Problem set 2

FFR110

Amanda Siklund
amasik@student.chalmers.se

Axel Johansson
axejoh@student.chalmers.se

February 25, 2023

Solution propulsion

Problem 1

Task 1a)

The time evolution of n in a given habitat is described by the equation,

$$\frac{\partial n}{\partial t} = rn\left(1-\frac{n}{k}\right) - \frac{An}{1+n/B} + D\frac{\partial^2 n}{\partial x^2} \tag{1}$$

To make the equation dimensionless, we introduce the following variables: dimensionless time $\tau = At$, position $\xi = x\sqrt{\frac{A}{D}}$, and population size $u(\xi,\tau) = \frac{n(x,t)}{B}$. This results in the following equation:

$$A\cancel{B}\frac{\partial}{\partial \tau} = r\cancel{B}u\left(1-\frac{Bu}{k}\right) - \frac{A\cancel{B}u}{1+u} + \cancel{D}\cancel{B}\frac{\partial^2 u}{\frac{\cancel{D}}{A}\partial \xi^2}. \tag{2}$$

Cancel the A,

$$\frac{\partial}{\partial \tau} = \frac{r}{A}u\left(1-\frac{Bu}{k}\right) - \frac{u}{1+u} + \frac{\partial^2 u}{\partial \xi^2}. \tag{3}$$

By inserting the dimensionless parameters $\rho = \frac{r}{A}$ and $q = \frac{K}{B}$, we get the following equation:

$$\frac{\partial}{\partial \tau} = \rho u\left(1-\frac{u}{q}\right) - \frac{u}{1+u} + \frac{\partial^2 u}{\partial \xi^2} \tag{4}$$

The steady state is found by neglecting diffusion and solving $\frac{\partial u}{\partial \tau} = 0$, which yields:

$$u\left(\rho\left(1-\frac{u}{q}\right) - \frac{1}{1+u}\right) = 0 \tag{5}$$

This results in the following solution,
$u_3 = 0$ and

$$u_{1,2} = \frac{q-1}{2} \pm \sqrt{\left(\frac{q-1}{2}\right)^2 + q - \frac{\rho}{q}}. \tag{6}$$

Task 1b)

The model invasion of the prey population from one direction was modeled using three different initial values of ξ_0 and u_0. The first result is shown in Figure 2, using $\xi_0 = 20$ and $u_0 = u_1^*$, where u_1^* is the largest homogeneous steady state. The traveling wave during different time steps is shown in Figure 1. By studying the traveling wave over time we can see that the population expands over the habitat. The velocity of the wave was numerically calculated to $C = 0.1795$. Since the velocity is positive, the traveling wave is moving away from the fixed point u=0 towards the fixed point $u_0 = u_1^*$. This means that $u = 0$ is unstable and $u_0 = u_1^*$ is stable.

Figure 4 presents the case when $\xi_0 = 50$ and $u_0 = u_2^*$. In the opposite from the previous case, the population shrinks over the habitat, as seen in Figure 3. This means the traveling wave moves in the other direction and the velocity was numerically calculated to $C = -0.7092$. A negative velocity results in the traveling wave moving towards the fixed point $u = 0$ and away from $u_0 = u_2^*$, which results in $u = 0$ being stable and $u_0 = u_2^*$ unstable.

The last case is presented in Figure 6 using $\xi_0 = 20$ and $u_0 = 1.1 \cdot u_2^*$ as initial values. By studying the traveling wave over time, it is clear that it expands over the habitat, as shown in Figure 5. The velocity was calculated to the same value as in the first case, namely $C = 0.1795$. Similar to that case it means that $u = 0$ is unstable and $u_0 = u_1^*$ is stable.

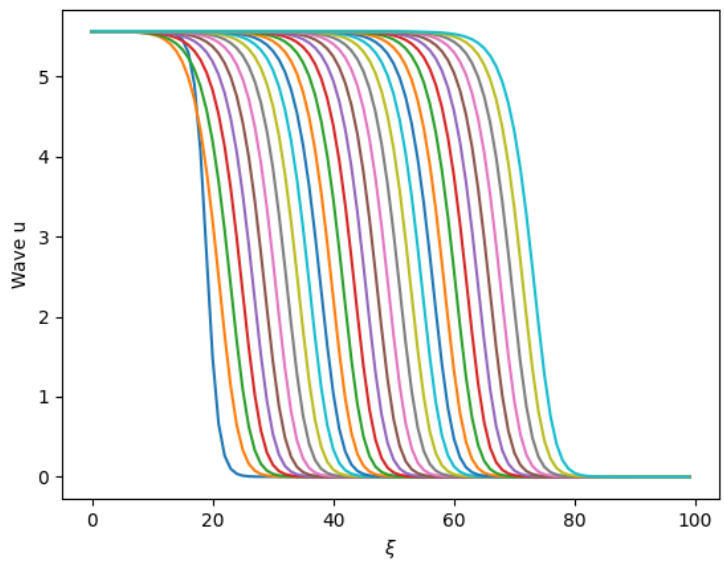

Figure 1: Traveling wave simulated oveer time with $\xi_0 = 20$ and $u_0 = u_1^*$.

