CompBio HW2

Axel Eiman - 980626-8414
William Truvé - 980312-4271

February 2022

Problem 2, Diffusion driven instability

Task a)

Neglecting diffusion we find the steady states where $\frac{\partial u}{\partial t} = \frac{\partial v}{\partial t} = 0$:

$$\frac{\partial v}{\partial t} = bu - u^2 v = 0 \tag{1}$$

$$bu = u^2 v \tag{2}$$

One possible solution here is $u^* = 0$. Inserting this into $\frac{\partial u}{\partial t}$ we see that this requires $a = 0$ which is not possible since a is a positive constant. Instead we continue:

$$b = uv \Rightarrow v = \frac{b}{u} \tag{3}$$

Inserting into $\frac{\partial u}{\partial t} = 0$ yields:

$$\frac{\partial u}{\partial t} = a - (b+1)u + u^2 \frac{b}{u} = a - bu - u + bu = 0 \tag{4}$$

$$u^* = a \Rightarrow v^* = \frac{b}{a} \tag{5}$$

Task b)

We have the same kind of model described in the notes for lecture 7. The bifurcation where stability changes occurs when we set the discriminant to zero from eq. (7) according to the lecture notes.

$$(D_c J_{11} + J_{22})^2 - 4 D_c \det \mathbb{J} = 0 \tag{6}$$

Where $\mathbb{J}$ is the jacobian of the system and J_{ij} are the elements in position i, j. Solving this equation for the parameter D_c gives us the value where the system starts exhibiting diffusion-driven instability, which is at $D_c = 2.6921$.

Task c)

When simulating, the values for u and v are initialized near the steady state for each space on the lattice. To iterate the dynamics, the discrete laplace operator was implemented using python and the numpy roll function. This enables fast computation of adding the neighboring values in all four directions and subtracting the own value multiplied by four. As well as being easy to follow the formula for the discrete laplace operator it also gives periodic boundary conditions by using the roll function. The integrations were run with a time step of 0.01, and the resulting heatmaps of u are shown for different values of D below.

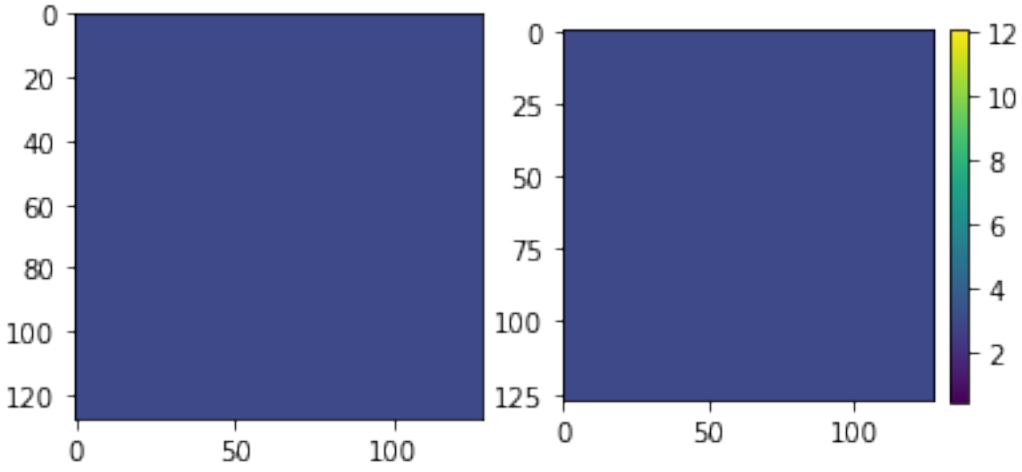

Figure 1: $D = 2.3$

With $D = 2.3$ there are only extremely small variations, so compared with the other simulations, u is basically spatially homogeneous.

