Numerical differentiation

Differentiation of a quantity makes us able to track its change with respect to some variable. The most common example is the calculation of velocity from the function of position. If the position is given as a function $x = f(t)$ of time, the velocity is calculated as the derivative of this function

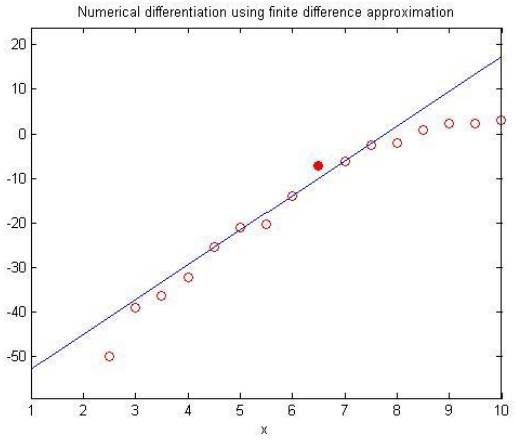
$$v = \frac{df(t)}{dt}$$

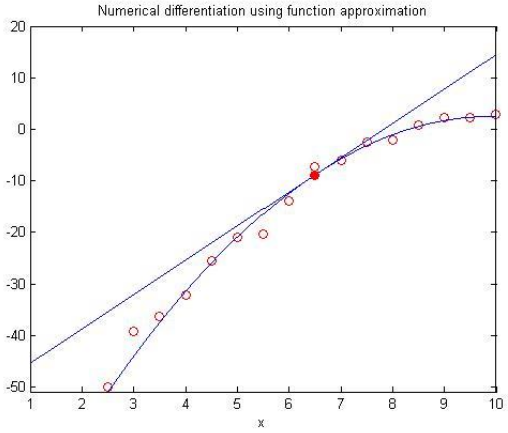
and the acceleration can be calculated as the second derivative of the velocity function:

$$a = \frac{dv(t)}{dt} = \frac{d^2 f}{dt}.$$

The derivative can also be used to find the minimum or maximum of a function. Analytic (symbolic in MATLAB) derivation can be used when the function can be given in analytic form that is somewhat easy to differentiate. If the function is given as a set of data points or if it is not possible to differentiate it analytically, numeric differentiation is the only option. Numeric differentiation also plays a very important role in solving differential equations.

If the function is acquired as a set of data points, the differential can be approximated by finite differences or another option can be to first approximate the function using some form of interpolation or regression and then differentiate this resulting analytic (and mostly simple) function.





Finite difference approximation

Let's suppose that we can only acquire the function values in a limited set of data points. In such a case, we can approximate the differential by the slope of the lines connecting the points. This can be done in two ways:

- using the right side or forward difference:

$$f'(x_i) = y'_i \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

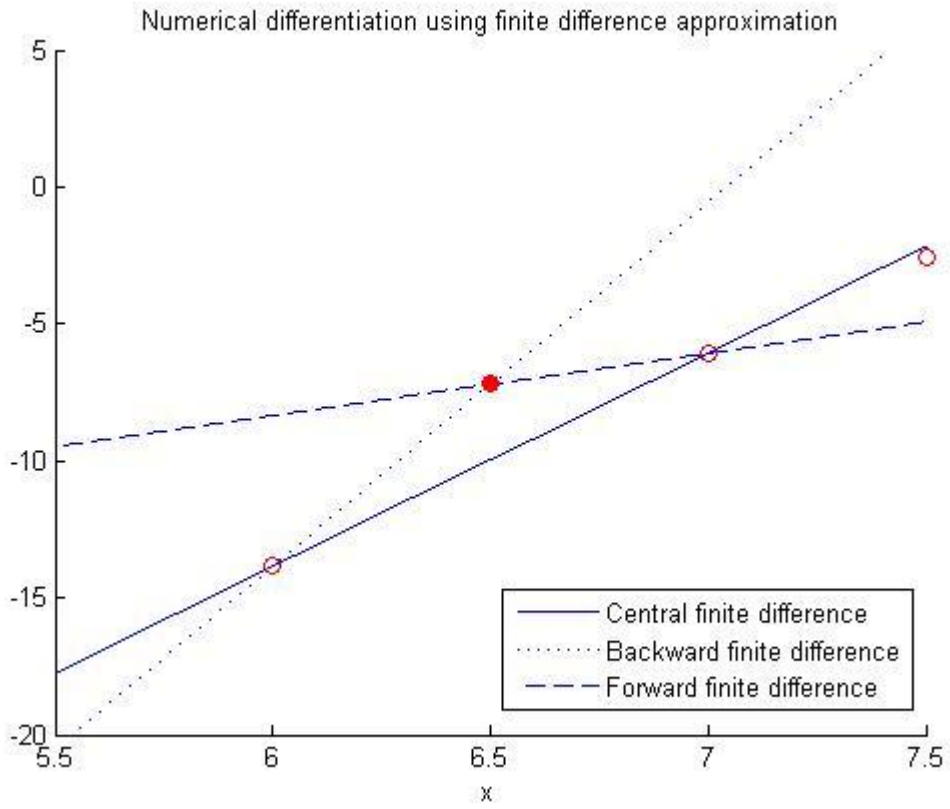
- using the left side of backward difference:

$$f'(x_i) = y'_i \approx \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

As the errors of the forward and backward differences usually have the opposite sign, it is better to take the average of the two. If the spacing of the data points is homogeneous and $x_{i+1} - x_i = x_i - x_{i-1} = h$, this will result in the central difference:

$$f'(x_i) = y'_i \approx \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}} = \frac{y_{i+1} - y_{i-1}}{2h}$$

Generally, the central difference is a better approximation of the derivative then the other two. As the spacing of the points becomes smaller, the accuracy further increases.



Errors of finite difference approximations

Taylor series can be used to approximate the truncation errors corresponding to the forward and backward differences:

$$f(x+h) = f(x) + h \cdot f'(x) + \frac{h^2}{2} \cdot f''(c)$$

where c is an unknown value between x and $x+h$. Let $x = x_i$ and $x + h = x_{i+1}$ and solve the equation for $f'(x)$:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2} \cdot f''(c)$$

The formula above is the one used to calculate the forward difference with an extra subtraction on the left side, the error term $-\frac{h}{2} \cdot f''(c)$. That is, the magnitude of the truncation error is $\mathrm{O}(h)$ (big Ordo h). If we change h to $-h$ in the formula, we get the truncation error of the backward difference. The error of the central difference can be computed using third order Taylor series:

$$f(x_{i+1}) = f(x_i + h) = f(x_i) + h \cdot f'(x_i) + \frac{h^2}{2} \cdot f''(x_i) + \frac{h^3}{3!} \cdot f'''(c_1)$$

$$f(x_{i+1}) = f(x_i - h) = f(x_i) - h \cdot f'(x_i) + \frac{h^2}{2} \cdot f''(x_i) - \frac{h^3}{3!} \cdot f'''(c_2)$$

where $x_i < c_1 < x_{i+1}$ and $x_{i-1} < c_2 < x_i$. Subtract the two equations from each other and solve it for $f'(x_i)$:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{h^2}{3!} \cdot \frac{f'''(c_1) + f'''(c_2)}{2}$$

which shows that the truncation error of the central difference is $\mathrm{O}(h^2)$, which results in a much better approximation if h can be reduced.

We can approximate the differential using even more points, using 5 points for example gives us a central formula with $\mathrm{O}\left(h^4\right)$

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

The forward and backward difference can also be made better by involving more points. Using three points, the formulas for the forward and the backward differences become:

$$f'(x_i) = y_i' \approx \frac{-3y_i + 4y_{i+1} - y_{i+2}}{2h} + \mathrm{O}(h^2)$$

$$f'(x_i) = y_i' \approx \frac{y_{i-2} - 4y_{i-1} + 3y_i}{2h} + \mathrm{O}(h^2)$$

Higher order differentials

Similar numeric formulae can be derived for higher order derivatives as well. For example, the second derivative using central difference and 3 points:

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

Central differences for higher order derivatives will always have an error term that is proportional to h^2.

