Homework Assignment 4.2

Max Green TIF155

December 11, 2025

Exercise a

(See last pages for code and code output).
From the code: $r_0 = \sqrt{\mu}, T = \frac{2\pi}{\mu\nu+\omega}$

Exercise b

$$r = \sqrt{X_1^2 + X_2^2}, \quad \theta = \arctan\left(\frac{X_2}{X_1}\right).$$

$$\dot{X}_1 = \frac{d}{dt}[r\cos(\theta)] = \dot{r}\cos(\theta) - r\sin(\theta)\dot{\theta},$$
$$\dot{X}_2 = \frac{d}{dt}[r\sin(\theta)] = \dot{r}\sin(\theta) + r\cos(\theta)\dot{\theta}.$$
$$\dot{r} = \mu r - r^3, \quad \dot{\theta} = \omega + \nu r^2.$$

$$\dot{X}_1 = (\mu - r^2)X_1 - \omega X_2 - \nu(r^2)X_2,$$
$$\dot{X}_2 = (\mu - r^2)X_2 + \omega X_1 + \nu(r^2)X_1,$$
$$\dot{X}_1 = \left(\mu - X_1^2 - X_2^2\right)X_1 - \omega X_2 - \nu(X_1^2 + X_2^2)X_2,$$
$$\dot{X}_2 = \left(\mu - X_1^2 - X_2^2\right)X_2 + \omega X_1 + \nu(X_1^2 + X_2^2)X_1.$$

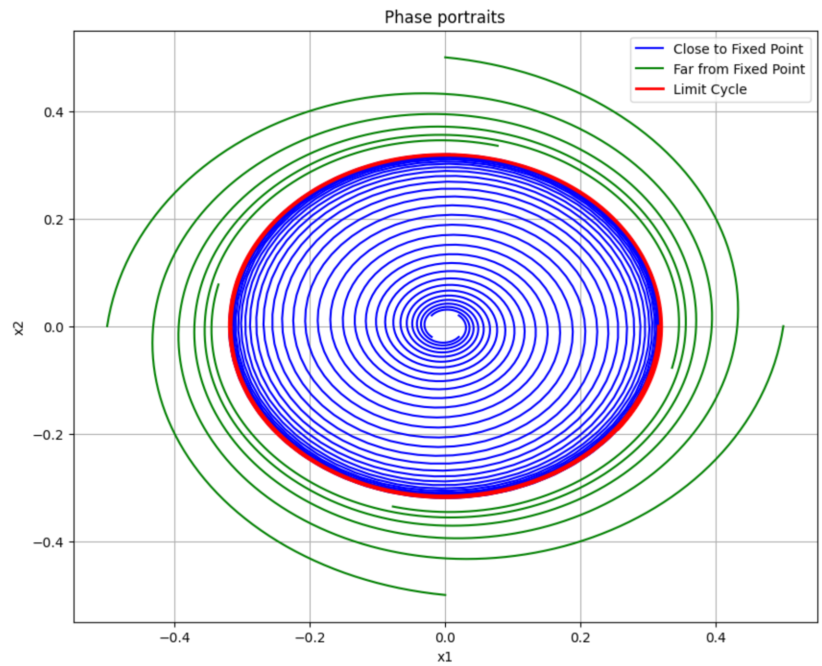


Figure 1: Phase portraits

Exercise c

To make the two systems identical, we compare corresponding terms in the equations for $\dot{X}_1$ and $\dot{X}_2$. This gives:

