PNS Assignment 1

Emer Keeling

Due: 23rd October 2023

1 Introduction

This is one part of the submission of assignment 1 for Practical Numerical Solutions, module MAU34601, Michaelmas Term 2023/2024, Trinity College Dublin. This aim of this assignment is to demonstrate programming skills in C++ and an understanding of the 4th Order Runge Kutta methods for numerical integration. The questions assigned are seen in this document in bold and answered underneath. The source code for this assignment in written in C++ using a text editor and compiler in 'Visual Studio Code 2' version: 1.82.2 (Universal) using Microsoft's C++ extension version: 1.17.5 and the graphs are plotted using the same software with Microsoft's Python Extension version: 2023.18.0. This document is written in LaTeX in Overleaf.
Note: some important results are highlighted in the colour purple.

2 Question 1 : Fourth Order Runge Kutta

Find the analytic solution to the following first order differential equation:

$$\frac{dx}{dt} = (t-1)^2(x+1) \tag{1}$$

with initial value $x(0) = 0$**.**
Solution: This is a separable differential equation. Separating the variables t and x onto either side of the equation gives:

$$\frac{1}{(x+1)}dx = (t-1)^2 dt \tag{2}$$

integrate both sides:

$$\int \frac{1}{(x+1)}dx = \int (t-1)^2 dt \tag{3}$$

make the following substitutions: $u = (x+1) \Rightarrow dx = du,\ v = (t-1) \Rightarrow dt = dv$

$$\int \frac{1}{u}du = \int v^2 dv \tag{4}$$

$$\ln u = \frac{1}{3}v^3 + c \tag{5}$$

$$\ln(x+1) = \frac{1}{3}(t-1)^3 + c \tag{6}$$

$$e^{\frac{1}{3}(t-1)^3+c} = x+1 \tag{7}$$

$$x = e^{\frac{1}{3}(t-1)^3+c} - 1 \tag{8}$$

using the boundary condition $x(0) = 0$ to find the constant of integration:

$$0 = e^{\frac{1}{3}(-1)^3+c} - 1 \tag{9}$$

$$1 = e^{\frac{1}{3}(-1)^3+c} \tag{10}$$

$$\ln(1) = \frac{1}{3}(-1)^3 + c \tag{11}$$

$$c = \frac{1}{3} \tag{12}$$

sub the value for c back into the original expression:

$$x = e^{\frac{1}{3}((t-1)^3+1)} - 1 \tag{13}$$

this is the analytical solution to this ODE and appears in the source code as the function “ansol”.

Write a C++ program which uses the fourth-order Runge Kutta method to construct an approximate solution for $t \in [0,2]$.
Source code for this program is available as an attachment to the submission.

Make a plot for this solution for step-size $h = 0.2$
See this plot of x and t values below.

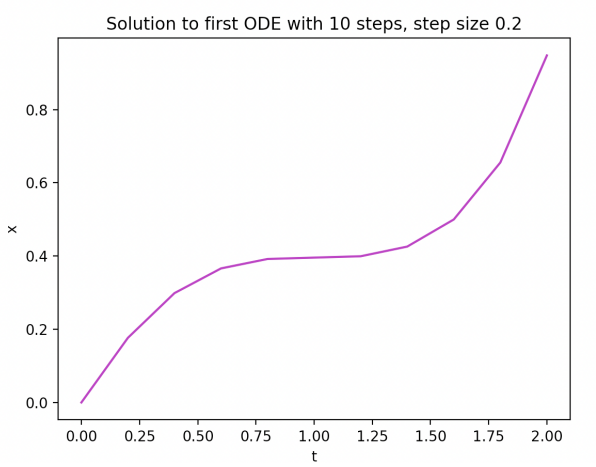

Figure 1: Plot for Solution of ODE using 4th Order Runge Kutta Methods with Step size 0.2

By evaluating the discrepancy between the numerical and analytic solutions at $t = 2$ for a range of step-sizes, illustrate the fourth order accuracy of the method:
See this plot of h and discrepancy values below.

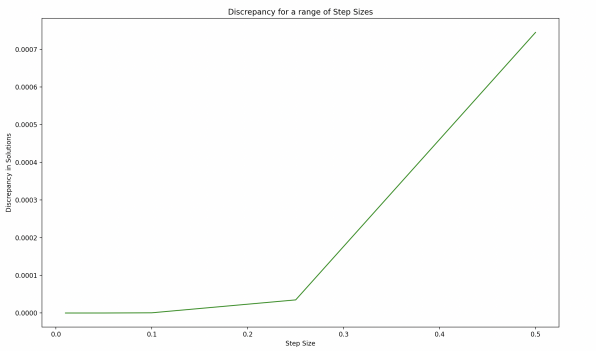

Figure 2: Linear Plot of the Discrepancy between the Estimated Solution and the Analytical Solution for a Range of Step Sizes at t=2

To improve analysis, these values can be plotted on a log-log plot also seen below ;

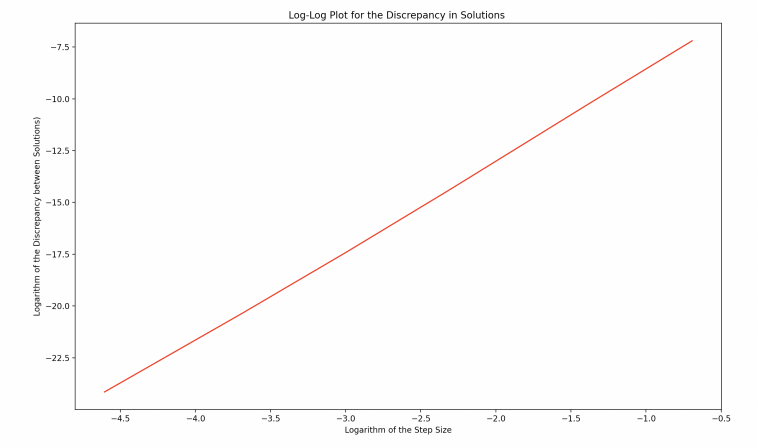

Figure 3: Log-Log Plot of the Discrepancy between the Estimated Solution and the Analytical Solution for a Range of Step Sizes at t=2

In order to estimate the slope (m) of this linear plot accurately, one can pick two points which lie on the line and not within the data set. The points that will be used are $(-1.440, -10.58)$ and $(-3.910, -21.35)$.

