11 January 2024

Name: **Nijat Kazimli**

Index Number: **323983**

Group: **JA2**

Predictor-corrector Adams-Bashforth-Moulton method of order 2.

Project number **10 /** Subgroup **A**

1 Task Description

The task is about implementing Predictor-corrector Adams-Bashforth-Moulton method of order 2, while using the second order Runge-Kutta method with $\gamma = ½$ to calculate the starting value Y_1.

2 Project Folder Content

AdamsBashforthMoulton.m file contains the main function which takes three arguments – an ODE function handle, timespan, and y_0.

test.m file contains the testing function, where you can find a numerical analysis for different cases. This function does not take any arguments.

testGraphs.m file contains a testing script for plotting two different ODEs' solutions with each having both numerical and analytical results. Those ODEs have different behaviors against different step sizes.

testDifferentFunctions.m file contains a testing script for having 3 different ODE functions' both analytical and numerical results in a table form.

3 Method Description

The method begins by initializing essential parameters, including the step size (h) and the length of the time span (n). The initial values of the system (y0) are used to initialize the solution vector (y). Additionally, a 2nd order Runge-Kutta step is employed to calculate an intermediate value (Y1) to kickstart the process.

$$K = f(t_0 + \frac{1}{2}h, y_0 + \frac{1}{2}h * f(t_0, y_0))$$

$$Y1 = y_0 + h * K$$

The main computation loop involves both predictor and corrector steps, following the Adams-Bashforth-Moulton approach. In the predictor step, the method estimates the next value of the solution using the Adams-Bashforth formula, incorporating information from previous steps. Subsequently, in the corrector step, the Adams-Moulton formula is applied to refine the prediction by considering the derivative at the predicted point. This process is iterated over the specified time interval, updating the solution vector at each step.

$$y_{pred} = y_i + \frac{1}{2}h\,(3 * f(t_i, y_i) - f(t_{i-1}, y_{i-1}))$$

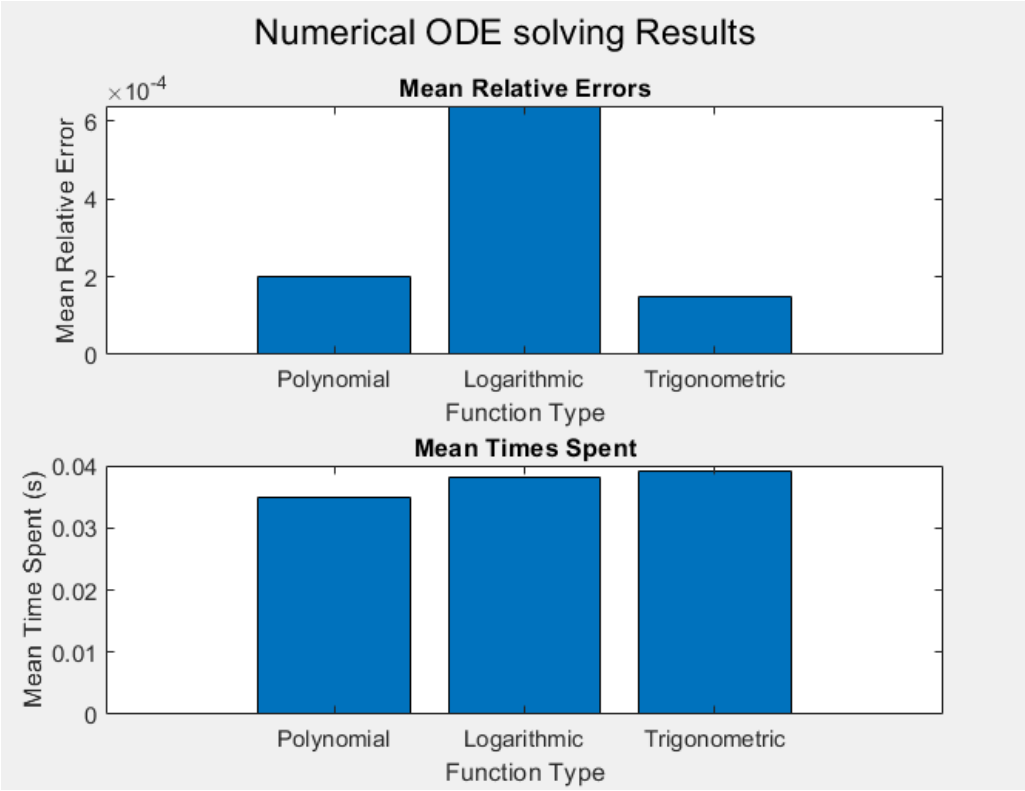
$$y_{i+1} = y_i + \frac{1}{2}h\,(f(t_{i+1}, y_{pred}) + f(t_i, y_i))$$

