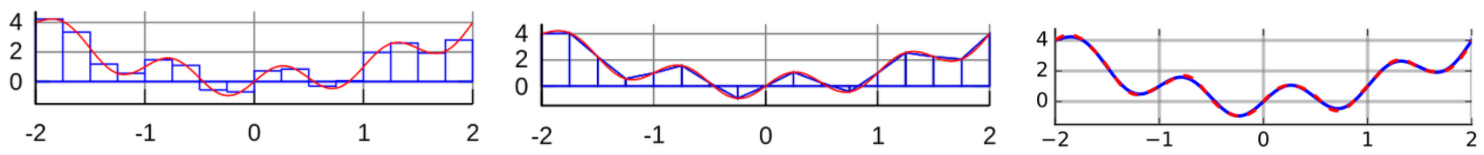
Numerical integration

Numerical integration means an approximation of the integral, which gives is used to determine the definite integrals value. There is a wide range of applications, including the calculation of curve length ($L = \int_a^b \sqrt{1}\, dx$), area, volume or solving differential equations.



Numerical integration using trapezoidal rule

The most known method is the application of the trapezoidal rule, where the neighbouring points are connected with a line, and the trapezoid under this line approximates the integral in that region. We apply this both in case of discrete data points or analytical functions, when the integral function is not known or couldn't be determined. In the latter case we introduce *n* number of intermediate points along the region we want to integrate (meaning *n-1* number of segments) and apply the trapezoidal rule on the mentioned points.

In case of a single range:

$$\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2}\left(b-a\right)$$

In case of multiple intervals the area of the trapezoids must be summed up. For $N = n - 1$ segments:

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{i=1}^{N}(f(x_i)+f(x_{i+1}))(x_{i+1}-x_i)$$

In the upper formula it is not necessary to let the segments be equally long, but in that case the expression could be simplified:

if $(x_2 - x_1)=(x_3 - x_2)= \cdots =(x_n - x_{n-1})= h$, then: $\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{N}(f(x_i)+f(x_{i+1}))$

Lets see how this works in an example!

Earth's density (ρ) changes in terms of the radius (R) as we saw earlier, just load **earth_density.txt** file! Determine the mass of the Earth using the following integral formula:

$$M_{\text{Earth}} = \int_0^{6370} \rho 4\pi R^2 dr$$

For the density data, load **earth_density.txt** file!

```
% Mass of the Earth
clc; close all; clear all;
data = load('earth_density.txt');
R = data(:,1)*1000; % km to meters
ro = data(:,2);
```

```
figure(6); plot(R,ro,'m*-')
```

Lets calculate the function values for the relevant radius values!

```
fx = 4*pi*ro.*R.^2;
```

To solve this, lets use the built-in Matlab function **trapz**, which determines the definite integral based on discrete data points using the trapezoidal rule. It is not necessary for the points to be equally distanced.

```
        M = trapz(R,fx) %  6.0261e+24 kg
```

```
M =
    6.026095773514430e+24
```

This means the mass of the Earth is $6.0261 \bullet 10^{24}$ kg. This is quite a good approximation for the curently accepted mass of the Earth $(5.9722 \pm 0.0006) \bullet 10^{24}$ kg.

Numerical integral using the Simpson rule

The trapezoidal rule approximates the function between the neighbouring points with a line. A more accurate result could be achieved using a higher order approximating function. The most known of these method is using the simpson formula, which applies a second or third order polynomial to approximate the function in the relevant range (Simpson's 1/3 method, Simpson's 3/8 method). In case of the second order Simpson-formula a parabola is fitted for three neighbouring points. This can be applied using the Newton's form of the interpolating polynomials for three points:

$$p(x) = a_1 + a_2(x - x_1) + a_3(x - x_1)(x - x_2)$$

Where the coefficients are the following:

$$a_1 = y_1; a_2 = \frac{y_2 - y_1}{x_2 - x_1};\ a_3 = \frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1}$$

In case of equal intervals (h):

$$a_1 = y_1; a_2 = \frac{y_2 - y_1}{h};\ a_3 = \frac{y_3 - 2\,y_2 + y_1}{2\,h^2}$$

