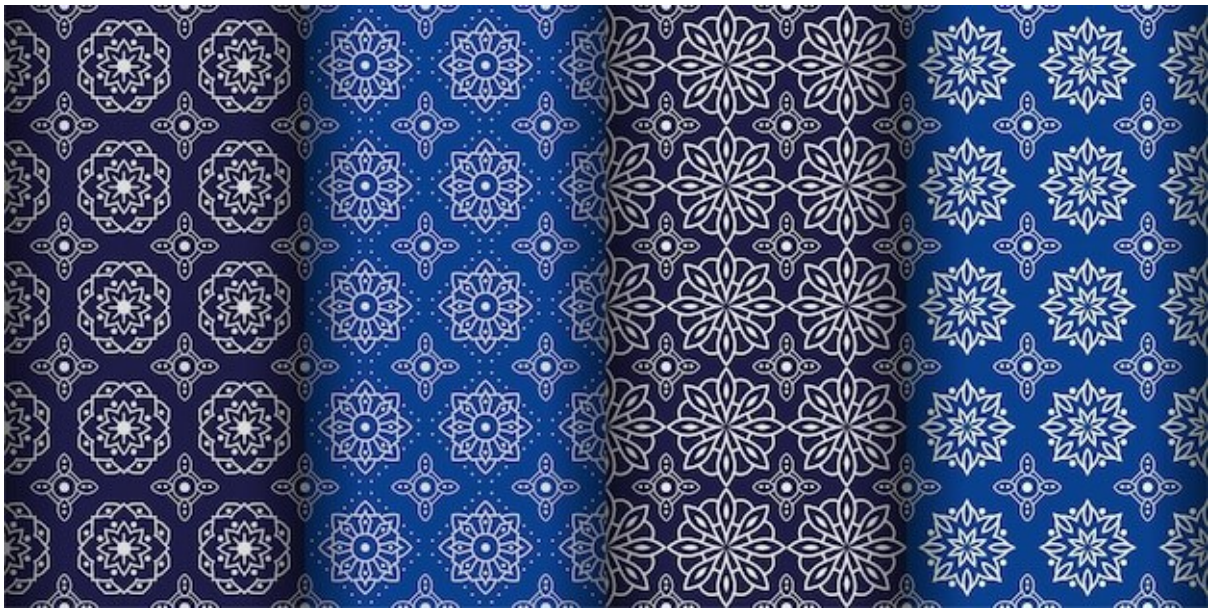
Batik Pattern Inspiration from the Gray-Scott Model



"Love is like a batik created from many emotional colors,
it is a fabric whose pattern and brightness may vary."

Diane Ackerman.

東北大学

Mathematical Modeling and Computation

FINAL REPORT

Muhammad Alfiyandy Hariansyah

Department of Aerospace Engineering

IMAC-G 2021 – C1TM9124

August 10th, 2022

1. INTRODUCTION

Batik is a technique of wax-resist dyeing applied to the whole cloth. This technique originated from the island of Java, Indonesia. Batik is made either by drawing dots and lines of the resist with a spouted tool called *canting*, or by printing the resist with a copper stamp called a *cap*, see Figure 1. The applied wax resists dyes and therefore allows the artisan to color selectively by soaking the cloth in one color, removing the wax with boiling water, and repeating if multiple colors are desired.

On October 2, 2009, UNESCO officially recognized the batik (written batik (*batik tulis*)) and stamped batik (*batik cap*) as a Masterpiece of Oral and Intangible Heritage of Humanity from Indonesia [1] and encouraged the Indonesian people and the Indonesian government to safeguard, transmit, promote, and develop the craftmanship of batik. Since then, Indonesia celebrates "the National Batik Day" annually on October 2. Nowadays, Indonesians wear batik in honor of this ancient tradition.

