Chapter 7

Pendulum Motion

7.1 Introduction

In this lab we will apply the Euler and Verlet methods to the pendulum problem. We will compare the results of the Verlet problem to the small angle approximation.

If you prefer, you can complete the shorter, but more challenging sequence of problems: 7.8, 7.10, and at least two of the optional challenge problems. You will only receive minimal instructor help while you are taking this approach.

7.2 Preparation

Suppose you want to produce a plot of $f(t) = A\exp(-t/\lambda)$ versus t from $t = 0$ to 10 and $A = \lambda = 5$. For a visually smooth plot, you want about $N = 100$ points, but we'll start with a smaller number $N = 11$ for easy debugging. You create an array containing the 11 time values you want to plot:

```
MAX = 10
N = 11
t = np.linspace(0,MAX,N)
print(t)
```

You calculate the $y = f(t)$ values like this:

```
A    = 5
LAMB = 5
y = A*np.exp(-t/LAMB)   # !!!
print(y)
```

Make sure that you thoroughly understand the line marked "!!!". That is calculating one y value for every t value, and so y is an array with the same shape and size as t:

```
print("t shape: ", np.shape(t), "y shape: ", np.shape(y))
print("t size:  ", np.size(t),  "y size: ", np.size(y))
```

With two arrays of the same shape, plotting them is a simple matter. Here we use the red line format, and add some axes labels:

```
plt.plot(t,y,"r-")
plt.xlabel("t (s)")
plt.ylabel("y")
```

If you look closely, you'll see kinks in the plot for $N = 11$. Increase to $N = 100$ for a visually smooth plot.

You are expected to do all of this on your own, from a prompt like this:

△ **Jupyter Notebook Exercise 7.1:** Plot the function $f(t) = A\exp(-t/\lambda)$ from $t = 0$ to 10, $A = 5$ and $\lambda = 5$ as a red line.

Now suppose instead that you already have a set of y-values in a `np.array` `yarr`, and you would like to plot y vs t, knowing that these y values were sampled starting at $t = 0$ with a uniform step size of $\tau = 0.2$ between each sample, i.e. at $t = 0, 0.2, 0.4, \ldots$. In this case, you would create a `np.array` containing your time values as:

```
tau = 0.2
t = np.arange(yarr.size)*tau
```

△ **Jupyter Notebook Exercise 7.2:** Starting from the y values contained in:

```
yarr = np.array([1,1,1,2,4,7,3,2,1,1,0,0])
```

which you know correspond to time values starting at $t = 0$ with constant step size $\tau = 0.5$. Plot y vs t as blue circles and add axes labels.

7.3 Pendulum Motion

In lecture we showed a pendulum of length L can be described by the angle θ with respect to vertical (rest position), the angular velocity:

$$\omega = \frac{d\theta}{dt}$$

and the angular acceleration:

$$\alpha = \frac{d\omega}{dt} = \frac{d^2\theta}{dt^2}$$

In a constant gravitational field with acceleration g, the angular acceleration is:

$$\alpha = -\frac{g}{L}\sin\theta$$

Which we can write as:

$$\alpha = -\omega_L^2 \sin\theta$$

where

$$\omega_L = \sqrt{\frac{g}{L}}.$$

This problem, which is quite simple to pose, has no analytic solution in terms of elementary functions. Instead, we will rely on numerical techniques to solve it.