$$\mu = \frac{1}{10}, \quad \omega = 1, \quad \nu = 1.$$

Exercise d

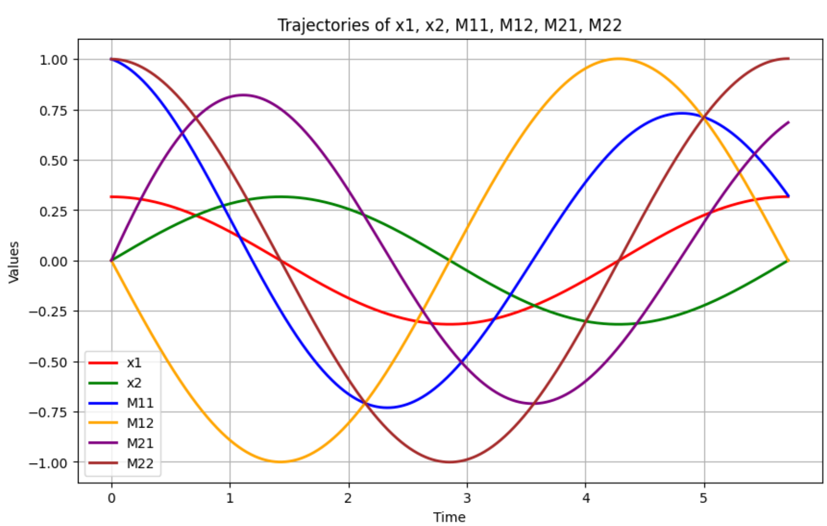


Figure 2: Curves

Exercise e

See code.

Exercise f

See code.

Exercise g

$$\frac{dM}{dt} = J(t)M \to \int_{M(0)}^{M(T)} \frac{dM}{M} = \int_0^T J(t')dt' \tag{1}$$

$$J_P = \begin{bmatrix} \mu - 2r^2 & 0 \\ 2\nu r & 0 \end{bmatrix} \to M_P = e^{JT} \tag{2}$$

Transform to Cartesian with transform matrix H:

$$H = \frac{dP}{dC} = \begin{bmatrix} \frac{dr}{dX_1} & \frac{dr}{dX_2} \\ \frac{d\theta}{dX_1} & \frac{d\theta}{dX_2} \end{bmatrix}, \quad M_C = H^{-1} * M_P * H \tag{3}$$

The code gives

$$M_C = \begin{bmatrix} e^{-\frac{\mu}{\mu\nu+\omega}} & \\ \nu - \nu e^{-\frac{\mu}{\mu\nu+\omega}} & 1 \end{bmatrix} = \begin{bmatrix} e^{-\frac{4\pi}{11}} & 0 \\ 1 - e^{-\frac{4\pi}{11}} & 1 \end{bmatrix} \tag{4}$$

Exercise h

Symbolic analysis of eigenvalues, see code.

