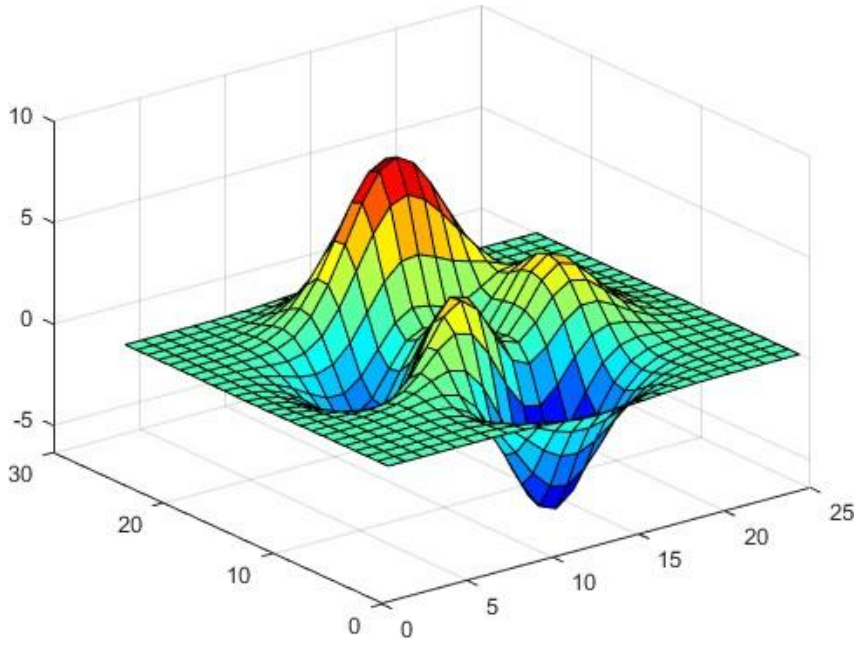
MEEM 4730 Project 1 Report

Dynamic Systems simulation using
Euler, AB-2 & RK-4 integration methods



Created & Submitted by: Vrushaketu Mali

02-Feb-2020

Table of Contents

Chapter 1. Introduction

What do we do when we face a differential equation that we can't solve? The answer depends on what we are looking for. If we are only looking for final solution we can always sketch a direction field.

However, this approach is only really good for getting general trends in solutions and for long term behavior of solutions. There are times when we will need something more. For instance, maybe we need to determine how a specific solution behaves, including some values that the solution will take.

In these cases, we rely to numerical methods that will allow us to approximate solutions to differential equations. There are many different methods that can be used to approximate solutions to a differential equation and out of those, three methods we are going to discuss in this project which are Euler integration, 2^{nd} order Adams-Bashforth Integration and 4^{th} order Runge-Kutta Integration.

Chapter 2. Summary

This project aims at creating a set of MatLab functions that can be utilized to simulate any dynamic system by either Euler, AB-2 or RK-4 integration method.

There are three different dynamic systems considered in this project and all are evaluated with all three different integration methods mentioned above. The simulation results are also verified with the Simulink solver and those are also discussed in this project.

Objective:

The objectives of the project are

A. To develop the MatLab functions for all three different integration methods described above to solve three dynamic systems mentioned below.

    1. Linear Time-invariant System

The equation of system is $\dot{x} + 4x = u$

    2. Linear Time-variant system

The equation of the system is $m\ddot{\omega} + c\dot{\omega} + k(t)\omega(t) = u(t)$

    3. Non-linear system

The equation of the system is $\ddot{\omega} - \mu\,(1 - \omega^2)\,\dot{\omega} + \omega = u$

B. To verify the codes for system function by comparing the simulation result with exact solution result
C. To verify the codes for integration function by comparing the matlab script result with Simulink auto solver results
D. To verify the effect of higher order integration method and step size on the accuracy of the simulation

Accomplishment:

All three integration equations are successfully coded in MatLab script. The simulation graphs are plotted for all three dynamic systems described above. The verification of these simulation results is also accomplished which will be found in subsequent sections.

Chapter 3. Outline and structure of the different codes

Following are different matlab codes developed for this project. The significance of all these scripts is mentioned on right hand side. Also, these scripts are discussed in detail in subsequent section.

script of function – initArrays - This is a helper function and it initializes the arrays like time vector, input vector & output vector.

script of function – u1 – This is an input function for system 1.

script of function – f1 - This is a function describing the equation for dynamic system 1.

script of function – y1 – This is an output function for system 1.

script of function – u2 – This is an input function for system 2.

script of function – f2 - This is a function describing the equation for dynamic system 2

script of function – y2 – This is an output function for system 2.

script of function – u3 - This is an input function for system 3.

script of function – f3 - This is a function describing the equation for dynamic system 3.

script of function – y3 - This is an output function for system 2.

script of function – myEuler – This is a function for Euler integration method.

script of function – myAB2 - This is a function for AB2 integration method.

script of function – myRK4 – This is a function for RK4 integration method.

script of function – simTest – This function returns the input, output and time values for all three systems along with its plots.

script– makeSolutionData – This function returns the solution data of all three systems simulated with three integration methods.

Chapter 4. Codes Description

4.1 initArrays function

```
 function [t,x,u,y] = initArrays(ss, tEnd, dt)
% This function initializes the arrays time vector, input vector & output
vector.

 % Inputs -
 % ss - ss is cell array from which we can extract either s1 or s2 or s3
 % where s1 is system 1 array, s2 & s3 for system 2 & 3 respectively.
 % s1 contains function handles f1, y1 & u1 along with initial condition of
system 1
 % s2 contains function handles f2, y2 & u2 along with initial condition of
system 2
 % s3 contains function handles f3, y3 & u3 along with initial condition of
system 3
 % tEnd - End time of the simulaton for particular system
 % dt - time step for chosen integration method

 % Outputs -
 % t - time vector as per tEnd and dt value
 % u - input vector
 % y - output vector
 % x = state vector

 % Below are the functions Called
   [n,r,m] = calcDims(ss);     % Calculate dimensions of ss cell array
   tEndSim = ceil(tEnd/dt)*dt;  % Calculate end time of the simulation
   N = tEndSim/dt + 1;     % N is the number of simulation time steps
   u = zeros(N,m);         % Initialize the input vector of N rows and m columns
   y = zeros(N,r);         % Initialize the output vector of N rows and r columns
   x = zeros(N,n);         % Initialize the state vector of N rows and n columns

   t = (0:dt:tEndSim)';  % Create the time vector from 0 to simulation end time
                         % with spacing of dt which is integration step size.
 end

 function [n,r,m] = calcDims(ss)   % Calculate the dimension of ss cell array.

   yFn = ss{2};      % second element of cell array ss defining output function
   uFn = ss{3};      % third element of cell array ss defining input function
   x0  = ss{4};      % fourth element of cell array ss defining initial condition

   n = length(x0);     % n is the number of initial conditions for called system.
   r = length(yFn(x0)); % r is the number of outputs for called system.
   m = length(uFn( 0)); % m is the number of inputs for called system.

 end
```

4.2 Dynamic system 1: Function u1, f1 & y1.

Now, let's consider the first dynamic system as defined in the project description.

$$\dot{x} + 4x = u$$
$$y = x$$
$$u(t) = 0;\ \ x_0 = 2$$

This is first order linear time-invariant system.

Input u is constant and is zero. The initial condition of x at t = 0 is 2.0 and x is the output.