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{14}$$

$$m = \frac{-10.58 + 21.35}{-1.440 + 3.910} \tag{15}$$

$$\Rightarrow m = 4.360 \tag{16}$$

$$ln|x_{analytical}(2) - x_{rk}(2)| \approx 4.36 \ln(h) \tag{17}$$

Then we can estimate the relationship between step-size and discrepancy to be given by

$$|x_{analytical}(2) - x_{rk}(2)| \approx h^{(4.36)} \tag{18}$$

What value of h is required for a solution accurate to 8 significant figures?
By trial and error one may find the lowest value for number of steps n for which the solution for x has converged - in this case it is checked against the value for 10,000 steps. The corresponding value of step size is the largest possible value of h required to obtain a solution accurate to 8 significant figures.

n	h	x (8 sf)
10	0.2	0.947721
...	...	...
50	0.04	0.94773403
51	0.392	0.94773404
52	0.385	0.94773404
...	...	...
100	0.02	0.94773404
...	...	...
10000	0.0002	0.94773404

Clearly from above the smallest value for n at which we obtain a solution at $t = 2$ accurate to 8 significant figures is 51 and the corresponding value of h is given by $h = \frac{2}{n} = 0.392$ (3 dp). This is the largest value of the step size that will give a solution at $t = 2$ which is accurate to 8 significant figures.
Answer: h=0.392

We can check this convergent solution of the estimated solution (found using fourth order Runge Kutta) since we have an analytical solution. Recall, our analytical solution is given by

$$x = e^{\frac{1}{3}((t-1)^3+1)} - 1 \tag{19}$$

At $t = 2$, x is given by

$$x = e^{\frac{2}{3}} - 1 \tag{20}$$

So accurate to 8 significant figures the analytical solution at $t = 2$ is

$$x = 0.94773404 \tag{21}$$

This coincides with the estimated solution for 10,000 steps and therefore also for 51 steps (step size h=0.392 as we have seen).

3 Question 2 : Runge-Kutta for higher-order ODE

Write C++ code that uses fourth-order Runge-Kutta to evaluate y numerically, where y(x) obeys the fourth-order ODE:

$$10\frac{d^4y}{dx^4} + 50\frac{d^2y}{dx^2} + xy^3 = 0 \tag{22}$$

with initial data:

$$y = 1,\ \frac{dy}{dx} = \frac{d^2y}{dx^2} = \frac{d^3y}{dx^3} = 0 \tag{23}$$

when $x = 0$.
Source code for this program is available as an attachment to the submission.

Use this code to evaluate $y(3)$ numerically, giving a solution accurate to 6 significant figures.
In order to get a solution accurate to 6 significant figures, one must determine at what step size the solution for y(30) converges, in a similar fashion as to part 1. This is demonstrated in the table below. The table begins at $n = 22$ as this is the lowest integer value of steps the program could estimate y(30) to 6 significant figures.

n	h	y(30) (6 sf)
22	0.09	4.62032×10^{63}
...	...	...
99	0.0202	-0.617402
100	0.02	-0.617391
101	0.198	-0.61738
...	...	...
500	0.004	-0.61704
...	...	...
1000	0.002	-0.617039
...	...	...
2000	0.001	-0.617039
...	...	...
3000	0.0007	-0.617039
...	...	...
10000	0.0002	-0.617039

We can see that the solution eventually converges somewhere between 500 and 1000. It stabilises after this point. We do not need to find the exact point in this case, and we can assume it is safe to use 2000 steps to determine a value for y(30) accurate to 6 significant figures. This is also demonstrated in the graph below (figure 4) which depicts the stabilisation of the estimated solution before this point.

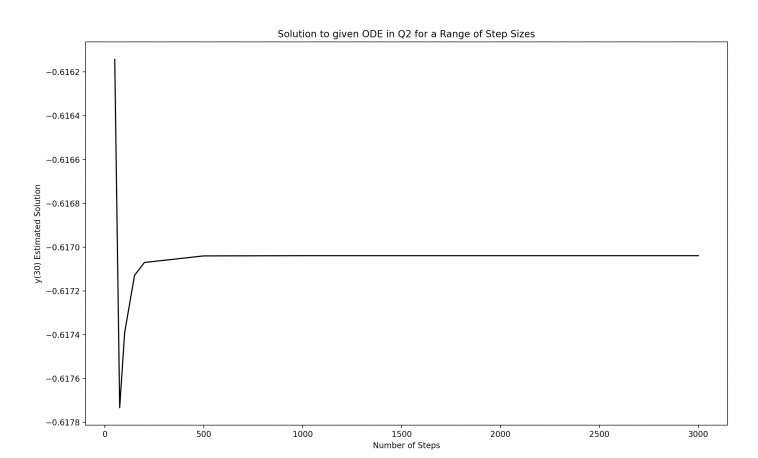

Figure 4: Stabilisation of Estimated Solution for y(30) over a Range of Step Sizes

Running the program in the source code file with an appropriately high number of steps n - in this case 2000 (for enough significant figures in the solution), and setting the step size in the program to be $h = \frac{30}{n}$, one obtains the solution y=-0.617391 (6 sf).