toymodel.py

```python
#%% Import libraries
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

#%% a)
r, mu, omega, nu, theta = sp.symbols('r mu omega nu theta', positive=True)

r_eq = mu * r - r**3
r_0 = sp.solve(r_eq, r)

r_0 = [sol for sol in r_0 if sol.is_real and sol > 0][0]

dtheta_dt = omega + nu * r_0**2
T = 2 * sp.pi / dtheta_dt

print("Radius of the limit cycle (r_0):", r_0)
print("Period of the limit cycle (T):", T)
# %% b)
mu = 1 / 10
nu = 1
omega = 1

def system(t, state):
    x1, x2 = state
    dx1 = mu * x1 + nu * x1**2 * x2 - x1 * x2**2 - x1**3 + nu * x2**3 + omega * x2
    dx2 = -nu * x1**3 - nu * x1 * x2**2 - x1**2 * x2 - omega * x1 + mu * x2 - x2**3
    return [dx1, dx2]

fixed_point = [0, 0]

# Time intervals
time_start = 0
time_span_close = 50
time_span_far = 7
time_span_limit_cycle = 12

distance_close = 0.02
distance_far = 0.5
distance_limit_cycle = np.sqrt(mu) - 0.001

initial_conditions_close = [
    [fixed_point[0] + distance_close, fixed_point[1] + distance_close],
    [fixed_point[0] - distance_close, fixed_point[1] + distance_close],
    [fixed_point[0] + distance_close, fixed_point[1] - distance_close],
    [fixed_point[0] - distance_close, fixed_point[1] - distance_close]
]

initial_conditions_far = [
    [fixed_point[0] + distance_far, fixed_point[1]],
```

```
    [fixed_point[0] - distance_far, fixed_point[1]],
    [fixed_point[0], fixed_point[1] + distance_far],
    [fixed_point[0], fixed_point[1] - distance_far]
]

initial_condition_limit_cycle = [fixed_point[0] + distance_limit_cycle, fixed_point[1]]

def solve_and_plot(initial_conditions, t_span, color, label):
    for ic in initial_conditions:
        sol = solve_ivp(system, t_span, ic, t_eval=np.linspace(t_span[0], t_span[1], 500))
        plt.plot(sol.y[0], sol.y[1], color=color, label=label if label else None)
        label = None  # Ensure label is only used once

plt.figure(figsize=(10, 8))

solve_and_plot(initial_conditions_close, (time_start, time_span_close), color='blue', label="Close to Fixed Point")

solve_and_plot(initial_conditions_far, (time_start, time_span_far), color='green', label="Far from Fixed Point")

sol_limit_cycle = solve_ivp(
    system,
    (time_start, time_span_limit_cycle),
    initial_condition_limit_cycle,
    t_eval=np.linspace(time_start, time_span_limit_cycle, 500)
)
plt.plot(sol_limit_cycle.y[0], sol_limit_cycle.y[1], color='red', label="Limit Cycle", linewidth=2)

# Add labels and legend
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Phase portraits")
plt.legend()
plt.grid()
plt.show()


# %% d)
t = sp.symbols('t')
X1, X2 = sp.symbols('x1 x2', cls=sp.symbols)
mu = 1/10
omega = 1
nu = 1

f1 = mu * X1 - nu * X1**2 * X2 - X1 * X2**2 - X1**3 - nu * X2**3 - omega * X2
f2 = nu * X1**3 + nu * X1 * X2**2 - X1**2 * X2 + omega * X1 + mu * X2 - X2**3

H11 = sp.diff(f1, X1)
H12 = sp.diff(f1, X2)
H21 = sp.diff(f2, X1)
H22 = sp.diff(f2, X2)
```

```python
J11_func = sp.lambdify((X1, X2), H11)
J12_func = sp.lambdify((X1, X2), H12)
J21_func = sp.lambdify((X1, X2), H21)
J22_func = sp.lambdify((X1, X2), H22)

def system(t, Y):
