Laboratory 1

Solving differential equations with nonzero initial conditions

1. Purpose of the exercise:

- learning symbolic and numerical methods of differential equations solving with MATLAB
- using Simulink to create models of differential equations
- saving received solutions

2. Theoretical introduction

2.1. Symbolic solution of differential equations – function **dsolve**

Symbolic method consists in mathematical-expressions-based calculations rather than numbers-based calculations (as for numerical methods), resulting also in mathematical expression. Using symbolic variables and function **dsolve**, one can solve ordinary differential equation of any order.

In symbolic representation of differential equation initial conditions, symbol **D** is used. **D** corresponds to first order derivative, while **D2** – to second order derivative, etc. The letter **D** denotes differentiation with respect to the independent variable, i.e. usually d/d**t**.

Function **dsolve** may also be used for solving of system of differential equations as well as initial conditions definition. Consecutive equations are separated by commas, and are followed by initial conditions, also separated by commas:

dsolve*('equation1', 'equation2', ... , 'condition1' , 'condition2');*

Remark. Support of character vectors and strings will be removed in the future – use *sym* objects in future MATLAB releases to define differential equations instead, as in *Example 1* below.

Example 1:

Solve differential equation:

$$\frac{d^2x}{dt^2} + 3\frac{dx}{dt} + 2x = 0$$

with initial conditions: $x(0) = 0$, $\frac{dx}{dt}(0) = 2$, using function **dsolve**.

To solve a problem, create m-file *solv1.m*:

```
% Current syntax
syms x y;       % define symbolic variables 'x' and 'y'
y = dsolve('D2x + 3*Dx + 2*x=0' , 'x(0)=0' , 'Dx(0)=2')     % solve equation with initial conditions

% Syntax required for future releases
syms x(t) y(t)       % define symbolic variables 'x', 'y' representing functions of the independent variable 't'
Dx = diff(x);        % assign diff(x) to Dx
D2x = diff(x,2);     % assign diff(x,2) to D2x
y(t) = dsolve(D2x + 3*Dx + 2*x == 0, x(0) == 0, Dx(0) == 2)   % solve equation with initial conditions
```

```
pretty(y);                          % print a symbolic solution formula in a semi-graphical form
t=0:0.01:9.99;                      % define time vector 't'
w=subs(y);                          % calculate solution vector 'w' by substitution of time vector 't'
plot(t,w,'r');                      % plot 'w' versus 't' in red face
xlabel('Time [s]');
ylabel('Signal amplitude');
title('Solution of the ordinary differential equation');
grid;
```

Solution of the differential equation as a mathematical expression and time graph is obtained by execution of *solv1* in MATLAB Command Window.

2.2. Numerical solution of differential equations – function **ode**

MATLAB contains some functions which solve an initial value problem of ordinary differential equation by, among others:
- Runge-Kutta low order method (function **ode23**),
- Runge-Kutta medium order method (function **ode45**).

These functions solve an initial value problem of ordinary system of equations like:

$$\frac{dx}{dt} = F(t, x), \quad x(t_0) = x_0$$

Function syntax:

[T, X] = ***ode23****('F(t, x)', [t0 tk], x0, options)*
[T, X] = ***ode45****('F(t, x)', [t0 tk], x0, options)*

Input parameters should comply with the following criteria :

- first parameter should be a string, containing defined by the user name of the function that returns a value of F(t, x),
- *[t0, tk]* – time range in which the solution is being calculated,
- *x0* – initial condition – a vector containing a solution at the starting point,
- *options* – an additional optional parameters that may be set with the help of *odeset* instruction: *options = odeset('Parameter_name_1', value_1, 'Parameter_name_2', value_2,...).*

Each row in the solution array X corresponds to a time step returned in the column vector T.