The second order derivatives using forward and backward differences:

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

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

How can these formulae be derived? Let's take a look at the 3 point forward difference formula. We can start from the Taylor series of the function at the three data points used (y_i, y_{i+1}, y_{i+2}):

$$y_i = f(x_i) = f(x_i) + 0 \cdot h \cdot f'(x_i) + 0 \cdot \frac{h^2}{2!} \cdot f''(x_i) + \mathrm{O}(h^3)$$

$$y_{i+1} = f(x_i + h) \approx f(x_i) + h \cdot f'(x_i) + \frac{h^2}{2!} \cdot f''(x_i) + \mathrm{O}(h^3)$$

$$y_{i+2} = f(x_i + 2 \cdot h) \approx f(x_i) + 2 \cdot h \cdot f'(x_i) + \frac{4 \cdot h^2}{2!} \cdot f''(x_i) + \mathrm{O}(h^3)$$

The above is a linear system of equations with 3 unknowns and 3 equations. We need to figure out a linear combination of the rows that cancel out all the term except the second derivatives. If we denote the coefficients (the multiplicators) of the lines as c_1, c_2, c_3, we can write:

$$c_1 \cdot y_i + c_2 \cdot y_{i+1} + c_3 \cdot y_{i+2} = 0$$

$$c_1 \cdot 0 \cdot y_i + c_2 \cdot 1 \cdot y_{i+1} + c_3 \cdot 2 \cdot y_{i+2} = 0$$

$$c_1 \cdot 0 \cdot y_i + c_2 \cdot \frac{1}{2} \cdot y_{i+1} + c_3 \cdot 2 \cdot y_{i+2} = 1$$

The first row corresponds to the first column of the previous system (the column for $f(x_i)$), the second row to the column containing $f'(x_i)$ and the third row to the column containing $f''(x_i)$. As in the end we only need the second derivative, we equate every row except the last with 0. By making the last row equal to one, we find a combination which will results in $1 \cdot f''(x_i)$ (+ the error term).

(Similarly, if we wanted to find the approximation of the first derivative, we would need to make the second row - and only the second row - equal to 1.)

Writing the system using matrix notation:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & \frac{1}{2} & 2 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

If we solve this using Gauss elimination for example and find the values of the coefficients, we get: $c_1 = 1,\ c_2 = -2,\ c_3 = 1$. In other words:

$$1 \cdot y_i - 2 \cdot y_{i+1} + 1 \cdot y_{i+2} = 0 \cdot f(x_i) + 0 \cdot h \cdot f'(x_i) + 1 \cdot h^2 \cdot f''(x_i) + \mathrm{O}(h^3) = h^2 \cdot f''(x_i) + \mathrm{O}(h^3)$$

$$f''(x_i) \approx \frac{y_i - 2y_{i+1} + y_{i+2}}{h^2} + \mathrm{O}(h)$$

(The sign of the $\mathrm{O}(h)$ is arbitrary as it only symbolizes the proportion of the error term to the step size h.)

Numeric differentiation is practice

The following table contains the launch data of the space shuttle:

t(s)	0	10	20	30	40	50	60	70	80	90	100	110	120
h(m)	-8	241	1244	2872	5377	8130	11617	15380	19872	25608	31412	38309	44726

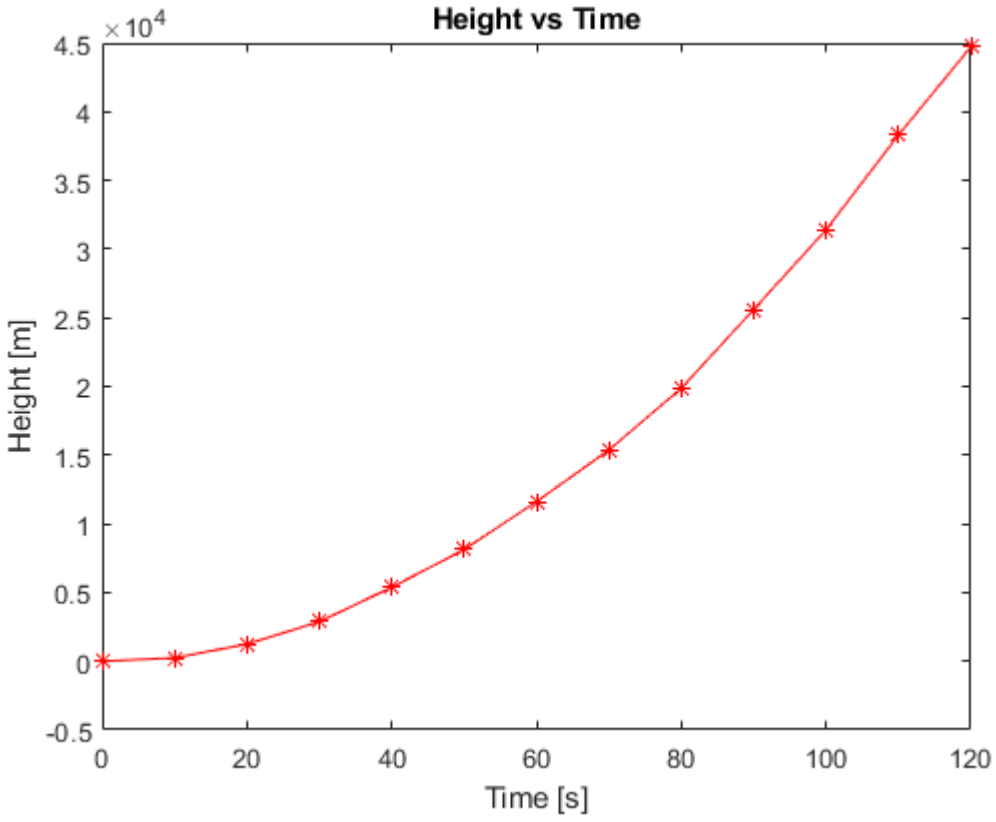
Let's solve the following two tasks!

• What will be the velocity of the space shuttle after 30 seconds using forward/backward or central difference formula?

• Determine the velocity of the space shuttle as a function of time! Where possible, use the central difference formula!

We can load the data from the **`shuttle.txt`** file and plot the data first:

```
data = load('shuttle.txt');
t = data(:, 1);
h = data(:, 2);
figure(1);
plot(t, h, 'r*-');
title('Height vs Time');
xlabel('Time [s]');
ylabel('Height [m]');
```



Let's first answer the first question, what will be the speed of the space shuttle after 30 seconds! The velocity is the first derivative of the position function (height in our case) with respect to time. This can be approximated with finite difference methods, using forward/backward or central difference formulas. Let's see the final result of each approximation at the given time! Let's use the formulas given in the finite difference approximation part! Examining the data, we see that the value at t=30 seconds is the 4th measurement data (0 s, 10 s, 20 s, 30 s). Given formulas for i=4 will be:

forward diff. formula	backward diff. formula	central difference formula
$y_4' \approx \frac{y_5 - y_4}{x_5 - x_4}$	$y_4' \approx \frac{y_4 - y_3}{x_4 - x_3}$	$y_4' \approx \frac{y_5 - y_3}{x_5 - x_3}$

```
% velocity at t=30 s
t(4) % 30 s, 4th point!
```

```
ans = 30
```

```
% forward difference formula
v1 = (h(5)-h(4))/(t(5)-t(4))
```

```
v1 = 250.5000
```

```
% backward difference formula
```

```
v2 = (h(4)-h(3))/(t(4)-t(3))
```

```
v2 = 162.8000
```

```
% central difference formula
v3 = (h(5)-h(3))/(t(5)-t(3))
```