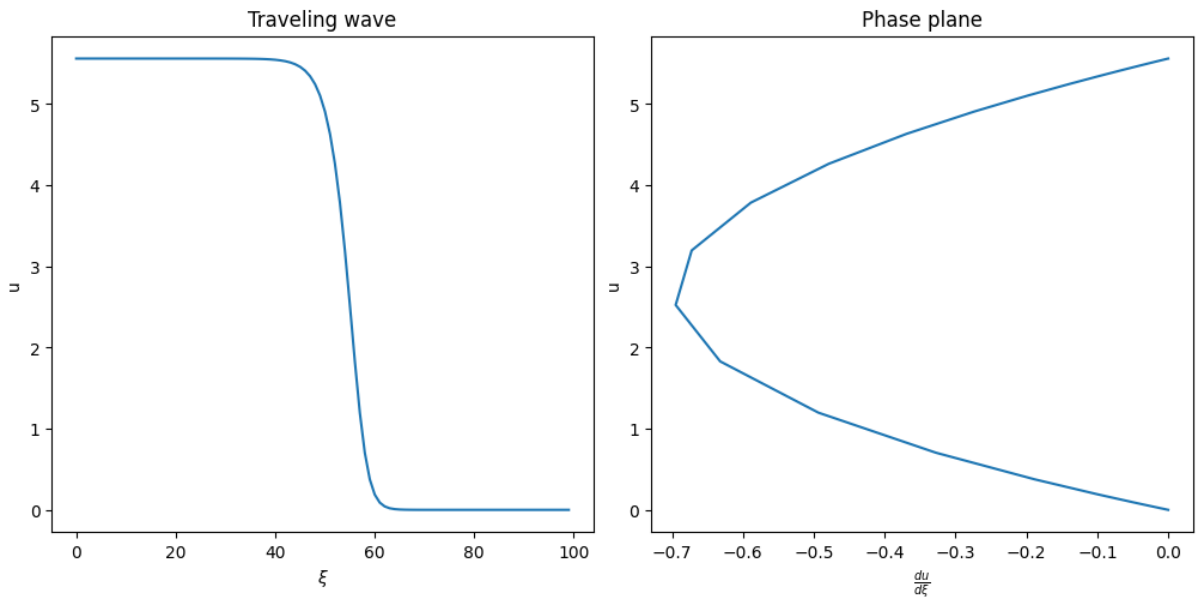

Figure 2: A representation of the traveling wave and the corresponding trajectory in the phase plane at a fixed time step $\tau = 200$ using $\xi_0 = 20$ and $u_0 = u_1^*$.

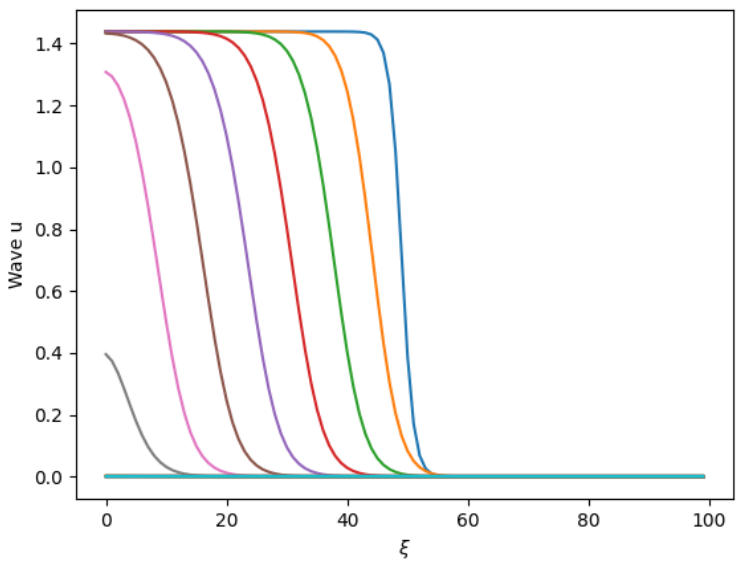

Figure 3: Traveling wave simulated over time with $\xi_0 = 50$ and $u_0 = u_2^*$.

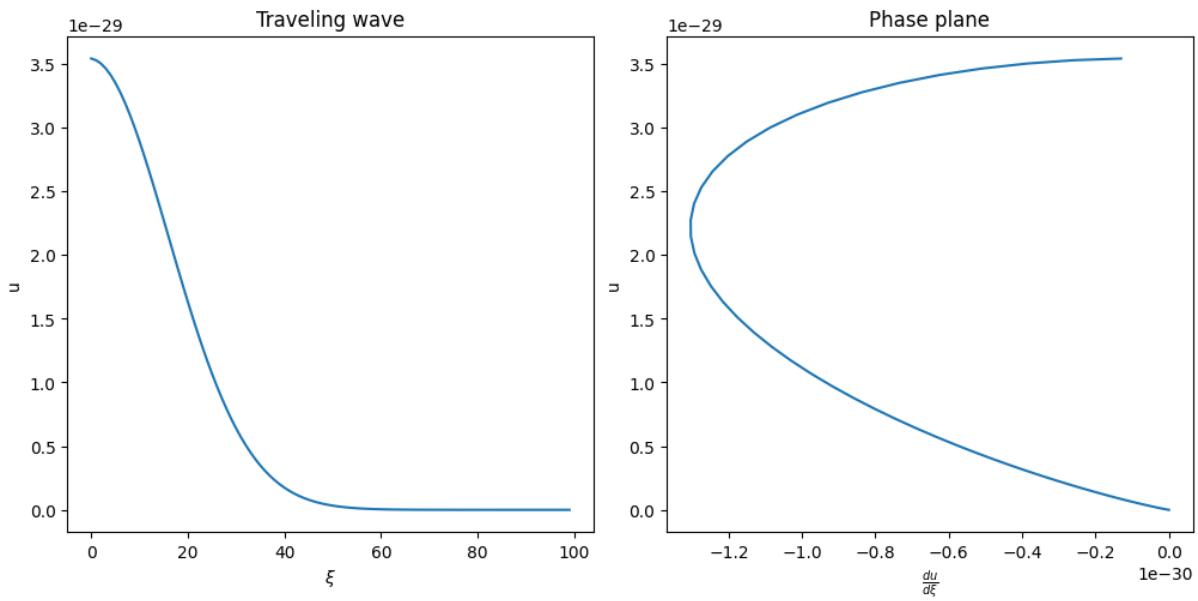

Figure 4: A representation of the traveling wave and the corresponding trajectory in the phase plane at a fixed time step $\tau = 200$ using $\xi_0 = 50$ and $u_0 = u_2^*$.