Example 2:

Solve differential equation numerically:

$$\frac{d^2x}{dt^2} + 3\frac{dx}{dt} + 2x = 0,$$ with initial conditions: $x(0) = 0 \quad \dot{x}(0) = 2$

Use **ode45** function and equation model prepared in *Simulink*. Compare received results. To solve this problem with the use of **ode** method family, we need two files:

- first is MATLAB function *function1.m* – to define the equation:

```
function xdot=function1(t,x)          % System of differential equations
xdot=zeros(2,1);
xdot(1)=x(2);
xdot(2)=(-2*x(1)-3*x(2));
```

- second is standard MATLAB script *solv2.m* – to insert input parameters, call *ode45* function and draw solution graph:

```
t0=0;
clc
disp(This function solves ordinary differential equation using ');
disp('Runge – Kutta method and gives its graph representation ');
disp(' ');disp('Equation form:');disp(' ');
disp(' x``+ 3•x`+ 2•x = 0');
x01=input('Insert x01 = ');
x02=input('Insert x02 = ');
tk=input('Give simulation time tk = ');
x0=[x01 x02];
[t,x]=ode45('function1', [t0, tk],x0)
plot(t,x,'r-.');
xlabel('Time [s]'); ylabel('Signal amplitude');
title('Solution of the ordinary differential equation ');
grid;
```

To show the final graph you should execute:
>> solv2

2.3. Numerical solution of differential equations – *Simulink*

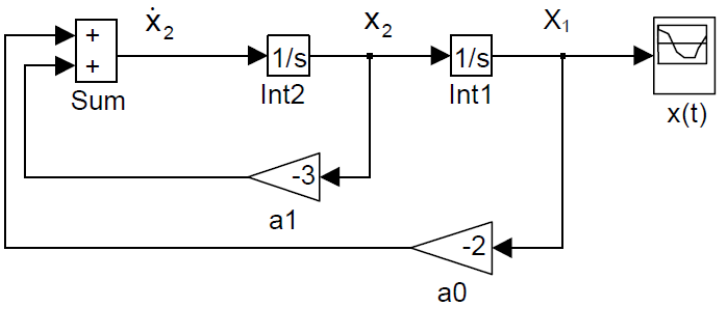
Another way of solving ordinary differential equations is by using *Simulink*. Assuming variables:

$$\begin{cases} x_1 = x \\ x_2 = \dot{x} = \dot{x}_1 \end{cases}$$

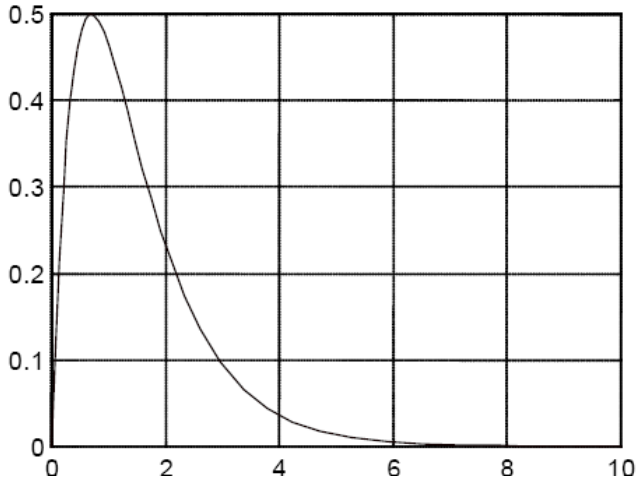
you get a system of equations:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \ddot{x} = -3\dot{x} - 2x = -3x_2 - 2x_1 \end{cases}$$

On the basis of it, we create a *Simulink* model as presented below:



After inserting initial conditions into the integrators (*Int1* and *Int2*) and choosing suitable simulation parameters, finally we get the graph:



3. Procedure of the laboratory:

3.1. Solve these differential equations:

a) $\frac{d^2y}{dt^2} + \frac{dy}{dt} + 2y = 4,$ $\quad y(0) = 0 \text{ i } \dot{y}(0) = 0$

initial conditions

b) $2\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + y = 4,$ $\quad y(0) = 1 \text{ i } \dot{y}(0) = 1$

Use **dsolve** and **ode45** functions as well as equation model prepared in *Simulink*. Draw y(t) plot received as a result of simulation. Compare results achieved with all of these methods.