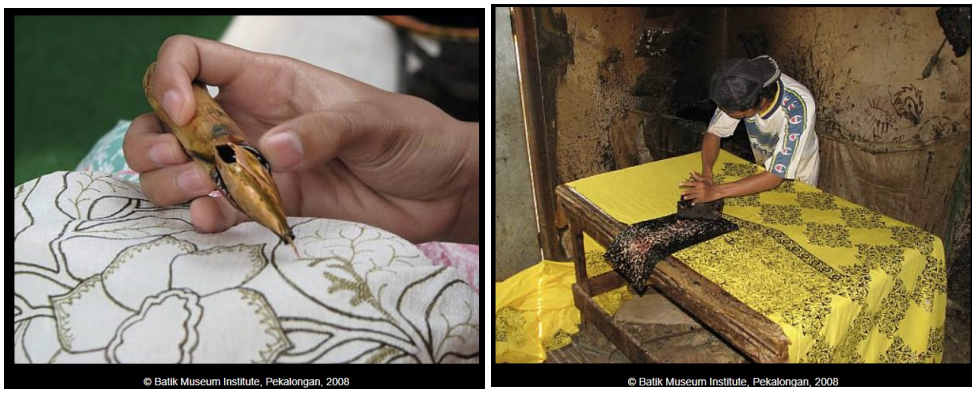


Figure 1. The written batik (left) and the stamped batik (right).

Batik was originally worn at local traditional events. However, in modern times, it is often worn in formal events. In 2013, for example, the leaders of Asia-Pacific Economic Cooperation (APEC) were wearing it at the APEC meeting in Bali, see Figure 2.

Figure 2. The leaders of APEC wearing batik at APEC 2013 meeting in Bali.

Batik has various patterns that originate from different places. The patterns are influenced by the cultures and values of the people. Figure 3 shows several ancient batik patterns from different places in Indonesia.

Figure 3. Several batik patterns from different places in Indonesia.

Although there are ancient patterns that have been around for centuries, many new patterns are being created over time. The ancient patterns were often inspired from the patterns that appear in nature, for example, leaves, trees, flowers, etc. In modern times, most of the young generations in Indonesia are not interested in the old patterns and are constantly finding new patterns for their outfits. Hence, it is important to find a way to obtaining new batik patterns.

In this final report, I aim to introduce a new way to create batik patterns from the famous mathematical model called the Gray-Scott (GS) model. The idea is to numerically solve the GS equations and track the pattern as the iteration increases. By varying the equation parameters, some interesting patterns that are proper for batik motifs can be found at certain iterations.

This report is structured as follows. In Section 2, the mathematical modeling including the equations being solved is explained. Section 3 details the numerical simulation setup including the machine being used to perform the simulation. In Section 4, the simulation results are discussed where I produce 4 different patterns by changing parameters of the same equations. Finally, the conclusion is drawn in the Section 5. The codes are provided in the Appendix.

2. MODELING

The Gray-Scott model is a type of reaction-diffusion system which corresponds to a change in space and time of the concentration of one or more chemical substances. The GS model is one of the archetypes of the fascinating "complex behavior from simple model" paradigm pioneered by the great Alan Turing in his masterpiece paper, *The Chemical Basis of Morphogenesis* (1952) [2].

In the GS model, two hypothetical substances A and B are being used to "draw on a blank canvas". The two chemicals are let to react and diffuse on a 2D grid according to some mathematical equations as follows.

$$\frac{\partial A}{\partial t} = D_A \nabla^2 A - AB^2 + f(1 - A) \tag{1}$$

$$\frac{\partial B}{\partial t} = D_B \nabla^2 B + AB^2 - (k + f)B \tag{2}$$

Figure 4 illustrates the whole process of the reaction-diffusion system. In our canvas illustration, imagine we are pouring chemical A into a canvas at a given feed rate f. In the diffusion process, the two chemicals are allowed to spread out across the grid, governed by the $D_A \nabla^2 A$ and $D_B \nabla^2 B$ terms, where D_A and D_B are the diffusion coefficients. The chemical A can be converted into a chemical B by the presence of two chemical Bs, this is the reaction process. This process is governed by the AB^2 term. Meanwhile, the chemical B is also removed at a given kill rate k.