Let's define the system input, function and output as $u1$, $f1$ & $y1$ respectively.

```
%%
% u1 is the function for input of dynamic system defined by f1.

function uOut = u1(~)  % function handle

uOut = 0;  % constant input of zero units for system 1
```

```
%%

% Here, we are writing the function of first dynamic system.

% Given equation of dynamic system is ẋ + 4x = u

function fOut = f1(x,u,~)   % f1 is the function of system 1.

fOut = -4*x + u;  % output of this function is derivative of x which will be
used to calculate x.
```

```
%%
% This function returns the output value of system 1.

function yOut = y1(x,~,~)   % function handle

% Inputs to this function is x array.

yOut = x;    % Output value of the equation stored in yOut variable.
```

4.3 Dynamic system 2: Function u2, f2 & y2.

Now, let's consider the second dynamic system as defined in the project description.

$$m\ddot{\omega} + c\dot{\omega} + k(t)\omega(t) = u(t)$$
$$y = \omega, \dot{\omega}$$
$$u(t) = 100\sin(8\pi t)$$

This is second order linear time-variant system because the parameter k and input u inside the equation varies with respect to time t.

Let's define the system input, function and output as $u2$, $f2$ & $y2$ respectively.

```
%%
% u2 is the function for input of dynamic system defined by f2.

function uOut = u2(t,~)  % function handle

uOut = 100 sin(8πt)  % equation for sinusoidal input u in terms of time t.
                     % amplitude = 100 N & frequency = 8π Hz
```

```
%%
% Here, we are writing the function of second dynamic system.
% Given equation of dynamic system is  mẅ + cẇ + k(t)ω(t) = u(t)

function fOut = f2(x,u,t,~)   % f2 is the function of system 2.

% Below are the given parameters of this system.
m = 1;  % mass in kg
c = 0.5; % damping coefficient in N-m-s
k = 30+(3*(sin(2*pi*t))); % spring stiffness in N/m

% Now, we will define the state vector.
% since this is second order system, there will be two state variables
% which are w = x1 and w' = x2.

fOut = zeros(1,2);  % define the function matrix of 1 row and 2 columns

fOut(1) = x(2);    % This will store the value of first state variable
fOut(2) = (1/m)*(u - c*x(2) - k*x(1));  % This will store the value of second
state variable
```

```
% This function returns the output value of system 2.
function yOut = y2(x,~)   % function handle
% Input to this function is x array.
% Output of this function is y matrix with values w and w'

yOut = x;    % Output value of the equation stored in yOut variable.
% Y matrix for this system will be of N x 2 because of two outputs.
```

4.4 Dynamic system 3: Function u3, f3 & y3.

Now, let's consider the second dynamic system as defined in the project description.

$$\ddot{\omega} - \mu(1 - \omega^2)\dot{\omega} + \omega = u$$
$$y = \omega, \dot{\omega}$$
$$u(t) = 1.2\sin(0.2\pi t)$$

This is second order non-linear system.

Let's define the system input, function and output as $u3$, $f3$ & $y3$ respectively.

```
%%
% u3 is the function for input of dynamic system defined by f3.


function uOut = u3(t,~)  % function handle


uOut = 1.2 sin(0.2πt)  % equation for sinusoidal input u in terms of time t
                        % amplitude = 1.2 units & frequency = 0.2π Hz
```

```
%%
% Here, we are writing the function of third dynamic system
% forced van der pol oscillator
% Given equation of dynamic system is ω̈ − μ(1 − ω²)ω̇ + ω = u


function fOut = f3(x,u,~)   % f3 is the function of system 3.


% Below are the given parameters of this system.
μ = 8.53;  % value of mu in the equation which indicates the non-linearity

% Now, we will define the state vector.
% since this is second order system, there will be two state variables
% which are w = x1 and w' = x2.


fOut = zeros(1,2);  % define the function matrix of 1 row and 2 columns because
of two state variables
fOut(1) = x(2);    % This will store the value of first state variable
fOut(2) = u + μ*(1-(x(1))^2)*x(2) - x(1); % This will store the value of second
state variable
```

```
% This function returns the output value of system 3.
function yOut = y2(x,~)   % function handle
% Input to this function is x array.
% Output of this function is y matrix with values w and w'
yOut = x;    % Output value of the equation stored in yOut variable.
% Y matrix for this system will be of N x 2 because of two outputs.
```

4.5 myEuler function

Description:

Out of three integration methods, first we are going to write down the matlab function for Euler integration. This method was originally devised by Euler and is therefore called, Euler's method.

As per this method, when we integrate over a tiny region from time step = nh to (n+1)h where h is the step size then the integration equation is given as :

$$\int_{nh}^{(n+1)h} f(t)dt \approx hfn$$

When we apply this integration to any differential equation, then we get the difference equation as

$$x_{n+1} = x_n + hf_n$$

This difference equations is written in matlab code provided below.

```
% MYEULER implements Euler integration.
function [t,y,u] = myEuler(ss,tEnd,dt)
%% Description, Inputs & Outputs
% This method was originally devised by Euler and
% is therefore called, Euler's method.
% As per this method, when we integrate over a tiny region from
% time step = nh to (n+1)h where h is the step size then the difference
% equation is given as x_(n+1)=x_n + hf_n

% Inputs - 3 inputs for this function - ss, tEnd & dt
% ss = This an array with four elements which are function handle f, output
% g, input u and the initial condition of the system.
% tEnd = value of time in sec for which we want to run the simulation
% dt = value of integration step size for particular system in sec

% Outputs - 3 outputs from this function - t, y & du
% create list of the outputs, describing their data types and units
% t = a time vector of size N x 1 where N is the number of total time steps
% y = an output vector of size N x r where r is the number of outputs for
% particular system.
% u = an input vector of size N x m where m is the number of input for
% particular system.

% Functions Called - initArrays


%%
% unpack the input cell array that contains function handles that define
% the dynamic system and its initial condition array.

  fFn = ss{1};   % Function handle for the dynamic system function
  yFn = ss{2};   % Function handle for the output of the system
  uFn = ss{3};   % Function handle for the input of the system
  x0  = ss{4};   % initial condition of the system
```

```
% create the time vector and acquire memory for the state, input and
% output arrays
  [t,x,u,y] = initArrays(ss, tEnd, dt);


% Set the values of the first element, t=0, for the state, input and output
% arrays.
  x(1,:) =     x0;    % value of all state variables at time t = 0
  u(1,:) = uFn(0);    % value of all inputs at time t = 0
  y(1,:) = yFn(x0);   % value of all outputs at time t = 0


% Implement Euler integration and keep track of the quantities that must
% be returned by the function.
  for n=2:length(t)
    x(n,:) = x(n-1,:) + ...                    % equation for Euler integration
      dt * fFn( x(n-1,:), u(n-1,:), t(n-1) );
    u(n,:) = uFn(t(n));                % input u of system at nth integration step
    y(n,:) = yFn(x(n,:),u(n,:),t(n)); % output y of system at nth integration
step
  end
end
```

4.6 myAB2 function with Euler integration startup scheme

Description:

AB2 stands for Adams Bashforth 2nd order integration method. This is an explicit integration method meaning that the equation of x_{n+1} is evaluated by using older values of x such as x_n or x_{n-1}. Also, this method is a multistep method meaning that it uses more than one previous values therefore they need help in getting started.

The start-up scheme for n-step integration evaluates the another starting point value at n - 1 for using in main integration equation. Therefore, AB method needs another starting point value along with the initial condition to compute the integration. In this project, we will be using AB2 integration with Euler integration startup scheme. That means, the another starting value will be computed through Euler integration first and then it will execute AB-2 integration function.

