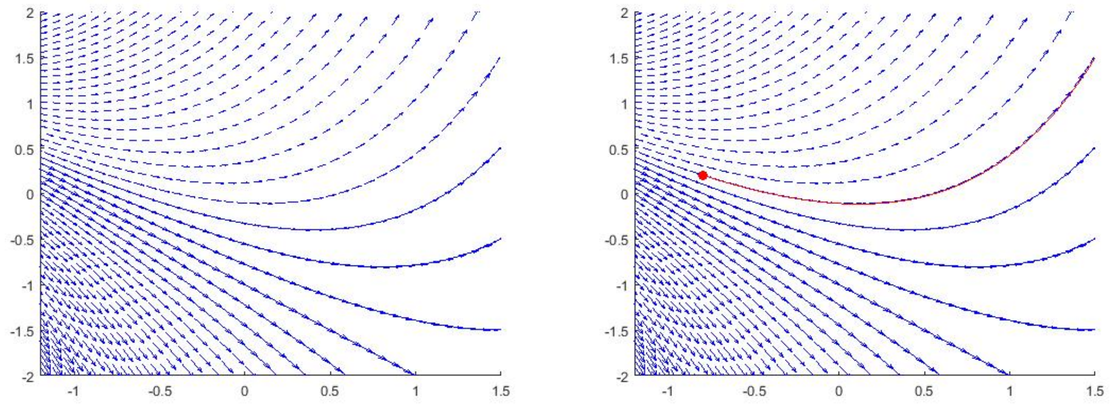
DIFFERENTIAL EQUATIONS

Differential equations are equations containing derivatives of the unknown function. The solution of the equation is not a single value but a function that satisfies the system of differential equations. For ordinary differential equations this is a function of one variable.

For a unique solution one (or more) points of the solution function have to be given.

Let us consider a simple example. The differential equation $\frac{dy}{dx} = y + x$ is given and we seek functions that satisfy this condition. On the left subfigure below all functions shown satisfy this condition. This is the phase portrait of the differential equation. If we seek a particular solution which passes through point $x = -0.8, y = 0.2$ then this solution is unique (plotted with red color on the right subfigure below).



In initial value problems the value of the function and its derivative(s) at an initial point are known and these are used to find the solution. In boundary value problems at least one value (of the function and its derivatives) is given not at the initial point but at the endpoint. This complicates our problem because also the particular initial value must be determined that yields the specified values at the endpoint.

In case the differential equation contains only linear expressions of the function or its derivatives it is called linear. For example:

$$\frac{dy}{dx} + a \cdot x^2 + b \cdot y = 0 \text{ – linear differential equation}$$

$$\frac{dy}{dx} + a \cdot x \cdot y + b \cdot y^2 = 0 \text{ – nonlinear differential equation}$$

The equation is of order n when the highest derivative of the unkonwn function is of order n. In many cases, especially for nonlinear equations the unknown function can only be determined numerically. When this is the case we have no analytical solution instead function values at distinct points are determined by numerical quadrature. Our goal is to use numerical procedures that give points of the solution by the least number of steps and smallest error with a prescribed local error bound.

FIRST-ORDER ORDINARY DIFFERENTIAL EQUATION – INITIAL VALUE PROBLEM

General form of a first-order differential equation (where t denotes independent variable) is:

$$y' = \frac{dy}{dt} = f(t, y)$$

In univariate case there is only one independent variable, t and one dependent variable, y. The function $f(t, y)$ defines the first derivative. In initial value problems it is a known initial condition that solution passes through point (t_0, y_0):

$$y(t_0) = y_0$$

EULER'S METHOD

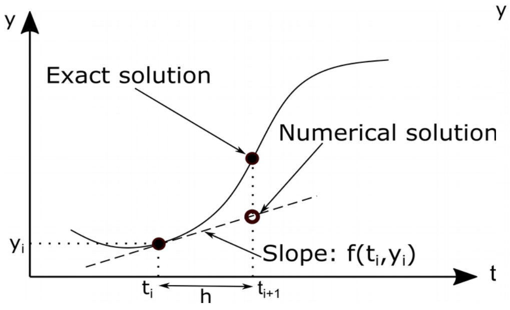
We would like to determine function values of the solution in a specified interval with prescribed step size (h). Slope (m) of the function is specified as constant within each interval h. If function value at the start of an interval and slope are known then we can approximate function value at the endpoint by a line with slope m starting at the known point.

