Scientific Computing Exercise Set 2

Keyuan Wang(13594869) & Jialin Dong(13610848)

I. INTRODUCTION

Diffusion is an essential mechanism in many physical phenomena. Diffusion equation also plays an important role in modelling these phenomena mathematically. From the motion of molecules to weather prediction, many natural phenomenons can be described by *partial differential equation* (*PDE*) systems. In this paper, the numerical solution of two diffusion PDE systems describing real-world phenomenons is discussed.

The first one is *Diffusion Limited Aggregation*, proposed by T.A. Witten Jr. and L.M. Sander in 1981 [1], which describes the process where particles undergoing Brown motion aggregate together and form a cluster. Such a process can be numerically modeled either by Monte Carlo Simulation with the random walker, or by solving the *Laplace Equation*. Both of the methods will be discussed in this paper.

Apart from DLA, the Gray-Scott model is discussed, which is a reaction-diffusion model raised by P.Gray and S.K.Scott [2]. It describes a chemical reaction system with two components U and V involved. Both chemical components diffuse in the system, and react with each other in the following way

$$U + 2V \rightarrow 3V$$

$$V \rightarrow P$$

in which U is continuously supplied to the system, and react with V to produce more V, while V spontaneously transfer to P, and P does not participate in the reaction. Such a process can be described by a reaction-diffusion system, and solved numerically.

In the next chapters, the theory behind these models will be clearly presented. An overview of the methods used for modelling, simulation and speeding up the solving process will be given. And the experiment results will be discussed. Finally, some conclusions will be summarized from the results.

II. THEORY

A. Diffusion Limited Aggregation

The Laplace Equation is a time-independent second-order partial differential equation

$$\nabla^2 c = 0 \tag{1}$$

where $\nabla^2 c$ is the Jacobian of concentration. It assumes the initial conditions and periodic boundary condition are listed as follows:

$$\begin{cases} c(x, y = 1; t) = 1, c(x, y = 0; t) = 0 \\ c(x = 0, y; t) = c(x = 1, y; t) \\ c(x, y; t = 0) = 0, (0 \leq x \leq 1, 0 \leq y < 1) \end{cases} \tag{2}$$

Discretizing the spatial and temporal domain gives

$$\begin{cases} x \in (1\delta x, 2\delta x, i\delta x \dots N\delta x) \\ y \in (1\delta y, 2\delta y, j\delta y \dots N\delta y) \\ t \in (1\delta t, 2\delta t, k\delta t \dots N_t\delta t) \end{cases} \tag{3}$$

Discretizing this system by *finite difference approach*

$$c_{i,j} = \frac{1}{4}(c_{i+1,j} + c_{i-1,j} + c_{i,j+1} + c_{i,j-1}) \tag{4}$$

In this paper, the *Successive Over Relaxation* (*S.O.R*) method is applied to solve Equation 4.

$$c_{i,j}^{k+1} = \frac{\omega}{4}(c_{i+1,j}^{k} + c_{i-1,j}^{k+1} + c_{i,j+1}^{k} + c_{i,j-1}^{k+1}) + (1 - \omega)c_{i,j}^{k} \tag{5}$$

This method only converges with $0 < \omega < 2$.
A threshold is also needed to check if the solution converges

$$\delta \equiv \max_{i,j} |c_{i,j}^{k+1} - c_{i,j}^{k}| < \epsilon \tag{6}$$

where ϵ is some small number, say 10^{-5}.

It is worthy noting that iterative S.O.R method can only be implemented sequentially, cell by cell. For a modern multi-core computer, parallel algorithms can be up to $100\times$ times faster than sequential algorithms. Therefore, a parallel S.O.R iterative method raised by D.J. Evans [3] is introduced as well.

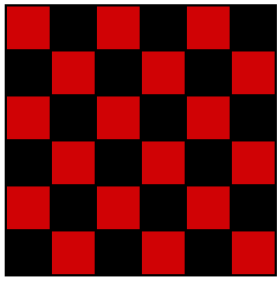

Fig. 1: Red-Black Ordering for a 6x6 grid

To apply the parallel algorithm, the 2-D grid is separated into two parts first: Red block and black block. Red block consists of all grid cells $i + j$ odd, and the black block consists of all cells $i+j$ even. Figure 1 shows an example with a 6×6 grid.

