MATH*6020 Project
Solution of 2D diffusion-reaction equations using numerical methods implemented in FORTRAN.

Vincent Emond
0746222

April 15, 2016

Abstract

An analysis of a semi-linear diffusion reaction problems of the form $u_t = D\Delta u - F(u)$ was performed by implementation of numerical methods in FORTRAN. These problems appear in many areas such as chemistry, biology, and math homework. The problem was broken up into three parts. Solutions were calculated and tested with various numerical methods. This report includes a description of the problems, calculations and visualizations of the solutions.

Contents

The Problem: 2D Diffusion reaction equation

The goal of the project was to solve a semi-linear diffusion reaction problem of the form $u_t = D\Delta u - F(u)$. The problem can be imagined as simulating a cell's uptake of nutrients or oxygen (solute) across a cell wall (boundary). If the solute cannot penetrate the whole cell there will be a non uniform distribution of the solute thoughout the cell which will of course be highest at the boundary. We can solve this diffusion problem through a numerical analysis using a solver implemented in FORTRAN. This solution can then be visualized graphically and the result can be extrapolated to create a model of the above solute diffusion problem. The purpose is to understand and visualize the diffusion model through numerical solutions and the idea could be expanded to further applications by altering the boundary conditions and parameters of the differential equation in order to simulate other real world applications.

This problem was modeled by the non-dimensional semilinear parabolic boundary value problem $\frac{\partial(u)}{\partial(t)} = \Delta u - \frac{k_1 * u}{k_2 + u}$ on the rectangular domain $\Omega = [0,1] * [0,1]$. The boundary conditions are as follows:

$\frac{\partial u}{\partial x} = 0$ at x=0
$\frac{\partial u}{\partial y} = 0$ at y=0
u=1 at x=1, y=1

This was solved in Part 3. Parts 1 and 2 can be treated as special cases of Part 3.

Part 1: Non-dimensional linear elliptic boundary value problem

The problem presented in part 1 is to solve a non-dimensional linear elliptic boundary value problem given by

$$\Delta u = ku$$

on the rectangular domain $\Omega = [0,1] \times [0,1]$ with boundary conditions given as follows

$\frac{\partial u}{\partial x} = 0$ at x=0
$\frac{\partial u}{\partial y} = 0$ at y=0
u=1 at x=1
u=1 at y=1

Compiling

Enter the following in the terminal to compile the code:

```
gfortran -fdefault-real-8 M6020-P1-VincentEmond.f90
M6020-P1-CG-VincentEmond.f90
```

Numerical Methods

The problem was organized such that the cell who's diffusion we were simulating was represented on the rectangular domain $\Omega = [0,1] \times [0,1]$ as a square grid containing g grid points per side on the domain. This gave a density of grid points (unit length/grid point) along each axis of h=$\frac{1}{g}$. The grid was arranged as follows where g_0 lies at the boundary 0 on the domain and g^2 lies at the other boundary, 1, on the domain.

g	2g	3g	...	g^2
...	...	...	...	...
2	g+2	2g+2	...	(g-1)g+2
1	g+1	2g+1	...	(g-1)g+1

Recall: g= number of grid points along each boundary of the domain such that we have a gxg grid. Organizing the grid in this way allows for a grid point to be identified using only one index p, rather than two indeces i and j. ie the point p=1 in the grid above corresponds to i=1, j=1, likewise p=2 corresponds to i=1,j=2. This simplifies the use of an iterative solver whereby we can solve for the values of each grid point and only iterate through one index instead of requiring two indeces. To solve the problem the value of the differential equation which served as our model had to be determined for each grid point. For points p that did not lie along the boundary, (g¡p¡(g-1)g and p $\neq$ 0 or 1 (mod g)) their value, u, can be approximated using the following equation,

$$\Delta u = ku = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j-1} + u_{i,j+1} - 4ui, j}{h^2} \tag{1}$$