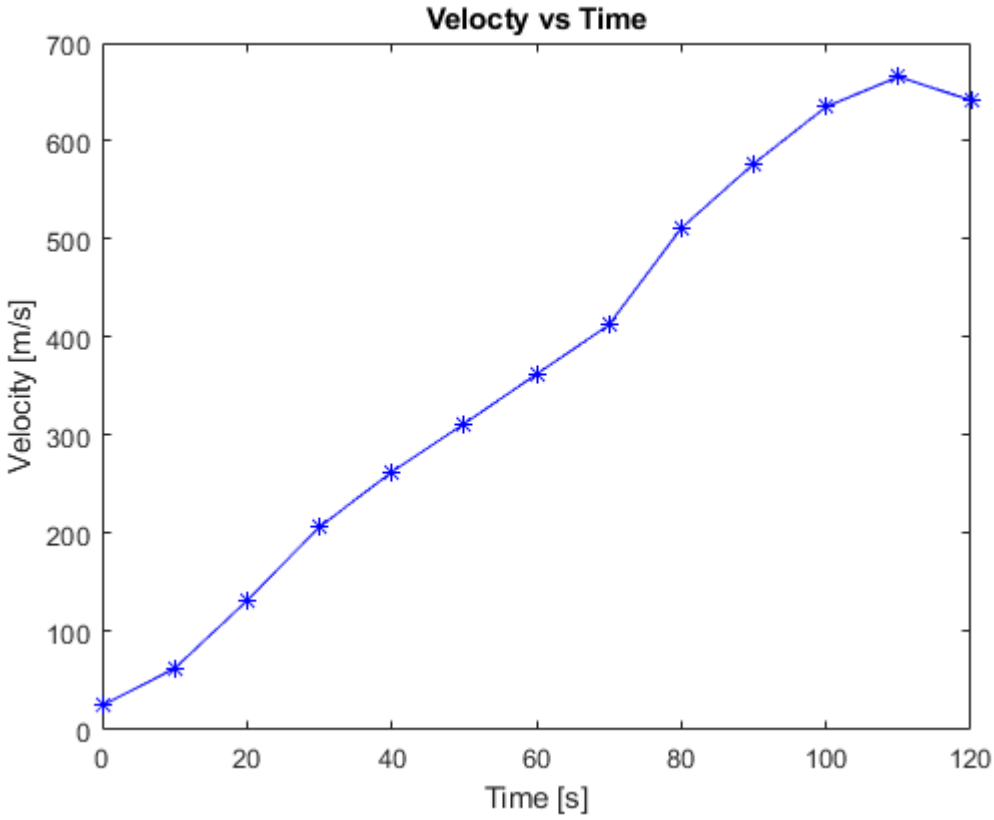
```
v3 = 206.6500
```

In the second question, we want to determine the velocity of the space shuttle as a function of time, where possible using the central difference formula. The velocity function is the first derivative of the position function (height in our case). The central difference formula can only be used from the second point to the next to last point, as it takes into account the point to the left and to the right of the current data point as well. In the first point, we can only use a forward difference and in the last point only a backward difference. Let's create a custom function that takes care of this calculation for us:

```
function dx = derivative(x, y)
% Numeric differentiation using finite differences

    n = length(x);
    dx(1) = (y(2)-y(1))/(x(2)-x(1)); % forward difference

    for i = 2:n-1
        dx(i) = (y(i+1)-y(i-1))/(x(i+1)-x(i-1)); % central difference
    end

    dx(n) = (y(n)-y(n-1))/(x(n)-x(n-1)); % backward difference

end
```

This function is located in the **`derivative.m`** file. Let's use it to calculate the velocity function and plot its graph:

```
v = derivative(t, h);
figure(2);
plot(t, v, 'b*-');
xlabel('Time [s]');
ylabel('Velocity [m/s]');
title('Velocty vs Time');
```

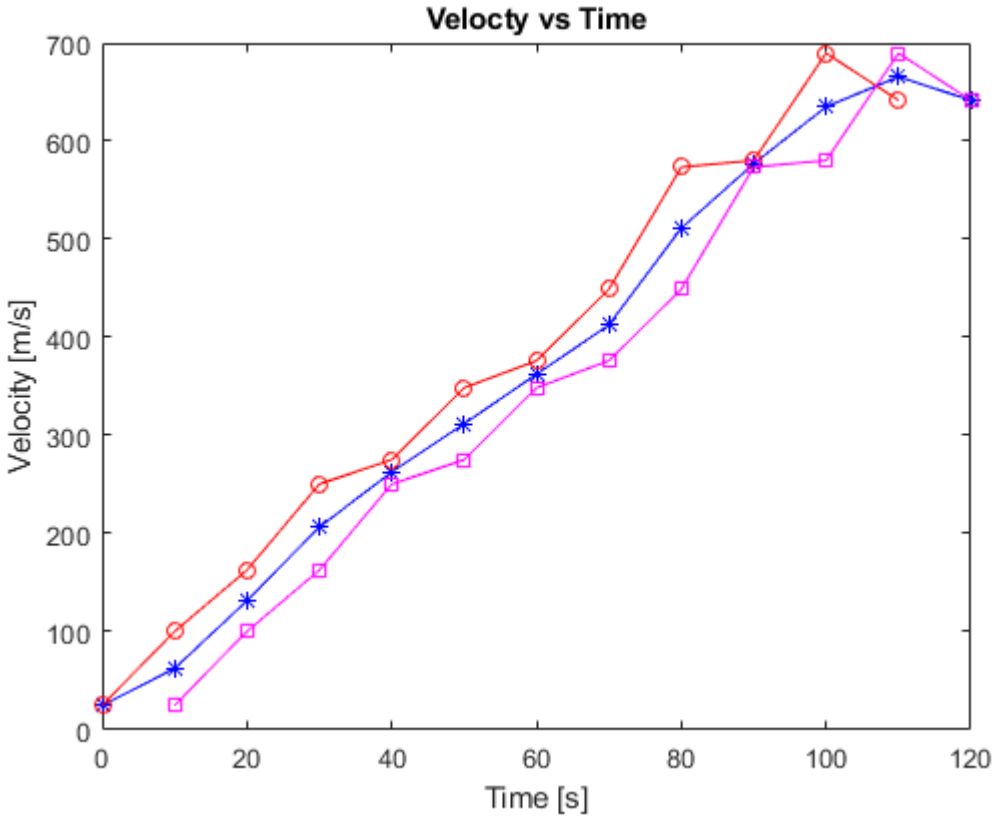


The built-in **`diff`** command can also be used to calculate the finite differences. We can first compute the differences in time and height and divide the two to get the finite differences for approximating the derivative:

```
dh = diff(h);
dt = diff(t);
dhdt = dh./dt;
```

The calculation of the forward and backward differences are essentially the same. The forward differences can be computed until the next to last point, while the backward differences can be computed from second point:

```
figure(2);
hold on;
plot(t(1:end-1), dhdt, 'ro-');
plot(t(2:end), dhdt, 'ms-');
```



From the figure, we can see that the central differences give a somewhat smoother curve that the other two.

The applied **`derivative.m`** function approximated the derivative from the second point to the next to last point an error of $\mathrm{O}(h^2)$, while the error is $\mathrm{O}(h)$ in the first and the last point. We could make the approximation more accurate by using 3 point approximations in the first and the last point as well, giving an error of $\mathrm{O}\left(h^2\right)$. Using the previously mentioned formula for the 3 point forward and backward differences:

```
v1 = (-3*h(1) + 4*h(2) - h(3))/(t(3) - t(1))
```

```
v1 = -12.8000
```

```
vn = (h(end-2) - 4*h(end-1) + 3*h(end))/(t(end) - t(end-2))
```

```
vn = 617.7000
```

