NUMERICALLY SOLVING ODES

Author: John Hu

Date Created: 05/25/22
Date Edited: 05/25/22
Version 1.0

Abstract

In this SOP, the principle of operation of solving ODEs is discussed. The relevance of ODEs is discussed as well as the two major types which include Euler's Method and subsequent Runge Kutta methods. These have applications ranging from simple harmonic motion to fluid mechanics.

Contents

1 Description of ODE

The ODE is given as an IVP in the form of

$$\frac{dy}{dx} = f(t, y) \text{ and } y(t) = y_0.$$

Note that the functions are only dependent on one variable t rather than multiple variables in a PDE. The order of an ODE refers to the highest value derivative in the function. For example, the simple harmonic motion equation

$$\frac{d^2x}{d^2t} + \omega^2 x = 0$$

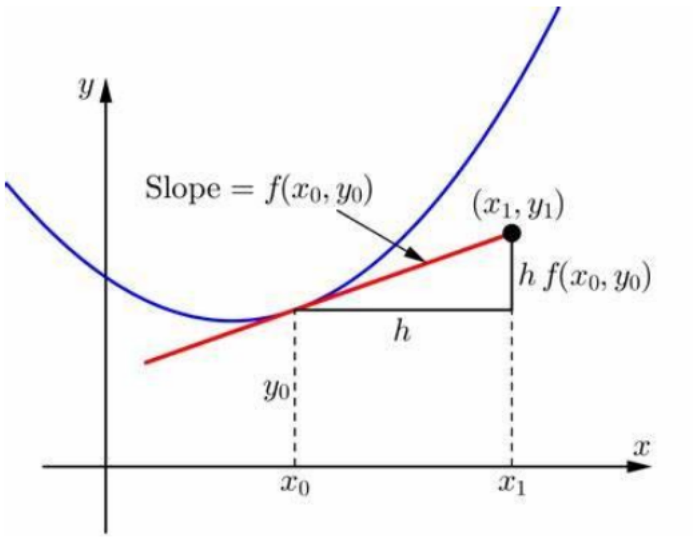
is a second order differential equation. When the order is greater than 1, it is possible that there will no longer be a closed form solution to the differential equation.

2 Description of Euler's Method

2.1 Forward Euler's Method

TThe simplest form of Euler's Method is the forward Euler's method computes the value given an IVP in terms of x. Then by defining a step size h, solving the derivative y′ and adding the product of the derivative and the x value times the step size. Reducing the step size h can yield improved results with the analytical solution by reducing the truncation error. By repeatedly taking the slope of the tangent line, the function can be approximated by multiplying the slope of the secant and the step size.[1]

$$y \approx y_0 + f(t_0, y_0)(t - t_0)$$



2.2 Backwards Euler's Method

Backwards Euler's method follows the same procedure but calculates the derivative based on the previous values instead of the subsequent values.

$$y_{n+1} \approx y_n + hf(t_{n+1}, y_{n+1})$$

3 Solving First Order ODEs with Euler's Method

Example: It is well known that the exact solution to the differential equation

$$\frac{dy}{dx} = e^{-x}$$

yields the solution $y = e^{-x}$. Find the value when $y(0) = -1$.

Using the data processing libraries *numpy* and *matplotlib* with Python.

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

Define the initial IVP as well as the value of h.

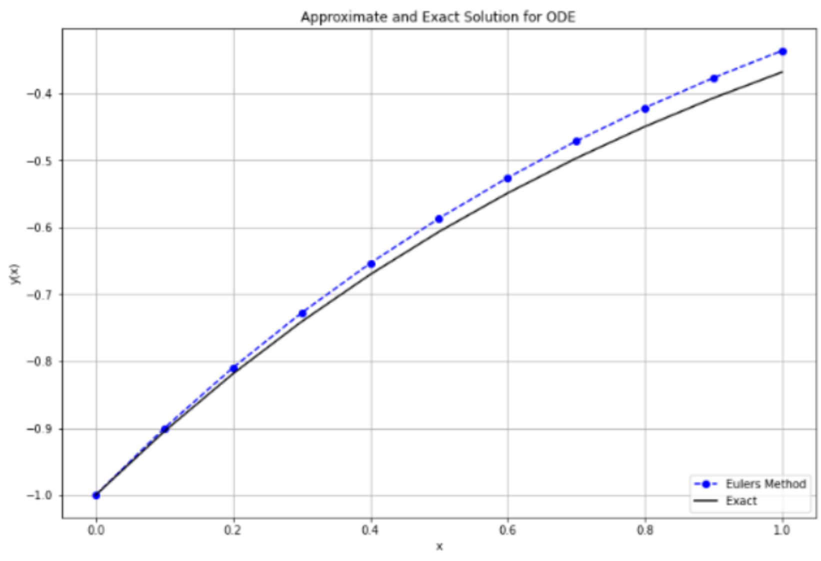
```
f = lambda x,y:np.exp(-x)
h=0.1
x=np.arrange(0,1+h,h)
y0=-1

y=np.zeros(len(x))
y[0]=y0
```

