CS @ ILLINOIS
COMPUTER SCIENCE

CS 101 James Scholar Project

Introduction to Programming for Engineers and Scientists

Kenneth Tochihara | May 7th, 2019

Introduction

This project focused on numerically solving ordinary differential equations. Exercise E.4 from Appendix E in *A Primer on Scientific Programming with Python* by Hans Petter Langtangen states the following:

> We want to make a program **`odesolver_cml.py`** which accepts an ODE problem to be specified on the command line. The command-line arguments are `f`, `u0`, `dt`, and `T`, where `f` is the right-hand side `f(u, t)` specified as a string formula, *`u0`* is the initial condition, *`dt`* is the time step, and `T` is the final time of the simulation. A fifth optional argument can be given to specify the name of the numerical solution method (set any method of your choice as default value). A curve plot of the solution versus time should be produced and stored in a file **`plot.png`**.
>
> Use `StringFunction` from `scitools.StringFunction` for convenient conversion of a formula on the command line to a Python function.

DEVELOPMENT SOFTWARE

The primary software used to develop this program was Atom. This editor allows for downloadable packages that make the coding experience easier. The program's syntax was checked and avoided wasting time solving these issues.

GitHub was utilized to keep accurate notes on the progress of the project. A package for GitHub was installed onto Atom, allowing for a universal editing environment that allows access to cloud storage.

However, the `scitools` library was not appropriately importing into the specific python version in Atom. Therefore, to avoid problem, all the testing and running the program was done on the terminal on python version 3.7.1.

Program

IMPORTED LIBRARIES AND CONSTANTS

The utilized libraries known were imported in the beginning to avoid any issues figuring out which library is not imported. It is easier to put them in the front to clearly state which libraries are being utilized for this program.

In terms of constants, there is only one dictionary defined that will be utilized to nicely name our titles in the plots produced. A corresponding short string is used to associate it with the actual method utilized for calculation.

DEFINED FUNCTIONS

This project relies on a variety of different numerical methods to compute the solution to ordinary differential equations (ODE). In this section, the Forward-Euler (Equation 1), Midpoint (Equation 2), 2nd Order Runge-Kutta (Equation 3), 4th Order Runge-Kutta (Equation 4), 3rd Order Adams-Bashforth (Equation 9), Midpoint With Iterations (Equation 10), and Backward-Euler (Equation 12) methods are all defined as functions. They all output an array of data points received from these approximation methods. The following equations were taken from the textbook directly.

$$u_{k+1} = u_k + \Delta t\, f(u_k,\, t_k), \quad \Delta t = t_{k+1} - t_k$$

Equation 1: Forward-Euler Method

$$u_{k+1} = u_{k-1} + 2\Delta t\, f(u_k,\, t_k), \quad 2\Delta t = t_{k+1} - t_k$$

Equation 2: Midpoint Method

$$u_{k+1} = u_k + K_2$$

Equation 3: 2nd Order Runge-Kutta Method

$$u_{k+1} = u_k + \frac{1}{6}(K_2 + 2K_2 + 2K_3 + K_4)$$

Equation 4: 4th Order Runge-Kutta Method

$$K_1 = \Delta t\, f(u_k,\, t_k)$$

Equation 5: Supplementary to Runge-Kutta Method

$$K_2 = \Delta t\, f\left(u_k + \frac{1}{2}K_1,\, t_k + \frac{1}{2}\Delta t\right)$$

Equation 6: Supplementary to Runge-Kutta Method

$$K_3 = \Delta t\, f\left(u_k + \frac{1}{2}K_2,\, t_k + \frac{1}{2}\Delta t\right)$$

Equation 7: Supplementary to Runge-Kutta Method

$$K_4 = \Delta t\, f(u_k + K_2,\, t_k + \Delta t)$$

Equation 8: Supplementary to Runge-Kutta Method

$$u_{k+1} = u_k + \frac{\Delta t}{12}(\,23f(u_k,\, t_k) - 16f(u_{k-1},\, t_{k-1}) + 5f(u_{k-2},\, t_{k-2}))$$

Equation 9: 3rd Order Adam-Bashforth Method

$$u_{k+1} = v_N$$

Equation 10: Midpoint With Iterations Method

$$v_q = u_k + \frac{1}{2}\Delta t\left(f\left(v_{q-1}, t_{k+1}\right) + f(u_k, t_k)\right), \quad q = 1, \dots, N, \quad v_0 = u_k$$

Equation 11: Supplementary to Midpoint With Iterations Method

$$u_{k+1} = u_k + \Delta t\, f(u_{k+1}, t_{k+1}), \quad \Delta t = t_{k+1} - t_k$$

Equation 12: Backward-Euler Method

USER INTERFACE LOOP

The software includes a user interface loop that allows the user to run multiple equations in one session. Each iteration can run a completely new problem, taking in new initial values each time. The infinite while loop can be ended with a `KeyboardInterrupt` error which will safely end the program since the while loop contains an exception handler.

Once the user begins to run the program, they will have the ability to control numerous variables outlined in the prompt. Many of these variables will be considered global variables so that it will not be necessary to pass through any variables to the functions. One of these inputs is `f(u, t)`, a function defined as the right-hand-side of the equation in question. This information will be logged by the `StringFunction` method allowing for the user to type in any string to get an equation saved in `f`. Then, the user will enter a short string that represents the method desired to solve the equation.

The program will then take the data and initialize variables such as `arrT` and then encounter an if statement that will determine what function will be utilized based on the method desired by the user. The necessary variables will then be used to compute `arrU` which represents the data points from the specific method across `arrT`.

SAVING THE DATA

After the arrays are established, the program will put together the plot given from passing through the functions. These data points, as they approximate are plotted onto the graph and shown to the user using `plt.show()`. This will act as a pause point to allow for the user to confirm their graph before moving on with the program.

Each figure was plotted within the limits of the maximum `y` values which the graphs were titled according to the method. Seen in the figure below, the specific file was stored into directory **`Saved-Figures`** and recorded the time using the `datetime` library. Here is an example of a saved figure:

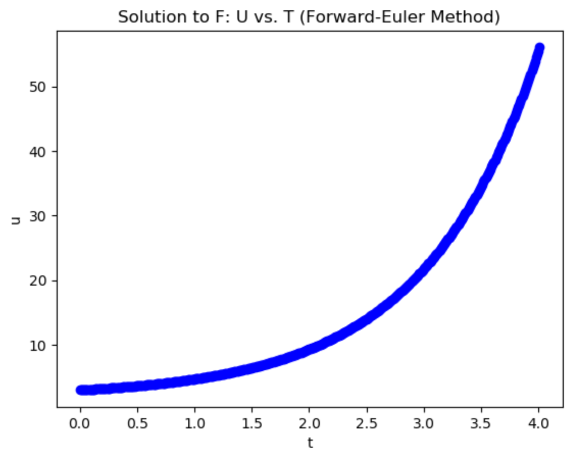

Figure 1: Plot of the Solution to f(u, t) = u

Once the figure is closed, the program will continue on to save and clear the figure. Since this program is running on an infinite loop, the figure must be cleared so that the data from the previous will not be seen in the next iteration. Therefore, clearing the figure allows for the user to start on a completely fresh slate and begin to solve a new ordinary differential equation.

Conclusion

This program utilized numerous concepts learned in CS 101 by implementing libraries, functions, loops, plotting and exception handlers. It was rather simple to implement the problem but still took time to debug and perfect the final product. Overall, this project was a great experience taking a problem and implementing it using python.