When we apply this integration to any differential equation, then we get the difference equation as

$$x_{n+1} = x_n + \frac{h}{2}(3f_n - f_{n-1})$$

This difference equations is written in matlab code provided below

```
%MYAB2 implements Adams Bashforth-2nd order integration.
function [t,y,u] = myAB2(ss,tEnd,dt)
%% Description, Inputs & Outputs
%AB2 stands for Adams Bashforth 2nd order integration method.
%This is an explicit integration method meaning that the equation ofx_(n+1)
% is evaluated by using older values of x such as x_n or x_(n-1).
% Also, this method is a multistep method meaning that it uses more than
% one previous values therefore they need help in getting started.
% The start-up scheme for n-step integration evaluates the another starting
% point value at n - 1 for using in main integration equation.
% Therefore, AB method needs another starting point value along with the
% initial condition to compute the integration.
% In this project, we are using AB2 integration with Euler integration
% startup scheme.
% That means, the another starting value will be computed through
% Euler integration first and then it will execute AB-2 integration function.

% Outputs - 3 outputs from this function - t,y & du
% create list of the outputs, describing their data types and units
% t = a time vector of size N x 1 where N is the number of total time steps
% y = an output vector of size N x r where r is the number of outputs for
% particular system.
% u = an input vector of size N x m where m is the number of input for
% particular system.

% Functions Called - initArrays
```

```
%%
% unpack the input cell array that contains function handles that define
% the dynamic system and its initial condition array.

  fFn = ss{1};  % Function handle for the dynamic system function
  yFn = ss{2};  % Function handle for the output of the system
  uFn = ss{3};  % Function handle for the input of the system
  x0  = ss{4};  % initial condition of the system

% create the time vector and acquire memory for the state, input and
% output arrays
  [t,x,u,y] = initArrays(ss, tEnd, dt);

% Set the values of the first element, t=0, for the state, input and output
% arrays.
  x(1,:) =     x0;  % value of all state variables at time t = 0
  u(1,:) = uFn(0);  % value of all inputs at time t = 0
  y(1,:) = yFn(x0); % value of all outputs at time t = 0

%% Startup scheme - Euler Integration
% Use Euler Integration for startup now to generate another starting point
for n=1     % at first time step
    fn = fFn( x(n,:), u(n,:), t(n)); % function f evaluated at nth step
    x(n+1,:) = x(n,:) + (dt*fn) ; % standard euler integration equation
    u(n+1,:) = uFn(t(n+1)); % Input u evaluated at n+1 step
    y(n+1,:) = yFn(x(n+1,:),u(n+1,:),t(n+1)); % Output y evaluated at n+1 step
end

%% Main AB2 Integration loop
% Implement AB-2 integration and keep track of the quantities that must
% be returned by the function.
for n=2:length(t)-1  % from 2nd step onwards
    fn = fFn( x(n,:), u(n,:), t(n));  % function f evaluated at nth step
    fn1 = fFn( x(n-1,:), u(n-1,:), t(n-1)); % function f evaluated at (n-1)th
step
    x(n+1,:) = x(n,:) + (dt/2)*(3*fn-fn1) ;  % standard AB2 integration equation
    u(n+1,:) = uFn(t(n+1)); % Input u evaluated at n+1 step
    y(n+1,:) = yFn(x(n+1,:),u(n+1,:),t(n+1));  % Output y evaluated at n+1 step
end
```

4.7 myRK4 function

Description:

RK4 stands for Runge Kutta 4nd order integration method. This is an explicit integration method meaning that the equation of x_{n+1} is evaluated by using older values of x. This method uses 4 estimates of function and output which include the intermediate estimate at n+1/2. The more estimates results in more accuracy.

$$\int_{nh}^{(n+1)h} f(t)dt \approx hfn$$

When we apply this integration to any differential equation, then we get the difference equation as

$$x_{n+1} = x_n + \frac{h}{6}(f_n + 2\hat{f}_{n+\frac{1}{2}} + 2\hat{\hat{f}}_{n+\frac{1}{2}} + \hat{f}_{n+1})$$

This difference equations is written in matlab code provided below.

```
%% MYRK4 implements RungeKutta 4th order integration.
function [t,y,u] = myRK4(ss,tEnd,dt)
%% Description, Inputs & Outputs
% RK4 stands for Runge Kutta 4nd order integration method.
% This is an explicit integration method meaning that the equation of
% x_(n+1) is evaluated by using older values of x.
% This method uses 4 estimates of function and output
% which include the intermediate estimate at n+1/2.
% The more number of estimates results in more accuracy.

% Inputs - 3 inputs for this function - ss,tEnd & dt
% ss = This an array with four elements which are function handle f, output
% g, input u and the initial condition of the system.
% tEnd = value of time in sec for which we want to run the simulation
% dt = value of integration step size for particular system in sec

% Outputs - 3 outputs from this function - t,y & du
% create list of the outputs, describing their data types and units
% t = a time vector of size N x 1 where N is the number of total time steps
% y = an output vector of size N x r where r is the number of outputs for
% particular system.
% u = an input vector of size N x m where m is the number of input for
% particular system.

% Functions Called - initArrays

%% Input arrays, Initial condition and integration equation
% unpack the input cell array that contains function handles that define
% the dynamic system and its initial condition array.

  fFn = ss{1};  % Function handle for the dynamic system function
  yFn = ss{2};  % Function handle for the output of the system
  uFn = ss{3};  % Function handle for the input of the system
  x0  = ss{4};  % initial condition of the system
```

```
% create the time vector and acquire memory for the state, input and
% output arrays
  [t,x,u,y] = initArrays(ss, tEnd, dt);

% Set the values of the first element, t=0, for the state, input and output
% arrays.
  x(1,:) =     x0;  % value of all state variables at time t = 0
  u(1,:) = uFn(0);  % value of all inputs at time t = 0
  y(1,:) = yFn(x0); % value of all outputs at time t = 0

% Implement RK-4 integration and keep track of the quantities that must
% be returned by the function.

%% Implement RK-4 integration
for n = 1:length(t)-1

    %funtion f evaluated at t = n
    fn = fFn( x(n,:), uFn(t(n)), t(n) );
    % intermediate output estimate at n = 1/2 using funtion f at t = n
    xnhat12 = x(n,:) + dt/2 * fn;

    % function f evaluated at n = 1/2
    fnhat12 = fFn( xnhat12, uFn(t(n)+dt/2) , t(n)+dt/2 );
    % another intermediate output estimate at n = 1/2 using funtion f
    % at n = 1/2
    xndoublehat12 = x(n,:) + dt/2 * fnhat12;

    % second evaluation of function f at n = 1/2
    fndoublehat12 = fFn( xndoublehat12, uFn(t(n)+dt/2) , t(n)+dt/2 );
    % output estimate at n = 1 using second evaluation of function f at n =
1/2
    xnhat1 = x(n,:) + dt * fndoublehat12;

    % function f evaluated at t = t(0) + dt
    fnhat1 = fFn(  xnhat1, uFn(t(n)+dt), t(n)+dt );
    % output estimate at n+1
    x(n+1,:) = x(n,:) + dt/6*(fn + 2*fnhat12 + 2*fndoublehat12 + fnhat1);

    % input u of system at n+1 integration step
    u(n+1,:) = uFn(t(n+1));
    % output y of system at n+1 integration step
    y(n+1,:) = yFn(x(n+1,:), u(n+1,:), t(n+1));
end
```