Using S.O.R, all the cells belong to red block are updated first

$$c_{i,j}^{k+1} = \frac{\omega}{4}(c_{i+1,j}^{k} + c_{i-1,j}^{k} + c_{i,j+1}^{k} + c_{i,j-1}^{k}) + (1-\omega)c_{i,j}^{k} \tag{7}$$

Then the cells belong to black block are updated based on the value of red cells.

$$c_{i,j}^{k+1} = \frac{\omega}{4}(c_{i+1,j}^{k+1} + c_{i-1,j}^{k+1} + c_{i,j+1}^{k+1} + c_{i,j-1}^{k+1}) + (1-\omega)c_{i,j}^{k} \tag{8}$$

Note that all the red cells should be updated before the black cells, but the updating sequence between the cells with the same color is not important. Thus, the cells with same color can be updated parallelly.

To model the DLA process, an aggregation cluster is initialized on the grid. The concentrations within the cluster are set to 0, similar to a sink object. Then the PDE system with DLA cluster involved is solved. After convergence. A bunch of possible growth candidates located around the cluster are generated. Figure 2 shows a possible configuration of cluster and growth candidates.

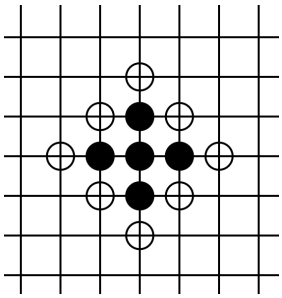

Fig. 2: Aggregation cluster (filled dots) and possible growth candidates (open circles).

Each of the growth candidate is associated with a possibility

$$p_g(i,j) = \frac{c_{i,j}^{\eta}}{\sum_{\text{growth candidates}} c_{i,j}^{\eta}} \tag{9}$$

where η determines the shape of the DLA object, which will be discussed in following chapters. For each time step, only one growth candidate will be added to the cluster. The selection procedure is based on the associated possibility. After new candidate added, the system is solved again, and the whole procedure repeats until a given number of iteration is reached.

B. Monte Carlo Simulation of DLA

This part simulates the DLA by Monte Carlo simulation. At first, it sets a single seed at the bottom of the grid. And then, a series of random walkers are generated, which starting roaming within the grid. There are four directions: up, down, left, and right, for walkers at one time step. If the walker come into a cell neighboring the object, it will stop roaming and becomes a part of the whole cluster. Besides that, the left and right boundaries are periodical boundaries for constraining the domain. If it reaches to the top and bottom boundaries, it will be removed and create new one.

Also for the Monte Carlo Simulation, the growth probability η is replaced as sticking probability p_s. Which means if a walker enters into a cell neighboring the cluster, it stops at this place with probability p_s. If not sticking, the walker continues the next movement. However, the walker is not allowed to move into a site belonging to the cluster.

Here it sets a discretized domain of this Monte Carlo system. The whole grid consists of $N \times N$ cells. The number of time steps are N_t. Discretizing the spatial and temporal domain:

$$\begin{cases} x \in (1\delta x, 2\delta x, i\delta x \dots N\delta x) \\ y \in (1\delta y, 2\delta y, j\delta y \dots N\delta y) \\ t \in (1\delta t, 2\delta t, k\delta t \dots N_t\delta t) \end{cases} \tag{10}$$

where i represents the mesh point in the spatial domain, and k represents the mesh point in the temporal domain. The basic spatial and temporal units are δx, δy, and δt.

It assume the initial conditions and periodic boundary condition are listed as follows:

$$\begin{cases} c(x=0,y;t) = c(x=N,y;t) \\ c(x,y=N+1;t) = c(x,y=0-1;t) = null \\ c(x,y;t=0) = 0, (0 \le x \le N, 0 \le y < N) \end{cases} \tag{11}$$

C. The Gray-Scott Model

As introduced in the Chapter I, the Gray-Scott model can be described by a reaction-diffusion system

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + f(1-u) \tag{12}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (f+k)v \tag{13}$$

where D_u and D_v denote the diffusion coefficient for u and v respectively, f controls the rate at which u is supplied, and $(f+k)$ controls the rate at which v decays.

