PHYS 5120 HW1

TANUWIJAYA, Randy Stefan *
(20582731)
rstanuwijaya@connect.ust.hk

Department of Physics - HKUST

September 22, 2022

1. The linear and nonlinear pendulums

Consider a pendulum with a arm of length $l = 10\,\text{cm}$ holding a bob of mass m. The arm is massless. The gravitational acceleration $g = 9.81\,\text{m/s}^2$

1. A simple pendulum is linear, i.e., $\sin\theta \approx \theta$. Write an equation of motion for the pendulum using Newton's second law. Derive the analytic solution. How long is the swing period?

   The force acting on the pendulum along and perpendicular to the arm are given by:

   $$\Sigma F_{\parallel} = T - mg\cos\theta = 0 \tag{1}$$
   $$\Sigma F_{\perp} = -mg\sin\theta = mr\ddot{\theta} \tag{2}$$

   For small θ, $\sin\theta \approx \theta$, we can solve the ODE by assuming $\theta = \theta_0 e^{i\omega t}$ and $\ddot{\theta} = -\omega^2 e^{i\omega t}$. Thus,

   $$-mg\theta = mr\ddot{\theta}$$
   $$-mg\theta_0 e^{i\omega t} = -mr\omega^2 e^{i\omega t} \iff \omega = \sqrt{\frac{g}{l}}$$

   Therefore,

   $$\boxed{T = \frac{2\pi}{\omega} = 2\pi\sqrt{\frac{l}{g}} = 0.634\,\text{s}} \tag{3}$$

*LaTeX source code: https://github.com/rstanuwijaya/hkust-computational-material/

2. The pendulum is released from a standstill at $\theta = 2.4°$. Write a program to solve the linear pendulum using the velocity Verlet method and a proper time step.

The follwing code plots the comparison between the analytical and numerical solutions.

The analytical solution to the angle of the linear pendulum is given by:

$$\theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{l}}t\right) \tag{4}$$

Whereas for the numerical solution, we use the velocity Verlet method:

$$\theta(t+h) = \theta(t) + \omega(t)h + \frac{\tau(t)}{2I}h^2 \tag{5}$$

$$\omega(t+h) = \omega(t) + \frac{\tau(t)+\tau(t+h)}{2I}h \tag{6}$$

where $\theta(0) = \theta_0$, $\tau(t) = -mgl\sin\theta(t)$ and $I = ml^2$.

On the numerical simulation, we set $dt = 0.01\,\mathrm{s}$ for a total duration of $10\,\mathrm{s}$. The source code for the numerical simulation for linear pendulum is attached in Appendix 1.. The output of the code is given in Figure 1, which shows the comparison between the analytical and numerical solutions. It shows that linear angle approximation agrees well with the numerical solution.

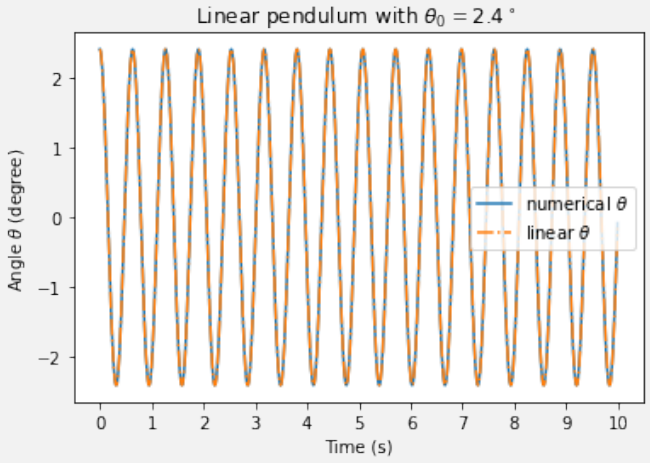

Figure 1: Comparison between the analytical and numerical solutions for the linear pendulum.