Define the recursive function using the above formula with Forward Euler's Method.

```
for i in range(0,len(x)-1):
    y[i+1]=y[i]+h*f(x[i],y[i])
```

Finally plotting out the function using plt.plot()

```
plt.figure(figsize=(12,8))
plt.plot(t,y,'bo--',label='Eulers Method')
plt.plot(t,-np.exp(-t),'k',label-'Exact')
plt.title('Approximate and Exact Solution for ODE')
plt.xlabel('x')
plt.ylabel('y(x)')
plt.grid()
plt.legend(loc='lower right')
plt.show()
```



Alternatively, using the Scipy odeint library can simplify this process while providing additional functionality to deal with higher order differential equations.

Importing scipy odeint with Python.

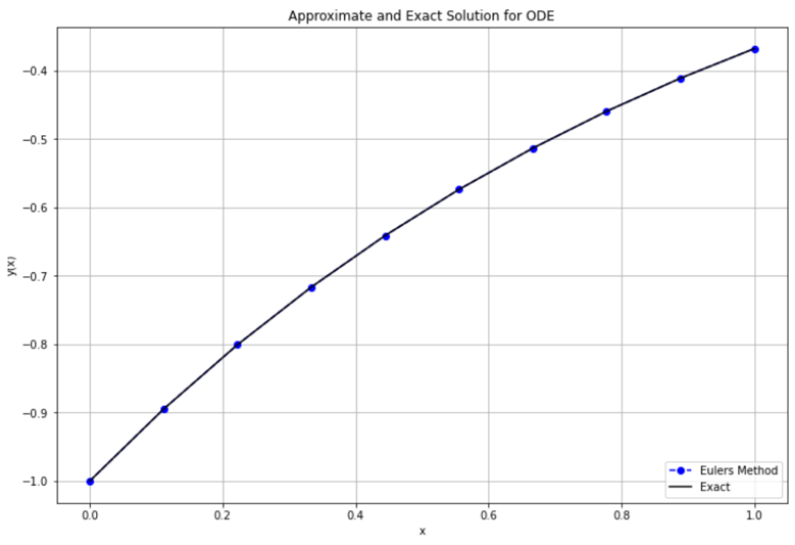
```
from scipy.integrate import odeint
```

Defining a function with y in front of x

```
def f(y,x):
    return np.exp(-x)

x=np.linspace(0,1,10)
y0=-1
y=odeint(f,y0,x)
```

Solving and plotting out the solution.

```
plt.figure(figsize=(12,8))
plt.plot(x,y,'bo--',label='Eulers Method')
plt.plot(x,-np.exp(-x),'k',label-'Exact')
plt.title('Approximate and Exact Solution for ODE')
plt.xlabel('x')
plt.ylabel('y(x)')
plt.grid()
plt.legend(loc='lower right')
plt.show()
```



4 Solving Second Order ODEs with Euler's Method

A second order ODE is given as an IVP in the form of

$$\frac{dy}{dx} = f(t,y), \frac{d^2y}{dx^2} = f(t,y,\frac{dy}{dx}) \text{ and } y(t) = y_0 \text{ and } \frac{dy}{dx}|x = 0 = y_0'$$

