Shraddha Raghuram
5211972
ShraddhaRaghuram@student.tudelft.nl

Assignment 5: Numerical Solution of a Non-Linear Coupled Reaction-Diffusion Problem

2022-10-23

1 Schnakenberg Model

The Schnakenberg model suggests a physical mechanisms behind the emergence of Turing patterns in nature. The model shows that a chemical reaction between two substances, the 'slow' activator u and the 'fast' inhibitor v, leads to the emergence of regular patters from noise. Such reactions happen, for instance, in animal skins and lead to the characteristic appearance of cheetah's and zebra's. The process is described by the following system of non-linear coupled reaction-diffusion PDE's:

$$\frac{\partial u}{\partial t} = D_u \Delta u + k(a - u + u^2 v), \tag{1}$$

$$\frac{\partial v}{\partial t} = D_v \Delta v + k(b - u^2 v), \tag{2}$$

$$u(x,y) \in \Omega, t \in (0,T];$$

$$u(x,y,0) = u_0(x,y), v(x,y,0) = v_0(x,y), (x,y) \in \Omega; \tag{3}$$

$$-D_u \nabla u \cdot \mathbf{n} = 0, -D_v \nabla v \cdot n = 0, (x,y) \in \partial\Omega. \tag{4}$$

The rates of diffusion are determined by the corresponding diffusivity constants $D_u = 0.05$ and $D_v = 1.0$. The reaction constants are: k = 5, a = 0.1305 and b = 0.7695. The initial conditions are:

$$u_0(x,y) = a + b + r(x,y), v_0(x,y) = \frac{b}{(a+b)^2} \tag{5}$$

where r(x, y) is a small nonuniform perturbation in the concentration of the activator. All parameters are tuned here to the regime where a pattern is expected to appear.
The computational domain$\Omega = (0,4) \times (0,4)$ is a square. And the pattern should be almost completely formed at T = 20.

2 Theory

1. Discretize the problem in space on a doubly-uniform grid with the Finite-Difference Method.

   (a) For a doubly-uniform grid on a rectangular domain and lexicographic ordering of the unknowns, derive the symmetric matrix A of the negative 2D FD Laplacian with zero Neumann BC's.
   FD Laplacian can be approximated using the following equation:

$$\frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{\frac{16}{N_x^2}} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{\frac{16}{N_y^2}} = f_{i,j}$$

   Taking $N_x = 2$, and $Ny = 2$,
   for inner point,
   $i = 1, j = 1$,
   Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{0,1} + 16u_{1,1} - 4u_{2,1} - 4u_{1,0} - 4u_{1,2}) = f_{1,1}$$

   for boundary points,
   $i = 0, j = 0$,
   Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{-1,0} + 16u_{0,0} - 4u_{1,0} - 4u_{0,-1} - 4u_{0,1}) = f_{0,0}$$

$i = 0, j = 1,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{-1,1} + 16u_{0,1} - 4u_{1,1} - 4u_{0,0} - 4u_{0,2}) = f_{0,1}$$

$i = 0, j = 2,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{-1,2} + 16u_{0,2} - 4u_{1,2} - 4u_{0,1} - 4u_{0,3}) = f_{0,2}$$

$i = 1, j = 0,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{0,0} + 16u_{1,0} - 4u_{2,0} - 4u_{1,-1} - 4u_{1,1}) = f_{1,0}$$

$i = 2, j = 0,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{1,0} + 16u_{2,0} - 4u_{3,0} - 4u_{2,-1} - 4u_{2,1}) = f_{2,0}$$

$i = 1, j = 2,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{0,2} + 16u_{1,2} - 4u_{2,2} - 4u_{1,1} - 4u_{1,3}) = f_{1,2}$$

$i = 2, j = 1,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{1,1} + 16u_{2,1} - 4u_{3,1} - 4u_{2,0} - 4u_{2,2}) = f_{2,1}$$

$i = 2, j = 2,$
Simplification of the above equation yields:

$$\frac{1}{16}(-4u_{1,2} + 16u_{2,2} - 4u_{3,2} - 4u_{2,1} - 4u_{2,3}) = f_{2,2}$$

