Modeling Pendulum Motion

Sergey Voronin

October 5, 2015

1 Introduction

We describe here the basics of pendulum motion. Figure 1 below illustrates a single pendulum system. The length of the pendulum is l. It makes an angle of θ from the equilibrium position. Note that the range of θ is assumed to be in the interval $-\pi \leq \theta < \pi$ (this is a choice one makes - the range chosen can be different, as long as the corresponding math is consistent, everything will work out). The green bob is attached to the pendulum rod. The direction of the bob's motion is along the circular arc, while the direction of the instantaneous velocity vector is along the black line with the arrow, tangent to the arc. The pendulum rod is supposed to be massless, while the Bob has mass m. Both the angle and velocity of the Bob are functions of time: $\theta(t)$ and $v(t) = \frac{d\theta}{dt}(t)$. By Newton's second law:

$$F = ma \implies -mg\sin(\theta) = ma \implies a = -g\sin(\theta) \tag{1.1}$$

It remains to express the acceleration a in terms of l and θ. Let us take s to be the arc length along the circular direction of motion. Then we have that:

$$\frac{s}{2\pi l} = \frac{\theta}{2\pi} \implies s = l\theta \implies v = \frac{ds}{dt} = l\frac{d\theta}{dt} \implies a = \frac{d^2s}{dt^2} = l\frac{d^2\theta}{dt^2} \tag{1.2}$$

Substituting into (1.1), we arrive at the second order nonlinear ODE:

$$l\frac{d^2\theta}{dt^2} = -g\sin(\theta) \implies \frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = 0 \tag{1.3}$$

The initial value problem for the single pendulum system without air resistance is usually given as:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = 0 \quad ; \quad \theta(0) = \theta_0 \quad \text{and} \quad \frac{d\theta}{dt}(0) = v_0 \tag{1.4}$$

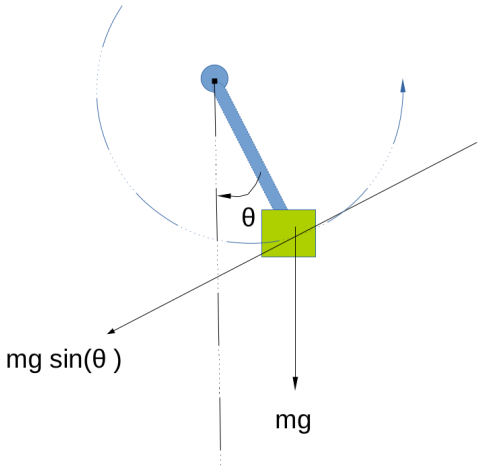


Figure 1: A single pendulum system. Pendulum moves along a circular arc.

If we like to model a damped system (modeling motion with some kind of simple air resistance scheme) we can include a term in Newton's second law equations proportional to the angular velocity:

$$F = ma \implies -\alpha v - mg\sin(\theta) = ma \implies a = -\frac{\alpha}{m}v - g\sin(\theta) \tag{1.5}$$

Plugging in for v and a in terms of θ, we obtain:

$$l\frac{d^2\theta}{dt^2} = -\frac{\alpha}{m}l\frac{d\theta}{dt} - g\sin(\theta) \implies \frac{d^2\theta}{dt^2} = -\frac{\alpha}{m}\frac{d\theta}{dt} - g\sin(\theta)$$

and the IVP:

$$\frac{d^2\theta}{dt^2} + \frac{\alpha}{m}\frac{d\theta}{dt} + \frac{g}{l}\sin(\theta) = 0 \quad ; \quad \theta(0) = \theta_0 \quad \text{and} \quad \frac{d\theta}{dt}(0) = v_0 \tag{1.6}$$

Where as before, the initial conditions give the initial displacement (angle) and velocity (directed speed) of the bob.

1.1 Numerical Modeling

We want to model (1.6). An analytical solution of (1.6) cannot be obtained in terms of elementary functions. However, it is easy to model numerically via numerical methods. We first rewrite the IVP as a first order system, by defining $p = \theta$ and $q = \theta'$. This way, we have $p' = q$ and $q' = \theta'' = -\frac{\alpha}{m}\frac{d\theta}{dt} - \frac{g}{l}\sin(\theta)$ giving:

$$\begin{aligned} \frac{dp}{dt} &= q \\ \frac{dq}{dt} &= -\frac{\alpha}{m}q - \frac{g}{l}\sin(p) \\ p(0) &= \theta_0 \quad \text{and} \quad q(0) = v_0 \end{aligned}$$

