THE SIMPLE PENDULUM

DERIVING THE EQUATION OF MOTION

The simple pendulum is formed of a light, stiff, inextensible rod of length l with a bob of mass m. Its position with respect to time t can be described merely by the angle θ (measured against a reference line, usually taken as the vertical line straight down). The fact that its position can be described by using only one variable means that the simple pendulum has only one degree of freedom.

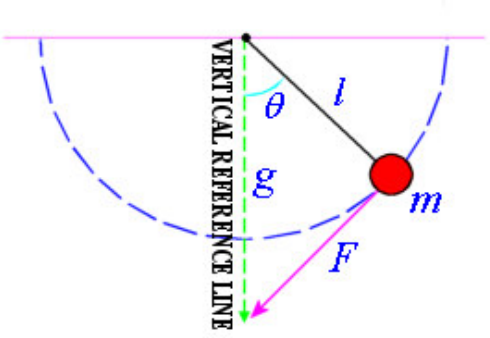

Figure 1: The Simple Pendulum

From **Figure 1**, it can be seen that the driving force of the pendulum is:

$$F = -mg\sin(\theta)$$

Remember that the acceleration g is moving downwards, hence the negative sign.

Now if we take the displacement of the bob from its equilibrium state (hanging exactly straight up or down) to be s, then the acceleration of the bob is $\ddot{s}$.

Newton's second law of motion states that:

$$F = ma$$

Where F denotes a force, m denotes a mass, and a denotes acceleration.

So therefore:

$$-mg\sin(\theta) = m\ddot{s}$$
$$-g\sin(\theta) = \ddot{s} \qquad \textbf{(1)}$$

And when the rod makes an angle θ with the vertical, then the displacement s of the bob is given by:

$$s = l\theta$$

Differentiating this twice with respect to time t gives us:

$$\ddot{s} = l\ddot{\theta}$$

Substituting this into **(1)** and rearranging:

$$l\ddot{\theta} = -g\sin(\theta)$$

Thus:

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta) \qquad \textbf{(2)}$$

PROGRAMMING THE JAVA APPLET

Theoretically, it should be possible to calculate the position of the bob at any time t by using equation **(2)**, which requires only the initial values of θ at time $t = 0$ and $\dot{\theta}$ at $t = 0$.

In practice however, it would be difficult to program Java to solve this second order ordinary differential equation; and so in order to simplify the programming, this needed to be converted into an easier equation for the computer to understand.

The ideal solution was to integrate it numerically – this would form an easily programmable, structured methodology to solve the equation, using small time-steps that could be incremented inside the program using a loop.

At this point I also started to plan how I was going to use Java to display the pendulum. I broke down the problem into stages, and solved each individually within the program before drawing the pendulum as the final stage.

The first step was to calculate the angle θ at time t. θ is the only variable needed to calculate the position of the bob, and once the position of the bob has been calculated it was possible to draw the pendulum. To begin with I created a Java function that used the fourth-order Runge-Kutta method to integrate equation **(2)** numerically to find θ from $\ddot{\theta}$.

Unfortunately, Java cannot plot the motion of the pendulum just by using the angle θ – it uses (x, y) coordinates to plot shapes. Therefore, the second stage of the program was to use θ to calculate the (x, y) coordinates of the centre of the bob at time t. This was done using simple trigonometry:

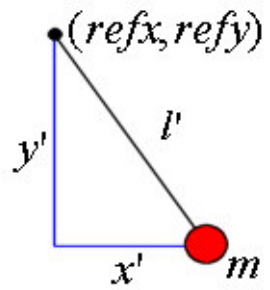

Figure 2: Finding the position of the bob with respect to a reference point

Where l' is the visual length of the rod in pixels. I chose to make this a constant so that the users can input any value they like for l. However, this means that they need to use their imagination to visualise the scale of the applet.

At this point it is worth noting that Java plots coordinates with the $(0, 0)$ origin at the top left of the screen, as opposed to the usual convention of having the origin at the bottom left.

Thus, it can be seen from **Figure 2** that the coordinates of the bob (x, y) with respect to the reference point with coordinates $(refx, refy)$ is:

$$(x, y) = (refx + x', refy + y')$$

Where:

$$x' = l' \sin(\theta)$$
$$y' = l' \cos(\theta)$$

The final stage was to actually draw the pendulum and show the motion. This was made simple by using the coordinates of the bob (calculated in the second stage). The pendulum could be drawn by plotting a line from the reference point $(refx, refy)$ to the (x, y) coordinates of the centre of the bob, and then drawing a small circle around those same coordinates to display the bob.

I used a recursive loop to indefinitely increase time t, calculate θ and the (x, y) bob coordinates, and to draw the pendulum in its new position. I also had to 'repaint' over the applet in white to effectively wipe the drawing area clean after each stage of the loop. This gives the impression of animation. In order to make the pendulum move in 'real-time', I integrated a time-delay into the loop so the screen refreshed at a delayed rate.

Once this was complete, I could see which numerical integration methods showed more realistic motion. After experimenting with the fourth-order Runge-Kutta method and Euler's method, I decided that the most accurate motion was described by using a geometric method (closely related to Euler's). This was because the pendulum gained momentum when Euler's method was used, lost momentum when Runge-Kutta was used; and remained constant when the geometric method was used. Since I had not taken friction into account in the equations of motion, the pendulum should maintain its momentum, and so the geometric method was clearly the best.

The geometric method I used was taken from an unpublished book by Leimkuhler and Reich (2000).

According to this method, then if we want to integrate equation **(2)** then we first need to convert it into the appropriate form as follows:

$$\dot{\theta} = f(t, \theta, \dot{\theta}) = y$$

$$\dot{y} = f'(t, \theta, \dot{\theta}) = -\frac{g}{l}\sin(\theta)$$

Now:

$$\theta_{n+1} = \theta_n + \Delta t y_{n+1}$$

$$y_{n+1} = y_n + \Delta t f'(\theta_n)$$

Or in other words:

$$\theta_{n+1} = \theta_n + \Delta t \dot{\theta}_{n+1} \qquad \textbf{(3)}$$

$$\dot{\theta}_{n+1} = \dot{\theta}_n + \Delta t(-\frac{g}{l}\sin(\theta_n)) \qquad \textbf{(4)}$$

Where Δt is a small time-step.

Although both equations **(3)** and **(4)** were required in the Java program, equation **(3)** is the one that gives a value for θ at time t.

Once this basic applet was finished, I was able to make it more interactive by adding text fields for the user to input their own numbers for the rod length l, and the initial values needed to begin the numerical integration process (θ and $\dot{\theta}$ at time $t = 0$).