Franklin Smith Math 252 Written assignment
Programming Euler's Method with Python

In calculus we use Euler's method to numerically approximate the solution to an often troublesome first order initial value problem

y' = f(x,y), y(x0) = y0

for a table of values. To begin, it is trivial for us to choose the interval [x0,xf] that we want to find a solution on, in addition to the number of points n that we are going to approximate in that interval. Then getting \Delta x = (xf-x0)/(n-1) we have n evenly spaced points x0, x1, ..., xn, with xj = x0 + j * \Deltax. Then we have to fill in the value of y(xi) in the table.

x	y
x0	y0
x1	y(x1)
x2	y(x2)
...	...
xn	y(xn)

Since we do not initially know the value of y(x), I can only fill in the table with approximate values of y at each x-point.

The algorithm of Euler's method is as follows. Since we know that y(x0) = y0, the differential equal tells us also that y'(x0) = f(x0,y0). So using the tangent line approximation of the unknown function we get that,
y(x) = y'(x0)(x-x0) + y(x0) = (x-x0) f(x0,y0) + y0. for x near x0.
Using the tangent line approximation we end up with
y(x1)=(x1-xo) f(x0,y0)+ y0 = \Delta x f(x0,y0) + y0.

So y(x1) is approximately equal to y1:= \Delta x f(x0,y0) + y0. Since y(x1) = y1, the differential equation tells us that y'(x1) = f(x1,y(x1) = f(x1,y1). So we have the tangent line approximation

y(x) = y'(x1)(x-x1) + y(x1) = (x-x1) f (x1,y1) + y1, for x close to x1.

So using the tangent line approximation, we have
y(x2) = (x2-x1) f(x1,y1) + y1 = \Delta f(x1,y1) + y1.

So y(x2) is equal to y2 := \Delta x f(x1,y1) + y1. Continuing in this way, we obtain approximations y(xj) yj, where the yj are defined recursively by the equation

yj+1 := \Deltax f(xj, yj) + yj

The table of values at this point

x	*y*
x0	*y0*
x1	*y1*
…	…
xn	*yn*

this gives us our approximation to the solution of the differential equation, which is the Euler's Method.

Programming Euler's Method in Python
Buckle Up and hold on
For starters we know that the accuracy of Euler's method depends on the number of points that we choose in the interval [x0, xf], as well as the size of the interval [x0,xf]. If you want to approximate the solution for a longer time, then you have to increase the number of points we are approximating in order to keep your approximation solution accurate. The point of programming this is so we don't have to do all of the calculations by hand.

In the first part of our python file were going to start coding, the first two lines of code is

```
import bumpy as np
from matplotlib import pyplot as pet
```

This part of code includes python packages and libraries that the creators of python have written for us. numpy is a big library of routines for numerical calculation in python, and marplot lib has plotting packages for making beautiful graphs. Under this code we shall start using math for our numerical approximation to the initial value problem.
For this example, our initial value problem will be
y' = -y + sin(x), y(0) = 1.

We use Euler's method to approximate the value of the function in the interval [0,10] with 101 points. Then x0 = 0, y0 = 1, xf = 10, n = 101, and \Deltax = (xf - x0)/(n-1) = 0.1.
The below code tells the computer this information.

```
x0 = 0
y0 =1
xf = 10
```

```
n = 101
deltax = (xf-x0)/(n-1)
```

After this we are going to create a vector of length 101 who's entries are our x-values. To do this we enter the code:

```
x = np.linespace(x0, xf, n)
```

Now our program has a vector called x, given by x = [0.0, 0.1, 0.2, ... , 10.0]. The next thing we want to do is tell the computer how to generate the y-values. Now we create an array to put the y-values into.

```
y = np.zeros([n])
```

This makes an array y of size n where all the entries are 0.0. The next thing we have to do is fix this so that the entries are the estimates given by the (forward) Euler method. In order to do this we use a for loop, which is a way to tell our program to iterate an instruction for a number of times

```
y[0] = y0
for i in range(1,n):
        y[i] = (-y[i-1] + np.sin(x[i-1])) + y[i-1]
```

The first line of the above code sets the first item of the array y to the value y0. The loop then sets the ith entry of the array to the (i-1)th entry plus \Deltax times our estimate of the derivative from the differential equation. The array y now has the estimates of the solution from Euler's method.
We also use a for loop to print out the data we made in a chill way

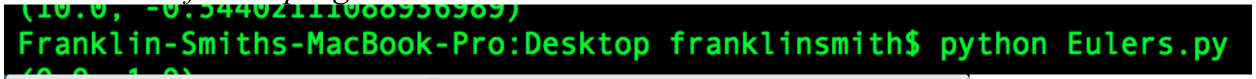
```
For i in range (n):
        print(x[i], y[i])
```