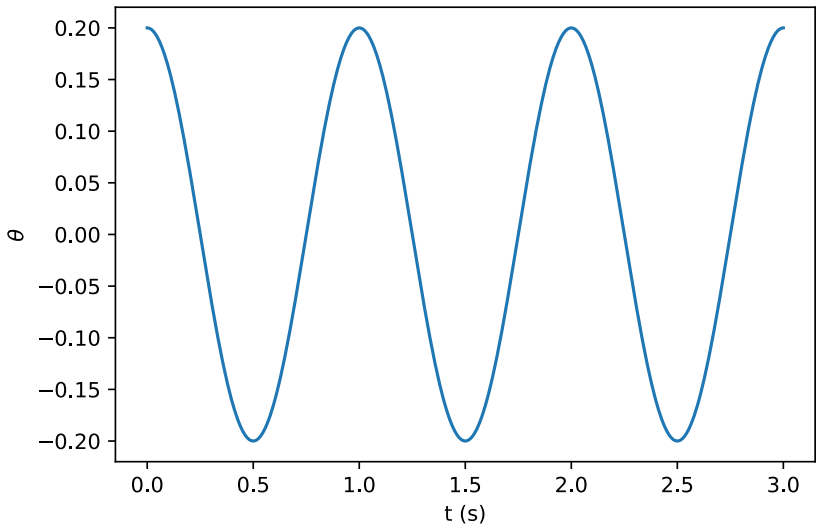

Figure 7.1: The small angle approximation for pendulum motion with period $T_L = 1$ s

.

7.4 Small Angle Approximation

Analytic solutions are the precious gems that we use to validate our numerical techniques. We'll start our analysis in a region where we can solve the problem analytically. For small displacements of the pendulum, θ is small, and so:

$$\sin\theta \approx \theta$$

and

$$\alpha = -\omega_L^2 \sin\theta \approx -\omega_L^2 \theta$$

The differential equation which describes the motion is:

$$\frac{d^2\theta}{dt^2} = -\omega_L^2 \theta$$

We showed in lecture that if the initial angular velocity is zero ($\omega_0 = 0$) and the initial theta position is θ_0, then the solution is:

$$\theta(t) = \theta_0 \cos(\omega_L t) \tag{7.1}$$

as plotted in Fig. 7.1. The pendulum rocks back and forth following a sine wave with period:

$$T_L = \frac{2\pi}{\omega_L} = 2\pi\sqrt{\frac{L}{g}}$$

△ **Jupyter Notebook Exercise 7.3:** Calculate ω_L for a pendulum with $L = 1$ m and $g = 9.8$ m/s^2. What are the units of ω_L?

△ **Jupyter Notebook Exercise 7.4:** Suppose you want to construct a pendulum such that for small oscillations the period $T_L = 1$ s. What should be the value of ω_L? What length L should you use, assuming that $g = 9.8$ m/s?

△ **Jupyter Notebook Exercise 7.5:** Reproduce the plot in Fig. 7.1. Set the constants T_L, ω_L, θ_0, and N as follows:

```
TL = 1           # period in seconds
wL = 2*np.pi/TL  # small-angle angular frequency of pendulum
A  = 0.2         # theta at t=0 (amplitude)
```

You must complete this problem **without using an explicit for loop**. To set the y axis label to the fancy θ do:

```
plt.ylabel("$\\theta$")
```

.

7.5 The Failure of the Euler Method

The Euler equations for angle θ and angular velocity ω are:

$$\theta_{n+1} = \theta_n + \tau\,\omega \tag{7.2}$$
$$\omega_{n+1} = \omega_n + \tau\,\alpha \tag{7.3}$$

where α is the angular acceleration and τ is the time step.

△ **Jupyter Notebook Exercise 7.6:** Implement a function

```
def euler(tau, theta, omega, alpha):
    # your code here
    return theta, omega
```

which implements one iteration of the Euler method. Test your `euler` function with these test values:

```
print(np.around(euler(-0.01, -0.28, -0.30, -0.06),3))
print(np.around(euler( 0.94,  0.32, -0.85, -0.86),3))
print(np.around(euler( 0.92, -0.16,  0.38, -0.32),3))
print(np.around(euler( 0.31,  0.12, -0.91, -0.76),3))
print(np.around(euler(-0.14,  0.96,  0.66, -0.73),3))
```

which should produce the following output:

```
[-0.277 -0.299]
[-0.479 -1.658]
[0.19  0.086]
[-0.162 -1.146]
[0.868 0.762]
```

You'll use this now thoroughly tested function more below. Don't change it!

