Math 225B Homework 1

Nhat Thanh Tran

February 6, 2023

In this project we will use various of numerical method to solve the pendulum equation:

$$\frac{dp}{dt} = -\sin(q), \quad \frac{dq}{dt} = p$$

over the time interval $[0, 20]$ with initial data $(p, q)(0) = (0, 2)$ by the followings methods.

1. Euler's method at time step $h = 0.08, 0.04$. Plot the energy $E = p^2/2 + 1(1 - \cos(q))$ along the solutions.

2. Symplectic Euler's method:

$$q_{n+1} = q_n + hp_n, p_{n+1} = p_n - h\sin(q_{n+1})$$

   at $h = 0.08, 0.04$. Plot the energy E and numerical energy $E_{num} = p^2/2 + 0.5hp\sin(q) + (1 - \cos(q))$ along the solutions.

3. Modified Euler with Butcher tableau:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array}$$

Table 1: Butcher tableau for Modified Euler

   at $h = 0.08, 0.04$ Plot the energy E along the solutions.

4. Implicit midpoint method with Butcher tableau:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 0 & 1/2 \\ \hline & 0 & 1 \end{array}$$

Table 2: Butcher tableau for Modified Euler

   at $h = 0.08, 0.04$ Plot the energy E, E_{num} along the solutions.

   For the explicit and symplectic Euler, we implemented the update as given in the problem. For the modified Euler we have the update step as follow:

$$\begin{aligned} &f((p, q), t) = (-\sin(q), p), \\ &y_n = (p_n, q_n) \\ &k_1 = hf(y_n, t_n), \quad k_2 = hf(y_n + hk_1, t_n + h), \\ &y_{n+1} = y_n + 1/2(k_1 + k_2). \end{aligned}$$

For the implicit midpoint method we have:

$$
\begin{aligned}
&f((p,q),t) = (-\sin(q), p),\\
&y_n = (p_n, q_n),\\
&k_1 = hf(y_n, t_n), \quad k_2 = hf(y_n + 1/2k_2, t_n + 1/2h),\\
&\text{(we solve } k_2 \text{ using Newton root finding algorithm)},\\
&y_{n+1} = y_n + k_2.
\end{aligned}
$$

We implemented the code for each of the problem in Python. See the appendix for all of the code. We plot the phase plane and energy along the solutions. For explicit Euler, from Figure 1 we can see that the trajectory of the phase plan traverse outward, this also indicate by their energy increase over time. The larger the h value the larger the energy grows and faster the trajectory go away from the cycle. For modified Euler, from Figure 2 we observe that the phase plan show a cycle of the solution, however the energy plots indicate that the energy in the system increase over time. The change is very small compare to the explicit Euler. And similar to the explicit Euler, the energy grows faster for larger h.

Now, we use Figure 3 to examine the symplectic Euler. We observe that the phase plane shows a cycle solution. Also the energy in the system is oscillating but does not increase or decrease overtime. This demonstrate that the method does conserve the energy up to oscillation. The numerical energy plots show similar behavior. Also, the larger the h value, the larger the magnitude of the oscillation. From the plot we can see that the magnitude in the oscillation match with the value of $O(h)$, namely, for $h = 0.08$ the magnitude is about 0.1, whereas for $h = 0.04$, the magnitude is about 0.04. For numerical energy, the magnitude is about 0.001, 0.0002 respectively.

Lastly, Figure 4 shows the results for implicit method. The phase plane show that this is a cycle solution. Energy plots show the energy stays constant up to an order of about $O(h^2)$ in oscillation. Numerical energy on the other hand behave like the energy in the symplectic Euler.

