# Homework 4 Report

## Group 334-7

| Final Report | Cameron Rabiyan | Maya Apotheker | Manny Gamboa |
|---|---|---|---|


**Abstract:**

One of the first topics differential equations students learn in the subject, is the initial value problem, in which an initial condition is applied to the derivative or integral to a function. In this project, we were tasked with creating a program that implemented both Python and Fortran to solve a first-order ordinary differential equation (ODE) with an initial condition.


**Methods:**

Purpose:

The euler method is a tool used to solve first order differential equations with the following conditions, $\frac{dy}{dt} = y'(t) = f(t, y(t)),\ y(t_o) = y_o$. Using the fact $y'(t_o) \approx \frac{y(t+\Delta t)-y(t)}{\Delta t}$, by definition of derivative, we can assume that $y'(t + \Delta t) \approx y(t) + \Delta t f(t, y(t))$. In our code we have 2 main components, a fortran implementation and a python implementation. These two are used simultaneously to compute the euler method of the desired function to find a numerical solution and then compared to the real solution of the function given by $y(t) = -\sqrt{2ln(t^2 + 1) + 4}$.

Fortran Implementation:

In our specific Fortran implementation, it had 3 components, a makefile, a euler.F90 file, and a main.F90 file. Our first file, the makefile, contains instructions for the compilation of our fortran code. This file is compiled using double precision. Our second file, the euler.F90 file, which is where we assign the function and the other constants needed to

perform the euler method on the function $\frac{dy}{dt} = \frac{2t}{y(1+t^2)}$ with the initial condition $y(0) = (-2)$ . Five arguments are passed into our fortran subroutine in this file which executes euler's method and stores the respective data. These arguments are:

- t_0: are initial point of t, which is a constant that remains for all outputs of our program.
- t_f: is the final point of t, which is a constant that remains for all outputs of our program.
- y_0 : is the initial value of y, which in our case would be -2.
- N: is our total number increments. This value changes to create 4 different output files; one where there is 8 increments, another where there are 16 increments, another with 32 increments, and the last with 64 total increments.
- File_name: which decides the name of the output file that will decide the contain our data. In our program this file will be labeled output_x.dat where x is the total number of increments.

Our last file within our Fortran implements is the main.F90 file. This is our main program file which organizes our order of operations when this file is executed.

Python Implementation:

In our Python implementation only had 1 primary component, a file titled pyRun_euler.py in which uses subprocesses in order to run checks in order to execute the Fortran component of our code, collects data, and calculates error. Our fortran implementation executes the following tasks:

- Compiles Fortran codes using a makefile and checks for the existence of previous made files in order to store new data properly.
- Executes Fortran code.
- Stores this data in 4 files; output_8.dat, output_16.dat, output_32.dat, output_64.dat. These files contain the data for euler methods calculated using 8, 16, 32, 64 increments of N, respectively.

- Uses the following formula in order to calculate the error within our euler method calculation:

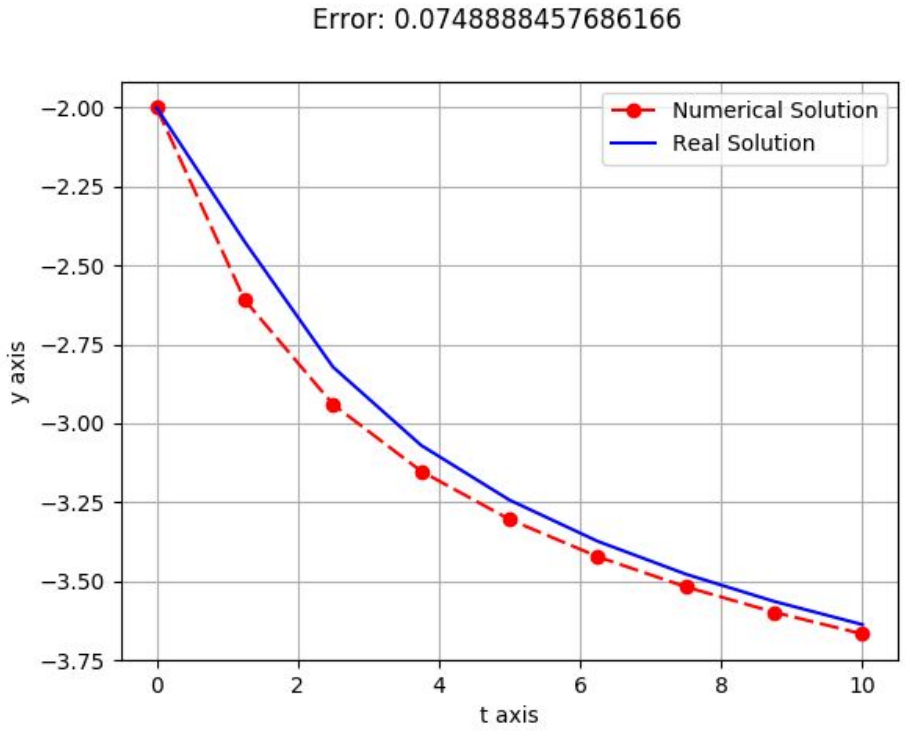
$$Error = \frac{1}{N}\sum_{i=0}^{N-1}\left|Real\ Solution(x_i) - Numerical\ Solution(x_i)\right|$$