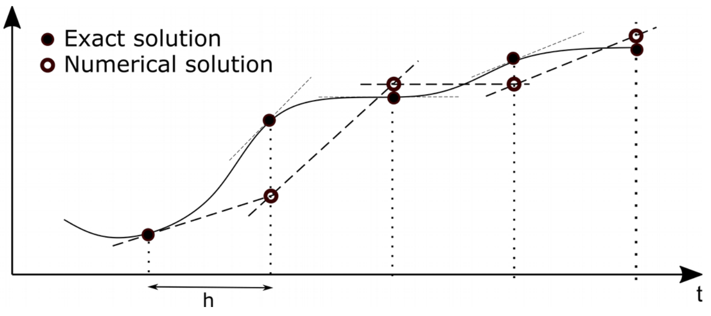
In Euler's method it is assumed that $m = f(t, y)$ is constant within each integration interval ($h = t_{i+1} - t_i$) and it is the same as evaluated at the beginning of the interval.

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y) \cdot h \approx y_i + f(t_i, y_i) \cdot h = y_i + m_i \cdot h$$

$$t_{i+1} = t_i + h$$

where m is slope evaluated at the start of the interval considered constant within the interval. Local error of the method is O(h^2), global error is O(h), i.e. this is a first order method.





.

```
>  function [t,y] = euler (f, y0, a, b, h)
>  n = round((b - a)/h);
>  t(1) = a;
>  y(1) = y0;
>  for i = 1 : n - 1
>      y(i + 1) = y(i) + h*f(t(i), y(i));
>      t(i + 1) = t(i) + h;
>  end
```

SOLUTION OF FIRST-ORDER DIFFERENTIAL EQUATIONS WITH EULER'S METHOD

Let us consider the following problem. Water (approx. 4000 m^3) is discharged at a bottom circular orifice with radius r=5 cm at height h=0 from a water tower with a spherical container of radius R=10 m. At the start of discharge ($t=0$) water level in the container is 17.44 m. Coefficient of contraction of the orifice is $\mu=0.85$. Determine water level in the container after 12 hours.

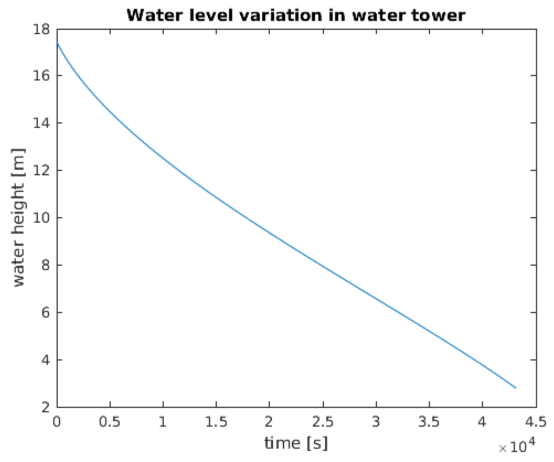
Instantaneous water level of the water tower (measured from bottom up) is described by the following first-order ordinary differential equation:

$$\frac{dh}{dt}=\frac{-\mu r^2\sqrt{2gh}}{2hR-h^2}$$

where $R=10\,m, r=0.05\,m, g=9.81\frac{m}{s^2}, \mu=0.85$.

Specify this problem in Matlab and solve it by Euler's method.

```
>  R = 10; r = 0.05; g = 9.81; mu = 0.85;
>  % Specify derivative function
>  f = @(t,h) -mu*r^2*sqrt(2*g*h)/(2*h*R-h.^2)
>  %dhdt = @(t,h) -sqrt(2*g*h)/(alfa*h*(2*R-h));
>  % Specify initial value, domain and step size
>  h0 = 17.44; t0 = 0; tv = 12*3600 % 12 hours = 43200 s
>  % step size 60 s
>  d = 60;
>
>  % solution by Euler's method
>  [T, H] = euler(f, h0, t0, tv, d);
>  figure(2)
>  plot(T,H);
>  xlabel('time [s]')
>  ylabel('water height [m]')
>  title('Water level variation in
   water tower')
```



Last element of vector H gives water level after 12 hours:

```
>  H(end) % 2.7712 m
```

Euler's method is a first order one with error O(*h*). Let us consider other methods which are more accurate.

IMPROVEMENTS ON EULER'S METHOD (HEUN, MIDPOINT, RUNGE-KUTTA METHODS)

Estimation of function values are the same for Euler's, Heun, Midpoint and Runge-Kutta methods, the only difference is in the calculation of slope. In Euler's method derivatives are computed at the starting point of each interval and these are used as slopes (see figures above).