Next to make a nice graph of our data we use the plot function from matplotlib

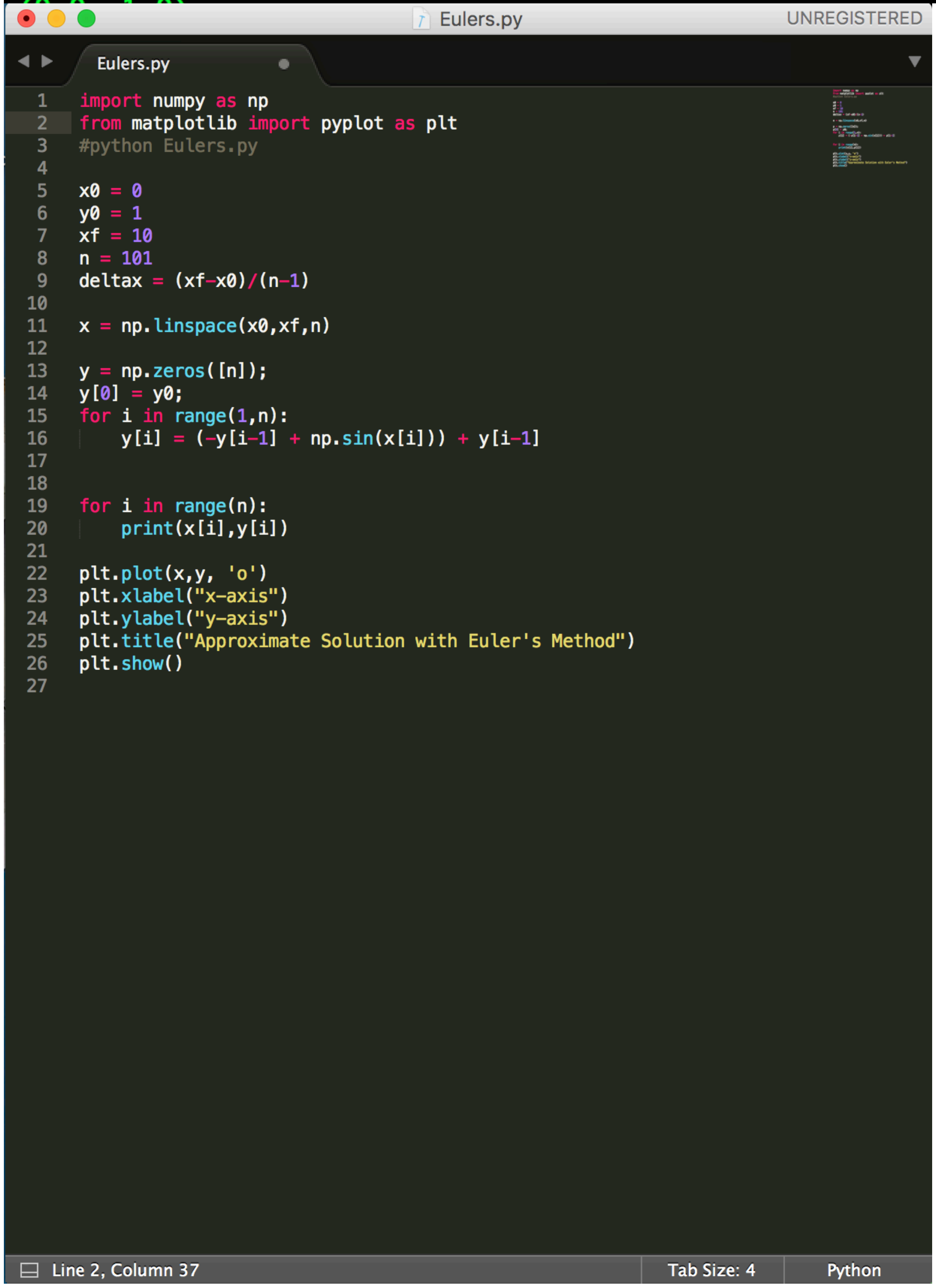
```
plt.plot(x,y, 'o')
plt.xlabel("Value of X")
plt.ylabel("Value of Y")
plt.title("Approximate solution with Euler's Method")
plt.show()
```

If anyone reading this paper is too lazy to code it yourself. Download the python source code file from this link.
https://github.com/fsmith503/ComputerScience/blob/master/EulersMethodCalc2Python/Eulers.py
To execute the python file on your own computer
1. Download the python file in your Desktop
2. Open computer terminal