△ **Jupyter Notebook Exercise 7.7:** Apply the Euler method to the problem of small oscillations of a pendulum with $T_L = 1$ s as in Fig. 7.1. First, set the parameters of your code just as before:

```
TL = 1           # period in seconds
wL = 2*np.pi/TL  # small-angle angular frequency of pendulum
A  = 0.2         # theta at t=0 (amplitude)
```

Then implement the Euler method as follows:

```
θ := A
ω := 0
τ := 0.0003
Create an empty array tjth which will contain θ positions of the trajectory
Append θ to the array tjth.
Repeat N times:
    Calculate α = −ω_L^2 θ
    Update θ and ω for acceleration α and time step τ by calling euler().
    Append θ to the array tjth.
Create an array t containting N+1 appropriately spaced time values.
Plot tjth versus t
```

As always, first debug your code using a small value for N. Then, you should reproduce something that closely resembles Fig. 7.1 with $N = 10000$.

△ **Jupyter Notebook Exercise 7.8:** Repeat the exercise above (you can use cut and paste) with $\tau = 0.01$ and $N = 1000$. Yikes! Is energy conserved?

7.6 The Verlet Method

The Verlet Equation for this problem is:

$$\theta_{n+1} \;=\; 2\theta_n - \theta_{n-1} + \tau^2\,\alpha \tag{7.4}$$

Notice that with the Verlet method, we will not need to calculate angular velocity ω in order to get the θ trajectory.

△ **Jupyter Notebook Exercise 7.9:** Implement a function

```
def verlet(tau, theta, oldth, alpha):
    # your code here
    return theta
```

which returns θ_{n+1} from θ_n =theta and θ_{n-1} =oldth using the verlet method. Test your `verlet` function with these test values:

```
print(np.around(verlet(-0.01, -0.28, -0.30, -0.06),3))
print(np.around(verlet( 0.94,  0.32, -0.85, -0.86),3))
print(np.around(verlet( 0.92, -0.16,  0.38, -0.32),3))
print(np.around(verlet( 0.31,  0.12, -0.91, -0.76),3))
print(np.around(verlet(-0.14,  0.96,  0.66, -0.73),3))
```

which should produce the following output:

```
-0.26
0.73
-0.971
1.077
1.246
```

You'll use this now thoroughly tested function more below. Don't change it!

△ **Jupyter Notebook Exercise 7.10:** In a previous exercise you showed that the Euler method, when applied to the problem of small oscillations of a pendulum with $T_L = 1$ s for $\tau = 0.01$ and $N = 1000$, is unstable. Instead, apply the Verlet method. You can reuse (by copying and pasting) much of your code from that previous exercise with a few changes:

- You will call your `verlet` function instead of `euler`.
- You no longer need to keep track of ω (`omega`)
- You will now have to keep track of two θ values at all times. At each updat:

$$(\theta_n, \theta_{n-1}) \to (\theta_{n+1}, \theta_n)$$

- You can start things off with `oldth = theta = A`

With this method, you should produce many oscillations with no sign of instability.

△ **Jupyter Notebook Exercise 7.11:** So far, we have been using the small angle approximation. Modify your code to use the exact formula for the angular acceleration $\alpha = \omega_L^2 \sin\theta$ and set the initial position to $A = 2$. This is a trivial change to your numerical simulation, but it makes an analytic solution impossible!

△ **Jupyter Notebook Exercise 7.12:** (Optional Challenge) Run you Verlet analysis for $N = 1000$ steps for $A = 3$ and set τ appropriately so that you see a bit more than one period of motion. Plot the trajectory as a black line and read off the period T. Superimpose a plot of a cosine with amplitude A and period T.

△ **Jupyter Notebook Exercise 7.13:** (Optional Challenge) We've been starting things off with approximately zero angular velocity by setting `oldth = theta = A`. You can add velocity to the initial state with:

```
V = 1
theta = A
oldth = A - tau*V
```

Give the pendulum enough of a whack that it reaches all the way to top and keeps going. When plotting this trajectory, you can use `np.mod` to keep the theta values in the range from -π to π if you want.

△ **Jupyter Notebook Exercise 7.14:** (Optional Challenge) Fix the Euler method for this pendulum problem by forcing energy to be conserved at each step. Compare your results to the Verlet method.