Title: Report on steady-state and transient analysis of a diffusion-reaction process

Author: Slava Ermolaev

Date: 12/12/2022

Table of Contents

1. Introduction

1.1. Problem

The goal of this project is to study the steady-state and transient characteristics of the presented set of chemical reactions:

$$A + 2B \rightarrow^{k_1} D$$
$$A \rightarrow^{k_2} P$$
$$B \rightarrow^{k_3} U$$
$$U \rightarrow^{k_4} F$$

Where A, B, D, P, U, and F are chemical species and k_1, k_2, k_3, and k_4 are chemical reaction rates. The reactions take place in the following membrane reactor:

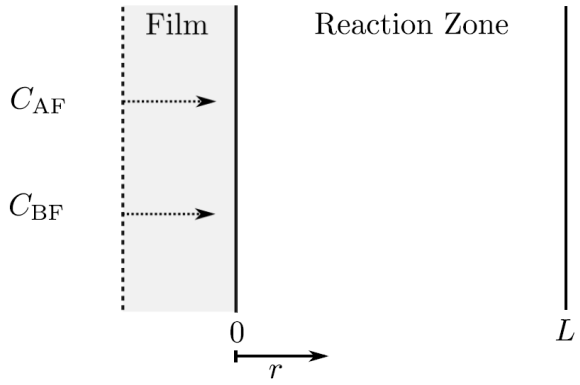

Figure 1: Schematic of membrane reactor

Where C_{AF} and C_{BF} are fixed concentrations of reactants A and B. Fixed concentrations of reactants enter the reaction zone through the left film membrane. Once the reactants enter the reaction zone at r = 0, they begin to react. It is assumed that the molecular diffusion through the reaction zone is one-dimensional in the horizontal r direction and follows Fick's law. The species deposit at the right side of the reaction zone at r = L with different deposition rates. The diffusion coefficients are constant and equal: $D_A = D_B = D_U = D_F = \bar{D}$. The following partial differential equations describe the change in species concentrations with position r and time t:

$$\frac{\partial C_A}{\partial t} = \bar{D}\frac{\partial^2 C_A}{\partial r^2} - k_1 C_A C_B^2 - k_2 C_A \quad (1)$$

$$\frac{\partial C_B}{\partial t} = \bar{D}\frac{\partial^2 C_B}{\partial r^2} - 2k_1 C_A C_B^2 - k_3 C_B \quad (2)$$

$$\frac{\partial C_U}{\partial t} = \bar{D}\frac{\partial^2 C_U}{\partial r^2} + k_3 C_B - k_4 C_U \quad (3)$$

$$\frac{\partial C_F}{\partial t} = \bar{D}\frac{\partial^2 C_F}{\partial r^2} + k_4 C_U \quad (4)$$

Where:

- C_A – species A concentration
- C_B – species B concentration

- C_U – species U concentration
- C_F – species F concentration
- k_1 – first reaction rate constant
- k_2 – second reaction rate constant
- k_3 – third reaction rate constant
- k_4 – fourth reaction rate constant
- $\bar{D}$ – diffusion coefficient
- r – distance
- t – time

1.2. Dimensionless Model

The following dimensionless variables and parameters were defined:

$$x = \frac{r}{L};\ y_A = \frac{C_A}{C_{AF}};\ y_B = \frac{C_B}{C_{AF}};\ y_U = \frac{C_U}{C_{AF}};\ y_F = \frac{C_F}{C_{AF}};\ \beta = \frac{C_{BF}}{C_{AF}}$$

$$D = \frac{\bar{D}}{k_1 C_{AF}^2 L^2}, \qquad \tau = k_1 C_{AF}^2 t, \qquad \gamma = \frac{k_2}{k_1 C_{AF}^2}, \qquad \delta = \frac{k_3}{k_1 C_{AF}^2}, \qquad \zeta = \frac{k_4}{k_1 C_{AF}^2}$$

Using defined dimensionless variables and parameters, equations 1-4 can be rewritten as:

$$\frac{\partial y_A}{\partial \tau} = D\frac{\partial^2 y_A}{\partial x^2} - y_A y_B^2 - \gamma y_A \tag{5}$$

$$\frac{\partial y_B}{\partial \tau} = D\frac{\partial^2 y_B}{\partial x^2} - 2y_A y_B^2 - \delta y_B \tag{6}$$

$$\frac{\partial y_U}{\partial \tau} = D\frac{\partial^2 y_U}{\partial x^2} + \delta y_B - \zeta y_U \tag{7}$$

$$\frac{\partial y_F}{\partial \tau} = D\frac{\partial^2 y_F}{\partial x^2} + \zeta y_U \tag{8}$$

The boundary conditions are the following:

$$at\ x = 0 \qquad\qquad at\ x = 1,$$

$$\frac{\partial y_A}{\partial x}(0,t) = -(1 - y_A), \qquad \frac{\partial y_A}{\partial x}(1,t) = \epsilon y_A$$

$$\frac{\partial y_B}{\partial x}(0,t) = -(\beta - y_B), \qquad \frac{\partial y_B}{\partial x}(1,t) = \eta y_B^2$$

$$y_U(0,t) = 0, \qquad \frac{\partial y_U}{\partial x}(1,t) = -\theta y_U$$

$$y_F(0,t) = 0, \qquad \frac{\partial y_F}{\partial x}(1,t) = 0$$

1.3. Methods to Obtain Steady-State Solution

For steady state solution, all derivatives with respect to τ become 0. Therefore, equations 5-8 become coupled homogeneous ordinary differential equations. This type of system can be solved by employing centered finite difference method. The domain was divided into N nodes and numbered from i = 1 to i = N. The length of the dimensionless domain is L = 1 (from x = 0 to x = 1). The distance between nodes, h, was calculated with the following formula:

$$h = \frac{L - 0}{N - 1} = \frac{1}{N - 1} \tag{9}$$

Species that have derivative boundary conditions must have fictitious nodes on that side. Species A has 2 fictitious nodes, 1 on each side. Species B has 2 fictitious nodes, 1 on each side. Species U has 1 fictitious node for i = N + 1. Species F has 1 fictitious node for i = N + 1.

The following convention will be employed to signify the value of the species: $y_X^{(i)}$, where X signifies the species and can be A, B, U, and F, and i signifies the node number.

For species A, the discretized form of equation 5 for the general inner node i is the following:

$$f_i = D\left(\frac{y_A^{(i+1)} - 2y_A^{(i)} + y_A^{(i-1)}}{h^2}\right) - y_A^{(i)} y_B^{(i)2} - \gamma y_A^{(i)} \tag{10}$$

Species A has 2 fictitious nodes, at i = 0, the derivative boundary condition gives the following relation:

$$y_A^{(0)} = y_A^{(2)} + 2h\left(1 - y_A^{(1)}\right) \tag{11}$$

Therefore, for i = 1 and applying relation in equation 11, equation 10 becomes:

$$f_1 = \mathrm{D}\left(\frac{y_A^{(2)} - 2y_A^{(1)} + y_A^{(2)} + 2h\left(1 - y_A^{(1)}\right)}{h^2}\right) - y_A^{(1)} y_B^{(1)2} - \gamma y_A^{(1)} \tag{12}$$

Similarly, the following relation for i = N + 1 is derived using the derivative boundary condition at the right side:

$$y_A^{(N+1)} = y_A^{(N-1)} + 2h\epsilon y_A^{(N)} \tag{13}$$

Applying equation 13 to equation 10 for i = N, the following function appears:

$$f_N = \mathrm{D}\left(\frac{y_A^{(N-1)} - 2h\epsilon y_A^{(N)} - 2y_A^{(N)} + y_A^{(N-1)}}{h^2}\right) - y_A^{(N)} y_B^{(N)2} - \gamma y_A^{(N)} \tag{14}$$

Therefore, for species A there are N discretized equations from i = 1 to i = N. For i = 1, equation 12 applies, for i = N, equation 14 applies, and for all other i's, general equation 10 applies.