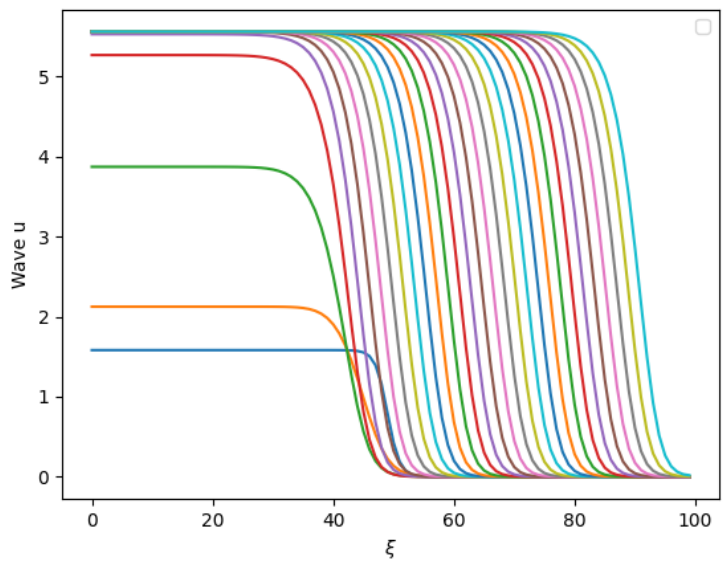

Figure 5: Traveling wave simulated over time with $\xi_0 = 50$ and $u_0 = 1.1u_2^*$.

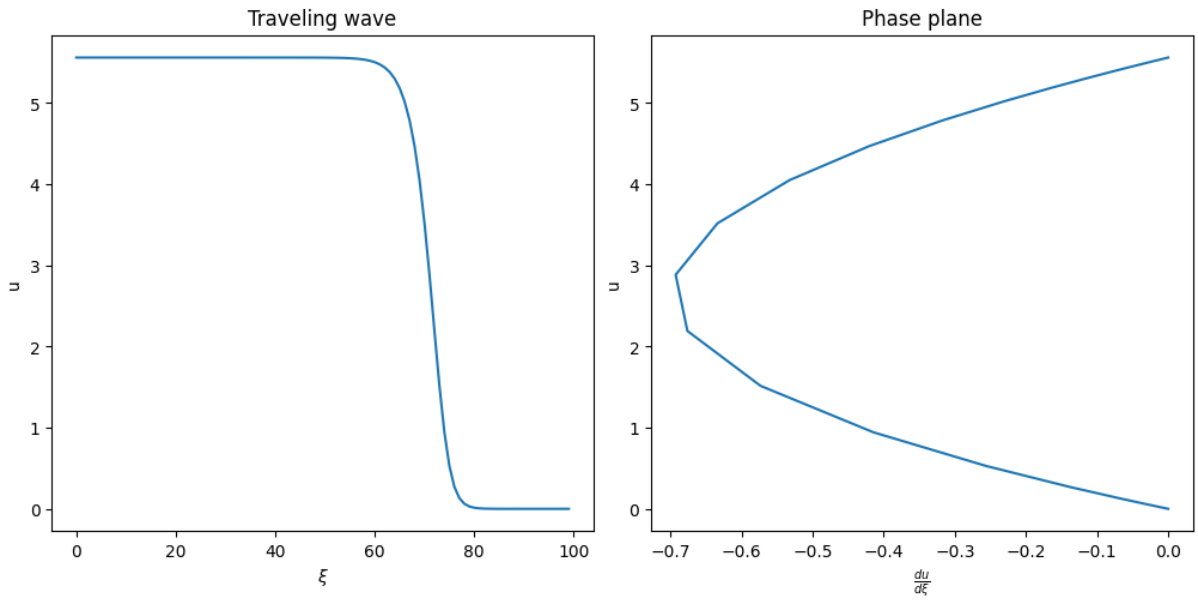

Figure 6: A representation of the traveling wave and the corresponding trajectory in the phase plane at a fixed time step $\tau = 200$ using $\xi_0 = 50$ and $u_0 = 1.1u_2^*$.

Task 1c)

A local outbreak is simulated using the population size,

$$u(\xi, t=0) = u_0 e^{-(\xi-\xi_0)^2}. \tag{7}$$

For the first initial condition, using $\xi_0 = 50$ and $u_0 = 3u_1^*$, the population dies out as shown in Figure 7. However, the peak is shifted downwards and expands a bit before it dies out. Figure 8 shows the other case, using $\xi_0 = 50$ and $u_0 = 3u_1^*$. As in the previous case the peak is shifted downwards but will instead end up in a traveling wave expanding over the habitat.

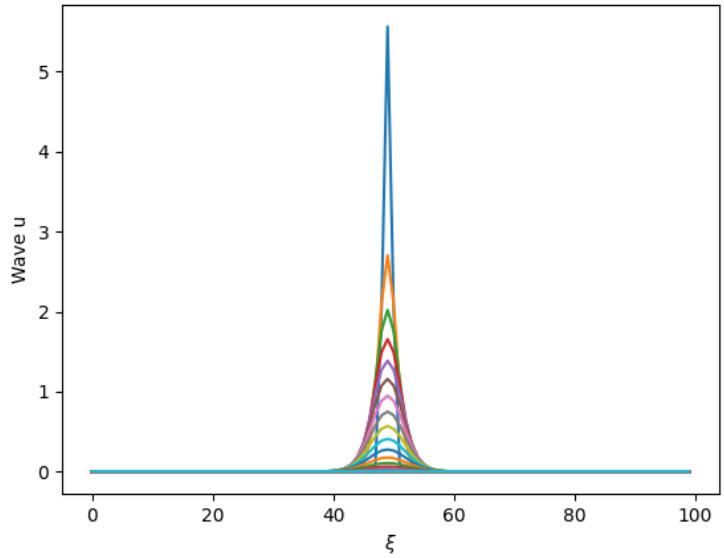

Figure 7: Wave with initial conditions $\xi_0 = 50$ and $u_0 = u_1^*$.