    x1, x2, M11, M12, M21, M22 = Y


    J11_val = J11_func(x1, x2)
    J12_val = J12_func(x1, x2)
    J21_val = J21_func(x1, x2)
    J22_val = J22_func(x1, x2)

    dx1_dt = mu * x1 - nu * x1**2 * x2 - x1 * x2**2 - x1**3 - nu * x2**3 - omega * x2
    dx2_dt = nu * x1**3 + nu * x1 * x2**2 - x1**2 * x2 + omega * x1 + mu * x2 - x2**3

    dM11_dt = J11_val * M11 + J12_val * M21
    dM12_dt = J11_val * M12 + J12_val * M22
    dM21_dt = J21_val * M11 + J22_val * M21
    dM22_dt = J21_val * M12 + J22_val * M22

    return [dx1_dt, dx2_dt, dM11_dt, dM12_dt, dM21_dt, dM22_dt]

x1_0 = np.sqrt(mu)
x2_0 = 0
M11_0, M12_0 = 1, 0
M21_0, M22_0 = 0, 1

Y0 = [x1_0, x2_0, M11_0, M12_0, M21_0, M22_0]

t0 = 0
t_max = 2 * np.pi / (omega + nu * mu)
t_span = (t0, t_max)
t_eval = np.linspace(t0, t_max, 1000)

sol = solve_ivp(system, t_span, Y0, t_eval=t_eval)

t_vals = sol.t
x1_vals, x2_vals, M11_vals, M12_vals, M21_vals, M22_vals = sol.y

plt.figure(figsize=(10, 6))
plt.plot(t_vals, x1_vals, label='x1', color='red', linewidth=2)
plt.plot(t_vals, x2_vals, label='x2', color='green', linewidth=2)
plt.plot(t_vals, M11_vals, label='M11', color='blue', linewidth=2)
plt.plot(t_vals, M12_vals, label='M12', color='orange', linewidth=2)
plt.plot(t_vals, M21_vals, label='M21', color='purple', linewidth=2)
plt.plot(t_vals, M22_vals, label='M22', color='brown', linewidth=2)

# Add labels and legends
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Trajectories of x1, x2, M11, M12, M21, M22')
plt.legend()
```

```
plt.grid(True)
plt.show()

# %% e)
M11_at_t_max = M11_vals[-1]
M12_at_t_max = M12_vals[-1]
M21_at_t_max = M21_vals[-1]
M22_at_t_max = M22_vals[-1]

M_matrix = np.round([[M11_at_t_max, M12_at_t_max], [M21_at_t_max, M22_at_t_max]], 4)

print("Matrix at t = t_max:")
print(M_matrix)
# %% f
solMat = np.array([[M11_at_t_max, M12_at_t_max],
                   [M21_at_t_max, M22_at_t_max]])

sigma_12 = np.linalg.eigvals(solMat)

sigma_tilde = np.round([1 / t_max * np.log(sigma_12[1]), 1 / t_max * np.log(sigma_12[0])],
4)

print(sigma_tilde)
# %% f

X1, X2, mu, nu, omega, r, T = sp.symbols('x1 x2 mu nu omega r T', real=True)

H11 = sp.diff(sp.sqrt(X1**2 + X2**2), X1)
H12 = sp.diff(sp.sqrt(X1**2 + X2**2), X2)
H21 = sp.diff(sp.atan2(X2, X1), X1)
H22 = sp.diff(sp.atan2(X2, X1), X2)

H = sp.Matrix([[H11, H12], [H21, H22]])

H_Inv = H.inv()

J = sp.Matrix([[mu - 3 * r**2, 0], [2 * nu * r, 0]])

M_P = sp.exp(J * T)

T_val = 2 * sp.pi / (omega + nu * mu)
X1_val = sp.sqrt(mu)
X2_val = 0
r_val = sp.sqrt(X1_val**2 + X2_val**2)

H_val = H.subs({X1: X1_val, X2: X2_val})
Hinv_val = H_Inv.subs({X1: X1_val, X2: X2_val})
Mp_val = M_P.subs({T: T_val, r: r_val, mu: mu, nu: nu})

M_C = (Hinv_val * Mp_val * H_val).simplify()

M_C

# %% g
```

```python
#This is done here to avoid exact values earlier
mu_val = 1/10
nu_val = 1
omega_val = 1

subs_values = {mu: mu_val, nu: nu_val, omega: omega_val}
M_C_val = M_C.subs(subs_values)

sigma_12 = M_C_val.eigenvals().keys()

T_val_sub = T_val.subs(subs_values)
real_sigma12 = [sp.N(sp.log(ev) / T_val_sub, 4) for ev in sigma_12]

# Display the results
real_sigma12

# %%