The overall result is a numerical approximation of the solution to the ODE over the given time span, incorporating a combination of Adams-Bashforth and Adams-Moulton steps to enhance accuracy.

4 Testing Methodology

I evaluate specific function types, recording both the time spent on each function and its accuracy. It's worth noting that identical polynomials are employed within logarithmic and trigonometric functions to ensure a fair comparison. Here is the outcome for the step size h = 5e-5:

Graphical Representation:



Polynomials:

Relative Error	Time Spent (in seconds)
0.000141	0.036306
0.000042	0.035888
0.000035	0.035018
0.000348	0.034492
0.000282	0.033853
0.000322	0.034600
0.000003	0.035534
0.000063	0.033756
0.000296	0.033842

0.000093	0.033428
Mean: 0.000162	Mean: 0.034672

Trigonometrics:

Relative Error	Time Spent (in seconds)
0.000103	0.039016
0.000031	0.039457
0.000025	0.041046
0.000245	0.039701
0.000252	0.041099
0.000241	0.039123
0.000003	0.039184
0.000060	0.038101
0.000284	0.038190
0.000074	0.038307
Mean: 0.000132	Mean: 0.039322

Logarithmics:

Relative Error	Time Spent (in seconds)
0.000295	0.037507
0.000236	0.038322
0.000304	0.038964
0.001851	0.037213
0.000621	0.037922
0.000787	0.037348
0.000011	0.038038
0.000117	0.037176
0.000402	0.038122
0.000264	0.037484

