Euler's Method with Python

Intro. to Differential Equations

February 1, 2016

1 Euler's Method with Python

1.1 Euler's Method

We first recall Euler's method for numerically approximating the solution of a first-order initial value problem

$$y' = f(x, y),\ y(x_0) = y_0$$

as a table of values. To start, we must decide the interval $[x_0, x_f]$ that we want to find a solution on, as well as the number of points n that we wish to approximate in that interval. Then taking $\Delta x = (x_f - x_0)/(n-1)$ we have n evenly spaced points $x_0, x_1, \ldots, x_n$, with $x_j = x_0 + j \cdot \Delta x$. Then our objective is then to fill in the values of $y(x_i)$ in the table below.

x	y
x_0	y_0
x_1	$y(x_1)$
x_2	$y(x_2)$
$\vdots$	$\vdots$
x_n	$y(x_n)$

Of course since we don't "know" the exact value of $y(x)$, we'll only be able to fill in the table with approximate values of y at each x-point.

The strategy of Euler's method is as follows. Since we know that $y(x_0) = y_0$, the differential equation also tells us $y'(x_0) = f(x_0, y_0)$. Therefore using a tangent line approximation of the unknown function, we have that

$$y(x) \approx y'(x_0)(x - x_0) + y(x_0) = (x - x_0)f(x_0, y_0) + y_0, \quad \text{for } x \text{ close to } x_0.$$

Using this tangent line approximation, we approximate that

$$y(x_1) \approx (x_1 - x_0)f(x_0, y_0) + y_0 = \Delta x f(x_0, y_0) + y_0.$$

Therefore $y(x_1)$ is approximately equal to $y_1 := \Delta x f(x_0, y_0) + y_0$. Now since $y(x_1) \approx y_1$, the differential equation also tells us $y'(x_1) = f(x_1, y(x_1)) \approx f(x_1, y_1)$. Thus again we have a tangent line approximation

$$y(x) \approx y'(x_1)(x - x_1) + y(x_1) \approx (x - x_1)f(x_1, y_1) + y_1, \quad \text{for } x \text{ close to } x_1.$$

Using this tangent line approximation, we approximate that

$$y(x_2) \approx (x_2 - x_1)f(x_1, y_1) + y_1 = \Delta x f(x_1, y_1) + y_1.$$

Therefore $y(x_2)$ is approximately equal to $y_2 := \Delta x f(x_1, y_1) + y_1$. Continuing in this way, we obtain approximations $y(x_j) \approx y_j$, where the y_j are defined *recursively* by the equation

$$y_{j+1} := \Delta x f(x_j, y_j) + y_j.$$

The table of values

x	y
x_0	y_0
x_1	y_1
$\vdots$	$\vdots$
x_n	y_n

defined in this way then gives us our approximation to the solution of the differential equation. This is the (forward) **Euler's method**.

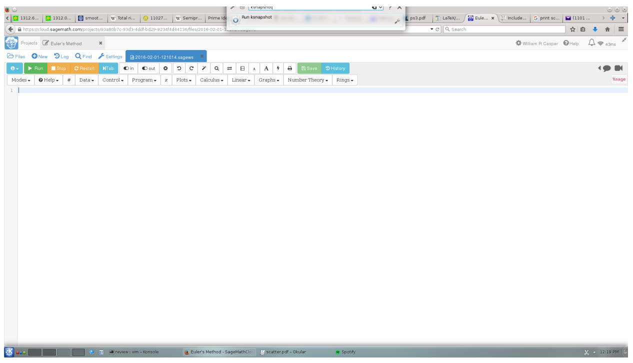
1.2 Implementing Euler's Method with Python

The accuracy of Euler's method depends highly on the number of points that you choose in the interval $[x_0, x_f]$, as well as the size of the interval $[x_0, x_f]$. If you want to approximate the solution for a longer time, then you need to increase the number of points you approximate, in order to keep your approximate solution accurate. Doing so many calculations by hand would be quite a burdensome task, so we instead look to a computer for some assistance. In this section, we will demonstrate how to implement Euler's method using python. We assume that users have a computer with a web browser – it is not necessary to have python installed on your system!