```

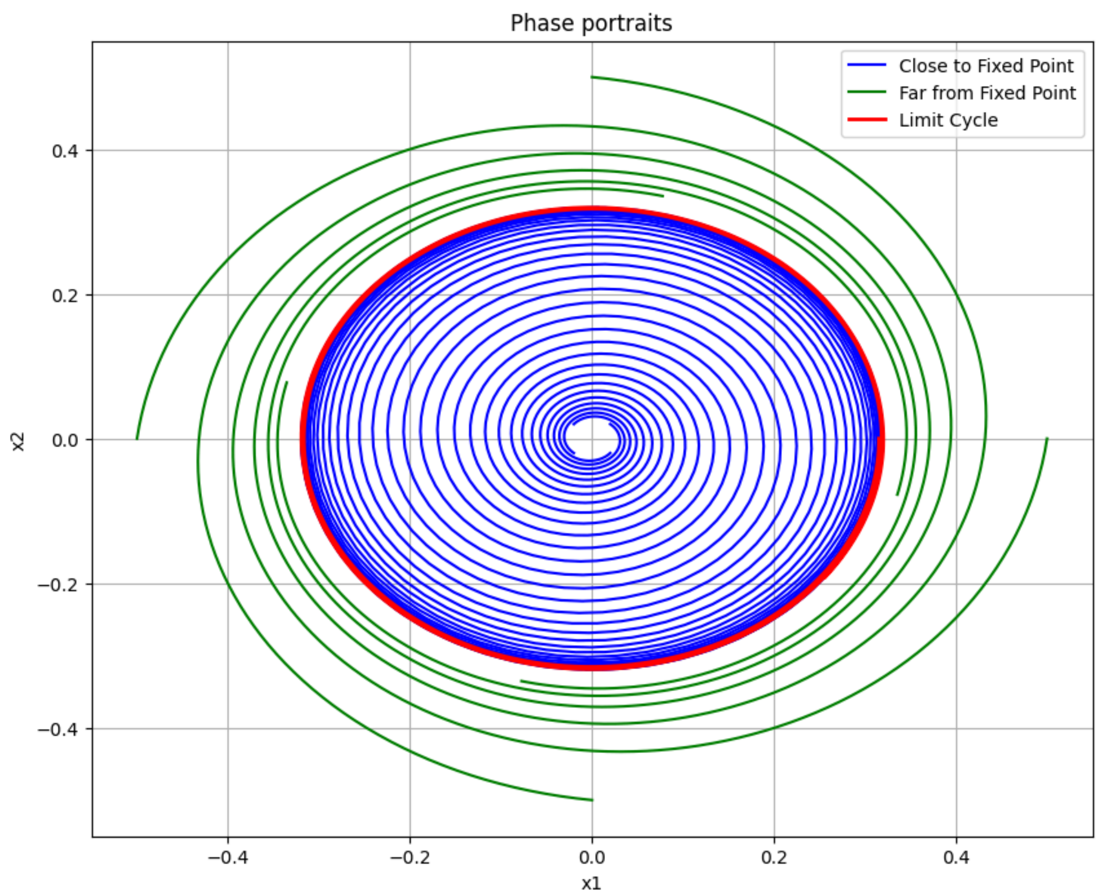
In [ ]:
```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
```

