Euler's Method

Amanda Rojas

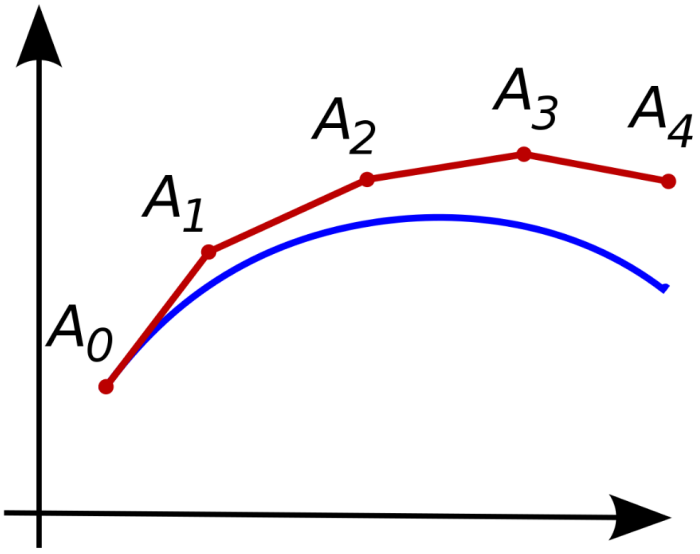
Fall 2020

Abstract

In this lab we study Euler's Method, which is another technique used to study differential equations. Euler's Method uses the idea of linear approximation in order to estimate the solution to an ODE given initial conditions.

1 Introduction

Euler's method is based on approximation and therefore has a certain margin of error. The local error, or the error per step, is directly proportional to the global error, or the error at a given time.
Euler's method is a first-order method and is commonly called a predictor-corrector method. It is commonly used as a skeleton off of which more complex methods are based.
It is commonly used in nonlinear, dynamical systems where calculations are impossible to make. In those situations an approximation is most useful in order to gain some prediction of the model in question.



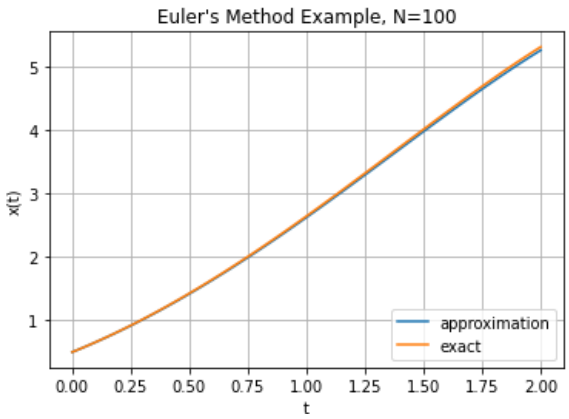
2 Theory

Euler's method is defined by the following equation:

$$x_{n+1} = x_n + hf(t_n, x_n) \tag{1}$$

You begin by computing the function f(t_0, x_0), or$f(t, x)$.
This results in the slope of the tangential line at point (t, x).
Next you multiply this slope by h, which is the step size, or change in t. This results in a change in the x value and a new value to repeat the process. A common thought is to decrease h in order to reduce the step size. This is beneficial except for the fact that it increases the necessary computations required to estimate the result. The python algorithm for Euler's method is so accurate that one can barely tell the difference between the approximation and the exact result of the function.

```
[12] import numpy as np

     # Euler's method
     def euler_mthd( f, a, b, N, IV ):
         h = (b-a)/float(N)          # determine step-size
         t = np.arange( a, b+h, h )  # create mesh
         x = np.zeros((N+1,))        # initialize x
         t[0], x[0] = IV             # set initial values
         for i in range(1,N+1):      # apply Euler's method
             x[i] = x[i-1] + h * f( t[i-1], x[i-1] )
         return t, x                 # Return mesh and solution
```



3 Theory Cont.

There are modifications to Euler's method that have made it more accurate and reduce the power needed to compute with said accuracy. Below are example of modifications to the Euler method:

The Trapezoid Rule:

$$x_{k+1} = x_k + \frac{1}{2}h[f(t_k, x_k) + f(t_k + h, x_k f(t_k, x_k))] \tag{2}$$

The Modified Euler Method:

$$x_0 = \alpha \tag{3}$$

$$x_{i+1} = x_i + \frac{1}{2}h[f(t_i, x_i) + f(t_{i+1}, x_I + hf(t_i, x_i))] \tag{4}$$

```
[18] import numpy as np

     def modified_euler_method( f, a, b, N, IV ):
         h = (b-a)/float(N)          # determine step-size
         t = np.arange( a, b+h, h )  # create mesh
         x = np.zeros((N+1,))        # initialize x
         t[0], x[0] = IV             # set initial values
         for i in range(1,N+1):      # apply Modified Euler Method
             f1 = f( t[i-1], x[i-1] )
             f2 = f( t[i], x[i-1] + h * f1 )
             x[i] = x[i-1] + h * ( f1 + f2 ) / 2.0
         return t, x                 # Return mesh and solution
```

The Heun Method:

$$x_0 = \alpha \tag{5}$$

$$x_{i+1} = x_i + \frac{h}{4}[f(t_i, x_i) + 3f(t_i + \frac{2}{3}h, x_i + \frac{2}{3}hf(t_i, x_i))] \tag{6}$$

```
[19] import numpy as np

     def heun_method( f, a, b, N, IV ):
         h = (b-a)/float(N)          # determine step-size
         t = np.arange( a, b+h, h )  # create mesh
         x = np.zeros((N+1,))        # initialize x
         t[0], x[0] = IV             # set initial values
         for i in range(1,N+1):      # apply Heun's Method
             f1 = f( t[i-1], x[i-1] )
             f2 = f( t[i-1] + (2.0/3.0) * h, \
                     x[i-1] + (2.0/3.0) * h * f1 )
             x[i] = x[i-1] + h * ( f1 + 3.0 * f2 ) / 4.0
         return t, x                 # Return mesh and solution
```

The Runge-Kutta Method:

$$x_0 = \alpha \tag{7}$$

$$k_1 = hf(t_i, x_i) \tag{8}$$

$$k_2 = hf(t_i + \frac{h}{2}, x_i + \frac{1}{2}k_i) \tag{9}$$

$$k_3 = hf(t_i + \frac{h}{2}, x_i + \frac{1}{2}k_2) \tag{10}$$

$$k_4 = hf(t_{i+1}x_i + k_3) \tag{11}$$

$$x_{i+1} = x_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{12}$$

4 Procedures

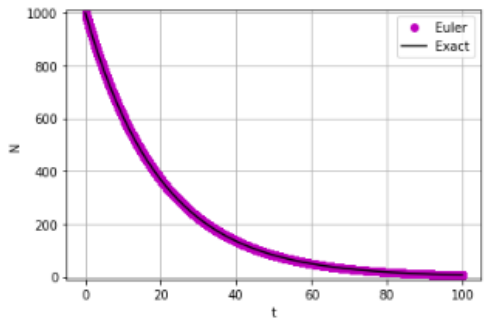
$$y_{n+1} = y_n + hf(t_n, y_n).$$

```
# example first order deq using euler method
# radioactive decay
import numpy as np
import math
import matplotlib.pyplot as plt
```

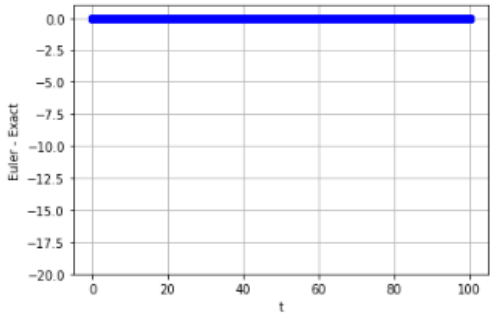
```
[3] # set parameters
    k = 0.05
    N0 = 1000.
    dt = 0.001
    tfinal = 100.

    # initialize array of time t
    t = np.arange(0.,tfinal+dt,dt)
    npoints = len(t)

    # initialize arrays for solution
    N = np.zeros(npoints)
    NExact = N0 * np.exp(-k*t) # compute exact solution for comparison
```

```
[4] # Euler Solution
    N[0] = N0
    for i in range(npoints-1):
        N[i+1] = N[i] - k*N[i]*dt
```

```
plt.plot(t,N,'mo',label='Euler')
plt.plot(t,NExact,'k',label='Exact')
plt.xlabel('t')
plt.ylabel('N')
plt.legend()
plt.axis([-5,105,-10,1010])
plt.grid('on')
```