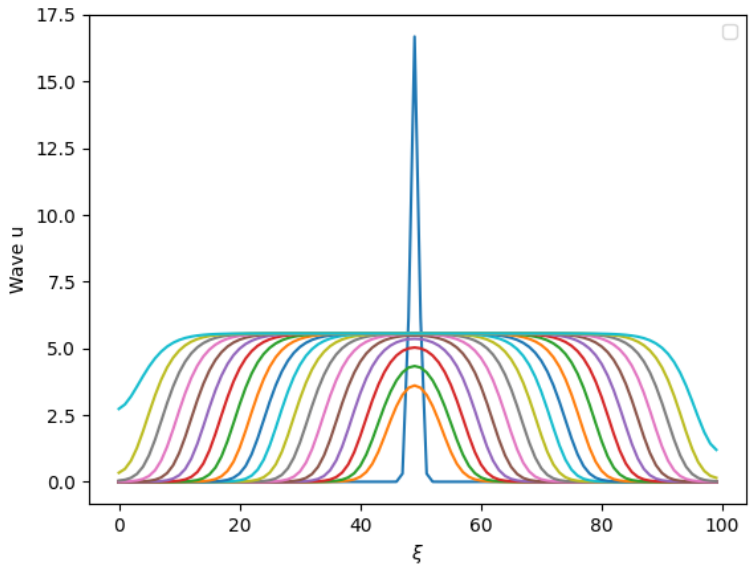

Figure 8: Wave with initial conditions $\xi_0 = 50$ and $u_0 = 3u_1^*$.

Python script

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

def ramp(u0, xi, xi_0, L, time_steps, dt):
    u = np.zeros((3, L, int(time_steps / dt)))
    u[:, :, 0] = u0 / (1 + np.exp(xi - xi_0[:, None]))
    return u


def ramp_c(uc0, xi, xic_0, time_steps, dt, L):
    uc = np.zeros((2, L, int(time_steps / dt)))
    uc[0, :, 0] = uc0[0] * np.exp(-(xi - xic_0[0]) ** 2)
    uc[1, :, 0] = uc0[1] * np.exp(-(xi - xic_0[1]) ** 2)
    return uc


def simulation(q, u, dt, rho, time_steps):
    for t in range(int(time_steps / dt - 1)):
        u[:, 0, t + 1] = u[:, 0, t] + dt * (
                rho * u[:, 0, t] * (1 - u[:, 0, t] / q) - u[:, 0, t] / (1 + u[:, 0, t]) + (u[

        u[:, -1, t + 1] = u[:, -1, t] + dt * (
                rho * u[:, -1, t] * (1 - u[:, -1, t] / q) - u[:, -1, t] / (1 + u[:, -1, t]) -
                u[:, -1, t] - u[:, -2, t]))

        u[:, 1:-1, t + 1] = u[:, 1:-1, t] + dt * (
                rho * u[:, 1:-1, t] * (1 - u[:, 1:-1, t] / q) - u[:, 1:-1, t] / (1 + u[:, 1:-
                (u[:, :-2, t] + u[:, 2:, t] - 2 * u[:, 1:-1, t]))
    return u


def draw_b(u, time_steps, dt):
    for j in range(len(u)):
        plt.figure()
        plt.xlabel(r"$\xi$")
        plt.ylabel("Wave u")
        for i in np.linspace(0, time_steps / dt - 5, 30):
            plt.plot(u[j, :, int(i)])
    plt.legend()
    plt.show()


def draw_c(u, time_steps, dt):
    for j in range(len(u)):
        plt.figure()
        plt.xlabel(r"$\xi$")
        plt.ylabel("Wave u")
        for i in np.linspace(0, time_steps / dt * 0.2, 20):
            plt.plot(u[j, :, int(i)])
    plt.legend()
```

```
    plt.show()


def draw_traveling_wave(u):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5), constrained_layout=True)
    ax1.plot(u[0, :, 30000])
    ax1.set_xlabel(r"$\xi$")
    ax1.set_ylabel("u")
    ax1.set_title('Travelling wave')

    v = np.diff(u[0, :, 30000])
    ax2.plot(v, u[0, :-1, 20000])
    ax2.set_xlabel(r"$\frac{du}{d\xi}$")
    ax2.set_ylabel("u")
    ax2.set_title('Phase plane')
    plt.show()


def wave_speed(u, dt):
    print(10 / ((np.argmax(u[0, 60, :] > 2) - np.argmax(u[0, 50, :] > 2)) * dt))
    print(10 / ((np.argmax(u[1, 30, :] < 0.5) - np.argmax(u[1, 20, :] < 0.5)) * dt))
    print(10 / ((np.argmax(u[2, 70, :] > 2) - np.argmax(u[2, 60, :] > 2)) * dt))


def main():
    rho = .5
    L = 100
    q = 8
    xi = np.arange(1, L + 1)
    xi_0 = np.array([20, 50, 50])
    u0 = np.array([[(q - 1) / 2 + np.sqrt(((q - 1) / 2) ** 2 - q * (1 - rho) / rho)],
                   [(q - 1) / 2 - np.sqrt(((q - 1) / 2) ** 2 - q * (1 - rho) / rho)],
                   [1.1 * ((q - 1) / 2 - np.sqrt(((q - 1) / 2) ** 2 - q * (1 - rho) / rho))]]
    time_steps = 300
    dt = 1 / 100
    u = ramp(u0, xi, xi_0, L, time_steps, dt)
    u = simulation(q, u, dt, rho, time_steps)
    draw_b(u, time_steps, dt)

    uc0 = np.array([u0[0], u0[0] * 3])
    xic_0 = np.array([50, 50])
    uc = ramp_c(uc0, xi, xic_0, time_steps, dt, L)
    uc = simulation(q, uc, dt, rho, time_steps)

    draw_c(uc, time_steps, dt)
    # draw_traveling_wave()
    wave_speed(u, dt)