Using the boundary condition on the left finite difference derivative:

$$\frac{u_{0,j} - u_{-1,j}}{h_x} = 0$$
$$u_{0,j} = u_{-1,j}$$

Similarly,

$$u_{i,0} = u_{i,-1}$$

Using the boundary condition on the right finite difference derivative:

$$\frac{u_{3,j} - u_{2,j}}{h_x} = 0$$
$$u_{3,j} = u_{2,j}$$

Similarly,

$$u_{i,3} = u_{i,2}$$

Rewriting the above equations, we get:

$$\frac{1}{16}(-4u_{0,0} + 16u_{0,0} - 4u_{1,0} - 4u_{0,0} - 4u_{0,1}) = f_{0,0}$$
$$\frac{1}{16}(-4u_{0,1} + 16u_{0,1} - 4u_{1,1} - 4u_{0,0} - 4u_{0,2}) = f_{0,1}$$
$$\frac{1}{16}(-4u_{0,2} + 16u_{0,2} - 4u_{1,2} - 4u_{0,1} - 4u_{0,2}) = f_{0,2}$$

$$\frac{1}{16}(-4u_{0,0} + 16u_{1,0} - 4u_{2,0} - 4u_{1,0} - 4u_{1,1}) = f_{1,0}$$
$$\frac{1}{16}(-4u_{1,0} + 16u_{2,0} - 4u_{2,0} - 4u_{2,0} - 4u_{2,1}) = f_{2,0}$$
$$\frac{1}{16}(-4u_{0,2} + 16u_{1,2} - 4u_{2,2} - 4u_{1,1} - 4u_{1,2}) = f_{1,2}$$
$$\frac{1}{16}(-4u_{1,1} + 16u_{2,1} - 4u_{2,1} - 4u_{2,0} - 4u_{2,2}) = f_{2,1}$$
$$\frac{1}{16}(-4u_{1,2} + 16u_{2,2} - 4u_{2,2} - 4u_{2,1} - 4u_{2,2}) = f_{2,2}$$

and the inner point

$$\frac{1}{16}(-4u_{0,1} + 16u_{1,1} - 4u_{2,1} - 4u_{1,0} - 4u_{1,2}) = f_{1,1}$$

System Matrix for the inner and boundary points can be approximated as follows:

$$\begin{bmatrix} 1/2 & -1/4 & 0 & -1/4 & 0 & 0 & 0 & 0 & 0 \\ -1/4 & 3/4 & -1/4 & 0 & -1/4 & 0 & 0 & 0 & 0 \\ 0 & -1/4 & 1/2 & 0 & 0 & -1/4 & 0 & 0 & 0 \\ -1/4 & 0 & 0 & 3/4 & -1/4 & 0 & -1/4 & 0 & 0 \\ 0 & -1/4 & 0 & -1/4 & 1 & -1/4 & 0 & -1/4 & 0 \\ 0 & 0 & -1/4 & 0 & -1/4 & 3/4 & 0 & 0 & -1/4 \\ 0 & 0 & 0 & -1/4 & 0 & 0 & 1/2 & -1/4 & 0 \\ 0 & 0 & 0 & 0 & -1/4 & 0 & -1/4 & 3/4 & -1/4 \\ 0 & 0 & 0 & 0 & 0 & -1/4 & 0 & -1/4 & 1/2 \end{bmatrix}$$

(b) Find the matrices D_x and D_y, such that:

$$A_{xx} = D_x^T D_x, A_{yy} = D_y^T D_y,$$

$$A = I_y \otimes A_{xx} + A_{yy} \otimes I_x$$

By applying the trial and error method, we get

$$D_x = D_y = \begin{bmatrix} 0 & 0 & 0 \\ -1/2 & 1/2 & 0 \\ 0 & -1/2 & 1/2 \\ 0 & 0 & 0 \end{bmatrix}$$

(c) Let $\mathbf{u}(t)$ and $\mathbf{v}(t)$ be the lexicographic vectors of the grid values $u(x_i, y_j, t)$ and $v(x_i, y_j, t)$ at time t. After spatial discretization we obtain two coupled systems of ODE's:

$$\mathbf{u}' = \mathbf{f}_u(\mathbf{u}, \mathbf{v}), \mathbf{v}' = \mathbf{f}_v(\mathbf{u}, \mathbf{v}) \tag{6}$$

Give the explicit expression for the vector functions $\mathbf{f}_u$ and $\mathbf{f}_v$ in terms of $A, \mathbf{u}, \mathbf{v}$, etc. Rewrite (6) as a single system of ODE's.

$$\mathbf{w}' = \begin{bmatrix} \mathbf{u}' \\ \mathbf{v}' \end{bmatrix} = \begin{bmatrix} \mathbf{f}_u \\ \mathbf{f}_v \end{bmatrix} = \begin{bmatrix} D_{\mathbf{u}}\Delta u + k(a - u + u^2 v) \\ D_{\mathbf{v}}\Delta v + k(b - u^2 v) \end{bmatrix}$$