  This output data is then stored into result_8.png, result_16.png, result_32.png, result_64.png. These files contain the error within our euler method calculation for 8, 16, 32, 64 increments of N, respectively.
- These result_8.png, result_16.png, result_32.png, result_64.png are then used to be graphed. Numerical results are plotted using dotted red lines, real results are plotted using solid blue lines, displays calculated error, labels our vertical and horizontal axis as “t-axis” and “y-axis,” and creates a grid.

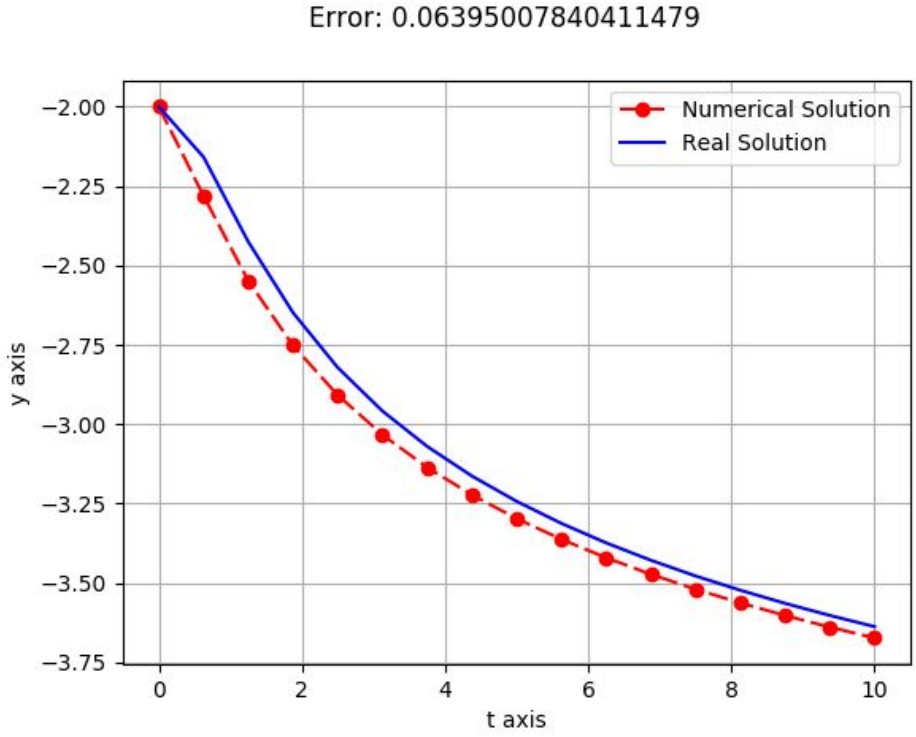
All these steps are executed consecutively and and all in one execution of our pyRun_euler.py file.

