EE1103: Numerical Methods

Programming Assignment # 7

Lakshmiram S, EE22B117

January 27, 2023

Contents

1 Problem 1

1.1 Approach

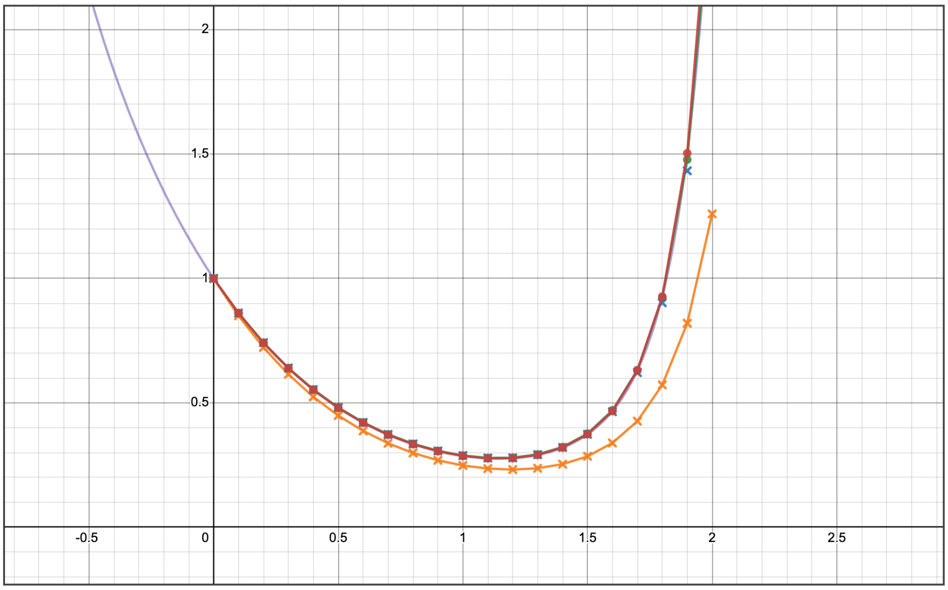
- To start with, we are required to calculate the analytical solution of the presented differential equation , which can be found out using calculus.
- Next we develop c code for all the methods mentioned in the question to solve the differential equation numerically.(it is straightforward formula implementation.)
- the clock function is used to compare the relative computational time of the various methods.

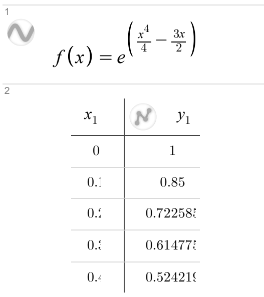
1.2 Results

The analytical solution obtained is $y = e^{0.25t^4 - 1.5t}$
For an interactive graph, click here.

Table 1: table summarising the outcomes of the methods and percentage errors computed with respect to the true value of the function evaluated at that point.