Assume the system is implemented in two dimensions (i.e. in a x-y plane), discretize the system

$$u_{i,j}^{k+1} = u_{i,j}^{k} + \frac{\delta t}{\delta^2 x} D_u (u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j-1}^{k} + u_{i,j+1}^{k} - 4u_{i,j}^{k}) - \delta t \left(u_{i,j}^{k} (v_{i,j}^{k})^2 - f(1 - u_{i,j}^{k}) \right) \tag{14}$$

$$v_{i,j}^{k+1} = v_{i,j}^{k} + \frac{\delta t}{\delta^2 x} D_v (v_{i+1,j}^{k} + v_{i-1,j}^{k} + v_{i,j-1}^{k} + v_{i,j+1}^{k} - 4v_{i,j}^{k}) + \delta t \left(u_{i,j}^{k} (v_{i,j}^{k})^2 - (f + k)(v_{i,j}^{k}) \right) \tag{15}$$

Assume a periodic boundary condition in both x and y direction

$$\begin{cases} c(x, y = 1; t) = c(x, y = 0; t) \\ c(x = 0, y; t) = c(x = 1, y; t) \end{cases} \tag{16}$$

For the initial condition, assume $u = 0.5$ everywhere in the system, and $v = 0.25$ in a small square in the center of the system, say the length is 0.2
Then the Gray-Scott model can be solved numerically at any given time point.

III. METHODS

In this paper, all codes are based on Python 3.9. A powerful library `Numpy` is used for modelling and simulation. The `Numba` package is used to accelerate the computing speed. Besides, `Matplotlib` and `Seaborn` are used to generate plots and animations.

A. Diffusion Limited Aggregation

The algorithm 1 shows how to solve the Laplace equation numerically with iterative S.O.R. method. while the algorithm 2 shows how the parallelized S.O.R. method is implemented. It should be noted that all the concentrations in red cells ($i + j =$ odd) are calculated simultaneously, same for all the concentrations in the black cells ($i + j =$ even). To speed up the simulation process, the `Numba` package is used, which can accelerate the program by translating Python functions to optimized machine code at runtime. `Numba` also support `CUDA` GPU acceleration.

Based on the Laplace equation, algorithm 3 shows how the diffusion limited aggregation is produced. In the experiment, the simulation is started with the analytical solution for an empty system: $c(x, y) = y$. And for each growth step, the S.O.R. iteration is started with the solution of the previous growth step to speed up the simulation.

B. Monte Carlo Simulation of DLA

The algorithm 4 of Monte Carlo combined with sticking probability illustrates how the simulation works. The sticking probability can be set from 0 to 1. If the sticking probability was set to 1, it means that there is no consideration of sticking event.

C. The Gray-Scott Model

The algorithm 5 shows how the Gray-Scott model is solved numerically. To speed up the simulation, the *Array programming* technique [4] is employed to encode this algorithm, which allows the program to operate on a set of values at once. The `Numpy` package provides good support of it with the *ndarray* data type.

Algorithm 1 Successive Over Relaxation

for $i = 0, 1, 2, ..., N$ **do**
  for $j = 0, 1, 2, ..., N - 1$ **do**
    $c_{i,j}^{0} \leftarrow I(x_i, y_j)$
  end for
end for
$\delta \equiv \max_{i,j} |c_{i,j}^{k+1} - c_{i,j}^{k}|$
while $\delta > \epsilon$ **do**
  for $i = 0, 1, 2, ..., N$ **do**
    $c_{i,N}^{k} = 1, c_{i,0}^{k} = 0$
    for $j = 1, 2, ..., N - 1$ **do**
      if i = 0 **then**
        $i - 1 \leftarrow N$
      else if i = N **then**
        $i + 1 \leftarrow 0$
      end if
      $c_{i,j}^{k+1} \leftarrow \frac{\omega}{4}(c_{i+1,j}^{k} + c_{i-1,j}^{k+1} + c_{i,j+1}^{k} + c_{i,j-1}^{k+1}) + (1 - \omega) c_{i,j}^{k}$
    end for
  end for
end while=0

Algorithm 2 Red-Black Ordering S.O.R.

