Matlab graphics

On this lecture we'll focus on the graphical visualizations in Matlab:

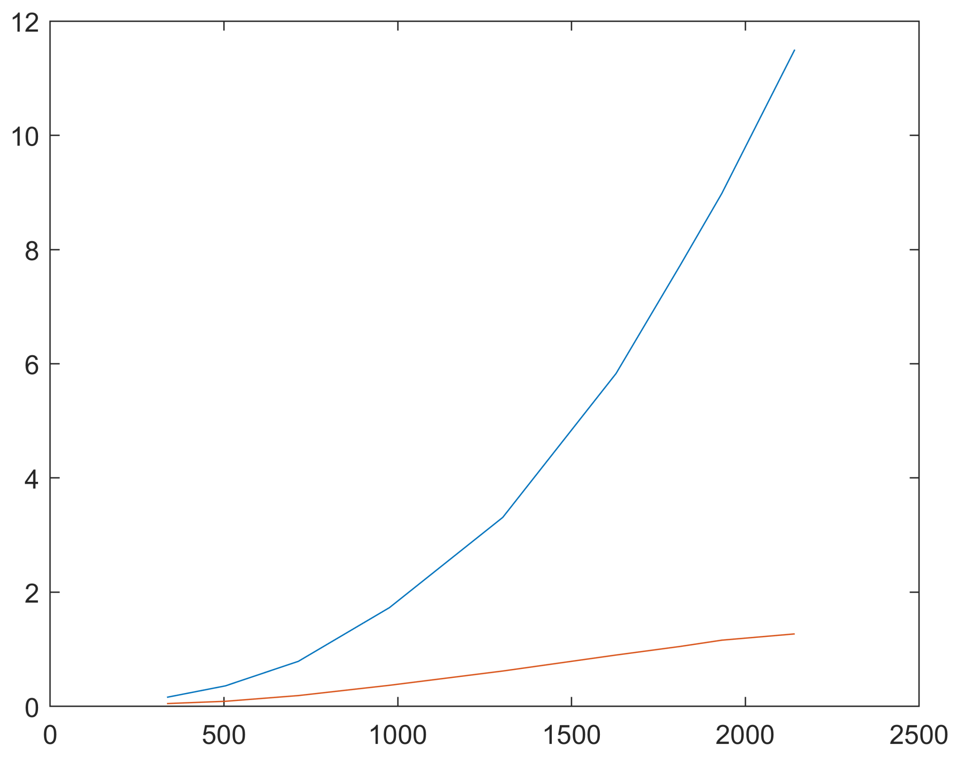
2D Visualization

Let us consider a problem in hydraulic engineering. A basic problem in planning of water storage reservoirs is the morphological characteristic curve of the reservoir, which gives volume and area variation as a function of water height. Load data from **reservoir.txt**, separate the variables (the coloumns), and plot the volume and the surface of the reservoir in function of the water level on one figure.

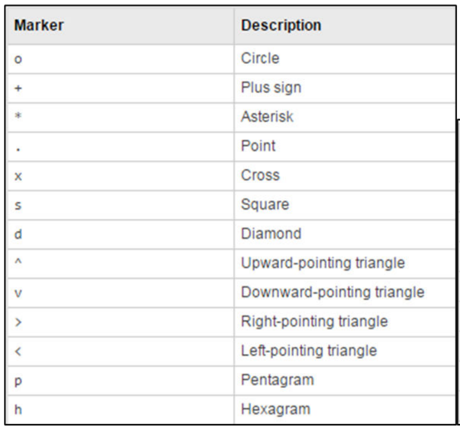
```
clc; clear all; close all;
% Loading data
format shortG
data1 = load('tarozas.txt')
```

```
data1 = 9×3
         336         0.16         0.05
         504         0.36         0.09
         714         0.79         0.19
         976         1.73         0.37
        1302         3.31         0.62
        1628         5.83          0.9
        1812         7.72         1.05
        1932         8.98         1.16
        2142         11.5         1.27
```

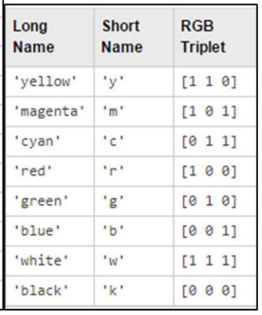
```
H = data1(:,1); % water level [cm]
V = data1(:,2); % volume, million cubic meter
F = data1(:,3); % surface, square kilometers
% Visualize the data
figure(1);
plot(H,V,H,F)
```



```
plot(H,V);
hold on;
h2 = plot(H,F,'om--','LineWidth',2);
```



Marker	Description
o	Circle
+	Plus sign
*	Asterisk
.	Point
x	Cross
s	Square
d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
p	Pentagram
h	Hexagram

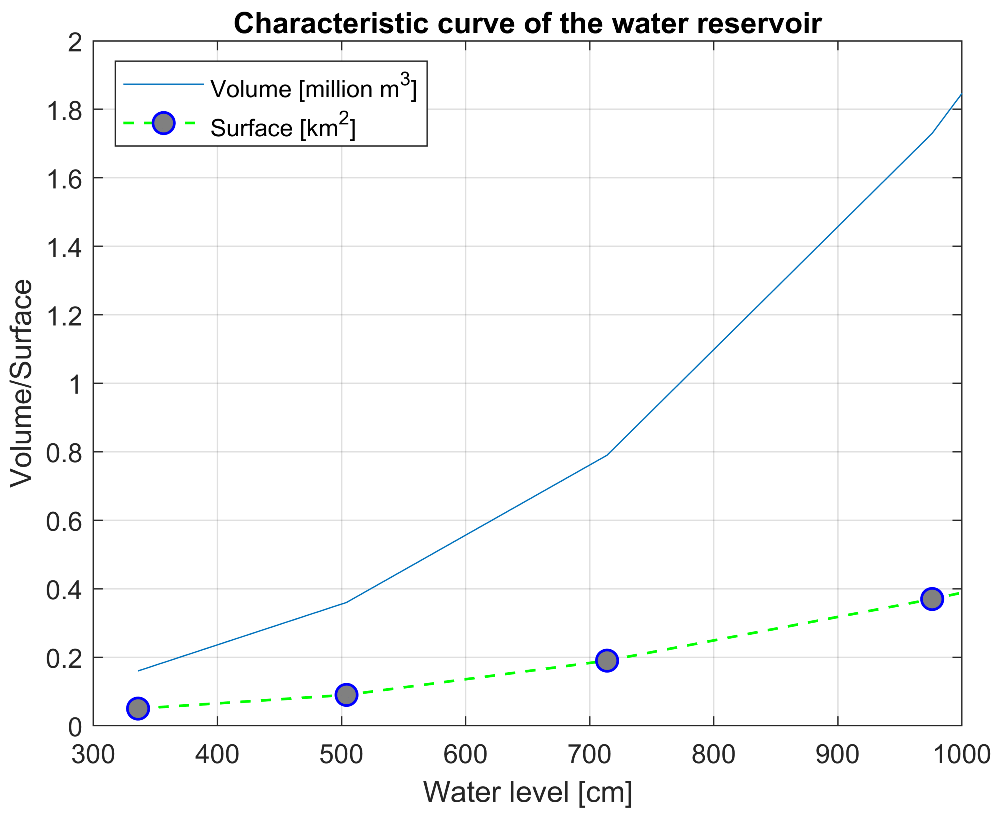


Long Name	Short Name	RGB Triplet
'yellow'	'y'	[1 1 0]
'magenta'	'm'	[1 0 1]
'cyan'	'c'	[0 1 1]
'red'	'r'	[1 0 0]
'green'	'g'	[0 1 0]
'blue'	'b'	[0 0 1]
'white'	'w'	[1 1 1]
'black'	'k'	[0 0 0]

```
get(h2)
```

```
    AlignVertexCenters: off
            Annotation: [1×1 matlab.graphics.eventdata.Annotation]
          BeingDeleted: off
            BusyAction: 'queue'
         ButtonDownFcn: ''
              Children: [0×0 GraphicsPlaceholder]
              Clipping: on
                 Color: [1 0 1]
             ColorMode: 'manual'
           ContextMenu: [0×0 GraphicsPlaceholder]
             CreateFcn: ''
       DataTipTemplate: [1×1 matlab.graphics.datatip.DataTipTemplate]
             DeleteFcn: ''
           DisplayName: ''
      HandleVisibility: 'on'
               HitTest: on
         Interruptible: on
              LineJoin: 'round'
             LineStyle: '--'
         LineStyleMode: 'manual'
             LineWidth: 2
                Marker: 'o'
       MarkerEdgeColor: 'auto'
       MarkerFaceColor: 'none'
         MarkerIndices: [1 2 3 4 5 6 7 8 9]
            MarkerMode: 'manual'
            MarkerSize: 6
                Parent: [1×1 Axes]
         PickableParts: 'visible'
              Selected: off
    SelectionHighlight: on
           SeriesIndex: 2
                   Tag: ''
                  Type: 'line'
              UserData: []
               Visible: on
                 XData: [336 504 714 976 1302 1628 1812 1932 2142]
             XDataMode: 'manual'
           XDataSource: ''
                 YData: [1×9 double]
           YDataSource: ''
                 ZData: [1×0 double]
           ZDataSource: ''
```

```
set(h2,'Color','g','LineWidth',1,'MarkerSize',8,...
        'MarkerEdgeColor','b','MarkerFaceColor', [0.5,0.5,0.5]);
% Titles, legend, coordinate system grid, setting axis limits
xlabel('Water level [cm]')
ylabel('Volume/Surface')
title('Characteristic curve of the water reservoir')
grid on;
axis([300,1000,0,2])
legend('Volume [million m^3]','Surface [km^2]','Location','NW')
```



```
% legend(h2,'Surface [km^2]')
```

You can also delete named graphs from your figure with the **delete** command. Use the **close** command to close the current figure, and the **close all** command to close all.