Substituting the coefficients into the polynomial results in the following formula for the three points:

$$\int_{x_1}^{x_3} f(x)dx \approx \int_{x_1}^{x_3} p(x)dx = \frac{h}{3}\left(f(x_1) + 4\,f(x_2) + f(x_3)\right)$$

The general form:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \frac{h}{3}\left(f(x_{i-1}) + 4\,f(x_i) + f(x_{i+1})\right)$$

Let the *n* number of points be equally distanced by *h* in the given range [a,b]: $x_1 = a, \quad x_n = b$. The *n* number of points cuts the interval into $N = n - 1$ number of segments. For applying the Simpson rule we need 3 points to fit a parabola; we can calculate the integral for two consecutive segments, therefor we should divide the range always into even number of segments:

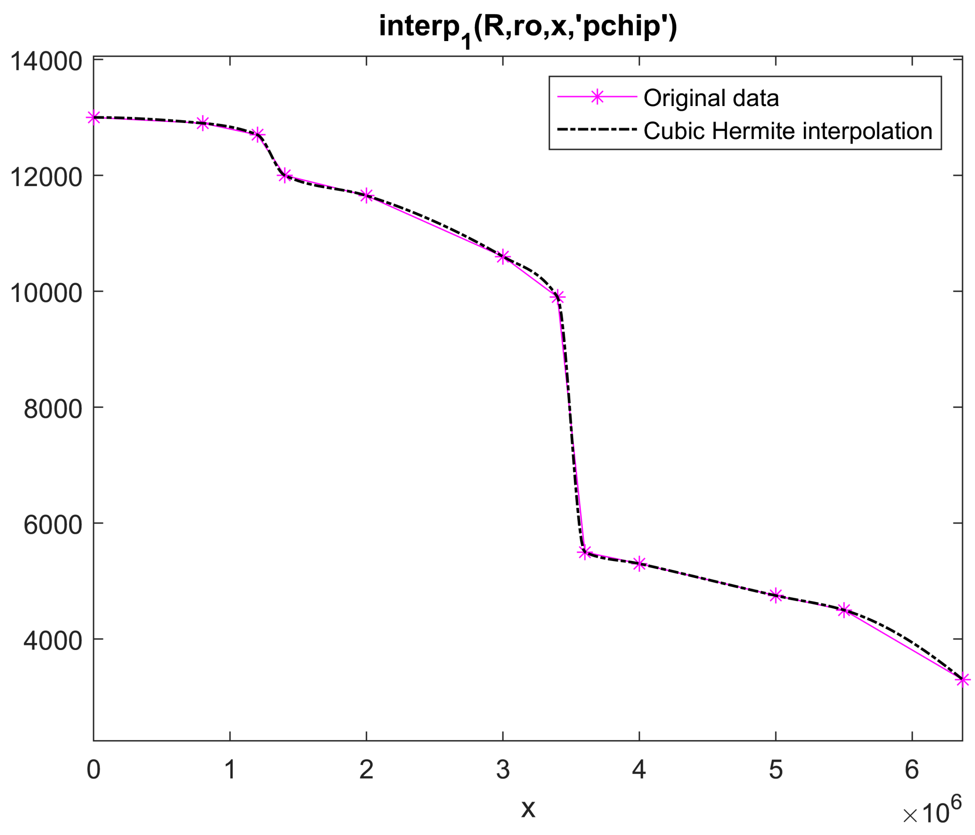
$$\int_a^b f(x)dx = \int_{x_1=a}^{x_3} f(x)dx + \int_{x_3}^{x_5} f(x)dx + \cdots \int_{x_{n-1}=x_N}^{x_n=b} f(x)dx = \sum_{i=2,4,6\ldots}^{N} \int_{x_{i-1}}^{x_{i+1}} f(x)dx$$

The formula for the three points applied on all of the points results in the followings:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4\sum_{i=2,4,6\ldots}^{n-1} f(x_i) + 2\sum_{j=3,5,7\ldots}^{n-2} f(x_j) + f\left(b\right)\right]$$