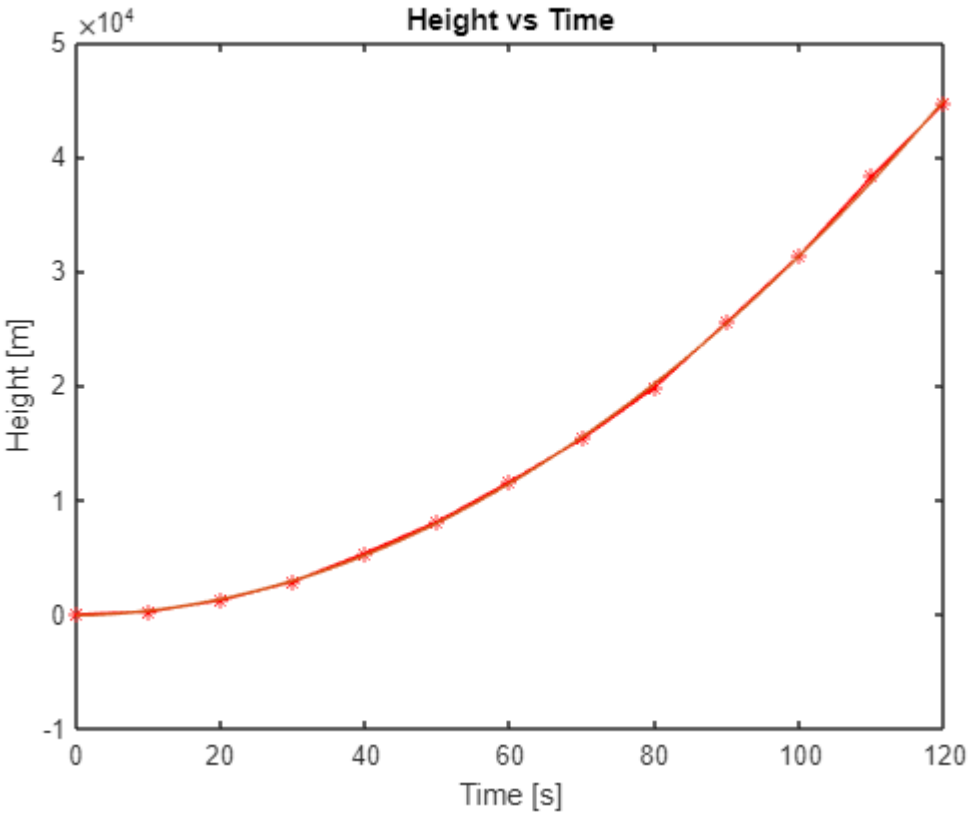
Numeric differentiation by function approximation

Let's look at the other approach for approximating the derivative. In this case, we first fit a function to the data points, preferably one that can be differentiated easily, and we compute the analytic derivative of that function. Determine the value of the derivative function using polynomial fitting at time t=30 s! Let's use a quadratic polynomial as our model:

```
c = polyfit(t, h, 2)
```

```
c = 1×3
    3.0470   10.1151  -89.3626
```

```
p = @(x) polyval(c, x);
figure(1);
hold on;
fplot(p, [min(t), max(t)]);
```



We can see that the quadratic polynomial fits the data really well.

If we defined the polynomial in symbolic form, we could use the same **`diff`** command to differentiate the function symbolically.

```
syms x
ps = c(1)*x.^2 + c(2)*x + c(3)
```

```
ps =
```

$$\frac{3050\,x^2}{1001}+\frac{50626\,x}{5005}-\frac{3144168283806241}{35184372088832}$$

```
v2 = diff(ps, x) % using symbolic derivation
```

```
v2 =
```

$$\frac{6100\,x}{1001}+\frac{50626}{5005}$$

```
v2 = matlabFunction(v2)
```

```
v2 = function_handle with value:
    @(x)x.*6.093906093906094+1.011508491508492e+1
```

```
v2(30)
```

```
ans = 192.9323
```

In the case of polynomials, we can also use the built-in **`polyder`** command and if we use this, we don't even have to define the polynomial symbolically. Let's take a look at both solutions:

```
c1 = polyder(c) % using the polyder command which takes the coefficient of the original polynor
```

```
c1 = 1×2
    6.0939   10.1151
```

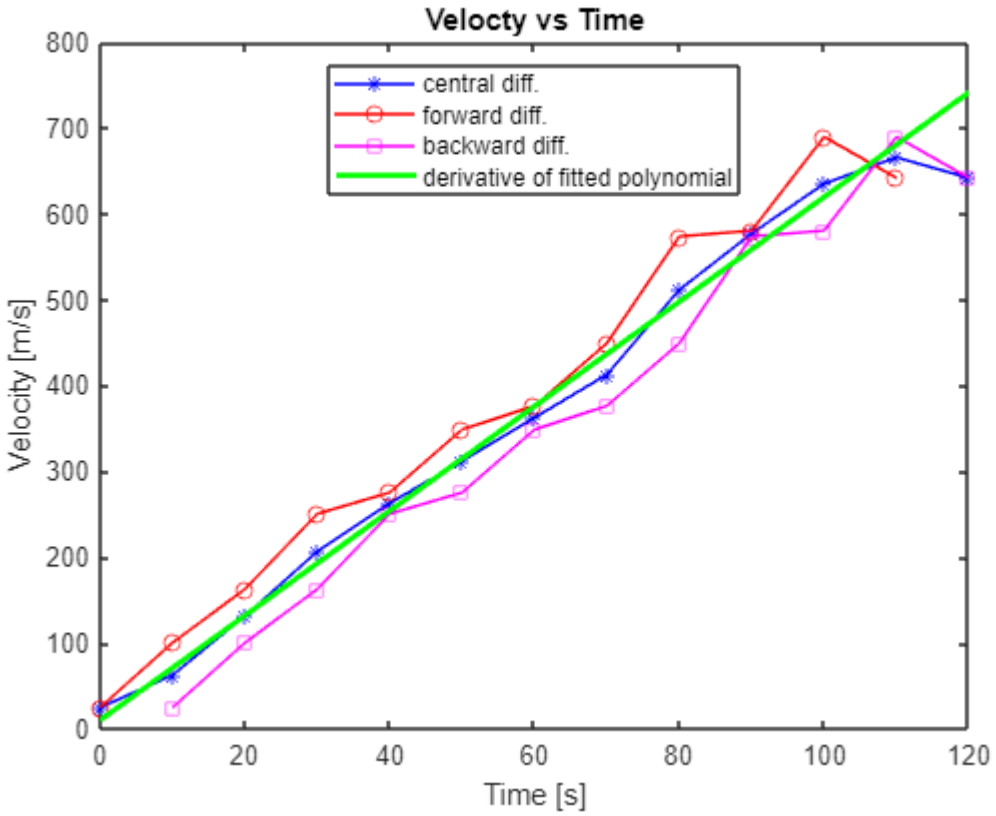
```
v2 = @(x) polyval(c1, x)
```

```
v2 = function_handle with value:
    @(x)polyval(c1,x)
```

```
v2(30)
```

```
ans = 192.9323
```

```
figure(2);
fplot(v2, [min(t), max(t)], 'g', 'LineWidth', 2);
legend('central diff.', 'forward diff.', 'backward diff.', 'derivative of fitted polynomial',
```



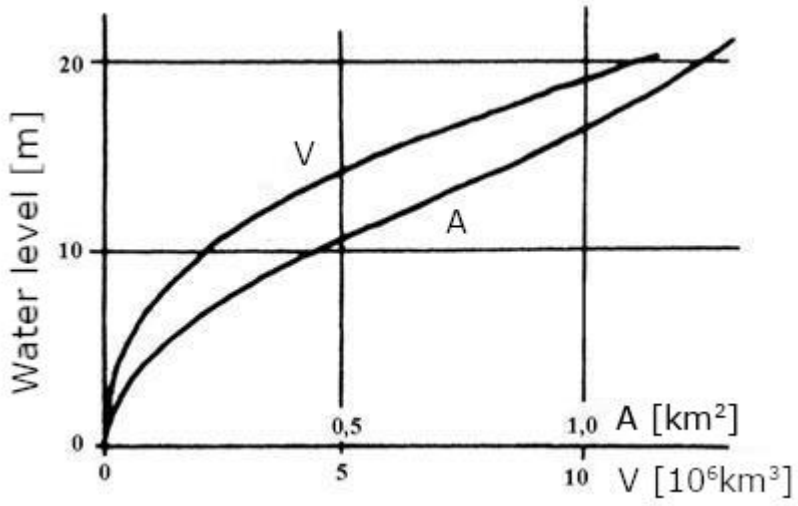
The derivative function became even smoother, resulting in a line. In the case of time t=30 s, the speed obtained by the central difference formula (v=206.65 m/s) was the closest to the speed obtained from the derivation of the polynomial fitting (v=192.93 m/s).