for $i = 0, 1, 2, ..., N$ **do**
  for $j = 0, 1, 2, ..., N - 1$ **do**
    $c_{i,j}^{0} \leftarrow I(x_i, y_j)$
  end for
end for
$\delta \equiv \max_{i,j} |c_{i,j}^{k+1} - c_{i,j}^{k}|$
while $\delta > \epsilon$ **do**
  for all $i, j \in (i + j) =$ odd **do**
    if i = 0 **then**
      $i - 1 \leftarrow N$
    else if i = N **then**
      $i + 1 \leftarrow 0$
    end if
    $c_{i,j}^{k+1} \leftarrow \frac{\omega}{4}(c_{i+1,j}^{k} + c_{i-1,j}^{k} + c_{i,j+1}^{k} + c_{i,j-1}^{k}) + (1 - \omega) c_{i,j}^{k}$
  end for
  for all $i, j \in (i + j) =$ even **do**
    if i = 0 **then**
      $i - 1 \leftarrow N$
    else if i = N **then**
      $i + 1 \leftarrow 0$
    end if
    $c_{i,j}^{k+1} \leftarrow \frac{\omega}{4}(c_{i+1,j}^{k+1} + c_{i-1,j}^{k+1} + c_{i,j+1}^{k+1} + c_{i,j-1}^{k+1}) + (1 - \omega) c_{i,j}^{k}$
  end for
end while=0

Algorithm 3 Diffusion Limited Aggregation

0: **INITIALIZE** the lattice with empty DLA cluster
0: **INITIALIZE** the DLA cluster in the grid
for $k = 1, 2, 3, ..., N_k$ **do**
1. **U**pdate the lattice with DLA cluster by solving the Laplace Equation 1
2. **G**enerate growth candidates
3. **C**alculate the associated weights by Equation 9
4. **S**elect one growth candidate g based on the associated weights
5. **U**pdate the DLA cluster with g
end for=0

Algorithm 4 Monte Carlo Simulation

for $i = 0, 1, 2, ..., N$ **do**
for $j = 0, 1, 2, ..., N$ **do**
$c^0_{i,j} \leftarrow 0$
end for
end for
$walkers \equiv N_t$
for $k = 1, 2, ..., N_t - 1$ **do**
while $true$ **do**
if $walker \in candidates$ **then**
$p_{random} \in [0, 1]$
if $p_{random} < p_{stick}$ **then**
$objects \leftarrow walker$
break
else
$back \leftarrow walker$
end if
else if $walker \in objects$ **then**
break
else
if $move \in boundaries$ **then**
$periodicmove \leftarrow walker$
else
$move \leftarrow walker$
end if
end if
end while
end for=0

Algorithm 5 The Gray-Scott model

for $i = 0, 1, 2, ..., N$ **do**
for $j = 0, 1, 2, ..., N - 1$ **do**
$u^0_{i,j} \leftarrow I_u(x_i, y_j)$
$v^0_{i,j} \leftarrow I_v(x_i, y_j)$
end for
end for
for $k = 1, 2, ..., N_t - 1$ **do**
for $i = 0, 1, 2, ..., N$ **do**
for $j = 1, 2, ..., N - 1$ **do**
if i = 0 **then**
$i - 1 \leftarrow N$
else if i = N **then**
$i + 1 \leftarrow 0$
end if
$u^{k+1}_{i,j} \leftarrow$ Equation 14
$v^{k+1}_{i,j} \leftarrow$ Equation 15
end for
end for
end for=0

