

toymodel.py

```
1 #%% Import libraries
2 import sympy as sp
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from scipy.integrate import solve_ivp
6
7 #%% a)
8 r, mu, omega, nu, theta = sp.symbols('r mu omega nu theta', positive=True)
9
10 r_eq = mu * r - r**3
11 r_0 = sp.solve(r_eq, r)
12
13 r_0 = [sol for sol in r_0 if sol.is_real and sol > 0][0]
14
15 dtheta_dt = omega + nu * r_0**2
16 T = 2 * sp.pi / dtheta_dt
17
18 print("Radius of the limit cycle (r_0):", r_0)
19 print("Period of the limit cycle (T):", T)
20 # %% b)
21 mu = 1 / 10
22 nu = 1
23 omega = 1
24
25 def system(t, state):
26     x1, x2 = state
27     dx1 = mu * x1 + nu * x1**2 * x2 - x1 * x2**2 - x1**3 + nu * x2**3 + omega * x2
28     dx2 = -nu * x1**3 - nu * x1 * x2**2 - x1**2 * x2 - omega * x1 + mu * x2 - x2**3
29     return [dx1, dx2]
30
31 fixed_point = [0, 0]
32
33 # Time intervals
34 time_start = 0
35 time_span_close = 50
36 time_span_far = 7
37 time_span_limit_cycle = 12
38
39 distance_close = 0.02
40 distance_far = 0.5
41 distance_limit_cycle = np.sqrt(mu) - 0.001
42
43 initial_conditions_close = [
44     [fixed_point[0] + distance_close, fixed_point[1] + distance_close],
45     [fixed_point[0] - distance_close, fixed_point[1] + distance_close],
46     [fixed_point[0] + distance_close, fixed_point[1] - distance_close],
47     [fixed_point[0] - distance_close, fixed_point[1] - distance_close]
48 ]
49
50 initial_conditions_far = [
51     [fixed_point[0] + distance_far, fixed_point[1]],
```

```

52     [fixed_point[0] - distance_far, fixed_point[1]],
53     [fixed_point[0], fixed_point[1] + distance_far],
54     [fixed_point[0], fixed_point[1] - distance_far]
55 ]
56
57 initial_condition_limit_cycle = [fixed_point[0] + distance_limit_cycle, fixed_point[1]]
58
59 def solve_and_plot(initial_conditions, t_span, color, label):
60     for ic in initial_conditions:
61         sol = solve_ivp(system, t_span, ic, t_eval=np.linspace(t_span[0], t_span[1], 500))
62         plt.plot(sol.y[0], sol.y[1], color=color, label=label if label else None)
63         label = None # Ensure label is only used once
64
65 plt.figure(figsize=(10, 8))
66
67 solve_and_plot(initial_conditions_close, (time_start, time_span_close), color='blue',
68 label="Close to Fixed Point")
69
70 solve_and_plot(initial_conditions_far, (time_start, time_span_far), color='green',
71 label="Far from Fixed Point")
72
73 sol_limit_cycle = solve_ivp(
74     system,
75     (time_start, time_span_limit_cycle),
76     initial_condition_limit_cycle,
77     t_eval=np.linspace(time_start, time_span_limit_cycle, 500)
78 )
79 plt.plot(sol_limit_cycle.y[0], sol_limit_cycle.y[1], color='red', label="Limit Cycle",
80 linewidth=2)
81
82 # Add labels and legend
83 plt.xlabel("x1")
84 plt.ylabel("x2")
85 plt.title("Phase portraits")
86 plt.legend()
87 plt.grid()
88 plt.show()
89
90
91 # %% d)
92 t = sp.symbols('t')
93 X1, X2 = sp.symbols('x1 x2', cls=sp.symbols)
94 mu = 1/10
95 omega = 1
96 nu = 1
97
98 f1 = mu * X1 - nu * X1**2 * X2 - X1 * X2**2 - X1**3 - nu * X2**3 - omega * X2
99 f2 = nu * X1**3 + nu * X1 * X2**2 - X1**2 * X2 + omega * X1 + mu * X2 - X2**3
100 H11 = sp.diff(f1, X1)
101 H12 = sp.diff(f1, X2)
102 H21 = sp.diff(f2, X1)
103 H22 = sp.diff(f2, X2)

