Using Runge-Kutta method to observe the behavior of a particle inside a double-well potential

Kenneth M. Leo

National Institute of Physics, University of the Philippines Diliman, Philippines

Introduction

The most common use of computers for Physics is to find the solutions of a differential equation, or sets of differential equations [1]. Out of all the popular methods used in solving ordinary differential equations (ODEs), the Runge-Kutta method is considered to be both the most popular and most stable technique [2]. This method was developed in 1901 by two German mathematicians, Carl Runge and Martin Kutta, which Runge used in solving differential equations he encountered while researching on the atomic spectra [3].

The Runge-Kutta method is a set of methods - with different orders giving results of varying degrees of accuracy. The first order Runge-Kutta method is also known as the Euler's method because its original inventor is named Leonard Euler [1]. Given an ODE with a general form of

$$\frac{dx}{dt} = f(x,t) \tag{1}$$

and an initial condition which fixes the value of x at some t, then the value of x after a short interval h can be written using a Taylor expansion:

$$\begin{aligned} x(t+h) &= x(t) + h\frac{dx}{dt} + \frac{1}{2}h^2\frac{d^2x}{dt^2} + ... \\ &= x(t) + hf(x,t) + O(h^2) \end{aligned} \tag{2}$$

If the value of the interval h is small, then h^2 will be smaller which means the term $O(h^2)$ can be removed from Eq. 2, thus the final equation for the value of x after interval h is

$$x(t+h) = x(t) + hf(x,t) \tag{3}$$

The value of x can then be calculated at a succession of evenly spaced points [1]. If h is small enough, the method can approximate a solution to the differential equation with decent accuracy. The error of the solution is in the order h^2 [4].

A better method, the second-order Runge-Kutta method (commonly known as RK2) which is also called the midpoint method, is introduced to produce a more accurate result which runs faster than the Euler method. The difference between the two methods is that RK2 uses intervals $t + \frac{1}{2}h$ instead of $t + h$ and do the extrapolation in these intervals. This produces significantly better results than Euler's method. In mathematical terms, the value of *x(t+h)* and *x(t* when expanded using Taylor expansion with step size $t + \frac{1}{2}h$ is:

$$\begin{aligned} x(t+h) &= x(t+\frac{1}{2}h) + \frac{1}{2}h(\frac{dx}{dt})_{t+\frac{1}{2}h} + \frac{1}{8}h^2(\frac{d^2x}{dt^2})_{t+\frac{1}{2}h} + O(h^3) \\ &= x(t+\frac{1}{2}h) - \frac{1}{2}h(\frac{dx}{dt})_{t+\frac{1}{2}h} + \frac{1}{8}h^2(\frac{d^2x}{dt^2})_{t+\frac{1}{2}h} + O(h^3) \end{aligned} \tag{4}$$

Simplifying the two expressions above will give

$$x(t+h) = x(t) + hf(x(t+\frac{1}{2}h), t+\frac{1}{2}h) + O(h^3) \tag{5}$$

The error according to Eq. 4 is now in the order h^3 for only two calculations of the function f [4]. Since the value of $t + \frac{1}{2}h$ is not known, Euler's method can be used to approximate its value and then substituting it to Eq. 5. Now the complete calculation for a single step using RK2 is

$$k_1 = hf(x,t) \tag{6a}$$

$$k_2 = hf(x + \frac{1}{2}k_1, t + \frac{1}{2}h) \tag{6b}$$

$$x(t+h) = x(t) + k_2 \tag{6c}$$

To further improve the accuracy of the Runge-Kutta, the fourth-order or RK4 was introduced. The RK4 method was derived using similar methods, and it turns out to have error in the 5th order (h^5). The complete calculation for a single step using RK4 is very similar to Eq. 6 and is shown below:

$$k_1 = hf(x,t) \tag{7a}$$

$$k_2 = hf(x + \frac{1}{2}k_1, t + \frac{1}{2}h) \tag{7b}$$

$$k_3 = hf(x + \frac{1}{2}k_2, t + \frac{1}{2}h) \tag{7c}$$

$$k_4 = hf(x + k_3, t + h) \tag{7d}$$

$$x(t+h) = x(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{7e}$$

Algorithm

Programming the fourth-order Runge-Kutta method is straightforward. A function is defined which takes the differential equation *f*, initial values, step size, and interval as parameters. If the function f is just dependent on one variable, for example x, the program in *Python* would look like this

```
def rk4(f,r,t,h):
    k1 = h * f(r,t)
    k2 = h * f(r+0.5*k1, t+0.5*h)
    k3 = h * f(r+0.5*k2, t+0.5*h)
    k4 = h * f(r + k3, t+h)
    return (k1 + 2*k2 + 2*k3 + k4) / 6
```

For differential equations with two dependent variables, it is better to separate it into two partial differential equations and create an array of functions as an input for the RK4 function.

