Development of an Interactive Graphical Users Interface (GUI) to Investigate the Damping of a Simple Spring Mass System Embedded in Fluid

Jerold Young & Onic I. Shuvo

February 14, 2019

Abstract

Differential equations are capable of approximating the analytical solutions of any system undergoing change. Often, some systems described by higher order differential equations are complex enough that a purely analytical solution to the equations is not easily tractable. It is in these complex systems where computer simulations and numerical methods are useful. We have studied a spring-mass system which requires a second order differential equations to describe. We solved the equation of motion numerically and approximate the final position and velocity of the test mass with different numerical methods under damping conditions as the whole system embedded in fluid. We developed a tool that demonstrate the simulation of the simple spring mass system which allows the user to select different parameters for any specific condition.

1 Introduction

One of the most studied example of a second order system is a mass oscillating on a spring. A mass is attached to fixed spring where gravity is normal to the direction of motion, the spring is pulled back and held at rest, finally the mass is then released and oscillates. The governing equations for this system can be derived using Newton's law,

$$F = ma$$

where F is the force exerted on mass and a is acceleration. But the force exerted by the spring is proportional to the amount that it is stretched,

$$F = -kx$$

where k is called the spring constant and x is the displacement of the spring from the equilibrium state. Equating both the expressions for the force lead to the express

$$ma = -kx$$

Recall that acceleration is the second derivative of position and we have a second order differential equation

$$m\frac{d^2x}{dt^2} = -kx$$

Now if the mass is embedded in a fluid characterized by the dynamic viscosity η what gives us the drag coefficient of that fluid c, then at a displacement x from the equilibrium point, the force acting on the mass is

$$m\frac{d^2x}{dt^2} = -kx - cv$$

where v is the velocity of the test particle.

2 Initial Value Problem (IVP)

The initial conditions for the mass of our spring system are that the mass is pulled a little distance, held steady, then released. Mathematically, the initial conditions are

$$x(t=0) = x_0$$

$$v(t=0) = 0$$

When solving differential equations numerically we usually like to work with systems of equations that involve only first derivatives. This is convenient because the numerical implementation can be generalized to solve any problem, regardless of size and order of the highest derivative. In the above example, the second order system is transformed quite easily using the relationships between velocity, position, and acceleration,

$$a = \frac{dv}{dt}$$

$$v = \frac{dx}{dt}$$

Now we can rewrite the equation for the force acting on the mass as the following first order differential equation

$$m\frac{dv}{dt} = -kx - c\frac{dx}{dt}$$

So the acceleration on the particle could be found by the equation

$$\frac{dv}{dt} = -\frac{k}{m}x - \frac{c}{m}\frac{dx}{dt}$$

2.1 Implementation of Euler's Method

If the force acting on a particle is known, the acceleration acting on the particle can be calculated:

$$\mathbf{a}(t, x, v) = \frac{\mathbf{F}(t, x, v)}{m}$$

From the initial conditions, the instant moment we released the mass after a initial pull ($t = 0$),

$$\frac{dv}{dt} = -x_0, \qquad \frac{dx}{dt} = 0$$

After the release the spring is contracting so the acceleration of the mass is negative but the position of the mass is not yet changed. Now, we know the value of the function (position and velocity are given from the initial condition) and the value of their derivatives from the governing equation. We can simply apply Euler's method to both the equations in our system to predict the state of the system a short time:

$$v_1 = v_0 + \tau\frac{dv}{dt}$$

$$x_1 = x_0 + \tau\frac{dx}{dt}$$

Substituting the value of the derivatives in to the above equations and generalizing beyond the first time step yields

$$v_{n+1} = v_n - \frac{k}{m}x_n\tau$$

$$x_{n+1} = x_n + \tau v_n$$

2.2 Implementation of Fourth-Order Runge-Kutta method

A very useful method to solve ordinary differential equations is Fourth-Order Runge-Kutta method. The method can be summarized as follows: If the initial conditions are known (x(t) and v(t)), then the future location x_{n+1} and velocity v_{n+1} of the particle at $t_{n+1} = t_n + \tau$ is given by:

$$\mathbf{k}_1 = \mathbf{a}(t_n, x_n, v_n)$$

$$\mathbf{k}_2 = \mathbf{a}(t_n + \frac{1}{2}\tau, x_n + \frac{1}{2}\tau v_n, v_n + \frac{1}{2}\tau \mathbf{k}_1)$$

$$\mathbf{k}_3 = \mathbf{a}(t_n + \frac{1}{2}\tau, x_n + \frac{1}{2}\tau v_n + \frac{1}{4}\tau^2 \mathbf{k}_1, v_n + \frac{1}{2}\tau \mathbf{k}_2)$$

$$\mathbf{k}_4 = \mathbf{a}(t_n + \tau, x_n + \tau v_n + \frac{1}{2}\tau \mathbf{k}_2, v_n + \tau \mathbf{k}_3)$$

$$x_{n+1} = x_n + \tau v_n + \frac{1}{6}\tau^2(\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3)$$

$$v_{n+1} = v_n + \frac{1}{6}\tau(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

3 Methods

The Matlab codes used in this project were the codes we used for the homework assignments (RK4 and ode45). The primary change for this project was instead of fixed values of the parameters in the script, we treated them as variables. We wanted the user to be able to interactively change the values of parameters as they desired. To do this we used technique's of object oriented programing. Primarily we used the *app.data* structure which allows us change the landscape of the GUI and also allows the GUI to pass user inputed values to its respective methods and functions. See below for a detailed walk through on how we created the mass parameter field in our project.

```
            Create MassKgEditFieldLabel

            app.MassKgEditFieldLabel = uilabel(app.UIFigure);
(code which  allows us to have a text label next to our numeric input box)
            app.MassKgEditFieldLabel.HorizontalAlignment = 'right';
(code which  tells matlab that the label will be right "justified")
            app.MassKgEditFieldLabel.Position = [445 446 60 22];
(code which sets the position for the field label the values represent locations
 on a grid plane in matlab referenceing postions of objects on the UIinterface)
            app.MassKgEditFieldLabel.Text = 'Mass (Kg)';
(code which actually labels the field as Mass(kg))

            Create M
    ( note for ease of use we choose M as the name for the massnumericeditfield so that we only ne
            app.M = uieditfield(app.UIFigure, 'numeric');
(code creating a numeric edit filed in our UI figure, this is where the use will
     enter NUMERIC values for the mass of an object non numeric values will cause the
     program to throw an exception)
            app.M.Position = [520 446 100 22];
    (code which sets the position for the field label the values represent locations
    on a grid plane in matlab referenceing postions of objects on the uiinterface)
            app.M.Value = 0.5;
(code which sets the default value of our mass numeric edit field)
```

Inside the methods such as RK4 were previously we would hard-code the value for mass as say m=2, where m is mass in our function. We know have m=app.M.Value; This tells the RK4 method that the value of m it is to use when running is the value of m that is currently in the numeric

edit box in the app. This data structure allows our project to be more efficient and flexible as now instead of writing innumerable scripts for every possible combination we know have very general methods that can adept with changing parameters. This allows our project to be used in many other cases besides the standard damped mass spring system immersed in water, we can now apply it to a mass spring system of any spherical mass, spring constant, time duration, and fluid all by simply changing a few input parameters. We have also added the corresponding animation of the spring mass system in the interface. It will show the final position for the test particle after the oscillation.

4 Results

Here we are going to demonstrate how would our interface output look like when the code runs. Depending on different dynamic viscosity of the fluid the damping effect will be different. We have created a table of about 28 fluids with dynamic viscosity. We are going to shown for water, glycerin and blood in this section. The damping effect as we have shown on section 2, differs with the nature of the fluid. Here we are going to see the difference between them in the interfaces and a animation that demonstrate the real picture of the motion of the mass. We have separated the interface into two different sections. One for RK4 and the other for ODE45. The precision in measuring the final position (we would say approximation) is also shown in two different separate small boxes.