Similarly for species B, there will be 2 fictitious nodes. Both boundary conditions give the following relations:

$$y_B^{(0)} = y_B^{(2)} + 2h\left(\beta - y_B^{(1)}\right) \tag{15}$$

$$y_B^{(N+1)} = y_B^{(N-1)} - 2h\eta y_B^{(N)2} \tag{16}$$

Applying equations 15 & 16 and discretizing equation 6, the following functions result:

$$f_1 = D\left(\frac{y_B^{(2)} - 2y_B^{(1)} + y_B^{(2)} + 2h\left(\beta - y_B^{(1)}\right)}{h^2}\right) - 2y_A^{(1)}y_B^{(1)2} - \delta y_B^{(1)} \tag{17}$$

$$f_N = D\left(\frac{y_B^{(N-1)} - 2h\eta\, y_B^{(N)2} - 2y_B^{(N)} + y_B^{(N-1)}}{h^2}\right) - 2y_A^{(N)}y_B^{(N)2} - \delta y_B^{(N)} \tag{18}$$

$$f_i = D\left(\frac{y_B^{(i+1)} - 2y_B^{(i)} + y_B^{(i-1)}}{h^2}\right) - 2y_A^{(i)}y_B^{(i)2} - \delta y_B^{(i)} \tag{19}$$

Similarly, f_1 and f_N apply to nodes i = 1 and i = N, respectively, and f_i applies to all other nodes.

For species U there will be 1 fictitious node at i = N + 1. The following relation is obtained from the derivative boundary condition:

$$y_U^{(N+1)} = y_U^{(N-1)} - 2h\theta y_U^{(N)} \tag{20}$$

According to the left boundary condition, $y_U^{(1)} = 0$.

Applying equation 20 and discretizing equation 7, the following functions result:

$$f_N = D\left(\frac{y_U^{(N-1)} - 2h\theta y_U^{(N)} - 2y_U^{(N)} + y_U^{(N-1)}}{h^2}\right) + \delta y_B^{(N)} - \zeta y_U^{(N)} \tag{21}$$

$$f_i = D\left(\frac{y_U^{(i+1)} - 2y_U^{(i)} + y_U^{(i-1)}}{h^2}\right) + \delta y_B^{(i)} - \zeta y_U^{(i)} \tag{22}$$

Similarly, f_N applies to node i = N, and f_i applies to all other nodes from i = 2 to i = N – 1.

For species F there will be 1 fictitious node at i = N + 1. The following relation is obtained from the derivative boundary condition:

$$y_F^{(N+1)} = y_F^{(N-1)} \tag{23}$$

According to the left boundary condition, $y_F^{(1)} = 0$.

Applying equation 23 and discretizing equation 8, the following functions result:

$$f_N = D\left(\frac{y_F^{(N-1)} - 2y_F^{(N)} + y_F^{(N-1)}}{h^2}\right) + \zeta y_U^{(N)} \tag{24}$$

$$f_i = D\left(\frac{y_F^{(i+1)} - 2y_F^{(i)} + y_F^{(i-1)}}{h^2}\right) + \zeta y_U^{(i)} \tag{25}$$

f_N applies to node i = N, and f_i applies to all other nodes from i = 2 to i = N – 1.

All the functions for all species will be set equal to zero and assembled in order into solution vector F. The following convention is used $f_X^{(i)}$, where X is species, and i is the node number. The following vector F results:

$$\begin{matrix} f_A^{(1)} \\ f_A^{(2)} \\ \vdots \\ f_A^{(N)} \\ f_B^{(1)} \\ \vdots \\ f_B^{(N)} \\ f_U^{(2)} \\ \vdots \\ f_U^{(N)} \\ f_F^{(2)} \\ \vdots \\ f_F^{(N)} \end{matrix} \tag{26}$$

Jacobian is assembled using discretized functions:

$$
\begin{matrix}
\frac{\partial f_A^{(1)}}{\partial y_A^{(1)}} & \frac{\partial f_A^{(1)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_A^{(1)}}{\partial y_A^{(N)}} & \frac{\partial f_A^{(1)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_A^{(1)}}{\partial y_B^{(N)}} & \frac{\partial f_A^{(1)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_A^{(1)}}{\partial y_U^{(N)}} & \frac{\partial f_A^{(1)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_A^{(1)}}{\partial y_F^{(N)}} \\
\frac{\partial f_A^{(2)}}{\partial y_A^{(1)}} & \frac{\partial f_A^{(2)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_A^{(2)}}{\partial y_A^{(N)}} & \frac{\partial f_A^{(2)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_A^{(2)}}{\partial y_B^{(N)}} & \frac{\partial f_A^{(2)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_A^{(2)}}{\partial y_U^{(N)}} & \frac{\partial f_A^{(2)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_A^{(2)}}{\partial y_F^{(N)}} \\
 & & & & & \vdots & & & & & & & \\
\frac{\partial f_A^{(N)}}{\partial y_A^{(1)}} & \frac{\partial f_A^{(N)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_A^{(N)}}{\partial y_A^{(N)}} & \frac{\partial f_A^{(N)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_A^{(N)}}{\partial y_B^{(N)}} & \frac{\partial f_A^{(N)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_A^{(N)}}{\partial y_U^{(N)}} & \frac{\partial f_A^{(N)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_A^{(N)}}{\partial y_F^{(N)}} \\
\frac{\partial f_B^{(1)}}{\partial y_A^{(1)}} & \frac{\partial f_B^{(1)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_B^{(1)}}{\partial y_A^{(N)}} & \frac{\partial f_B^{(1)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_B^{(1)}}{\partial y_B^{(N)}} & \frac{\partial f_B^{(1)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_B^{(1)}}{\partial y_U^{(N)}} & \frac{\partial f_B^{(1)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_B^{(1)}}{\partial y_F^{(N)}} \\
 & & & & & \vdots & & & & & & & \\
\frac{\partial f_B^{(N)}}{\partial y_A^{(1)}} & \frac{\partial f_B^{(N)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_B^{(N)}}{\partial y_A^{(N)}} & \frac{\partial f_B^{(N)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_B^{(N)}}{\partial y_B^{(N)}} & \frac{\partial f_B^{(N)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_B^{(N)}}{\partial y_U^{(N)}} & \frac{\partial f_B^{(N)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_B^{(N)}}{\partial y_F^{(N)}} \\
\frac{\partial f_U^{(2)}}{\partial y_A^{(1)}} & \frac{\partial f_U^{(2)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_U^{(2)}}{\partial y_A^{(N)}} & \frac{\partial f_U^{(2)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_U^{(2)}}{\partial y_B^{(N)}} & \frac{\partial f_U^{(2)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_U^{(2)}}{\partial y_U^{(N)}} & \frac{\partial f_U^{(2)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_U^{(2)}}{\partial y_F^{(N)}} \\
 & & & & & \vdots & & & & & & & \\
\frac{\partial f_U^{(N)}}{\partial y_A^{(1)}} & \frac{\partial f_U^{(N)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_U^{(N)}}{\partial y_A^{(N)}} & \frac{\partial f_U^{(N)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_U^{(N)}}{\partial y_B^{(N)}} & \frac{\partial f_U^{(N)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_U^{(N)}}{\partial y_U^{(N)}} & \frac{\partial f_U^{(N)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_U^{(N)}}{\partial y_F^{(N)}} \\
\frac{\partial f_F^{(2)}}{\partial y_A^{(1)}} & \frac{\partial f_F^{(2)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_F^{(2)}}{\partial y_A^{(N)}} & \frac{\partial f_F^{(2)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_F^{(2)}}{\partial y_B^{(N)}} & \frac{\partial f_F^{(2)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_F^{(2)}}{\partial y_U^{(N)}} & \frac{\partial f_F^{(2)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_F^{(2)}}{\partial y_F^{(N)}} \\
 & & & & & \vdots & & & & & & & \\
\frac{\partial f_F^{(N)}}{\partial y_A^{(1)}} & \frac{\partial f_F^{(N)}}{\partial y_A^{(2)}} & \cdots & \frac{\partial f_F^{(N)}}{\partial y_A^{(N)}} & \frac{\partial f_F^{(N)}}{\partial y_B^{(1)}} & \cdots & \frac{\partial f_F^{(N)}}{\partial y_B^{(N)}} & \frac{\partial f_F^{(N)}}{\partial y_U^{(2)}} & \cdots & \frac{\partial f_F^{(N)}}{\partial y_U^{(N)}} & \frac{\partial f_F^{(N)}}{\partial y_F^{(2)}} & \cdots & \frac{\partial f_F^{(N)}}{\partial y_F^{(N)}}
\end{matrix}
$$