Results and Discussion

The differential equation chosen in this study is the equation of motion describing a particle of mass m that is moving in a one-dimensional double-well potential was considered and is subject to friction that is proportional to its velocity. The RK4 was applied to this differential equation.

The equation of motion for the position *x(t)* as a function of time t is

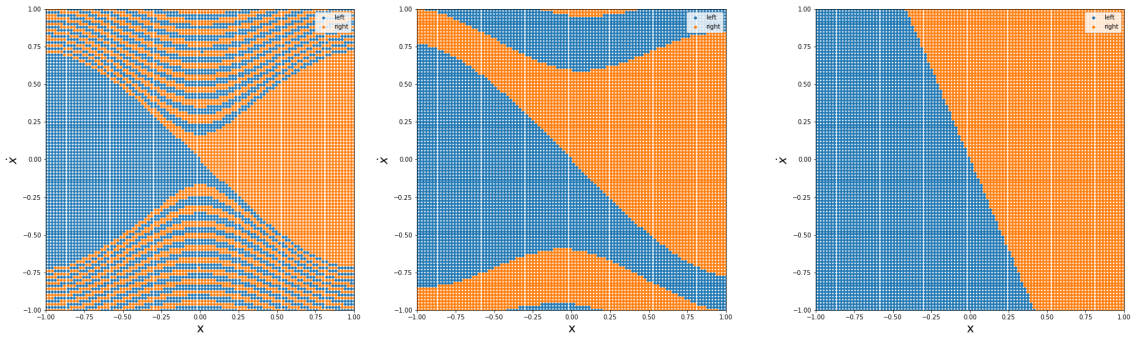
$$m\ddot{x} = \mu\dot{x} + ax - bx^3 \tag{8}$$

where μ, a, and b are constants. The particle will either end up in the left or right potential well after t = ∞ depending on the initial state. Using the constants $m = a = b = 1$ and $\mu = 0.1$, a scatter plot was of regions in the plane of initial states (x(0),$\dot{x}(0)$) was created. The scatter plot in Fig. 1 shows the initial states where the particles will end up either in the left well (x(∞) = -1) or in the right well (x(∞) = 1).

Before creating the scatter plot, the equation of motion was solved first using RK4 using different initial state values with $t = 20000$ to approximate infinity. If the output of the RK4 is equal or close to -1, the set of initial states (x(0),$\dot{x}(0)$) was stored in a list. Another list was also created for initial states producing an output of 1. After checking all the initial states, the scatter plot was created as shown in Fig. 1.

The blue scatter plot represents the initial states where the particle will end up in the left well while the orange scatter plot shows the initial states of a particle which will end up in the right well as time approaches ∞.

The plots in Fig 1 show that when the particle is initially at rest, and is initially located in the left well($x < 0$), the particle will end up in the left well after some large time t. The reverse is also true that when it is located at $x > 0$, it will remain in the right well after an infinite amount of time.



(a) $\mu = 0.01$ (b) $\mu = 0.1$ (c) $\mu = 2$

Figure 1: Scatter plot of regions with different μ in the plane of initial states (x(0),$\dot{x}(0)$) where the particle ends up either in the left well (x(∞) = -1) or in the right well (x(∞) = 1)

When the velocity is non-zero, the behavior of the particle is different for each values of μ. At high values of μ, this means that the friction acting upon the particle is huge which is why most of the particles that are initially in the left well stayed in the left well. The particle will only transfer from one well to another if its initial position is close to the center of the double-well potential. For $\mu = 0.1$, the particle will transfer from one well to another if the velocity has a large magnitude and has the opposite direction from where the particle originally is, i.e. the particle from the right well will transfer to the left well if the velocity is large and to the left. This is because the large velocity counteracts the effect of friction on the particle. For a double-well potential with a very small value for μ ($\mu = 0.01$), the final position of the particle at high velocities can become unpredictable. This is because the friction is very small which means the particle can almost freely transfer from one well to another. For small values of $\dot{x}$, the particle that started from the left will stay from the left unless it is close to the center. Without friction, the particle inside will act as a wave because it can freely move from one well to another without any restrictions and it will be impossible to tell the exact location of the particle in a certain time because nothing will counteract its movement.

References

[1] M. Newman, *Computational Physics* (CreateSpace Independent Publishing Platform, CreateSpace, 4900 LaCross Road, North Charleston, SC 29406, USA, 2013).

[2] E. Ayars, *Computational Physics with Python* (California State University, Ohio, 2013).

[3] K. Christopher, Math 373 Runge-kutta, http://showard.sdsmt.edu/Math373/_AppliedNumMethodsText_SMH/RK_History.htm.

[4] W. Kinzel and G. Reents, *Physics by Computer: Programming Physical Problems Using Mathematica and C* (Springer-Verlag Berlin Heidelberg, Germany, 1998).

Appendix

Source Code:

`https://notebooks.azure.com/kennethleo/libraries/rungekutta`