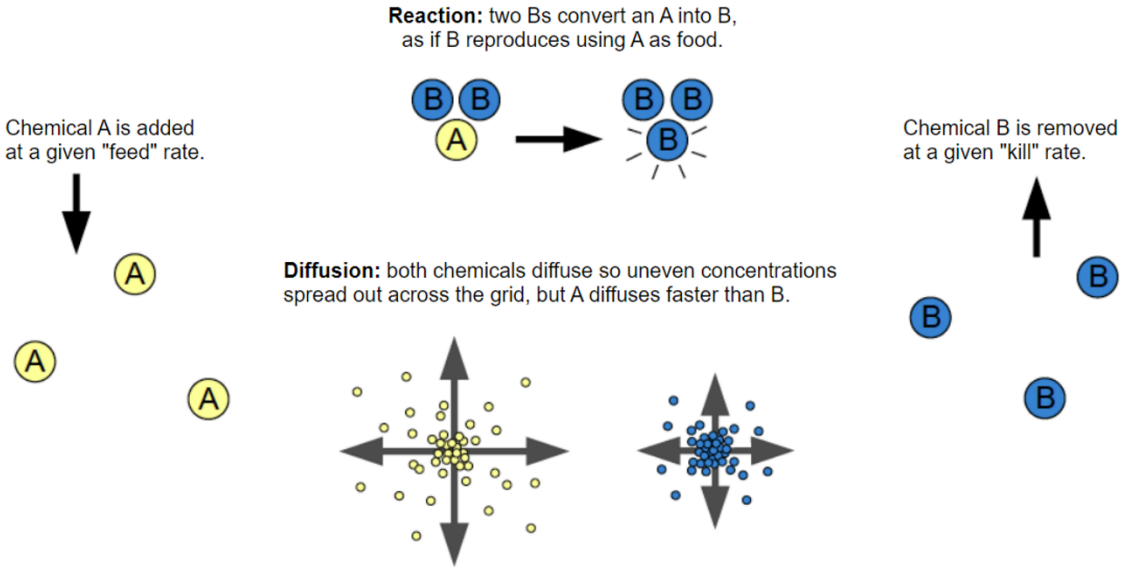


Figure 4. The reaction-diffusion process of two chemical substances [3].

3. SIMULATION

To numerically solve Eq. 1 and 2, both are discretized via Euler method as follows.

$$A^{n+1} = A^n + (D_A \nabla^2 A - AB^2 + f(1 - A)) \cdot \Delta t \tag{3}$$

$$B^{n+1} = B^n + (D_B \nabla^2 B + AB^2 - (k + f)B) \cdot \Delta t \tag{4}$$

where A^{n+1} and B^{n+1} are the values at iteration $n + 1$ and A^n and B^n are the values at iteration n. Δt is the time step. The Laplacians of A and B ($\nabla^2 A$ and $\nabla^2 B$) are calculated using the 5-point approximation, as follows.

$$\nabla^2 A \approx \frac{A_{i+1,j} + A_{i,j+1} + A_{i+1,j+1} + A_{i-1,j-1} - 4A_{i,j}}{\Delta s^2} \tag{5}$$

$$\nabla^2 B \approx \frac{B_{i+1,j} + B_{i,j+1} + B_{i+1,j+1} + B_{i-1,j-1} - 4B_{i,j}}{\Delta s^2} \tag{6}$$

where Δs is the numerical spatial step.

The simulations were done on a two-dimensional 256 x 256 grid. For the initial condition, we started by filling the grid with full chemical A ($A = 1$) and no chemical B ($B = 0$), except in the 32 x 32 rectangular region in the center where chemical B exists ($B = 1$). The boundary conditions are not important in this case, and we can remove the boundary values since we only care with the inner patterns. However, one might use the Neumann boundary condition so that the boundary values are the same with their adjacent values. In this report, we just "paint the boundaries with red".

Four simulations with four different sets of coefficients were conducted. The coefficient settings with their produced pattern names are listed below.

1. Cell division pattern
   $$D_A = 0.14, D_B = 0.06, f = 0.035, k = 0.065$$
2. Coral pattern
   $$D_A = 0.16, D_B = 0.08, f = 0.060, k = 0.062$$
3. Spiral pattern
   $$D_A = 0.12, D_B = 0.08, f = 0.020, k = 0.050$$
4. Middle East pattern
   $$D_A = 0.16, D_B = 0.08, f = 0.035, k = 0.060$$

The simulations were conducted on an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz (16 CPUs) machine. Although the machine has 16 processors, no parallel computation was applied. However, the code is structured so that a vectorized computation was done using NumPy library [4] in Python, see the Appendix for the codes.

4. RESULTS AND DISCUSSION

4.1. Initial pattern

We started by filling all the 256 x 256 grid cells with full homogenous chemical A. To initiate the reaction-diffusion process, we only filled a 32 x 32 grid in the center with chemical B. The initial pattern is shown in Figure 5 below.