4.8 simTest

This is a test simulator code to test and plot the graphs for all three integration schemes for the three different dynamic systems.

```
%% Test Simulator Code
% This code runs all integration functions (Euler, AB2 & RK4)
% on all three dynamic systems and returns the plots for all three
% system outputs computed through three different integration methods.

%%
% Cleanup
clearvars; % clear the workspace
close all

%% Dynamic System 1 - Linear Time-Invariant
% Equation of the system x' = -4*x + u
% x' = f1(x,u,t) where f1 and u1(t) are MATLAB functions
% y1(x,u,t) is also a MATLAB function

% System 1 defined by function handles f1,u1,y1 and 1 initial condition = 0
s1 = {@f1,@y1,@u1,2.0};

%% Dynamic System 2 - Linear Time-variant
% Equation of the system is m*w'' + cw'+ k(t)w(t)= u(t)
% w'' = f2(w,w',u,t) where f2 and u2(t) are MATLAB functions
% y2(w, w') is also a MATLAB function

% System 2 defined by function handles f2,u2,y2 and 2 initial conditions
% of value 0 each
s2 = {@f2,@y2,@u2,[0 0]};

%% Dynamic System 3 - Non linear - Forced Van der pol oscillator
% Equation of the system is w'' -  mu*(1 - w2)w'+ w = u
% w'' = f3(w, w',u,t) where f3 and u3(t) are MATLAB functions
% y3(w, w') is also a MATLAB function

% System 3 defined by function handles f3,u3,y3 and 2 initial conditions
% of value 0 each
s3 = {@f3,@y3,@u3,[0 0]};

%% Simulation with Euler Integration
% Simulate the dynamic systems with Euler Integration
%where the second and third arguments are the final time and time step.

% Implement myEuler on system 1 with tEnd = 2 secs and dt = 0.05 secs
% Return the arrays t1E, output of system 1 with Euler y1E and input u1E
[Out.t1E,Out.y1E,Out.u1E] = myEuler(s1,2.0,0.05);
```

```
% Implement myEuler on system 2 with tEnd = 10 secs and dt = 0.005 secs
% Return the arrays t2E, output of system 2 with Euler y2E and input u2E
[Out.t2E,Out.y2E,Out.u2E] = myEuler(s2,10.0,0.005);

% Implement myEuler on system 3 with tEnd = 50 secs and dt = 0.01 secs
% Return the arrays t3E, output of system 3 with Euler y3E and input u3E
[Out.t3E,Out.y3E,Out.u3E] = myEuler(s3,50.0,0.01);

%% Simulation with AB2 Integration
% Simulate the dynamic systems with AB-2 Integration
%where the second and third arguments are the final time and time step.

% Implement myAB2 on system 1 with tEnd = 2 secs and dt = 0.05 secs
% Return the arrays t1A, output of system 1 with AB2 y1A and input u1A
[Out.t1A,Out.y1A,Out.u1A] = myAB2(s1,2.0,0.05);

% Implement myAB2 on system 2 with tEnd = 10 secs and dt = 0.005 secs
% Return the arrays t2A, output of system 2 with AB2 y2A and input u2A
[Out.t2A,Out.y2A,Out.u2A] = myAB2(s2,10.0,0.005);

% Implement myEuler on system 3 with tEnd = 50 secs and dt = 0.01 secs
% Return the arrays t3A, output of system 3 with ABA y3A and input u3A
[Out.t3A,Out.y3A,Out.u3A] = myAB2(s3,50.0,0.01);

%% Simulation with RK4 Integration
% Simulate the dynamic systems with RK-4 Integration
%where the second and third arguments are the final time and time step.

% Implement myRK4 on system 1 with tEnd = 2 secs and dt = 0.05 secs
% Return the arrays t1R, output of system 1 with RK4 y1R and input u1R
[Out.t1R,Out.y1R,Out.u1R] = myRK4(s1,2.0,0.05);

% Implement myRK4 on system 2 with tEnd = 10 secs and dt = 0.005 secs
% Return the arrays t2R, output of system 2 with RK4 y2R and input u2R
[Out.t2R,Out.y2R,Out.u2R] = myRK4(s2,10.0,0.005);

% Implement myRK4 on system 3 with tEnd = 50 secs and dt = 0.001 secs
% Return the arrays t3R, output of system 3 with RK4 y23R and input u3R
[Out.t3R,Out.y3R,Out.u3R] = myRK4(s3,50.0,0.001);

%%
% Plot results
%Helper Functions
h = plotData(Out);

function h = plotData(Out)
```

```
 % Set some attributes to make the plot easier to read
  FS = 14; % font size
  LW =  2; % line width
%%
  h.fig = figure(1);  % figure window 1

  h.axs(1) = subplot(3,1,1);    % create first subplot in main plot of 3 rows
  [x,y] = stairs(Out.t1,Out.y1E); % y stairs
  h.ln(1)  = line(x,y,'Color','r'); % y line
  [x,y] = stairs(Out.t1,Out.u1);  % u stairs
  h.ln(2)  = line(x,y,'Color','k'); % u line
  h.ylb(1) = ylabel('Input and Output');  % y-axis label
  h.leg(1) = legend('y=x','u');        % legend
  h.ttl   = title({      % Generate the title
                ('$\dot{x} + 4x = u$')  %Put system equation in title
                ('$x(0) = 2$')  % Put initial condition in title
                ('$Solved  by  Euler Method$')  % Put method in title
                });
  h.ttl.Interpreter = 'LaTeX';
  grid;

  h.axs(2) = subplot(3,1,2);     % create second subplot in main plot of 3 rows
  [x,y] = stairs(Out.t1,Out.y1A); % y stairs
  h.ln(1)  = line(x,y,'Color','r'); % y line
  [x,y] = stairs(Out.t1,Out.u1);  % u stairs
  h.ln(2)  = line(x,y,'Color','k'); % u line
  h.ylb(1) = ylabel('Input and Output');  % y-axis label
  h.leg(1) = legend('y=x','u');        % legend
  h.ttl   = title({      % Generate the title
                ('$\dot{x} + 4x = u$')  %Put system equation in title
                ('$x(0) = 2$')  % Put initial condition in title
                ('$Solved  by  AB2 Method$')  % Put method in title
                });
  h.ttl.Interpreter = 'LaTeX';
  grid;

  h.axs(3) = subplot(3,1,3);     % create third subplot in main plot of 3 rows
  [x,y] = stairs(Out.t1,Out.y1R); % y stairs
  h.ln(1)  = line(x,y,'Color','r'); % y line
  [x,y] = stairs(Out.t1,Out.u1);  % u stairs
  h.ln(2)  = line(x,y,'Color','k'); % u line
  h.ylb(1) = ylabel('Input and Output');  % y-axis label
  h.leg(1) = legend('y=x','u');        % legend
  h.ttl   = title({      % Generate the title
                ('$\dot{x} + 4x = u$')  %Put system equation in title
                ('$x(0) = 2$')  % Put initial condition in title
                ('$Solved  by  RK4 Method$')  % Put method in title
                });
  h.ttl.Interpreter = 'LaTeX'; %Interpreter of title is latex
```

```
 grid;


%%
 h.fig = figure(2);  % figure window 2

 h.axs(1) = subplot(3,1,1);     % create first subplot in main plot of 3 rows
 [x,y] = stairs(Out.t2,Out.y2E(:,1)); % y stairs
 h.ln(1)  = line(x,y,'Color','k'); % y line
 [x,y] = stairs(Out.t2,Out.y2E(:,2)); % y stairs
 h.ln(2)  = line(x,y,'Color','r'); % y line
 [x,y] = stairs(Out.t2,Out.u2); % y stairs
 h.ln(3)  = line(x,y,'Color','b'); % u line
 h.ylb(1) = ylabel('Input and Output');  % y-axis label
 h.leg(1) = legend('{\omega}','{\dot\omega}','u');        % legend
 % Generate the title
 % Put system equation, initial condition & method in title
 h.ttl   = title({
              ('$m\ddot{\omega} + c{\dot\omega} + k(t){\omega} = u(t)$')
              ('$\omega(0) = 0;{\dot\omega}(0) = 0$')
              ('$Solved  by  Euler Method$')
               });
 h.ttl.Interpreter = 'LaTeX'; %Interpreter of title is latex
 h.leg.Interpreter = 'LaTeX'; %Interpreter of legend is latex
 grid;

 h.axs(2) = subplot(3,1,2);      % create second subplot in main plot of 3 rows
[x,y] = stairs(Out.t2,Out.y2A(:,1)); % y stairs
 h.ln(1)  = line(x,y,'Color','k'); % y line
 [x,y] = stairs(Out.t2,Out.y2A(:,2)); % y stairs
 h.ln(2)  = line(x,y,'Color','r'); % y line
 [x,y] = stairs(Out.t2,Out.u2); % y stairs
 h.ln(3)  = line(x,y,'Color','b'); % u line
 h.ylb(1) = ylabel('Input and Output');  % y-axis label
 h.leg(1) = legend('\omega','{\dot\omega}','u');        % legend
 % Generate the title
 % Put system equation, initial condition & method in title
 h.ttl   = title({
              ('$m\ddot{\omega} + {\dot\omega} + k(t){\omega} = u(t)$')
              ('${\omega}(0) = 0;{\dot\omega}(0) = 0$')
              ('$Solved  by  AB2 Method$')
               });
 h.ttl.Interpreter = 'LaTeX';   %Interpreter of title is latex
 h.leg.Interpreter = 'LaTeX';   %Interpreter of legend is latex
 grid;
```