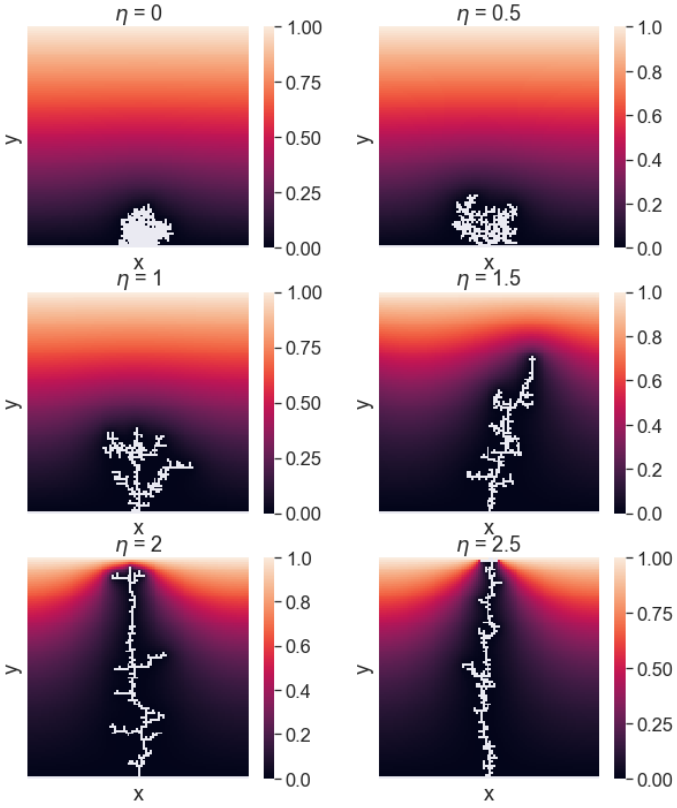

Fig. 3: Diffusion Limited aggregation with 300 growth steps. Grid size: 100×100.

Grid size	Iterative SOR	Parallel SOR	Efficiency
100×100	15,47 s	0.093 s	166×
200×200	178.27 s	1.19 s	150×
220×220	249.84 s	1.56 s	160×

TABLE I: Parallel SOR execution time vs. Iterative SOR execution time

IV. RESULTS

A. Diffusion Limited Aggregation

Figure 3 shows several DLA models with different η values on a 100×100 grid. It is obvious that when η is near 0, the DLA model has a 'flat' shape, and turns to be more and more 'straight' when η increases. When $\eta = 2.5$, the DLA model is very similar to a vertical straight line.

By applying the Red-Black ordering parallel algorithm and `Numba` package. The time required to solve the Laplace Equation is greatly reduced. As shown in the Table I, for a 100×100 grid, the element-wise iterative SOR method needs **15.47 seconds** to converge, while the parallel algorithm only taking **0.093 seconds**. The parallel algorithm is about **160 times** faster than the iterative method.

B. Monte Carlo Simulation of DLA

Figure 4 illustrates the process of cluster growth as time passing. It sets the sticking probability $t_{stick} = 1$, which means the particle must attach on objects if it hits the clusters. Looking through the whole evaluation, the cluster as a small point at start, and grows gradually as taller and stay thin, like a tree. Additionally, comparing the shape growth of $t = 0$ to $t = 6000$ and $t = 24000$ to $t = 30000$, it can be found that initial phase has less changes (nearly invisible) than the last phase. The cluster of $t = 30000$ is about twice larger than the cluster of $t = 24000$. This is because the larger cluster is more likely to catch the random particles roaming in the grid. Also, figure 6 demonstrates the exponential growth speed.

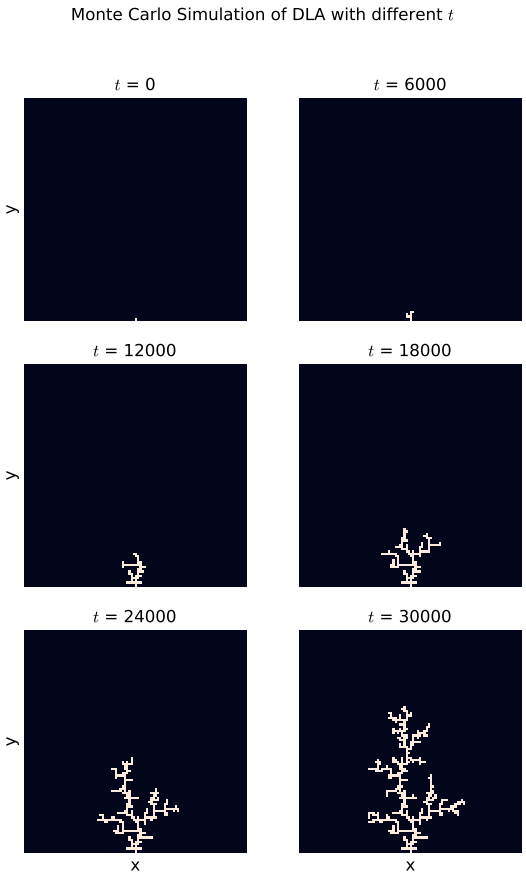

Fig. 4: Sticking probability $p_{stick} = 1$, the number of random walkers t from 0 to 30000, and grid size $N = 100 \times 100$.

Figure 5 compares the impacts of different p_{stick} on the shape of clusters with fixed t. It can be seen that as the p_{stick} increases, the cluster become larger and taller, which corresponds to the theory. With higher p_{stick}, the particle is more likely to be aggregated to the cluster.