Figure 2: Jacobian structure

The following convention is used for vector Y elements: $y_X^{(i)}$, where X is species and i is the node number. The following vector Y is assembled:

$$\begin{matrix} y_A^{(1)} \\ y_A^{(2)} \\ \vdots \\ y_A^{(N)} \\ y_B^{(1)} \\ \vdots \\ y_B^{(N)} \\ y_U^{(2)} \\ \vdots \\ y_U^{(N)} \\ y_F^{(2)} \\ \vdots \\ y_F^{(N)} \end{matrix} \tag{27}$$

Multivariable Newton is used to calculate the solution Y. $Y^{(M)}$ represents the previous iteration vector, and $Y^{(M+1)}$ represents the next iteration vector. Newton's method must be started with an initial guess $Y^{(0)}$ that must be close to the solution. The difference between two successive iterations is defined as:

$$\delta Y = Y^{(M+1)} - Y^{(M)} \tag{28}$$

The following system results:

$$J\delta Y = -F \tag{29}$$

Because Jacobian is a band matrix, the system is most efficiently solved using LU decomposition. This system was solved using LUSolver program developed in Homework #2 that performs LU decomposition with partial pivoting.

The Newton's method was iterated until the square of vector 2-norm $\left|\left|\delta Y\right|\right|_2^2 < 10^{(-2)}$.

1.4. Methods to Obtain Transient Solution

To solve transient solution of equations 5-8, explicit Euler method was attempted, but proved to be unsuccessful after results diverged. Implicit Euler was chosen thereafter because it is unconditionally stable.

Implicit Euler is defined as follows:

$$Y^{(M+1)} = Y^{(M)} + F^{(M+1)}dt \tag{30}$$

Where:

- $Y^{(M+1)}$ – vector Y in equation 27 for M + 1 iteration

- $Y^{(M)}$ – vector Y in equation 27 for M iteration
- $F^{(M+1)}$ – function F in equation 26 that uses y values at M + 1 iteration.
- dt – time step

$F^{(M+1)}$ solution vector consists of discretized functions f that use unknown values of y's at iteration M + 1. Therefore, the equation 30 can be rearranged as follows:

$$Y^{(M)} - Y^{(M+1)} + F^{(M+1)}dt = 0 \tag{31}$$

The equation 31 can be represented as an ODE system and can be solved using multivariable Newton's method as illustrated in the steady-state solution section 1.3. The Jacobian will be assembled in a similar manner as in figure 2. For example, for diagonal elements, the same expression as in figure 2 will be multiplied by dt and 1 will be subtracted.

2. Results and Discussion

2.1. Results

2.1.1. Question 1: Steady-state solution results

For all steady-state solutions in parts a-c, the number of nodal points that gives the accurate solution was determined to be N = 41.

For part (a), the following parameters were set:

$$\boldsymbol{\delta = 0}, \quad \boldsymbol{\zeta = 0}, \quad \boldsymbol{D = 0.1}, \quad \boldsymbol{\beta = 1.5},$$
$$\boldsymbol{\gamma = 0.05}, \quad \boldsymbol{\epsilon = 0.0}, \quad \boldsymbol{\eta = 0.0}, \quad \boldsymbol{\theta = 0.0}$$

All elements of each species were set to zeros as the initial guess for Newton's method.

For part (b), the following parameters were set:

$$\boldsymbol{\delta = 0.05}, \quad \boldsymbol{\zeta = 0.0}, \quad \boldsymbol{D = 0.1}, \quad \boldsymbol{\beta = 1.5},$$
$$\boldsymbol{\gamma = 0.02}, \quad \boldsymbol{\epsilon = 0.1}, \quad \boldsymbol{\eta = 0.05}, \quad \boldsymbol{\theta = 0.1}$$

As the initial guess for Newton's method, y_A vector was initialized using linspace function from 0.5 to 0.3 with N nodes, y_B was initialized using linspace function from 0.5 to 0.2 with N nodes, and y_U and y_F were all initialized to N zeros.

For part (c), the following parameters were set:

$$\boldsymbol{\delta = 0.05}, \quad \boldsymbol{\zeta = 0.03}, \quad \boldsymbol{D = 0.1}, \quad \boldsymbol{\beta = 1.5},$$
$$\boldsymbol{\gamma = 0.02}, \quad \boldsymbol{\epsilon = 0.1}, \quad \boldsymbol{\eta = 0.05}, \quad \boldsymbol{\theta = 0.1}$$

The same initial guess was used as for part (b) for Newton's method.

Species U and F concentrations were zero across all nodes for part (a).

For part (a), the following steady-state concentrations profile was produced:

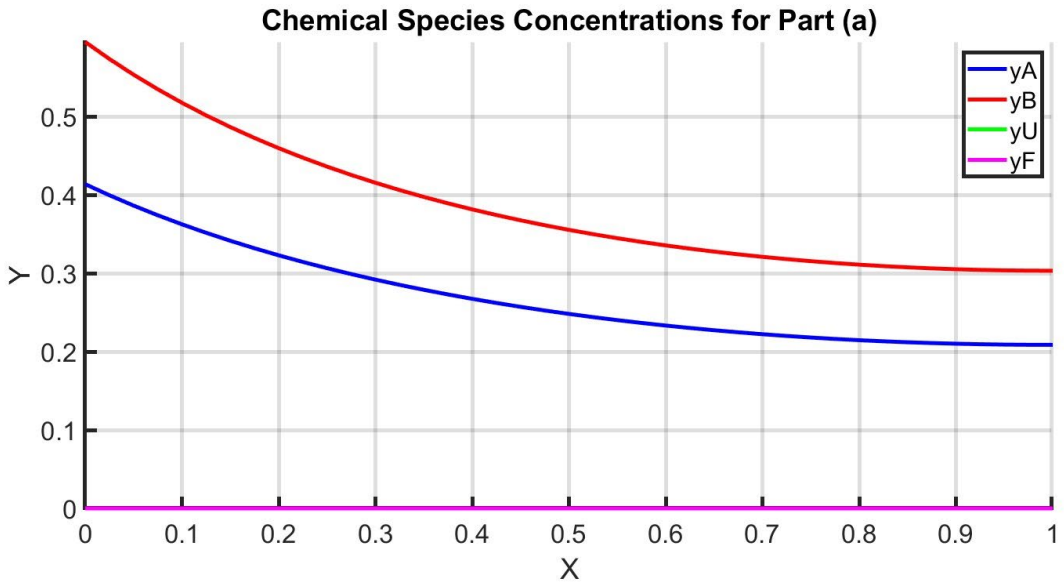

Figure 3: Chemical Species Concentrations for Part (a)

Species F concentrations were zero across all nodes for part (b).

For part (b), the following steady-state concentrations profile was produced:

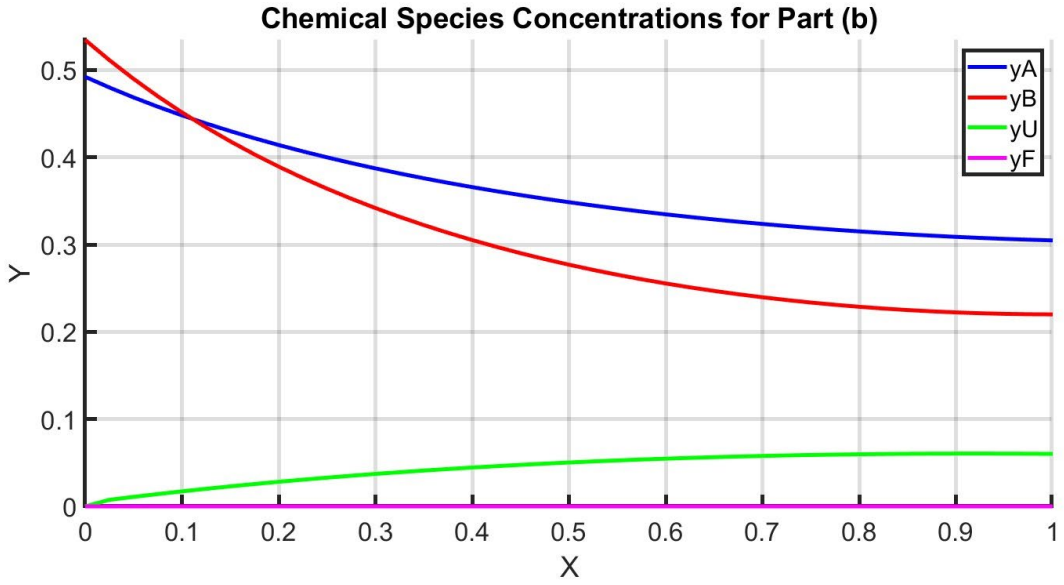

Figure 4: Chemical Species Concentrations for Part (b)

Species F concentrations were on the order of 10^{-3} for nodes from i = 4 to i = 41 for part (c).