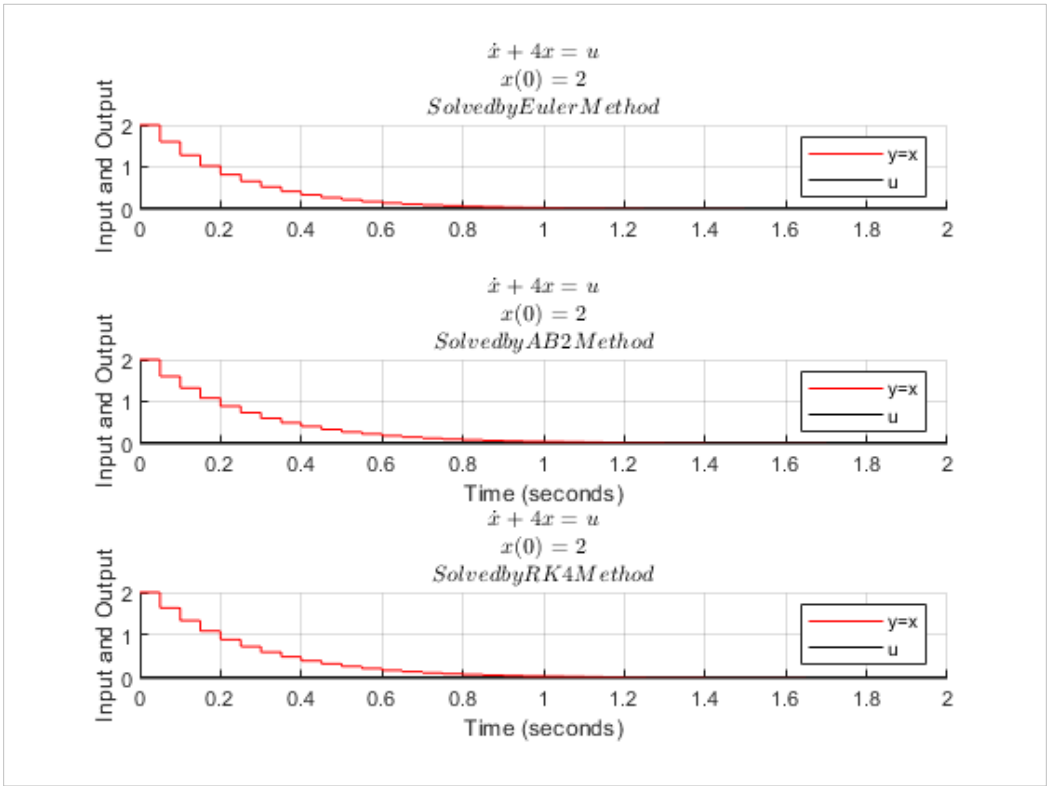
```
h.axs(3) = subplot(3,1,3);      % create third subplot in main plot of 3 rows
[x,y] = stairs(Out.t2,Out.y2R(:,1)); % y stairs
h.ln(1)  = line(x,y,'Color','k'); % y line
[x,y] = stairs(Out.t2,Out.y2R(:,2)); % y stairs
h.ln(2)  = line(x,y,'Color','r'); % y line
[x,y] = stairs(Out.t2,Out.u2); % y stairs
h.ln(3)  = line(x,y,'Color','b'); % u line
h.ylb(1) = ylabel('Input and Output');  % y-axis label
h.leg(1) = legend('w','{\dot{w}}','u');        % legend
% Generate the title
% Put system equation, initial condition & method in title
h.ttl   = title({
             ('$m\ddot{\omega} + c{\dot\omega} + k(t){\omega} = u(t)$')
             ('$\omega(0) = 0;{\dot\omega}(0) = 0$')
             ('$Solved  by  RK4 Method$')
              });
h.ttl.Interpreter = 'LaTeX';   %Interpreter of title is latex
h.leg.Interpreter = 'LaTeX';   %Interpreter of legend is latex
grid;

%%
h.fig = figure(3);  % figure window 3

h.axs(1) = subplot(3,1,1);      % create first subplot in main plot of 3 rows
[x,y] = stairs(Out.t3,Out.y3E(:,1)); % y stairs
h.ln(1)  = line(x,y,'Color','k'); % y line
[x,y] = stairs(Out.t3,Out.y3E(:,2)); % y stairs
h.ln(2)  = line(x,y,'Color','r'); % y line
[x,y] = stairs(Out.t3,Out.u3); % y stairs
h.ln(3)  = line(x,y,'Color','b'); % u line
h.ylb(1) = ylabel('Input and Output');  % y-axis label
h.leg(1) = legend('{\omega}','{\dot\omega}','u');        % legend
% Generate the title
% Put system equation, initial condition & method in title
h.ttl   = title({
             ('$\ddot{\omega} - \mu(1-\omega^2)\dot{\omega} + \omega = u$')
             ('$\omega(0) = 0; \dot{\omega}(0) = 0$')
             ('$Solved  by  Euler Method$')
              });
h.ttl.Interpreter = 'LaTeX';   %Interpreter of title is latex
h.leg.Interpreter = 'LaTeX';   %Interpreter of legend is latex
grid;

h.axs(2) = subplot(3,1,2);      % create second subplot in main plot of 3 rows
[x,y] = stairs(Out.t3,Out.y3A(:,1)); % y stairs
h.ln(1)  = line(x,y,'Color','k'); % y line
[x,y] = stairs(Out.t3,Out.y3A(:,2)); % y stairs
h.ln(2)  = line(x,y,'Color','r'); % y line
[x,y] = stairs(Out.t3,Out.u3); % y stairs
```

```
  h.ln(3)  = line(x,y,'Color','b'); % u line
  h.ylb(1) = ylabel('Input and Output');  % y-axis label
  h.leg(1) = legend('\omega','{\dot\omega}','u');        % legend
  % Generate the title
  % Put system equation, initial condition & method in title

  h.ttl   = title({
              ('$\ddot{\omega} - \mu(1-\omega^2)\dot{\omega} + \omega = u$')
              ('$\omega(0) = 0; \dot{\omega}(0) = 0$')
              ('$Solved  by  AB2 Method$')
               });
  h.ttl.Interpreter = 'LaTeX';   %Interpreter of title is latex
  h.leg.Interpreter = 'LaTeX';   %Interpreter of legend is latex
  grid;

  h.axs(3) = subplot(3,1,3);    % create third subplot in main plot of 3 rows
  [x,y] = stairs(Out.t3,Out.y3R(:,1)); % y stairs
  h.ln(1)  = line(x,y,'Color','k'); % y line
  [x,y] = stairs(Out.t3,Out.y3R(:,2)); % y stairs
  h.ln(2)  = line(x,y,'Color','r'); % y line
  [x,y] = stairs(Out.t3,Out.u3); % y stairs
  h.ln(3)  = line(x,y,'Color','b'); % u line
  h.ylb(1) = ylabel('Input and Output');  % y-axis label
  h.leg(1) = legend('w','{\dot{w}}','u');        % legend
  % Generate the title
  % Put system equation, initial condition & method in title
  h.ttl   = title({
              ('$\ddot{\omega} - \mu(1-\omega^2)\dot{\omega} + \omega = u$')
              ('$\omega(0) = 0; \dot{\omega}(0) = 0$')
              ('$Solved  by  RK4 Method$')
               });
  h.ttl.Interpreter = 'LaTeX';   %Interpreter of title is latex
  h.leg.Interpreter = 'LaTeX';   %Interpreter of legend is latex
  grid;

% Adjust its look to make it easier to read
for k = 1:length(h.axs)
 h.axs(k).FontSize = FS;       % Use font size value assigned above

    h.axs(k).FontWeight = 'normal'; % normal font
end    % For loop end

for k=1:length(h.ln)
 h.ln(k).LineWidth = LW;   % Use linewidth value assigned above

end  % For loop end
end  % helper function end
```