Example - Evaporation and leakage losses of irrigation water reservoir

Let's look at the following example. We have a water storage used for irrigation that suffers losses due to evaporation and leakage. Our task is to approximate the quantities of these losses ($Q_{\text{loss}}(t)\ [m^3/s]$) as a function of time if we know the value of the inlet ($Q_{\text{in}}(t)\ [m^3/s]$) the value of the outlet (the water used for irrigation, $Q_{\text{out}}(t)\ [m^3/s]$), the level of water inside the reservoir ($z(t)$) and the characteristic curve of the reservoir. The inlet, the outlet and the water level are all given as time series and are hence functions of time. The equation determining the volume of water inside the reservoir:

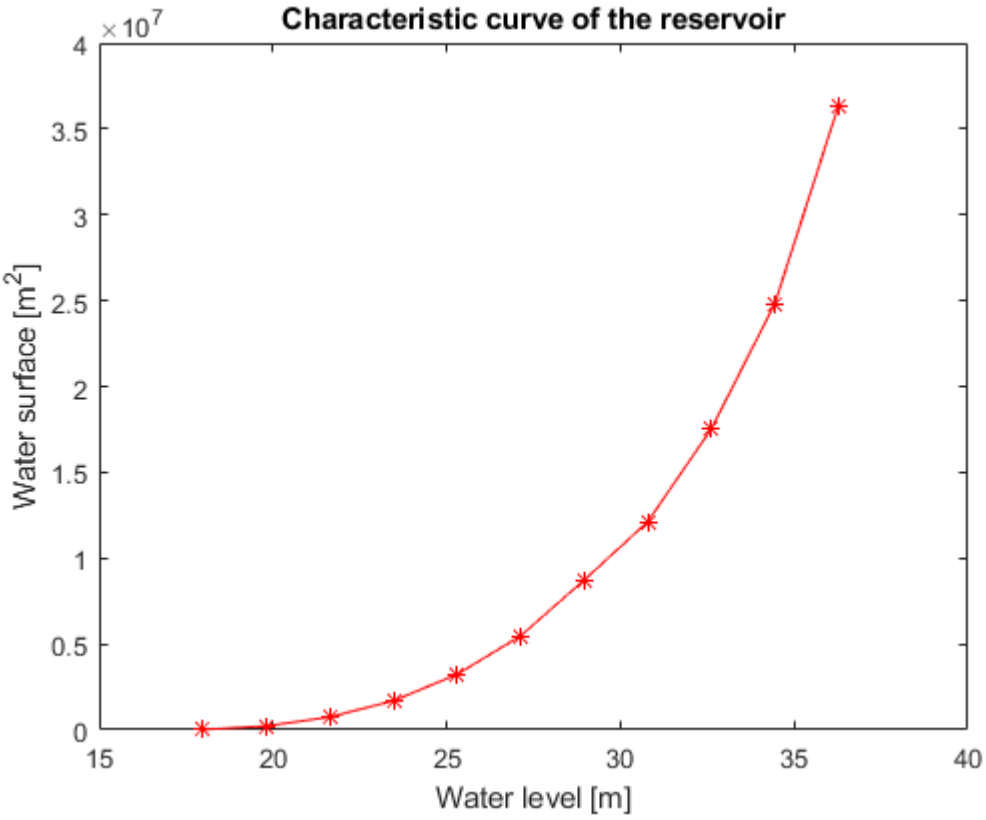
$$Q_{\text{in}}(t) - Q_{\text{out}}(t) - Q_{\text{loss}}(t) = A(z) \cdot \frac{dz}{dt}$$

where $\frac{dz}{dt}$ is the change in the water level and $A(z)$ is the area of the water inside the reservoir for a given water level (it can be determined from the characteristic curve of the reservoir):



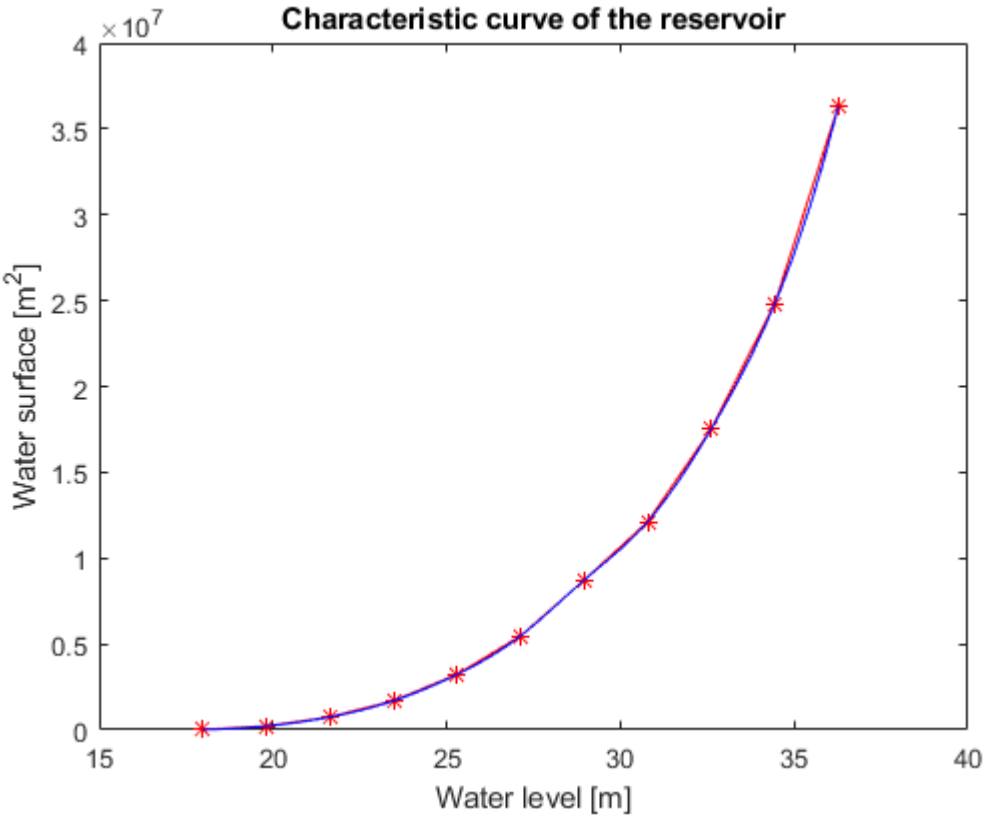
The figure above shows the characteristic curve of a reservoir for both water surface and the volume as function of the water level. The data for the $A(z)$ function for our reservoir is available in the **`irrigation.txt`** file. The water levels are given in m and the water surface values are given in m^2. Let's load the data and plot the characteristic curve.

```
clear all; close all;
data = load('irrigation.txt');
z = data(:, 1);
A = data(:, 2);
figure(1);
plot(z, A, 'r*-');
title('Characteristic curve of the reservoir');
xlabel('Water level [m]');
ylabel('Water surface [m^2]');
```



In the example, we will use cubic, second order splines to interpolate the data.

```
F = @(x) spline(z, A, x);
hold on;
fplot(F, [min(z), max(z)], 'b')
```