3. Let's consider a nonlinear pendulum, i.e., $\sin\theta = \theta$, which is released from a standstill at $\theta = 124°$. What is the total energy if you solve the motion equation exactly without any numerical errors? Use the velocity Verlet method to solve it numerically. Plot the energy as a function of time and show at least 12 swing periods. Compare the exact and numerical energies. Generally increase your time step until you find an obvious energy drift; that is, the total energy increases or decreases over a long time (not short time fluctuations). See, e.g., `https://en.wikipedia.org/wiki/Energy_drift`. Discuss your findings.

Assuming $m = 1\,\text{kg}$, the total energy of the system is given by:

$$E = K_{max} = U_{max} = mgl(1 - \cos\theta_0) = 1.528\,\text{J} \tag{7}$$

We choose this value because the potential energy of the system is zero when the kinetic energy is maximum.

The error analysis of the energy ΔE is given by:

$$\begin{aligned}
\Delta E &= \frac{1}{2}ml^2\Delta(\omega^2) - mgl\Delta(1-\cos\theta) \\
&= \frac{1}{2}ml^2\Delta(\omega^2) - mgl\Delta(\theta^2 + \dots) \\
&= \mathcal{O}((1+\mathcal{O}(h^2))^2) + \mathcal{O}((1+\mathcal{O}(h^4))^2) \\
&= \mathcal{O}(h^2)
\end{aligned}$$

The source code for the numerical simulation for the nonlinear pendulum is given in Appendix 2.. Comparison of the numerical solution of the nonlinear pendulum angle using velocity verlet method and linear approximation is given in Figure 2. It verifies that the linear approximation is not a good approximation when the angular amplitude is large.

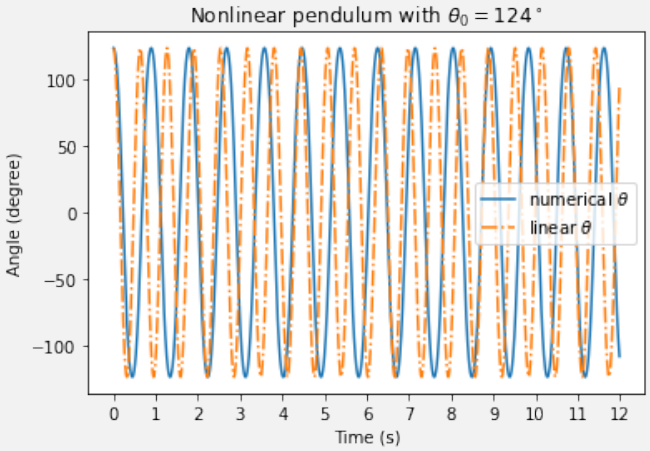

Figure 2: Linear approximation vs and numerical solutions for nonlinear pendulum.

The total energy of the system is given in Figure 3. It shows that when using $h = 0.01\,\text{s}$, the standard deviation of the energy is $0.000\,98\,\text{J}$, which is much smaller than the total energy. Thus, the energy drift is negligible.

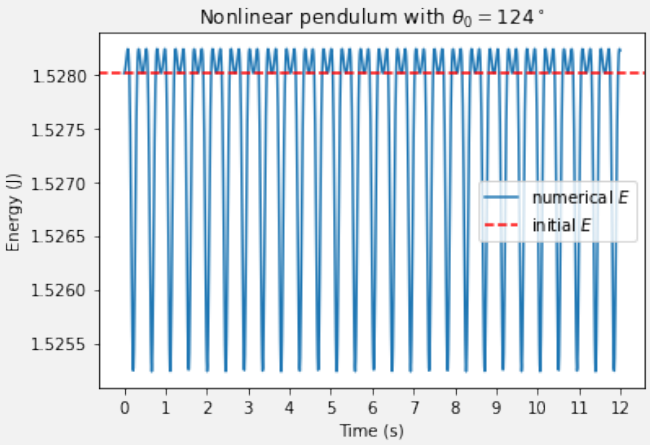

Figure 3: Total energy of the nonlinear pendulum as a function of time.