if __name__ == '__main__':
    main()
```

Problem 2

Task 2a)

Neglecting diffusion and determining the spatially homogeneous steady state by setting the equation to zero as follows,

$$\frac{\partial u}{\partial t} = a - (b+1)u + u^2 v = 0 \tag{8}$$

$$\frac{\partial v}{\partial t} = bu - u^2 v = 0 \tag{9}$$

From (9) we get $v = b/u$, inserted in (8) gives $u = a$ and $v = b/a$.

Task 2b)

The jacobian of the model is with u and v as above ant then with $b = 8$ and $a = 3$,

$$\mathbb{J} = \begin{bmatrix} b-1 & a^2 \\ -b & -a^2 \end{bmatrix} = \begin{bmatrix} 7 & 9 \\ -8 & -9 \end{bmatrix}. \tag{10}$$

To find the diffusion-driven instability we expand around the steady state with a perturbation $n^* + \delta n$ as,

$$\frac{\partial}{\partial t}(n^* + \delta n_0) = f(n^* + \delta n_0) + \nabla^2 \mathbb{D}(n^* + \delta n_0) \implies \tag{11}$$

$$\frac{\partial}{\partial t}\delta n = \mathbb{J}(n^*)\delta n + \nabla^2 \mathbb{D}\delta n \tag{12}$$

Separation of variables $\delta n = TR\delta n_0$ gives

$$\frac{1}{T}\frac{\partial}{\partial t}T\delta n_0 = \mathbb{J}\delta n_0 + \frac{\nabla^2 R}{R}\mathbb{D}\delta n_0. \tag{13}$$

Left side is dependent on time and rightside is not. Hence they are bot equal a constant. Call

$$\frac{1}{T}\frac{\partial}{\partial t}T = \lambda \tag{14}$$

and

$$\frac{\nabla^2 R}{R} = -k^2. \tag{15}$$

The expression becomes

$$\lambda \delta n_0 = (\mathbb{J} - k^2 \mathbb{D})\delta n_0 = \mathbb{K}\delta n_0. \tag{16}$$

$$\mathbb{K} = \mathbb{J} - k^2 \mathbb{D} = \begin{bmatrix} 7 & 9 \\ -8 & -9 \end{bmatrix} - k^2 \begin{bmatrix} 1 & 0 \\ 0 & d \end{bmatrix}. \tag{17}$$

To find the bifurcation we let the determinant of K equal zero, $|K| = 0 \implies 0 = dk^4 - 7dk^2 + 9k^2 + 9$. Solve for k^2

$$k^2 = \frac{9-7d}{2d} \pm \sqrt{\left(\frac{9-7d}{2d}\right)^2 - \frac{9}{d}}. \tag{18}$$

The discriminant has to be 0 or larger for a real solution. Hence

$$\left(\frac{9-7d}{2d}\right)^2 - \frac{9}{d} = 0 \quad \implies \quad d = D_v \quad \implies \quad D_v \geq 2{,}692. \tag{19}$$

