

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:

0	0	0
1	1	0
<hr/>		
	1/2	1/2

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:

0	0	0
1/2	0	1/2
<hr/>		
	0	1

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:

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

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.

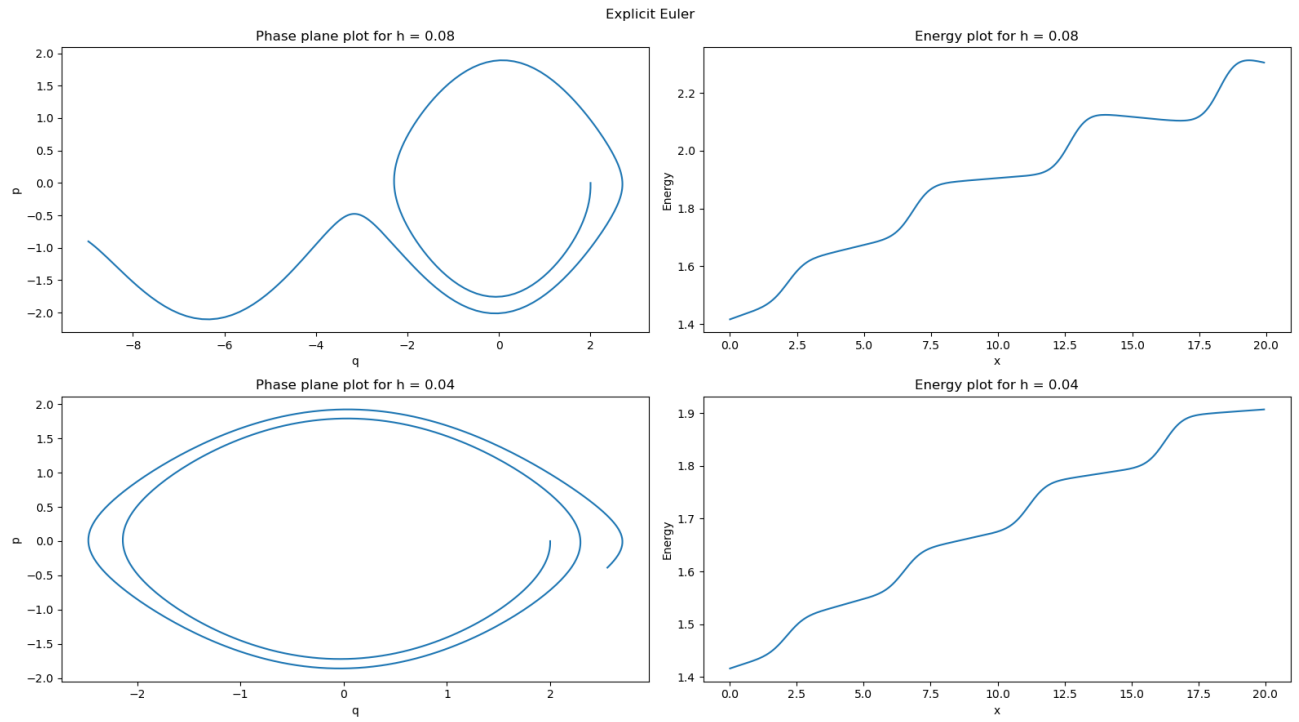


Figure 1: Explicit Euler phase plane and energy

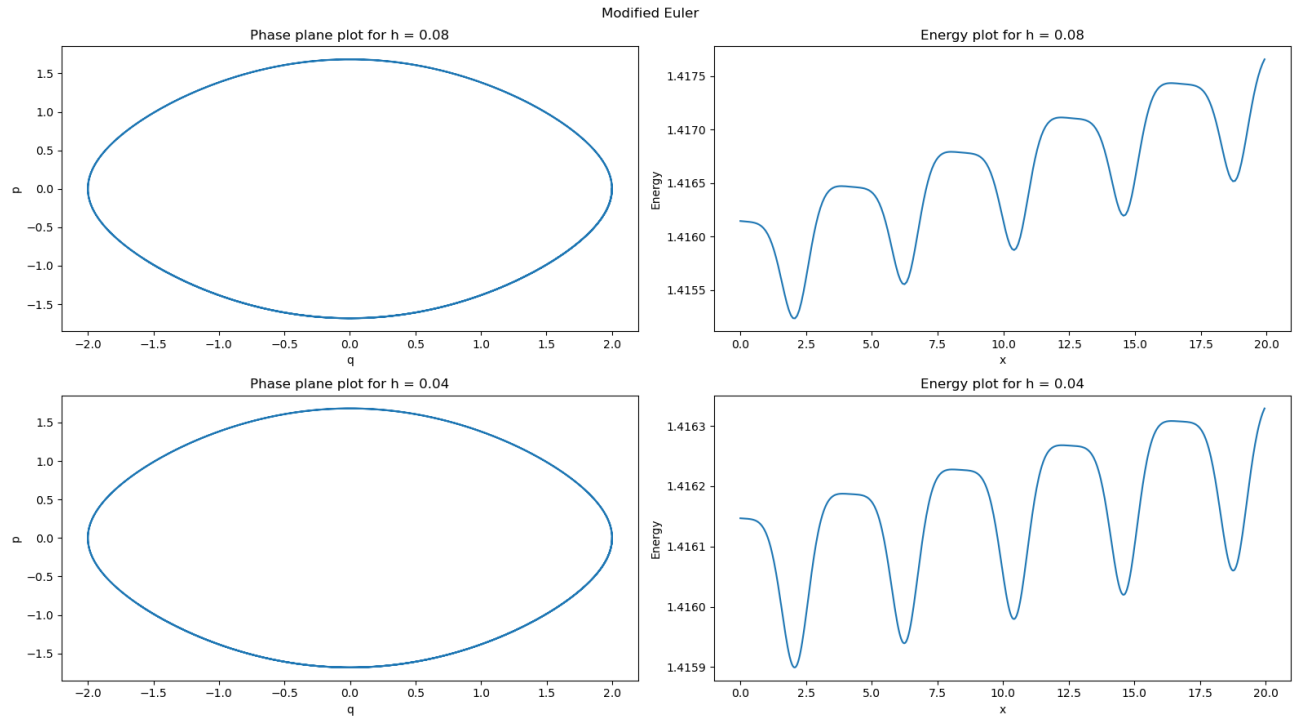


Figure 2: Modified Euler phase plane and energy

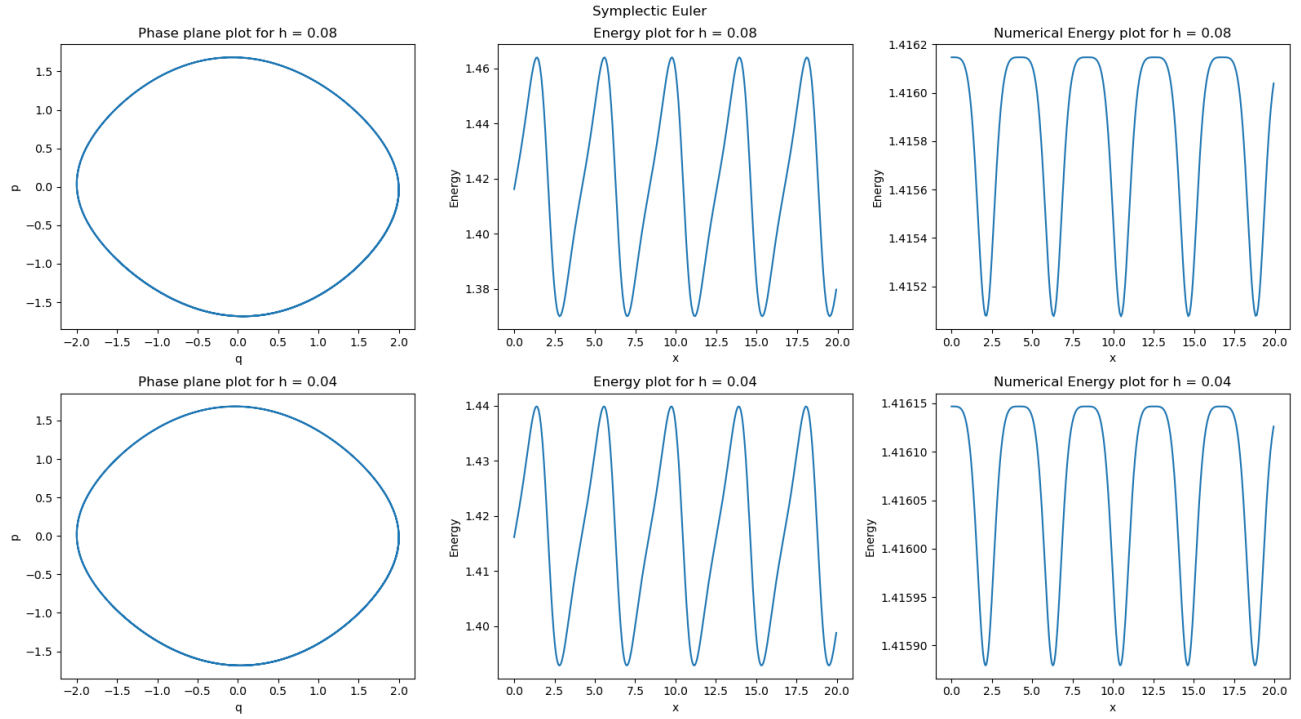


Figure 3: Symplectic Euler phase plane and energy

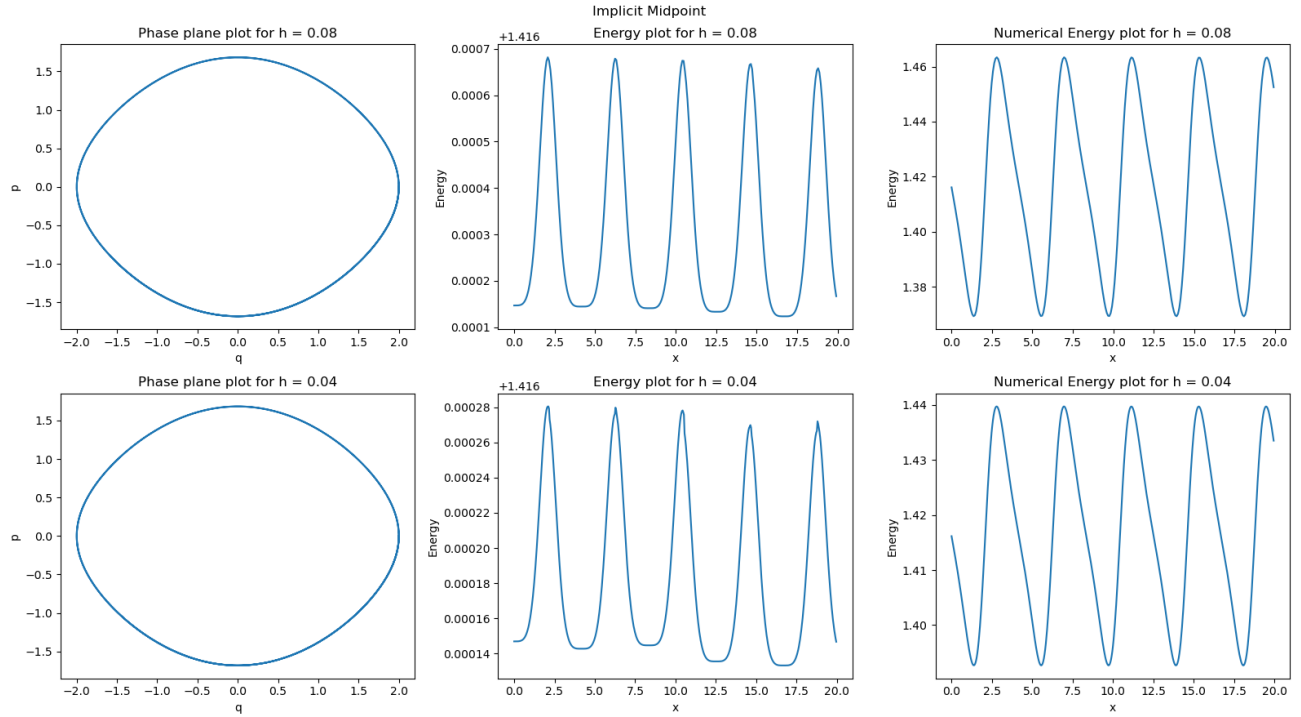


Figure 4: Implicit Midpoint phase plane and energy

Appendix

```
1 import numpy as np
2 import matplotlib as mpl
3 import matplotlib.pyplot as plt
4 import ctypes
5 import itertools as it
6 from scipy.optimize import newton_krylov
7
8 def explicit_euler(f, t, h, initial):
9     initial = initial.reshape(-1,1)
10    t = t.reshape(-1,1)
11    n, _ = initial.shape
12    nt, _ = t.shape
13    arr_results = np.zeros((n, nt))
14    arr_results[:,0] = initial.flatten()