For part (c), the following steady-state concentrations profile was produced:

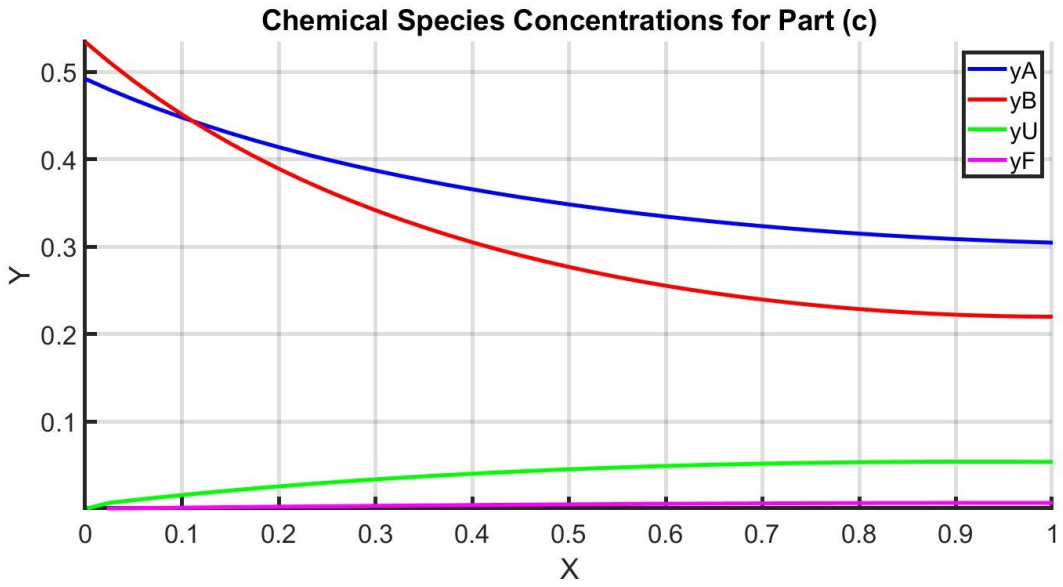

Figure 5: Chemical Species Concentrations for Part (c)

2.1.2. Question 2: Transient-state solution results

For transient system, for t = 0, all elements of vectors y were set to zeros as initial conditions. Parameters from part (c) of the steady-state system were used. Implicit Euler was advanced in time until 99% of the steady-state solution from part (c) was obtained. Time step was set to 1. Newton's method for implicit Euler was iterated until the square of 2-norm of the difference between 2 successive Y vectors was less than 0.01. The initial guesses for Newton's method were zero vectors for all species.

For part (c) parameters, the following evolution of chemical species concentrations overtime was obtained:

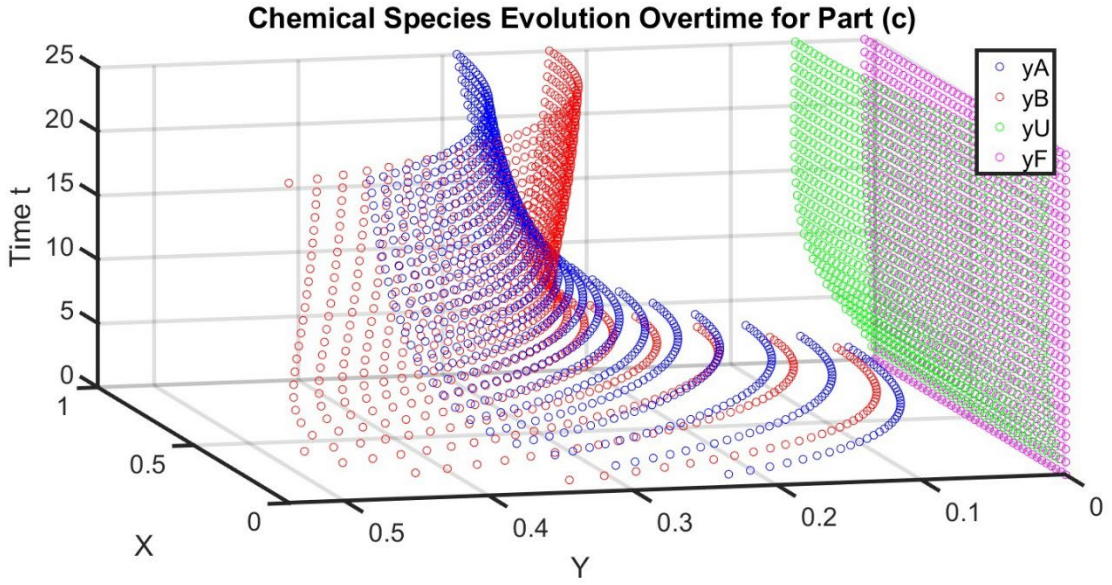

Figure 6: Chemical Species Evolution Overtime for Part (c)

Concentrations of species A and B increase over time. They increase faster near the left side at x = 0. Starting at t = 10, species B concentrations begin to decline, while A continues to increase. Species U and F increase over time. Particularly they increase faster towards the right side at x = 1.

2.2. Discussion

2.1.2. Question 1: Steady-state discussion

To determine the least number of nodes required for accurate steady-state solution, y_A values at x = 0.3 were compared for results with different nodes. The testing algorithm was developed that started Newton's method computation of results with N = 11 nodes. The number of nodes was increased by 10 at each iteration of the testing algorithm. The index of the node at x = 0.3 can be calculated using the following general formula:

$$i = \frac{0.3}{\frac{1}{N-1}} + 1 \tag{32}$$

Where i is the index at x = 0.3, and N is the total number of nodes. 1 is added to fraction to offset the node by 1 because i = 1 corresponds to x = 0. The algorithm saved result at x = 0.3 for y_A concentration for N nodes and increased N by 10 for next iteration. The algorithm then compared 2 results for N and N – 10 nodes by calculating percentage difference. The testing algorithm stopped once the percentage difference became less than 0.01%. For all parts (a), (b), and (c), the number of nodal points that give percentage difference less than 0.01% was determined to be N = 41.
The criteria of accuracy of the computed results for Newton's method was judged by calculating the square of 2-norm of the difference of 2 successive iteration vectors:

$$\left|\left|\delta Y\right|\right|_2^2 < 10^{(-2)} \tag{33}$$

δY is given by equation 28. Once the square of 2 norm became less than $10^{(-2)}$, the Newton's iterations were stopped, and $Y^{(M)}$ vector at M-th iteration was plotted.

As can be seen in figure 3, for parameters in part (a), the steady-state distribution of reactant species A and B concentrations is highest near x = 0 and gradually decreases towards x = 1. The product species U and F concentrations are zero for all x. There are several parameters that equal 0 in part (a):

$$\delta = 0, \qquad \zeta = 0, \qquad \epsilon = 0, \qquad \eta = 0, \qquad \theta = 0$$

Parameter δ can only be 0 if k_3 is 0, according to dimensionless definitions. k_3 is rate constant for the third reaction that converts B to U. Therefore, when δ is 0, the rate of reaction that produces U is 0, thus, there is no product U. Similarly, ζ can only be 0 if k_4 is zero, according to definitions of dimensionless parameters. k_4 is rate constant for the fourth reaction that produces F. Therefore, there is no F product. ϵ, η and θ correspond to derivative boundary conditions at x = 1 for y_A, y_B, and y_U, respectively. Therefore, when these parameters are 0, there is no change in y_A, y_B, and y_U concentration at the right boundary. The concentrations of y_A and y_B decrease across x because they get consumed by first reaction that produces product D. As species A and B diffuse across x, their concentrations decrease.