Task 2c)

The system dynamics were simulated using Python. The initial values of u and v were randomly distributed with small perturbations of order 10% around the steady state. Periodic boundaries were implemented using the numpy function roll, which allows elements to roll beyond the last position and be set at the first. By rolling it in all directions, subtracting the original position, and summing it up, we obtain the total difference to the cells nearby. To observe the evolution more closely, the system was updated with smaller time steps of $dt = 1/100$.

Figure 9-12 represent four different values of D_v that were tested and compared, namely $D_v = [2.3, 3, 5, 9]$. All cells are affected by the neighbour cells, and D_v can be seen as a scale of that impact. Larger values of D_v results in more grouping of the chemicals which can be seen in pattern with more contrast, meaning cells nearby have similar values and there is a larger interval in between them. This can be seen in Figure 12, where the lines are much sharper, and the concentration is higher than in Figure 9. Lower values of D_v lead to more blurred lines in the pattern and there is a smaller interval between the values in general. By comparing Figure 9 and 10 we can make the conclusion that the system exhibit a diffusion-driven instability in between 2.3 and 3 as presented in the previous task.

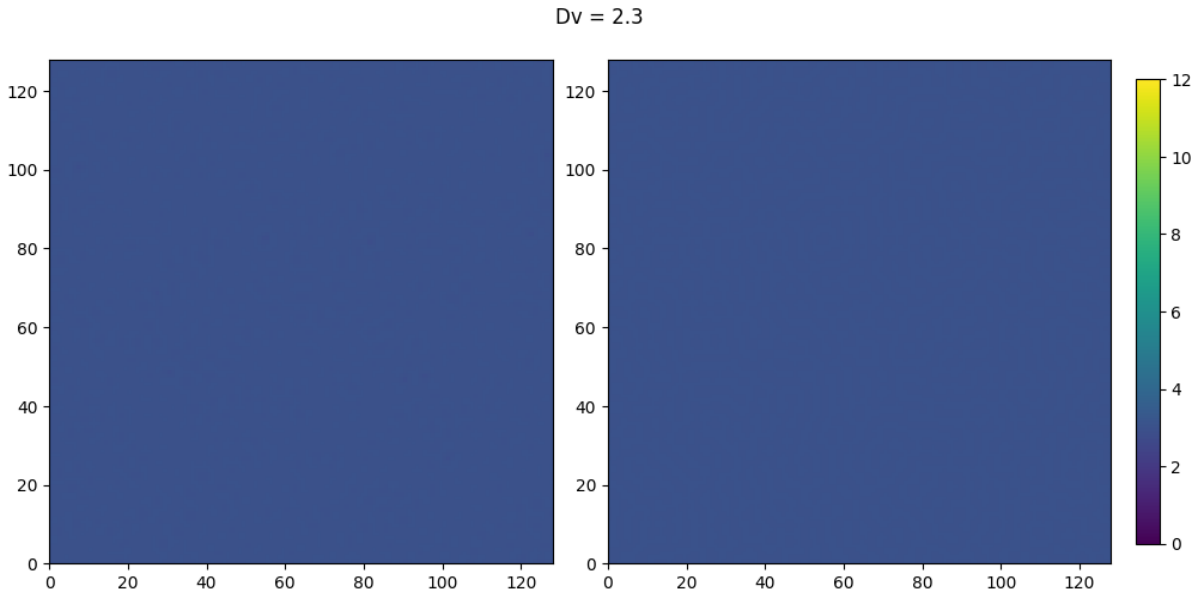

Figure 9: A representation of the concentration u using $D_v = 2.3$ after 500 and 50000 iterations, respectively.

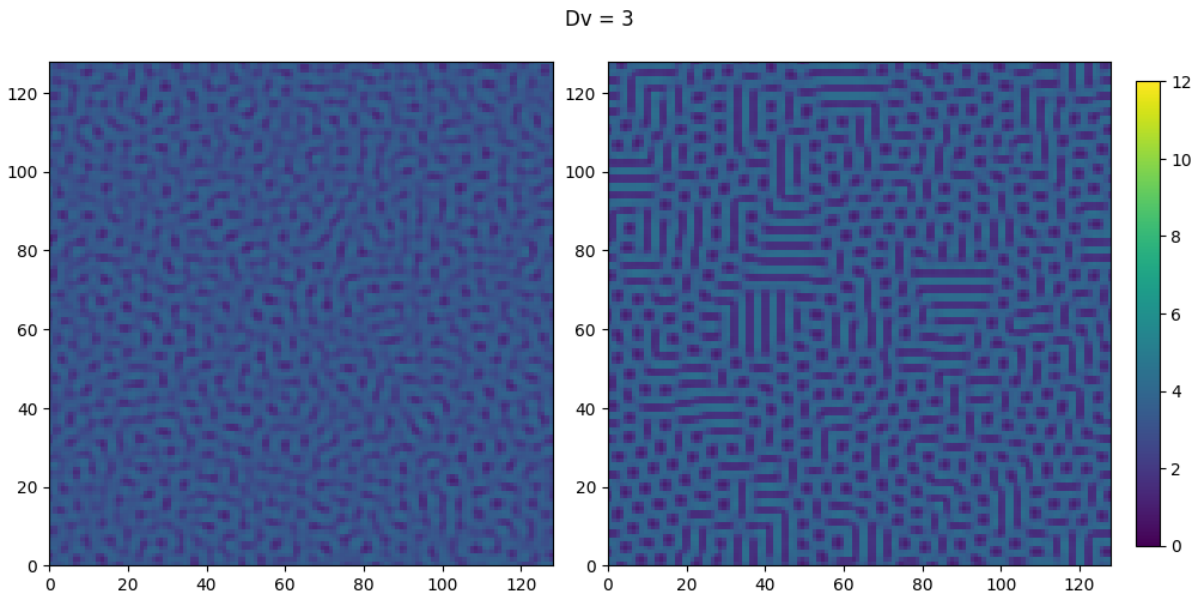

Figure 10: A representation of the concentration u using $D_v = 3$ after 500 and 50000 iterations, respectively.

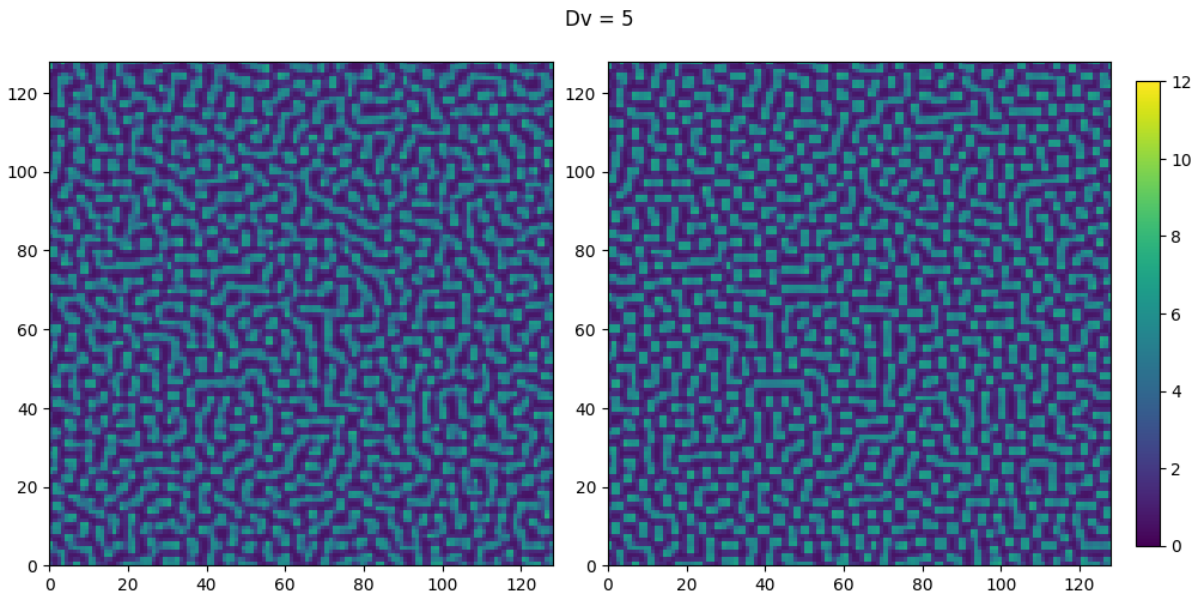

Figure 11: A representation of the concentration u using $D_v = 5$ after 500 and 50000 iterations, respectively.

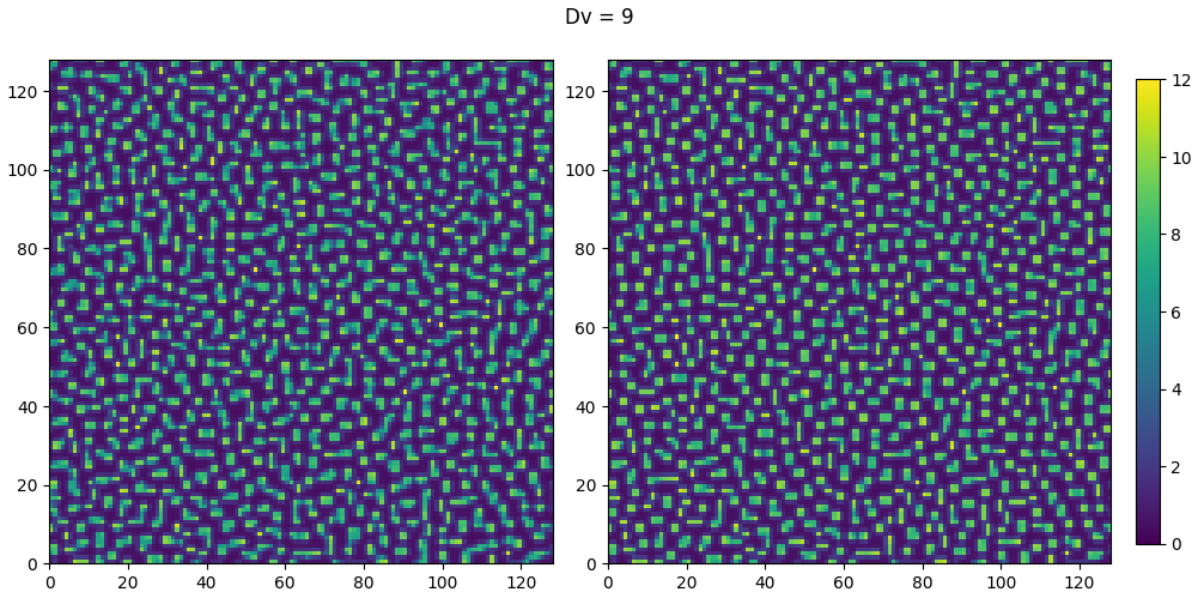

Figure 12: A representation of the concentration u using $D_v = 9$ after 500 and 50000 iterations, respectively.

Python script