15    for i in range(len(t)-1):
16        arr_results[:,i+1] = arr_results[:,i] + h*f(arr_results[:,i].flatten(),t[i]).
17        flatten()
18    return arr_results
19
20 def symplectic_euler(f, t, h, initial):
21     initial = initial.reshape(-1,1)
22     t = t.reshape(-1,1)
23     n, _ = initial.shape
24     nt, _ = t.shape
25     arr_results = np.zeros((n, nt))
26     arr_results[:,0] = initial.flatten()
27     for i in range(len(t)-1):
28         # compute q
29         arr_results[1,i+1] = arr_results[1,i] + h*arr_results[0,i]
30         # compute p
31         arr_results[0,i+1] = arr_results[0,i] - h*np.sin(arr_results[1,i+1])
32
33     return arr_results
34
35 def explicit_euler_mod(f, t, h, initial):
36     initial = initial.reshape(-1,1)
37     t = t.reshape(-1,1)
38     n, _ = initial.shape
39     nt, _ = t.shape
40     arr_results = np.zeros((n, nt))
41     arr_results[:,0] = initial.flatten()
42     for i in range(len(t)-1):
43         k1 = f(arr_results[:,i],t[i])
44         k2 = f(arr_results[:,i] + h*k1,t[i]+h)
45         arr_results[:,i+1] = arr_results[:,i] + h*(0.5*k1 + 0.5*k2).flatten()
46     return arr_results
47
