シミュレーション工学レポート
第 1 テーマ：動的モデル

氏名　　後藤　健一郎
学籍番号　2600230179-1
提出日　　2024 年 10 月 26 日

1　課題内容

$$\frac{dR}{dt} = aR + bJ \tag{1}$$
$$\frac{dJ}{dt} = cR + cJ \tag{2}$$

式 (1), (2) の挙動, あるいは平衡点は, 特性方程式における τ, Δ により定められる. 課題 1-1, 1-2 を踏まえ以下を実地せよ.
a) τ − Δ 平面を考える場合, (τ, Δ) に対し, 1 つ平衡点の種類が定まる. τ − Δ 平面を同じ種類の平衡点をもつ領域に分割せよ。(このように領域に分けられたものを相図と呼ぶ)
b) 領域毎、適当なシステムパラメータを定め、解の挙動のサンプル図を作成せよ.

2　理論と方法

微分方程式を解析的に解く手段としてオイラー法を用いて考える. 微分の定義より

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{y(x + \Delta x) - y(x)}{\Delta x} \tag{3}$$

について、$\Delta x \approx 0$ の時

$$y(x + \Delta x) = y(x) + \frac{dy}{dx}\Delta x \tag{4}$$

となる。さらに、$\frac{dy}{dx} = f(x, y)$ とおくと

$$y(x + \Delta) = y(x) + f(x, y)\Delta x \tag{5}$$

となり (5) 式がオイラーの式である. 現在の点 x の情報のみから Δx だけ離れた点の y の値を予測できる. 今回扱う式は連立の微分方程式であるが, y を $\boldsymbol{x}$, x を t と置き換えることで、ベクトルに対して同じ式が成り立つのでこのことを利用する.
変化量が十分に小さいと仮定した変形をしたのでコード中では十分に小さい値として 0.01 を変化量 dt=0.01 としている.

3　ソースコード

課題 a

```
import numpy as np
import matplotlib.pyplot as plt

# 定数
a = 0
b = 0.5
c = 0.5
d = 0

# 初期値
R0 = 0.5
J0 = 1

# シミュレーションの実行時間
T = 20

dt = 1e-2
steps = int(T / dt) + 1

# 微分された関数
def dx(x, y):
    return a*x + b*y

def dy(x, y):
    return c*x + d*y

# ベクトル場を描画
def plot_vector_field(dx: dx, dy:dy):

    # グリッドの作成
    x = np.linspace(-2, 2, 20)
    y = np.linspace(-2, 2, 20)
    X, Y = np.meshgrid(x, y)

    dR = dx(X, Y)
    dJ = dy(X, Y)

    plt.quiver(X, Y, dR, dJ, scale=25)
    plt.xlabel('R')
    plt.ylabel('J')
    plt.title(f"vector field")
```

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

plot_vector_field(dx, dy)
```

課題 b

```
import numpy as np
import matplotlib.pyplot as plt

# 定数
a = 0
b = 0.5
c = 0.5
d = 0

# 初期値
R0 = 0.5
J0 = 1

# シミュレーションの実行時間
T = 20

# 初期値の配列
initial_conditions = np.array([[0.5, 1],
                               [0.5, 0.5],
                               [-0.5, 1.5],
                               [-1.2, -0.3],
                               [0, 0],
                               [-0.5, -0.5],
                               [1, -1.5],
                               [-1, -1]])
    dt = 1e-2
    steps = int(T / dt) + 1

    # 微分された関数
def dx(x, y):
    return a*x + b*y

def dy(x, y):
    return c*x + d*y


# 解析解をオイラー法で求める関数
def euler(dx, dy, R0, J0, dt, steps):

    R_array = np.zeros(steps)
    J_array = np.zeros(steps)
    R_array[0] = R0
    J_array[0] = J0

    for i in range(1, steps):
        dR_dt = dx(R_array[i-1], J_array[i-1])
        dJ_dt = dy(R_array[i-1], J_array[i-1])
        R_array[i] = R_array[i-1] + dR_dt * dt
        J_array[i] = J_array[i-1] + dJ_dt * dt

    return R_array, J_array


# 解軌道を描画
def plot_trajectory(R0, J0):
    t = np.linspace(-2, T, steps)

    R, J = euler(dx, dy, R0, J0, dt, steps)
    plt.plot(t, R, label='R(t)')
    plt.plot(t, J, label='J(t)')
    plt.xlabel('t')
    plt.ylabel("R and J")
    plt.title("trajectory of R and J")
    plt.legend()

    plt.show()

# 解析解のペアをプロット
def plot_trajectory_pair(R0, J0):

    # 初期値による解析解を求める
    R, J = euler(dx, dy, R0, J0, dt, steps)

    plt.plot(R, J)
    plt.plot(R0, J0, marker='o', markersize=10) # 初
期値のみ強調表示
    plt.xlabel('R(t)')
    plt.ylabel('J(t)')
    plt.title("trajectory of (R(t), J(t))")

    plt.show()