Within **Heun's method** slope is the average of the two slopes evaluated at starting (m_i) and endpoints (m_{i+1}) of each interval. To calculate endpoint slope, however, endpoint function values are required since $m_{i+1}=f\left(t_{i+1}, y_{i+1}\right)$. Hence first in a so called predictor step an approximate endpoint function value is computed by Euler's method and this is used for calculating slope. Average of both slopes facilitates computation of endpoint function value.

1) Predictor step (Euler's method): $(y_{i+1})^{(0)} = y_i + m_i \cdot h = y_i + f(t_i, y_i) \cdot h$,
2) Corrector step: $t_{i+1} = t_i + h$, $m_{i+1} = f\left(t_{i+1}, (y_{i+1})^{(0)}\right)$

$$y_{i+1} = y_i + \frac{(m_i + m_{i+1})}{2} \cdot h$$

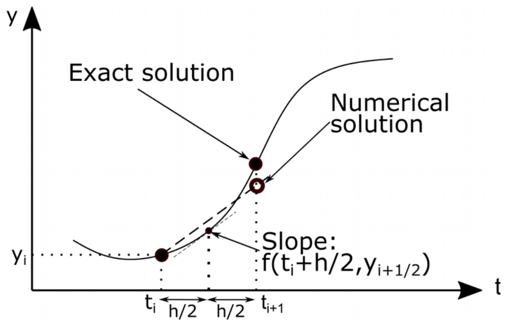
Local error of the method is O(h^3) and global error is O(h^2), i.e. this is a second order method, an order of magnitude more precise than Euler's method.

In case of **midpoint method** derivatives are computed at the midpoint and considered as constant slope for the whole interval. It is required first to evaluate a preliminary function value at the midpoint by Euler's method and then slope can be evaluated there.

1) Function value at the midpoint (Euler's method): $y_{i+\frac{1}{2}} = y_i + m_i \cdot \frac{h}{2} = y_i + f(t_i, y_i) \cdot \frac{h}{2}$,
2) Slope at the midpoint: $t_{i+\frac{1}{2}} = t_i + \frac{h}{2}$, $m_{i+\frac{1}{2}} = f\left(t_{i+\frac{1}{2}}, y_{i+\frac{1}{2}}\right)$

   Function value at the endpoint: $y_{i+1} = y_i + m_{i+\frac{1}{2}} \cdot h$

Local error term of the method is O(h^3) and global error term is O(h^2), i.e. similar to Heun's method this is also an order of magnitude more precise than Euler's method.



Precision of Euler's method can further be increased by evaluating derivatives at additional points and take their weighted average to get a slope that is considered constant. This is the most widely used **Runge-Kutta method** with a fourth order error term that has global truncation error of O(h^4). In Matlab it is implemented by the built-in function **ode45**.

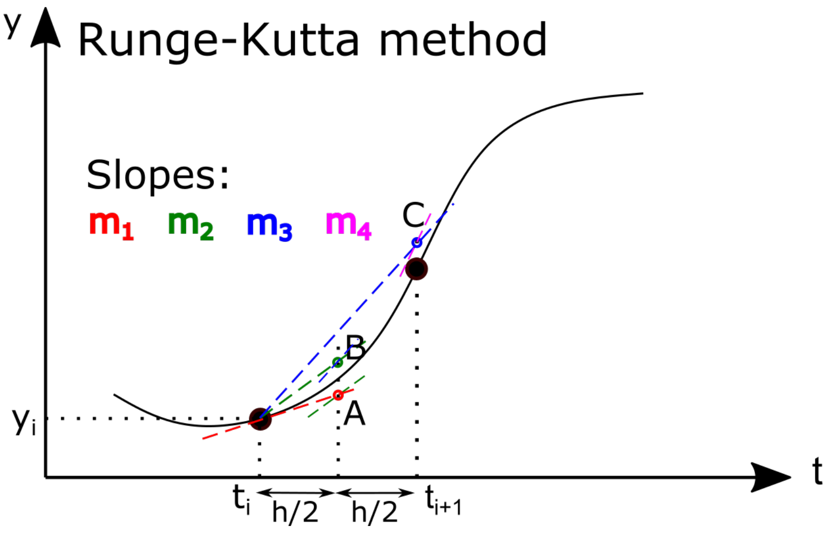
$$y_{i+1} = y_i + \frac{1}{6} \cdot (m_1 + 2m_2 + 2m_3 + m_4) \cdot h$$