Lets calculate the mass of the Earth using the Simpson-rule! For this you can apply the built-in **quad** function of Matlab! For this function we should use a function as an input, not discrete data points. Lets define a spline curve on the density data points for this! We saw earlier that a cubic interpolation would be better then a second order spline method, due to sharp angles at the connections, so apply a cubic interpolation formula (**interp1** function '**cubic'** or **'pchip'** method).

```
% Cubic Hermite interpolation
         ro_cubic = @(x) interp1(R,ro,x,'pchip');
         figure(6); hold on;
         g = ezplot(ro_cubic, [0 6370000]);
         set(g,'Color', 'k','Linestyle','-.','LineWidth',1);
         legend('Original data','Cubic Hermite interpolation')
```



Lets calculate the integral using the **quad** function (Simpson-rule)! For this define the quantity you wish to integrate as a function of the radius using the previously fitted interpolating funtion!

```
        fx_cubic = @(R) 4*pi*ro_cubic(R).*R.^2;
        M2 = quad(fx_cubic,0,6370000) % 5.9658e+24 kg
```

```
Warning: Maximum function count exceeded; singularity likely.
M2 = 
     5.965754658784433e+24
```

The result for the mass of the Earth is $5.9658 \bullet 10^{24}$ kg. This is a better approximation than the value we got using the trapezoidal rule.

Remark: The **quad** function is advised to to replace with the **integral** function in newer versions of Matlab. This works better in more complex cases. It uses an adaptive kvadrature instead of the conventional Simpson-rule to calculate the integral.

```
        M3 = integral(fx_cubic,0,6370000) % 6.0541e+24
```

```
M3 = 
     6.054113130310109e+24
```

Calculating multidimensional integrals on regular grid

Calculating two and three dimensional integral is a common task, e.g. calculating area, or volume. A two-dimensional definite integral could be expressed in the following form:

$$I = \int_{y_1}^{y_2} \int_{x_1}^{x_2} f(x, y)\, dx\, dy$$

In this case the integral consists of two steps: an inner and an outer integral. First we can process the outer integral using a previously mentioned method (trapezoidal-, Simpson-rule), where each of the terms will contain an inner integral part, which could be calculated numerically. This way the single-variate numerical integral could be generalized for a multidimensional case, on a regular grid.

The **integral2** function could be used in Matlab to calculate the double-integral on a regular grid, and **integral3** to do the same in three-dimensional.

```
%           q = integral2(fun,xmin,xmax,ymin,ymax)
%           q = integral3(fun,xmin,xmax,ymin,ymax,zmin,zmax)
```

For the former we saw earlier an example in case of the topic of 2D interpolation, where we calculated the volume of a terrain given in a regular grid. But if the integration range is an arbitrary shape, a new aspect should be applied.

Calculating multidimensional integrals on an irregular range

In an irregular range we can apply for the integration the **Monte-Carlo method**. It is a stochastic algorithm, which is using random numbers. The conventional integration methods are usually evaluating the integrandus on a regular grid, though in this case the function evaluation is processed in random locations. This method could be presented well on area/volume calculation, but it can be generalized for further applications.

Determining area using the Monte-Carlo method

We determined the boundaries of a catchment area, lets calculate the area based on the boundary points! Load the **catchment.txt**, and plot it with equally scaled axis!

```
          clc; clear all; close all;
          data = load('catchment.txt');
          x = data(:,1); y = data(:,2);
          figure(7); hold on;
          plot(x,y,'r-','LineWidth',2)
          axis equal
```

If we want to determine the area using Monte-Carlo method, the main idea is to determine the bounding box of the area and generate in the specific range *N* number of random points with normal distribution. Then we count how many points are inside the region (*n*) and determine the ratio (*ρ*) for the inner points and the total number of points. If we generate enough points, the points approximate well the ratio of the inner and total area:

$$\rho = \frac{n}{N}$$

If we know the area of the bounding box: $T = a \bullet b$, then the area in the arbitrary shaped range (in this case the catchment area) could be calculated:

$$T_v = \int_T f(x)\, dT \approx \rho \bullet T = \frac{n}{N} \bullet (a \bullet b)$$