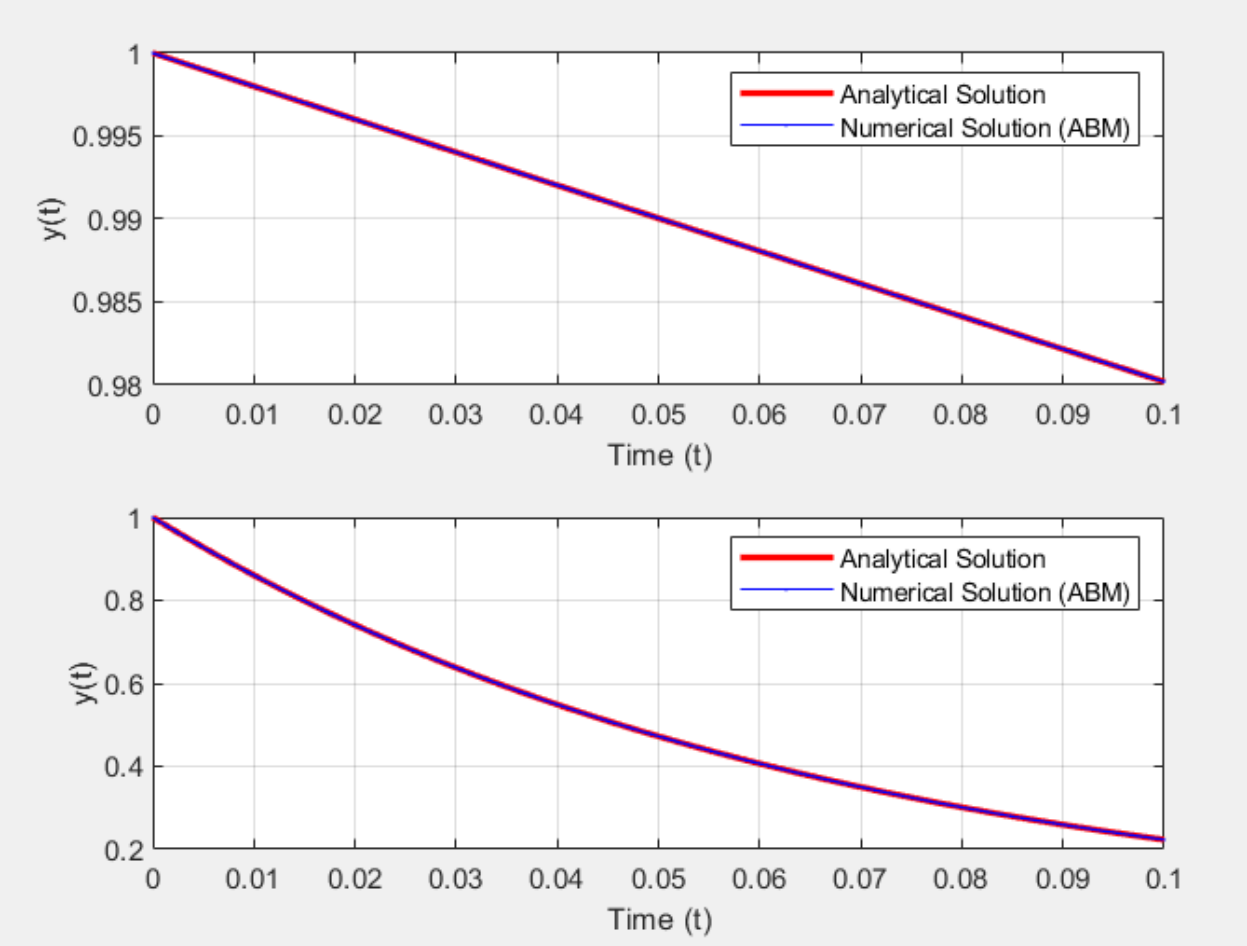
Mean: 0.000489	Mean: 0.037810

Then, I proceeded to analyze whether the step size affects the accuracy. To this purpose, I defined 2 different functions with known solutions and different behaviors in response to different step sizes:

