Numerical Derivation

The derivative (or differential) is describing the change of a quantity. It is used in many fields of engineering or science in general. The most known physical example is the displacement-velocity-acceleration formula. If we examine the position \(x\) according to time \(t\): \(x = f(t)\), then the velocity of an object \(v(t)\) is the derivative according to time of the displacement (therefor the tangent of the position-time diagram): \(v = \frac{df(t)}{dt}\). And the acceleration is the second derivative according to time, therefor the first derivative of velocity: \(a = \frac{dv(t)}{dt}\). We can also use the derivative of a function to find the minimum/maximum of a function, as we saw earlier.

The function which should be derivated could be defined as an analytical expression, or as values in a discrete manner. Simple mathematical expressions could be calculated analytically. In case of complicated mathematical expressions, or discrete data points the conventional method is to apply numerical derivation. Numerical derivation has also a major role while solving differential equations.

The major numerical derivation is the approximation of the derivative using finite differences. Another approach is to approximate the points using some kind of analytical expression (e.g. polynomial regression or interpolation) and we differentiate the analytical function.

Finite difference approximation

Let's assume we only know the function values that should be derivated as discrete data points, We can approximate the derivative using the slope of the line between the neighbouring points. For this there are multiple options, First the right side or forward difference:

\[
\left( f(x) \right)_i = f'_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\]

Or another option is the left side or backward difference:

\[
\left( f(x) \right)_i = f'_i = \frac{y_{i-1} - y_i}{x_i - x_{i-1}}
\]

Because the errors of the left and right side differentials usually have opposing signs, a better solution is to apply the average of the two. If the subdivision of the points are equally spaced \((x_{i+1} - x_i = x_i - x_{i-1} = h)\), it is leading to the following central differential formula:

\[
\left( f(x) \right)_i = f'_i = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}} = \frac{y_{i+1} - y_i}{2h}
\]

Usually the central differential formula is a better approximation of the derivative, than the left/right side differential. The decrease of distance between the intermediate points is also raising the accuracy.
The errors of finite difference approximation

To estimate the truncation errors of the left/right/central differences, we can apply Taylor series:

\[ f(x + h) = f(x) + h f'(x) + \frac{h^2}{2} f''(x) + O(h^3) \]

where \( c \) is an unknown number between \( x \) and \( x+h \). Let it be \( x = x_i \) \( x + h = x_{i+1} \) and then express \( f \) from the expression:

\[ f(x) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2} f''(c) \]

The formula above is the right side difference, that has an error of \( \frac{h}{2} f''(c) \), so it has a first order error. If we substitute \( h \) in the previous formula with \((-h)\), we get the left side difference. The central difference error formula could be derived from the third order Taylor series:

\[ f(x_i) = f(x_i + h) = f(x_i) + h f'(x_i) + \frac{h^2}{2} f''(x_i) + \frac{h^3}{6} f'''(c_i) \]

\[ f(x_i) = f(x_i - h) = f(x_i) - h f'(x_i) + \frac{h^2}{2} f''(x_i) - \frac{h^3}{6} f'''(c_i) \]

where \( x_i - h < c_i < x_{i+1} \) and \( x_{i-1} < c_i < x_i \). Just subtract the two equation from each other, and express \( f'' \):

\[ f(x_i) = \frac{f(x_{i+1}) - f(x_{i-1}) - 2 f(x_i)}{2 h} + \frac{h^2}{12} f'''(c_i) + O(h^4) \]

Based on this, the error of the central difference has a second order error \( O(h^2) \), which is a better approximation, so it is preferable to use.