As can be seen in figure 4, for parameters in part (b), the steady-state distribution of concentrations of species A and B is the highest near x = 0 and decreases towards x = 1. In this part, parameters δ, ϵ, η, and θ are not 0 anymore. Therefore, as previously discussed, parameter δ corresponds to reaction rate of reaction 3 that produces U. Concentration of species U increases as x increases. Furthermore, compared with figure 3, it appears that concentration of species B decreases faster, which corresponds to contribution of reaction 3 that consumes B to produce U. Concentration of species F is 0 across all x as ζ remains 0 in part (b), so the reaction rate constant for reaction 4 is 0. No product F is produced.

As can be seen in figure 5, for parameters in part (c), the steady-state distribution of concentrations of species A and B is essentially the same as in figure 4. However, in

this figure, concentrations of species F increase across x, and are on the order of 10^{-3} for most values of x. ζ is no longer 0, therefore, k_4 reaction rate constant for reaction 4 is not 0 and product F is produced from U. Compared to figure 3, concentrations of species U are slightly less for figure 4, which can be explained by reaction 4. Some U is consumed to produce small amounts of F.

2.1.3. Question 2: Transient-state discussion

For transient-state solution, explicit Euler was tried initially to integrate the system in time. However, results diverged for even small time-steps. For extremely small time-steps of 10^{-8} or less, explicit Euler did not immediately diverge, however, time-steps of this magnitude would require considerable amount of time to reach steady state solution. Therefore, implicit Euler was chosen as a method of integration because it is unconditionally stable for any time-step.

Implicit Euler system of homogeneous equations was constructed, and Newton's method was used to calculate Y concentrations vector for the next time step. The criteria to judge the accuracy of Newton's method was the square of 2-norm of the difference between 2 successive vectors $Y^{(M)}$ and $Y^{(M+1)}$. The criteria are shown in equation 33. When the square of 2-norm became less than 10^{-2}, Newton's iteration was terminated and $Y^{(M+1)}$ vector at next time step was returned. The implicit Euler was advanced in time until 99% of the steady state solution from Question 1 was reached. To calculate how close transient-state is to steady-state solution, the difference between the current time-step vector and the steady-state was taken, and the square of 2-norm of the difference was calculated. When square of the 2-norm became less than 0.01, the transient-state reached 99% of the steady-state.

As can be seen from figure 6, for time near t = 0, concentrations of species A and B are higher near x = 0 and approach 0 as x increases. Concentrations of species U and F are zero at t = 0. As time increases, concentrations of species A and B increase across all x, but the curve trend remains. The concentration is higher near x = 0 and decreases as x increases. With increasing time, concentrations of U and F begin to increase. Concentrations of U and F species are higher near x = 1 than x = 0. Starting at around t = 10, concentrations of species B begin to decrease for all x. A continues to increase but at a slower rate. U and F continue to increase as well, but at a slower rate. At time around t = 25, the steady state is reached as can be seen on top of the plot. At steady state, concentration of species B is higher than A at x = 0, but begins to decrease until it becomes less than A, just like in steady-state solution in figure 5. Concentrations of species U are low near x = 0 and increase towards x = 1. Concentrations of species F are low just like in figure 5. Overall, the steady state in figure 6 closely resembles figure 5.

3. Appendices

3.1. Main Program

```
%% Clear Cache
clear all %#ok<*CLALL>
close all
clc

%% Main Script

%  Set initial percentDiff
percentDiff = 100;

%  Assign number of nodes
    N = 11;

while percentDiff > 0.01

    %  To test different conditions, uncomment one of the parts of the code
for
    %  conditions a, b, and c

%     %  Assign conditions for case (a)
%     xi = 0;
%     xf = 1;
%     delta = 0;
%     zeta = 0;
%     D = 0.1;
%     beta = 1.5;
%     gamma = 0.05;
%     epsilon = 0.0;
%     eta = 0.0;
%     theta = 0.0;

%     %  Assign conditions for case (b)
%     xi = 0;
%     xf = 1;
%     delta = 0.05;
%     zeta = 0;
%     D = 0.1;
%     beta = 1.5;
%     gamma = 0.02;
%     epsilon = 0.1;
%     eta = 0.05;
%     theta = 0.1;

    %  Assign conditions for case (c)
    xi = 0;
    xf = 1;
    delta = 0.05;
    zeta = 0.03;
    D = 0.1;
    beta = 1.5;
```

```
gamma = 0.02;
epsilon = 0.1;
eta = 0.05;
theta = 0.1;


%  Assign vector x
x = linspace(xi,xf,N);
x = transpose(x);

%  Assign distance
h = (x(N) - x(1))/(N-1);

%% Part 1

%  Uncomment proper initial conditions for corresponding part before
%  running the code
%    % Initial conditions for part (a)
%    yA = zeros(N,1);
%    yB = zeros(N,1);
%    yU = zeros(N,1);
%    yF = zeros(N,1);

%  Initial conditions for part (b) & (c)
yA = linspace(0.5,0.3,N);
yA = transpose(yA);
yB = linspace(0.5,0.2,N);
yB = transpose(yB);
yU = zeros(N,1);
yF = zeros(N,1);

%  Combine vectors without initial conditions
Y0 = [yA; yB; yU(2:N); yF(2:N)];

%  Initiate error
error = 100;

while error > 10^-2

    %  Assign vector F(Y0)
    FunA = FA(yA,yB,N,h,D,gamma,epsilon);
    FunB = FB(yA,yB,N,h,D,beta,eta,delta);
    FunU = FU(yU,yB,N,h,D,theta,delta,zeta);
    FunF = FF(yF,yU,N,h,D,zeta);

    %  Combine to construct F solution vector
    F = [FunA; FunB; FunU; FunF];

    %  Get Jacobian
    J = Jacobian(yA,yB,yU,yF,N,h,D,gamma,epsilon,delta,eta,zeta,theta);

    %  Use LUSolver
    dY = LUSolver(J,-F);

    %  Calculate Y at new iteration
```

```
        Y = dY + Y0;

        %  Calculate square of 2-norm of vector
        error = norm(dY)^2;

        %  Display the error
        %fprintf('The error is: %.2f\n', error)

        %  Assign Y as previous iteration before new iteration
        Y0 = Y;
        %  Obtain vectors yA-yF from Y0
        yA = Y0(1:N);
        yB = Y0(N+1:2*N);
        yU(2:N) = Y0(2*N+1:3*N-1);
        yF(2:N) = Y0(3*N:4*N-2);

    end % End while loop

    %  Calculate the test node index
    testNodeIndex = round(0.3 / (1 / (N - 1)) + 1);

    %  If N is greater than 11
    if N > 11

        %  Calculate percent difference
        percentDiff = (abs(testValue - yA(testNodeIndex)) / ((testValue +
yA(testNodeIndex)) / 2)) * 100;

    end

    fprintf('Percent difference for N = %i is %.4f\n', N, percentDiff)

    %  Save value to be tested for next iteration
    testValue = yA(testNodeIndex);

    %  Increment the number of nodes by 10
    N = N + 10;


end

%  Get least number of nodes for percent difference accuracy to be less
%  than 0.1%
numberOfNodesRequired = N - 10;

%  Decrement number of nodes by 10
N = N - 10;

fprintf('Number of nodes needed for percent difference between values yA for N
nodes\nand yA for N-10 nodes at x = 0.3 to be less than 0.01%% is %i\n',
numberOfNodesRequired)

%  Copy vector to Y1Result for comparison with next part
Y1result = [yA; yB; yU; yF];
```

```
%   Plot solutions
figure(1) % Create figure 1 for the plot
hold on % Draw all on the same figure
grid on % Turn grid on
%  Plot yA
plot(x,yA,'b','LineWidth',3)
%  Plot yB
plot(x,yB,'r','LineWidth',3)
%  Plot yU
plot(x,yU,'g','LineWidth',3)
%  Plot yF
plot(x,yF,'m','LineWidth',3)

title('Chemical Species Concentrations for Part (c)','FontSize',24)
xlabel('X')
ylabel('Y')
%   Set limits on axes
xlim([min(x) max(x)])
ylim([min(Y0) max(Y0)])

%   Set screen position and figure size
set(gcf,'Position',[75 75 1275 600])
%   Format axes lengths and label fonts
set(gca,'LineWidth',3,'FontSize',20)

%   Set figure legend
legend('yA','yB','yU','yF')

%% Part 2

%  Set initial values for all species
yA = zeros(N,1);
yB = zeros(N,1);
yU = zeros(N,1);
yF = zeros(N,1);

%  Set time step
dt = 1;

%  Set current time
t = 0;

%  Set time coordinate vector of N nodes for 3-D scatter plot
tVec = ones(N,1)*t;

%% Implicit Euler

%  Construct vector Yi at time step t
Yi = [yA; yB; yU; yF];

%  Set norm
n = 1;

%  Create figure 2
figure(2)
```

```
hold on
grid on

%  Plot first layer of scatter 3-D plot at t = 0
scatter3(x,yA,tVec,'b')
scatter3(x,yB,tVec,'r')
scatter3(x,yU,tVec,'g')
scatter3(x,yF,tVec,'m')

while n > 0.01
    %  Call implicit Euler to advance over 1 iteration
    Yf =
ImplEuler(yA,yB,yU,yF,N,dt,h,D,gamma,epsilon,delta,eta,zeta,theta,beta);

    %  Advance current time
    t = t + dt;

    %  Create new time coordinate vector for 3-D scatter plot
    tVec = ones(N,1)*t;

    %  Calculate norm
    n = norm(Y1result-Yf)^2;

    fprintf('Norm at t = %.2f is: %.2f\n', t, n)

    %  Set vectors for next iteration
    yA = Yf(1:N);
    yB = Yf(N+1:2*N);
    yU = Yf(2*N+1:3*N);
    yF = Yf(3*N+1:4*N);

    %  Plot new layer of scatter 3-D plot
    scatter3(x,yA,tVec,'b')
    scatter3(x,yB,tVec,'r')
    scatter3(x,yU,tVec,'g')
    scatter3(x,yF,tVec,'m')

end

xlabel('X')
ylabel('Y')
zlabel('Time t')

title('Chemical Species Evolution Overtime for Part (c)','FontSize',24)

%   Set screen position and figure size
set(gcf,'Position',[75 75 1275 600])

%   Format axes lengths and label fonts
set(gca,'LineWidth',3,'FontSize',20)

%   Set figure legend
legend('yA','yB','yU','yF')
```

```
xlim([min(x) max(x)])
ylim([min(Yf) max(Yf)])
zlim([0 t])
```