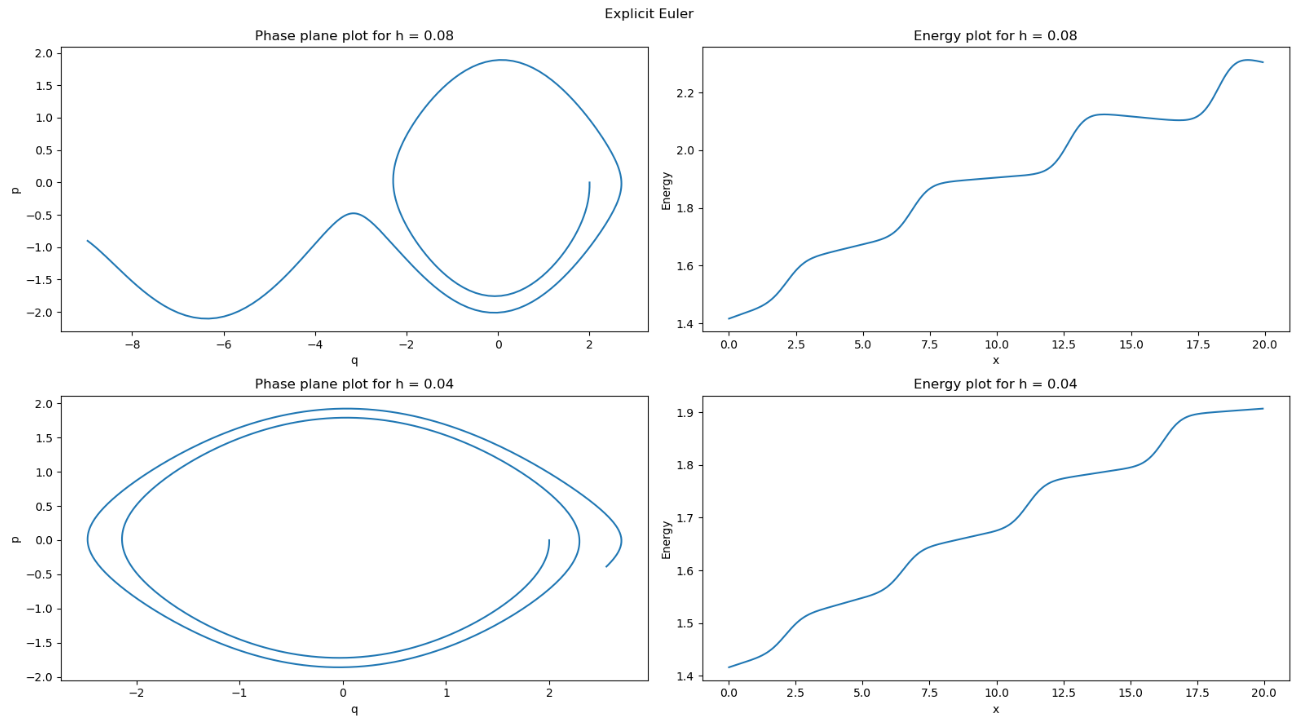

Figure 1: Explicit Euler phase plane and energy

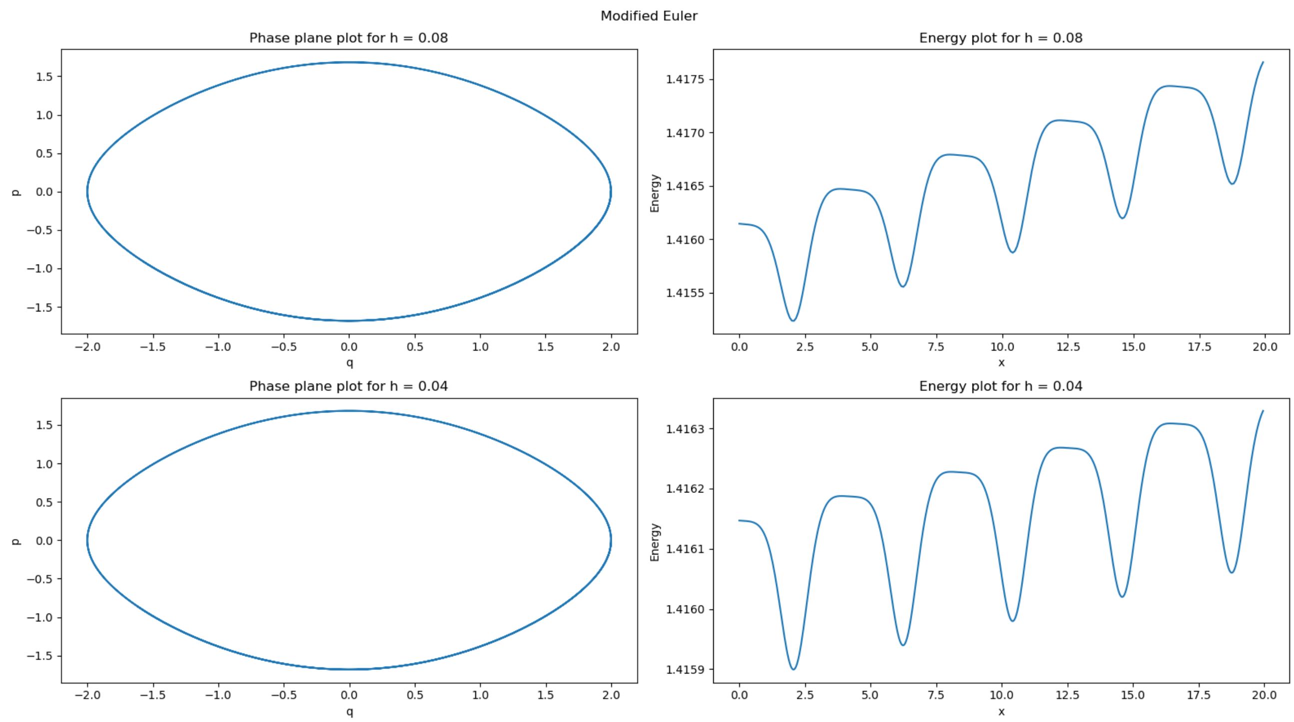

Figure 2: Modified Euler phase plane and energy

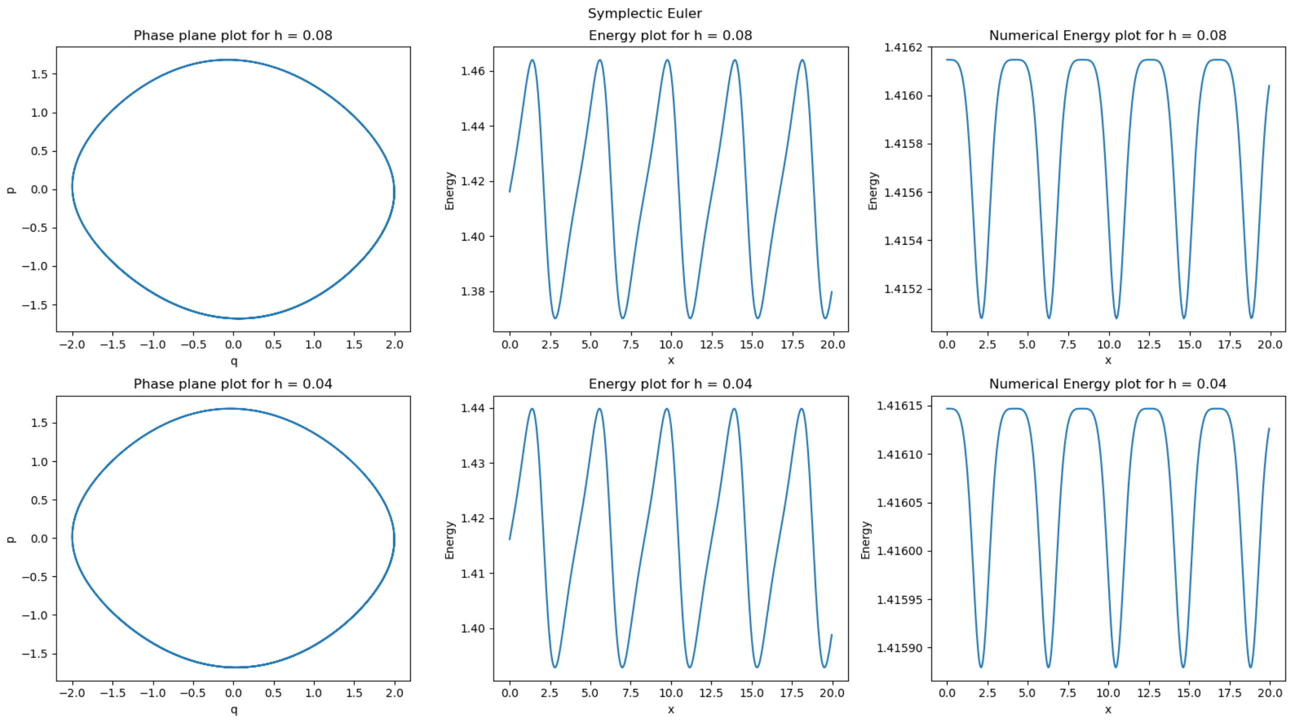

Figure 3: Symplectic Euler phase plane and energy

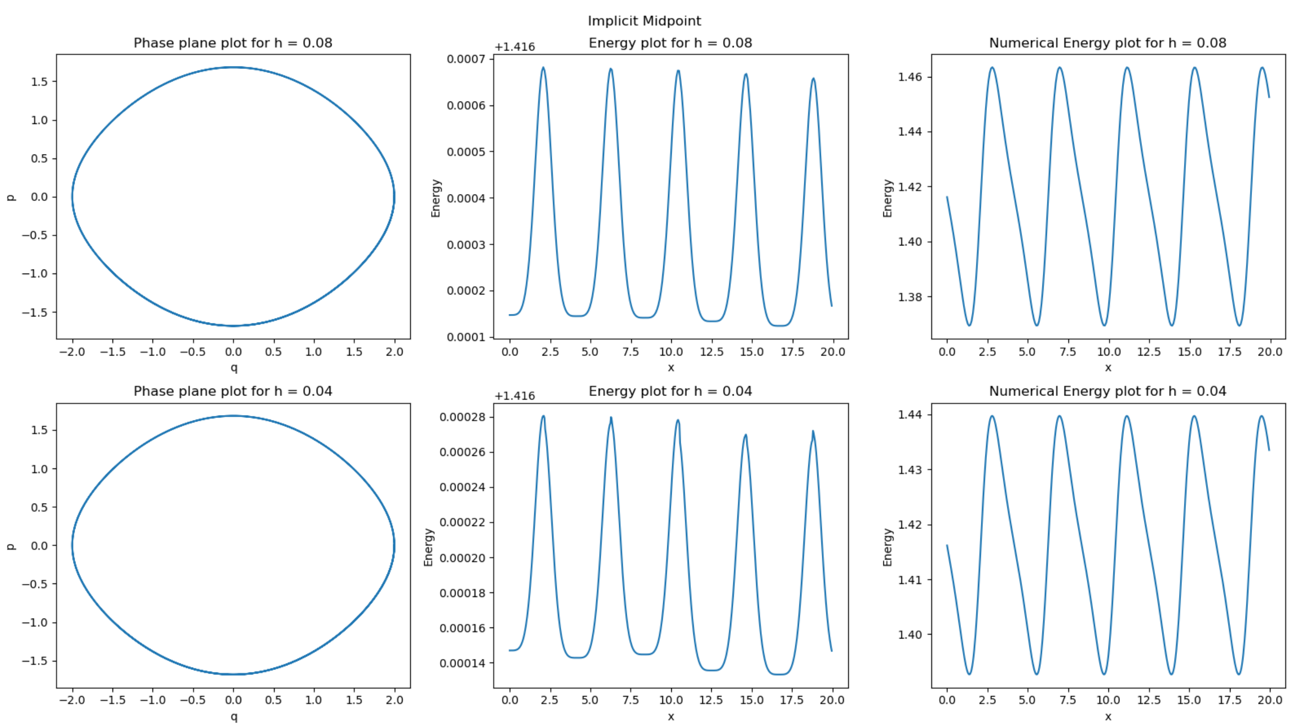

Figure 4: Implicit Midpoint phase plane and energy

Appendix

```python
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import ctypes
import itertools as it
from scipy.optimize import newton_krylov

def explicit_euler(f, t, h, initial):
    initial = initial.reshape(-1,1)
    t = t.reshape(-1,1)
    n, _ = initial.shape
    nt, _ = t.shape
    arr_results = np.zeros((n, nt))
    arr_results[:,0] = initial.flatten()
    for i in range(len(t)-1):
        arr_results[:,i+1] = arr_results[:,i] + h*f(arr_results[:,i].flatten(),t[i]).flatten()

    return arr_results

def symplectic_euler(f, t, h, initial):
    initial = initial.reshape(-1,1)
    t = t.reshape(-1,1)
    n, _ = initial.shape
    nt, _ = t.shape
    arr_results = np.zeros((n, nt))
    arr_results[:,0] = initial.flatten()
    for i in range(len(t)-1):
        # compute q
        arr_results[1,i+1] = arr_results[1,i] + h*arr_results[0,i]
        # compute p