To improve this approximation, more points could be involved, e.g. the 5 point central difference formula would have a fourth order error \( O(h^4) \):

\[ f(x_i) = y_i = \frac{y_{i-2} - 8 y_{i-1} + 8 y_{i+1} - y_{i+2}}{12 h} + O(h^4) \]

This is also valid for the error of the left/right side difference. For the first derivative in case of the left and right side difference using from 3 points:

\[ f'(x_i) = \frac{y_{i+1} - y_{i-1}}{2 h} + O(h^2) \]

\[ f'(x_i) = \frac{y_{i+1} - y_{i-1} - 2 y_i}{2 h} + O(h^2) \]

Higher order differentials

Higher order derivative could also be expressed using the central difference formula, in this case they have a second order error. The central difference formula for the second derivative using 3 points:

\[ f''(x_i) = \frac{y_{i+1} - 2 y_i + y_{i-1}}{h^2} + O(h^2) \]

The central difference formula for the third derivative using 4 points:

\[ f'''(x_i) = \frac{1}{h^3}(y_{i+2} - 2 y_{i+1} + 2 y_{i-1} - y_{i-2}) + O(h^2) \]

The central difference formula for the fourth derivative using 5 points:

\[ f^{(4)}(x_i) = \frac{1}{h^4}(y_{i+2} - 4 y_{i+1} + 6 y_i - 4 y_{i-1} + y_{i-2}) + O(h^2) \]

The left/right side difference formula for the second derivative using 3 points:

\[ f''(x_i) = \frac{y_{i+1} - 2 y_i + y_{i-1}}{h^2} + O(h^2) \]

\[ f''(x_i) = \frac{y_{i+2} - 2 y_{i+1} + y_i}{h^2} + O(h^2) \]

The left/right side difference formula for the second derivative using 4 points:
\[ f'(x) = y^2 - 5y_{i+1} + 4y_{i+2} - y_{i-1} + O(h^2) \]

\[ f'(x) = \frac{-y_{i-1} + 4y_{i} - 5y_{i+1} + 2y_{i+2} + O(h^2)}{h^2} \]

Applying finite differences

The height positions of a space shuttle are given in the first two minutes after launching:

<table>
<thead>
<tr>
<th>t(s)</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
<th>110</th>
<th>120</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(m)</td>
<td>-8</td>
<td>241</td>
<td>1244</td>
<td>2872</td>
<td>5377</td>
<td>8130</td>
<td>11617</td>
<td>15380</td>
<td>19872</td>
<td>25608</td>
<td>31412</td>
<td>38309</td>
<td>4472</td>
</tr>
</tbody>
</table>

Let's determine the velocity of the space shuttle according to time! Use the central difference formula wherever its possible! Load the data using the file 'space_shuttle.txt' and plot the height position according to time!

```matlab
clear all; close all;
data = load('space_shuttle.txt');
t = data(:,1); h = data(:,2);
figure(1); plot(t,h,'r-');
xlabel('time [s]');
ylabel('height [m]');
title('Position-time diagram');
```

The velocity is the first derivative of the position according to time. The central formula could be applies only from the second point to the second-to-last point. In the first point only the right side difference formula, and in the last point only the left side difference formula could be applied. Write a function, that calculates the first derivative in all positions, use the central difference formula wherever its possible! Use the previously created `derivalt.m`!

```matlab
v = derivalt(t,h);
figure(2); plot(t,v,'b-');
xlabel('time [s]');
ylabel('velocity [m/s]');
title('Velocity-time diagram');
```

The built-in matlab function `diff` could be also used to calculate the derivative in both numerical and symbolical cases. Though for numerical solution it only calculates a one side differences, but not central differences. It only calculates the differences between the neighbouring elements in a vector. The result will have one less number of elements, than the input vector. Lets determine the velocities and accelerations usin left side differences by applying the `diff` function!

```matlab

The figure shows well, that the central differences formula results in smoother curves, than the single side difference formula. Acceleration could be calculated similarly by differentiating the velocity!

Numerical derivation using fitting function

Lets have a look on the other approximation, when we fit an approximating function on the points. Let it be now a second order polynomial!