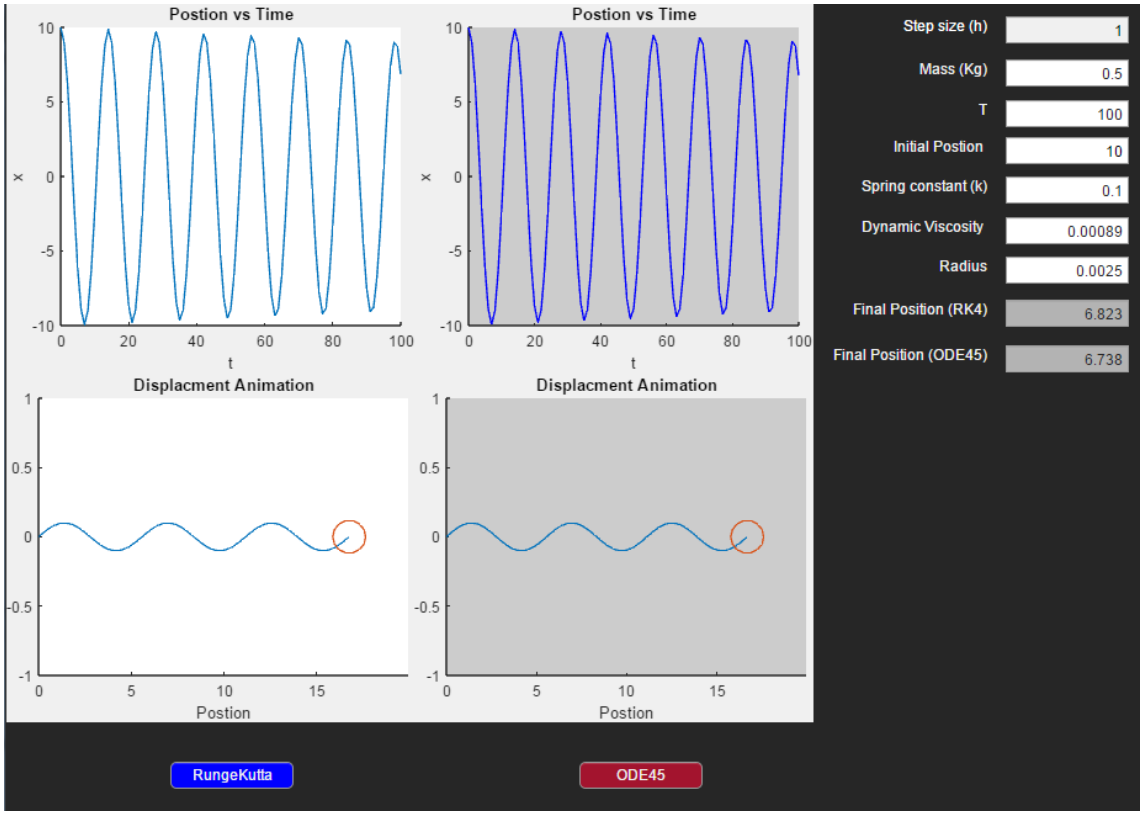

Figure 1: Final position approximation for a spherical test mass attached with a spring embedded in water.

We have omitted the Euler's method in the interface. Euler method was unable to precisely approximate the final position. We have decided to work with the RK4 and ODE45 only. As the Euler's method is not stable enough to approximate the test particle motion accurately. As a test we had run a simple Python code to see the approximation accuracy for RK4 and Euler's [Fig:2]. We can clearly see the difference in approximation by this numerical methods. We have also demonstrating the damping effect of the test mass embedded in in glycerin and blood with their own specific dynamic viscosities.