```
import numpy as np
import matplotlib.pyplot as plt
# Problem set 2.2
def laplace(u,h):
    return (np.roll(u,-h,axis=0)+np.roll(u,h,axis=0)+
        np.roll(u,-h,axis=1)+np.roll(u,h,axis=1)-4*u)/h**2

L=128
Dv = [2.3, 3, 5, 9]
a=3
b=8
h=1
timesteps=500
dt=1/100
uSteadyState=a
vSteadyState=b/a
u = np.random.uniform(0.9*uSteadyState,1.1*uSteadyState,(L,L))
v = np.random.uniform(0.9*vSteadyState,1.1*vSteadyState,(L,L))
uList=np.zeros((4,L,L))

value=D[0]
for t in range(int(timesteps/dt)):
    tmpU = u + (a - (b+1)*u + u**2*v + laplace(u,h))*dt
    tmpV = v + (b*u - u**2*v + value*laplace(v,h))*dt
    u = np.copy(tmpU)
    v = np.copy(tmpV)
    if t == 500:
        u1000=np.copy(u)

fig, (ax1, ax2)  = plt.subplots(1, 2, figsize=(10, 5),constrained_layout=True)
im1=ax1.imshow(u1000, vmin=0, vmax=12, extent=[0, 128, 0, 128])
im2=ax2.imshow(u, vmin=0, vmax=12, extent=[0, 128, 0, 128])
fig.suptitle(f"Dv = {value}")
fig.colorbar(im2, ax=ax2, shrink=0.85)
plt.show()
```

Problem 3

Task 3a)

$$1 = K \int_{-\pi/2}^{\pi/2} \cos^2\theta g(Kr\sin\theta)d\theta \tag{20}$$

We know that

$$g(\omega) = \frac{\gamma}{\pi}\frac{1}{\omega^2+\gamma^2} \Longrightarrow g(0) = \frac{1}{\pi\gamma} \tag{21}$$

and hence the second derivative becomes

$$g''(0) = \frac{-2}{\gamma^3\pi}. \tag{22}$$

We can compute K_c with the following equation,

$$K_c = \frac{1}{\int_{-\pi/2}^{\pi/2}\cos^2\theta d\theta g(0)} = \frac{2}{\pi g(0)} = 2\gamma. \tag{23}$$

Now solving the integral (20) expanded for small arguments of g,

$$1 = K\int_{-\pi/2}^{\pi/2} d\theta \cos^2[g(0) + \cancel{g'(0)} + \frac{g''(0)}{2!}(K^2r^2\sin^2\theta) + ...] \tag{24}$$

Knowing that

$$\int_{-\pi/2}^{\pi/2} d\theta \cos^2 = \frac{\pi}{2} \quad \text{and} \quad \int_{-\pi/2}^{\pi/2} d\theta \cos^2\sin^2 = \frac{\pi}{8}, \tag{25}$$

and what is mentioned above about g and K_c the expanded integral (24) becomes,

$$1 = K\left[\frac{1}{\pi\gamma}\frac{\pi}{2} + \frac{1}{\pi\gamma^3}\frac{\pi}{2^3}\frac{K^3r^2}{2}\right] = \frac{K}{K_c} - \frac{K^3}{K_c^3}r^2 \tag{26}$$

Moving around,

$$r^2 = \frac{K_c^2}{K^2} - \frac{K_c^3}{K^3} \tag{27}$$

The derivative of $r^2(K)$ will help us in the taylor expansion, it is as follows

$$\frac{\mathrm{d}}{\mathrm{d}K}r^2(K) = 3K_c^3K^{-4} - 2K_c^2K^{-3}\Big|_{K=K_c} = \frac{1}{K_c}. \tag{28}$$

The taylor expansion around $K = K_c$ is

$$r^2(K_c) = 0 + (K-K_c)r'(K_c) = \frac{K-K_c}{K_c} \tag{29}$$