Plotting functions

The **plot** function is meant to visualize discrete data series. If you have a mathematical formula, it can only help if you create a series of points along with the plotted function. If your formula is in an explicit form, the **fplot** or **ezplot** functions do the work (depending on which version you use, the latter might not be supported). The function fplot can only handle single-variable expressions, if you have a bivariate curve, use the **fimplicit** function.

Example:

Earth curvature and refraction:

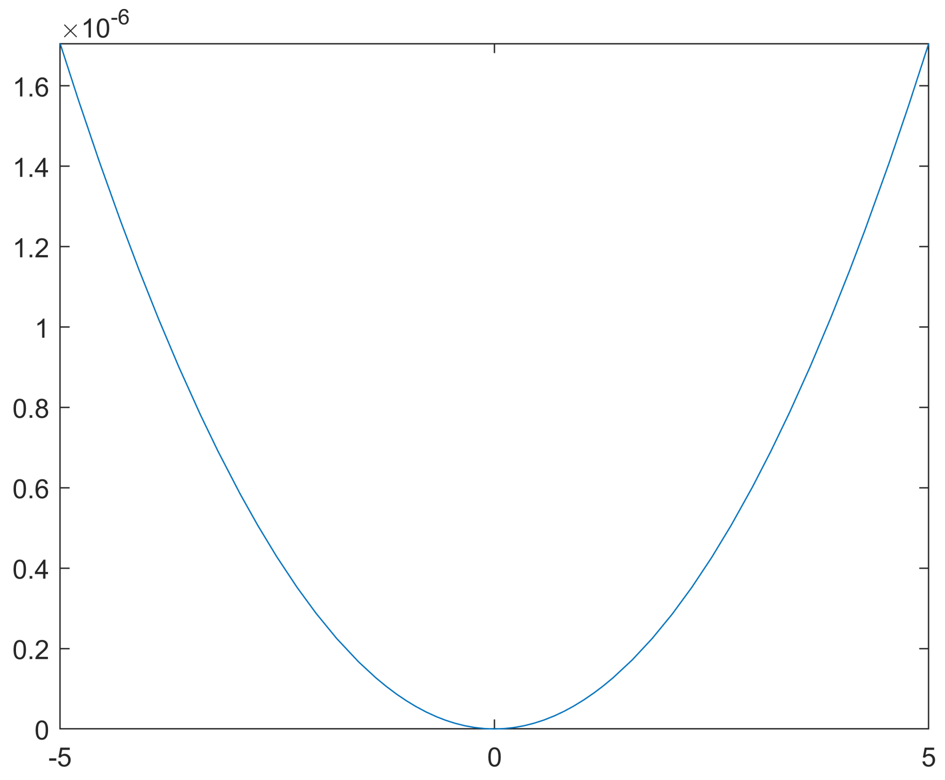
$$\Delta = \left(1 - k\right) \cdot \frac{d^2}{2 \cdot R}$$

where the refraction coefficient is k=0.13, and the earths' radius is R = 6378000;

```
R = 6378000; k = 0.13;
% Function definition as anonymous function:
delta = @(d) (1-k)*d.^2/(2*R)
```

```
delta = function_handle with value:
    @(d)(1-k)*d.^2/(2*R)
```

```
figure(3)
fplot(delta) % OR ezplot(delta)
```



```
fplot(delta,[0,1000]) % OR ezplot(delta,[0,1000])
hold on;
plot(xlim,[0.01,0.01]) % OR plot([0,1000],[0.01,0.01])
title('Effect of earth curvature and refraction on trigonometric elevation measurements')
xlabel('Horizontal distance [m]')
ylabel('Magnitude of error effect on height [m]')
% fplot uses the same setting for formatting
fplot(delta,[0,1000],'g--','LineWidth',2)
h3 = ezplot(delta,[0,1000])
```

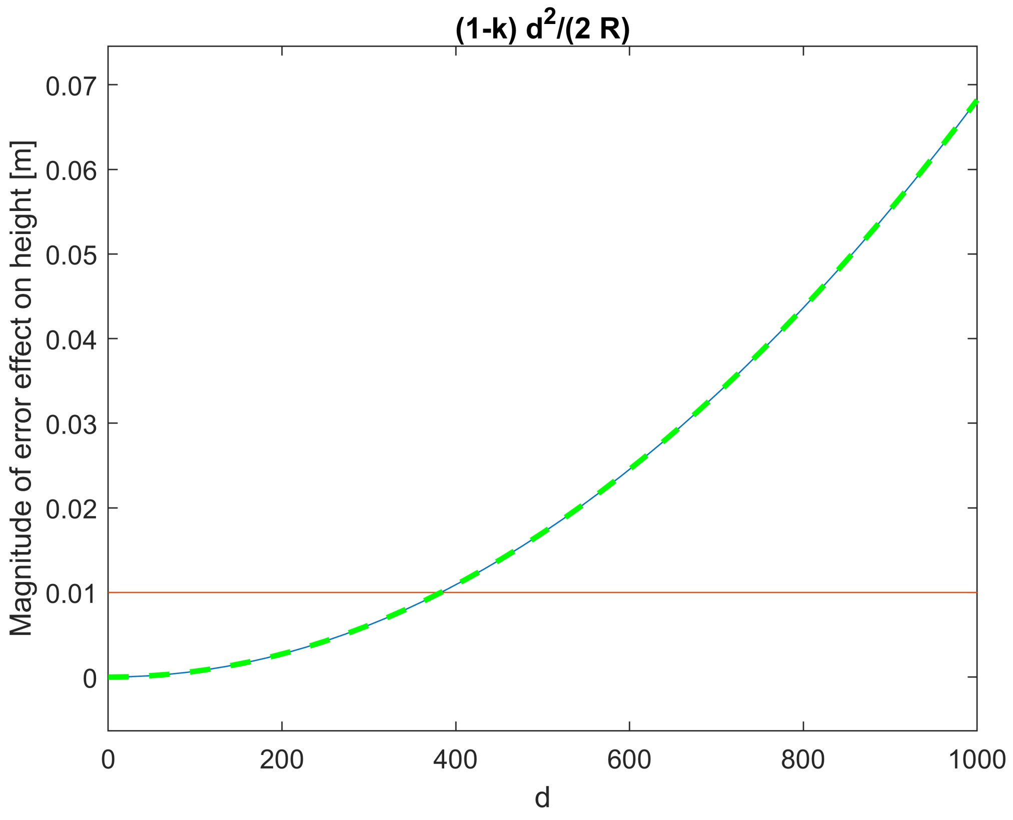
```
h3 = 
  Line with properties:

              Color: [0.494 0.184 0.556]
          LineStyle: '-'
          LineWidth: 0.5
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [1×434 double]
              YData: [1×434 double]
              ZData: [1×0 double]

  Show all properties
```

```
% ezplot needs the set function to change the graphical properties
```

```
set(h3,'Color','g','LineStyle','--','LineWidth',2)
```



3D Visualizations

So far, we have only used the two-dimensional visualization options, but often there is a need for spatial, 3D representations as well, be it a representation of a spatial curve or a surface. Let's look at some examples of these as well.

Representation of a 3D parametric curve

Plot the two spatial spirals given in the following parametric form!

$x1 = \cos(2 \bullet \pi \bullet t);\ y1 = \sin(2 \bullet \pi \bullet t);\ z1 = t$

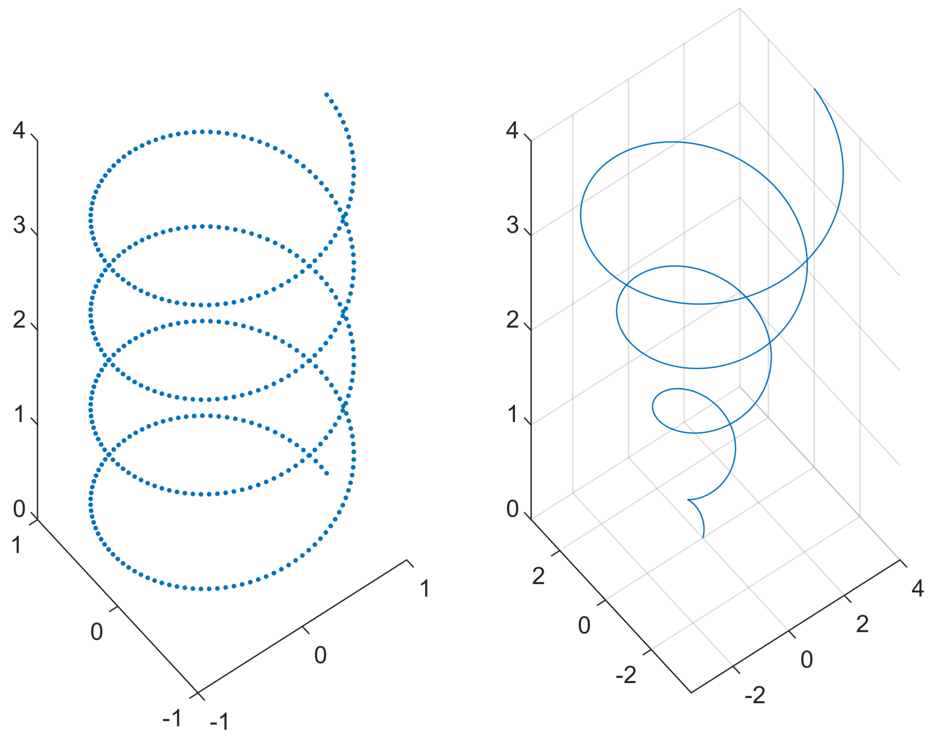
$x2 = t \bullet \cos(2 \bullet \pi \bullet t);\ y2 = t \bullet \sin(2 \bullet \pi \bullet t);\ \ z2 = t$