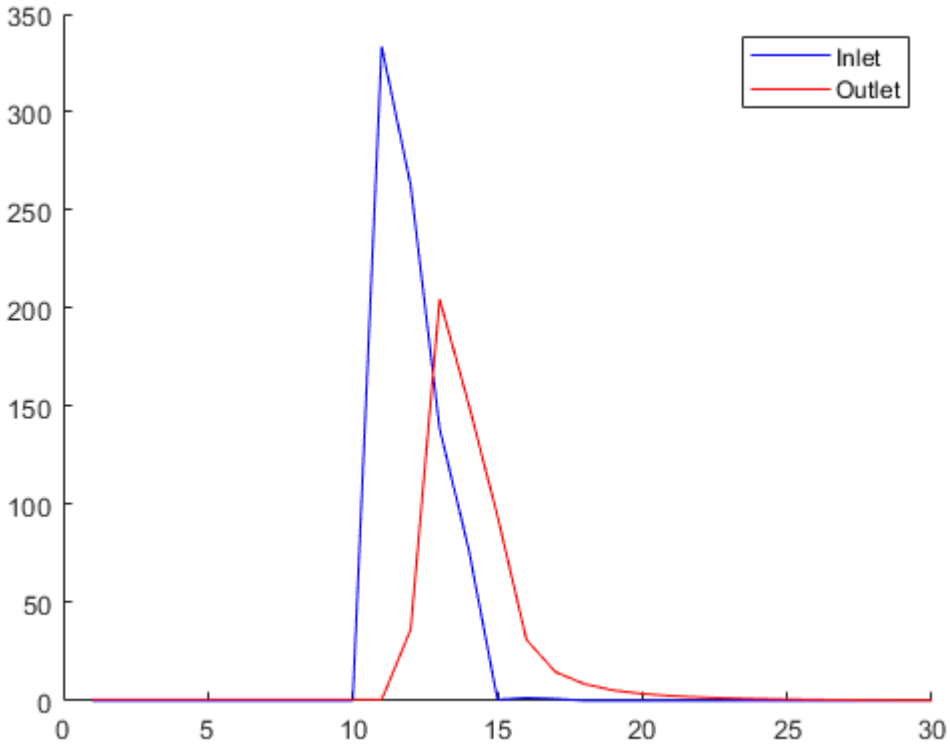


Data for the inlet ($Q_{\text{in}}(t)$), the water level ($z(t)$) and the outlet ($Q_{\text{out}}(t)$) are also available for a 30-day period in the **`timeseries.txt`**. The first column represents the number of the days, the second the inlet, the third the water level and the fourth the outlet. The water level is given in m, while the inlet and outlet values are given in m^3/s. Let's load this data into MATLAB as well:

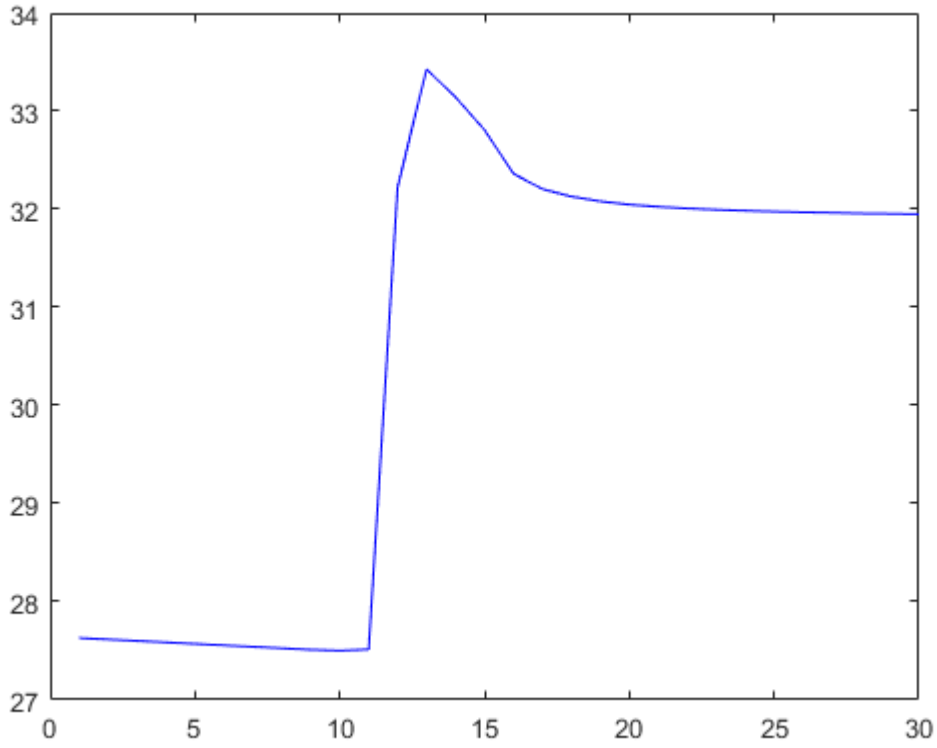
```
data = load('timeseries.txt');
t = data(:, 1);
Qin = data(:, 2);
z = data(:, 3);
Qout = data(:, 4);
```

We can create two plots showing the inlet/outlet and the change in the water level:

```
figure(2); hold on;
plot(t, Qin, 'b');
plot(t, Qout, 'r');
legend('Inlet', 'Outlet');
```




```
figure(3);
plot(t, z, 'b');
```



Using the data of the characteristic curve, let's compute the water surface values (A) at each water level value (z):

```
Az = F(z)
```

```
Az = 30×1
10^7 ×
    0.6312
    0.6288
    0.6261
    0.6235
    0.6209
    0.6183
    0.6157
    0.6131
    0.6105
    0.6090
     ⋮
```

Our equation for the reservoir is the following:

$$Q_{\text{in}}(t) - Q_{\text{out}}(t) - Q_{\text{loss}}(t) = A(z) \cdot \frac{dz}{dt}$$

We know the $Q_{\text{in}}(t)$, $Q_{\text{out}}(t)$ and $A(z)$ time series. In order to find the $Q_{\text{loss}}(t)$ function, we have to compute the first derivative of the water level, $\frac{dz}{dt}$. Because the water level values are measured at the beginning of each day, the effect of change in the water level during a day will only visible in the data for the next day. This means that we have to use forward differences. We can create a new custom function that calculates the derivative using only forward differences (one backward difference at the very last point), or we can use the **`diff`** command.

First, we convert the time values from day to second, as all the other values are given in $[\text{m}^3/s]$:

```
ts = t*24*3600;
dz = diff(z)./diff(ts);
dz = [dz; dz(end)] % derivative using forward differences (the last value is copied)
```

```
dz = 30×1
10^-4 ×
   -0.0016
   -0.0017
   -0.0017
   -0.0017
   -0.0017
   -0.0017
   -0.0017
   -0.0017
   -0.0010
    0.0014
     ⋮
```

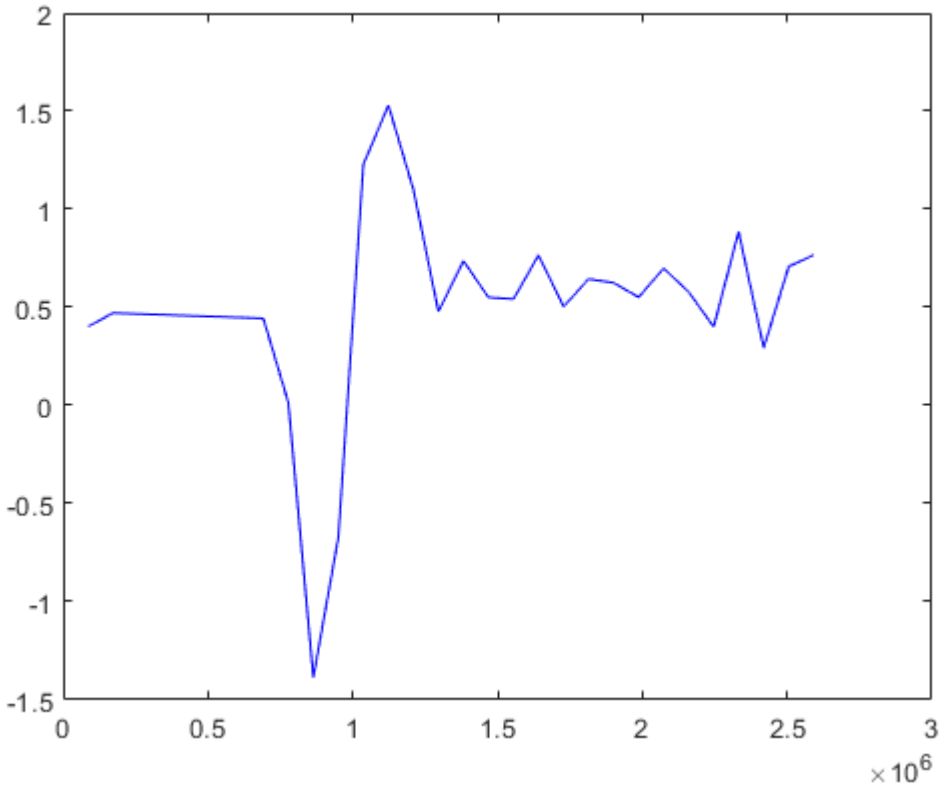
Now, we can use the equation above to find the loss function:

```
dV = Az.*dz;
```

```
Qloss = Qin - Qout - dV;
```