3.2. FA Function

```
function FunA = FA(yA,yB,N,h,D,gamma,epsilon)
%FA returns vector function FunA for N nodes in yA

FunA = zeros(N,1);

for i = 1:1:N
    if i == 1
        FunA(i) = D*(yA(i+1)-2*yA(i)+yA(i+1)+2*h*(1-yA(i)))/h^2-yA(i)*yB(i)^2-
gamma*yA(i);
    elseif i == N
        FunA(i) = D*(yA(i-1)-2*h*epsilon*yA(i)-2*yA(i)+yA(i-1))/h^2-
yA(i)*yB(i)^2-gamma*yA(i);
    else
        FunA(i) = D*(yA(i+1)-2*yA(i)+yA(i-1))/h^2-yA(i)*yB(i)^2-gamma*yA(i);
    end % End if statement
end % End solution for all f

end
```

3.3. FB Function

```
function FunB = FB(yA,yB,N,h,D,beta,eta,delta)
%FB returns vector function FunB for N nodes in yB

FunB = zeros(N,1);

for i=1:1:N
    if i == 1
        FunB(i) = D*(yB(i+1)-2*yB(i)+yB(i+1)+2*h*(beta-yB(i)))/h^2-
2*yA(i)*yB(i)^2-delta*yB(i);
    elseif i == N
        FunB(i) = D*(yB(i-1)-2*h*eta*yB(i)^2-2*yB(i)+yB(i-1))/h^2-
2*yA(i)*yB(i)^2-delta*yB(i);
    else
        FunB(i) = D*(yB(i+1)-2*yB(i)+yB(i-1))/h^2-2*yA(i)*yB(i)^2-delta*yB(i);
    end % End if statement
end % End for loop

end
```

3.4. FU Function

```
function FunU = FU(yU,yB,N,h,D,theta,delta,zeta)
%FU returns vector function FunU for N nodes in yU

FunU = zeros(N-1,1);

for i=1:1:N-1
    % If i == 1, boundary condition for yU at x = 0
```

```
    if i == 1
        FunU(i) = D*(yU(i+1)-2*yU(i))/h^2+delta*yB(i)-zeta*yU(i);
    elseif i == (N-1)
        FunU(i) = D*(yU(i-1)-2*h*theta*yU(i)-2*yU(i)+yU(i-1))/h^2+delta*yB(i)-
zeta*yU(i);
    else
        FunU(i) = D*(yU(i+1)-2*yU(i)+yU(i-1))/h^2+delta*yB(i)-zeta*yU(i);

    end % End if statement

end % End for loop

end
```

3.5. FF Function

```
function FunF = FF(yF,yU,N,h,D,zeta)
%FF returns vector function FunF for N nodes in yF

FunF = zeros(N-1,1);

for i=1:1:N-1
    % If i == 1, boundary condition for yF at x = 0
    if i == 1
        FunF(i) = D*(yF(i+1)-2*yF(i))/h^2+zeta*yU(i);

    elseif i == (N-1)
        FunF(i) = D*(yF(i-1)-2*yF(i)+yF(i-1))/h^2+zeta*yU(i);

    else
        FunF(i) = D*(yF(i+1)-2*yF(i)+yF(i-1))/h^2+zeta*yU(i);

    end % End if statement

end % End for loop

end
```