In the first case, we will plot the curve given by spatial coordinates with the **plot3** command. To do this, we must first calculate the coordinates of the points of the spiral taken at a given density. We want to draw the two spirals next to each other in a figure. Take the value of the parameter t between 0 and 4, with 0.01 intervals.

```
% definition with 3D coordinates
clc; clear; close all;
t = 0:0.01:4;
x1 = cos(2*pi*t);
y1 = sin(2*pi*t);
z1 = t;
figure(); subplot(1,2,1); plot3(x1,y1,z1,'.')
```

Of course, you can also display a spatial curve with parametric functions, using the **fplot3** command.

```
% definition with parametric functions
x2 = @(t) t.*cos(2*pi*t);
y2 = @(t) t.*sin(2*pi*t);
z2 = @(t) t;
subplot(1,2,2)
fplot3(x2,y2,z2,[0,4])
```



Parametric display of a bivariate function

It may also be necessary to display parametric surfaces, e.g. when displaying a sphere. Show a sphere with center (3,2,1) and radius R = 5. In this case, 2 parameters will be required, which means two angles, such as specifying latitude and longitude.

```
%% Parametric 3D surface
x0 = 3; y0 = 2; z0 = 1; R = 5;
x = @(t,s) x0 + R*sin(t).*cos(s)
```

```
x = function_handle with value:
    @(t,s)x0+R*sin(t).*cos(s)
```

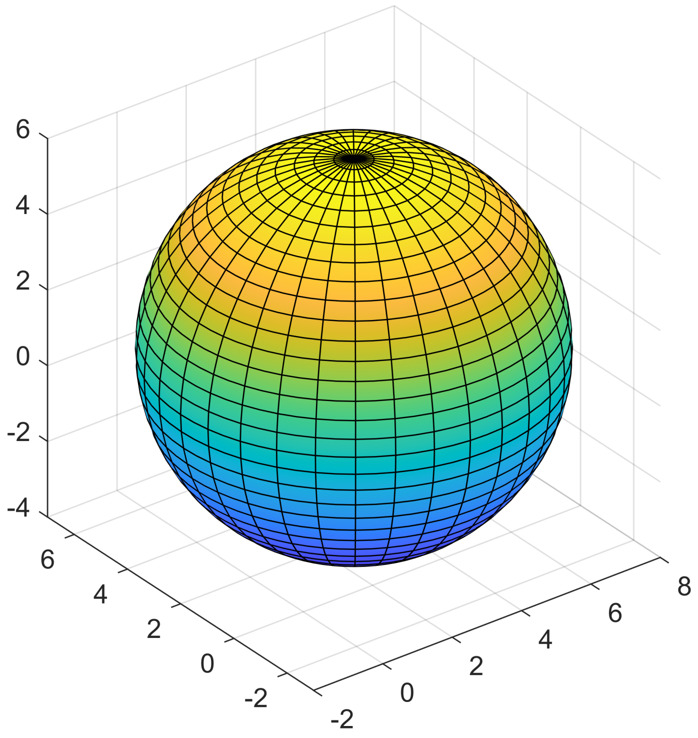
```
y = @(t,s) y0 + R*sin(t).*sin(s)
```

```
y = function_handle with value:
    @(t,s)y0+R*sin(t).*sin(s)
```

```
z = @(t,s) z0 + R*cos(t)
```

```
z = function_handle with value:
    @(t,s)z0+R*cos(t)
```

```
figure();
fsurf(x,y,z,[0,pi,-pi,pi])
axis equal
```



3D Surface Visualization

Let's visualize the following function:

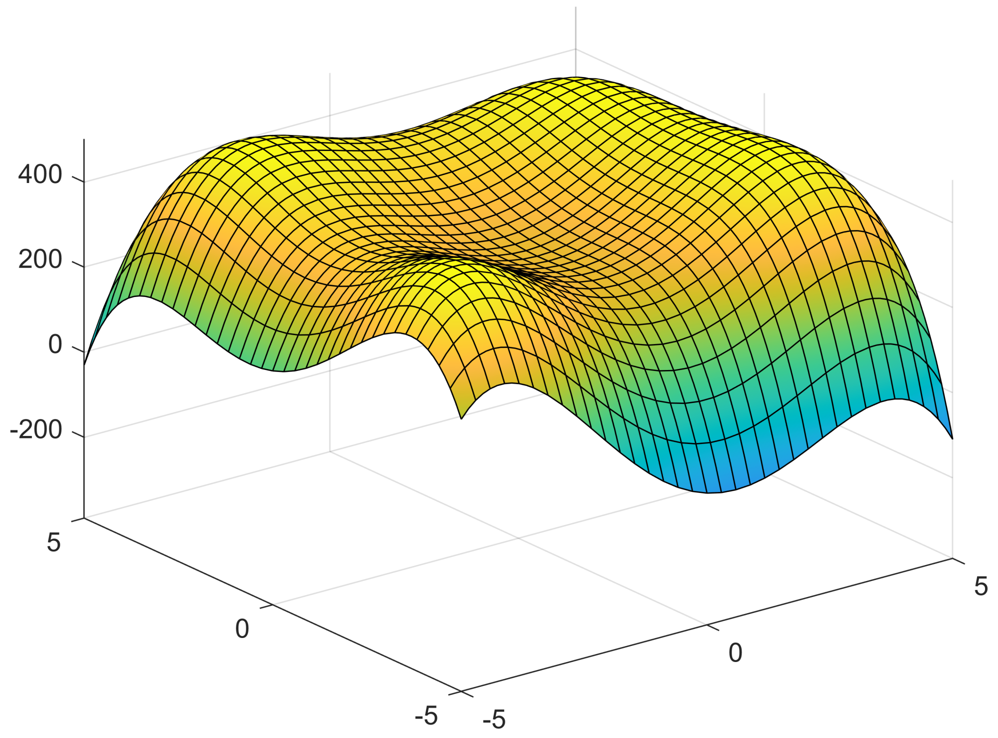
$$f(x,y) = 500 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2$$

This is a bivariate expression; the easieast way is to represent it as a surface. Use **fsurf** or **ezsurf** for this:

```
clc; close all; clear all;
f = @(x,y) 500 - (x.^2+y-11).^2 - (x+y.^2-7).^2
```

```
f = function_handle with value:
    @(x,y)500-(x.^2+y-11).^2-(x+y.^2-7).^2
```

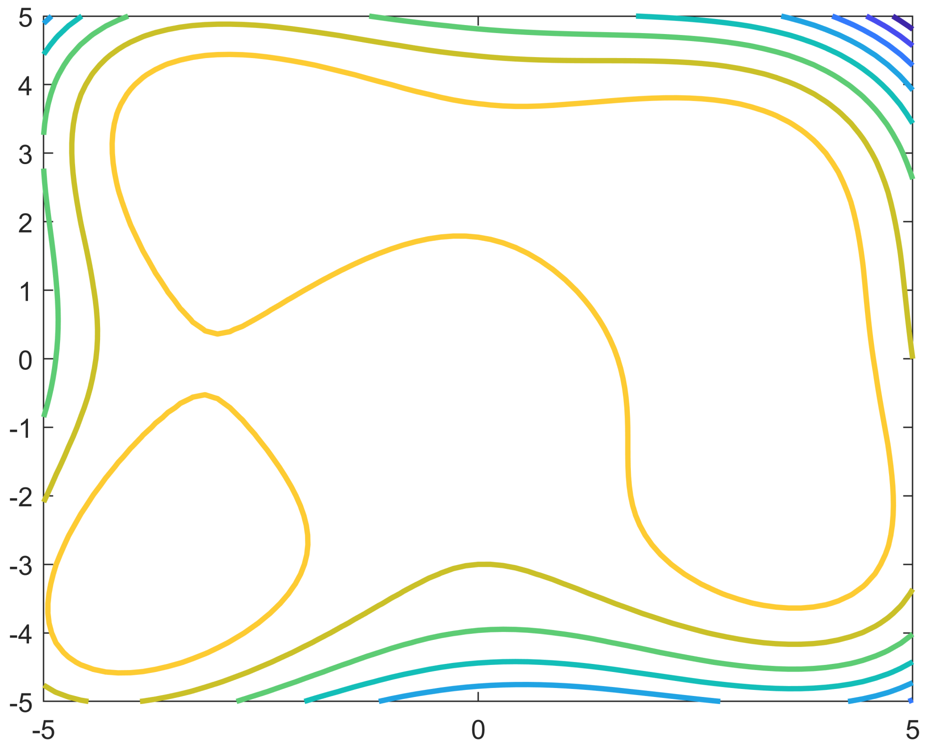
```
figure();
fsurf(f,[-5,5,-5,5]) % OR ezsurf(f,[-5,5,-5,5])
```