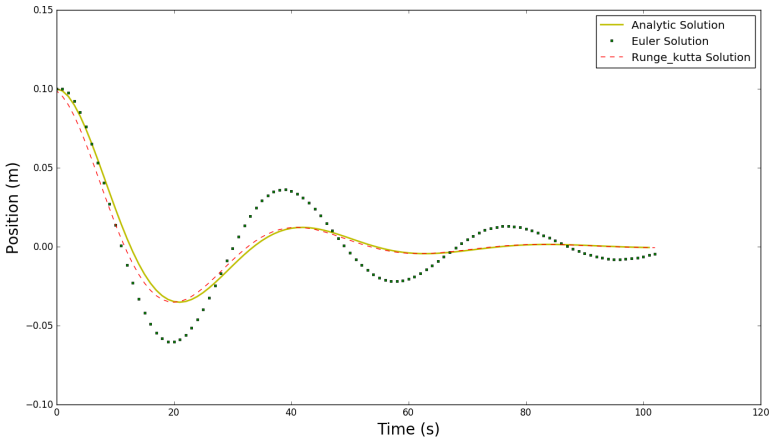

Figure 2: Approximating the position of the test mass. RK4 is approximating the solution more accurately than the Euler's method.

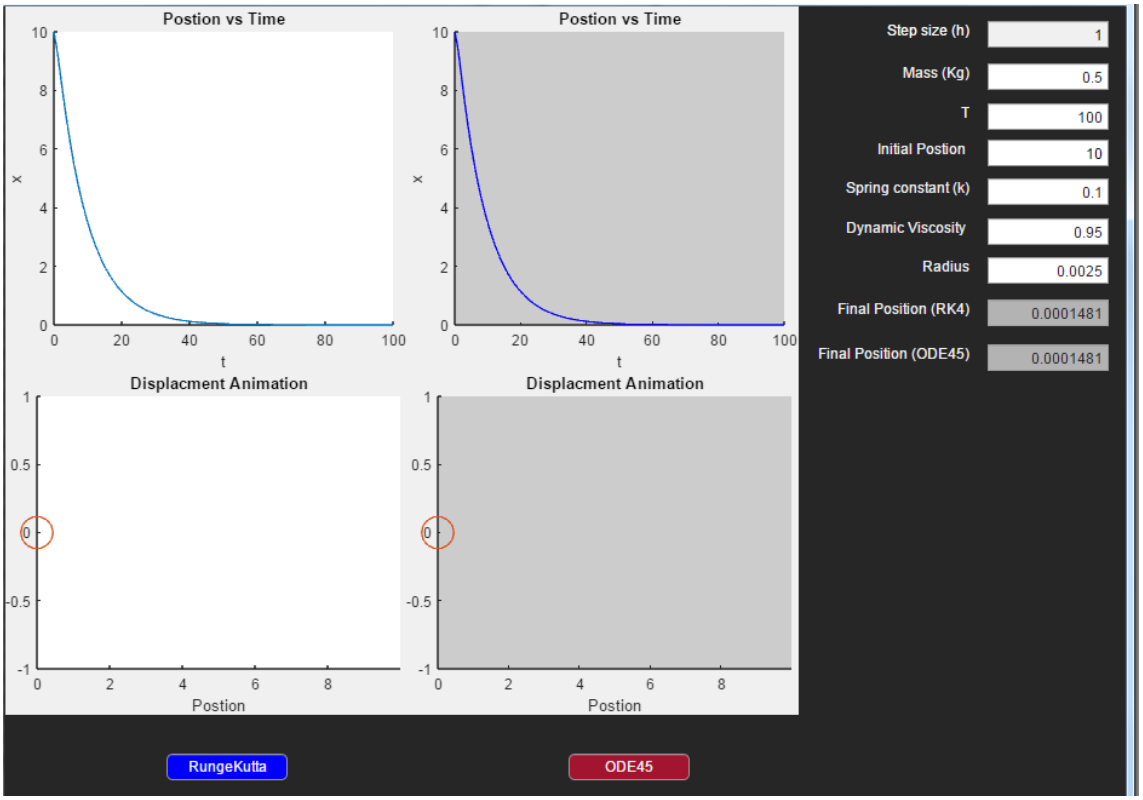

Figure 3: Final position approximation for a spherical test mass attached with a spring embedded in Glycerin.