Knowing that $\mu = \frac{K-K_c}{K_c}$ and $r = C\sqrt{\mu}$ results in

$$r^2 = \mu \Longrightarrow r = \sqrt{\mu} \Longrightarrow C = 1. \tag{30}$$

Task 3b)

Figure 13-15 presents the order parameter r over time for different values of K and N. As seen in Figure 13 there is no convergence for smaller values of K. By increasing K to just above $K_c = 0.2$ the model starts to converge as seen in Figure 14. This is expected from task 3a) since we calculated that $K_c = 2\gamma$ and that $\gamma = 0.1$. It can also be seen that the model works better with larger values of N. However, in this simulation, it works surprisingly well for smaller N also and it is not so clear in the figures since they converge fast for all N. However, different timescales could show this behavior more clearly. From this, we can conclude that the mean-field seems to work and it works better when N is larger. Worth noting is that the starting position is random making each run a little different.

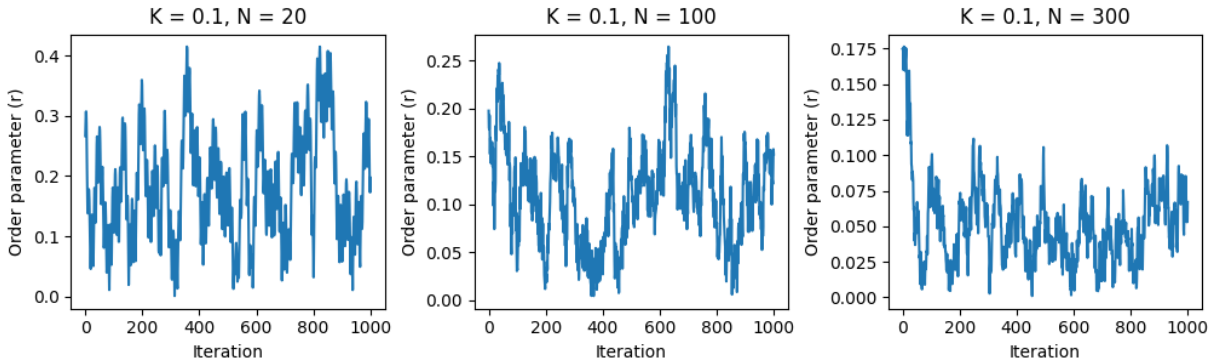

Figure 13: The results of using different N values and K=0.1

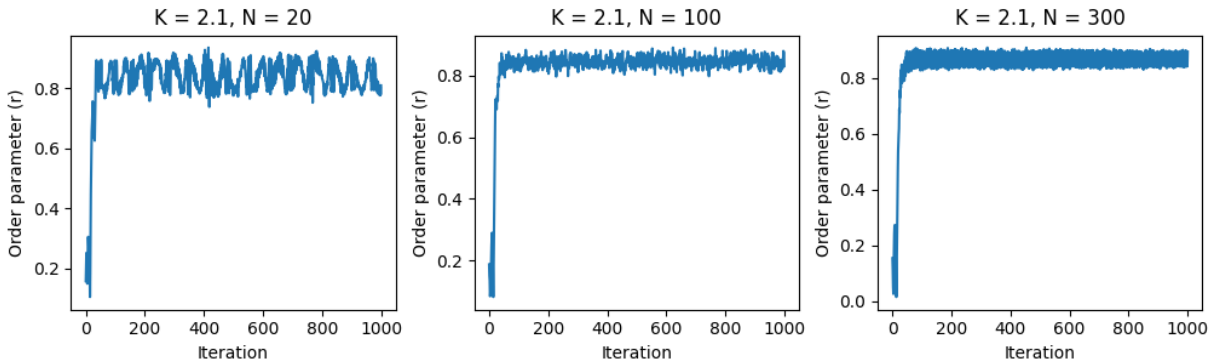

Figure 14: The results of using different different N and K=0.21

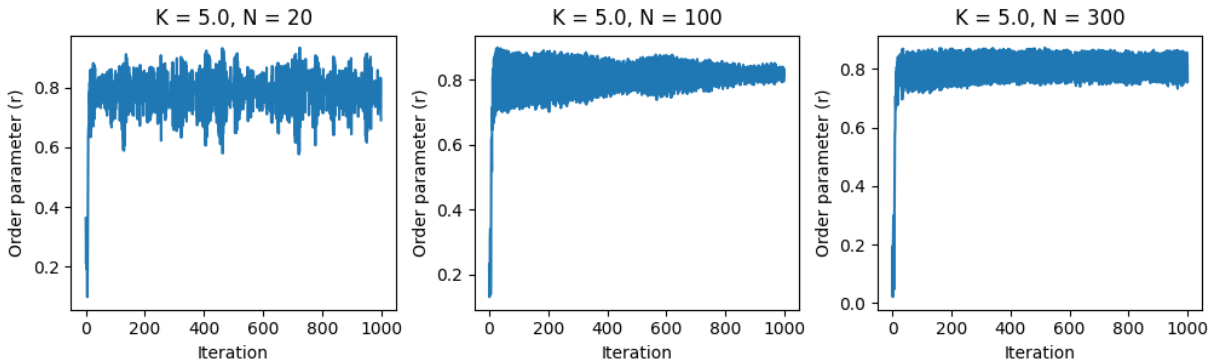

Figure 15: The results of using different N and K=5.

Python script

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import cauchy


def update_theta(theta, k, N, omega, dt):
    diff = np.angle(np.exp(1j*theta[:, np.newaxis])-np.exp(1j*theta[np.newaxis, :]))
    theta += dt * (omega + k/N * (np.sin(diff)).sum(axis=0))
    return theta


def calculate_r(theta, N):
    return np.absolute(np.sum(np.exp(1j * theta)))/N


def main():
    gamma = .1
    N = np.array([20, 100, 300])
    N = 20
    K = np.array([0.1, 0.23, 1])
    dt = 1/10
    time_max = 100
    r = np.zeros((3, int(time_max / dt)))

    fig, axes = plt.subplots(1, 3, constrained_layout=True, figsize=(10, 3))
    for i, k in enumerate(K):
        omega = cauchy.rvs(loc=0, scale=gamma, size=N)
        theta = np.random.uniform(low=-np.pi / 2, high=np.pi / 2, size=N)
        for t in range(int(time_max / dt)):
            theta = update_theta(theta, k, N, omega, dt)
            r[i, t] = calculate_r(theta, N)

        ax = axes[i]
        ax.set_title(f"K={K[i]}, N={N}")
        ax.plot(np.array(range(len(r[i,:])))*dt, r[i, :])
        ax.set_xlabel("Time (t)")
        ax.set_ylabel("Order parameter (r)")
    plt.show()


if __name__ == '__main__':
    main()
```