Comparative study of Euler's method and Runge-Kutta method to solve an ordinary differential equation through a computational approach

Dharma Raj Paudel[1*], Mohan Raj Bhatt[1]
1Graduate School of Science and Technology, Mid-West University, Surkhet, Nepal.
*dharma.paudel@mu.edu.np

Abstract

Euler's and Runge-Kutta's methods are used to solve ordinary differential equations. Euler's methods become appropriate method for solving the equations. When the steps are small, they give reasonably accurate results. However, if the steps are not so small, the Runge-kutta method is preferred to solve the problem. This paper uses the Python program to show the results of both methods. This computational approach shows that the Runge-Kutta method is better for small steps at solving differential equations than Euler's method.

Introduction

Ordinary differential equations are frequently employed in domains such as population dynamics, electronic circuits, chemistry, reaction kinetics, geometry, mechanics, and many more. They also appear in the numerical solution of time-dependent partial differential equations, which are even more surprisingly common in our technologically advanced and financially regulated society, after semi-discretization in space. The solution of a differential equation in closed form is usually not possible for functions that can be evaluated instantly on a computer, even with the most brilliant mathematicians of the eighteenth and nineteenth centuries. Because of this, one is forced to rely on numerical methods that can roughly solve a differential equation to any necessary level of precision (Hairer & Christian, 2012).

The approximate value of the solution in the numerical approach lies at distinct positions (Süli & Mayers, 2003). Since it can be challenging to find precise yet approximate solutions, the field of numerical analysis focuses on developing and evaluating methods to help accomplish these objectives. Therefore, being able to calculate the error associated with such an approximation is crucial. Therefore, the goal of this work is to rigorously analyze and compare the Euler and Runge-Kutta methods in order to show how effective they are in comparison to other similar techniques. It also looks at how the steps affect how accurate the procedures are (Lanlege et al., 2019).

Leonhard Euler created the Euler method in 1768, which is the foundation of all modern advanced numerical techniques. With a specified beginning value, it is a first-order numerical approach for solving ordinary differential equations (ODEs). The simplest and clearest approach for numerically integrating an ordinary differential equation is this one. Euler's idea was to solve an initial value problem by incrementally increasing its solution by small steps. Instead of depending just on the asymmetrical and sometimes inaccurate Riemann rule, Runge's method involved estimating the solution using more exact formulas, such as the midpoint and trapezoidal methods (Butcher, 1996).

In new computational software, the step size for solving initial value problems can be varied while accounting for estimates of the error produced at each step (Butcher, 2007). Runge Kutta's method performs better when the step size is raised since Euler's method becomes less accurate (Kaw & Charlie, 2010). Here, we focus only on the comparison of Euler's method and Runge-Kutta's method. We discuss them separately at first then we will notify the differences by both theoretical and computational approaches. We use Python programs to visualize them separately.

Materials and Methods

Euler Method

First-order numerical methods for solving first-order ordinary differential equations with an initial value are referred to as the Euler method. For example:

$dx/dt = 2x/t$ …. (1)

where we can assign the initial condition $x = 0$ at $t = 0$. A first-order differential equation with one variable can be expressed in generic form as:

$dx/dt = f(x,t)$ … (2)

According to Taylor expansion, we can write x after a short interval h as (https://www.intmath.com/):

$x(t+h)=x(t)+h\, dx/dt + h\, dx/dt + \frac{1}{2} h^2\, d^2x/dt^2 + \dots$

$\quad = x(t) + hf(x,t) + O(h^2)$, …. (3) ,

using equation (1) and(2). Here, $O(h^2)$ refers to all the terms containing h^2 and higher order. If h is small then h^2 becomes very small, so

$x(t+h) = x(t)+hf(x,t)$ .. .. (4)

where the higher-order phrase has been disregarded. It results x(t+h) shortly after if we know the value of x at time t. To find the x for the next interval h, and so on, repeat the process.

First, we start with some known value of x, call it x_0. Then from equation (4)

$x(t_0,h)=x_1=x_0+hf(t_0,x_0)$

where x_1 be the value after the interval h from the initial value x_0 and $f(t_0,x_0)$ be the value of the derivative at the starting point,(t_0,x_0). Similarly, next value x_2 will be

$x_2= x_1+hf(t_1,x_1)$.

where $t_1=t_0+h$.