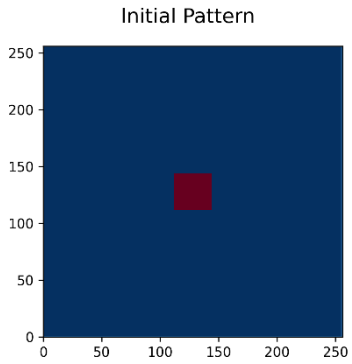


Figure 5. The initial pattern for all the simulation settings.

4.2. Pattern growth

Next, we performed all the 4 simulation settings and tracked the pattern growth as the iteration increased. Here, we set the space step $dS = 1.0$, the time step $dT = 1.0$, and the total time $T = 5000$. Thus, the maximum iteration number is 5000. We plotted 9 different patterns at 9 equally spaced iterations. Figure 6 (a)-(d) show the pattern growth for every simulation.

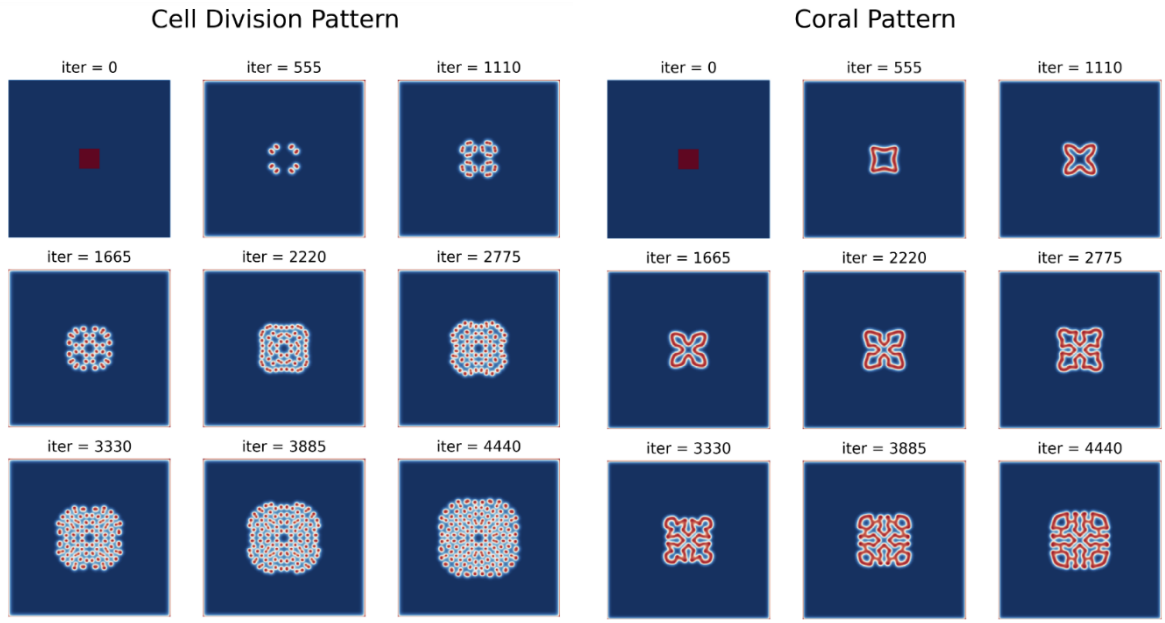
It can be observed that different simulation settings result in different pattern formations. Each pattern has their own characteristics. The cell division pattern resembles how the mitosis (cell division) occurs. The coral pattern resembles the coral pattern that we see in nature. The spiral pattern also consists of spiral-like geometries. The Middle East pattern resembles motifs that we often see in the Middle East cultures, greatly influenced by the Islamic cultures, e.g., mosque architectures.

Although each pattern at every iteration in the same simulation settings show different shapes, they share the same characteristics. The batik designer could use one of the patterns developed as the iteration increases. The chosen pattern can then be a building block that can be repeated to form a particular motif. That is the idea of tracking the pattern growth.

Table 1 shows the actual simulation time for every setting. It is noted that the times to plot the 9 patterns are included. The average time is 34.65 s, which is quite cheap to perform.