2. Numerical time-integration.

(a) Write down the iteration formula of the Forward-Euler (FE) Method for the coupled systems (6) or for the single system of ODE's that you have derived.
The Forward-Euler time-integration method can be applied to any non-linear system of ODE's of the form $\mathbf{u}' = f(\mathbf{u}, t)$. FE iterations are:

$$\mathbf{u}(t_{k+1}) = \mathbf{u}(t_k) + h_t \mathbf{f}(t_k, \mathbf{u}(t_k)), k = 0, 1, ...$$
$$\mathbf{u}^{k+1} = \mathbf{u}^k + h_t \mathbf{f}^k(\mathbf{u}^k), k = 0, 1, ...$$

For FE it does not matter if $\mathbf{f}^k(\mathbf{u}^k)$ is a nonlinear function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ but we know that FE may become unstable and requires sufficiently small time step h_t.

For the system of ODE's that I have derived:

$$\mathbf{u}(t_{k+1}) = \mathbf{u}(t_k) + h_t\mathbf{f}(t_k, \mathbf{u}(t_k), \mathbf{v}(t_k)), k = 0, 1, ...$$
$$\mathbf{u}^{k+1} = \mathbf{u}^k + h_t\mathbf{f}^k(\mathbf{u}^k, \mathbf{v}^k), k = 0, 1, ...$$
$$\mathbf{v}^{k+1} = \mathbf{v}^k + h_t\mathbf{f}^k(\mathbf{u}^k, \mathbf{v}^k), k = 0, 1, ...$$

(b) To derive an approximate stability condition, consider a simpler linear problem:

$$\frac{\partial u}{\partial t} = \Delta u - ku, k > 0, \tag{7}$$

with zero Dirichlet BC's. What is the FE method's stability condition on the number of time sub-intervals N_t for this problem?

$$\mathbf{u}(t_k) = \mathbf{u}(t_{k-1}) + h_t(-A\mathbf{u}(t_{k-1}) - k\mathbf{u}(t_{k-1})) = \mathbf{u}_h(t_{k-1})[I - Ah_t - kh_t] = [I - Ah_t - kh_t]^k\mathbf{u}_0$$

Let A be diagonizable: $A = V\Lambda V^{-1}$ with eigenvalues $\lambda_j, j = 1, ..., n$

$$\mathbf{u}_h(t_k) = V[I - h\Lambda - khI]^kV^{-1}\mathbf{u}_0 = \sum_{j=1}^{n} b_j(0)(1 - h\lambda_j - kh)^k\mathbf{v}_j$$

Hence, for $\lambda_j > 0, e^{-\lambda_j t_k}$ always decays with t_k but $(1 - h\lambda_j - kh)^k$ may be growing with k (from the power) if h is too lrge. The FE method is not always stable.
Conditional stability of the Forward Euler Method:
Deriving the condition for the stability of the Forward-Euler Method:

$$-1 < 1 - h\lambda_j - kh < 1$$
$$-2 < -h\lambda_j - kh < 0$$
$$0 < h(\lambda_j + k) < 2$$
$$0 < h < \frac{2}{\lambda_j + k}$$
$$0 < \Delta t < \frac{2}{max_j(\lambda_j) + k}$$

(c) Write down the iteration formula of the Backward-Euler Method for the coupled system (6) or for the single system that you have derived.
Consider the Backward-Euler method applied to $\mathbf{u}' = \mathbf{f}(\mathbf{u}, t)$:

$$\mathbf{u}(t_{k+1}) = \mathbf{u}(t_k) + h_t\mathbf{f}(t_{k+1}, \mathbf{u}(t_{k+1})), k = 0, 1, ...$$
$$\mathbf{u}^{k+1} = \mathbf{u}^k + h_t\mathbf{f}^{k+1}(\mathbf{u}^{k+1})$$

If $\mathbf{f}^k(\mathbf{u}^k)$ is a nonlinear function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, then to find $\mathbf{u}^{k+1}$ we need to solve the nonlinear problem

$$\mathbf{u}^{k+1} = h_t\mathbf{f}^{k+1}(\mathbf{u}^{k+1}) + \mathbf{u}^k$$

at each time step k. Here $\mathbf{u}^k$ is a known vector and $\mathbf{f}^{k+1}(....)$ is a known function.

$$\mathbf{u}^{k+1} = \mathbf{u}^k + h_t\mathbf{f}_u^{k+1}(\mathbf{u}^{k+1}, \mathbf{v}^{k+1})$$
$$\mathbf{v}^{k+1} = \mathbf{v}^k + h_t\mathbf{f}_v^{k+1}(\mathbf{u}^{k+1}, \mathbf{v}^{k+1})$$