In [ ]:
```
r, mu, omega, nu, theta = sp.symbols('r mu omega nu theta', positive=True)

r_eq = mu * r - r**3
r_0 = sp.solve(r_eq, r)

r_0 = [sol for sol in r_0 if sol.is_real and sol > 0][0]

dtheta_dt = omega + nu * r_0**2
T = 2 * sp.pi / dtheta_dt

print("Radius of the limit cycle (r_0):", r_0)
print("Period of the limit cycle (T):", T)
```

```
Radius of the limit cycle (r_0): sqrt(mu)
Period of the limit cycle (T): 2*pi/(mu*nu + omega)
```

In [ ]:
```
mu = 1 / 10
nu = 1
omega = 1

def system(t, state):
    x1, x2 = state
    dx1 = mu * x1 + nu * x1**2 * x2 - x1 * x2**2 - x1**3 + nu * x2**3 + omega *
    dx2 = -nu * x1**3 - nu * x1 * x2**2 - x1**2 * x2 - omega * x1 + mu * x2 - x2
    return [dx1, dx2]

fixed_point = [0, 0]

# Time intervals
time_start = 0
time_span_close = 50
time_span_far = 7
time_span_limit_cycle = 12

distance_close = 0.02
distance_far = 0.5
distance_limit_cycle = np.sqrt(mu) - 0.001

initial_conditions_close = [
    [fixed_point[0] + distance_close, fixed_point[1] + distance_close],
    [fixed_point[0] - distance_close, fixed_point[1] + distance_close],
    [fixed_point[0] + distance_close, fixed_point[1] - distance_close],
    [fixed_point[0] - distance_close, fixed_point[1] - distance_close]
]

initial_conditions_far = [
    [fixed_point[0] + distance_far, fixed_point[1]],
    [fixed_point[0] - distance_far, fixed_point[1]],
    [fixed_point[0], fixed_point[1] + distance_far],
    [fixed_point[0], fixed_point[1] - distance_far]
]

initial_condition_limit_cycle = [fixed_point[0] + distance_limit_cycle, fixed_po
```

```python
def solve_and_plot(initial_conditions, t_span, color, label):
    for ic in initial_conditions:
        sol = solve_ivp(system, t_span, ic, t_eval=np.linspace(t_span[0], t_span
        plt.plot(sol.y[0], sol.y[1], color=color, label=label if label else None
        label = None  # Ensure label is only used once

plt.figure(figsize=(10, 8))

solve_and_plot(initial_conditions_close, (time_start, time_span_close), color='b

solve_and_plot(initial_conditions_far, (time_start, time_span_far), color='green

sol_limit_cycle = solve_ivp(
    system,
    (time_start, time_span_limit_cycle),
    initial_condition_limit_cycle,
    t_eval=np.linspace(time_start, time_span_limit_cycle, 500)
)
plt.plot(sol_limit_cycle.y[0], sol_limit_cycle.y[1], color='red', label="Limit C

# Add labels and legend
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Phase portraits")
plt.legend()
plt.grid()
plt.show()
```