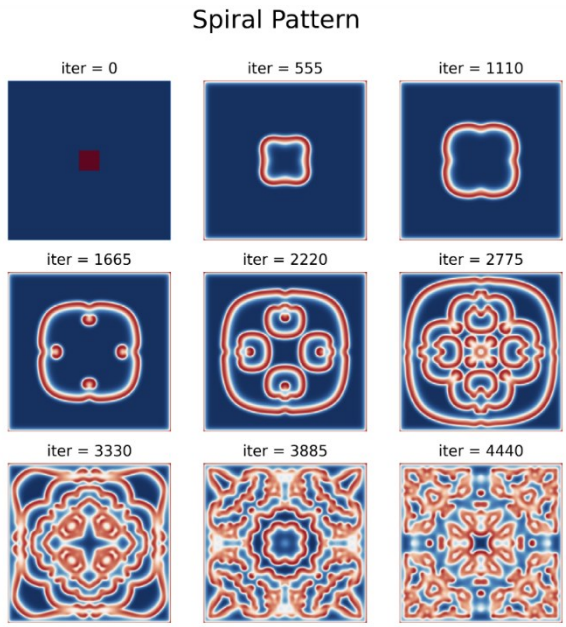
Table 1. The simulation times for 5000 iterations.

Cell Division	Coral	Spiral	Middle East	Average
36.32 s	36.53 s	31.54 s	34.19 s	**34.65 s**

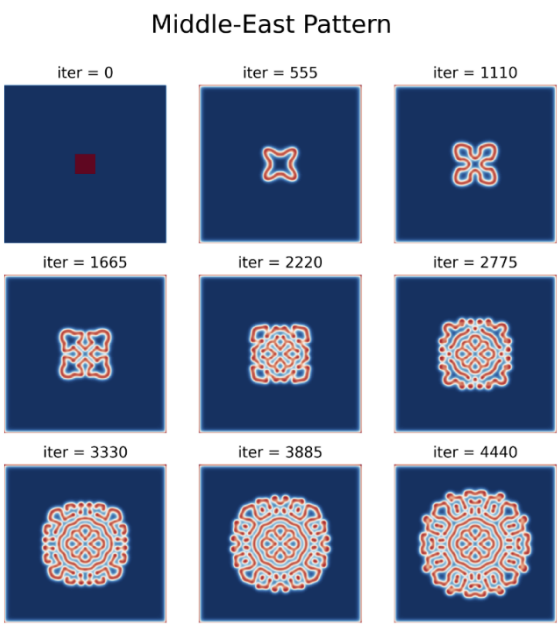


(a) Cell Division Pattern

(b) Coral Pattern



(c) Spiral Pattern



(d) Middle East Pattern

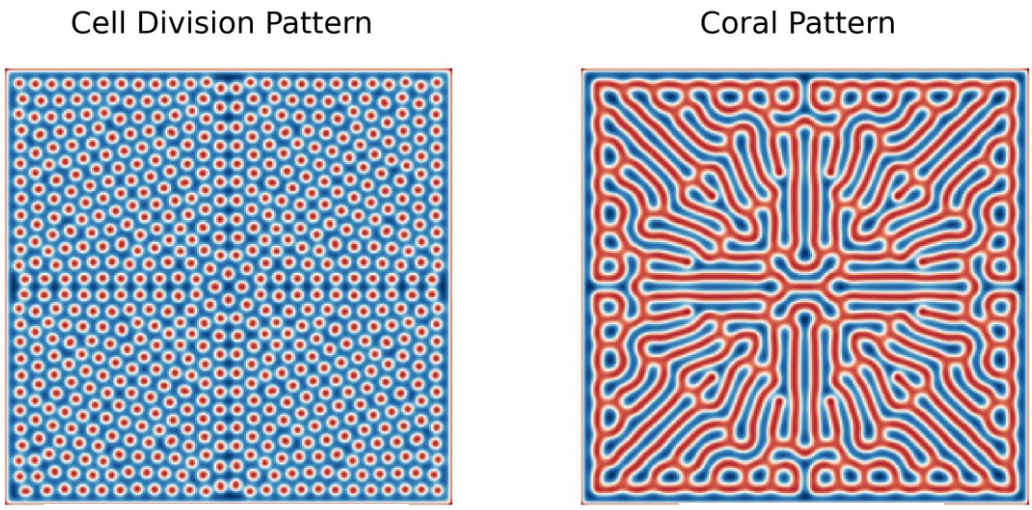
Figure 6. The pattern growth for every simulation setting.