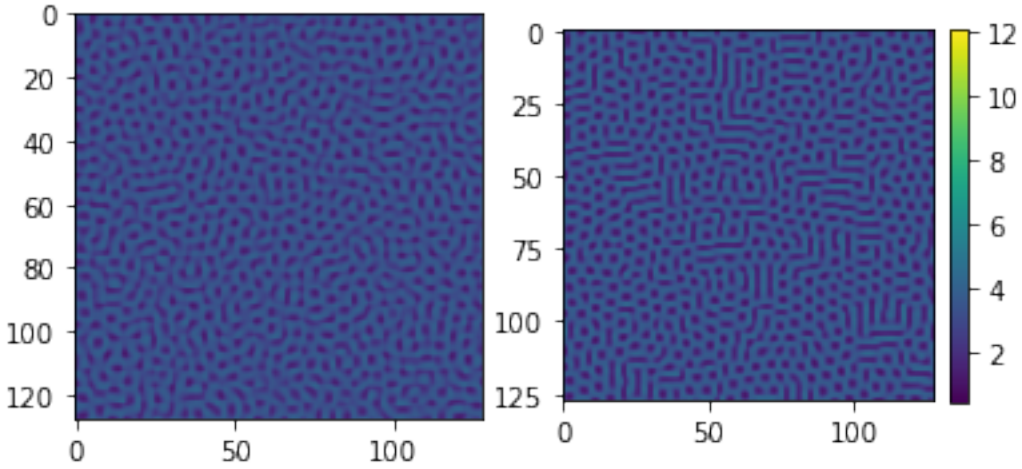

Figure 2: $D = 3$

With $D = 3$, which is larger than the threshold we found for diffusion-driven instability we find patterns in u as evident in figure 1. The patterns are kind of spotted, with some spots being drawn out in longer formations.

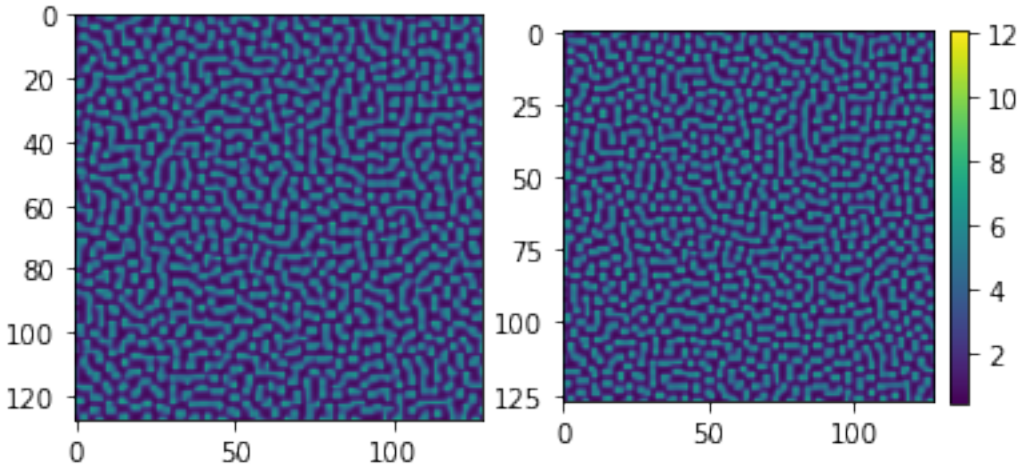

Figure 3: $D = 5$

When D increases we see more pronounced spots in figure 3 & 4. The "drawn out" spots or connected spots depending on how you want to describe the pattern start disappearing with as D is increased.

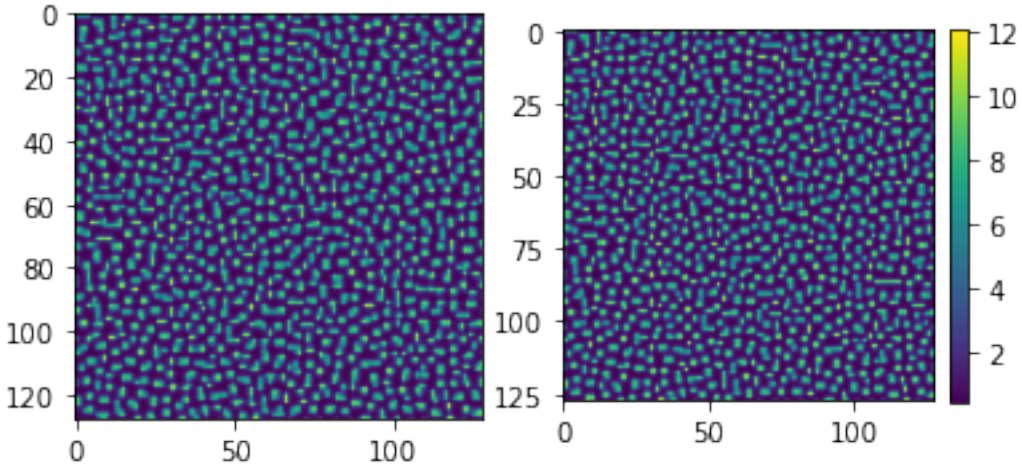

Figure 4: $D = 9$

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

a = 3
b = 8
L = 128
dt = 0.01
D_u = 1
D_v = [2.3, 3, 5, 9]
steady_u = a
steady_v = b/a
```