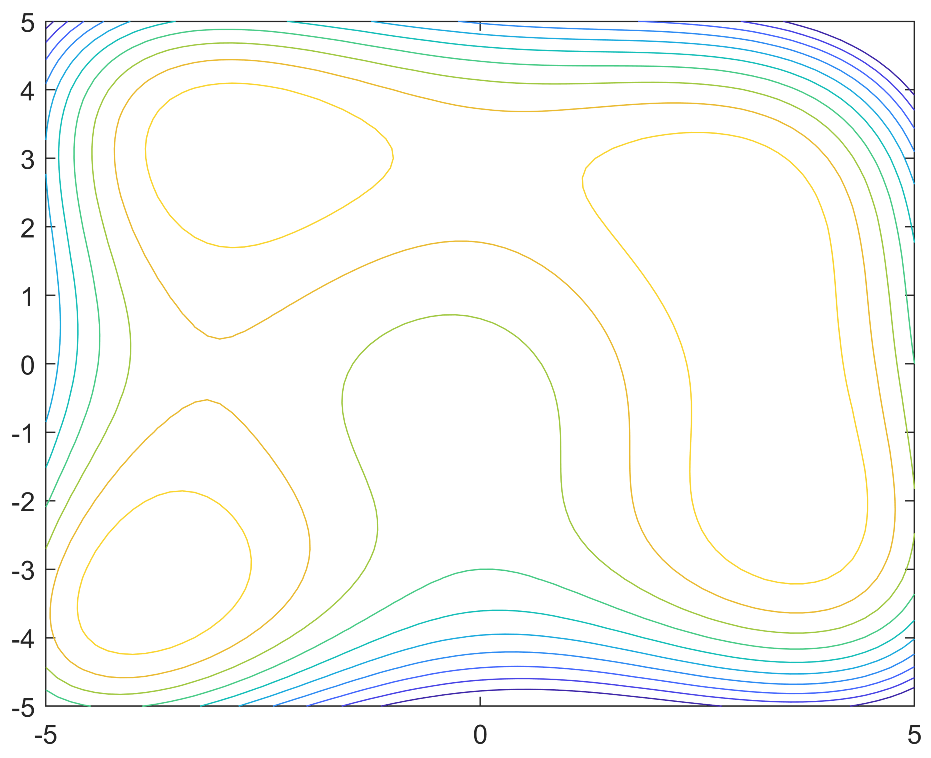
```
fsurf(f,[-5,5,-5,5],'ShowContours','on')
```

In many cases, a purely contouring representation of a bivariate function is much more illustrative. We can also represent in a contour map form using **fcontour** (or **ezcontour**):

```
figure();
fcontour(f,'LineWidth',2) % vagy ezcontour(f)
```



```
fcontour(f, [-5,5,-5,5],'LevelList',0:50:600)
% contour line labels can only be displayed with ezcontour
h = ezcontour(f,[-5,5,-5,5])
```



```
h = 
  Contour with properties:

     LineColor: 'flat'
     LineStyle: '-'
     LineWidth: 0.5
          Fill: off
     LevelList: [-300 -200 -100 0 100 200 300 400]
         XData: [60×60 double]
         YData: [60×60 double]
         ZData: [60×60 double]

  Show all properties
```

```
set(h,'ShowText','on','LevelList',0:50:600)
```

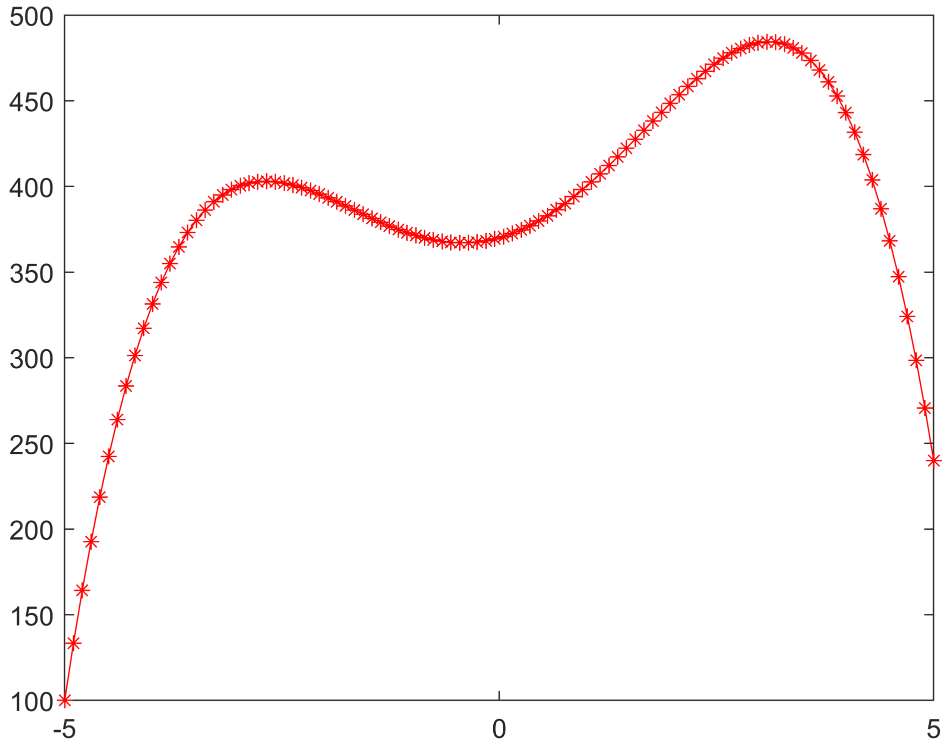
Display of sections, surface lines

It is often necessary to take sections on a surface and display them. This is usually done in the simplest way by taking many points between the two endpoints of the section and calculating the height values at these points. Consider a S-N section on the previous surface at x = -2.

```
% Sections, S-N section at x = -2
ym1 = linspace(-5,5,100);
xm1 = -2*ones(size(ym1));
zm1 = f(xm1,ym1); % calculation of heights
figure(1); hold on;
plot3(xm1,ym1,zm1,'r*','LineWidth',2) % the plot
```

Show the section in the y-z plane in a separate figure.

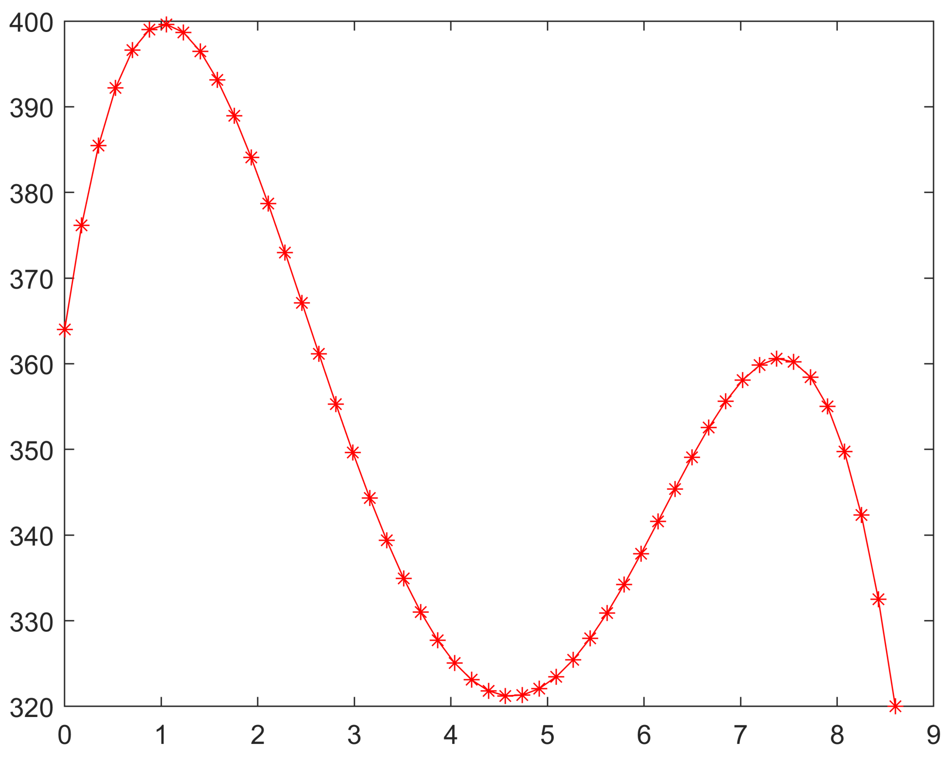
```
figure(5); plot(ym1,zm1,'r*-')
```



Similarly, you can add an intersection between two arbitrary points. Let now be the two points: A (-4.3); B (1, -4). Draw the section between the two points in the spatial figure and in a separate 2D figure as a function of the horizontal distances for the heights.

```
% An arbitrary directional section: A(-4,1); B(3,-4)
xm2 = linspace(-4,3,50);
ym2 = linspace(1,-4,50);
zm2 = f(xm2,ym2); % calculation of heights
figure(1); plot3(xm2,ym2,zm2,'k*','LineWidth',2) % the plot

% heights as a function of horizontal distances
dx = diff(xm2); dy = diff(ym2); dt = sqrt(dx.^2+dy.^2);
tav = [0, cumsum(dt)]; % distances from the starting point
figure(6); plot(tav,zm2,'r*-')
```



We can also draw the boundary line of a planned building in the figure. Let this building now be a 4x3 meter rectangular building with the lower left corner point (-1, -1). Draw the boundary line of the building in both the contour diagram and the 3D diagram. You can easily draw it in the contour diagram with a command called **rectangle**, where you need to specify the coordinates and dimensions of the lower left corner point.

```
% Placement of a rectangular building
figure(2); hold on; rectangle('Position',[-1,-1,4,3])
```

Drawing in the 3D figure is already a bit more difficult, as heights would have to be calculated along the boundary line of the building. This would require condensing the points along the closed shape. This can be done by fitting a linear spline, that is, a broken line curve, to the points, as in the case of multivalued curves.

```
% fitting a linear spline to the corner points
% entering corner points from starting point to starting point - closed curve!
x = [-1,3,3,-1,-1]; y = [-1,-1,2,2,-1]; plot(x,y,'r*')
% parameter should be the distance from the starting point
dx = diff(x); dy = diff(y); dt = sqrt(dx.^2+dy.^2);
tav = [0, cumsum(dt)] % distances from the starting point: 0 4 7 11 14
```

```
tav = 1×5
     0     4     7    11    14
```

```
max(tav) % 14
```

```
ans =
    14
```

```
xsp = @(t) interp1(tav,x,t)
xsp = function_handle with value:
    @(t)interp1(tav,x,t)
```