Lets solve the example in Matlab! First draw the bounding box around the shape!

```
a = max(x)-min(x)
```

```
a = 6669
```

```
b = max(y)-min(y)
```

```
b = 13169
```

```
rectangle('Position',[min(x),min(y),a,b])
```

How would the integral look like if we've a set of points in a regular grid? Let's test it out with 100 intermediate points!

```
h=500;
[X Y]=meshgrid(min(x):h:max(x), min(y):h:max(y));
% Representing the grid
figure(2)
plot(X,Y,'r+')
hold on
plot(x,y,'b')
```

We can use the **inpolygon** function in Matlab to determine which points are inside the polygon. It results in a logical vector, where the ones represent the points, that are inside the shape.

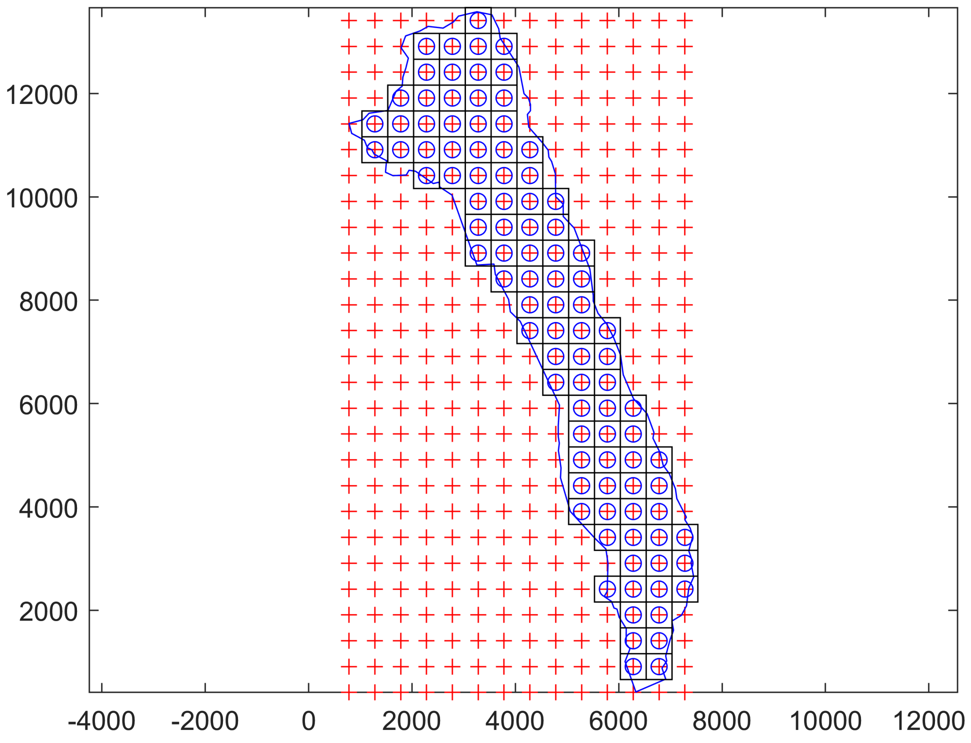
```
K=inpolygon(X,Y,x,y); % points inside are set to 1, outside to 0
plot(X(K),Y(K),'bo') % Selecting points inside the polygon
```

Plotting the relevant area which will be summerized as a rectangle

```
for i=1:size(K,1)
    for j=1:size(K,2)
        if K(i,j)==1
            rectangle('Position',[X(i,j)-h/2,Y(i,j)-h/2,h,h]);
        end
    end
end
Areg=h^2*nnz(K)
```