Chapter 5. Plots and results of simTest function

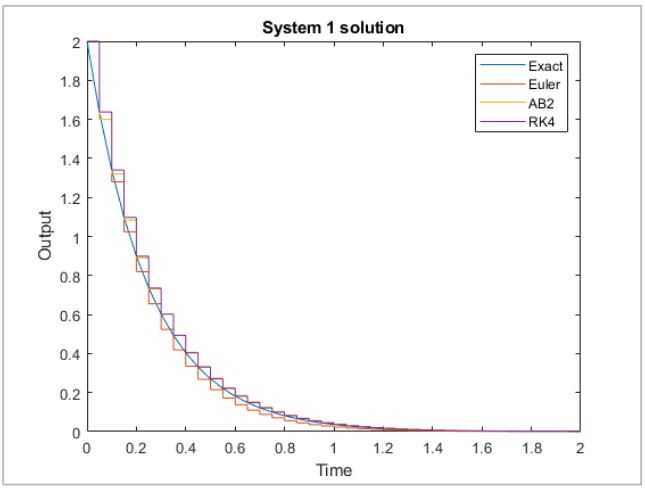
In simTest code, we have written the script for plotting the input/output vs time graphs for all three systems integrated by all three different integration methods.
In order to get rid of a greater number of plots, we have used subplot function in the script so that we have one plot for every system with all integration methods
Let's take a look at the plots system by system.

A. Dynamic system 1:

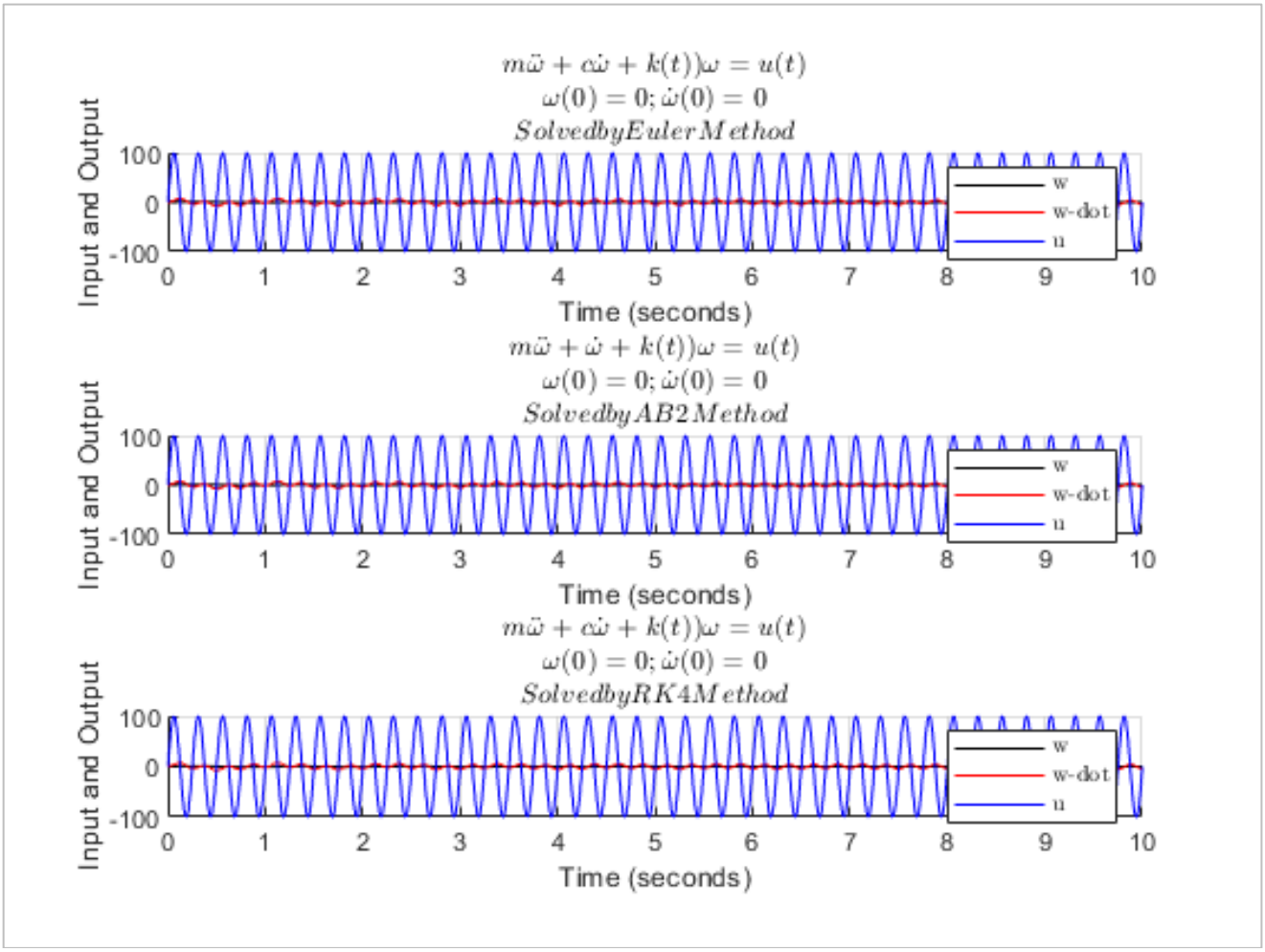


Results:

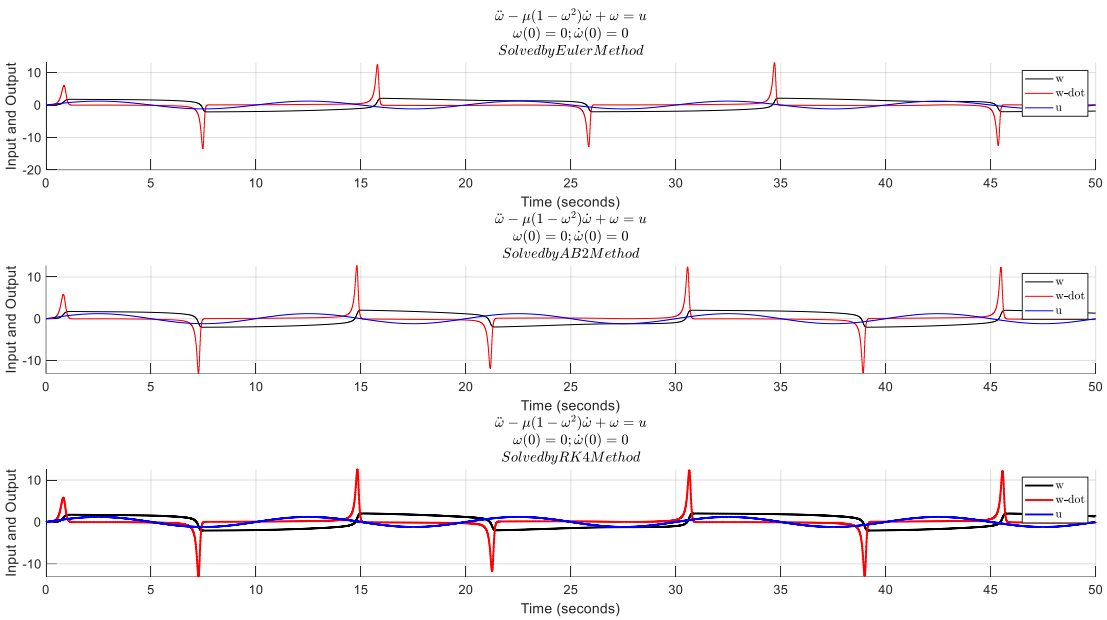
Max error between exact and simulated value by Euler	0.080399 units
Max error between exact and simulated value by AB2	0.037461 units
Max error between exact and simulated value by RK4	1.1593E-05 units