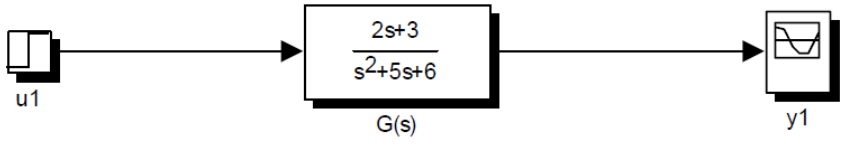
3.2. Prepare differential equation model in Simulink:

a) $\frac{d^2y}{dt^2} + 5\frac{dy}{dt} + 6y = 2\frac{du}{dt} + 3u,$ $\quad y(0) = \dot{y}(0) = 0$

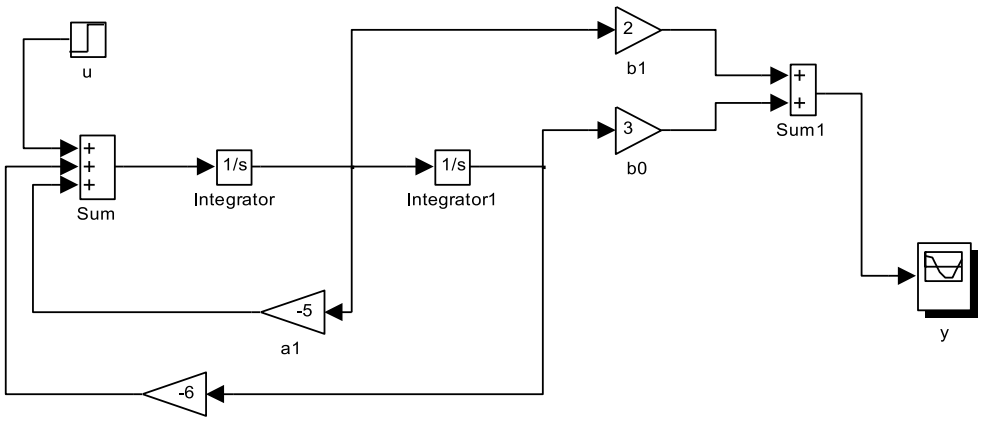
initial conditions

b) $\frac{d^2z}{dt^2} + 7\frac{dz}{dt} + 10z = \frac{du}{dt} + 3u,$ $\quad z(0) = \dot{z}(0) = 0$

Use *Transfer Fcn*, *State-Space*, and *Integrator* blocks. Register and compare so obtained step responses when amplitude of input is 1.
Equation *a)* represented by a transfer function model (*Transfer Fcn*) is shown below:



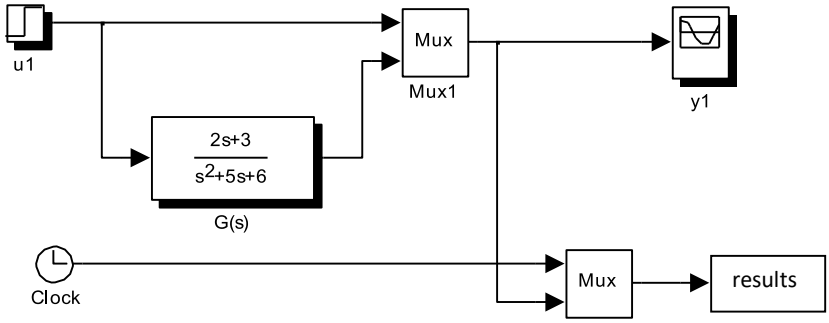
Simulink model of the equation *a)* using *Integrator* blocks is of the form:



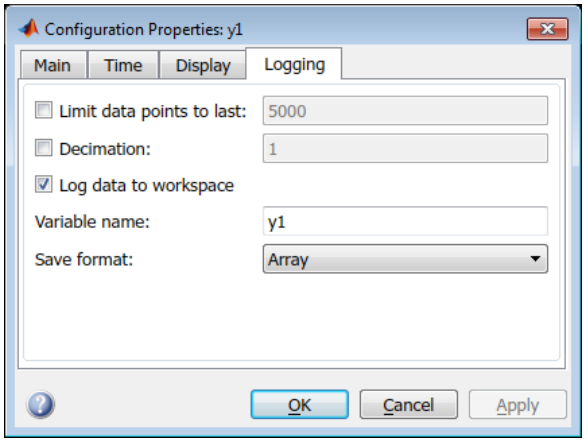
After choosing suitable simulation parameters, we get the graphical solutions that are similar to one another.

3.3. Write the results using a function **save**

Create a Simulink model:



Open *Scope* block *y1* and press its Configuration Properties icon (the one with 'cogwheel') to obtain dialogue window:



In the 'Logging' ('History' or 'Data history' in older Simulink versions) tab, mark 'Log data to workspace' checkbox and enter 'Variable name' (e.g. *y1*).

After simulation completion, two matrices *results* and *y1* will be created in MATLAB workspace: *results* containing three column vectors of variables including simulation time, input and step response, and *y1* containing two column vectors of variables including simulation time and step response. To write these matrices to the hard disk, execute the command:

>> *save filename results y1*

or:

>> *save filename* to save all the workspace. Default file format is binary MAT-file.

Use *clear* command to empty the workspace, then insert matrices *results* and *y1* into the workspace by typing:

>> *load filename*

To receive the graphical results of equation *3.2 a)* again, follow the instructions below:

```
>> t = results(:,1) % time vector
>> u = results(:,2) % forcing vector
>> y = results(:,3) % response vector
>> plot( t, u, 'r', t, y, 'g')
>> grid
```

Alternatively, using the scope data:

```
>> t = y1(:,1) % time vector
>> y = y1 (:,2) % response vector
>> plot( t, y, 'g')
>> grid
```

or simply:

```
>> plot(y1(:,1), y1(:,2)); grid
```

3.4 Solve differential equations:

a) $\frac{d^2y}{dt^2} + 4\frac{dy}{dt} + 29y = 0,$ initial conditions $y(0) = 0, \dot{y}(0) = 15$

b) $\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + 10x = 0,$ initial conditions $x(0) = 0, \dot{x}(0) = 9$

Draw y(t) plot received as a result of simulation and compare it with the solution obtained with MATLAB code.

References:

[1] G.F. Franklin, J.D. Powell, E. Emami-Naeini "Feedback control of dynamic systems", Prentice Hall, New York, 2006.

[2] K. Ogata "Modern control engineering", Prentice Hall, New York, 1997.

[3] R.H. Cannon "Dynamics of physical systems", Mc-Graw Hill, 1967 (available in Polish as: R.H. Cannon "Dynamika układów fizycznych", WNT, Warszawa, 1973).

[4] J. Kowal "Podstawy automatyki", v.1 and 2, UWND, Kraków, 2006, 2007 (in Polish).

[5] W. Pełczewski "Teoria sterowania", WNT, Warszawa, 1980 (in Polish).

[6] Brzózka J., Ćwiczenia z Automatyki w MATLABIE i Simulinku, Wydawnictwo Mikon, Warszawa 1997 (in Polish).

[7] Zalewski A., Cegieła R., MATLAB: obliczenia numeryczne i ich zastosowania, Wydawnictwo Nakom, Poznań 1996 (in Polish).

[8] MATLAB/Simulink documentation: http://www.mathworks.com/help/