We can continue this process until we get the desired results. With the points (x_0,t_0),(x_1,t_1), ..etc, a graph x vs t is obtained so that we get the desired values of 'x' for the value 't'. It is not possible to get x(t) for all values of t, only at a finite set of points. However, if h is small enough, it would yield pretty accurate results (Newman, 2013). Here, the computational approach can be utilized to obtain the results of the differential equation. Figure (1) gives the result for the differential equation

$dx/dt = - x^3+\sin t$ …. (5)

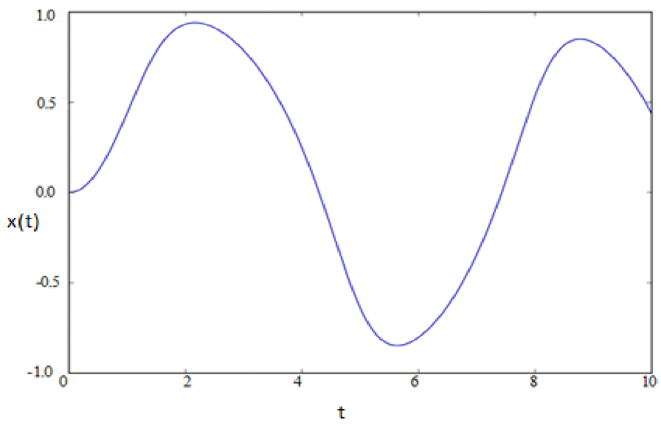

Figure 1: Numerical solution of an ordinary differential equation obtained by using Euler's method

Example

Suppose a differential equation:

$dx/dt = - x^3+\cos t$ … (6)

we have given initial condition x = 0 at t=0. Python program (Van Rossum & Drake, 2003) is used to plot the results. Suppose the interval from t= 0 to t = 10 is divided in 1000 steps. The Python source code to plot the results (Newman, 2013):

```
“from math import*
```

```
from numpy import arrange
from pylab import plot, xlabel,ylabel, show
def f(x,t):
    return  -x**3+cost(t)
a = 0.0
b = 10.0
N = 1000
h = (b-a)/N
x = 0.0
tpoints = arrange(a,b,h)
xpoints = []
for t in tpoints:
     xpoints.apend(x)
    x+=h*f(x,t)
plot(tpoints, xpoints)
xlabel("t")
ylabel("x(t)")
show()"
```

To run these codes, the jupyter notebook is used through the anaconda prompt (Rolon-Mérette et al., 2016).

Euler's method yields only approximate solutions because we neglect the h^2 and all higher terms in equation (3). This error can be minimized when the 'h' term becomes small.

Runge-Kutta method

In Euler's method, we have neglected the order h^2 term (or higher). Nevertheless, we can maintain the order h^2 term, which is equivalent, due to the Runge-Kutta method.

$\frac{1}{2} h^2 d^2x/dt^2$

This method is also called the second-order Runge-Kutta method, while Euler's method is a Runge-Kutta method of the first order. Using Taylor's expansion, we get the value of x(t+h) around t+½h

$x(t+h) = x(t+\frac{1}{2}h) + \frac{1}{2} h(dx/dt)_{t+\frac{1}{2} h} + \frac{1}{8} h^2 (d^2x/dt^2)_{t+\frac{1}{2} h} + O(h^2)$ …. (7)

Similarly, we can derive an expression for x(t):

$x(t) = x(t+\frac{1}{2}h) - \frac{1}{2}h(dx/dt)_{t+\frac{1}{2} h +} + \frac{1}{8}h^2(d^2x/dt^2)_{t+\frac{1}{2}h} + O(h^3)$ … (8)

Subtracting the second expression from the first and then rearranging:

$x(t+h) – x(t) = h(dx/dt)_{t+\frac{1}{2} h} + O(h^3)$.

$= x(t)+hf(x(t+\frac{1}{2}h),t +\frac{1}{2}h)+O(h^3)$ … (9)

Thus, the term $O(h^2)$ disappeared completely. Therefore, the error is of the order h^3. In this regard, the Runge- Kutta method is accurate to order h^2 and error is of order $h^{3,}$ in contrast, Euler's method is a first-order method with an error of order h^2.

Approximating x(t+½h) using Euler's method (Newman, 2013):

$x(t+\frac{1}{2}h)= x(t)+\frac{1}{2}hf(x,t)$ …. (10)

Let, $k_1 = hf(x,t)$ … (11)

Then , $k_2 = hf(x+\frac{1}{2}k_1,t+\frac{1}{2}h)$, … (12)

Therefore, equation (9) by using equations (10),(11), and (12) becomes:

$X(t+h) = x(t) + k_2$.

Now, a small adjustment is made to the Runge-Kutta method program from Euler's approach (Newman, 2013):

```
"from math import*
from numpy import arange
from pylab import plot,
xlabel,ylabel, show
def f(x,t):
return  -x**3+cos(t)
a = 0.0
b = 10.0
N = 10
h = (b-a)/N
tpoints = arange(a,b,h)
xpoints = []
x = 0.0

for t in tpoints:
      xpoints.append(x)
       k1 = h*f(x,t)
       k2= h*f(x+0.5*k1,t+0.5*h)
       x +=k2
plot(tpoints, xpoints)
xlabel("t")
ylabel("x(t)")
show()"
```