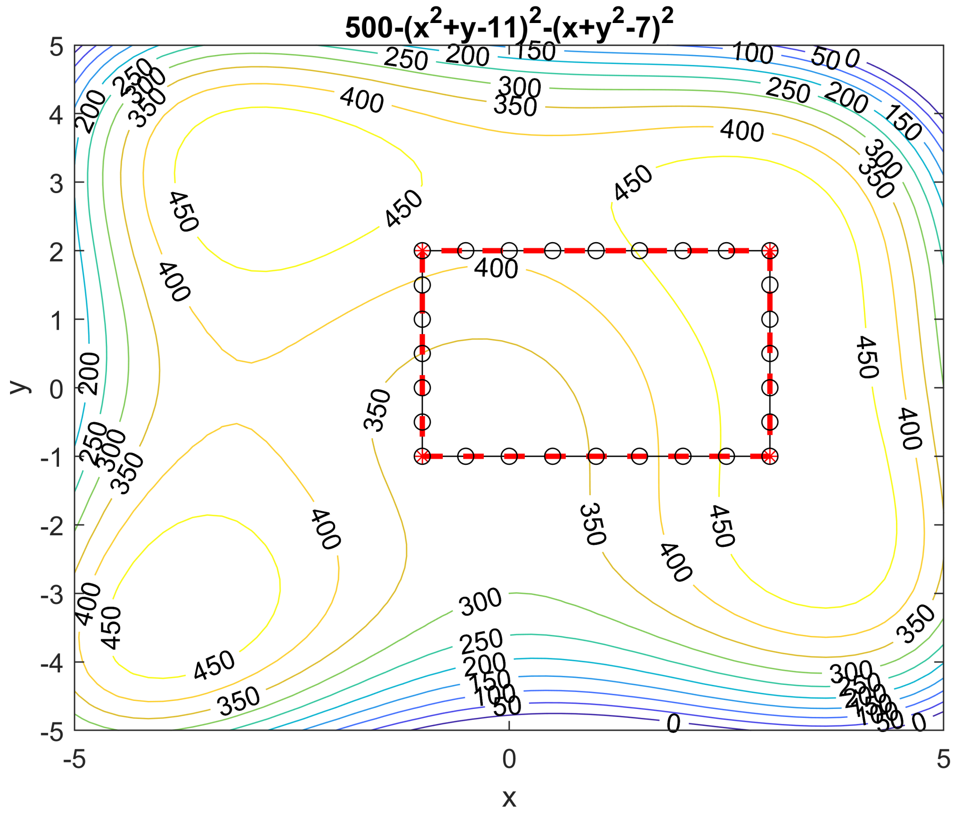
```
ysp = @(t) interp1(tav,y,t)
```

```
ysp = function_handle with value:
    @(t)interp1(tav,y,t)
```

```
fplot(xsp,ysp,[0,max(tav)],'r--','LineWidth',2)
```

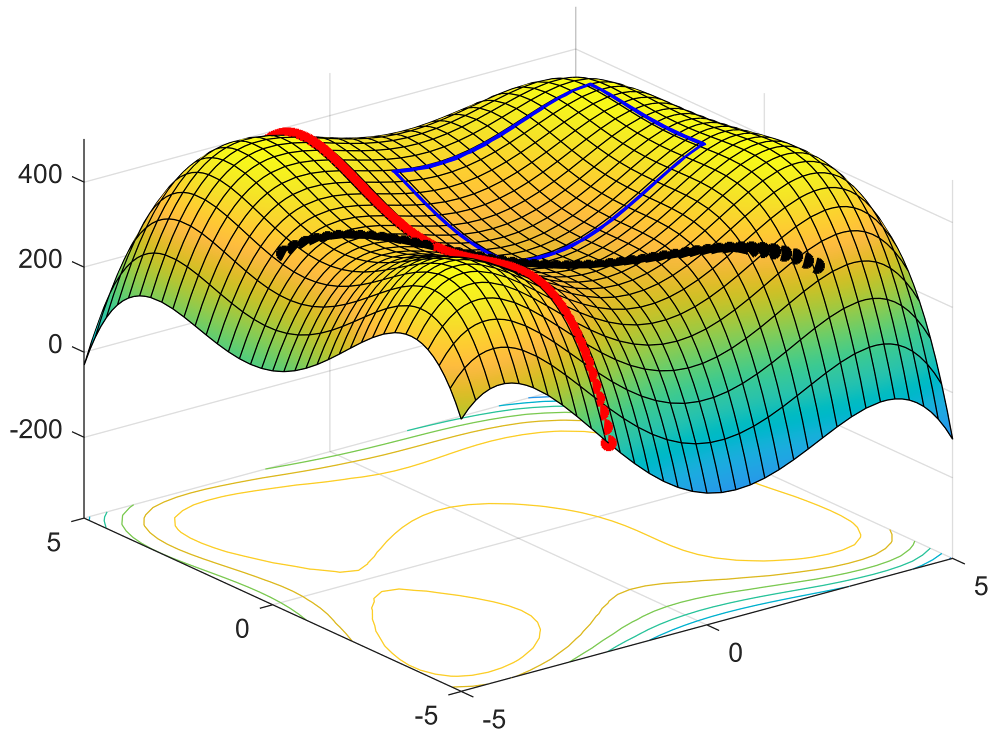
Using the fitted linear spline functions, we can add points along the side of the building and then calculate the heights at these points as well. Now add points every half meter around the perimeter of the building!

```
% Adding points every half meter along the perimeter of the building
ti = 0:0.5:max(tav);
xi = xsp(ti);
yi = ysp(ti);
plot(xi,yi,'ko')
```



Calculate the heights of the points and draw the outline of the building in the spatial diagram.

```
% Spatial coordinates
zi = f(xi,yi);
figure(1); plot3(xi,yi,zi,'b','LineWidth',2)
```



Display points given in a regular grid

It is often necessary to represent a surveyed terrain. The surveyed points may be located at points on a regular grid or irregularly scattered. Usually, the latter is the typical case. Scattered points are typically fitted with an interpolation or regression method and displayed as a function or interpolated to the grid's corner points, and this grid is displayed. Multidimensional interpolation and regression will only be discussed later, so now we will only talk about the display of points in the regular grid.

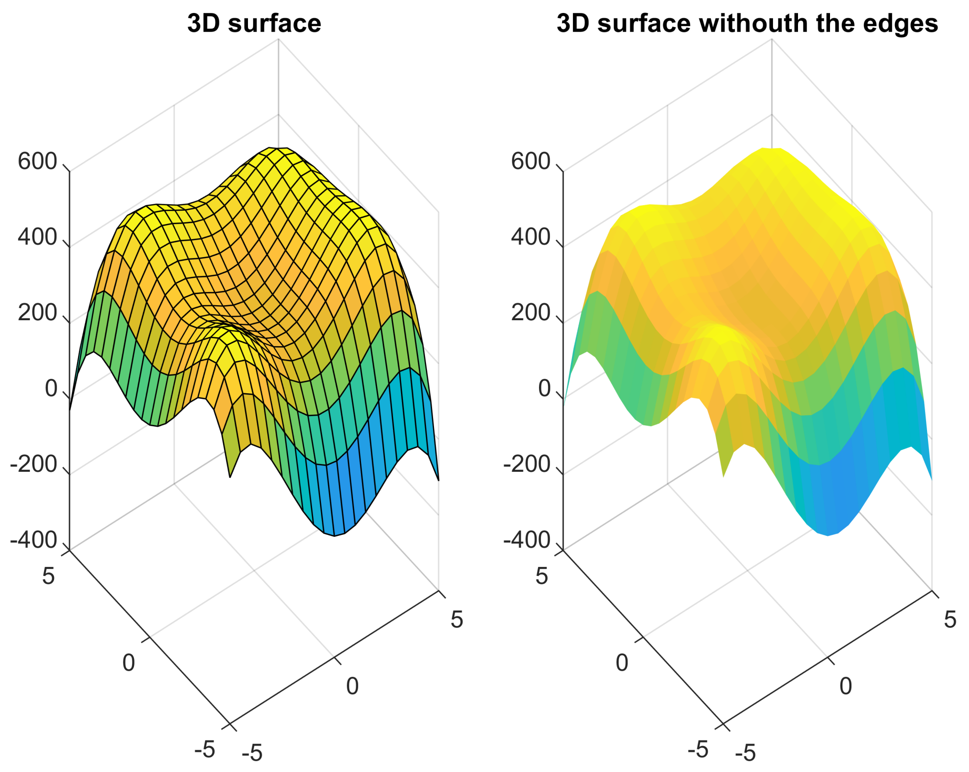
Let us return to our previous function f(x, y) and calculate the function's values at the points of a regular grid. Thus we will get three matrices for the X, Y, and Z coordinates. First, create a vector in the x and y directions, as dense as we want to add the grid, and then use the **meshgrid** command to generate the X, Y matrices of the grid. The Z values will be calculated using the function f(x, y). The grid points can be displayed with the **plot3** command and the grid itself with the **mesh** command.

```
close all
x = -5:0.5:5; y = -5:0.5:5;
[X,Y] = meshgrid(x,y);
Z = f(X,Y)
```

```
Z = 21×21
       250      299.69         304      275.69         226      164.69 ⋯
    341.69      400.88      414.19      394.38      352.69      298.88
       384      452.19         474      462.19         428      381.19
    389.69      466.38      496.19      491.88      464.69      424.38
       370      454.69         492      494.69         474      439.69
    334.69      426.88      471.19      480.38      465.69      436.88
       292      391.19         442      457.19         448      424.19
```