The first thing we want to do is go to

```
https://cloud.sagemath.com/
```

and create an account. Then create a new project – call it whatever you want, such as "Euler's Method". Click on the created project and create a new Sage worksheet by selecting the appropriate item out of the drop-down menu. You should now be presented with a page which looks like this one.



Now in the big blank part of the page, we're going to enter our first bit of code. The code is

Listing 1: Importing External Libraries

```
import numpy as np
from matplotlib import pyplot as plt
```

This bit of code includes some fancy packages and things that other people have written for us. In particular, numpy is a huge library of routines for numerical calculation in python, and matplotlib has plotting packages for making pretty graphs. Under this bit of code, we will find our numerical approximation to an initial value problem. For the purposes of this demonstration, let's suppose that our initial value problem is

$$y' = -y + \sin(x), \quad y(0) = 1.$$

Let's use Euler's method to approximate the value of the function in the interval $[0, 10]$ with 101 points. Then $x_0 = 0$, $y_0 = 1$, $x_f = 10$, $n = 101$, and $\Delta x = (x_f - x_0)/(n-1) = 0.1$. The following code tells the computer this information.

Listing 2: Defining Basic Data

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

We next want to create a vector of length 101 whose entries are our x-values. To do this, we enter the following code

Listing 3: Defining x-values

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

Now the computer has a vector called x, given by $x = [0.0, 0.1, 0.2, \dots, 10.0]$. The next thing we want to do is tell the computer how to generate the y-values. The first thing we do is create an array to store the y-values in.

Listing 4: Initializing Array for y-values

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

This creates an array y with n entries that are all 0.0. The next thing we want to do is fix this so that the entries are the estimates given by the (forward) Euler method. To do this, we use a **for loop**, which is a way of telling the computer to repeat an instruction a certain number of times:

Listing 5: For Loop for Euler's Method

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

The first line sets the first entry of the array y to the value $y0$. The loop then sets the i'th entry of the array to the $(i-1)$'th entry plus Δx 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 can also use a for loop to print out the data that we've generated in a pretty way

Listing 6: Printing the Data

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

Finally, to make a pretty graph of our data, we can use the plot function from matplotlib.

Listing 7: Plotting the Solution

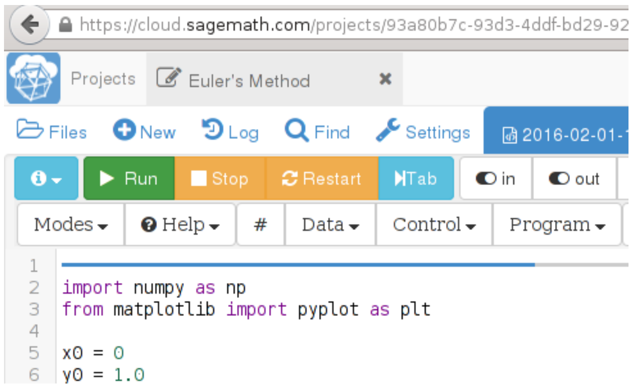
```
plt.plot(x,y,'o')
plt.xlabel("Value of x")
plt.ylabel("Value of y")
plt.title("Approximate Solution with Forward Euler's Method")
plt.show()
```

In summary, our code is:

Listing 8: Summary of Code

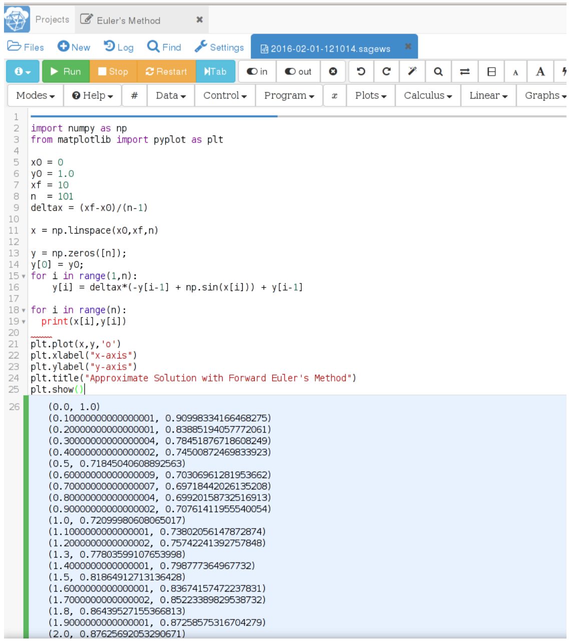
```
import numpy as np
from matplotlib import pyplot as plt

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] = deltax*(-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("Value of x")
plt.ylabel("Value of y")
plt.title("Approximate Solution with Forward Euler's Method")
plt.show()
```