        arr_results[0,i+1] = arr_results[0,i] - h*np.sin(arr_results[1,i+1])

    return arr_results

def explicit_euler_mod(f, t, h, initial):
    initial = initial.reshape(-1,1)
    t = t.reshape(-1,1)
    n, _ = initial.shape
    nt, _ = t.shape
    arr_results = np.zeros((n, nt))
    arr_results[:,0] = initial.flatten()
    for i in range(len(t)-1):
        k1 = f(arr_results[:,i],t[i])
        k2 = f(arr_results[:,i] + h*k1,t[i]+h)
        arr_results[:,i+1] = arr_results[:,i] + h*(0.5*k1 + 0.5*k2).flatten()
    return arr_results

def implicit_midpoint(f, t, h, initial):
    initial = initial.reshape(-1,1)
    t = t.reshape(-1,1)
    n, _ = initial.shape
    nt, _ = t.shape
    arr_results = np.zeros((n, nt))
    arr_results[:,0] = initial.flatten()
    k2 = np.zeros((2,1))
    for i in range(len(t)-1):
        k1 = f(arr_results[:,i],t[i])
        # k2[0,0] = newton_krylov(lambda x: k2_eq(x, arr_results[1,i], h),0)
```

```python
        # k2[1,0] = 2*arr_results[0,i]/h
        k2 = newton_krylov(lambda x: x - h*f(arr_results[:,i] + 0.5*x,t[i]+0.5*h), arr_results[:,i])
        arr_results[:,i+1] = arr_results[:,i] + k2.flatten()
    return arr_results


def compute_energy(arr_p):
    return np.power(arr_p[0,:],2)/2 + 1- np.cos(arr_p[1,:]).flatten()