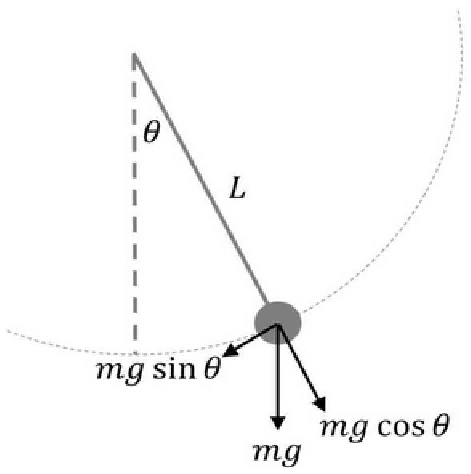
Note that the second order derivative is dependent on the first order derivative so they must be solved simultaneously. Applying Euler's Method yields two equations.

$$y \approx y_0 + f(t_0, y_0)(t - t_0)$$

$$\frac{dy}{dx} \approx \frac{dy}{dx}|x = 0 + f(0, t_0, y_0)(t - t_0)$$

Example: It is well known that the equation for an undamped pendulum is

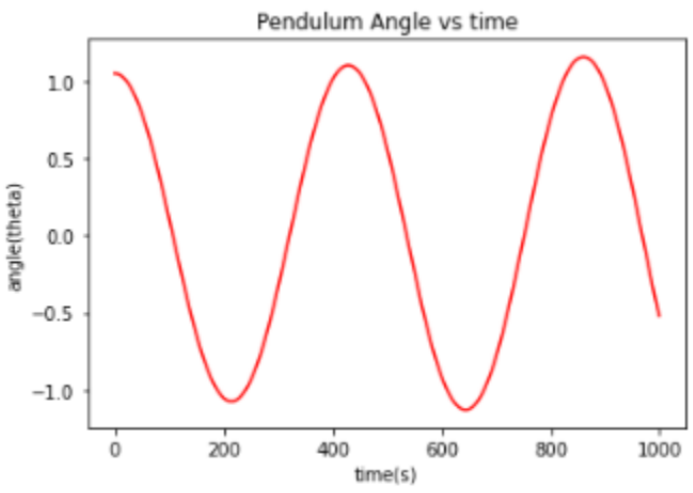
$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta$$



]

where l is the length of the pendulum. Find the value given $\theta(0) = 60$ or $\frac{\pi}{3}$ rad. [2]

```
theta_init=np.pi/3
w_init=0

def Pendulum(theta0,w0,n_iter=1000,dt=0.01,g=10,l=1):
    phase_traject=np.zeros((n_iter,2))
    phase_traject[0,:]=np.array([theta_init,w_init])

    for i in range(n_iter-1):
        theta_d=-g/l*np.sin(phase_traject[i,0])
        phase_traject[i+1,1]=phase_traject[i,1]+dt*theta_d
        phase_traject[i+1,0]=phase_traject[i,0]+dt*phase_traject(i,1)

    return phase_traject

dt=0.005
n_iter=1000

path=trajectory(theta_init,theta_d_init,dt=dt,n_iter=n_iter)
plt.title("Pendulum Angle vs Time")
plt.plot(path[:,0])
plt.xlabel(time(s))
plt.ylabel('angle(theta)')
plt.show()
```



5 Description of the Runge-Kutta Methods

The Runge Kutta methods are a family of equations/numerical method which can solve first order differential equations of the form

$$\frac{dy}{dx} = f(x, y)$$

with an IVP. The constants k_n rom n=1 to n=4 represent the values of slopes at the beginning, midpoint, and end of the interval from the given x value to the desired x value. [3]

RK1(Euler's Method): $k_1 = f(x_i, y_i)$
RK2: $k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$
RK3: $k_3 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h)$
RK4(most common): $k_4 = f(x_i + h, y_i + k_3 h)$ [4]

REFERENCES

[1] Euler's method. https://tutorial.math.lamar.edu/classes/de/eulersmethod.aspx.

[2] Simulating a simple pendulum in python. https://jeremykao.net/2017/07/03/simulating-a-simple-pendulum-in-python-part-1/.

[3] Runge-kutta methods. https://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node5.html.

[4] Runge-kutta 4th order method to solve differential equation. https://www.geeksforgeeks.org/runge-kutta-4th-order-method-solve-differential-equation/.