$m_1 = f(t_i, y_i)$ - slope at the starting point → compute point A

$m_2 = f\left(t_i + \frac{h}{2}, y_i + m_1 \cdot \frac{h}{2}\right)$ - slope at point A → compute point B

$m_3 = f\left(t_i + \frac{h}{2}, y_i + m_2 \cdot \frac{h}{2}\right)$ - slope at point B → compute point C

$m_4 = f(t_i + h, y_i + m_3 \cdot h)$ - slope at point C



SOLUTION OF FIRST ORDER DIFFERENTIAL EQUATIONS BY RUNGE-KUTTA METHOD

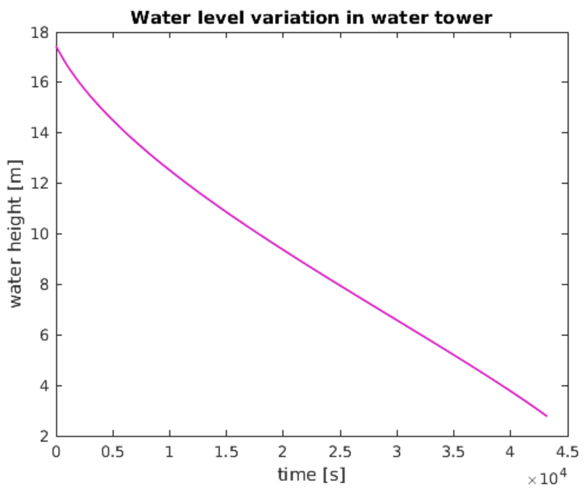
Let us solve the water tower problem above with Runge-Kutta method as well. Use Matlab's built-in **ode45** function.

Most simply it can be called like this:

```
>       [TOUT,YOUT] = ode45(ODEFUN,TSPAN,Y0)
```

where ODEFUN is a reference to function $y'=f(t,y)$, TSPAN is either only an interval [T0 TV] or a vector covering the computation interval with a given step size and Y0 is initial value of the function y.

```
>  % Solution by Runge-Kutta method
>  [T1, H1] = ode45(f, [0,43200], h0);
>  H1(end) % 2.7779 m
>  % or with given step size
>  [T2, H2] = ode45(f, 0:60:43200, h0);
>  H2(end) % 2.7713 m
>  hold on;
>  plot(T1,H1,'r')
>  plot(T2,H2,'m')
```



In this problem there are no perceptible differences between results of the methods used, only a couple of mm discrepancies in water levels are seen. It is interesting to see what the step sizes are in case only the start and endpoints were given:

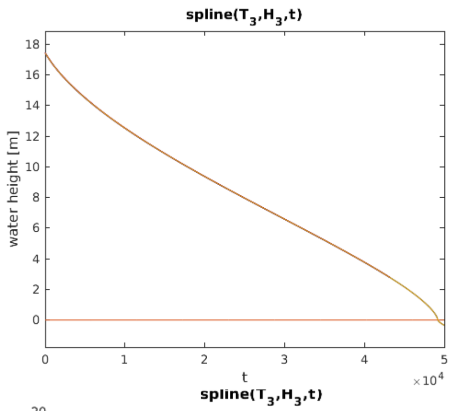
```
>  min(diff(T1)) % 995.1333
>  max(diff(T1)) % 1.1649e+03
```

We see step size variation between 995 and 1165 seconds. Smaller step sizes generally increase precision of results but also increase computation time.

<u>Additional tasks:</u>

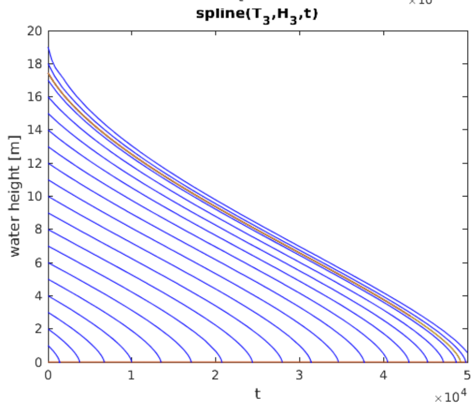
Let us evaluate discharge time in hours required to empty the container. As a first step compute discharge for a longer time, say for 50000 seconds instead of 43200 seconds as before. Be aware that for negative *h* complex numbers will be yielded for solution. Solution of the problem can be accomplished by spline fitting and total discharge time is computed by zero finding.

```
>  % With given step size
>  [T3, H3] = ode45(f, 0:60:50000, h0)
>  plot(T3,H3,'k')
>  ezplot('0',[0 50000])
>  % discard complex part
>  H3 = real(H3);
>  % spline fitting
>  sp =@(t) spline(T3,H3,t)
>  ezplot(sp,[0 50000])
>  % Compute time in hours for zero water
   level
>  x0 = fzero(sp,49000) % 4.9192e+04 s
>  x0 = x0/3600 % 13.6644 h
```