3. Newton-Raphson algorithm.

(a) Write down the nonlinear problem that you need to solve at each step of the Backward-Euler algorithm. This is easier to do in the single-system formulation derived in 1(c). Identify the residual that will be minimized by the Newton-Raphson algorithm at a given time step. Write down the (inner) iteration formula of the Newton-Raphson algorithm.
The non-linear problem that we need to solve at each step of the Backward-Euler algorithm is as

follows:

$$\mathbf{w}' = \begin{bmatrix} \mathbf{u}' \\ \mathbf{v}' \end{bmatrix} = \begin{bmatrix} \mathbf{f}_u \\ \mathbf{f}_v \end{bmatrix} = \begin{bmatrix} -D_{\mathbf{u}}Au + k(a - u + u^2 v) \\ -D_{\mathbf{v}}Av + k(b - u^2 v) \end{bmatrix}$$

The residual that will be minimized at every time step is:

$$\mathbf{u}^k + h_t \mathbf{f}_u(\mathbf{u}^{k+1}) - \mathbf{u}^{k+1}$$

$$\mathbf{v}^k + h_t \mathbf{f}_v(\mathbf{v}^{k+1}) - \mathbf{v}^{k+1}$$

The inner iteration formula of the Newton-Raphson algorithm is as follows:

$$\mathbf{u}_{i+1}^{k+1} = u_i^{k+1} + [I - h_t J(\mathbf{u}_i^{k+1})]^{-1}[\mathbf{u}^k + h_t \mathbf{f}(\mathbf{u}_i^{k+1}) - \mathbf{u}_i^{k+1}]$$

$$\mathbf{v}_{i+1}^{k+1} = v_i^{k+1} + [I - h_t J(\mathbf{v}_i^{k+1})]^{-1}[\mathbf{v}^k + h_t \mathbf{f}(\mathbf{v}_i^{k+1}) - \mathbf{v}_i^{k+1}]$$

(b) Compute the Jacobian matrix. Hint: use the single-system formulation, the Jacobian matrix will have four blocks:

$$J(\mathbf{u}, \mathbf{v}) = \begin{bmatrix} \frac{\partial \mathbf{f}_u(\mathbf{u},\mathbf{v})}{\partial \mathbf{u}} & \frac{\partial \mathbf{f}_u(\mathbf{u},\mathbf{v})}{\partial \mathbf{v}} \\ \frac{\partial \mathbf{f}_v(\mathbf{u},\mathbf{v})}{\partial \mathbf{u}} & \frac{\partial \mathbf{f}_v(\mathbf{u},\mathbf{v})}{\partial \mathbf{v}} \end{bmatrix} \tag{8}$$

Provide explicit expressions for all blocks of the matrix J.

$$\frac{\partial \mathbf{f}_u(\mathbf{u}, \mathbf{v})}{\partial \mathbf{u}} = -D_{\mathbf{u}}A - k + 2k\mathbf{u}\mathbf{v}$$

$$\frac{\partial \mathbf{f}_u(\mathbf{u}, \mathbf{v})}{\partial \mathbf{v}} = \mathbf{u}^2 k$$

$$\frac{\partial \mathbf{f}_v(\mathbf{u}, \mathbf{v})}{\partial \mathbf{u}} = -2k\mathbf{u}\mathbf{v}$$

$$\frac{\partial \mathbf{f}_v(\mathbf{u}, \mathbf{v})}{\partial \mathbf{v}} = -AD_{\mathbf{v}} - k\mathbf{u}^2$$

Therefore, the Jacobian is as follows:

$$\begin{bmatrix} -D_{\mathbf{u}}A - k + 2k\mathbf{u}\mathbf{v} & \mathbf{u}^2 k \\ -2k\mathbf{u}\mathbf{v} & -AD_{\mathbf{v}} - k\mathbf{u}^2 \end{bmatrix}$$