```
curr_max = -np.inf
curr_min = np.inf

def laplacian(grid):
    return np.roll(grid, 1, axis=0) + np.roll(grid, -1, axis=0) + np.roll(grid, 1, axis=1)

def dudt(u,v,dt):
    return (a*np.ones(u.shape) - (b+1)*u + np.multiply(v,np.square(u)) + D_u*laplacian(u))*

def dvdt(u,v,dt,D_v):
    return (b*u - np.multiply(v,np.square(u)) + D_v*laplacian(v))*dt


u = np.random.uniform(low=0.9 * steady_u, high=1.1 * steady_u, size=(L, L))
v = np.random.uniform(low=0.9 * steady_v, high=1.1 * steady_v, size=(L, L))

D_v = 2.3
for t in range(10000):
    u += dudt(u, v, dt)
    v += dvdt(u, v, dt, D_v)
    if t == 1000:
        u_1000_1 = u.copy()
        if np.max(u) > curr_max:
            curr_max = np.max(u)
        if np.min(u) < curr_min:
            curr_min = np.min(u)

if np.max(u) > curr_max:
    curr_max = np.max(u)
if np.min(u) < curr_min:
    curr_min = np.min(u)

u_end_1 = u.copy()

fig, (ax1, ax2) = plt.subplots(1, 2)

im1 = ax1.imshow(u_1000_1)
im2 = ax2.imshow(u_end_1)
im1.set_clim(curr_min,curr_max)
im2.set_clim(curr_min,curr_max)
fig.colorbar(im1, fraction=0.046, pad=0.04)

plt.show()

u = np.random.uniform(low=0.9 * steady_u, high=1.1 * steady_u, size=(L, L))
v = np.random.uniform(low=0.9 * steady_v, high=1.1 * steady_v, size=(L, L))
```

```
D_v = 5
for t in range(10000):
    u += dudt(u, v, dt)
    v += dvdt(u, v, dt, D_v)
    if t == 1000:
        u_1000_3 = u.copy()
        if np.max(u) > curr_max:
            curr_max = np.max(u)
        if np.min(u) < curr_min:
            curr_min = np.min(u)

if np.max(u) > curr_max:
    curr_max = np.max(u)
if np.min(u) < curr_min:
    curr_min = np.min(u)
u_end_3 = u.copy()

fig, (ax1, ax2) = plt.subplots(1, 2)

im1 = ax1.imshow(u_1000_3)
im2 = ax2.imshow(u_end_3)
im1.set_clim(curr_min,curr_max)
im2.set_clim(curr_min,curr_max)
fig.colorbar(im1, fraction=0.046, pad=0.04)

plt.show()

u = np.random.uniform(low=0.9 * steady_u, high=1.1 * steady_u, size=(L, L))
v = np.random.uniform(low=0.9 * steady_v, high=1.1 * steady_v, size=(L, L))

D_v = 9
for t in range(10000):
    u += dudt(u, v, dt)
    v += dvdt(u, v, dt, D_v)
    if t == 1000:
        u_1000_4 = u.copy()
        if np.max(u) > curr_max:
            curr_max = np.max(u)
        if np.min(u) < curr_min:
            curr_min = np.min(u)

if np.max(u) > curr_max:
    curr_max = np.max(u)
if np.min(u) < curr_min:
    curr_min = np.min(u)
```

```
u_end_4 = u.copy()

fig, (ax1, ax2) = plt.subplots(1, 2)

im1 = ax1.imshow(u_1000_4)
im2 = ax2.imshow(u_end_4)
im1.set_clim(curr_min,curr_max)
im2.set_clim(curr_min,curr_max)
fig.colorbar(im1, fraction=0.046, pad=0.04)

plt.show()



syms D u v

a = 3;
b = 8;
% u = a;
% v = b/a;

dudt(u,v) = a-(b+1)*u + u^2*v;
dvdt(u,v) = b*u -u^2*v;

% jacobian(dudt,u)
% jacobian(dvdt,v)
% det(jacobian([dudt,dvdt],[u,v]))

disc(u,v) = (D*jacobian(dudt,u) + jacobian(dvdt,v))^2-4*D*det(jacobian([dudt,dvdt],[u,v]))

sol = solve(disc(a,b/a)==0, D)
```