When we use $h = 0.12$, we can see that the energy drift is significant. The standard deviation of the energy is 0.17 J, which is comparable to the total energy. Figure 4 and 5 shows the angle and total energy of the nonlinear pendulum when $h = 0.12$. The energy drift is caused by the numerical error in the velocity Verlet method, which is in order of $\mathcal{O}(h^2)$

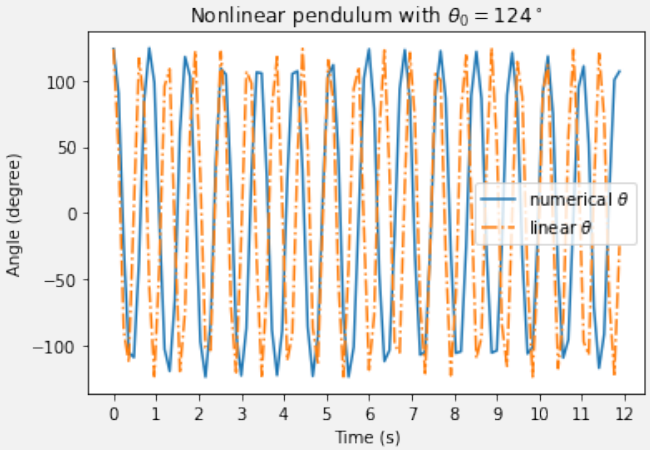

Figure 4: Angle of the nonlinear pendulum when $h = 0.12$.

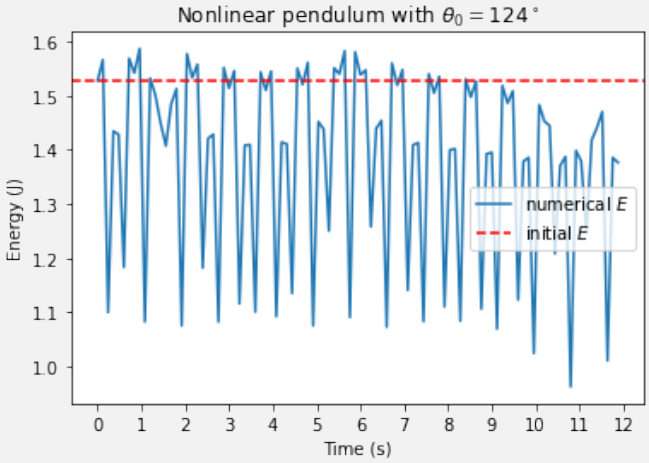

Figure 5: Total energy of the nonlinear pendulum when $h = 0.12$.

For longer simulation time, we can see both angle and energy further drifts. Figure 6 shows the angle and total energy of the nonlinear pendulum when $h = 0.12$ and $t = 30\,\mathrm{s}$.

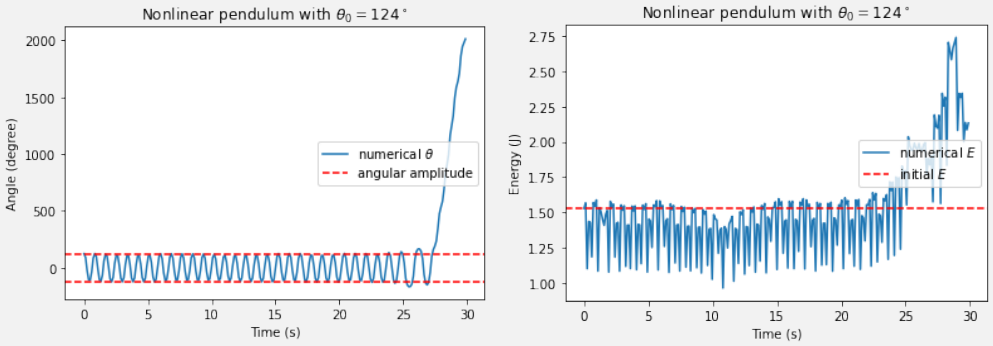

Figure 6: Angle and energy of the nonlinear pendulum for $h = 0.12$ and $t = 30\,\mathrm{s}$.