B. <u>Dynamic system 2:</u>



C. Dynamic system 3:

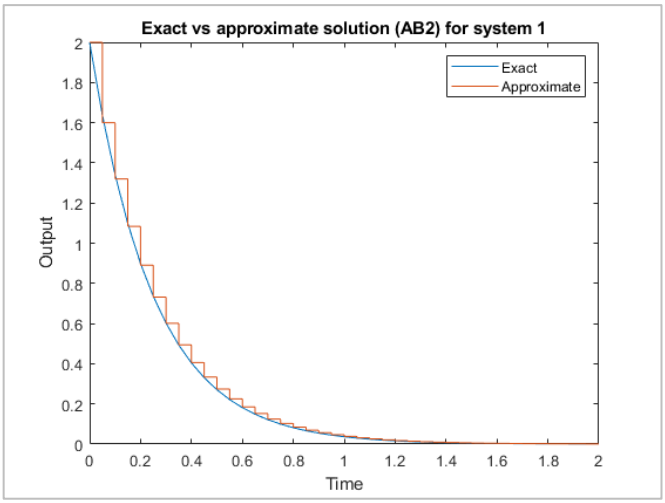


Chapter 6. Verification of functions f1, u1, y1 & myAB2

As we discussed earlier, f1, y1 & u1 are the functions for defining system 1. So, in order to get correct results we need to verify whether these functions are coded properly or not.
Therefore, one way of verifying this is to check the simulated solution against the exact solution equation. Let us try this for system 1.
System 1 is defined by the equations as $\dot{x} + 4x = u$
Its exact solution is given by $y(t) = x_0 e^{-4t}$
Let's write a matlab code for comparing the exact and simulated result by AB2 integration.

```
% Solve system 1 by AB2 integration and verify solution with exact solution
y0 = 2;                  % Initial Condition
dt=0.05;                   % Time step
t = 0:dt:2;                % t goes from 0 to 2 seconds.
yexact = y0*exp(-4*t);     % Exact solution of system 1
s1 = {@f1,@y1,@u1,2.0};     % Define system 1
[Out.t1E,Out.y1E,Out.u1E] = myAB2(s1,2.0,0.05);  % Specify AB2 integration
plot(t,yexact');  %plot the figure
hold on
plot(t,Out.y1E);  %plot t vs output
hold on   % hold the figure
axis([0 2 0 2]);  % x & y axis from 0 to 2
title('Exact vs approximate solution for system 1');  %Give title
xlabel('Time');  % Give x-label
ylabel('Output');  %Give y-label
legend('Exact','Approximate');  %Give legend
hold off
```

Max error between exact and simulated value	0.080399 units
Max error at t	0.025 seconds



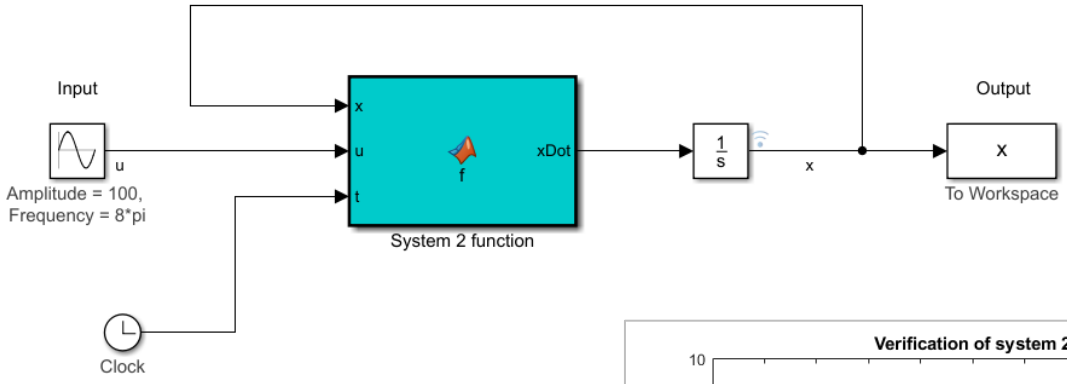
Conclusion:

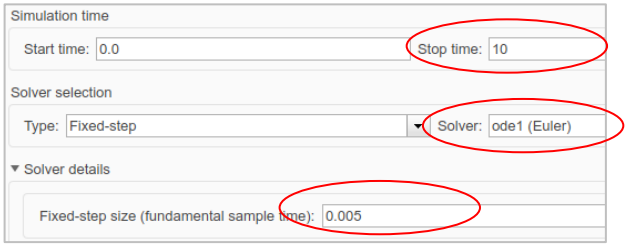
From the analysis of comparison of exact vs Simulated results, it can be seen that the functions f1, u1, y1 & mAB2 are Coded properly as both results follow same trend. However, the in-accuracy in the simulated result is more as compared to exact result so either using higher order integration or reducing the time step can yield improvement.

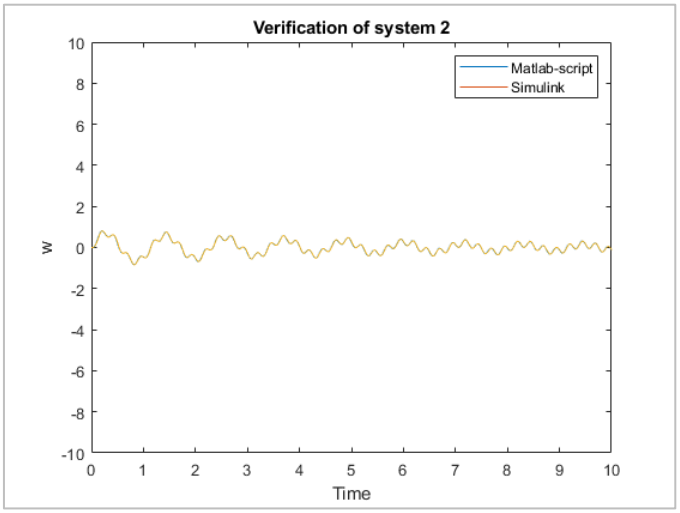
Chapter 7. Verification of functions f2, u2, y2 & myEuler

For system 2, we will verify the functions f2, u2, y2 & myEuler. In this chapter, we will verify the codes by running the system 2 through Simulink with solver as ode1-Euler.

Below is the Simulink model created for system 2 and is solved by Euler method.



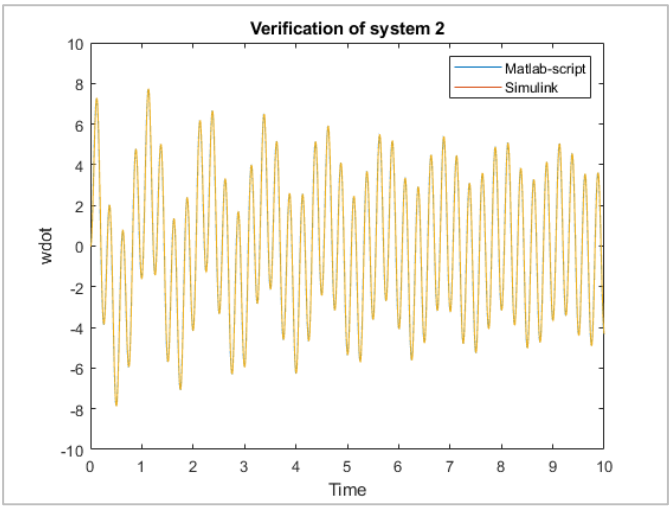




Results and analysis:

Max error of w between script based integration and Simulink solver	1.5E-14 units