Plotting the loss function:

```
figure(4);
plot(ts, Qloss, 'b-');
```



The evaporation (h_v) is proportional to the water surface and is usually given in mm/day. The values of the loss function are now in m^3/s, but we can change this to mm/day:

$$h_v = \frac{Q_{\text{loss}} \cdot 86400 \cdot 1000}{A}$$

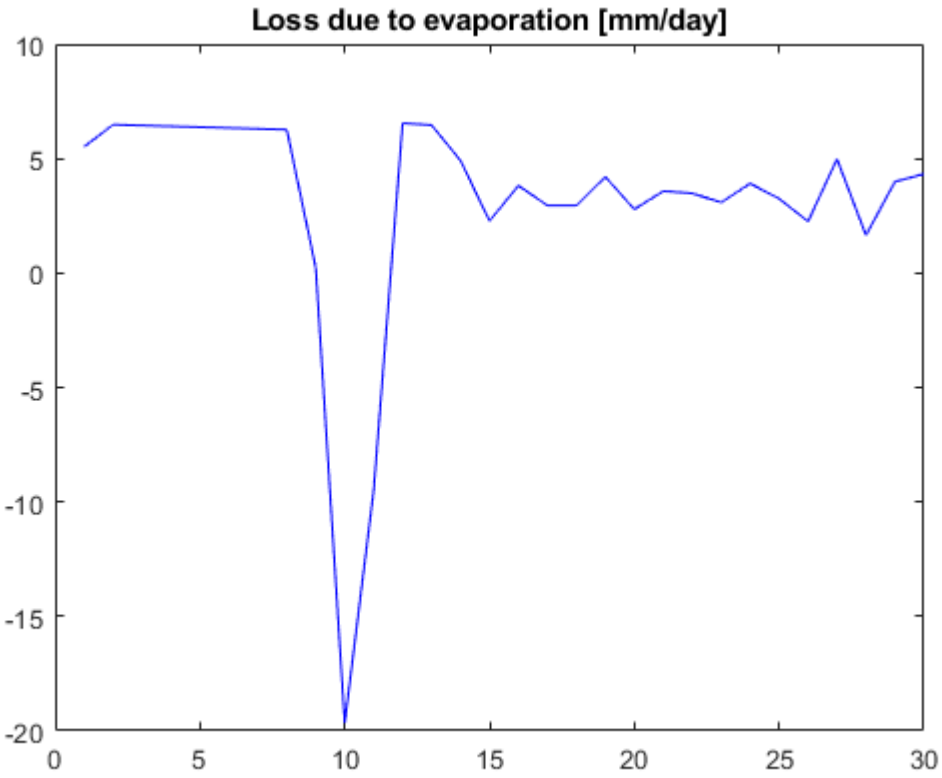
where 86400 is the number of seconds in a day and we multiply by 1000 to give the results in mm and not in m. Let's plot the transformed evaporation function (it's value is generally under 10 mm/day for our conditions and its value can be negative in case of precipitation):

```
evap_mm = Qloss * 86400 * 1000 ./ Az
```

```
evap_mm = 30×1
    5.5137
    6.4805
    6.4448
    6.4088
    6.3726
    6.3362
    6.2996
    6.2628
    0.2258
```

```
   -19.6615
      ⋮
```

```
figure(5);
plot(t, evap_mm, 'b');
title('Loss due to evaporation [mm/day]');
```



Derivation in bivariate case

The gradient is a generalization of the derivative for the multivariate case. The value of the derivative of a function can be calculated in the one-variable case, its value will be a scalar. In the case of two variables, the gradient takes its place, which, unlike the derivative, returns a vector, not a scalar, resulting in a vector field. In the bivariate case, the gradient vector of a function f is:

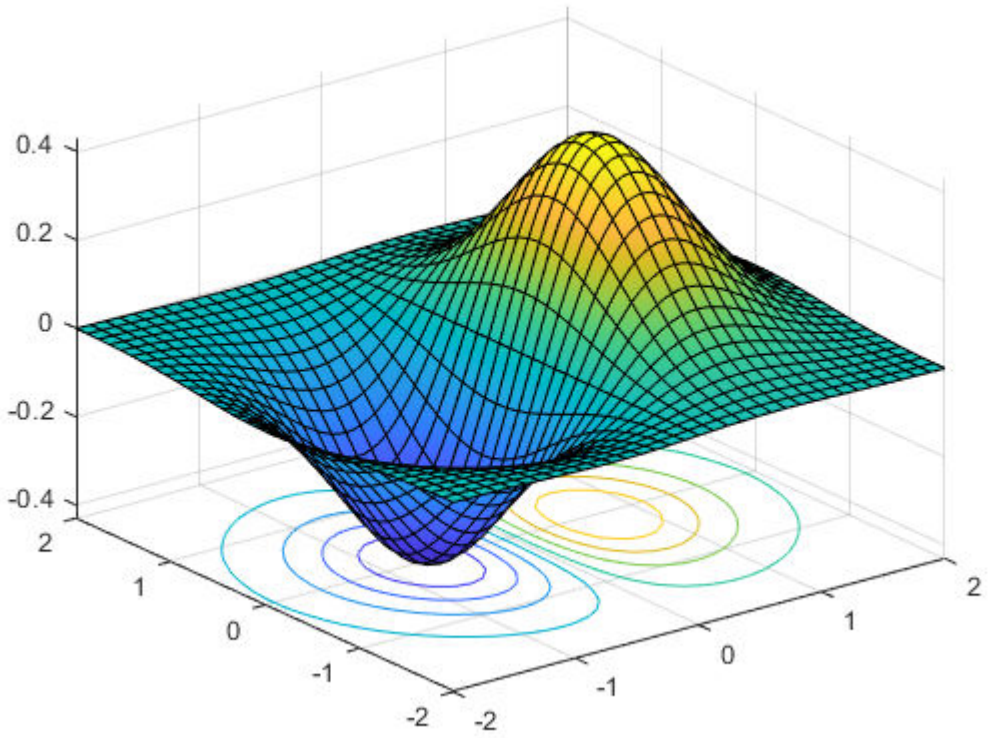
$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

In the same way that the derivative gives the slope of the tangent, the gradient points in the direction of the greatest slope of the surface, it gives the direction and value of the greatest change of the function at the given point.

Determine the gradient of the following two-variable function and plot the vector field on the range x∈[-2,2] and y∈[-2,2]!

$$f(x,y) = x\, e^{-(x^2+y^2)}$$

```
%% Defining a two-variable surface
clc; clear all; close all;
f = @(x,y) x .* exp(-(x.^2 + y.^2));
figure(1)
fsurf(f,[-2,2,-2,2],'ShowContours','on')
```




```
figure()
h = ezcontour(f,[-2,2,-2,2])
```

```
h =
  Contour with properties:

    LineColor: 'flat'
    LineStyle: '-'
    LineWidth: 0.5000
         Fill: off
    LevelList: [-0.4000 -0.3000 -0.2000 -0.1000 0 0.1000 0.2000 0.3000 0.4000]
        XData: [60×60 double]
        YData: [60×60 double]
        ZData: [60×60 double]

  Show all properties
```

```
set(h,'ShowText','on')
```

If we also want to label the contour lines, fcontour cannot be used in Matlab, only ezcontour can be used for this.

Let's calculate the elements of the gradient vector, i.e. the partial derivatives with respect to x and y symbolically! For this, we could use the diff command symbolically, or the gradient command in one step.

```
%% Gradient calculation symbolically
syms x y
fs = x .* exp(-(x.^2 + y.^2));
G = gradient(fs, [x, y])
```

```
G =
```

$$\begin{pmatrix} \mathrm{e}^{-x^2-y^2} - 2\,x^2\,\mathrm{e}^{-x^2-y^2} \\ -2\,x\,y\,\mathrm{e}^{-x^2-y^2} \end{pmatrix}$$

Let's represent the vector field in the contour map by calculating the values of the gradient in the points of a grid. A vector field can be displayed with the quiver command. The first two input parameters of the quiver command are the coordinates (x,y) of the starting points of the vectors, and the second two parameters are the magnitude of the vectors in the x,y direction.