4.3. Full simulations

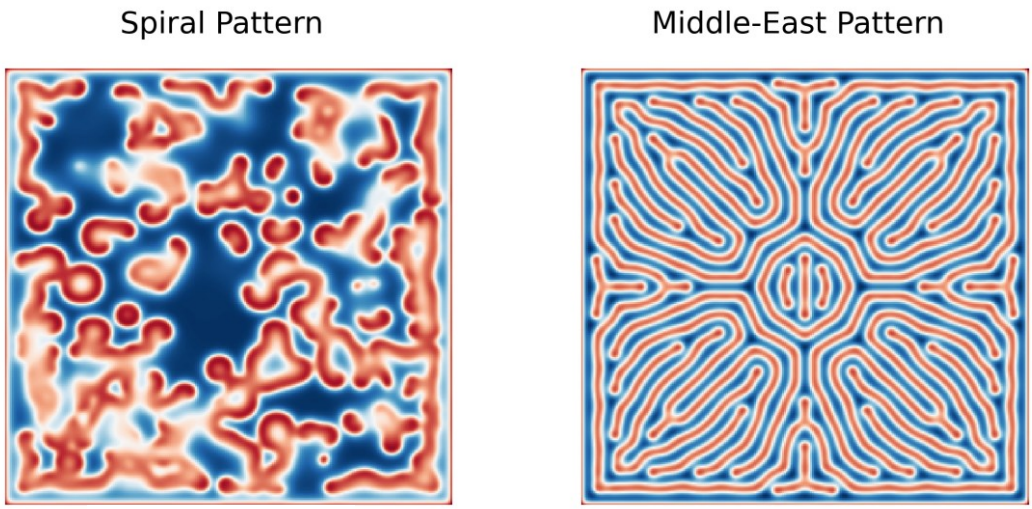
We also performed full simulations with more iterations using the same settings: the space step $dS = 1.0$, the time step $dT = 1.0$. However, the total time is now $T = 32000$, making the maximum iteration number to be 32000. The resulting patterns at the last iteration are shown in Figure 7, and the actual simulation times are listed in Table 2. The actual simulation does not include the plotting time since there is no need to plot in between iterations. The average simulation time is 111.07 s, which is still quite cheap to perform.

Table 2. The simulation times for 32000 iterations.

Cell Division	Coral	Spiral	Middle East	Average
109.42 s	112.26 s	110.09 s	112.50 s	**111.07 s**



(a) Cell Division Pattern (b) Coral Pattern



(c) Spiral Pattern (d) Middle East Pattern

Figure 7. The pattern for each simulation after 32000 iterations.

It can be observed that after 32000 iterations, each simulation seems to converge to their respective patterns. The coral pattern looks almost like the Middle East pattern in their fully developed phase, although they show quite different features in their underdeveloped phases. In my opinion, these fully developed patterns are less useful than their underdeveloped patterns, shown in Figure 6. It is since they are not proper to be used as the building block of a batik motif. However, the designer could freely choose whichever pattern produced by the Gray-Scott model.

5. CONCLUSION

In this report, we utilized the Gray-Scott model to help find inspirations for batik patterns/motifs. The Gray-Scott model is discretized using Euler method and solved numerically in Python with the help of NumPy [4] and SciPy [5] as the computing libraries and Matplotlib [6] as the visualization library. I conducted four different simulation settings with the same initial pattern condition and tracked the intermediate patterns developed in between iterations. The full simulations with more iterations were also conducted. The pattern growth seems to be useful to be used as the building block that can be repeated to form a batik motif. The fully developed pattern is thought to be less useful, however the batik designer could still use it. The designers can play around with the simulation parameters to produce different patterns. The initial condition can also be changed, e.g., create a circular shape as the initial seed or apply several of them at different places instead of just one in the center. To wrap up, the Gray-Scott model is useful in obtaining batik pattern inspirations for batik designers and can act as an alternative way to produce patterns.

REFERENCES

[1] https://ich.unesco.org/en/RL/indonesian-batik-00170

[2] Alan Turing. *The Chemical Basis of Morphogenesis*. Philosophical Transactions of the Royal Society of London B. 237: 32-72 (1952).

