~\Documents\Skola dokue\Master Year 1\DynamicalStuff master 1\HW3\3.2\bifurication.py

```python
#%%
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from numpy.linalg import eigvals

a = 4 / 9
b = 5 / 9
epsilon = 1
I_vals = np.linspace(0, 1, 100)

def system_b(p, I):
    x, y = p
    eq1 = (x - (1/3) * x**3 - y + I) / epsilon
    eq2 = x + a - b * y
    return [eq1, eq2]

def jacobian_matrix(x, y):
    return np.array([
        [(1 - x**2) / epsilon, -1 / epsilon],
        [1, -b]
    ])

def system_c(t, state, I):
    x, y = state
    x_dot = (1 / epsilon) * (x - (1 / 3) * x**3 - y + I)
    y_dot = x + a - b * y
    return [x_dot, y_dot]

real_parts = []
imag_parts = []

initial_guess = [0, 0]

for I in I_vals:
    fixed_point = fsolve(system_b, initial_guess, args=(I,))
    x_fp, y_fp = fixed_point

    J = jacobian_matrix(x_fp, y_fp)

    eigenvalues = eigvals(J)

    real_parts.append(np.real(eigenvalues))
    imag_parts.append(np.abs(np.imag(eigenvalues)))

# Convert lists to arrays for easy manipulation
real_parts = np.array(real_parts)
imag_parts = np.array(imag_parts)

plt.figure(figsize=(10, 6))
plt.plot(I_vals, real_parts[:, 0], label='Re[λ]', color='blue')
```

```python
plt.plot(I_vals, imag_parts[:, 0], '--', label='|Im[λ]|', color='green')

I_c = 68 / 405  # Hopf bifurcation point
plt.axvline(x=I_c, color='red', linestyle='--', label='Hopf Bifurcation')

plt.xlabel('I')
plt.ylabel('Eigenvalue components')
plt.title('Real and Imaginary Parts of Eigenvalues vs I')
plt.legend()
plt.grid(True)
plt.show()

# %% c)
from scipy.integrate import solve_ivp

t_span = (0, 100)
t_eval = np.linspace(0, 100, 1000)

initial_conditions = [[-2, -2], [2, 2], [-1, 1], [1, -1]]

I_below = I_c - 0.1
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

for ic in initial_conditions:
    sol_below = solve_ivp(system_c, t_span, ic, args=(I_below,), t_eval=t_eval)
    ax[0].plot(sol_below.y[0], sol_below.y[1], label=f'Start point: {ic}')

ax[0].set_title(f'Phase Portrait for I = {I_below:.3f} (Below I_c)')
ax[0].set_xlabel('x(t)')
ax[0].set_ylabel('y(t)')
ax[0].grid(True)

I_above = I_c+0.1
for ic in initial_conditions:
    sol_above = solve_ivp(system_c, t_span, ic, args=(I_above,), t_eval=t_eval)
    ax[1].plot(sol_above.y[0], sol_above.y[1], label=f'Start point: {ic}')

ax[1].set_title(f'Phase Portrait for I = {I_above:.3f} (Above I_c)')
ax[1].set_xlabel('x(t)')
ax[1].set_ylabel('y(t)')
ax[1].grid(True)

# Display the legend
for a in ax:
    a.legend()

plt.tight_layout()
plt.show()

# %% d)

# Parameters
a = 1
b = 1
```

```
epsilon = 1 / 100
I = 0.1

def system_d(t, state):
    x, y = state
    x_dot = (1 / epsilon) * (x - (1 / 3) * x**3 - y + I)
    y_dot = x + a - b * y
    return [x_dot, y_dot]

def x_nullcline(x, I):
    return x - (1 / 3) * x**3 + I

x_vals = np.linspace(-2.2, 2.2, 50)
y_vals = np.linspace(-1, 2, 50)
X, Y = np.meshgrid(x_vals, y_vals)

U = (1 / epsilon) * (X - (1 / 3) * X**3 - Y + I)
V = X + a - b * Y

plt.figure(figsize=(6, 6))
plt.streamplot(X, Y, U, V, color='gray', density=1)

plt.plot(x_vals, x_nullcline(x_vals, I), 'b--')

# Fixed point (approximate solution)
def fixed_point_equation(state):
    x, y = state
    return [(1 / epsilon) * (x - (1 / 3) * x**3 - y + I), x + a - b * y]

fixed_point = fsolve(fixed_point_equation, [0, 0])
plt.plot(fixed_point[0], fixed_point[1], 'go', label='Fixed Point')

t_span = [0, 20]
t_eval = np.linspace(0, 20, 1000)

# Small perturbation
initial_condition_small = [fixed_point[0], fixed_point[1] - 0.1]
sol_small = solve_ivp(system_d, t_span, initial_condition_small, t_eval=t_eval)

# Large perturbation
initial_condition_large = [fixed_point[0], fixed_point[1] - 0.25]
sol_large = solve_ivp(system_d, t_span, initial_condition_large, t_eval=t_eval)

plt.plot(sol_small.y[0], sol_small.y[1], 'g-', label='Small Perturbation')
plt.plot(sol_large.y[0], sol_large.y[1], 'm-', label='Large Perturbation')

plt.xlim(-2.2, 2.2)
plt.ylim(-1, 2)
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.title('Phase Portrait with x Nullclines and Trajectories for I = 0.1')
plt.legend()
plt.grid(True)
plt.show()
```

```python

# Plot x against time for both trajectories
plt.figure(figsize=(10, 5))
plt.plot(sol_small.t, sol_small.y[0], 'g-', label='Small Perturbation')
plt.plot(sol_large.t, sol_large.y[0], 'm-', label='Large Perturbation')
plt.xlabel('Time')
plt.ylabel('$x(t)$')
plt.title('Time Series of $x(t)$ for Different Perturbations')
plt.legend()
plt.grid(True)
plt.show()

# %%

```