Scientific Computing Project 4b

Jakob Schauser, pwn274

October 2023

1 The simulation in general

I have implemented the simulation as follows:

The laplacian is written out using for-loops, as I am JIT (just in time)-compiling the code, making it faster than using np.roll or implementations like that.

I liked my own implementation from project 4a, so I tried building the solution in a similar manner.

```
def get_equations(dx : float, D_p : float, D_q : float, C : float, K : float):
    laplacian = get_laplacian(dx)
    @njit
    def dp_dt(p_q):
        p = p_q[:,:,0]
        q = p_q[:,:,1]
        return D_p*laplacian(p) + p*p*q + C - (K+1)*p

    @njit
    def dq_dt(p_q):
        p = p_q[:,:,0]
        q = p_q[:,:,1]
        return D_q*laplacian(q) - p*p*q + K*p

    return dp_dt, dq_dt
```

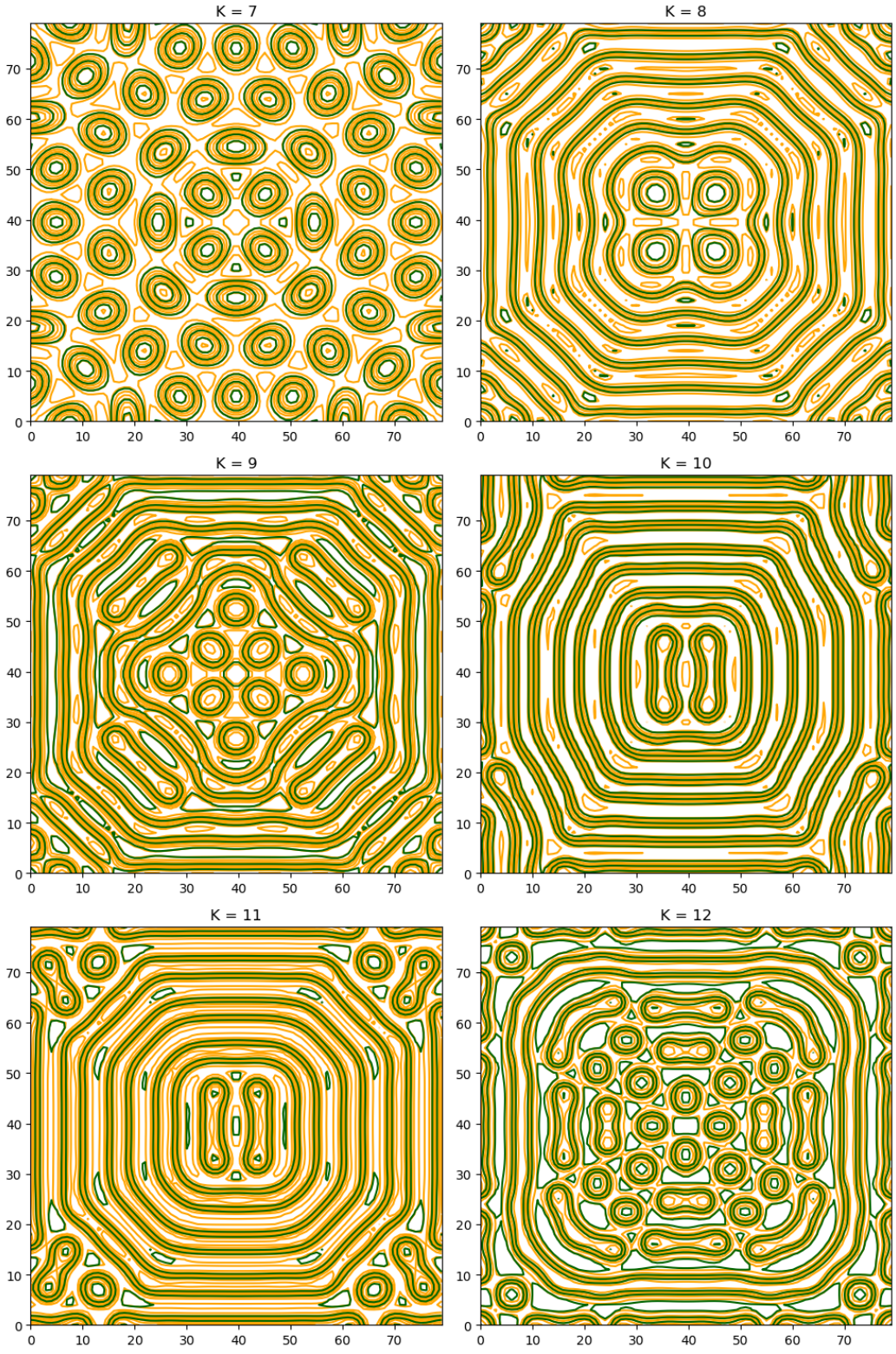
These differential equations can then be solved by:

```
def get_step(dt : float, dp_dt, dq_dt):
    @njit
    def step(p_q):
        p_q[:,:,0] += dp_dt(p_q)*dt
        p_q[:,:,1] += dq_dt(p_q)*dt
        return p_q
    return step

@njit
def the_loop(p_q, n_steps : int, step_fn):
    for i in prange(n_steps):
        p_q = step_fn(p_q)
    return p_q
```