48 def implicit_midpoint(f, t, h, initial):
49     initial = initial.reshape(-1,1)
50     t = t.reshape(-1,1)
51     n, _ = initial.shape
52     nt, _ = t.shape
53     arr_results = np.zeros((n, nt))
54     arr_results[:,0] = initial.flatten()
55     k2 = np.zeros((2,1))
56     for i in range(len(t)-1):
57         k1 = f(arr_results[:,i],t[i])
58         # k2[0,0] = newton_krylov(lambda x: k2_eq(x, arr_results[1,i], h),0)
```

```

59     # k2[1,0] = 2*arr_results[0,i]/h
60     k2 = newton_krylov(lambda x: x - h*f(arr_results[:,i] + 0.5*x,t[i]+0.5*h),
arr_results[:,i])
61     arr_results[:,i+1] = arr_results[:,i] + k2.flatten()
62     return arr_results
63
64
65 def compute_energy(arr_p):
66     return np.power(arr_p[0,:],2)/2 + 1- np.cos(arr_p[1,:]).flatten()
67
68 def compute_sym_euler_energy(arr_sol, h):
69     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,:]))
70
71
72
73 def f(x, t): return np.array([-np.sin(x[1]), x[0]])
74
75
76 a = 0
77 b = 20
78 n = 100
79 initial= np.array([0,2])
80 h = [0.08, 0.04]
81 plot_type = ["phase plane", "energy"]
82
83
84 fig, axs = plt.subplots(nrows=len(h), ncols=2, figsize=(16,9))
85
86 for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
87     t = np.arange(start=a, stop=b, step=step)
88     s = explicit_euler(f, t, step, initial)
89     energy = compute_energy(s)
90     if pt == "energy":
91         ax.plot(t, energy)
92         ax.set_xlabel("x")
93         ax.set_ylabel("Energy")
94         ax.set_title(f'Energy plot for h = {step}')
95     else:
96         ax.plot(s[1:],s[0,:])
97         ax.set_xlabel("q")
98         ax.set_ylabel("p")
99         ax.set_title(f'Phase plane plot for h = {step}')
100     # ax.plot(s[1,:])
101
102 fig.tight_layout()
103 fig.suptitle("Explicit Euler")
104 fig.tight_layout()
105 plt.savefig("project1_expEuler")
106 plt.close
107
108 # Symplectic
109 fig, axs = plt.subplots(nrows=len(h), ncols=3, figsize=(16,9))
110 plot_type = ["phase plane", "energy", "numerical energy"]
111 for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
112     t = np.arange(start=a, stop=b, step=step)
113     s = symplectic_euler(f, t, step, initial)
114
115     if pt == "energy":
116         energy = compute_energy(s)
117         ax.plot(t, energy)
118         ax.set_xlabel("x")