Next, we rewrite the system of first order ODEs in familar vector valued function form. Notice that these ODEs cannot be written in terms of a matrix (as in $\vec{x}' = A\vec{x}$) because the ODEs are nonlinear due to $\sin(p)$ term above. Letting:

$$\vec{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} p(t) \\ q(t) \end{bmatrix} \quad \text{and} \quad \vec{f}(t) = \begin{bmatrix} q(t) \\ -\frac{\alpha}{m} q(t) - \frac{g}{l} \sin\left(p(t)\right) \end{bmatrix} = \begin{bmatrix} u_2(t) \\ -\frac{\alpha}{m} u_2(t) - \frac{g}{l} \sin\left(u_1(t)\right) \end{bmatrix}$$

We get:

$$\frac{d}{dt}\vec{u}(t) = \vec{f}(t, \vec{u}) \quad ; \quad \vec{u}(0) = \begin{bmatrix} u_1(0) \\ u_2(0) \end{bmatrix} = \begin{bmatrix} p(0) \\ q(0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ v_0 \end{bmatrix} \tag{1.7}$$

We can solve (1.7) using the Runge-Kutta 4th order scheme using the following sequence of steps. We first discretize a time interval, say $t_i = 0$ to $t_f = 10$ using N steps, resulting in time step $h = \frac{t_f - t_i}{N}$. Then, we perform numerical integration of the system by looping the formulas:

$$\begin{aligned} k_1 &= h\vec{f}(t, \vec{u}) \\ k_2 &= h\vec{f}\left(t + \frac{h}{2}, \vec{u} + \frac{k_1}{2}\right) \\ k_3 &= h\vec{f}\left(t + \frac{h}{2}, \vec{u} + \frac{k_2}{2}\right) \\ k_4 &= h\vec{f}(t + h, \vec{u} + k_3) \\ u &= u + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ t &= t + h \end{aligned}$$

Then at each step, $u(1) \approx \theta(t)$ and $u(2) \approx v(t)$.

In Matlab, one can program this via roughly the following code:

```
function [angles,speeds] = rk4sys_integrator( N, ti, tf, u0, f )
    u = u0; h = (tf - ti)/N; t = ti;
    angles = zeros(N,1);
    speeds = zeros(N,1);
    for i = 1 : N
        k1 = f(t,u);
        k2 = f(t + h/2, u + h*k1/2);
        k3 = f(t + h/2, u + h*k2/2);
        k4 = f(t + h, u + h*k3);
        u = u + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        t = t + h;
        angles(i) = u(1);
        speeds(i) = u(2);
        fprintf('angle = %f, speed = %f\n', u(1), u(2));
    end
end

```

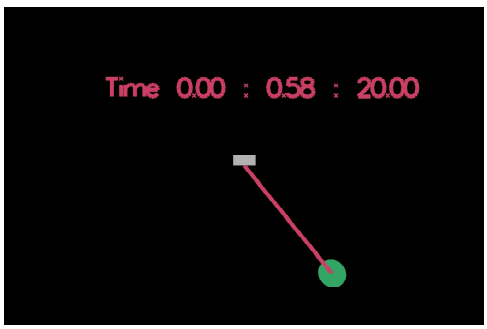
```
% driver for single pendulum simulation
m = 2; % mass
alpha = 1; % air resistance constant
g = 9.8;
l = 10;
theta0 = pi/4; % initial angle of pendulum
v0 = 0; % initial directed speed (positive is counterclockwise)

% vector rhs function
f = @(t,u)[ u(2); -(alpha/m)*u(2) - (g/l)*sin(u(1)) ];

% integrate using rk4
[angles,speeds] = rk4sys_integrator( 2000, 0, 20, [theta0; v0], f );

% at this point we have an array of angles and directed speeds, we can plot the
% pendulum motion using a function like the following
function plot_pendulum(l,angles,save_images)
    close all;
    figure(1);
    for i=1:length(angles)
        theta = angles(i);
        if 0 <= theta < pi/2
            x = l*cos(pi/2-theta);
            y = -l*sin(pi/2-theta);
        elseif pi/2 <= theta < pi
            x = l*cos(theta - pi/2);
            y = l*sin(theta - pi/2);
        elseif -pi/2 <= theta < 0
            x = -l*cos(pi/2-abs(theta));
            y = -l*sin(pi/2-abs(theta));
        elseif -pi <= theta < -pi/2
            x = -l*cos(abs(theta) - pi/2);
            y = l*sin(abs(theta) - pi/2);
        else
            fprintf('invalid theta\n');
        end
        x
        y
        plot([0 x], [0 y], 'linewidth',3);
        hold on
        nsegments = 75;
        r = 0.3;
        th = 0:2*pi/nsegments:2*pi;
```