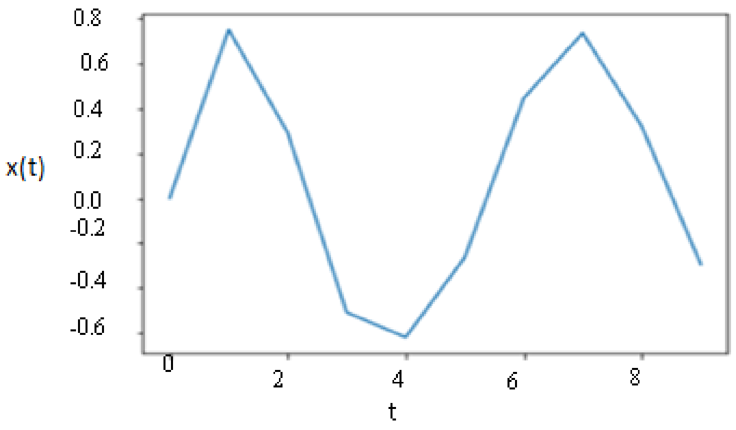

Figure 2: The plot for N=10 in second-order Runge -Kutta method

where we can put N with different values like 10, 20, 50, … . and plot the results. Figure(2) shows the plot for N=10.

Results and Discussion

The differential equation given by equation (6) is solved by both Runge-Kutta(R-K) method and Euler's Method, the comparative result is shown in figure (3).

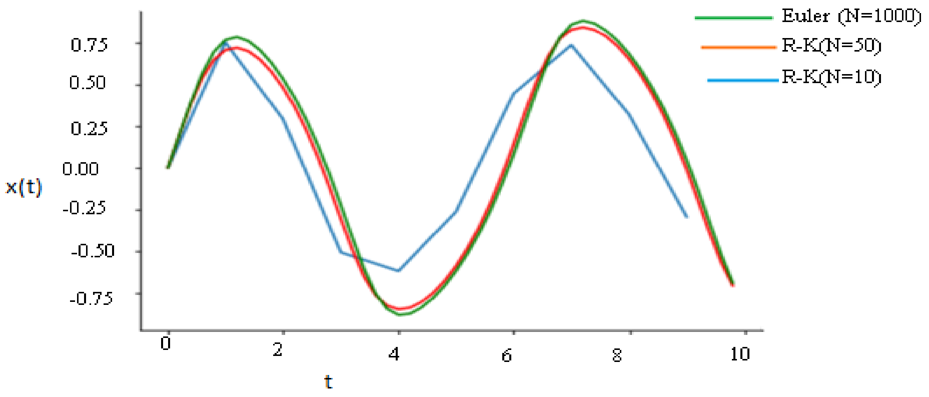

Figure 3: Comparison of the Euler's and Runge-Kutta methods

Figure (3) shows the plot for N = 10 and 50 using the Runge-Kutta(R-K) method while N =1000 using Euler's method. It shows that the Runge-kutta method approaches Euler's method for N =50. This means the Runge-kutta method gives accurate results even when we make 'h' bigger than in Euler's method i.e. former gives the same level of accuracy for large h (i.e. for N =50) as that of Euler's method for small h (i.e. for N =1000). This also shows that as the value of N increases, then the Runge-Kutta curves approach to Euler's method.

Conclusions

Differential equations are solved using both Euler's method and the Runge-Kutta method. Euler's approach suffices to solve the differential equation when the step size is modest. Nonetheless, the Runge-Kutta technique is employed to solve the problem if the step size is big since it produces more accurate results, i.e., up to second order (to the order h^2). Only accurate answers up to the first order (to the order h) are obtained using Euler's approach.. Hence, the Runge-Kutta method even for relatively larger step sizes (h) gives accurate results as that of Euler's method having smaller step sizes.

References

Butcher, J. (2007). Runge-kutta methods. *Scholarpedia*, *2*(9), 3147.

Butcher, J. C. (1996). A history of Runge-Kutta methods. *Applied numerical mathematics*, *20*(3), 247-260.

Hairer, E., & Ch, L. (2012). Numerical solution of ordinary differential equations. *The Princeton companion to applied mathematics*, 293-305.

https://www.intmath.com/differential-equations/11-eulers-method-des.php [Retrieved on 9/21/2019]

Kaw, A., & Charlie, B. (2010). *Solving ODEs Euler Method and RK2/RK4*. Retrieved from http://www.mpia.de/~klahr/Lecture_03_OdE.pdf [Retrieved on 9/20/2019].

Lanlege, D., Kehinde, R., Sobanke, D., Abdulganiyu, A. and Garba, U. (2019). *Comparison of Euler and Range-Kutta methods in solving ordinary differential equations of order two and four*. [online] Ljs.academicdirect.org. Available at: http://ljs.academicdirect.org/A32/010_037.htm [Retrieved on 9/17, 2019].

Newmann, M. (2013). *Computational Physics.*

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2016). Introduction to Anaconda and Python: Installation and setup. *Quant. Methods Psychol*, *16*(5), S3-S11.

Süli, E., & Mayers, D. F. (2003). *An introduction to numerical analysis*. Cambridge university press.

VanRossum, G., & Drake, F. L. (2010). *The python language reference* (Vol. 561). Amsterdam, Netherlands: Python Software Foundation.