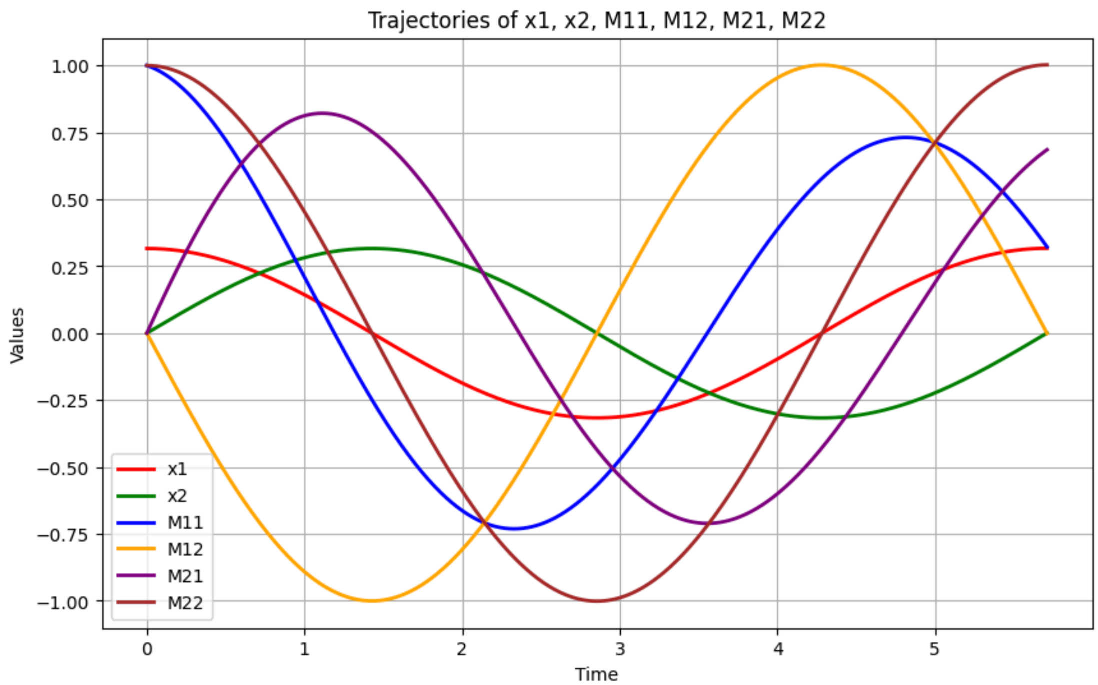


In [ ]:

```python
t = sp.symbols('t')
X1, X2 = sp.symbols('x1 x2', cls=sp.symbols)
```

```python
mu = 1/10
omega = 1
nu = 1

f1 = mu * X1 - nu * X1**2 * X2 - X1 * X2**2 - X1**3 - nu * X2**3 - omega * X2
f2 = nu * X1**3 + nu * X1 * X2**2 - X1**2 * X2 + omega * X1 + mu * X2 - X2**3

H11 = sp.diff(f1, X1)
H12 = sp.diff(f1, X2)
H21 = sp.diff(f2, X1)
H22 = sp.diff(f2, X2)

J11_func = sp.lambdify((X1, X2), H11)
J12_func = sp.lambdify((X1, X2), H12)
J21_func = sp.lambdify((X1, X2), H21)
J22_func = sp.lambdify((X1, X2), H22)

def system(t, Y):
    x1, x2, M11, M12, M21, M22 = Y


    J11_val = J11_func(x1, x2)
    J12_val = J12_func(x1, x2)
    J21_val = J21_func(x1, x2)
    J22_val = J22_func(x1, x2)

    dx1_dt = mu * x1 - nu * x1**2 * x2 - x1 * x2**2 - x1**3 - nu * x2**3 - omega
    dx2_dt = nu * x1**3 + nu * x1 * x2**2 - x1**2 * x2 + omega * x1 + mu * x2 -

    dM11_dt = J11_val * M11 + J12_val * M21
    dM12_dt = J11_val * M12 + J12_val * M22
    dM21_dt = J21_val * M11 + J22_val * M21
    dM22_dt = J21_val * M12 + J22_val * M22

    return [dx1_dt, dx2_dt, dM11_dt, dM12_dt, dM21_dt, dM22_dt]

x1_0 = np.sqrt(mu)
x2_0 = 0
M11_0, M12_0 = 1, 0
M21_0, M22_0 = 0, 1

Y0 = [x1_0, x2_0, M11_0, M12_0, M21_0, M22_0]

t0 = 0
t_max = 2 * np.pi / (omega + nu * mu)
t_span = (t0, t_max)
t_eval = np.linspace(t0, t_max, 1000)

sol = solve_ivp(system, t_span, Y0, t_eval=t_eval)

t_vals = sol.t
x1_vals, x2_vals, M11_vals, M12_vals, M21_vals, M22_vals = sol.y

plt.figure(figsize=(10, 6))
plt.plot(t_vals, x1_vals, label='x1', color='red', linewidth=2)
plt.plot(t_vals, x2_vals, label='x2', color='green', linewidth=2)
plt.plot(t_vals, M11_vals, label='M11', color='blue', linewidth=2)
plt.plot(t_vals, M12_vals, label='M12', color='orange', linewidth=2)
plt.plot(t_vals, M21_vals, label='M21', color='purple', linewidth=2)
plt.plot(t_vals, M22_vals, label='M22', color='brown', linewidth=2)
```

```python
# Add labels and legends
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Trajectories of x1, x2, M11, M12, M21, M22')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```python
M11_at_t_max = M11_vals[-1]
M12_at_t_max = M12_vals[-1]
M21_at_t_max = M21_vals[-1]
M22_at_t_max = M22_vals[-1]

M_matrix = np.round([[M11_at_t_max, M12_at_t_max], [M21_at_t_max, M22_at_t_max]]

print("Matrix at t = t_max:")
print(M_matrix)
```

```
Matrix at t = t_max:
[[ 3.2230e-01 -1.0000e-03]
 [ 6.8530e-01  1.0023e+00]]
```

In [ ]:
```python
solMat = np.array([[M11_at_t_max, M12_at_t_max],
                   [M21_at_t_max, M22_at_t_max]])

sigma_12 = np.linalg.eigvals(solMat)

sigma_tilde = np.round([1 / t_max * np.log(sigma_12[1]), 1 / t_max * np.log(sigm

print(sigma_tilde)
```

```
[ 0.0002 -0.1977]
```

In [ ]:
```python
X1, X2, mu, nu, omega, r, T = sp.symbols('x1 x2 mu nu omega r T', real=True)

H11 = sp.diff(sp.sqrt(X1**2 + X2**2), X1)
H12 = sp.diff(sp.sqrt(X1**2 + X2**2), X2)
```

```python
H21 = sp.diff(sp.atan2(X2, X1), X1)
H22 = sp.diff(sp.atan2(X2, X1), X2)

H = sp.Matrix([[H11, H12], [H21, H22]])

H_Inv = H.inv()

J = sp.Matrix([[mu - 3 * r**2, 0], [2 * nu * r, 0]])

M_P = sp.exp(J * T)

T_val = 2 * sp.pi / (omega + nu * mu)
X1_val = sp.sqrt(mu)
X2_val = 0
r_val = sp.sqrt(X1_val**2 + X2_val**2)

H_val = H.subs({X1: X1_val, X2: X2_val})
Hinv_val = H_Inv.subs({X1: X1_val, X2: X2_val})
Mp_val = M_P.subs({T: T_val, r: r_val, mu: mu, nu: nu})

M_C = (Hinv_val * Mp_val * H_val).simplify()

M_C
```

Out[ ]:

$$\begin{bmatrix} e^{-\frac{4\pi\mu}{\mu\nu+\omega}} & 0 \\ \nu - \nu e^{-\frac{4\pi\mu}{\mu\nu+\omega}} & 1 \end{bmatrix}$$

In [ ]:

```python
#This is done here to avoid exact values earlier
mu_val = 1/10
nu_val = 1
omega_val = 1

subs_values = {mu: mu_val, nu: nu_val, omega: omega_val}
M_C_val = M_C.subs(subs_values)

sigma_12 = M_C_val.eigenvals().keys()

T_val_sub = T_val.subs(subs_values)
real_sigma12 = [sp.N(sp.log(ev) / T_val_sub, 4) for ev in sigma_12]

# Display the results
real_sigma12
```

Out[ ]: [-0.2000, 0]