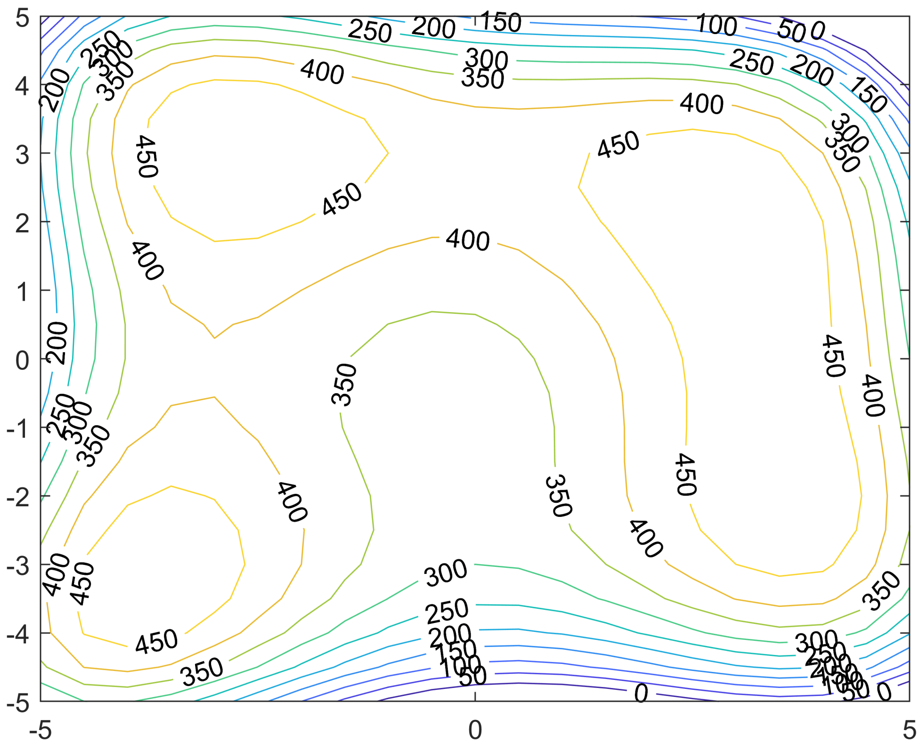
```
      248.69        354.38        411.19        431.88        427.69        408.38
         210        321.69           384        409.69           410        394.69
      179.69        296.88        364.19        394.38        398.69        386.88
    ⋮
```

```
figure();subplot(1,2,1);
plot3(X,Y,Z,'r*')
title('3D points')
subplot(1,2,2);
mesh(X,Y,Z)
title('3D meshgrid')
figure(); subplot(1,2,1)
surf(X,Y,Z)
title('3D surface')
subplot(1,2,2); surf(X,Y,Z,'EdgeColor','none')
title('3D surface withouth the edges')
```



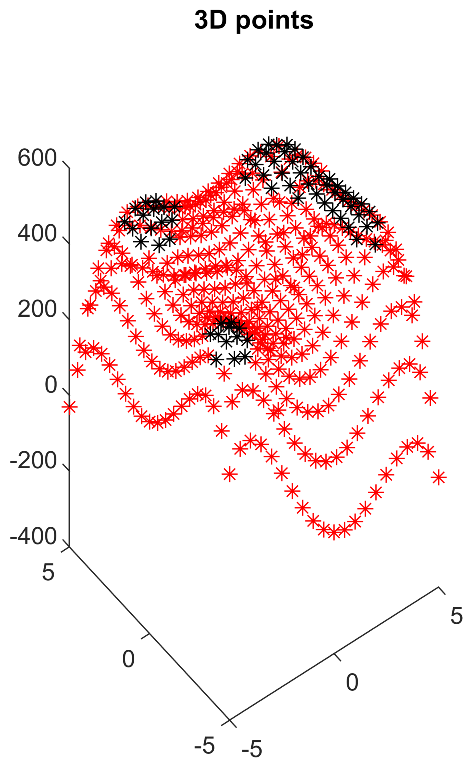
To represent the contour lines on a discrete dataset, use the **contour** function.

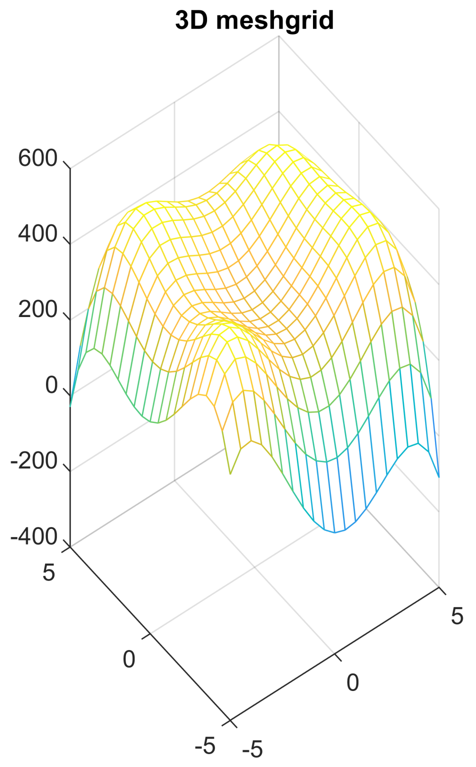
```
figure()
contour(X,Y,Z,'ShowText','on','LevelList',0:50:600)
```



It may also be necessary to work with spatial points that meet certain conditions, e.g. above / below a given height. This can also be done here with logical indexing, which we've also used for univariate regression. Select all the points above 460 m!

```
% delecting points above 460 meters
felt = Z>460;
xh = X(felt); yh = Y(felt); zh = Z(felt);
figure(1); subplot(1,2,1); hold on; plot3(xh,yh,zh,'k*')
```





Plotting Finite Element Mesh

In mechanics, a finite element method is often used to determine the stress state and deformation of a statically loaded element of complex shape. In this case, the body to be modeled is divided into simpler geometric elements:

- in the case of a plane, for example, triangles or squares;

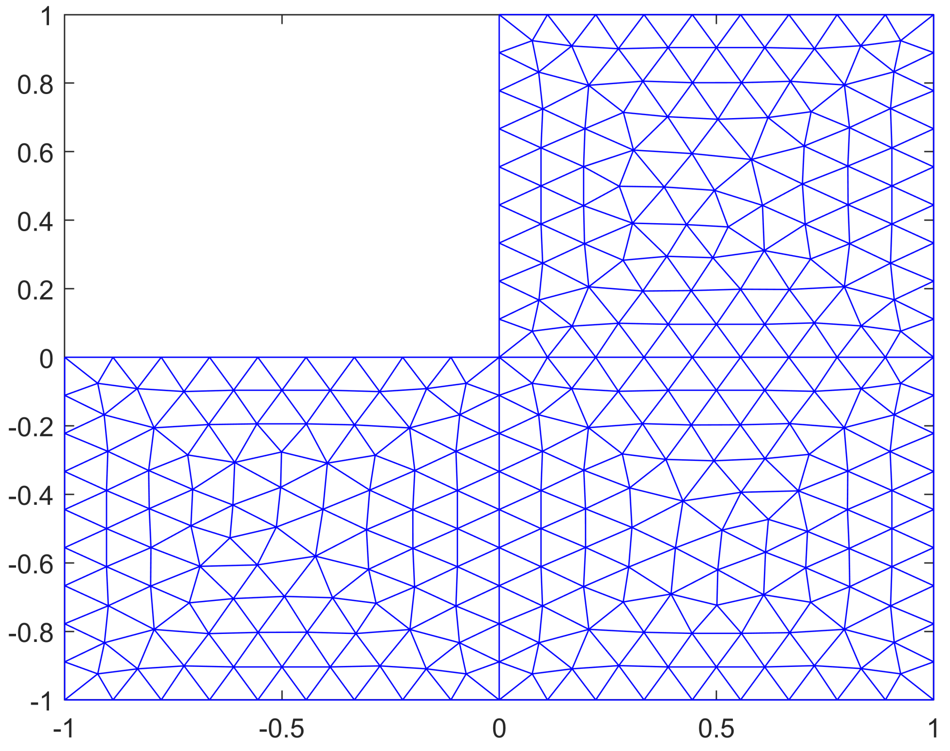
- in the case of space, columns, or tetrahedra.

In the model, the elements are connected only at their nodes. Based on the nodes, the elements' stiffness matrix can be written, and then the displacement of the nodes and then the approximate stresses can be calculated by solving a system of linear equations. Here we do not deal with the problem's solution, only with the display of a ready-made plane grid and the results.

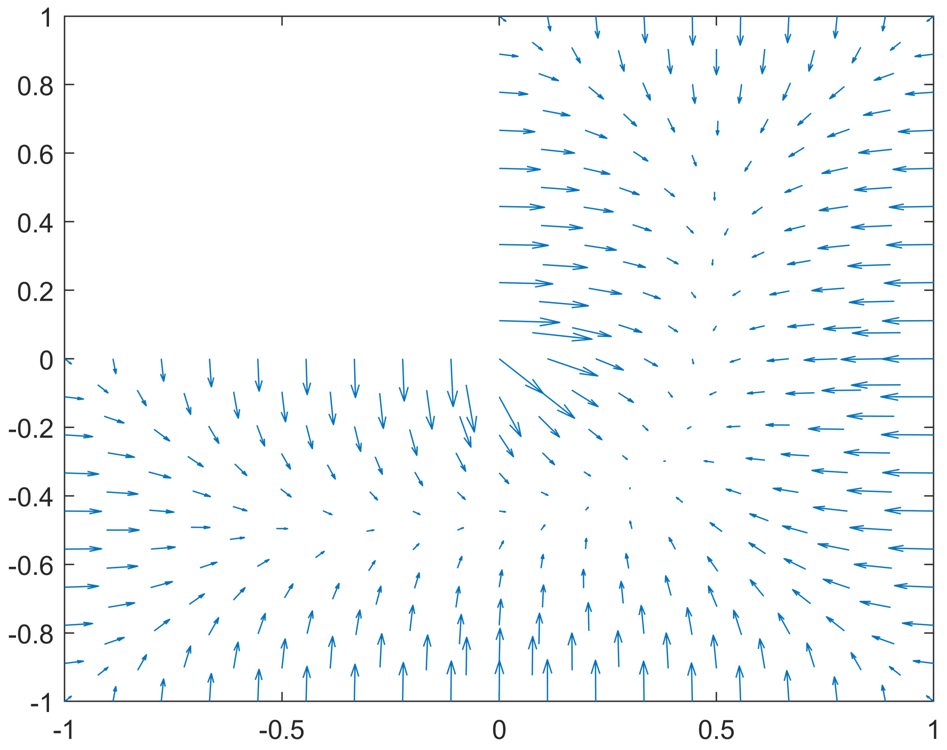
Load the **fem_triangles.mat** file into Matlab! There are six matrix variables saved in this file. The ***nx*** and ***ny*** column vectors contain the coordinates of the nodes. Here the planar element is divided into triangles; the matrix ***T*** consists of three columns with the sequence number of nodes belonging to triangles. The vector ***u*** contains the magnitude of the resulting node stresses, and the vectors ***ux***, ***uy*** contain the gradient of the solution.