Max error of wdot between script based integration and Simulink solver	9.99E-14 units



<u>Conclusion:</u>

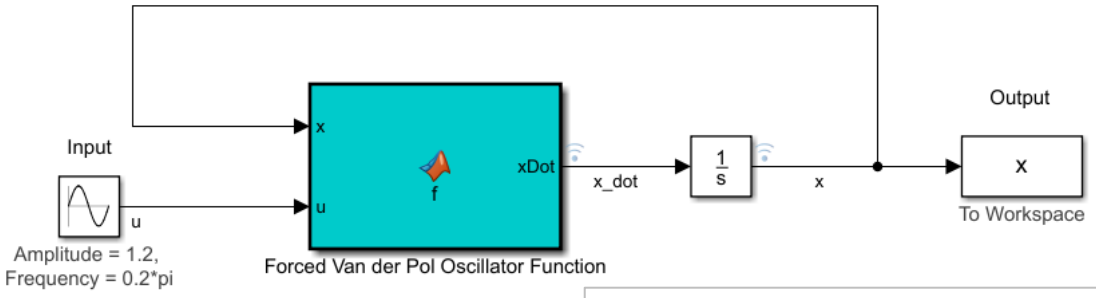
From the both figures, it can be seen that the values of matlab solution and Simulink solution are very very close to each other therefore <u>two graphs are overlapped on each other</u>.

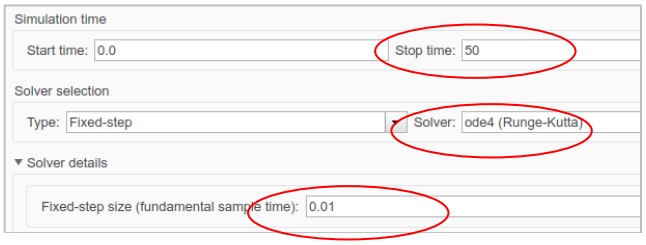
From the analysis of comparison of matlab vs Simulink results, it can be seen that the functions f2, u2, y2 & mAB2 are Coded properly as both results are matching.

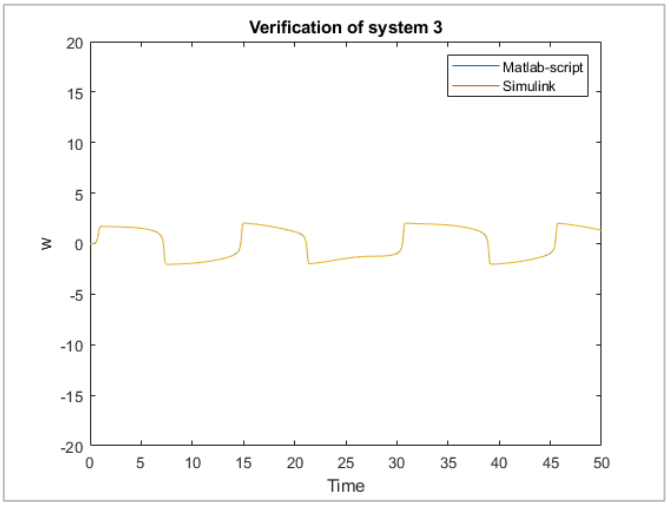
Chapter 8. Verification of functions f3, u3, y3 & myRK4

For system 3, we will verify the functions f3, u3, y3 & myRK4. In this chapter, we will verify the codes by running the system 2 through Simulink with solver as RK-4.

Below is the Simulink model created for system 3 and is solved by RK4 method.



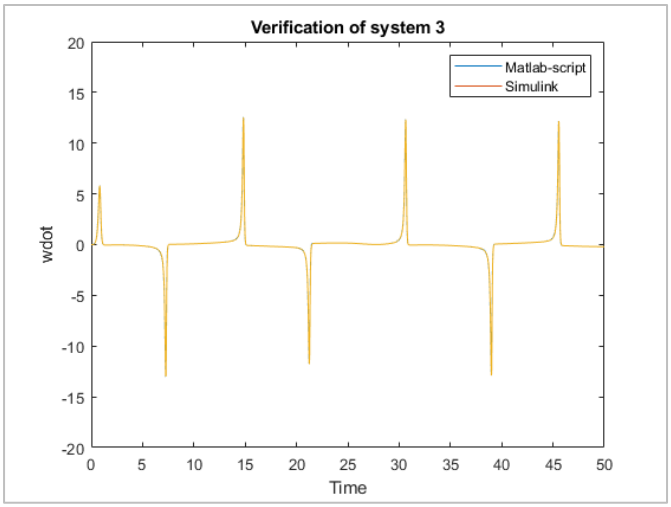




Results and analysis:

Max error of w between script based integration and Simulink solver	1.5E-14 units

Max error of wdot between script based integration and Simulink solver	9.99E-14 units



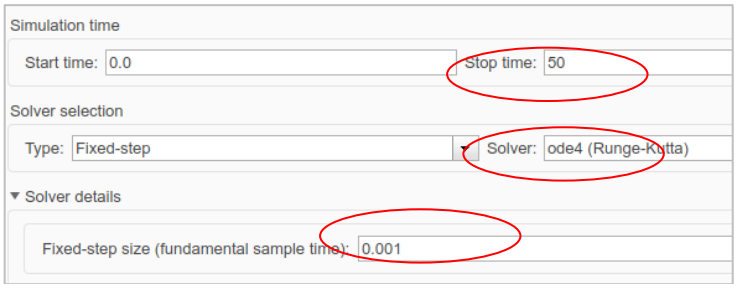
Conclusion:

From the both figures, it can be seen that the values of matlab solution and Simulink solution are very very close to each other therefore two graphs are overlapped on each other.

From the analysis of comparison of matlab vs Simulink results, it can be seen that the functions f3, u3, y3 & mRK4 are Coded properly as both results are matching.

Chapter 9. Comparison of different time steps for system 3

One way to verify the effect of smaller step size is to run the simulation by reducing the step size and then comparing the accuracy against the accuracy with previous step size.
Let's do that exercise for system 3. We need to change the step size from 0.01 to 0.001 in Simulink solver.



Solving system with time step size of 0.01 and RK4 method

Max error between script based integration and Simulink solver	1.5E-14 units

Solving system with time step size of 0.001 and RK4 method

Max error between script based integration and Simulink solver	1.0E-21 units

Conclusion:

From the comparison of max error for RK integration methods at t = 0..01 seconds and t = 0.001 seconds, it endorse the assumption that ***smaller step size with same integration method improves the accuracy in simulated results.***

Chapter 10. Conclusion

- From the comparison of max error for all three different integration methods of system 1, it can be inferred that the RK-4 method is more accurate than other two and thus endorse the assumption that ***higher order integration method improves the accuracy at same time step.***

- From the analysis of comparison of exact vs Simulated results of system 1, it can be seen that the functions f1, u1, y1 & mAB2 are Coded properly as both results follow same trend. However, the in-accuracy in the simulated result is more as compared to exact result so either using higher order integration or reducing the time step can yield improvement.

- From the analysis of comparison of matlab vs Simulink results for system 2, it can be seen that the functions f2, u2, y2 & mAB2 are Coded properly as both results are matching.

- From the analysis of comparison of matlab vs Simulink results, it can be seen that the functions f3, u3, y3 & mRK4 are Coded properly as both results are matching.

- From the comparison of max error for RK integration methods at t = 0.01 seconds and t = 0.001 seconds for system 3, it endorses the assumption that ***smaller step size with same integration method improves the accuracy in simulated results.***

References:

1. **Canvas Notes-MEEM 4730 – Dr.Gordon Parker, Michiagn Technological University**
2. Numerical Analysis : Solving ODE IVPs – Sachin Patwardhan, Indian Institute of Technology, Bombay
3. Simple Numerical Integrators – Determining Step Size – Joel Feldman
4. System Dynamics By Katsuhiko Ogata