```

```

103 J11_func = sp.lambdify((X1, X2), H11)
104 J12_func = sp.lambdify((X1, X2), H12)
105 J21_func = sp.lambdify((X1, X2), H21)
106 J22_func = sp.lambdify((X1, X2), H22)
107
108 def system(t, Y):
109     x1, x2, M11, M12, M21, M22 = Y
110
111
112     J11_val = J11_func(x1, x2)
113     J12_val = J12_func(x1, x2)
114     J21_val = J21_func(x1, x2)
115     J22_val = J22_func(x1, x2)
116
117     dx1_dt = mu * x1 - nu * x1**2 * x2 - x1 * x2**2 - x1**3 - nu * x2**3 - omega * x2
118     dx2_dt = nu * x1**3 + nu * x1 * x2**2 - x1**2 * x2 + omega * x1 + mu * x2 - x2**3
119
120     dM11_dt = J11_val * M11 + J12_val * M21
121     dM12_dt = J11_val * M12 + J12_val * M22
122     dM21_dt = J21_val * M11 + J22_val * M21
123     dM22_dt = J21_val * M12 + J22_val * M22
124
125     return [dx1_dt, dx2_dt, dM11_dt, dM12_dt, dM21_dt, dM22_dt]
126
127 x1_0 = np.sqrt(mu)
128 x2_0 = 0
129 M11_0, M12_0 = 1, 0
130 M21_0, M22_0 = 0, 1
131
132 Y0 = [x1_0, x2_0, M11_0, M12_0, M21_0, M22_0]
133
134 t0 = 0
135 t_max = 2 * np.pi / (omega + nu * mu)
136 t_span = (t0, t_max)
137 t_eval = np.linspace(t0, t_max, 1000)
138
139 sol = solve_ivp(system, t_span, Y0, t_eval=t_eval)
140
141 t_vals = sol.t
142 x1_vals, x2_vals, M11_vals, M12_vals, M21_vals, M22_vals = sol.y
143
144 plt.figure(figsize=(10, 6))
145 plt.plot(t_vals, x1_vals, label='x1', color='red', linewidth=2)
146 plt.plot(t_vals, x2_vals, label='x2', color='green', linewidth=2)
147 plt.plot(t_vals, M11_vals, label='M11', color='blue', linewidth=2)
148 plt.plot(t_vals, M12_vals, label='M12', color='orange', linewidth=2)
149 plt.plot(t_vals, M21_vals, label='M21', color='purple', linewidth=2)
150 plt.plot(t_vals, M22_vals, label='M22', color='brown', linewidth=2)
151
152 # Add labels and legends
153 plt.xlabel('Time')
154 plt.ylabel('Values')
155 plt.title('Trajectories of x1, x2, M11, M12, M21, M22')
156 plt.legend()

```

```

157 plt.grid(True)
158 plt.show()
159
160 # %% e)
161 M11_at_t_max = M11_vals[-1]
162 M12_at_t_max = M12_vals[-1]
163 M21_at_t_max = M21_vals[-1]
164 M22_at_t_max = M22_vals[-1]
165
166 M_matrix = np.round([[M11_at_t_max, M12_at_t_max], [M21_at_t_max, M22_at_t_max]], 4)
167
168 print("Matrix at t = t_max:")
169 print(M_matrix)
170 # %% f
171 solMat = np.array([[M11_at_t_max, M12_at_t_max],
172                     [M21_at_t_max, M22_at_t_max]])
173
174 sigma_12 = np.linalg.eigvals(solMat)
175
176 sigma_tilde = np.round([1 / t_max * np.log(sigma_12[1]), 1 / t_max * np.log(sigma_12[0])],
177 4)
178
179 print(sigma_tilde)
180 # %% f
181
182 X1, X2, mu, nu, omega, r, T = sp.symbols('x1 x2 mu nu omega r T', real=True)
183
184 H11 = sp.diff(sp.sqrt(X1**2 + X2**2), X1)
185 H12 = sp.diff(sp.sqrt(X1**2 + X2**2), X2)
186 H21 = sp.diff(sp.atan2(X2, X1), X1)
187 H22 = sp.diff(sp.atan2(X2, X1), X2)
188
189 H = sp.Matrix([[H11, H12], [H21, H22]])
190
191 H_Inv = H.inv()
192
193 J = sp.Matrix([[mu - 3 * r**2, 0], [2 * nu * r, 0]])
194
195 M_P = sp.exp(J * T)
196
197 T_val = 2 * sp.pi / (omega + nu * mu)
198 X1_val = sp.sqrt(mu)
199 X2_val = 0
200 r_val = sp.sqrt(X1_val**2 + X2_val**2)
201
202 H_val = H.subs({X1: X1_val, X2: X2_val})
203 Hinv_val = H_Inv.subs({X1: X1_val, X2: X2_val})
204 Mp_val = M_P.subs({T: T_val, r: r_val, mu: mu, nu: nu})
205
206 M_C = (Hinv_val * Mp_val * H_val).simplify()
207
208 M_C
209 # %% g

```

```
210 #This is done here to avoid exact values earlier
211 mu_val = 1/10
212 nu_val = 1
213 omega_val = 1
214
215 subs_values = {mu: mu_val, nu: nu_val, omega: omega_val}
216 M_C_val = M_C.subs(subs_values)
217
218 sigma_12 = M_C_val.eigenvals().keys()
219
220 T_val_sub = T_val.subs(subs_values)
221 real_sigma12 = [sp.N(sp.log(ev) / T_val_sub, 4) for ev in sigma_12]
222
223 # Display the results
224 real_sigma12
225
226 # %%
```