Only the change of dynamic viscosity largely affecting the damping of the test mass. Notice that we haven't change any of the parameters from the water example. Definitely the mass of the test sphere and spring constant can also affect the damping oscillation. But here we are mostly concern how the fluid creating the drag to make the influence in the spring mass system. We are going to show another example, this time the whole system embedded in blood.

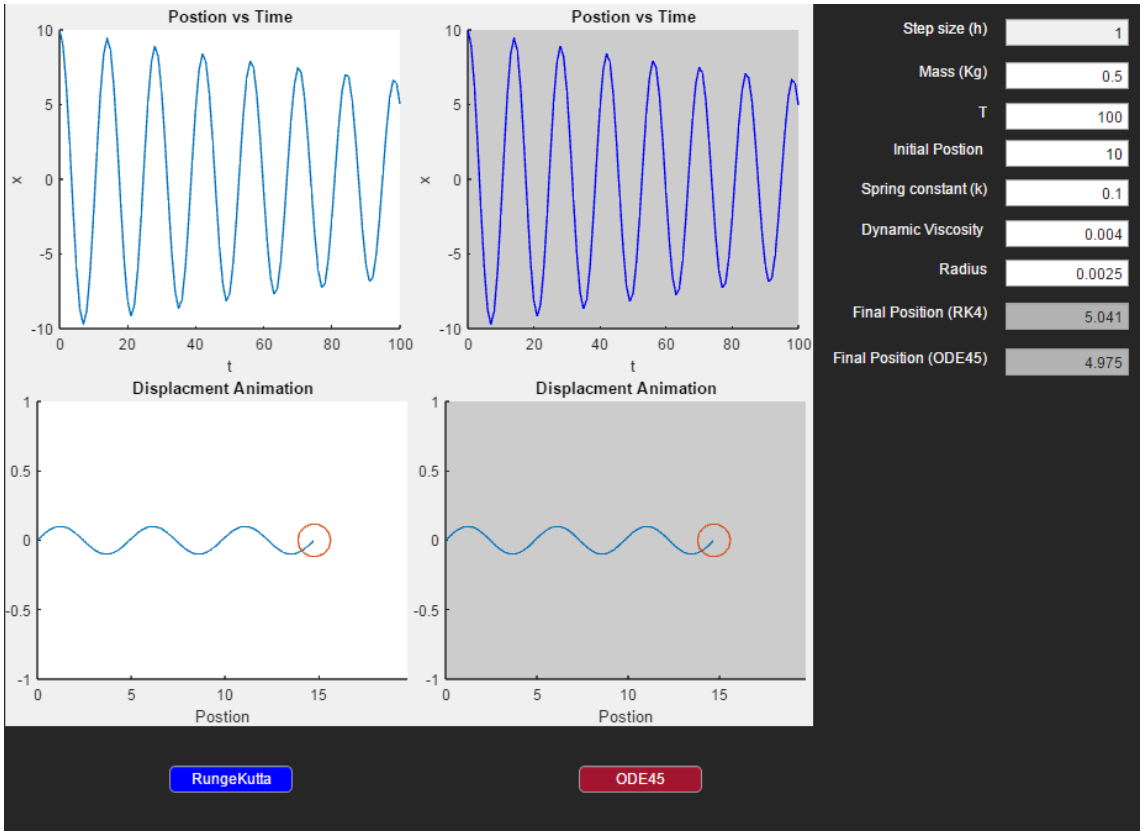

Figure 4: Final position approximation for a spherical test mass attached with a spring embedded in blood.

It is also causing damping but something we can clearly visualize here is that the damping is greater than water but less than glycerin as blood has a dynamic viscosity in between of the other two fluids.

5 Conclusion

Our project is a solid base for fluid mechanics or dynamics study using MATLAB. Our code could be very basic package of the MATLAB library that people can work on to develop more or can design according to what they need to study. Biophysics and Aerodynamics or any other surface physics could be a crucial place where studying different objects motion embedded under fluid is vital.