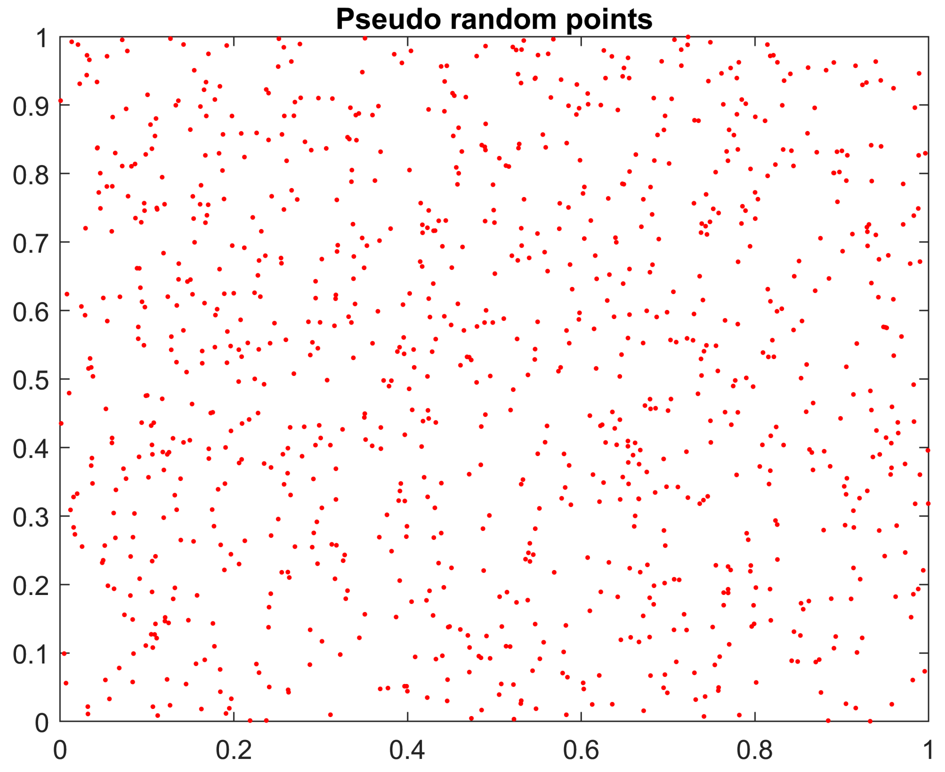
```
Areg = 24250000
```

```
axis equal
```

