconds=0.1;  % how often to draw?

timeOld=now;

while 1

    % HERE GO THE INSTRUCTIONS OF MAIN ALGORITHM

    if (now-timeOld)*(60*60*24)>plotEveryNseconds  % begin of plotting subroutine

        % HERE GO THE INSTRUCTIONS FOR PLOTTING FIGURE

        drawnow          %% issue drawnow to force drawing
        pause(0.0001);   %% issue a pause to allow drawing

        timeOld=now;     %% reset timer
    end                                          % end of plotting subroutine
end
```

Continuous drawing example: Gravitation