# ベクトル場を描画
def plot_vector_field(dx: dx, dy:dy):

    # グリッドの作成
    x = np.linspace(-2, 2, 20)
    y = np.linspace(-2, 2, 20)
    X, Y = np.meshgrid(x, y)

    dR = dx(X, Y)
    dJ = dy(X, Y)
```

```
    plt.quiver(X, Y, dR, dJ, scale=25)
    plt.xlabel('R')
    plt.ylabel('J')
    plt.title(f"vector field")
    plt.grid()
    plt.show()

# 相図の描画をする関数
def plot_phase_field(initial_conditions:
    np.ndarray, dx, dy):

    # まずベクトル場を表示
    # グリッドの作成
    x = np.linspace(-2, 2, 20)
    y = np.linspace(-2, 2, 20)
    X, Y = np.meshgrid(x, y)

    dR = dx(X, Y)
    dJ = dy(X, Y)

    plt.quiver(X, Y, dR, dJ, scale=25)
    plt.title("phase field")
    plt.grid()
    plt.xlim(-2, 2)
    plt.ylim(-2, 2)

    # 複数の初期による解軌道を計算
    # 初期値による解析解を求める
    for i in range(len(initial_conditions)):
        R, J = euler(dx,
                     dy,
                     initial_conditions[i][0],
                     initial_conditions[i][1],
                     dt,
                     steps)

        plt.plot(initial_conditions[i][0],
                 initial_conditions[i][1],
                 marker='o') # 初期値のみ強調
表示
        plt.plot(R, J)

# 相図の描画
plot_phase_field(initial_conditions, dx, dy)
```

4 実行例

まず課題 a の相図の様子を Figure1 に表す. 微分方程式の初期値の値によってその解軌道は Figure1 のベクトル図に従って変化していくことを表している. また直線

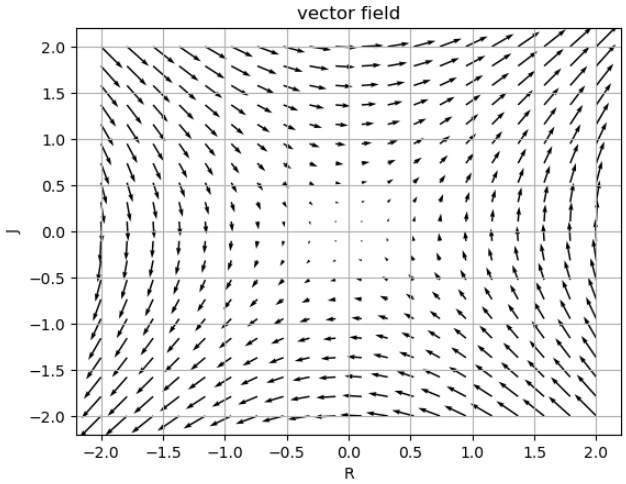

Figure 1: 解軌道を決めるベクトル場の様子. 横軸はロミオからの好意, 縦軸はジュリエットからの好意を表す

$y = x$ と $y = -x$ 上はベクトルがこの直線上にちょうど乗るように存在しているため, 初期値がこの直線上の場合, 解軌道は直線になる. つまりこの直線を漸近線とした双曲線のような形でベクトル場が形成され, 漸近線で分けたとすれば座標平面を 4 つの領域に分割されていることがわかる.

次に課題 b について実際にいくつか初期値を与えることで解軌道をシミュレーションしたものを Figure2 に示す. 初期値が与えられると, その点のベクトル場に従い解析的な解が求まることがわかる. また, 離散的な値をとるオイラー法でも十分にグラフの概形を把握できることもわかる.

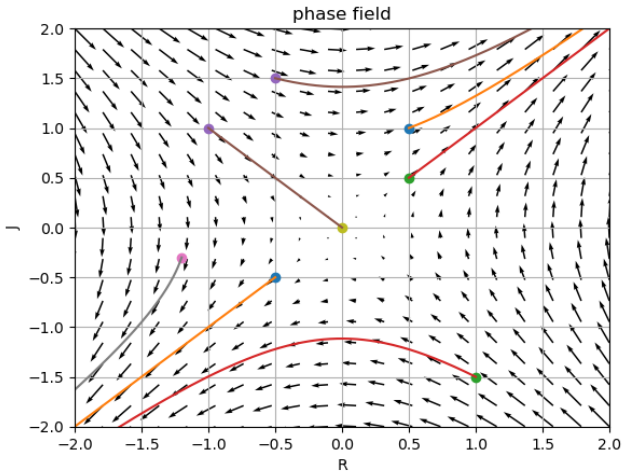

Figure 2: 各初期値による解軌道の様子. 横軸はロミオからの好意, 縦軸はジュリエットからの好意を表す

5　考察

まず初めにオイラー法の過程通りに変化量を十分に小さくとれば、解析的に微分方程式の解が求められることがわかった．また解軌道から今回のような係数の条件設定では原点は微分方程式の固定点であり変数に依らない一定の値であることがわかった．またその性質上一度固定点に到達するとそれ以上解軌道は進まないこともグラフの描画により改めて確認された．さらなる考察として固定点に到達することのない解軌道群は初期値が異なれば交わることがないということが Figure2 から推察される．しかしこの証明には今回のように離散的数値解析のアプローチではなく別途連続的に数学でのアプローチが必須だと思われる．