How does discharge time vary with initial water height? Let us plot trajectories or phase portrait from 20 meters initial water level down to a level of 1 meter.

```
>  % Trajectories or phase portrait
>  for i=20:-1:1
>      [T, H] = ode45(f, [0,50000], i);
>      plot(T,H,'b')
>  end
>  axis([0 50000 0 20])
```



SOLUTION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL EQUATIONS

In many problems the process depends on several variables that may be interrelated. In this case we must solve a system of differential equations instead of a single differential equation.

$$\frac{dy_1}{dt}=f_1(t, y_1, y_2, y_3 \ldots, y_n)$$

$$\frac{dy_2}{dt}=f_2(t, y_1, y_2, y_3 \ldots, y_n)$$

$$\ldots$$

$$\frac{dy_n}{dt}=f_n(t, y_1, y_2, y_3 \ldots, y_n)$$

Initial values for interval [a,b]:

$$y_1(a)=Y_1, y_2(a)=Y_2, \cdots, y_n(a)=Y_n,$$

A subset of such equations can be solved by generalizing the explicit methods we studied before: $t_{i+1}=t_i+h$

$$y_{1,i+1}=y_i+m_{1,i}\cdot h$$

$$\dots$$

$$y_{n,i+1}=y_i+m_{n,i}\cdot h$$

Slopes $m_1 \dots m_n$ can be computed using the procedures we have discussed. For Euler's method, for example, these are equal to derivatives computed at the starting point of each interval:

$$y_{1,i+1}=y_i+f_1(t, y_1, y_2, y_3 \dots, y_n)\cdot h$$

$$\dots$$

$$y_{n,i+1}=y_i+f_n(t, y_1, y_2, y_3 \dots, y_n)\cdot h$$

Improved Euler's methods and Runge-Kutta method may likewise be generalized. Let us consider a simple example:

$$f_1=\frac{dx}{dt}=x\,t-y\,;\,x(0)=1$$

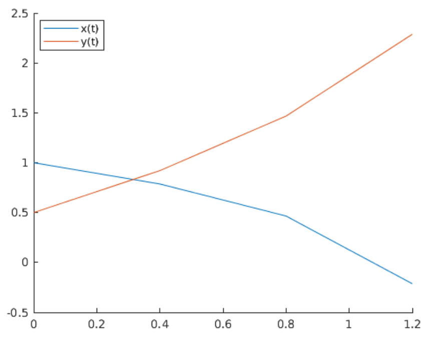
$$f_2=\frac{dy}{dt}=y\,t+x\,;\,y(0)=0.5$$

Let us solve this problem by using Matlab's built-in Runge-Kutta method. In case of more than one variable it is advisable to specify the system of differential equations of the system in a separate file. Instead of x,y variables it is required to use a vector variable in order to call Matlab's built-in functions. Put the system of first-order differential equations into a separate file named **diffeqs.m**. Let us create a vector y containing the two variables: y=[x, y],, i.e. $y_1=x\,,\,y_2=y$.

```
>  function dydt = diffeqs(t,y)
>      f1 = y(1)*t - y(2);
>      f2 = y(2)*t + y(1);
>      dydt = [f1; f2];
>  end
```

Let us solve this system by using Runge-Kutta method **ode45** from time $t=0$ to time $t=1.2$ with step size $h=0.4$. Now two initial values are needed, one for *x* and the other for *y*: $x(0)=1\,;\,y(0)=0.5$.

```
>  t = 0:0.4:1.2;
>  x0 = 1; y0 = 0.5;
>  [T, Y] = ode45(@diffeqs, t, [x0; y0])
>  figure(1);hold on;
>  plot(T,Y(:,1)); plot(T,Y(:,2));
>  legend('x(t)','y(t)','Location','best')
```



Please note that when our system of differential equations is given in a separate *.m file we need to prepend a @ sign before the name of the function. When our system of equations is not too complicated then instead of using a separate file we may use one-line functions in the following manner:

```
>  dydt = @(t,y) [y(1)*t - y(2); y(2)*t + y(1)]
>  [T2, Y2] = ode45(dydt, t, [x0; y0])
```

It may happen that our differential equation does not explicitly depend on *t*, but also in this case variable *t* must be given in the definition of the function, because otherwise Matlab's built-in procedures are unable to interpret the problem.