3 Implementation

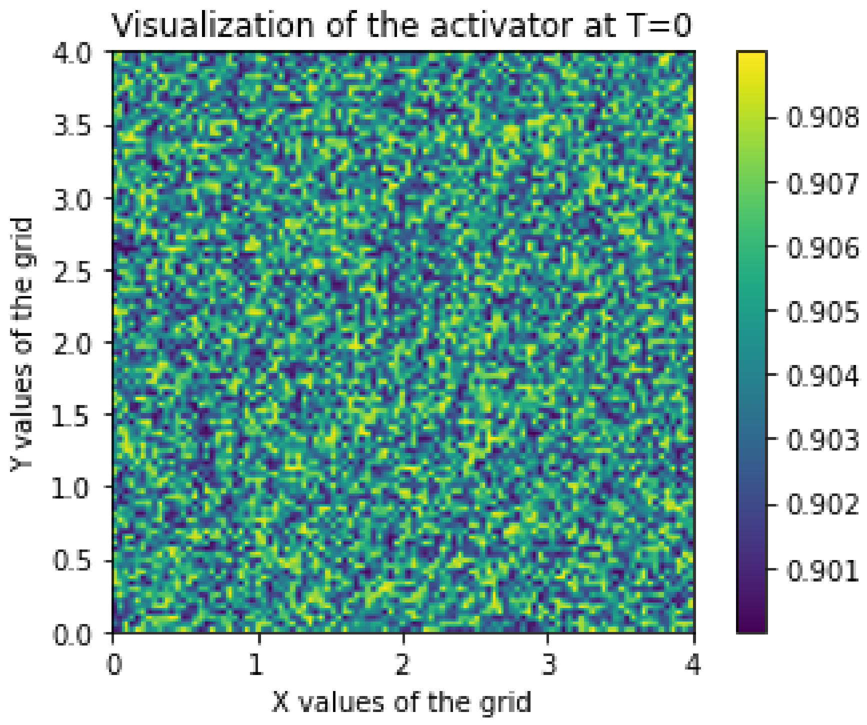
All results should be presented for $N_x = N_y = 100$. The inhomogeneous perturbation r(x, y) in (5) should be obtained as 0.01*(a+b)*np.random.rand(...). Use the same initial guess with the FE and BENR Methods, i.e., do not generate a new random perturbation when running the second method. Both methods must be implemented in the same file.

4. Programming the Forward-Euler (FE) Method. With the FE Method it is possible to work in terms of the coupled systems (6) and vectors $\mathbf{u}$ and $\mathbf{v}$. Your code and report will be evaluated based on the following:

   (a) The report should explain your empirical choice of a stable N_t. For finding the stability regime it is recommended to use the animation of the inhibitor v(x, y, t), similar to the one of Assignment 5, and perform many tests with a modest grid $N_x = N_y = 25$ and different values of k, e.g. k = 2, 5, 10.
   Done in Jupyter Notebook.
   The empirical choice of a stable N_t is defined by the stability condition defined in 2. (b). Values above this Nt tend to give an overflow error.

   (b) The final code should contain one test run of a stable FE Method and print out Nt the CPU time.
   Done in Jupyter Notebook.

   (c) The report should contain the images of the activator u(x, y, t) and inhibitor v(x, y, t) at t = 0 and t = T , obtained with the FE Method.
   Done in Jupyter Notebook.