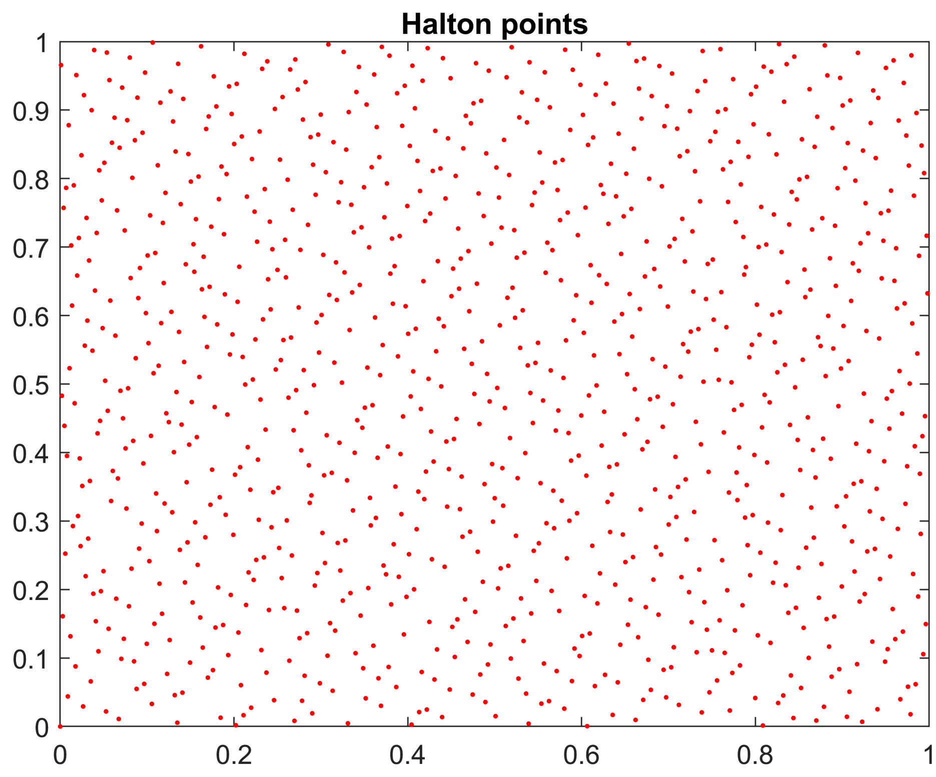


Lets generate random points in the relevant region! For this multiple functions could be used: one of them is the **rand** function, that generates pseudo-random numbers; another is using the Halton points (**haltonset**), that is based on the *van der Corput* series. Lets generate with these 1000 points! Both function is working in the range of [0,1].

```
% Generating pseudo random points
xyr = rand(1000,2);
figure(8);
plot(xyr(:,1),xyr(:,2),'r.')
title('Pseudo random points')
```



```
% Generating Halton points
hs = haltonset(2); % generating two-dimensional Halton series
xyh = net(hs,1000);
figure(9);
plot(xyh(:,1),xyh(:,2),'r.')
title('Halton points')
```



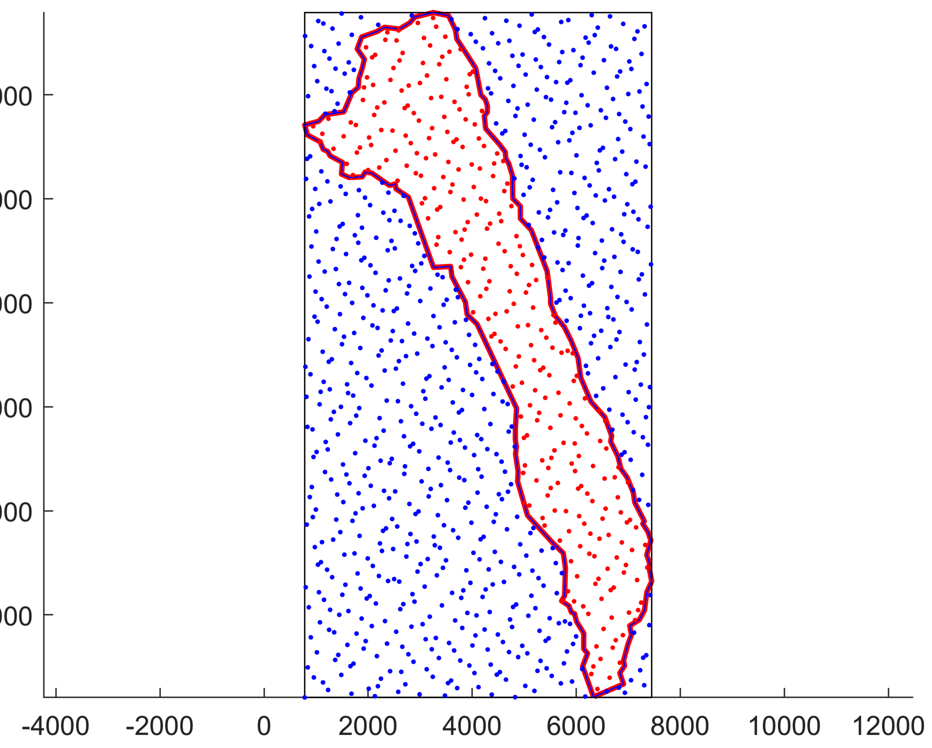
The previous figures present the pseudo random points, and the Halton points. The latter points are distributed a bit more equally, therefor we will use it this for our calculations.