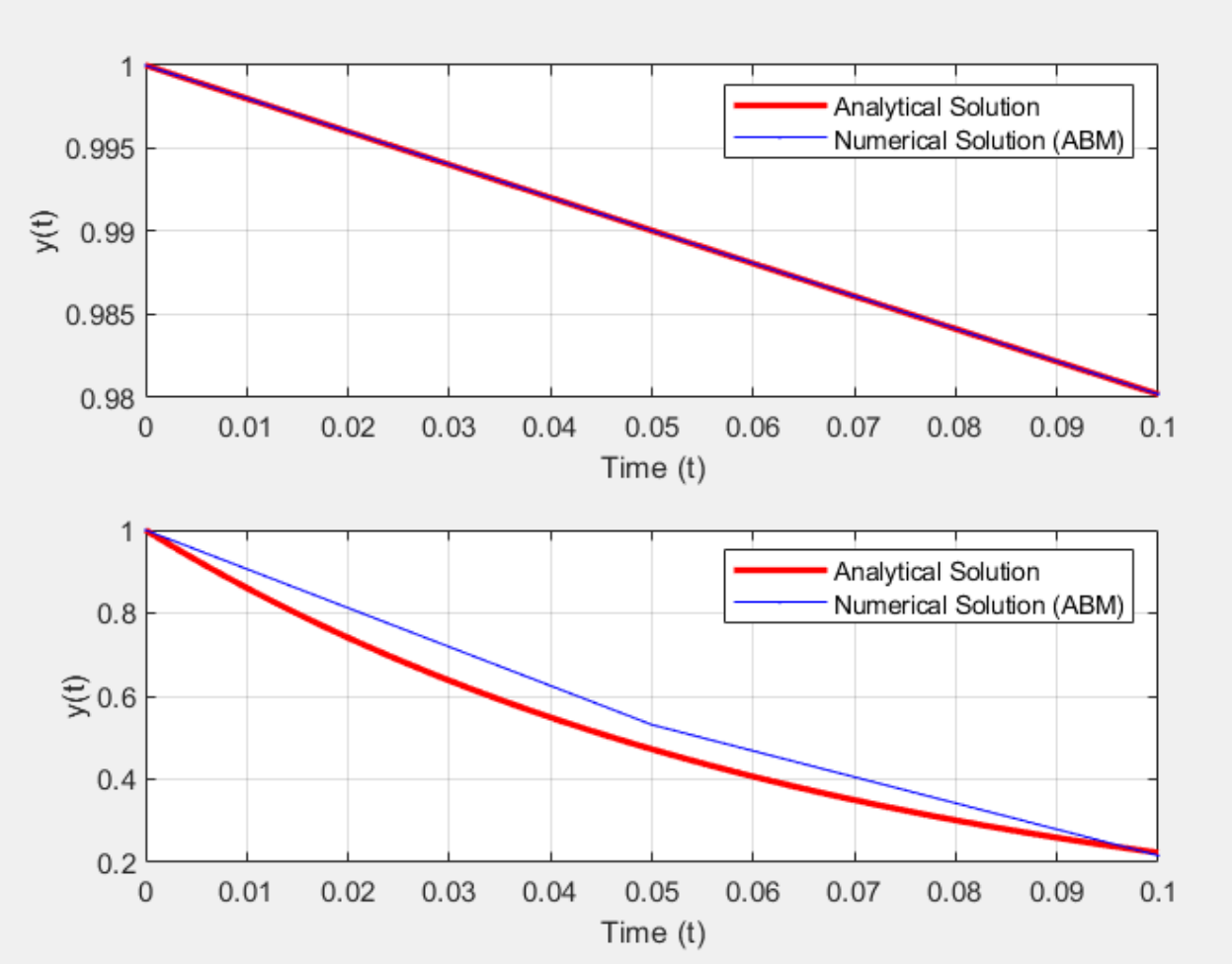
1) $y' = -0.2y$, $y(0)=1$
2) $y' = -15y$, $y(0)=1$

Here is the outcome:

h = 5e-5



h = 5e-2



Then, I proceeded to test 3 different ODEs each with a known solution, compared with the numerical solution in a table form:

$y' = 5 + sin(t), y(0) = 1$

$y = 5t - cos(t) + 2$ (known):

t	*analytical*	*numerical*
0	1	1
0.01	1.05	1.05
0.02	1.1002	1.1002
0.03	1.1504	1.1504
0.04	1.2008	1.2008
0.05	1.2512	1.2512
0.06	1.3018	1.3018
0.07	1.3524	1.3524
0.08	1.4032	1.4032
0.09	1.454	1.454
0.1	1.505	1.505

$y' = 3y + cos(t), y(0) = 1$

$y = sin(t) / 10 - 3cos(t) / 10 + 13e^{3t} / 10$ (known):

t	*analytical*	*numerical*
0	1	1
0.01	1.0406	1.0406
0.02	1.0824	1.0824
0.03	1.1256	1.1256
0.04	1.17	1.17
0.05	1.2158	1.2158
0.06	1.2629	1.2629
0.07	1.3115	1.3115
0.08	1.3616	1.3616
0.09	1.4132	1.4132
0.1	1.4663	1.4663

$y' = y * t, y(0) = 1$

$y = e^{t^2/2}$ (known):

T	*analytical*	*numerical*
0	1	1
0.01	1.0001	1.0001
0.02	1.0002	1.0002
0.03	1.0005	1.0005
0.04	1.0008	1.0008
0.05	1.0013	1.0013
0.06	1.0018	1.0018
0.07	1.0025	1.0025
0.08	1.0032	1.0032
0.09	1.0041	1.0041
0.1	1.005	1.005

5 Analysis

As you can see, the code works perfectly fine, and overall, the performance was very good.

When it comes to different steps sizes, as you can see from the results, smaller steps size means accuracy. However, for the functions the value of which do not change very fast the default setting (5e-5) is more than enough.