3.6. Jacobian Function

```
function J = Jacobian(yA,yB,yU,yF,N,h,D,gamma,epsilon,delta,eta,zeta,theta)
%JAOBIAN Returns Jacobian

%  Allocate Jacobian segments
JAdyA = zeros(N,N);
JAdyB = zeros(N,N);
JAdyU = zeros(N,N-1);
JAdyF = zeros(N,N-1);

JBdyA = zeros(N,N);
JBdyB = zeros(N,N);
JBdyU = zeros(N,N-1);
JBdyF = zeros(N,N-1);

JUdyA = zeros(N-1,N);
```

```
JUdyB = zeros(N-1,N);
JUdyU = zeros(N-1,N-1);
JUdyF = zeros(N-1,N-1);

JFdyA = zeros(N-1,N);
JFdyB = zeros(N-1,N);
JFdyU = zeros(N-1,N-1);
JFdyF = zeros(N-1,N-1);

%  Taking derivative of JA with respect to yA
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JAdyA(row,col) = D*(-2-2*h)/h^2-yB(row)^2-gamma;

        elseif row == 1 && col == 2
            JAdyA(row,col) = D*2/h^2;

        elseif row == N && col == N-1
            JAdyA(row,col) = D*2/h^2;

        elseif row == N && col == N
            JAdyA(row,col) = D*(-2*h*epsilon-2)/h^2-yB(row)^2-gamma;

        elseif abs(row-col) == 1
            JAdyA(row,col) = D/h^2;

        elseif row == col
            JAdyA(row,col) = -D*2/h^2-yB(row)^2-gamma;

        end

    end
end

%  Taking derivative of JA with respect to yB
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JAdyB(row,col) = -2*yA(row)*yB(row);

        elseif row == N && col == N
            JAdyB(row,col) = -2*yA(row)*yB(row);

        elseif row == col
            JAdyB(row,col) = -2*yA(row)*yB(row);

        end

    end
end

%  Taking derivative of JB with respect to yA
for row = 1:1:N
    for col = 1:1:N
```

```
        if row == 1 && col == 1
            JBdyA(row,col) = -2*yB(row)^2;

        elseif row == N && col == N
            JBdyA(row,col) = -2*yB(row)^2;

        elseif row == col
            JBdyA(row,col) = -2*yB(row)^2;

        end

    end
end

%  Taking derivative of JB with respect to yB
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JBdyB(row,col) = D*(-2-2*h)/h^2-4*yA(row)*yB(row)-delta;

        elseif row == 1 && col == 2
            JBdyB(row,col) = D*2/h^2;

        elseif row == N && col == N-1
            JBdyB(row,col) = D*2/h^2;

        elseif row == N && col == N
            JBdyB(row,col) = D*(-4*h*eta*yB(row)-2)/h^2-4*yA(row)*yB(row)-
delta;

        elseif abs(row-col) == 1
            JBdyB(row,col) = D/h^2;

        elseif row == col
            JBdyB(row,col) = -D*2/h^2-4*yA(row)*yB(row)-delta;

        end

    end
end

%  Taking derivative of JU with respect to yB
for row = 1:1:N-1
    for col = 1:1:N

        if (col-row) == 1
            JUdyB(row,col) = delta;

        end

    end
end

%  Taking derivative of JU with respect to yU
for row = 1:1:N-1
```

```
    for col = 1:1:N-1

        if row == (N-1) && col == (N-2)
            JUdyU(row,col) = 2*D/h^2;

        elseif row == (N-1) && col == (N-1)
            JUdyU(row,col) = D*(-2*h*theta-2)/h^2-zeta;

        elseif abs(row-col) == 1
            JUdyU(row,col) = D/h^2;

        elseif row == col
            JUdyU(row,col) = -2*D/h^2-zeta;
        end

    end
end

%  Taking derivative of JF with respect to yU
for row = 1:1:N-1
    for col = 1:1:N-1

        if row == col
            JFdyU(row,col) = zeta;
        end

    end
end

%  Taking derivative of JF with respect to yF
for row = 1:1:N-1
    for col = 1:1:N-1

        if row == (N-1) && col == (N-2)
            JFdyF(row,col) = 2*D/h^2;

        elseif abs(row-col) == 1
            JFdyF(row,col) = D/h^2;

        elseif row == col
            JFdyF(row,col) = -2*D/h^2+zeta;
        end

    end
end

J = [JAdyA, JAdyB, JAdyU, JAdyF;
     JBdyA, JBdyB, JBdyU, JBdyF;
     JUdyA, JUdyB, JUdyU, JUdyF;
     JFdyA, JFdyB, JFdyU, JFdyF];

end
```

3.7. LUSolver Function

This program was taken from homework assignment #2 and performs LU decomposition with pivoting to yield solution vector.

```
function [x] = LUSolver(A,b)
%LUSolver Solves system Ax=b using LU decomposition with partial pivoting
%   A is a square NxN matrix, b is Nx1 vector, x is Nx1 solution vector.
%   The function performs partial pivoting of A and then LU decomposition
%   to calculate x.

%% Error check and preparation

%  Check that A is a square matrix
if size(A,1) ~= size(A,2)
    error('Error: A must be a square matrix\n')
end % End square matrix check

%  Check that b is a vector and the same length as a dimension of A
if size(b,2) ~= 1
    error('Error: b must be a vector\n')
end % End vector check

if length(b) ~= size(A,1)
    error('Error: Length of b must be equal to N of NxN matrix A\n')
end % End length check

%  Check that determinant of A is not zero to have a unique solution
if det(A) == 0
    error('Error: Determinant of A is zero. No unique solution.\n')
end % End determinant check

%  Assign dimension of NxN matrix
N = size(A,1);

%  Preallocate NxN L and U matrices as well as Nx1 vectors y and x
L = zeros(N,N);
U = zeros(N,N);
y = zeros(N,1);
x = zeros(N,1);

%% Partial Pivoting

%  For each column
for col = 1:1:N
    %  Find the index of the maximum absolute value in the column
    [~, rowIndx] = max(abs(A(:,col)));

    %  Replace the row of the current submatrix with the row that has the
    %  largest absolute value in the column
    A([col rowIndx],:) = A([rowIndx col],:);

    %  Switch corresponding entries of vector b
    b([col rowIndx],:) = b([rowIndx col],:);
```

```
end % End pivoting

%% LU Decomposition

%  Assign diagonal of L to be filled with 1s
for i=1:1:N
    L(i,i) = 1;
end

%  Assign first row of U to be the first row of A
U(1,:) = A(1,:);
%  Compute the first column of L
L(:,1) = A(:,1)/U(1,1);

%  For the rest of indicies
for i=2:1:N
    %  From column i to N
    for j=i:1:N
        %  Compute U entries
        U(i,j) = A(i,j) - L(i,1:i-1)*U(1:i-1,j);
    end % End computation of U
    %  From row i+1 to N
    for k=i+1:1:N
        %  Compute L entries
        L(k,i) = (A(k,i) - L(k,1:i-1)*U(1:i-1,i))/U(i,i);
    end % End computation of L
end % End computation of L and U

%  Forward substitution
%  Compute first element of y
y(1) = b(1)/L(1,1);

%  For rest of y elements
for k=2:N
    %  Compue all elements of y vector
    y(k) = (b(k) - L(k,1:k-1)*y(1:k-1))/L(k,k);
end %  End y vector computation

%  Back substitution
%  Compute last element of vector x
x(N) = y(N)/U(N,N);

%  For rest of x elements
for k=N-1:-1:1
    %  Compute all elements of x
    x(k) = (y(k) - U(k,k+1:N)*x(k+1:N))/U(k,k);
end % End computation of vector x

end % End of LUSolver function
```

3.8. Implicit Euler Function

```
function Yf =
ImplEuler(yAi,yBi,yUi,yFi,N,dt,h,D,gamma,epsilon,delta,eta,zeta,theta,beta)
%IMPLEULER Returns Yf after one iteration over dt
```

```
%   Detailed explanation goes here

%  Make guess vectors
yA = zeros(N,1);
yB = zeros(N,1);
yU = zeros(N,1);
yF = zeros(N,1);

Y0 = zeros(4*N-2,1);

%  Combine vectors without initial conditions
Y0 = [yA; yB; yU(2:N); yF(2:N)];

%  initiate error
error = 100;

while error > 10^-2

    %  Assign vector F(Y0)
    FunA = FAp2(yA,yB,N,h,D,gamma,epsilon,dt,yAi);
    FunB = FBp2(yA,yB,N,h,D,beta,eta,delta,dt,yBi);
    FunU = FUp2(yU,yB,N,h,D,theta,delta,zeta,dt,yUi);
    FunF = FFp2(yF,yU,N,h,D,zeta,dt,yFi);

    %  Combine to construct F solution vector
    F = [FunA; FunB; FunU; FunF];

    %  Get Jacobian
    J = JacobianP2(yA,yB,yU,yF,N,h,D,gamma,epsilon,delta,eta,zeta,theta,dt);

    %  Use LUSolver
    dY = LUSolver(J,-F);

    %  Calculate Y at new iteration
    Y = Y0 + dY;

    %  Calculate 2-norm of vector
    error = norm(dY)^2;

    %  Display the error
    %fprintf('Implicit Euler error: %.2f\n',error)

    %  Assign Y as previous iteration before new iteration
    Y0 = Y;
    %  Obtain vectors yA-yF from Y0
    yA = Y0(1:N);
    yB = Y0(N+1:2*N);
    yU(2:N) = Y0(2*N+1:3*N-1);
    yF(2:N) = Y0(3*N:4*N-2);

end

%  Construct Yf
Yf = [yA; yB; yU; yF];
end
```

3.9. FAp2 Function

```
function FunA = FAp2(yA,yB,N,h,D,gamma,epsilon,dt,yAi)
%FAp2 returns vector function FunA for N nodes in yA for part 2

FunA = zeros(N,1);

for i = 1:1:N
    if i == 1
        FunA(i) = yAi(i)-yA(i)+(D*(yA(i+1)-2*yA(i)+yA(i+1)+2*h*(1-yA(i)))/h^2-
yA(i)*yB(i)^2-gamma*yA(i))*dt;
    elseif i == N
        FunA(i) = yAi(i)-yA(i)+(D*(yA(i-1)-2*h*epsilon*yA(i)-2*yA(i)+yA(i-
1))/h^2-yA(i)*yB(i)^2-gamma*yA(i))*dt;
    else
        FunA(i) = yAi(i)-yA(i)+(D*(yA(i+1)-2*yA(i)+yA(i-1))/h^2-yA(i)*yB(i)^2-
gamma*yA(i))*dt;
    end % End if statement
end % End solution for all f

end
```