	x values	actual function value	euler	total error %	heun's	error %	midpoint	error %	fourth order RK	error %
h=0.1	0	**1**	1	0	1	0	1	0	1	0
	0.1	**0.86073**	**0.85**	1.2466162443507276	**0.861292**	0.06529341373020012	**0.861262**	0.061808000185886044	**0.86073**	0
	0.2	**0.741115**	**0.722585**	2.5002867301295932	**0.742118**	0.13533662117216455	**0.742024**	0.12265302955682299	**0.741116**	0.00013493182569894764
	0.3	**0.638921**	**0.614775**	3.7791839679710013	**0.640254**	0.20863299218526685	**0.640097**	0.1840603141859582	**0.638922**	0.0001565138726115992
	0.4	**0.552335**	**0.524219**	5.0903889849457356	**0.5539**	0.28334253668515075	**0.553696**	0.24640842966676843	**0.552337**	0.0003620990884058568
	0.5	**0.479805**	**0.448941**	6.432613249132461	**0.481518**	0.3570200393910068	**0.481288**	0.3090838986671694	**0.479807**	0.0004168360062946406
	0.6	**0.419958**	**0.387212**	7.7974464113077975	**0.421751**	0.42694745665042433	**0.421515**	0.3707513608503648	**0.41996**	0.0004762381000009525
	0.7	**0.371586**	**0.337494**	9.174726711985922	**0.373409**	0.4905997534890882	**0.37318**	0.42897202800966244	**0.371587**	0.00026911670514315515
	0.8	**0.333671**	**0.298446**	10.556805955567013	**0.335495**	0.5466462473514306	**0.335274**	0.48041334128528435	**0.333672**	0.0002996964075477808
	0.9	**0.305448**	**0.268959**	11.946059558419106	**0.307266**	0.5951913255283997	**0.30704**	0.5212016447971446	**0.305449**	0.00032738796784682027
	1	**0.286505**	**0.248223**	13.361721435926077	**0.288331**	0.6373361721435906	**0.288069**	0.5458892514964869	**0.286506**	0.00034903404826207025
	1.1	**0.276934**	**0.235811**	14.849386496421538	**0.278809**	0.6770566272108011	**0.278453**	0.5485061422577193	**0.276935**	0.000361096867836107
	1.2	**0.277593**	**0.231826**	16.48708721041236	**0.279577**	0.7147154287031883	**0.27903**	0.5176643503258446	**0.277594**	0.0003602396314131681
	1.3	**0.290551**	**0.237112**	18.39229601687828	**0.29273**	0.7499543969905409	**0.291817**	0.4357238488251596	**0.290552**	0.000344173656250794
	1.4	**0.319947**	**0.253639**	20.724682525543287	**0.322408**	0.7691898970767006	**0.320819**	0.27254514028887267	**0.319948**	0.00031255176639529537
	1.5	**0.373673**	**0.285191**	23.678992059902644	**0.376448**	0.7426279126402034	**0.373593**	0.021409092977006362	**0.373673**	0
	1.6	**0.466919**	**0.338665**	27.468147580201276	**0.469765**	0.6095275625965136	**0.464464**	0.5257871279600926	**0.466917**	0.00042833981910063397
	1.7	**0.630038**	**0.426582**	32.29265536364473	**0.631717**	0.2664918623955993	**0.621476**	1.3589656496909643	**0.630023**	0.00238080877661145
	1.8	**0.927187**	**0.572175**	38.28914771238164	**0.92305**	0.44618830937016446	**0.902258**	2.6886701388177334	**0.92711**	0.008304689345298601
	1.9	**1.503845**	**0.820041**	45.47037759875519	**1.477458**	1.7546356173674922	**1.432619**	4.736259388434314	**1.503479**	0.024337614581295842
	2	**2.718282**	**1.259501**	53.66555052051258	**2.610843**	3.9524596785763895	**2.507051**	7.7707537334242645	**2.716554**	0.06356956342277777
time			**0.000027**		**0.000015**		**0.000015**		**0.000015**	
h=0.25	0	**1**	1	0	1	0	1	0	1	0
	0.25	**0.687961**	**0.625**	9.151826920421367	**0.696533**	1.2460008634210242	**0.695709**	1.126226632032917	**0.688016**	0.007994639230990743
	0.5	**0.479805**	**0.393066**	18.077969174977326	**0.492003**	2.5422828023884794	**0.490696**	2.2698804722752035	**0.479871**	0.01375558820771157
	0.75	**0.351376**	**0.25795**	26.58861162970721	**0.363927**	3.571957105778419	**0.363114**	3.3405810300077325	**0.351429**	0.015083557215054454
	1	**0.286505**	**0.188424**	34.233608488508054	**0.298268**	4.105687509816571	**0.297916**	3.9828275248250478	**0.286543**	0.013263293834307426
	1.25	**0.282339**	**0.164871**	41.605304261897935	**0.294408**	4.274648560772687	**0.292597**	3.6332210569563497	**0.282374**	0.012396445407827911
	1.5	**0.373673**	**0.183548**	50.88004752818641	**0.387902**	3.807874799624283	**0.377589**	1.047975101224876	**0.373684**	0.002943750284349888
	1.75	**0.755577**	**0.269586**	64.32051266780222	**0.753668**	0.25265459377403615	**0.702802**	6.9847282275664835	**0.75458**	0.13195213724081406
	2	**2.718282**	**0.529695**	80.51361117058495	**2.320435**	14.635972279550103	**2.029023**	25.35641997408657	**2.682811**	1.3049050834313662
time			**0.000023**		**0.000006**		**0.000037**		**0.000008**	
h=0.5	0	**1**	1	0	1	0	1	0	1	0
	0.5	**0.479805**	**0.25**	47.89549921322204	**0.539062**	12.350225612488419	**0.536133**	11.739769281270513	**0.481096**	0.26906764206293016
	1	**0.286505**	**0.078125**	72.73171497879618	**0.332703**	16.124674962042555	**0.346471**	20.930175738643293	**0.286932**	0.1490375386118953
	1.5	**0.373673**	**0.058594**	84.31944507631005	**0.408081**	9.20805088941402	**0.415156**	11.101417549568755	**0.373752**	0.021141479314800843
	2	**2.718282**	**0.113525**	95.82364890765564	**1.884184**	30.684748675818028	**1.591802**	41.44088067389623	**2.513072**	7.549253535873015
time			**0.000017**		**0.000004**		**0.000004**		**0.000004**	





1

$$f(x) = e^{\left(\frac{x^4}{4} - \frac{3x}{2}\right)}$$

2

x_1	y_1
0	1
0.1	0.85
0.2	0.72258
0.3	0.61477
0.4	0.52421

Figure 1: graph demonstrating the curves predicted by all the methods along with the true solution.