Appendix

1. Linear pendulum numerical simulation ($\theta_0 = 2.4°$)

```
x_init = 2.4*pi/180      # initial angle
v_init = 0               # initial angular velocity
t_init = 0
t_final = 10
h = 0.01                 # time step in second

m = 1
g = 9.8
l = 10 * 0.01
I = m*l**2

X, V, T = [], [], []
x_curr, v_curr, t_curr = x_init, v_init, t_init # initialize iterator

def calc_tau(x_curr):
        return -m*g*l*sin(x_curr)

for i in range(floor((t_final-t_init)/h)):
        X.append(x_curr)
        V.append(v_curr)
        T.append(t_curr)
        tau_curr = calc_tau(x_curr)
        x_next = x_curr + h*v_curr + tau_curr/(2*I)*h**2
        tau_next = calc_tau(x_next)
        v_next = v_curr + (tau_curr+tau_next)/(2*I)*h
        t_next = t_curr + h
        x_curr, v_curr, t_curr = x_next, v_next, t_next

X_linear = x_init*np.cos(sqrt(g/l)*np.array(T))

plt.plot(T, np.array(X)*180/pi, label=r"numerical $\theta$")
plt.plot(T, np.array(X_linear)*180/pi, '-.', label=r"linear $\theta$")
plt.xlabel("Time (s)")
plt.ylabel(r"Angle $\theta$ (degree)")
plt.legend(loc="center right")
plt.title(r"Linear pendulum with $\theta_0 = 2.4^\circ$ ")
plt.xticks(np.arange(0, t_final+1, 1))
plt.show()
```

2. Nonlinear pendulum numerical simulation ($\theta_0 = 124°$)

```
x_init = 124*pi/180      # initial angle
v_init = 0               # initial angular velocity
t_init = 0
t_final = 12

h = 0.01                 # time step in second
# h = 0.12                 # toggle this line to see the energy drift

m = 1
g = 9.8
l = 10 * 0.01
I = m*l**2

X, V, T = [], [], []
x_curr, v_curr, t_curr = x_init, v_init, t_init # initialize iterator

def calc_tau(x_curr):
        return -m*g*l*sin(x_curr)

for i in range(floor((t_final-t_init)/h)):
        X.append(x_curr)
        V.append(v_curr)
        T.append(t_curr)
        tau_curr = calc_tau(x_curr)
        x_next = x_curr + h*v_curr + tau_curr/(2*I)*h**2
        tau_next = calc_tau(x_next)
        v_next = v_curr + (tau_curr+tau_next)/(2*I)*h
        t_next = t_curr + h
        x_curr, v_curr, t_curr = x_next, v_next, t_next

X_linear = x_init*np.cos(sqrt(g/l)*np.array(T))

plt.plot(T, np.array(X)*180/pi, label=r"numerical $\theta$")
plt.plot(T, np.array(X_linear)*180/pi, '-.', label=r"linear $\theta$")
# plt.axhline(y=x_init*180/pi, color='r', linestyle='--', label=r"angular amplitude")
# plt.axhline(y=-x_init*180/pi, color='r', linestyle='--',)
plt.title(r"Nonlinear pendulum with $\theta_0 = 124^\circ$")
plt.xlabel(r"Time (s)")
plt.ylabel(r"Angle (degree)")
plt.legend(loc="center right")
plt.xticks(np.arange(0, t_final+1, 1))
plt.show()

E = m*g*l*(1-np.cos(np.array(X))) + 1/2*m*l**2*np.array(V)**2

plt.plot(T, E, label=r"numerical $E$")
plt.axhline(y=m*g*l*(1-cos(x_init)), color='r', linestyle='--', label=r"initial $E$")
plt.title(r"Nonlinear pendulum with $\theta_0 = 124^\circ$")
plt.legend(loc="center right")
plt.xlabel(r"Time (s)")
plt.ylabel(r"Energy (J)")
```

```
plt.xticks(np.arange(0, t_final+1, 1))
# plt.ylim(0, max(E))
plt.show()

print(f"initial energy $E$ = {m*g*l*(1-cos(x_init)):.3f} J")
print(f"standard deviation of $E$ = {np.std(E)}")
```