def compute_sym_euler_energy(arr_sol, h):
    return np.power(arr_sol[0,:],2)/2 + 0.5*h*arr_sol[0,:]*np.sin(arr_sol[1,:]) + (1-np.cos(arr_sol[1,:]))



def f(x, t): return np.array([-np.sin(x[1]), x[0]])


a = 0
b = 20
n = 100
initial= np.array([0,2])
h = [0.08, 0.04]
plot_type = ["phase plane", "energy"]


fig, axs = plt.subplots(nrows=len(h), ncols=2, figsize=(16,9))

for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
    t = np.arange(start=a, stop=b, step=step)
    s = explicit_euler(f, t, step, initial)
    energy = compute_energy(s)
    if pt == "energy":
        ax.plot(t, energy)
        ax.set_xlabel("x")
        ax.set_ylabel("Energy")
        ax.set_title(f'Energy plot for h = {step}')
    else:
        ax.plot(s[1,:],s[0,:])
        ax.set_xlabel("q")
        ax.set_ylabel("p")
        ax.set_title(f'Phase plane plot for h = {step}')
    # ax.plot(s[1,:])

fig.tight_layout()
fig.suptitle("Explicit Euler")
fig.tight_layout()
plt.savefig("project1_expEuler")
plt.close

# Symplectic
fig, axs = plt.subplots(nrows=len(h), ncols=3, figsize=(16,9))
plot_type = ["phase plane", "energy", "numerical energy"]
for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
    t = np.arange(start=a, stop=b, step=step)
