Type B: Initial Value Problem (IVB) of a First-Order Ordinary Differential Equation (ODE)

AM129 Foundations of Scientific Computing
Fall 2019
Julia Sales and Anthony Overton

How to run the code:

- First be in the Project/code/pyRun path
- Then run : python pyRun_project.py

Driver/Navigator:

Project	Julia Sales	Anthony Overton
Coding	65% (Main coder)	35% (helped)
Results	50% (Ran tests and debugged)	50% (Ran tests and debugged)
Report	35% (Wrote a portion)	65% (Wrote Majority)
	Driver	Navigator

Abstract:

The main goal of the program was to discover what happens when we increase the number of nodal points in a numerical approximation. As delta t, increase, we saw that the approximation mirrored the function f(t) more closely. For our project, we used the numerical method -Euler's Method- to solve a first-order nonlinear ordinary differential equation (ODE). This program calculated N amount of points, with N being four different values, increasing in size. The program calculates a numerical solution and a real solution and compares the two in the form of graphs. Numerical approximation, more broadly numerical analysis, is a very important field in scientific computing. It allows for realistic modeling of scientific phenomena, everywhere from microbiology to astrophysics.

Methods:

Our project was split into 2 parts. The first part, we had to write Euler's method and the dy/dt function in fortran. The second part was to write a python driver to make/compile, plot, find error, and run our code different N values. For our project, we used the numerical method -Euler's Method- to solve a first-order nonlinear ordinary differential equation (ODE):

$$\begin{cases} \dfrac{dy}{dt} = y'(t) = f(t, y(t)), \\ y(t_0) = y_0. \end{cases}$$

Specifically,

$$\begin{cases} \dfrac{dy}{dt} = \dfrac{2t}{y\left(1+t^2\right)}, \\ y(0) = -2 \end{cases}$$

And compare numerical approximations to the exact function,

$$y(t) = -\sqrt{2\ln\left(t^2+1\right)+4}$$

Part 1:

For the fortran function, we had to take in the t and y values to calculate the value for dy/dt. We had to create two fortran files, one called eueler.F90 and one called main.F90. Eueler.F90, contains the function dy/dt(t, y) which is called by the subroutine eulers_method. Main.F90 acts as a separate driver and calls eulers_method and outputs the .dat files.
The subroutine eulers_method implements the numerical approximation for our given ODE.

- In addition, it produced and output file containing *t* and our numerical solution.

$$0 = t_0 < t_1 < t_2 <\ldots< t_N = 10.$$

$$t_n = n\Delta t, \quad \text{where } \ n = 0, 1, \ldots, N, \ \text{ and } \ \Delta t = \frac{t_N}{N}.$$

- Over the domain [0,10], we used discrete points described by

$$y_n = y(t_n), \quad n = 0, 1, \ldots, N.$$

- Using the derivative of the function, *y'(t)*, and small enough delta t:

$$y'(t) = f(t, y(t)),$$

- we obtained the equation used for approximation:

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx f(t, y(t)).$$

- To get the y values, given y(0) = -2, we use this equation:

$$y_{n+1} = y_n + \Delta t f(t_n, y_n), \quad n = 0, 1, \ldots, N.$$

*Note that *f(t,y)* is given in the problem.

Part 2:

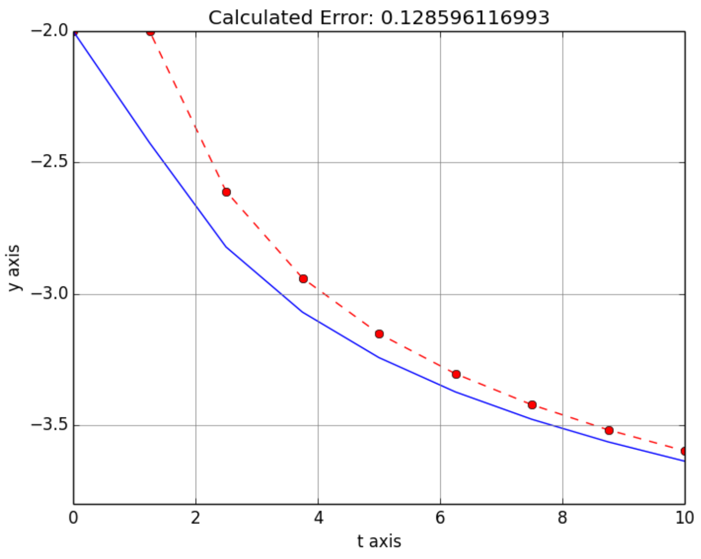
For pyRun in the python folder, we created four different functions and a main.

We had a make function that will compile our code through a make file. It will clean and re-compile if main.exe already exists in the directory. We also had a run function that will run our function by: ./main.exe. The function plot_data plotted the points on a graph and would execute four times, going through all N values.
Our other function was a helper function for calculating the error.

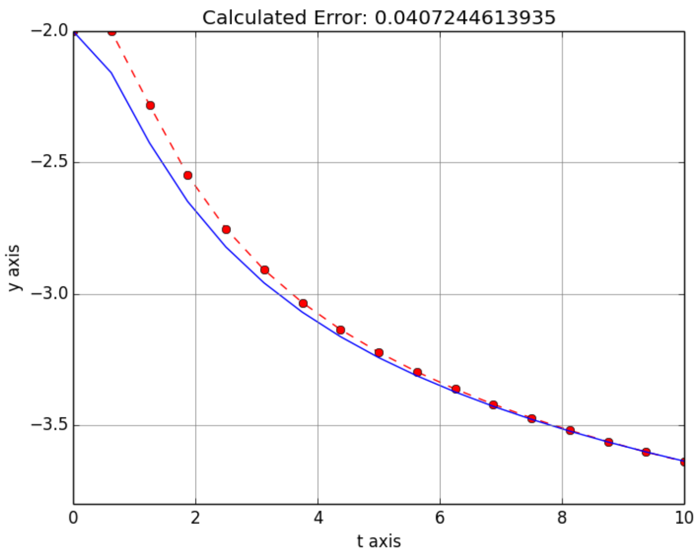
Results:

Upon running our python code, four figure files result_i.png for i = 8,16,32,64, are produced **in a single run**.