3. Terminal command: $ cd Desktop
4. Verify file is there Terminal command: $ ls
5. Execute Terminal command: $python Eulers.py
6. Grab popcorn, sit back and watch the show
Our entire software program is:



```
(10.0, -0.54402111088936989)
Franklin-Smiths-MacBook-Pro:Desktop franklinsmith$ python Eulers.py
(0.0, 1.0)
```



```python
import numpy as np
from matplotlib import pyplot as plt
#python Eulers.py

x0 = 0
y0 = 1
xf = 10
n = 101
deltax = (xf-x0)/(n-1)

x = np.linspace(x0,xf,n)

y = np.zeros([n]);
y[0] = y0;
for i in range(1,n):
    y[i] = (-y[i-1] + np.sin(x[i])) + y[i-1]


for i in range(n):
    print(x[i],y[i])

plt.plot(x,y, 'o')
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("Approximate Solution with Euler's Method")
plt.show()

```

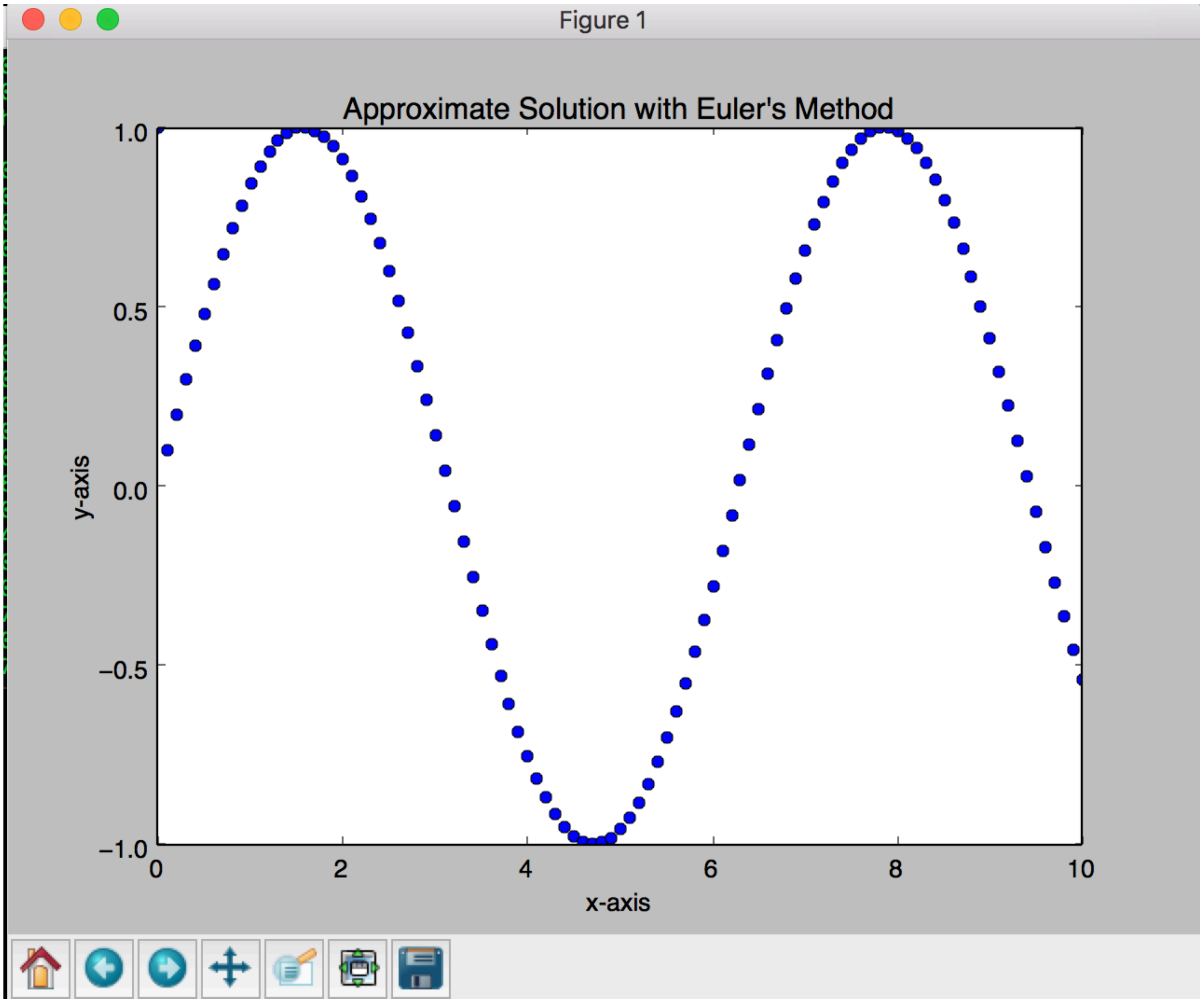


Figure 1
Approximate Solution with Euler's Method
1.0
0.5
0.0
−0.5
−1.0
y-axis
0
2
4
6
8
10
x-axis