```matlab
% differentiating using fitting function
clf; close all;
c = polyfit(t,h,2);
x = 3.0470 10.1151 -8.3626
p = @(x) polyval(c,x);
figure(1); hold on;
plot(t(:,end),diff(t),r-')
```

The parabola is clearly well fitted on the points. If we use this approximation to describe the data points symbolically (sym), then the `diff` function could be used just as earlier. Though there is a more simple way: the function `polyder` lets you convert the solution into symbolical form directly:

```matlab
% velocity
clf; close all;
c = 6.6659 10.1151
p1 = @(x) polyval(c,x);
figure(2);
h1 = expot(p1,[min(t),max(t)]);
set(h1,'Color','g','LineWidth',2);
```

In this case the derivative results in an even smoother form, than the previous lines!

**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 \(L = \int_a^b \sqrt{1 + (f'(x))^2} \, dx\), 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:

\[ \int_a^b f(x) \, dx \approx \frac{f(a) + f(b)}{2} (b - a) \]

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

\[ \int_a^b f(x) \, dx = \frac{1}{2} \sum_{i=1}^{n} (f(x_i) + f(x_{i+1}))(x_{i+1} - x_i) \]

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

\[ \int_a^b f(x) \, dx = \frac{b - a}{2} \sum_{i=1}^{n} (f(x_i) + f(x_{i+1})) \]

Let's see how this works in an example!

Earth's density \( \rho \) 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:

\[ M_{\text{earth}} = \int_0^{R_{\text{earth}}} 4\pi R^2 \, dR \]

For the density data, load earth_density.txt file!

<table>
<thead>
<tr>
<th>% Mass of the Earth</th>
<th>close all; clear all; data = load('earth_density.txt')</th>
</tr>
</thead>
<tbody>
<tr>
<td>R = data(:,1).*1000</td>
<td>ro = data(:,2)</td>
</tr>
<tr>
<td>figure(1); plot(R,ro,'r-*')</td>
<td></td>
</tr>
</tbody>
</table>

Let's calculate the function values for the relevant radius values!

\[ f(x) = 4\pi \rho \cdot r^2 \]

To solve this, let's use the built-in Matlab function \texttt{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 distant.

\[ M = \text{trapz}(R,f) \quad \% \quad 6.0261e+24 \text{ kg} \]

This means the mass of the Earth is \( 6.0261 \times 10^{24} \) kg. This is quite a good approximation for the currently accepted mass of the Earth \( 5,9722 \pm 0.0006 \times 10^{24} \) kg.

Numerical integral using the Simpson rule

The trapezoidal rule approximates the function between the neighboring points with a line. A more accurate result could be achieved using a higher order approximating function. The most known of these methods 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:

\[ p(x) = a_0 + a_1 (x-x_0) + a_2 (x-x_0)(x-x_1) \]

Where the coefficients are the following:

\[ a_0 = f(x_0); a_1 = \frac{y_1 - y_0}{x_1 - x_0}; a_2 = \frac{y_2 - y_1}{x_2 - x_1} \]

In case of equal intervals \( h \):

\[ a_0 = f(x_0); a_1 = \frac{y_1 - y_0}{h}; a_2 = \frac{y_2 - 2y_1 + y_0}{2h^2} \]

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

\[ \int_{x_0}^{x_1} f(x) \, dx \approx \int_{x_0}^{x_1} p(x) \, dx = \frac{h}{3} \left( f(x_0) + 4 f(x_1) + f(x_2) \right) \]

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 distant by \( h \) in the given range \([a,b]\). \( x_0 = a; x_n = b \). 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, therefore we should divide the range always into even number of segments:

\[ \int_a^b f(x) \, dx = \int_{x_1}^{x_2} f(x) \, dx + \int_{x_3}^{x_4} f(x) \, dx + \ldots + \int_{x_{2n-3}}^{x_{2n-2}} f(x) \, dx = \sum_{i=1}^{N} \int_{x_i}^{x_{i+1}} f(x) \, dx \]

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

\[ \int_a^b f(x) \, dx \approx \frac{h}{3} \left( f(a) + 4 \sum_{i=1}^{N} f(x_i) + 2 \sum_{i=1}^{N} f(x_i) + f(b) \right) \]

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

```matlab
% Cubic Hermite interpolation
ro_cubic = @(x) interp1(Rro,x,'cubic')
figure(1); hold on;
g = ezplot(ro_cubic, [0 6378000]);
set(g,'Color', 'k','LineStyle', '-.','LineWidth',1);
legend('Original data','Cubic Hermite interpolation')
```

Let's 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 function.