SECOND-ORDER DIFFERENTIAL EQUATIONS

An ordinary second-order differential equation assumes the following general form with independent variable *t* and dependent variable *y*:

$$\frac{d^2y}{dt^2}=f\left(t,y,\frac{dy}{dt}\right)$$

This equation can be solved within interval [a,b] if two conditions are given. When these conditions are specified for the start of the interval we have an initial value problem. The two initial conditions are values of y and $\frac{dy}{dt}$ at the starting point. Let us denote these with A and B.

$$y(a)=A;\ \left.\frac{dy}{dt}\right|_{t=a}=B$$

This second-order system can be re-casted into a system of two first-order differential equations that can be solved with methods studied above. To facilitate the solution let us introduce a new variable *w*.

$$w=\frac{dy}{dt}$$

then

$$\frac{dw}{dt}=\frac{d^2y}{dt^2}$$

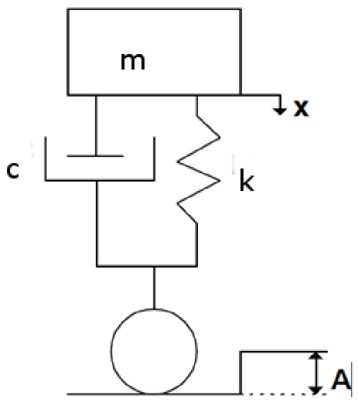
This procedure enables us to re-cast our second-order system into a system of two first-order differential equations.

$$f_1=\frac{dy}{dt}=w;\ y(a)=A$$

$$f_2=\frac{dw}{dt}=f(t,y,w);\ w(a)=B$$

Let us now consider a problem of solving such a second-order differential equation.

SOLUTION OF SECOND-ORDER DIFFERENTIAL EQUATION IN MATLAB



Let us simulate suspension of a car based on the following simple model when the car passes over an obstacle of height *A*. In this model *m* denotes mass of the car, *k* is spring stiffness (displacement of the spring is proportional with spring force), c is damping factor (damping force is

proportional with velocity of mass). Data of the problem are:
$m = 1000\,kg\,;\,k = 1000\frac{kg}{s^2}\,;\,c = 500\frac{kg}{s}\,;\,A = 0.1\,m$.

Damped free oscillations satisfy the following ordinary differential equation if we sum all forces acting on the mass:

$$m\ddot{x} + c\,\dot{x} + k\,x = 0$$

In the frame attached to the car we get equation of vertical motion:

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + k(x - A) = 0$$

Initial values:

$$x(0) = 0;\ \left.\frac{dx}{dt}\right|_{x=0} = 0$$

Let us solve our equation for $\frac{d^2x}{dt^2}$

$$\frac{d^2x}{dt^2} = \frac{1}{m}\left(k\,A - k\,x - c\frac{dx}{dt}\right)$$

Re-cast our second order differential equation into a system of first order differential equations.

Let

$$f_1 = \frac{dx}{dt} = w\ x(0) = 0\,;$$

$$f_2 = \frac{dw}{dt} = \frac{d^2x}{dt^2} = \frac{1}{m}(k\,A - k\,x - c\,w)\,;\ w(0) = 0$$

Specify our system of differential equations for Matlab in a separate file **springdiff.m**. Let y be a vector variable: $y = [x, w]$, i.e. $y(1) = x\,;\,y(2) = w$.

```
> function dydt = springdiff(t,y)
>    %  Data of the problem
>    m=1000; k=1000; A=0.1; c=500;
>    % y(1)=x; y(2)=w
>    f1 = y(2);
>    f2 = 1/m*(k*A - k*y(1) - c*y(2));
>    dydt = [f1; f2];
> end
```

Let us solve this problem by using Matlab's built-in command **ode45** that uses Runge-Kutta method. Specify 10^{-4} absolute and relative accuracy for time interval 0-15 seconds. We have not yet used options of **ode45** but we can set several such options using the **odeset()** function. Some of the most important options are:

- RelTol = scalar value of relative threshold valid for all components of y
- AbsTol= scalar or vector of absolute threshold which is valid for all solution components or applies elementwise
- MaxStep = maximum allowed step size
- InitialStep = suggested initial step size of t