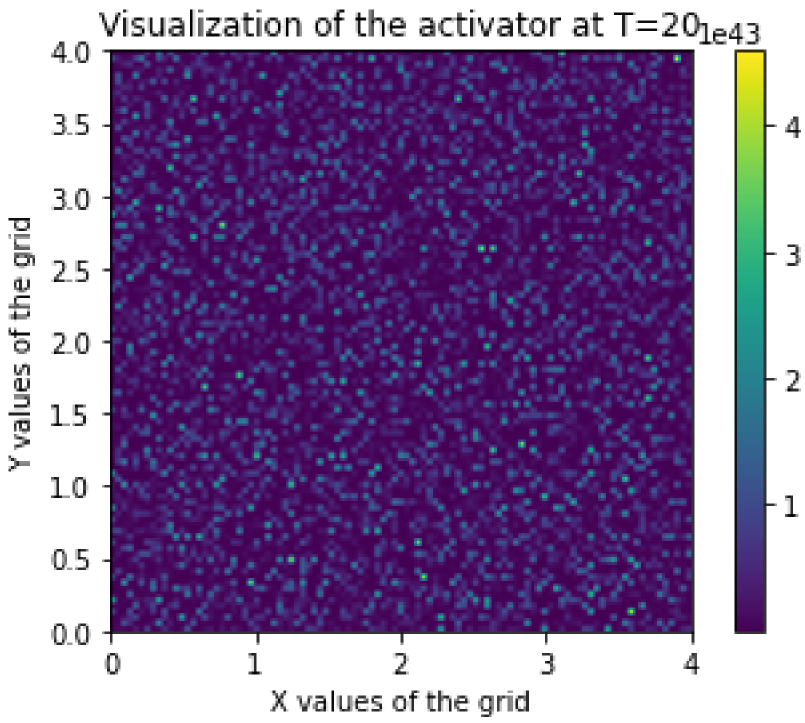
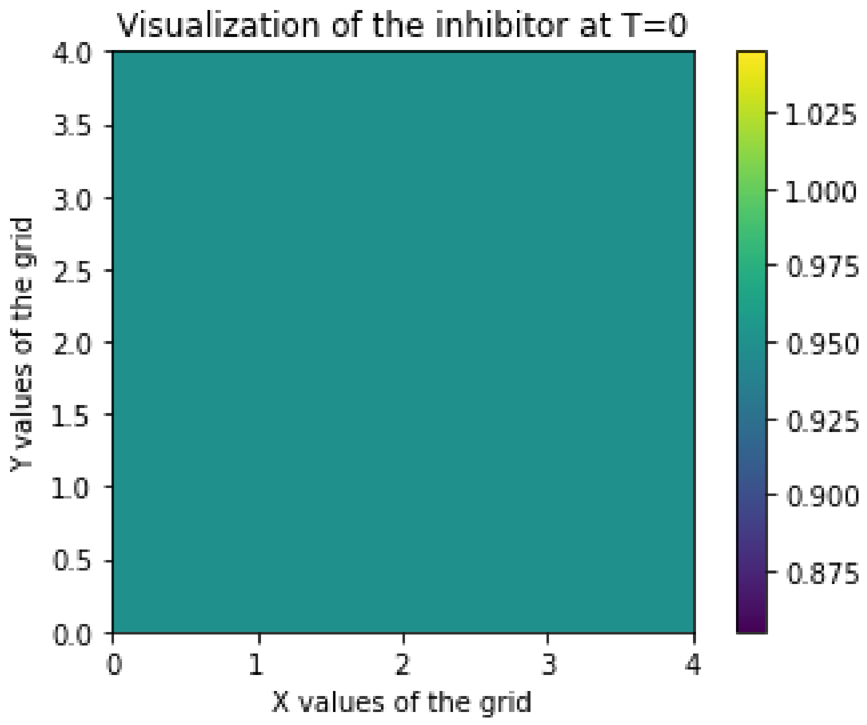


Figure 1: Plots from Matplotlib for 4. (c)