[3] Karl Sims. *Reaction-Diffusion Tutorial*. (2013). http://www.karlsims.com/rd.html

[4] NumPy: The fundamental package for scientific computing with Python. https://numpy.org/

[5] SciPy: Fundamental algorithms for scientific computing in Python. https://scipy.org/

[6] Matplotlib: Visualization in Python. https://matplotlib.org/

APPENDIX

The appendix is structured as follows:

(1) Importing libraries

(2) Defining functions

(3) Pattern growth simulation.

(4) Full pattern simulation.

1. Importing libraries

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import spdiags
import time
```

2. Defining functions

```
def laplacian(N):
    """
    Create a sparse matrix to calculate the laplacian
        N = grid size
    """
    e = np.ones(N**2)
    e2 = ([1]*(N-1)+[0])*N
    e3 = ([0]+[1]*(N-1))*N
    L = spdiags([-4*e,e2,e3,e,e],[0,-1,1,-N,N],N**2,N**2)
    return L
```

```
def initialize(N):
    """
    Setting up the initial condition
        N = grid size
    """
    N2, r = np.int64(N/2), 16

    # We start with every grid cell that has a lot of chemical A
    # and no chemical B except at the small region in the center
    A = np.ones((N, N))
    B = np.zeros((N, N))

    B[N2-r:N2+r, N2-r:N2+r] = 1.0

    A = A.reshape((N*N))
    B = B.reshape((N*N))

    return A, B
```

```
def drawPattern(A, ax=None):
    """
    Draw the pattern based on the chemical concentrations
        A = Concentration of chemical A
        ax = axis object
    """
    ax.pcolor(A.reshape((N,N)), cmap=plt.cm.RdBu)
    ax.set_axis_off()
#     ax.axis('tight')
```

```
def update(A, B, dT, Da, Db, F, K, L):
    """
    Update the iteration via Euler method
        A = Concentration of chemical A
        B = Concentration of chemical B
        dT = Time step
        Da = Diffusion coefficient of chemical A
        Db = Diffusion coefficient of chemical B
        F = Feed rate
        K = Kill rate
        L = Sparse matrix for Laplacian
    """
    A += (Da*L.dot(A) - A*B*B + F*(1-A)) * dT
    B += (Db*L.dot(B) + A*B*B - (F+K)*B) * dT

    return A, B
```

3. Pattern growth simulations.

Change Da, Db, F, and K with respective values from the simulation settings!

```
N = 256       # Grid size
T = 5000     # Total time
dT = 1.0      # Time step
n = int(T/dT) # Number of iterations
Da, Db, F, K = 0.14, 0.06, 0.035, 0.065  # Coefficients

fig, axes = plt.subplots(3,3, figsize=(8,8), dpi=400, facecolor='w', edgecolor='k')
step_plot = n // 9

# Simulation part
t1 = time.time()

A, B = initialize(N)
L = laplacian(N)

for i in range(n):
    # Iteration update
    update(A, B, dT, Da, Db, F, K, L)

    # Plot at 9 different times
    if i % step_plot == 0 and i < 9 * step_plot:
        ax = axes.flat[i // step_plot]
        drawPattern(A, ax=ax)
        ax.set_title(f'iter = {i}')

fig.suptitle(f'Cell Division Pattern',fontsize=20)
t2 = time.time()

print(f'Simulation time = {t2-t1} seconds')
```

4. **Full pattern simulation.**

Change Da, Db, F, and K with respective values from the simulation settings!

```
N = 256       # Grid size
T = 3.2e4     # Total time
dT = 1.0      # Time step
n = int(T/dT) # Number of iterations
Da, Db, F, K = 0.14, 0.06, 0.035, 0.065  # Coefficients

fig, ax = plt.subplots(1, dpi=400, figsize=(4,4), facecolor='w', edgecolor='k')

# Simulation part
t1 = time.time()

A, B = initialize(N)
L = laplacian(N)

for i in range(n):
    # Iteration update
    update(A, B, dT, Da, Db, F, K, L)

t2 = time.time()

print(f'Simulation time = {t2-t1} seconds')

# Plot at the last iteration
drawPattern(A, ax=ax)
fig.suptitle(f'Cell Division Pattern',fontsize=15);
fig.savefig('../../Figures/Cell_Division_Pattern.png',format='png',dpi=300)
```