```matlab
fx_cubic = @(R) 4*pi*ro_cubic(R).^2
M2 = quad(fx_cubic,0,6378000) % 5.9665e+24 kg
```

The result for the mass of the Earth is $5.9665 \times 10^{24}$ kg. This is a better approximation than the value we got using the trapezoidal rule.

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

```matlab
M3 = integral(fx_cubic,0,6378000) % 6.0541e+24
```

Calculating multidimensional integrals on regular grid

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

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

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.

```matlab
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, let's calculate the area based on the boundary points! Load the catchment.txt, and plot it with equally scaled axis!

```matlab
clear all; close all;
data = load('catchment.txt');
x = data(:,1); y = data(:,2);
figure(1); hold on;
plot(x,y,'-','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 ($\rho$) 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: $A = a \cdot b$, then the area in the arbitrary shaped range (in this case the catchment area) could be calculated:

$$T_c = \int \int f(x) \, dT = \rho A = \frac{n}{N} \cdot (a \cdot b)$$

Let's solve the example in Matlab! First draw the bounding box around the shape!

```matlab
a = max(x)-min(x) % 6.6698e+03
b = max(y)-min(y) % 1.31609e+04
rectangle('Position',[min(x),min(y),a,b])
```

Let's 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. Let's generate with these 1000 points! Both function is working in the range of [0, 1].

```matlab
% Generating pseudo random points
x = rand(1000,2);
figure(2);
```
The previous figures present the pseudo random points, and the Halton points. The latter points are distributed a bit more equally, therefore we will use this for our calculations.

Because the points are in the \([0,1]\times[0,1]\) range, let's transform them to the area of the relevant bounding box. For this, let's shift the points to the starting point, and multiply the side of the rectangle to the proper size.

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 nz.

To check the result, we can use the formulas used in geodesy, that divides the region into trapezoids (\( T = \sum_{i=1}^{n} \left( y_{i+1} + y_{i} \right) \left( x_{i+1} - x_{i} \right) / 2 \)).

The two methods 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 general form of the Monte-Carlo method

Let's express the Monte-Carlo method in a generalized form! Let \( f(x) \) be interpretable in a \( V \subset V_{x} \) domain, and we search the definite integral of this function on a \( V \) sub-domain \( V \subset V_{x} \). 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 \( X \in V \) 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:

\[ \int_{V} f(x) \, dV = \frac{V}{n} \sum_{i=1}^{n} f(x_{i}) \]

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

\[ \frac{V}{n} = \frac{n}{N} \rightarrow V = \frac{V_{x} \cdot n}{N} \]

ahol \( n \) a tartományba eső, \( X \) pedig az összes pont száma. Az integrál közelítése tehát:

\[ \int_{V} f(x) \, dV = \frac{V_{x}}{N} \sum_{i=1}^{n} f(x_{i}) \]

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 \( V_{x}/N \) is determining basically the relevant elementary area on a single point.
Calculating the rain volume 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:

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

The function above should be integrated for the total catchment area. Let’s solve this using Monte-Carlo method.

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

```
load csap.mat

csap = @(x,y)5e-3+6e-7.*x+3e-7.*y-1e-10.*x.^2-2e-11.*x.*y+2e-11.*y.^2

figure(4)
hold on
h=ezcontour(csap,[min(x) max(x) min(y) max(y)])
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 calculated that).

```
xb = x(h);
yb = y(h);
n = length(xb) % 280
% The average rain value in the area
CS = 1/n*sum(csap(xb,yb)) % 0.003852820226953 n
% The total amount of rain
CS = CS*n % 2.117959976699141e-05
```

Or we could use the generalized Monte-Carlo formula; in this case the calculation of the area of the irregular shape is not necessary, it's 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 = 1/n*sum(csap(xb,yb)) % 2.117955976699141e-05
```

```
function dx = derivalt(x,y)
% Numerical derivative using finite difference approximation
n = length(x);
dx(1) = (y(2)-y(1))/(x(2)-x(1)); % right side difference
for i = 2:n-1
    dx(i) = (y(i+1)-y(i-1))/(x(i+1)-x(i-1)); % central diff.
end
dx(n) = (y(n)-y(n-1))/(x(n)-x(n-1)); % left side difference
end
```