```

```

119         ax.set_ylabel("Energy")
120         ax.set_title(f'Energy plot for h = {step}')
121     elif pt == "numerical energy":
122         energy = compute_sym_euler_energy(s, step)
123         ax.plot(t, energy)
124         ax.set_xlabel("x")
125         ax.set_ylabel("Energy")
126         ax.set_title(f'Numerical Energy plot for h = {step}')
127     else:
128         ax.plot(s[1,:],s[0,:])
129         ax.set_xlabel("q")
130         ax.set_ylabel("p")
131         ax.set_title(f'Phase plane plot for h = {step}')
132     # ax.plot(s[1,:])
133
134 fig.suptitle("Symplectic Euler")
135 fig.tight_layout()
136 plt.savefig("project1_symEuler")
137 plt.close
138
139 # Modified Euler
140 plot_type = ["phase plane", "energy"]
141
142
143 fig, axs = plt.subplots(nrows=len(h), ncols=2, figsize=(16,9))
144
145 for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
146     t = np.arange(start=a, stop=b, step=step)
147     s = explicit_euler_mod(f, t, step, initial)
148     energy = compute_energy(s)
149     if pt == "energy":
150         ax.plot(t, energy)
151         ax.set_xlabel("x")
152         ax.set_ylabel("Energy")
153         ax.set_title(f'Energy plot for h = {step}')
154     else:
155         ax.plot(s[1,:],s[0,:])
156         ax.set_xlabel("q")
157         ax.set_ylabel("p")
158         ax.set_title(f'Phase plane plot for h = {step}')
159     # ax.plot(s[1,:])
160
161 fig.tight_layout()
162
163 fig.suptitle("Modified Euler")
164 fig.tight_layout()
165 plt.savefig("project1_modEuler")
166 plt.close
167
168 # Implicit Midpoint
169 fig, axs = plt.subplots(nrows=len(h), ncols=3, figsize=(16,9))
170 plot_type = ["phase plane", "energy", "numerical energy"]
171 for (step, pt), ax in zip(it.product(h,plot_type),axs.flat):
172     t = np.arange(start=a, stop=b, step=step)
173     s = implicit_midpoint(f, t, step, initial)
174
175     if pt == "energy":
176         energy = compute_energy(s)
177         ax.plot(t, energy)
178         ax.set_xlabel("x")
179         ax.set_ylabel("Energy")
180         ax.set_title(f'Energy plot for h = {step}')

```

```

181 elif pt == "numerical energy":
182     energy = compute_sym_euler_energy(s, step)
183     ax.plot(t, energy)
184     ax.set_xlabel("x")
185     ax.set_ylabel("Energy")
186     ax.set_title(f'Numerical Energy plot for h = {step}')
187 else:
188     ax.plot(s[1,:],s[0,:])
189     ax.set_xlabel("q")
190     ax.set_ylabel("p")
191     ax.set_title(f'Phase plane plot for h = {step}')
192     # ax.plot(s[1,:])
193
194 fig.tight_layout()
195
196 fig.suptitle("Implicit Midpoint")
197 fig.tight_layout()
198 plt.savefig("project1_impMid")
199 plt.close

```