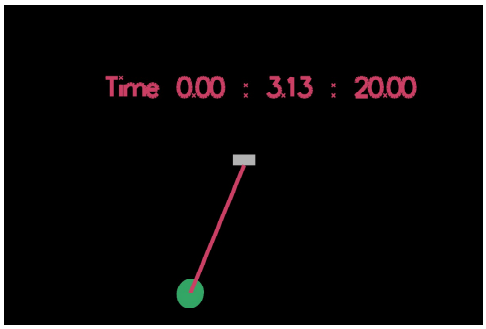
```
61          xunit = r * cos(th) + x;
62          yunit = r * sin(th) + y;
63          h = plot(xunit, yunit, 'r', 'linewidth',3);
64          hold off
65          xlim([-1.2*l,1.2*l]);
66          ylim([-1.2*l,1.2*l]);
67          pause(0.02);
68
69          if save_images == 1
70              if i==1 || mod(i,50) == 0
71                  print('-depsc',['images/frame', num2str(i), '.eps']);
72              end
73          end
74      end
75  end
```

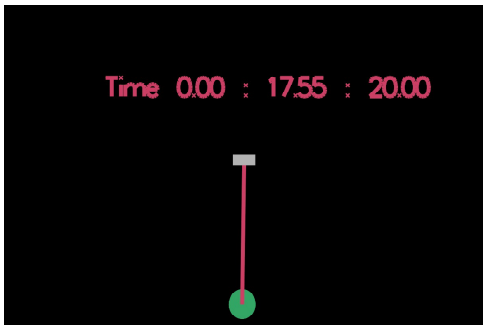
In order to plot the position of the pendulum as a function of time, we must convert the angles $\theta(t)$ into (x, y) coordinates on the xy-plane. We now briefly describe how the plotting is done. Note that the motion is entirely two dimensional. For simplicity we assume the pendulum rod is attached to an anchor at the origin $(0, 0)$, but can rotate freely around the anchor so that $-\pi \leq \theta < \pi$. There are four possibilities we must consider depending on which of the four quadrants the angle corresponds to. The following mapping formulas can be verified by making a small graph and considering the trigonometric relations in different quadrants of the $x - y$ plane:

- If $0 \leq \theta < \frac{\pi}{2}$, then $x = l\cos(\frac{\pi}{2} - \theta)$ and $y = -l\sin(\frac{\pi}{2} - \theta)$.
- If $\frac{\pi}{2} \leq \theta < \pi$, then $x = l\cos(\theta - \frac{\pi}{2})$ and $y = l\sin(\theta - \frac{\pi}{2})$.
- If $-\frac{\pi}{2} \leq \theta < 0$, then $x = -l\cos(\frac{\pi}{2} - |\theta|)$ and $y = -l\sin(\frac{\pi}{2} - |\theta|)$.
- If $-\pi \leq \theta < -\frac{\pi}{2}$, then $x = -l\cos(|\theta| - \frac{\pi}{2})$ and $y = l\sin(|\theta| - \frac{\pi}{2})$.

The pendulum motion is very predictable. Starting with no initial velocity and from an angle of $\frac{\pi}{4}$ with moderate damping (i.e. air resistance), we get motion similar to below: That is, after







some oscillations back and forth, the pendulum returns to it's equilibrium position. Without damping

($b = 0$) the pendulum would oscillate continuously between $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$. The RK4 scheme accuracy depends on the time step size h set via the time steps N parameter. You can use the codes provided here to experiment with different initial conditions for the angle and directed speed. For example, an interesting case is the unstable equilibrium when $\theta_0 = \pi$ and $v_0 = 0$ (i.e. pendulum bob pointing straight up) with finite precision arithmetic. Define π up to a few digits accuracy and observe what happens due to increasing roundoff errors.

The OpenGL version of the simulation is written with the standard Free Glut library. The C code is implemented similar to the Matlab code, except it is easier to treat the two components of the function f separately in the RK4 loop code. Essentially, the following code can be used for the ODE integration:

```
    theta = theta0;
    v = v0;
    for(i = 0; i<N; i++){
        k11 = h*v;
        k12 = -h*(alpha/m)*v - h*(g/l)*sin(theta);

        k21 = h*(v + k11/2);
        k22 = -h*(alpha/m)*(v + k12/2) - h*(g/l)*sin(theta + k12/2);

        k31 = h*(v + k21/2);
        k32 = -h*(alpha/m)*(v + k22/2) -h*(g/l)*sin(theta + k22/2);

        k41 = h*(v + k31);
        k42 =  -h*(alpha/m)*(v + k32) - h*(g/l)*sin(theta + k32);

        theta = theta + (k11 + 2*k21 + 2*k31 + k41)/6;
        v = v + (k12 + 2*k22 + 2*k32 + k42)/6;
    }
```

Note that in the simple classical case of this problem f does not depend on t. The OpenGL graphics part is simple: we compute an array of angles and speeds and then convert angles into (x, y) coordinates as before. We call the *display()* function each time we loop though the $\theta(t)$ array in order to plot a sequence of steps of the pendulum motion.