    s = symplectic_euler(f, t, step, initial)

    if pt == "energy":
        energy = compute_energy(s)
        ax.plot(t, energy)
        ax.set_xlabel("x")
```

```
        ax.set_ylabel("Energy")
        ax.set_title(f'Energy plot for h = {step}')
    elif pt == "numerical energy":
        energy = compute_sym_euler_energy(s, step)
        ax.plot(t, energy)
        ax.set_xlabel("x")
        ax.set_ylabel("Energy")
        ax.set_title(f'Numerical Energy plot for h = {step}')
    else:
        ax.plot(s[1,:],s[0,:])
        ax.set_xlabel("q")
        ax.set_ylabel("p")
        ax.set_title(f'Phase plane plot for h = {step}')
    # ax.plot(s[1,:])

fig.suptitle("Symplectic Euler")
fig.tight_layout()
plt.savefig("project1_symEuler")
plt.close

# Modified Euler
plot_type = ["phase plane", "energy"]


fig, axs = plt.subplots(nrows=len(h), ncols=2, figsize=(16,9))

for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
    t = np.arange(start=a, stop=b, step=step)
    s = explicit_euler_mod(f, t, step, initial)
    energy = compute_energy(s)
    if pt == "energy":
        ax.plot(t, energy)
        ax.set_xlabel("x")
        ax.set_ylabel("Energy")
        ax.set_title(f'Energy plot for h = {step}')
    else:
        ax.plot(s[1,:],s[0,:])
        ax.set_xlabel("q")