1.3 Inferences

- We understand that Euler's method is quite primitive . It can deviate from the actual value (as is clear from the errors obtained), and significant computational effort is required to achieve reasonable accuracy with this method.

- The others, namely Heunn's, midpoint and fourth order RK methods are very sophisticated and resemble closely the actual function for the step sizes we have considered. We therefore conclude that the last three methods are more efficient that the primitive euler's method both in terms of computational effort and time taken.

1.4 Contributions

I worked on this assignment independently

2 Problem 2

2.1 Approach

For the explicit euler method, we first find out those step sizes for which the function approaches a value without violently oscillating.This can be accomplished by observing the homogeneous part of the DE. We observe that for values of $h < \frac{2}{a}$, we get well behaved curves.
Implicit euler method, on the other hand, does not require any particular step size for stability. It is designed to be stable for all values of h and hence is *unconditionally stable.*

2.2 Results

Since h has to be less than 2×10^{-5}, we fix h as 1×10^{-5}. Doing so, we get the below graph. and the following for implicit euler :

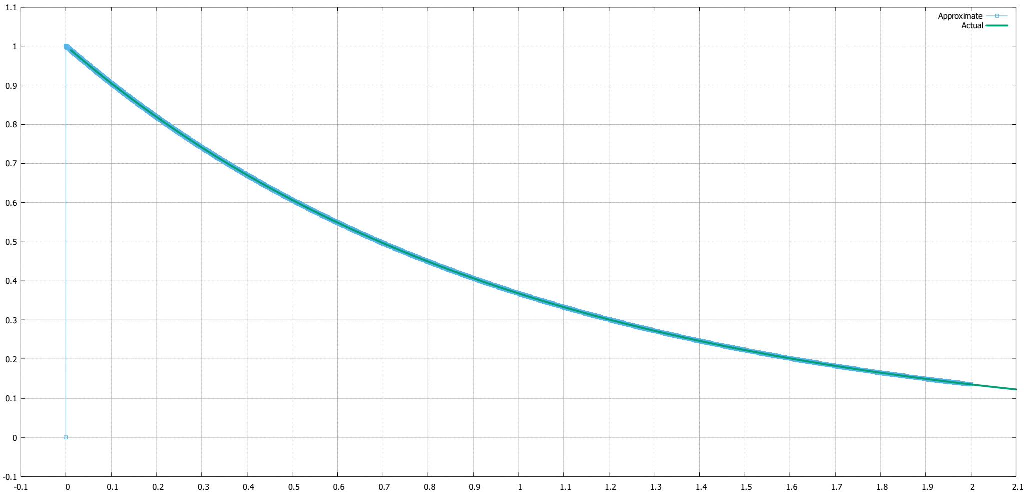


Figure 2: explicit euler method for a stable step size.

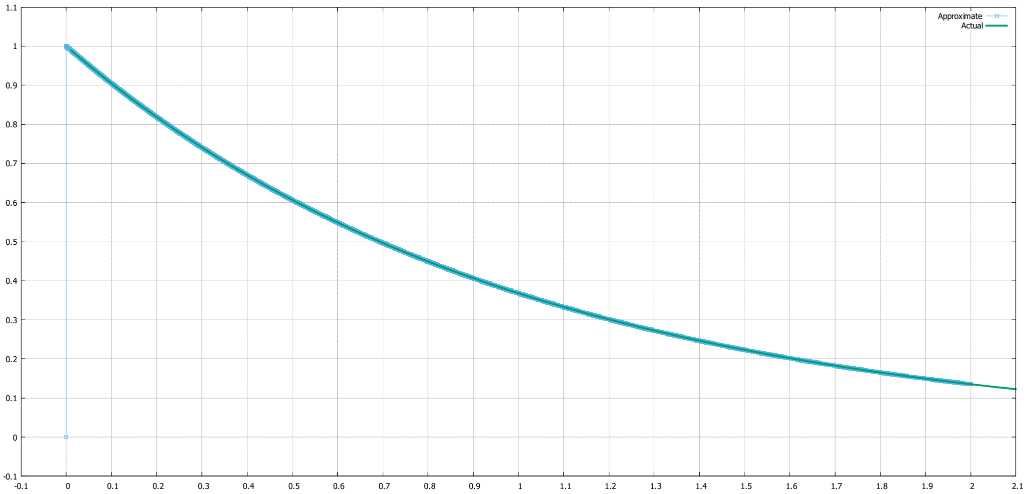


Figure 3: implicit euler method is always stable.

2.3 Inferences

This question demonstrates why explicit euler is yet again inefficient as to ensure stability, a very small value of h has to be chosen, and to be accurate enough, h has to be even smaller. This greatly decreases the efficiency of the process.
But, the implicit euler method seems to converge to the actual curve much better than the explicit method.

2.4 Contributions

I wrote the code for this question and the graphs were prepared by my collaborator.