**Results:**

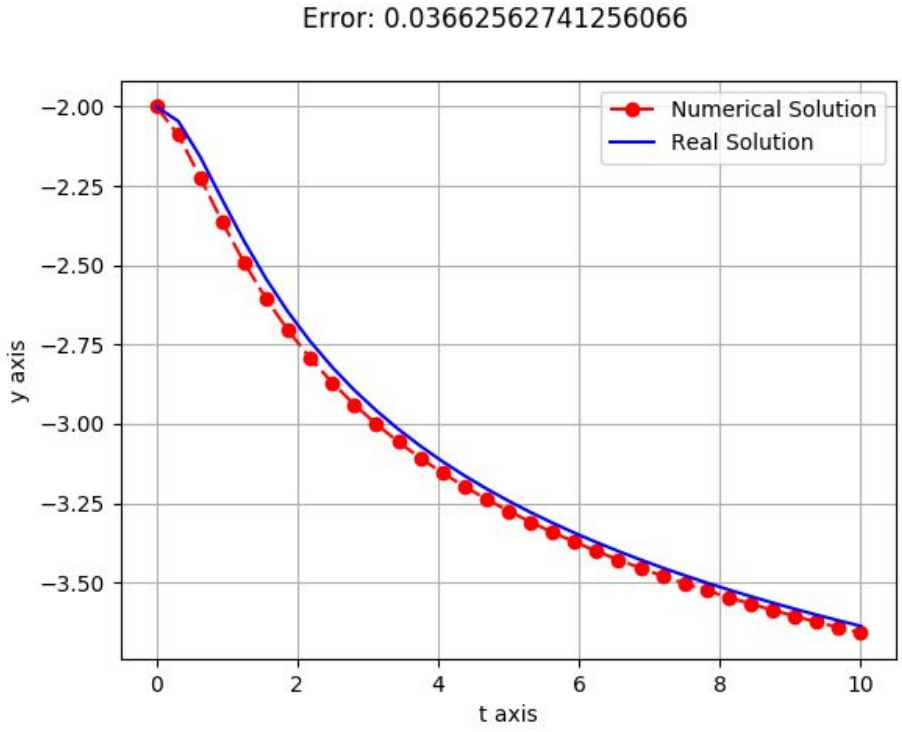
Step Size(N) = 8



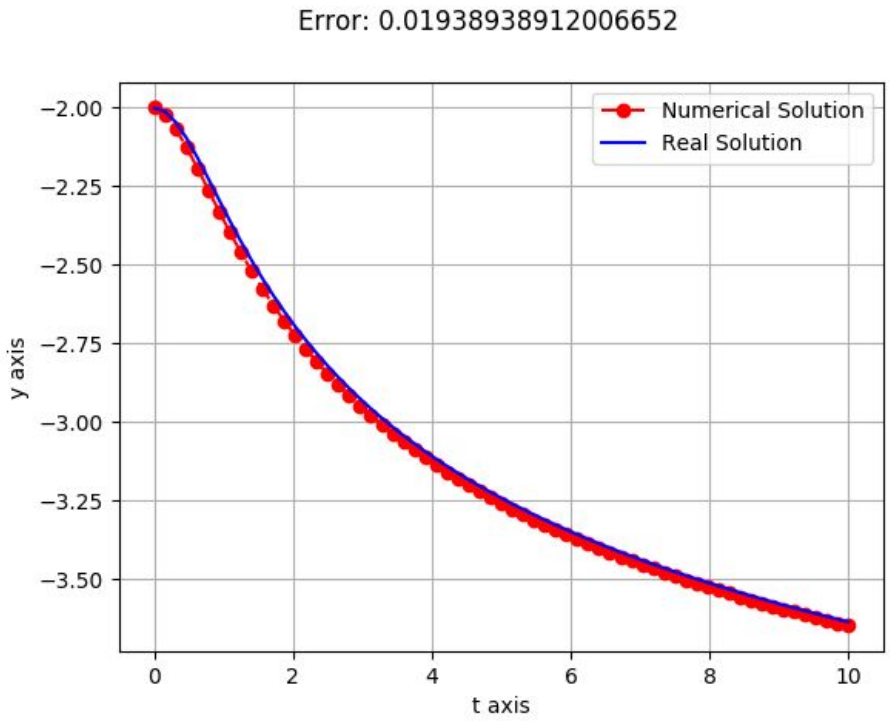
Step Size(N) = 16




Step Size(N) = 32



Step Size(N) = 64



**Findings:**

As expected, the larger the amount of increments allowed, the more accurate the solution to the initial value problem $\frac{dy}{dt} = \frac{2t}{y(1+t^2)}$ with the initial condition $y(0) = (-2)$ becomes.

**Conclusion:**

To conclude, as our $\Delta t$, defined as our step size, increases, the line traced by the grid points, approaches the exact value of the function. The larger the amount of increments allowed, the more accurate the solution to the initial value problem $\frac{dy}{dt} = \frac{2t}{y(1+t^2)}$ with the initial condition $y(0) = (-2)$ becomes. As the step size of our program increases, the error decreases, showing that the accuracy of the calculation is primarily controlled by the increments allowed to calculate the solution. When our step size is set to 64, it can be seen that our real solution closely hugs our numerical solution. In our 3 other graphs, the smaller the step size, the farther the graph drawn starts to stray from the real solution.