Because the points are in the [0,1]x[0,1] range, lets transform them to the area of the relevant bounding box. For this lets shift the points to the starting point, and multiply the side of the rectangle to the proper size!

```
% Transforming the Halton points into the range, where the shape is
xh = xyh(:,1)*a + min(x);
yh = xyh(:,2)*b + min(y);
figure(7);
plot(xh,yh,'b.')
```

To apply the Monte-Carlo method, we should determine the number of points inside the relevant shape. We can use the **inpolygon** function in Matlab for this, which results in a logical vector, where the ones represent the points, that are inside the shape. The non-zero elements could be counted using the function **nnz**.

```
% Calculating area using Monte-Carlo method
plot(x,y,'b')
k = inpolygon(xh,yh,x,y);
axis equal
plot(xh(k),yh(k),'r.')
```



```
n = nnz(k) % number of points inside the area: 280
```

```
n = 280
```

```
N = length(xh) % total number of points: 1000
```

```
N = 1000
```

```
T = a*b % total area: 87824061 m^2
```

```
T = 87824061
```

```
t = n/N*T % area of the relevant region: 2.4591e+07
```

```
t = 2.4591e+07
```

```
format long
t %  2.459073708000000e+07 = 24590737.08e+07 m^2
```

```
t =
     2.459073708000000e+07
```

To check the result, we can use the formula used in geodesy, that divides the region into trapezoids (

$T = \sum \frac{(y_{i+1} + y_i)(x_{i+1} - x_i)}{2}$) !

```
% To check: calculating the area based on the coordinates (trapezoidal method)
x1 = x([2:end,1]);
y1 = y([2:end,1]);
```

```
        Tp = sum((y1+y).*(x1-x)/2) %  24591531 m^2
```

```
Tp =
    24591531
```

The two method have similar results, and if we add more points to the process, it could be more accurate. To determine the area of an irregular shape, there are several methods, the advantage of the Monte-Carlo method is that it could be generalized for other cases too. For example, we could calculate the amount of rain that fell on the catchment area, if we know the distribution of the rain, e.g. if we have measured on a couple of points the amount of rain!

The generalized form of the Monte-Carlo method

Lets express the Monte-Carlo method in a generalized form! Let $f(x)$ be interpretable in a $x \in V_T$ domain, and we search the definite integral of this function on a *V* sub-domain $V \subset V_T$. The integral we are searching: $\int_V f(x)dV$.

The mean value of the function in the relevant domain could be determined as follows:

$$\bar{f}_V = \frac{1}{V} \int_V f(x)dV$$

The mean value of the function could be approximated as follows:

$$\bar{f}_V \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

where $x_i \in V$ and *n* is the number of points in the relevant domain.

By making the expressions equal, the integral could be approximated:

$$\frac{1}{V} \int_V f(x)dV \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

From this the integral could be expressed:

$$\int_V f(x)dV \approx \frac{V}{n} \sum_{i=1}^{n} f(x_i)$$

A   tartomány közelítését megkaphatjuk, a területszámításnál is használt módon. Amennyiben a véletlenszerűen felvett pontok egyenletes eloszlást követnek, akkor kellően sok pont esetén a tartományon belül lévő pontok száma úgy aránylik az összes ponthoz, mint a   tartomány nagysága az egész   tartományhoz:

$$\frac{V}{V_T} = \frac{n}{N} \quad \rightarrow \quad V = \frac{V_T \bullet n}{N}$$

ahol *n* a tartományba eső, *N* pedig az összes pont száma. Az integrál közelítése tehát:

$$\int_V f(x)dV = \frac{V_T}{N} \sum_{i=1}^{n} f(x_i)$$

Therefor the integral could be calculated as the product of the function values at the points in the relevant domain and of the total area/total point ratio. The ratio of the V_T/N is determining basically the relevant elementary area on a single point.

Calculating the rainvolume using Monte-Carlo method

On the catchment area measurements were made to determine the fallen rain of a storm; on the measurement stations 5-12 mm rain were measured according to the location. The question is the total amount of rain fallen on the catchment area.

The following second order polynomial function approximates the amount of rain on the stations:

$$f(x, y) = 0.005 + 6 \bullet 10^{-7}\, x + 3 \bullet 10^{-7}\, y - 10^{-10}\, x^2 - 2 \bullet 10^{-11}\, xy + 2 \bullet 10^{-11}\, y^2$$

The function above should be integrated for the total catchment area. Lets solve this using Monte-Carlo method!