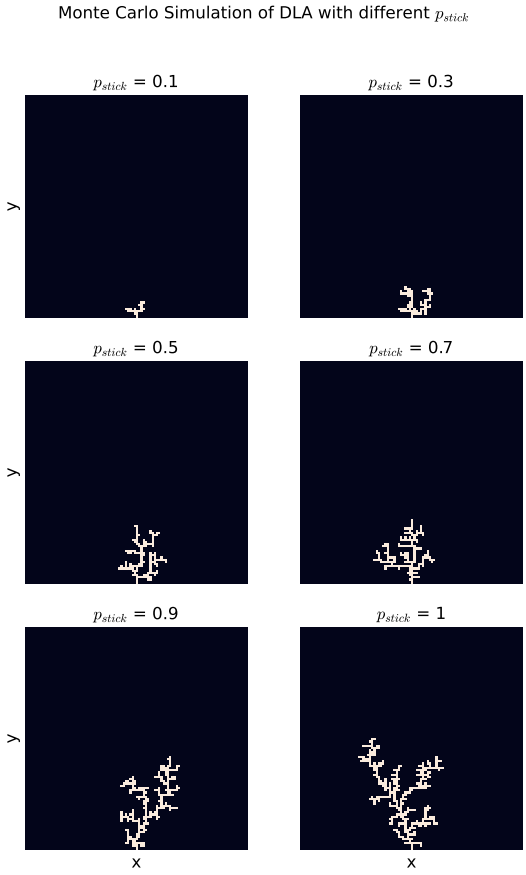

Fig. 5: Sticking probabilities from 0 to 1, fixed number of random walkers $t = 25000$, and grid size $N = 100 \times 100$.

Figure 6 gives an overview of objects trends with different p_{stick}, all of them express an exponential growth speed. The degree of rising is weak with small sticking probability.

C. The Gray-Scott Model

Figure 7 shows the concentrations of U with different parameters after 3000 iterations steps. For a larger f value, the object boundary becomes more smooth. Figure 8 shows the concentrations of U with the same parameters settings. But noise is added in each iteration step at every lattice site, which is normal distributed, with mean 0 and standard deviation 0.005.

V. DISCUSSION AND CONCLUSION

A. Diffusion Limited Aggregation

The η parameter has great influence on the growth of diffusion limited aggregation model. The reason is that η enlarges the associated weight for growth candidates with high concentrations. Take a look back at Equation

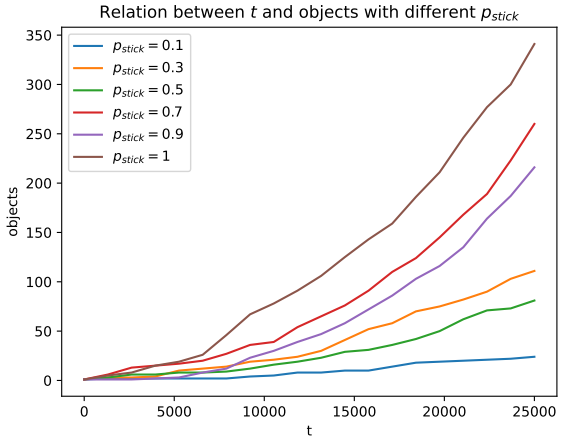

Fig. 6: Sticking probabilities from 0 to 1, fixed number of random walkers $t = 25000$, and grid size $N = 100 \times 100$.

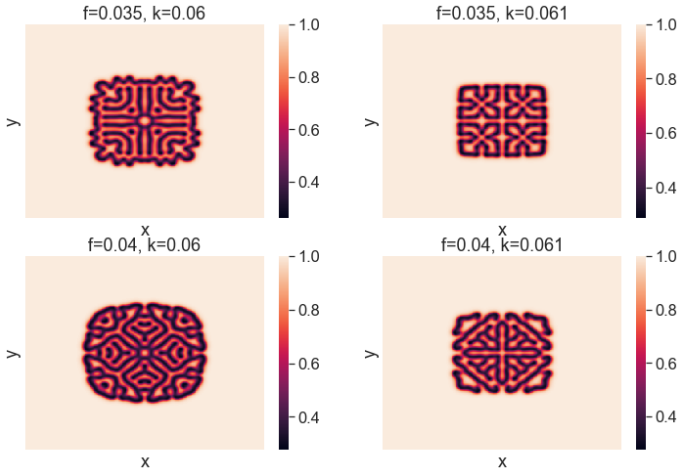

Fig. 7: Concentrations of U after 3000 iterations with different parameters. Grid size: 300×300