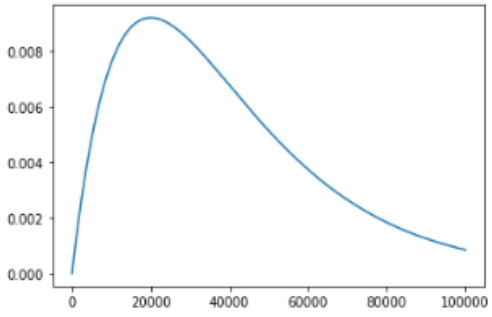



```
[6] plt.plot(t,N-NExact,'bo')
    plt.xlabel('t')
    plt.ylabel('Euler - Exact')
    plt.axis([-5,105,-20,1])
    plt.grid('on')
```



```
plt.plot(np.abs(N-NExact))
```

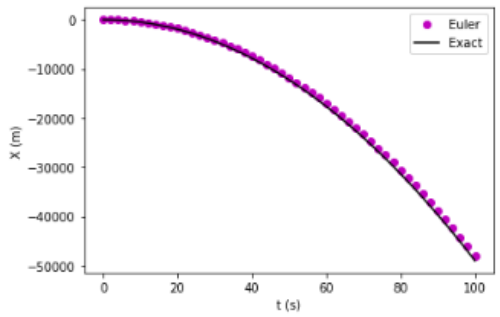
```
[<matplotlib.lines.Line2D at 0x7fb0b513cc50>]
```




```
[8]  # second order deq example
     # falling ball

     import numpy as np
     import matplotlib.pyplot as pl

     # set parameters
     g = 9.8
     X0 = 0.
     V0 = 0.
     dt  = 2
     tfinal = 100.

     # initialize arrays
     t = np.arange(0.,tfinal+dt,dt)
     npoints = len(t)
     X = np.zeros(npoints)
     V = np.zeros(npoints)
     XExact = X0 + V0*t - 0.5*g*t**2

     # Euler's method solution
     X[0] = X0
     V[0] = V0
     for i in range(npoints-1):
         X[i+1] = X[i] + V[i]*dt
         V[i+1] = V[i] - g*dt
```

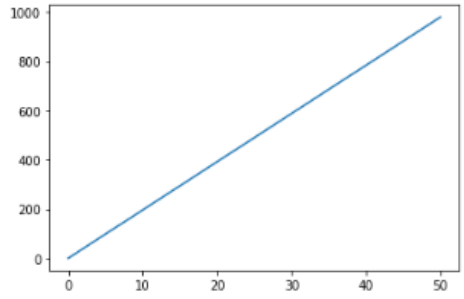
```
     # plot the results
     plt.ion()
     plt.plot(t,X,'mo',label='Euler')
     plt.plot(t,XExact,'k',label='Exact')
     plt.xlabel('t (s)')
     plt.ylabel('X (m)')
     plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb0b4c109e8>
```




```
[9]  plt.plot(np.abs(X-XExact))
```

```
[<matplotlib.lines.Line2D at 0x7fb0b4b79f98>]
```



5 Conclusions

Euler's method is a valuable computational tool for calculating derivatives giving initial conditions. In order to increase accuracy while maintaining the computational power needed, one must implement different modifications to Euler's method, like the ones we discussed here. Ultimately Euler's method is an important tool for accurately solving differential equations.