Plot the triangular mesh (**triplot** command) and the gradient vectors (**quiver** command) in a plane.

```
clc; clear all; close all;
load fem_triangles.mat
% T - triangle node numbers
% nx,ny - x,y coordinates of the nodes
% u - solution at nodes
% ux,uy - solution gradients
figure(1); triplot(T,nx,ny) % plot the triangular mesh
```

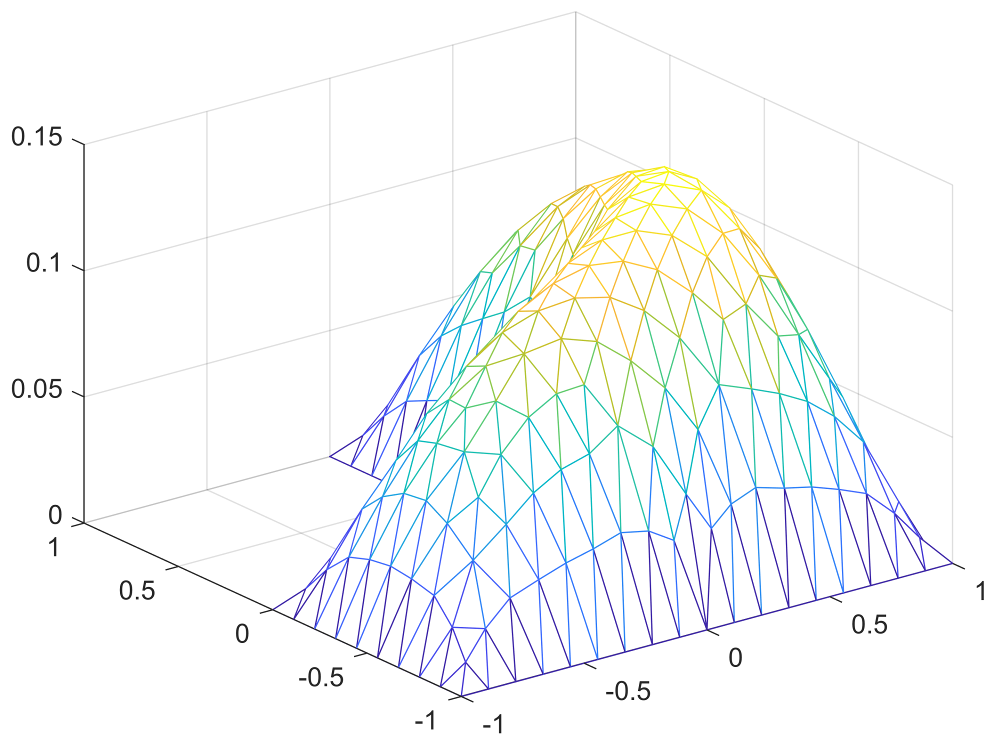


```
figure(2); quiver(nx,ny,ux,uy) % display gradient with vectors
```

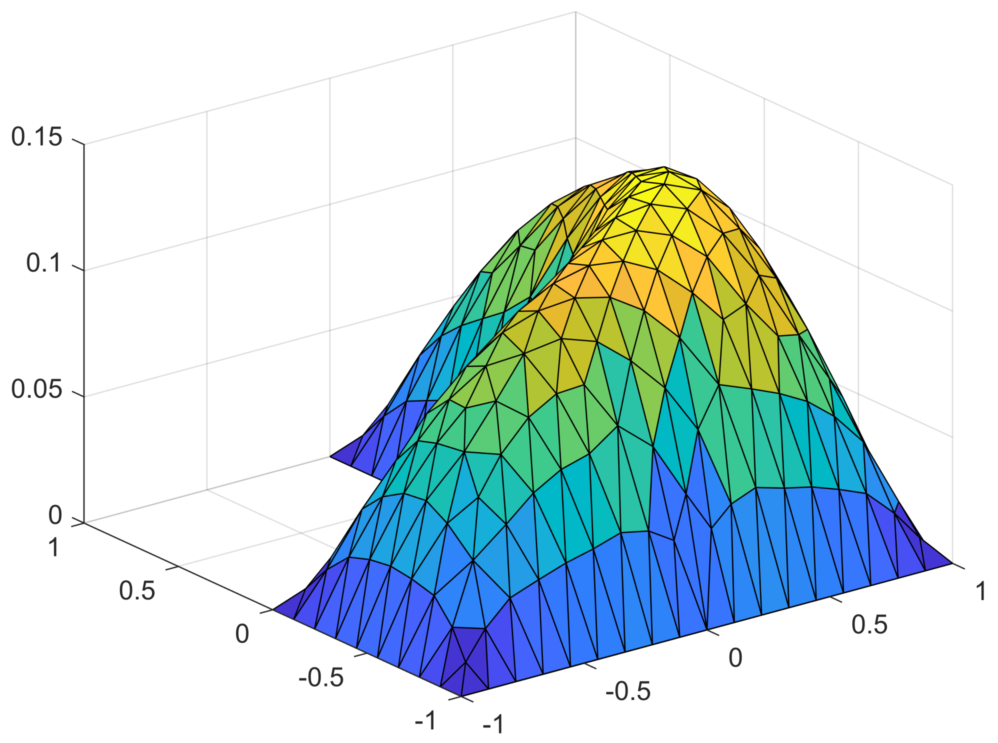


We can also represent solutions in space, either with a spatial triangular mesh (**trimesh** command) or as a surface (**trisurf** command).

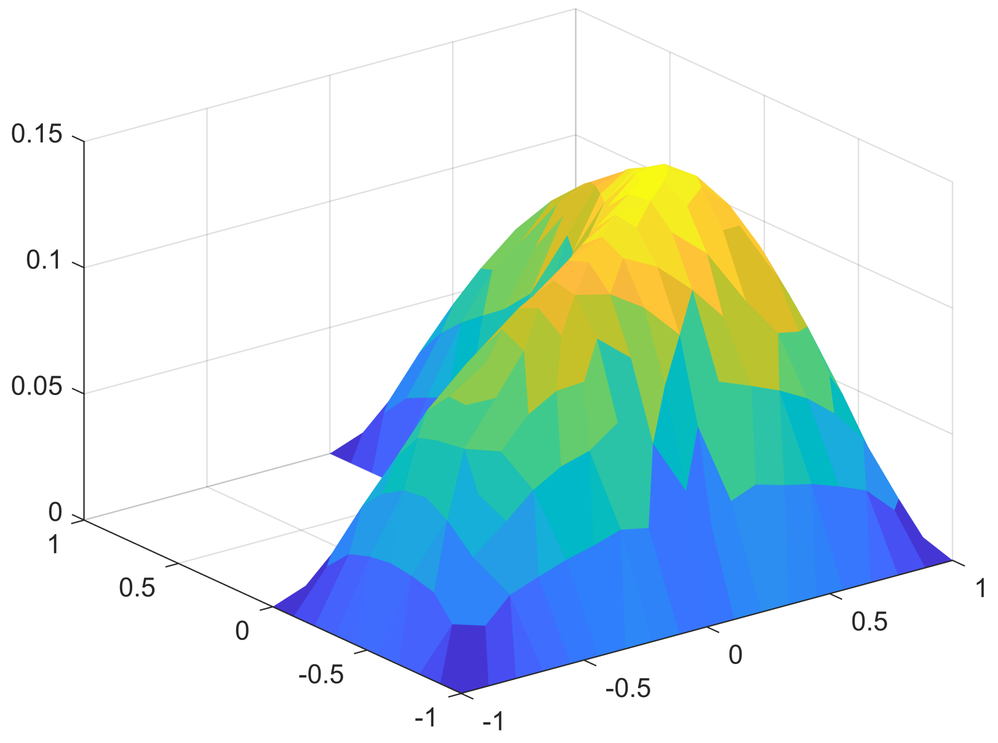
```
figure(3); trimesh(T,nx,ny,u) % spatial triangular mesh
```



```
figure(4); trisurf(T,nx,ny,u) % spatial surface from triangles
```



```
figure(5); trisurf(T,nx,ny,u,'EdgeColor','none')
```



Display of spatial flow processes

In civil engineering practice, it may be necessary to display not only 2D but also 3D vector fields, for example when modeling the flow of rivers around structures. Let us now look at such an example of what possibilities we have in this case. An example is the flow pattern of one of the waders of the Danube above the Mosoni-Danube with two spurs.

The data can be found in two files. The ‘aramlat.mat’ Matlab data file stores flow velocity vectors (5 meters x 5 meters x 0.25 m resolution) in a regular rectangular matrix of 217x195x38. The riverbed is irregular, so there are cells that are out of range. In these, the speed is 0,0,0. ‘Aramlat.txt’ is a simplified text version of the previous file that contains only non-zero speed elements. In the six columns, the coordinates (x, y, z) and the components of the velocity vectors (ux, uy, uz).