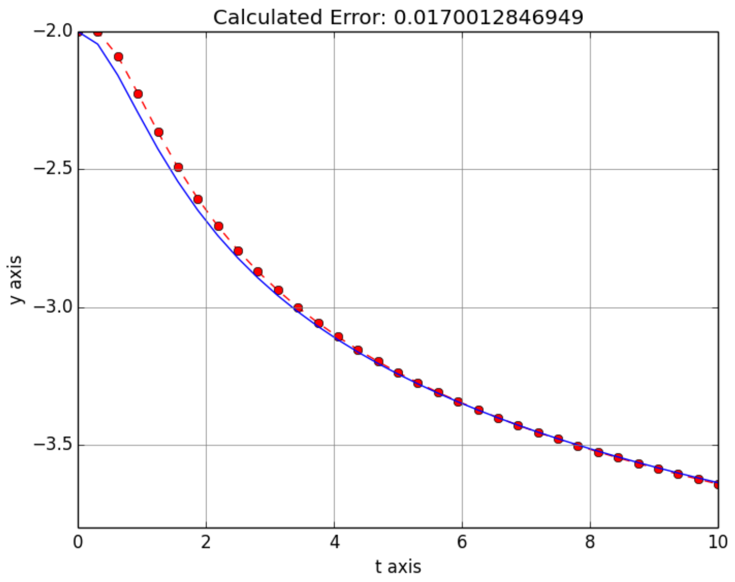
N = 8



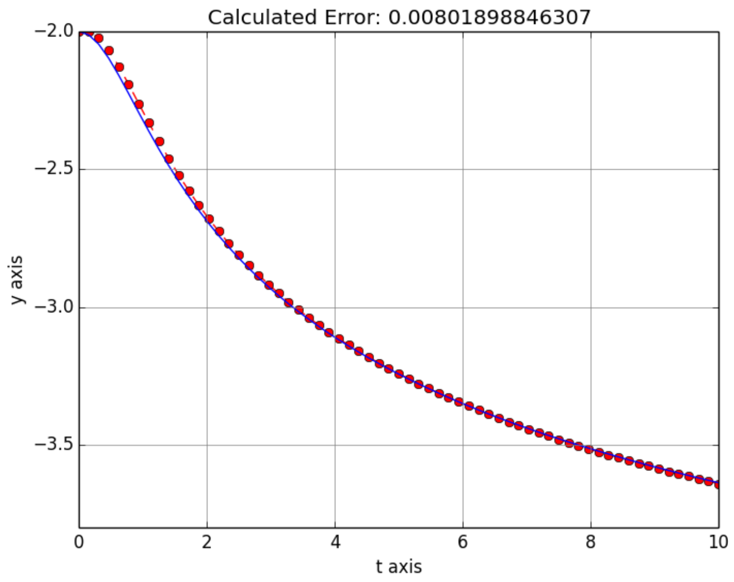
N = 16



N = 32



N = 64



Findings:

We see in our results showed that when delta t decreases, the numerical solution ***converges*** to the exact solution as delta t goes to zero.

- As you can see from the graphs above, as the number of points, N, increases, the more our numerical solutions resembled the real solution.

We see that:

- The more nodal points you have during a numerical approximation, the more accurate your solution will be.
- As N grew, so did our accuracy. N = 64 is clearly more accurate than N = 8.
- The solution to an ODE is a function, or class of functions.
- After executing Euler's Method, the points we collect can be plotted to form a graph (function).

Comments:

These are examples of what our output_i.dat files look like, in order of i=8,16,32,64. We only show 8 and 16.

output_8.dat

```
0      0.00000000  -2.000000000000
1      1.25000000  -2.000000000000
2      2.50000000  -2.609756097561
3      3.75000000  -2.940081589021
4      5.00000000  -3.151778652575
5      6.25000000  -3.304317673796
6      7.50000000  -3.422349787874
7      8.75000000  -3.518047462668
8     10.00000000  -3.598214266402
```

output_16.dat

```
0      0.00000000  -2.000000000000
1      0.62500000  -2.000000000000
2      1.25000000  -2.280898876404
3      1.87500000  -2.548230367453
4      2.50000000  -2.751913342312
5      3.12500000  -2.908544176623
6      3.75000000  -3.033295488769
7      4.37500000  -3.135891269712
8      5.00000000  -3.222478556442
9      5.62500000  -3.297074744182
10     6.25000000  -3.362409651770
11     6.87500000  -3.420406094948
12     7.50000000  -3.472461653352
13     8.12500000  -3.519619973547
14     8.75000000  -3.562678735319
15     9.37500000  -3.602259994453
16    10.00000000  -3.638857397981
```

Conclusion:

Based on our findings and results, it is clear that numerical approximations are better what done with smaller and more frequent delta's (within a fixed domain).

We created fortran subroutines and functions to calculate Euler's method and solve a nonlinear first-order ordinary differential equation. We made a python driver file that compiled, ran, our fortran files. It also plotted our results.

As we saw from the results in our output.dat files, we can see that our numeric results gets closer and closer to the exact result every time N increases. This shows that the more points we have and the bigger N we have, the more exact our graph will be.