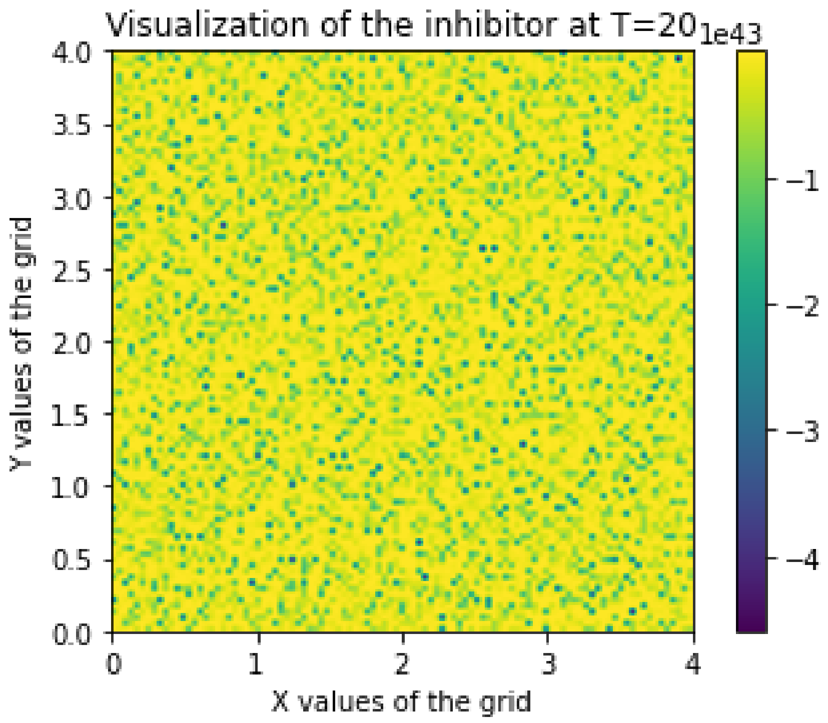


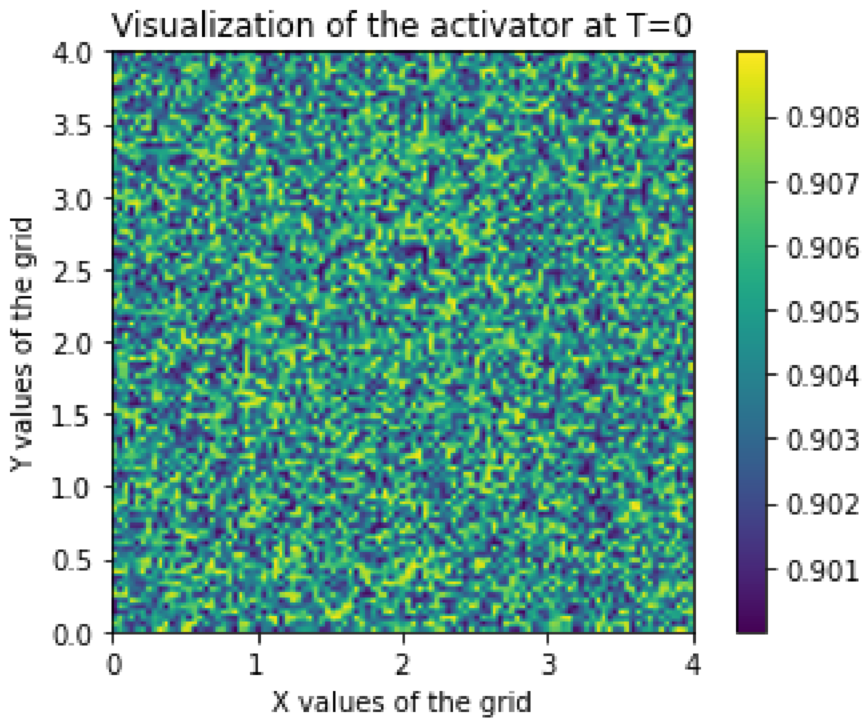
Figure 2: Plots from Matplotlib for 4. (c)

5. Programming the Backward-Euler-Newton-Raphson (BENR) Method.
   In the inner iterations of the BENR Method you will have to work with the single-system formulation. To joint the two vectors u and v into a single vector, you may use, e.g., the numpy.concatenate() function. To construct the sparse Jacobian matrix from its four blocks, you may use the scipy.sparse.bmat() function. The tolerance of inner iterations, i.e., the desired norm of the residual below which inner iterations are stopped and the code proceeds to the next outer iteration, may be set to 10^{-3}.

   (a) The report should explain your empirical choice of Nt for the BENR Method. It is recommended to let your code print out the norm of the residual at each inner iteration and observe the rate of convergence of the Newton-Raphson iterations and its dependence on N_t. Investigate the possible dependence on k as well.
   Done in Jupyter Notebook. Higher the N_t, slower the convergence. However, since there is no overflow error in this algorithm, we can take higher values of Nt as the solution for higher Nt appears more stable. For very low or very high k, the solution converges slowly.

   (b) The final code should contain one test run of the BENR Method with an automatically chosen Nt and print out Nt, the norm of the residual at each inner iteration, and the total CPU time.‘
   Done in Jupyter Notebook.

   (c) The report should contain the images of the activator u(x, y, t) and inhibitor v(x, y, t) at t = T , obtained with the BENR Method for the same initial guess as the FE Method.
   Done in Jupyter Notebook.