To run the code, we hit the green arrow that says "Run" at the top of the screen.



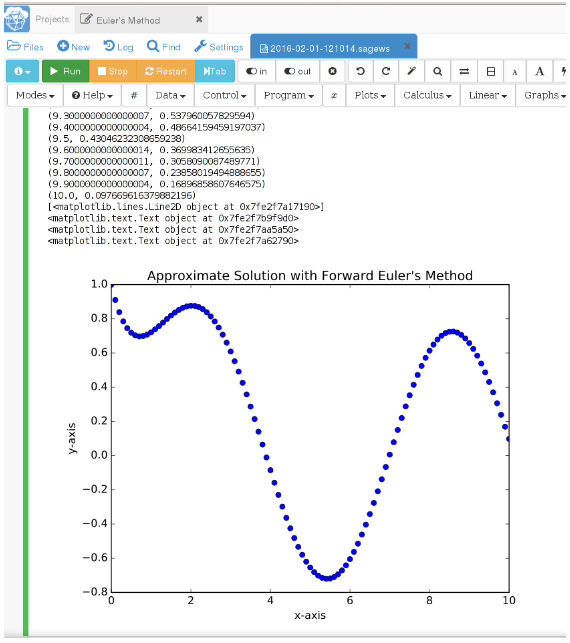
```
import numpy as np
from matplotlib import pyplot as plt

x0 = 0
y0 = 1.0
```

The result should be a bunch of data which is spit out plus a pretty graph a the end



```
import numpy as np
from matplotlib import pyplot as plt

x0 = 0
y0 = 1.0
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] = deltax*(-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 Forward Euler's Method")
plt.show()
```

```
(0.0, 1.0)
(0.10000000000000001, 0.90998334166468275)
(0.20000000000000001, 0.83885194057772061)
(0.30000000000000004, 0.78451876718608249)
(0.40000000000000002, 0.74500872469833923)
(0.5, 0.71845040608892563)
(0.60000000000000009, 0.70306961281953662)
(0.70000000000000007, 0.69718442026135208)
(0.80000000000000004, 0.69920158732516913)
(0.90000000000000002, 0.70761411955540054)
(1.0, 0.72099980608065017)
(1.1000000000000001, 0.73802056147872874)
(1.2000000000000002, 0.75742241392757848)
(1.3, 0.77803599107653998)
(1.4000000000000001, 0.798777364967732)
(1.5, 0.81864912713136428)
(1.6000000000000001, 0.83674157472237831)
(1.7000000000000002, 0.85223389829538732)
(1.8, 0.86439527155366813)
(1.9000000000000001, 0.87258575316704279)
(2.0, 0.87625692053290671)
```



```
(9.3000000000000007, 0.537960057829594)
(9.4000000000000004, 0.48664159459197037)
(9.5, 0.43046232308659238)
(9.6000000000000014, 0.369983412655635)
(9.7000000000000011, 0.3058090087489771)
(9.8000000000000007, 0.23858019494888655)
(9.9000000000000004, 0.16896858607646575)
(10.0, 0.097669616379882196)
[<matplotlib.lines.Line2D object at 0x7fe2f7a17190>]
<matplotlib.text.Text object at 0x7fe2f7b9f9d0>
<matplotlib.text.Text object at 0x7fe2f7aa5a50>
<matplotlib.text.Text object at 0x7fe2f7a62790>
```