Solve the function, you can load it from the csap.mat file!

```
load precipfun.mat
pfun % @(x,y)5e-3+6e-07.*x+3e-07.*y-1e-10*x.^2-2e-11.*x.*y+2e-11*y.^2
```

```
pfun = function_handle with value:
    @(x,y)5e-3+6e-07.*x+3e-07.*y-1e-10*x.^2-2e-11.*x.*y+2e-11*y.^2
```

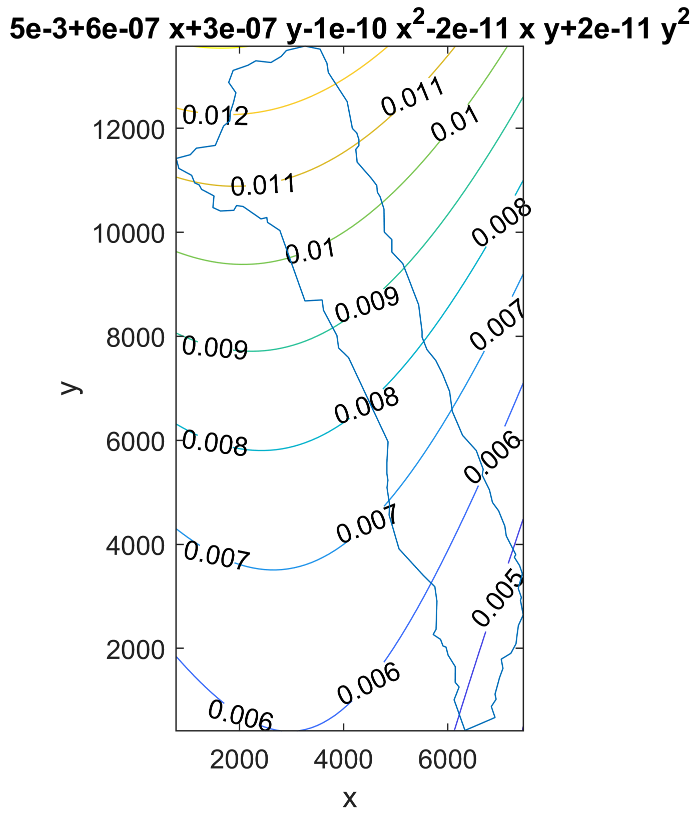
```
figure(10)
h=ezcontour(pfun,[min(x) max(x) min(y) max(y)])
```

```
h = 
  Contour with properties:

    LineColor: 'flat'
    LineStyle: '-'
    LineWidth: 0.500000000000000
         Fill: 'off'
    LevelList: [0.004000000000000 0.005000000000000 0.006000000000000 0.007000000000000 0.008000000000000 0.0090000
        XData: [60×60 double]
        YData: [60×60 double]
        ZData: [60×60 double]

  Show all properties
```

```
set(h,'Show','on'); hold on
plot(x,y); axis equal;
```



After generating 1000 random points on the relevant area, we can use these too. The total amount of rain could be calculated, if we determine the average rainfall value at the locations inside the catchment area, and we multiply it with the relevant area (because we already dcalculated that).

```
xb = xh(k);
yb = yh(k);
n = length(xb) % 280
```

```
n =
   280
```

```
% The average rain value in the area
cs = 1/n*sum(pfun(xb,yb)) % 0.008612820224953 m
```

```
cs =
   0.008612820224953
```

```
% The total amount of rain
CS = cs*t % 2.117955976691141e+05
```

```
CS =
     2.117955976691141e+05
```

Or we could use the generalized Monte-Carlo forumla, in this case the calculation of the area of the irregular shape is not necessaryt, its enough to calculate the area of the bounding box and the total number of points, and the sum of the function values at the points inside the relevant region.

```
% Using the generalized Monte-Carlo method,
% if we don't calculate the area separately
```

```
        CS2 = T/N*sum(pfun(xb,yb)) % 2.117955976691141e+05
```

```
CS2 =
     2.117955976691141e+05
```