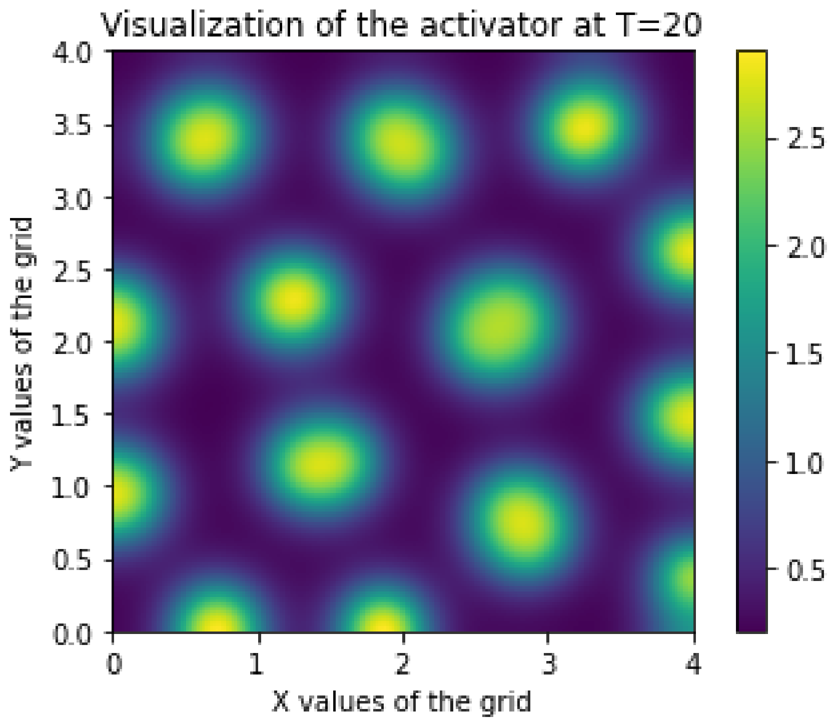
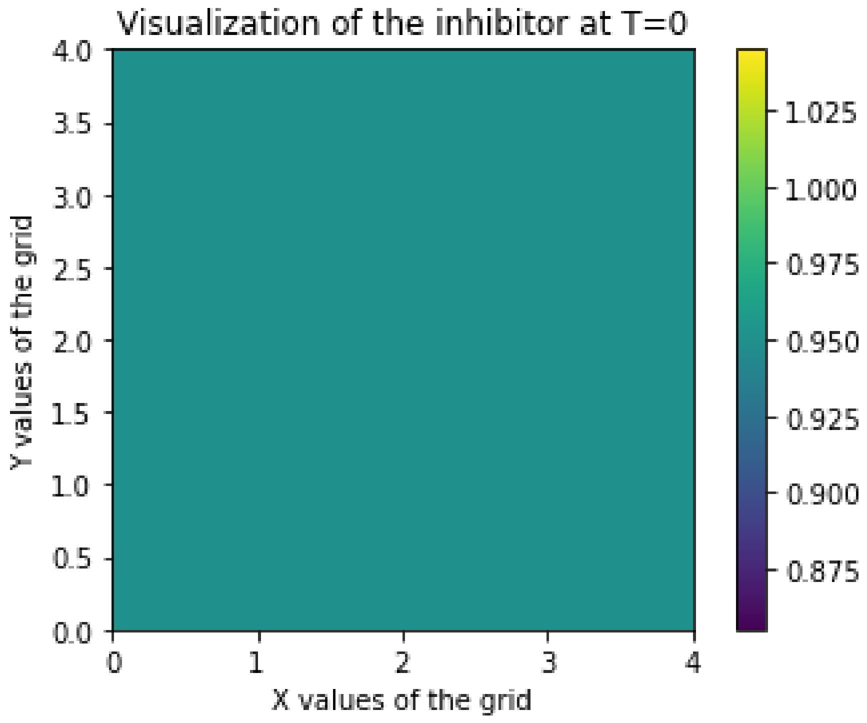


Figure 3: Plots from Matplotlib for 5. (c)



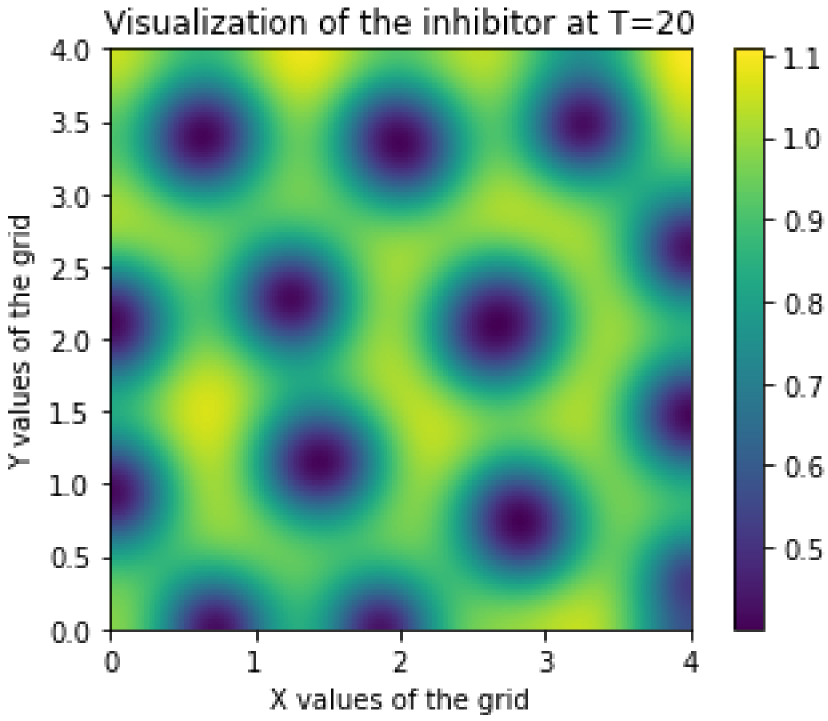


Figure 4: Plots from Matplotlib for 5. (c)