9, when $\eta = 0$, all the growth candidates will have the same associated weight, which means they have the same probability to be chosen, thus the development of DLA has no specific direction. If the η started to increase, the growth candidates with high concentrations will start to have high associated weights, making them more likely to be chosen. But this advantage is not determinant when η is smaller or equal to 1, since the grid size is large, the differences between concentrations on neighbouring lattice sites are relatively low, so the differences between associated weights for growth candidates are also low. However, If the η kept increasing after 1, the growth candidates with high concentrations will begin to play an determinant role in the development of DLA model, since their associated weights will be amplified by the large η, which makes them more and more likely to be

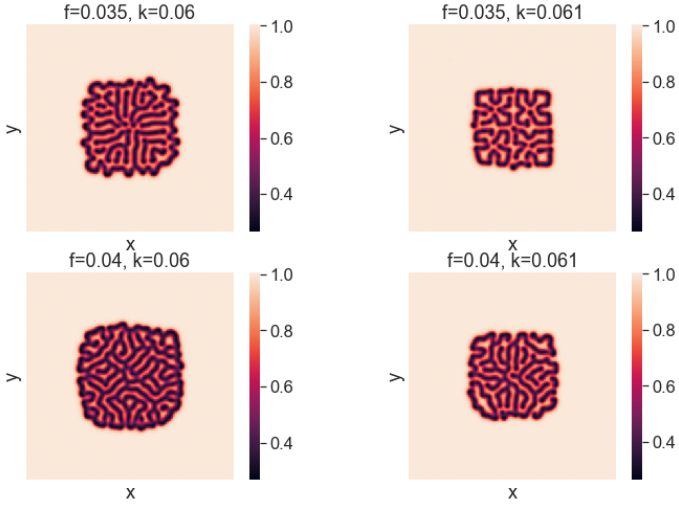

Fig. 8: Concentrations of U with noise after 3000 iterations with different parameters. Grid size: 300×300. Noise level $\sim N(0, 0.005)$

chosen. Therefore, With an increasing η, the development of DLA model tends to be a vertical straight line.

For the ω, it is not optimistic to set it to a specific value during the whole process of DLA modelling. The reason is that, in the computational domain, the DLA cluster has the same impact as a sink object. As discussed in the Set 1 report, the optimal ω value decreases with increasing sink area. Therefore, with the development of DLA cluster, the optimal ω also decreases. So the optimal way is to set the ω between **1.93** to **1.94**, which is the experimentally proved optimal value for a 100×100 grid, at the beginning of growth, then keep ω decreasing for each growth step following.

As discussed in the Chapter IV, the parallel SOR algorithm has a huge improvement compared to the iterative version. The drawback of parallel SOR is that adding object in the computational domain is quite troublesome, since the Red-Black blocks need to be reordered to fit the object.

B. Monte Carlo Simulation of DLA

When comparing the different results of two simulation methods, it can be found that the Monte Carlo Simulation with $p_{stick} = 1$ is quite similar with the DLA with $\eta = 1$. The bottom parts of both models are thin and top parts are diffusing towards up. However, for the general shapes of different clusters with different p_{stick}, it shows no obvious difference, which is not the same as the shape differences of DLA with various η.

C. The Gray-Scott Model

The development of Gray-Scott model is complicated, but in general, f controls the supply of U, thus a low f value will lead to an early steady state of the chemical reaction. $f + k$ controls the rate at which V decays, therefore, a high $f + k$ value will lead to a late steady state of the chemical process.

REFERENCES

[1] Thomas A Witten and Leonard M Sander. Diffusion-limited aggregation. *Physical review B*, 27(9):5686, 1983.

[2] P. Gray and S.K. Scott. Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Oscillations and instabilities in the system a + 2b $\rightarrow$ 3b; b $\rightarrow$ c. *Chemical Engineering Science*, 39(6):1087–1097, 1984.

[3] D.J. Evans. Parallel s.o.r. iterative methods. *Parallel Computing*, 1(1):3–18, 1984.

[4] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.