I would have loved a lower dx, but my solution diverged if it was lower than 0.5 without scaling dt accordingly.

Running the simulation once for each of the given K's I can generate the following pretty picture:

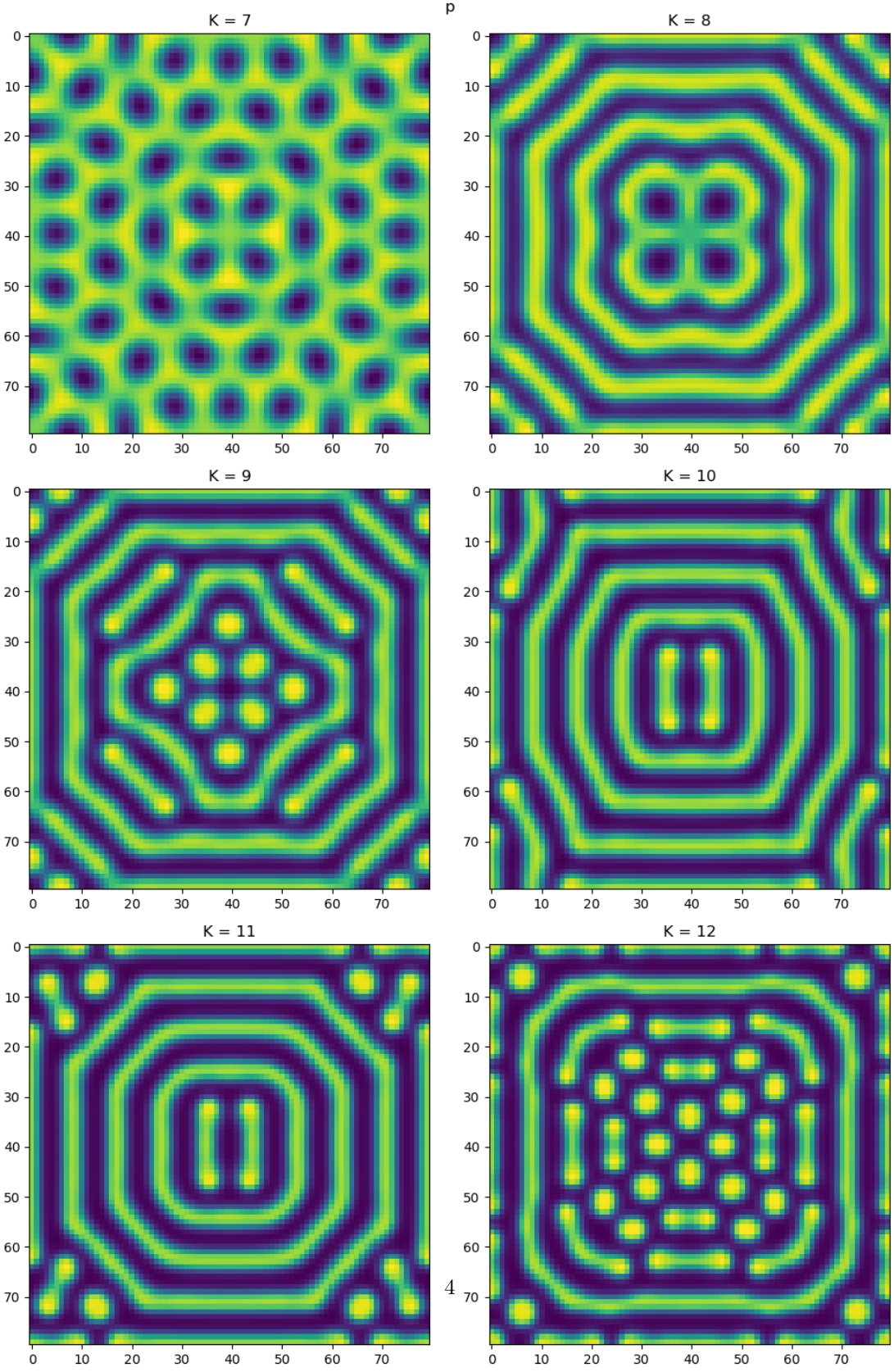

K = 7
K = 8
K = 9
K = 10
K = 11
K = 12
0
10
20
30
40
50
60
70

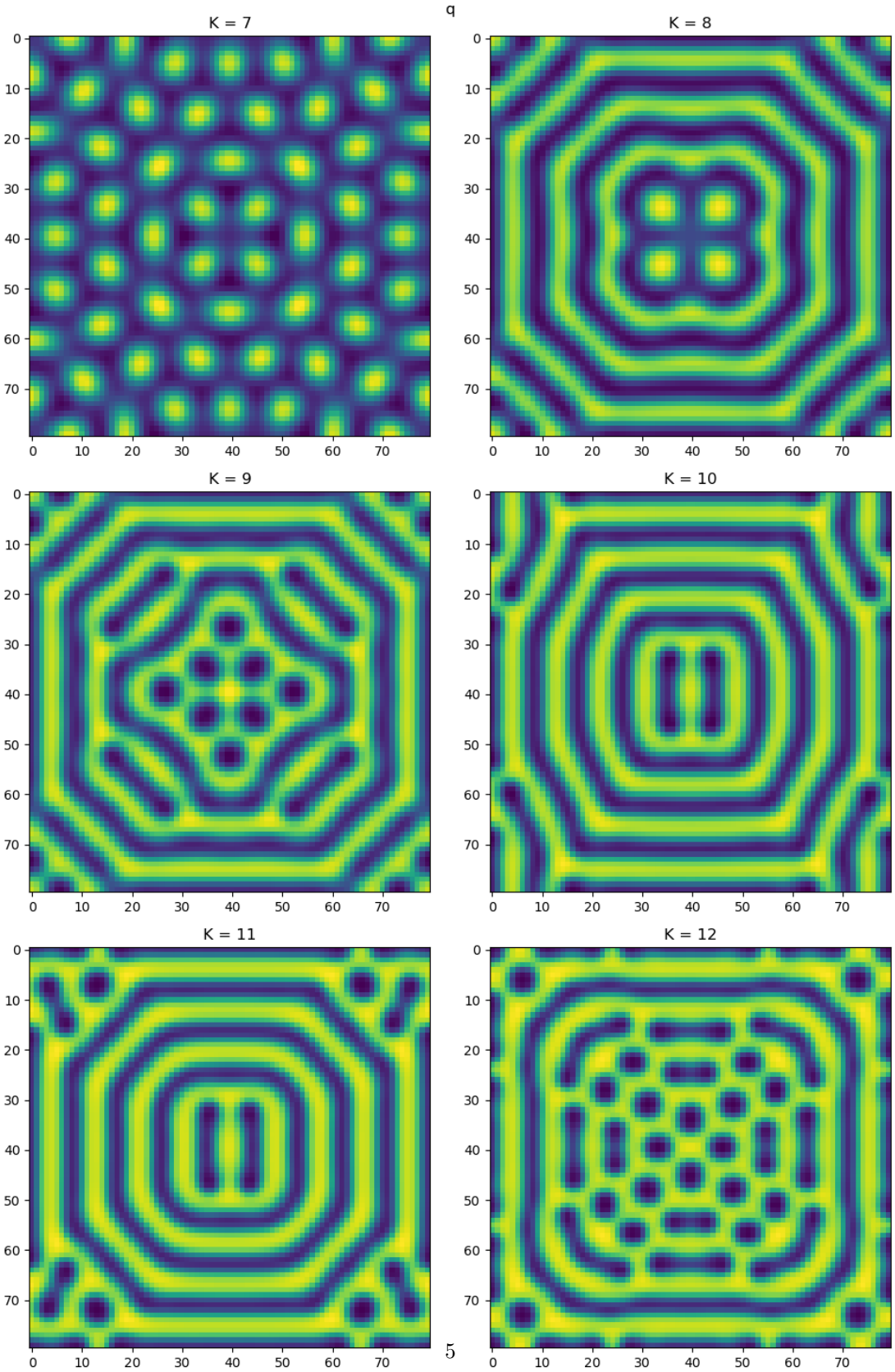
I have chosen to have the p contours be orange and the q contours be green. If it is not clear what the p- and q-solutions look like, they can be found individually in the appendix.

An interesting thing to note is, that for some of the K-values, there seems to have been a break in symmetry at some point. I am unsure how this might have turned up, but I suspect floating point/rounding errors.

Appendix

p




q
K = 7
K = 8
K = 9
K = 10
K = 11
K = 12
0
10
20
30
40
50
60
70