        ax.set_ylabel("p")
        ax.set_title(f'Phase plane plot for h = {step}')
    # ax.plot(s[1,:])

fig.tight_layout()

fig.suptitle("Modified Euler")
fig.tight_layout()
plt.savefig("project1_modEuler")
plt.close

# Implicit Midpoint
fig, axs = plt.subplots(nrows=len(h), ncols=3, figsize=(16,9))
plot_type = ["phase plane", "energy", "numerical energy"]
for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
    t = np.arange(start=a, stop=b, step=step)
    s = implicit_midpoint(f, t, step, initial)

    if pt == "energy":
        energy = compute_energy(s)
        ax.plot(t, energy)
        ax.set_xlabel("x")
        ax.set_ylabel("Energy")
        ax.set_title(f'Energy plot for h = {step}')
```

```python
    elif pt == "numerical energy":
        energy = compute_sym_euler_energy(s, step)
        ax.plot(t, energy)
        ax.set_xlabel("x")
        ax.set_ylabel("Energy")
        ax.set_title(f'Numerical Energy plot for h = {step}')
    else:
        ax.plot(s[1,:],s[0,:])
        ax.set_xlabel("q")
        ax.set_ylabel("p")
        ax.set_title(f'Phase plane plot for h = {step}')
    # ax.plot(s[1,:])

fig.tight_layout()

fig.suptitle("Implicit Midpoint")
fig.tight_layout()
plt.savefig("project1_impMid")
plt.close
```