First, load the text file, which contains significantly less data, so you can display 3D velocity vectors faster.

```
% 3D vectors
clc; clear all; close all;
adat = load('aramlat.txt');
x1 = adat(:,1);y1 = adat(:,2);z1 = adat(:,3);
ux1 = adat(:,4);uy1 = adat(:,5);uz1 = adat(:,6);
figure(1)
quiver3(x1,y1,z1,ux1,uy1,uz1) % not visible enough, too dense
daspect([25 25 1]) % it is advisable to adjust the axle ratios
```

Unfortunately, with so much data, it's not very clear, it's too dense, and the vectors aren't visible, only if we zoom in.

We can better visualize flows by drawing streamlines in the figure. However, to display the flow lines, you can only use data entered in a regular rectangular grid, so let's now read the data file 'aramlat.mat', which contains 6 variables, each given as a 3D grid. Streamlines can be displayed with the **streamline** command. You need to specify the coordinates, velocity vectors, and the point from which you want to start the flow line. You can change its properties (eg color, thickness) with the **set** command.

```
% plotting streamlines
load aramlat.mat
streamline(x,y,z,ux,uy,uz,200,1700,105)
h = streamline(x,y,z,ux,uy,uz,480,1400,104.5)
```

```
h = 
  Line with properties:

              Color: [0 0 1]
          LineStyle: '-'
          LineWidth: 0.5
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [1×963 double]
              YData: [1×963 double]
              ZData: [1×963 double]

  Show all properties
```

```
set(h,'LineWidth',2)
```

We can see the dynamics of the flow better by taking a starting section and specifying several starting points there in a grid. Now let this be the intersection at x = 200 m in a 5x7 matrix (every 30 meters in the y direction and every 1 meter in the z direction).

```
% Draw multiple streamlines together
x0 = 200
```

```
x0 = 
   200
```

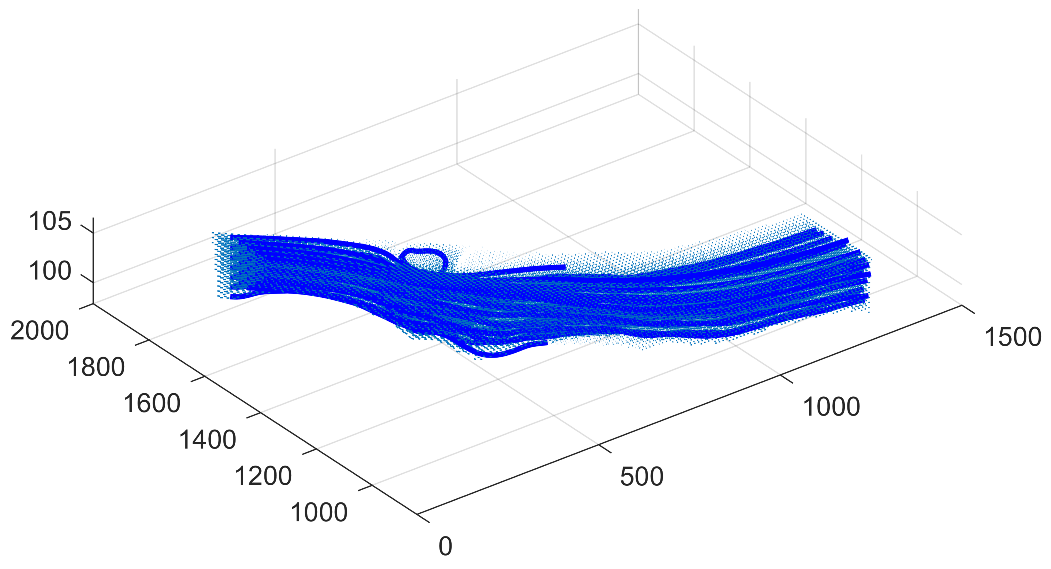
```
y0 = 1650:30:1800 % 5 value
```

```
y0 = 1×6
        1650        1680        1710        1740        1770        1800
```

```
z0 = 100:1:106 % 7 value
```

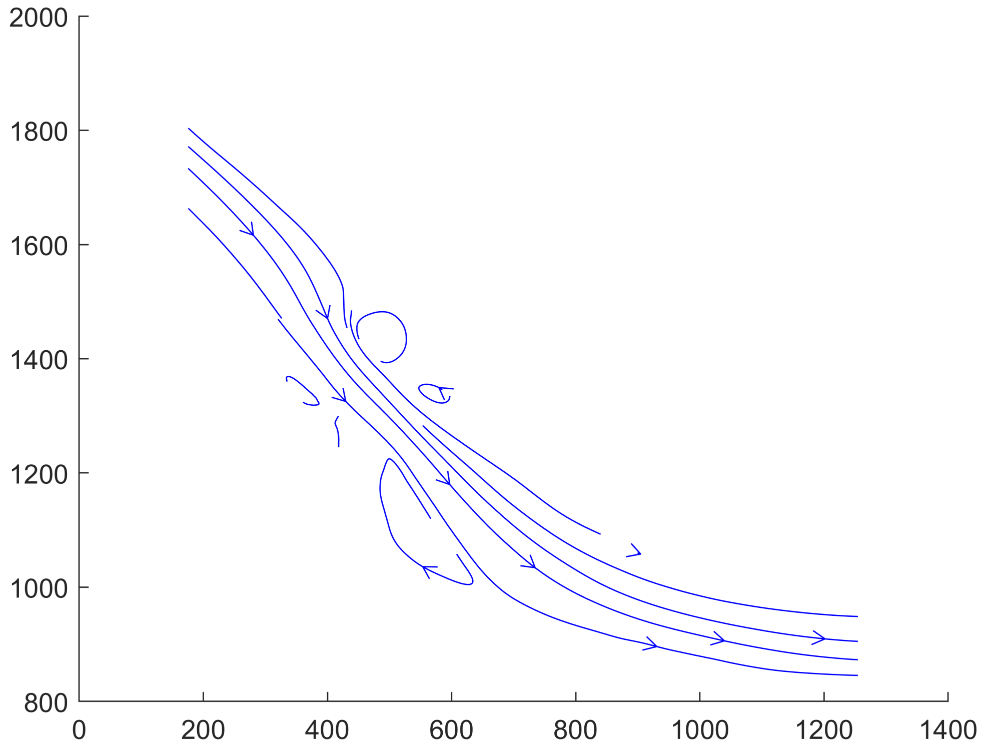
```
z0 = 1×7
   100   101   102   103   104   105   106
```

```
[XV, YV, ZV] = meshgrid(x0,y0,z0);
H = streamline(x,y,z,ux,uy,uz,XV,YV,ZV);
set(H,'LineWidth',2)
```



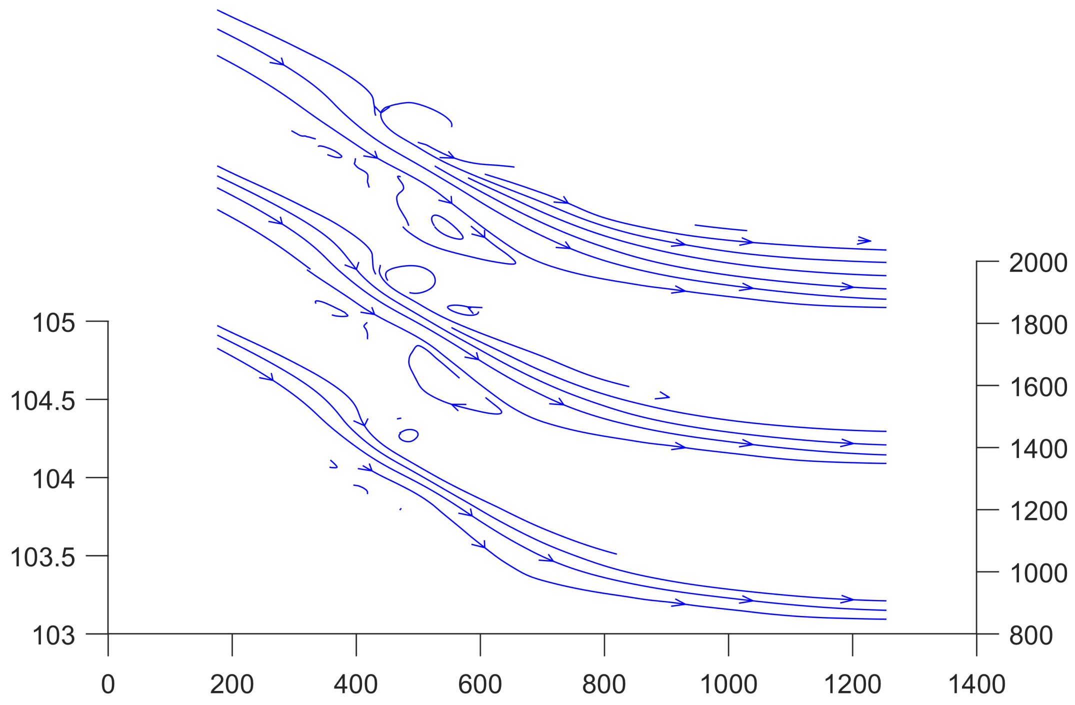
Another illustrative display method is to use the **streamslice** command to create sections. Make a section at a height of 104 meters! The location of the section must be set (of course, not only the z-direction section is available).

```
% Making streamline sections
figure(2)
streamslice(x,y,z,ux,uy,uz,[],[],104);
```



You can specify more than one value in a vector:

```
figure(3)
streamslice(x,y,z,ux,uy,uz,[],[],103:105);
view(0,50)
```


105
104.5
104
103.5
103
0
200
400
600
800
1000
1200
1400
2000
1800
1600
1400
1200
1000
800