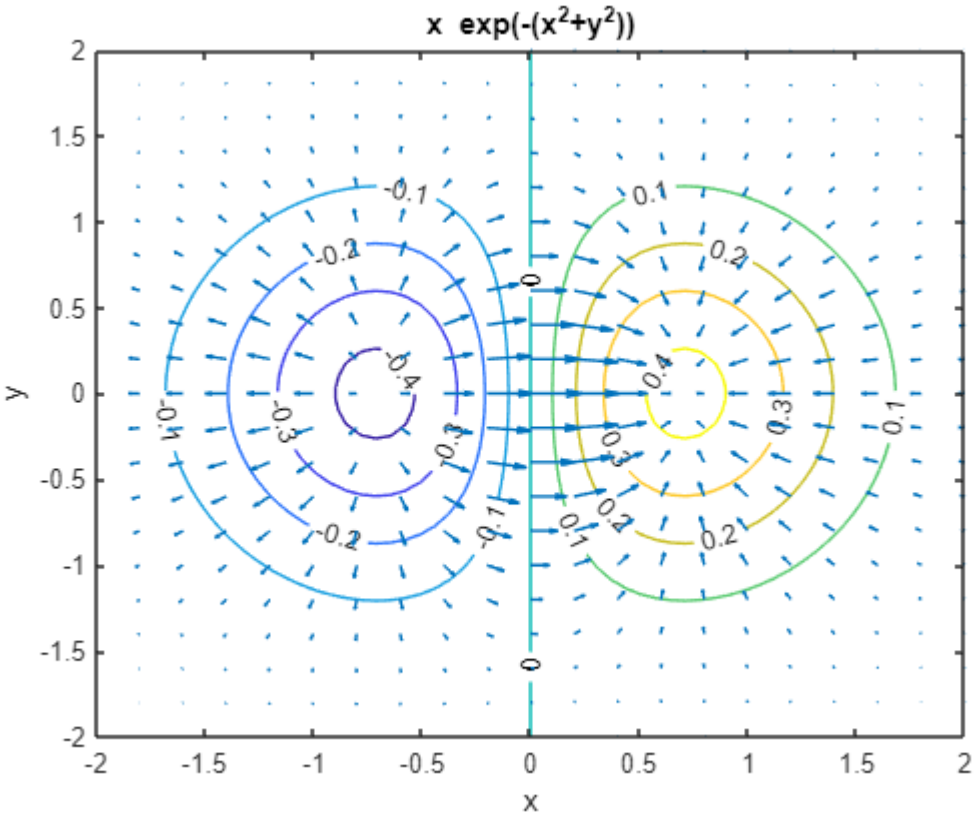
```
[X, Y] = meshgrid(-2:0.2:2,-2:0.2:2); % grid points
gx = matlabFunction(G(1)) % gradient in the x direction as a function
```

```
gx = function_handle with value:
    @(x,y)exp(-x.^2-y.^2)-x.^2.*exp(-x.^2-y.^2).*2.0
```

```
gy = matlabFunction(G(2)) % gradient in the y direction as a function
```

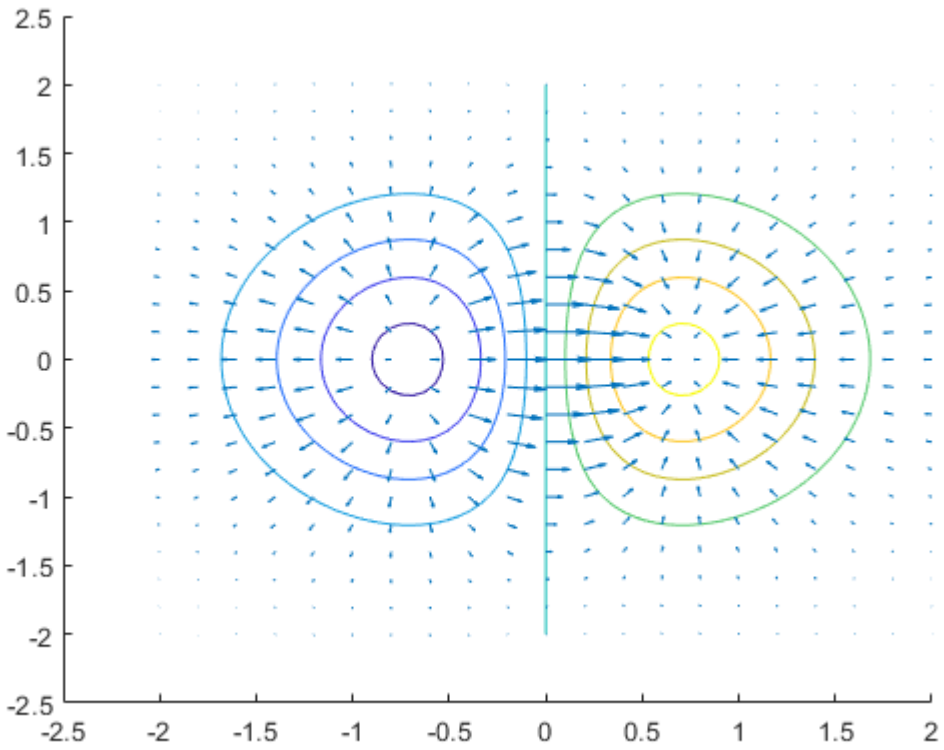
```
gy = function_handle with value:
    @(x,y)x.*y.*exp(-x.^2-y.^2).*-2.0
```

```
GX = gx(X,Y); GY = gy(X,Y); % magnitude of gradient vectors at the grid points
hold on;
quiver(X,Y,GX,GY)
```



The gradient command can also be used numerically to calculate the values of the gradient vector at grid points.

```
%% Gradient calculation numerically
Z = f(X,Y); % Z values at gridpoints
[px,py] = gradient(Z); % gradients at gridpoints
figure(3); hold on;
fcontour(f,[-2,2,-2,2])
quiver(X,Y,px,py)
```



Here, we first calculated the value of the function (Z) in the grid points X,Y, and then, assuming the division of the grid mesh to be even, the values of the gradient vector (px,py). The resulting figure is the same as the previous one.

The second partial derivatives are contained in the Hesse matrix, which in the two-variable case can be written as follows:

$$H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x\,\partial y} \\ \frac{\partial^2 f}{\partial y\,\partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

The second partial derivative of a function f with respect to the variable x can be calculated with the diff(f,x,2) command, or the entire Hessian matrix can also be entered symbolically using the hessian command. Determine the Hesse matrix of the previous function and calculate the value of the second derivatives at x=0.5 and y = 0.7!

```
% second partial derivative of a function f with respect to the variable x
d2x = diff(fs,x,2)
```

d2x = $4\,x^3\,e^{-x^2-y^2} - 6\,x\,e^{-x^2-y^2}$

```
% Hessian mátrix
```

```
H = hessian(fs)
```

H =

$$\begin{pmatrix} 4x^3\sigma_1 - 6x\sigma_1 & 4x^2y\sigma_1 - 2y\sigma_1 \\ 4x^2y\sigma_1 - 2y\sigma_1 & 4xy^2\sigma_1 - 2x\sigma_1 \end{pmatrix}$$

where

$$\sigma_1 = e^{-x^2-y^2}$$

```
Hfv = matlabFunction(H)
```

```
Hfv = function_handle with value:
    @(x,y)reshape([x.*exp(-x.^2-y.^2).*-6.0+x.^3.*exp(-x.^2-y.^2).*4.0,y.*exp(-x.^2-y.^2).*-2.0+x.^2.*y.*exp(-x.^2-y
```

```
Hfv(0.5,0.7)
```

```
ans = 2×2
   -1.1928   -0.3340
   -0.3340   -0.0095
```