3.10. FBp2 Function

```
function FunB = FBp2(yA,yB,N,h,D,beta,eta,delta,dt,yBi)
%FBp2 returns vector function FunB for N nodes in yB for part 2

FunB = zeros(N,1);

for i=1:1:N
    if i == 1
        FunB(i) = yBi(i)-yB(i)+(D*(yB(i+1)-2*yB(i)+yB(i+1)+2*h*(beta-
yB(i)))/h^2-2*yA(i)*yB(i)^2-delta*yB(i))*dt;
    elseif i == N
        FunB(i) = yBi(i)-yB(i)+(D*(yB(i-1)-2*h*eta*yB(i)^2-2*yB(i)+yB(i-
1))/h^2-2*yA(i)*yB(i)^2-delta*yB(i))*dt;
    else
        FunB(i) = yBi(i)-yB(i)+(D*(yB(i+1)-2*yB(i)+yB(i-1))/h^2-
2*yA(i)*yB(i)^2-delta*yB(i))*dt;
    end % End if statement
end % End for loop

end
```

3.11. FUp2 Function

```
function FunU = FUp2(yU,yB,N,h,D,theta,delta,zeta,dt,yUi)
%FUp2 returns vector function FunU for N nodes in yU for part 2

FunU = zeros(N-1,1);

for i=1:1:N-1
    % If i == 1, boundary condition for yU at x = 0
    if i == 1
```

```
        FunU(i) = yUi(i)-yU(i)+(D*(yU(i+1)-2*yU(i))/h^2+delta*yB(i)-
zeta*yU(i))*dt;
    elseif i == (N-1)
        FunU(i) = yUi(i)-yU(i)+(D*(yU(i-1)-2*h*theta*yU(i)-2*yU(i)+yU(i-
1))/h^2+delta*yB(i)-zeta*yU(i))*dt;
    else
        FunU(i) = yUi(i)-yU(i)+(D*(yU(i+1)-2*yU(i)+yU(i-1))/h^2+delta*yB(i)-
zeta*yU(i))*dt;

    end % End if statement

end % End for loop

end
```

3.12. FFp2 Function

```
function FunF = FFp2(yF,yU,N,h,D,zeta,dt,yFi)
%FFp2 returns vector function FunF for N nodes in yF for part 2

FunF = zeros(N-1,1);

for i=1:1:N-1
    % If i == 1, boundary condition for yF at x = 0
    if i == 1
        FunF(i) = yFi(i)-yF(i)+(D*(yF(i+1)-2*yF(i))/h^2+zeta*yU(i))*dt;

    elseif i == (N-1)
        FunF(i) = yFi(i)-yF(i)+(D*(yF(i-1)-2*yF(i)+yF(i-
1))/h^2+zeta*yU(i))*dt;

    else
        FunF(i) = yFi(i)-yF(i)+(D*(yF(i+1)-2*yF(i)+yF(i-
1))/h^2+zeta*yU(i))*dt;

    end % End if statement

end % End for loop

end
```

3.13. JacobianP2 Function

```
function J =
JacobianP2(yA,yB,yU,yF,N,h,D,gamma,epsilon,delta,eta,zeta,theta,dt)
%JAOBIAN Returns Jacobian for part 2

%  Allocate Jacobian segments
JAdyA = zeros(N,N);
JAdyB = zeros(N,N);
JAdyU = zeros(N,N-1);
JAdyF = zeros(N,N-1);
```

```
JBdyA = zeros(N,N);
JBdyB = zeros(N,N);
JBdyU = zeros(N,N-1);
JBdyF = zeros(N,N-1);

JUdyA = zeros(N-1,N);
JUdyB = zeros(N-1,N);
JUdyU = zeros(N-1,N-1);
JUdyF = zeros(N-1,N-1);

JFdyA = zeros(N-1,N);
JFdyB = zeros(N-1,N);
JFdyU = zeros(N-1,N-1);
JFdyF = zeros(N-1,N-1);

%  Taking derivative of JA with respect to yA
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JAdyA(row,col) = (D*(-2-2*h)/h^2-yB(row)^2-gamma)*dt-1;

        elseif row == 1 && col == 2
            JAdyA(row,col) = (D*2/h^2)*dt;

        elseif row == N && col == N-1
            JAdyA(row,col) = (D*2/h^2)*dt;

        elseif row == N && col == N
            JAdyA(row,col) = (D*(-2*h*epsilon-2)/h^2-yB(row)^2-gamma)*dt-1;

        elseif abs(row-col) == 1
            JAdyA(row,col) = (D/h^2)*dt;

        elseif row == col
            JAdyA(row,col) = (-D*2/h^2-yB(row)^2-gamma)*dt-1;

        end

    end
end

%  Taking derivative of JA with respect to yB
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JAdyB(row,col) = (-2*yA(row)*yB(row))*dt;

        elseif row == N && col == N
            JAdyB(row,col) = (-2*yA(row)*yB(row))*dt;

        elseif row == col
            JAdyB(row,col) = (-2*yA(row)*yB(row))*dt;

        end
```

```
    end
end

%  Taking derivative of JB with respect to yA
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JBdyA(row,col) = (-2*yB(row)^2)*dt;

        elseif row == N && col == N
            JBdyA(row,col) = (-2*yB(row)^2)*dt;

        elseif row == col
            JBdyA(row,col) = (-2*yB(row)^2)*dt;

        end

    end
end

%  Taking derivative of JB with respect to yB
for row = 1:1:N
    for col = 1:1:N
        if row == 1 && col == 1
            JBdyB(row,col) = (D*(-2-2*h)/h^2-4*yA(row)*yB(row)-delta)*dt-1;

        elseif row == 1 && col == 2
            JBdyB(row,col) = (D*2/h^2)*dt;

        elseif row == N && col == N-1
            JBdyB(row,col) = (D*2/h^2)*dt;

        elseif row == N && col == N
            JBdyB(row,col) = (D*(-4*h*eta*yB(row)-2)/h^2-4*yA(row)*yB(row)-
delta)*dt-1;

        elseif abs(row-col) == 1
            JBdyB(row,col) = (D/h^2)*dt;

        elseif row == col
            JBdyB(row,col) = (-D*2/h^2-4*yA(row)*yB(row)-delta)*dt-1;

        end

    end
end

%  Taking derivative of JU with respect to yB
for row = 1:1:N-1
    for col = 1:1:N

        if (col-row) == 1
            JUdyB(row,col) = delta*dt;

        end
```

```
    end
end

%  Taking derivative of JU with respect to yU
for row = 1:1:N-1
    for col = 1:1:N-1

        if row == (N-1) && col == (N-2)
            JUdyU(row,col) = (2*D/h^2)*dt;

        elseif row == (N-1) && col == (N-1)
            JUdyU(row,col) = (D*(-2*h*theta-2)/h^2-zeta)*dt-1;

        elseif abs(row-col) == 1
            JUdyU(row,col) = (D/h^2)*dt;

        elseif row == col
            JUdyU(row,col) = (-2*D/h^2-zeta)*dt-1;
        end

    end
end

%  Taking derivative of JF with respect to yU
for row = 1:1:N-1
    for col = 1:1:N-1

        if row == col
            JFdyU(row,col) = zeta*dt;
        end

    end
end

%  Taking derivative of JF with respect to yF
for row = 1:1:N-1
    for col = 1:1:N-1

        if row == (N-1) && col == (N-2)
            JFdyF(row,col) = (2*D/h^2)*dt;

        elseif abs(row-col) == 1
            JFdyF(row,col) = (D/h^2)*dt;

        elseif row == col
            JFdyF(row,col) = (-2*D/h^2+zeta)*dt-1;
        end

    end
end

J = [JAdyA, JAdyB, JAdyU, JAdyF;
     JBdyA, JBdyB, JBdyU, JBdyF;
     JUdyA, JUdyB, JUdyU, JUdyF;
```

```
        JFdyA, JFdyB, JFdyU, JFdyF];

    end
```