where we use the i,j indeces for ease of visulization. What we are doing is using the values, u, of surrounding points to approximate the value of the differential equation at the point ij, or equivalently p. For points along a boundary, the value u of points p was approximated by modifying equation 1 appropriately with Dirichlet and/or Neumann conditions. The modifications must be used because otherwise the approximation given by eq 1 would involve the use of points outside of the domain. In our case we have an interior boundary and an exterior boundary. Across the interior boundary there exists part of our solvent, in this case part of the cell as our boundary conditions dictate along x=0 and y=0. Across an exterior boundary the solute (oxygen or nutrients) exists and there is no cell (recall we are looking at a quadrant of a cell) and so u=1 along x=1 and y=1.

The modifications can be treated generally as follows: If the point is on the interior boundary (the bottom row or left column) then we use the Neumann Boundary Condition. We have that $\frac{\partial u}{\partial x} = 0$ at the boundary. Let's call the point outside the domain r and look at this in one dimension. We take a Taylor series for the values of points r, and p about the boundary.

$$u_p = f(\tfrac{h}{2}) = f(0) + \tfrac{h}{2}f'(0) + (\tfrac{h}{2})^2 f''(0) + O^3$$
$$u_r = f(\tfrac{-h}{2}) = f(0) - \tfrac{h}{2}f'(0) + (\tfrac{h}{2})^2 f''(0) + O^3$$
$$u_p - u_r = f'(0) = 0$$
$$u_p = u_r$$

We then sub in u_r into equation 1 such that it does not depend on any points outside of the domain.

By similar argument but with u=f(0)=1 (the value of u on the boundary) in the Taylor expansion and adding $u_r and u_p$, we can show for points along the exterior boundary we use Dirichlet boundary conditions given by

$$u_{boundary} = f(0) = 1$$
$$\frac{u_p+u_r}{2} = f(0) = 1$$
$$u_r = 2 - u_p$$

and similarly we substitute u_r in equation 1 so there are no points outside of the domain being evaluated.

To facilitate computation the points on the grid could be represented with a matrix which could be stored in diagonal format in a matrix A with 5 columns and g^2 rows where ioff=(-g,-1,0,1,g). Here is an example matrix for g=4, ioff=(-4,-1,0,1,4).

$$\begin{bmatrix} 0 & 1 & -2-h^2k & 1 & 1 \\ 0 & 1 & -3-h^2k & 1 & 1 \\ 0 & 1 & -3-h^2k & 1 & 1 \\ 0 & 1 & -4-h^2k & 1 & 1 \\ 0 & 1 & -3-h^2k & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & -5-h^2k & 1 & 0 \\ 1 & 1 & -6-h^2k & 1 & 0 \end{bmatrix}$$

This allows us to arrange our problem to compute Ax=b. The rhs could be determined based on boundary conditions. Any point along a boundary with Dirichlet conditions would have a value of $b_p = -2$ per boundary. So for the above example with g=4 the rhs had nonzero values

$$b_p = -2 \text{ for p=4,8,12,13,14,15 and}$$
$$b_{16} = -4.$$

With this we are ready to solve the problem. To do this we use an iterative solver to solve Ax=b by using the approximations outlined in equation 1 and the modifications that occur at the boundary. The solver will use values of surrounding points and boundaries to approximate values for each point p. It will continue this approximation until the difference in the value of the points for each iteration becomes small. In our case a change smaller than 1e-12 was the tolerance set for the solver to quit iterating.

Algorithm and Matrix Construction

Below is some sample code for the general construction of the matrix A.

```
    !main diagonal
!initialize for no boundary conditions
do i=1,n
p(i)=-4-k*h**2
```

```
enddo

    !compute main diag based on grid position

    do i=1,n

    !top row in grid
if (mod(i,g)==0) then
p(i)=p(i)-1
endif

    !left boundary in grid
if (i.LE.g) then
p(i)=p(i)+1
endif

    !right boundary in grid
if(i.GT.n-g) then
p(i)=p(i)-1
endif

    !bottom boundary in grid
if(mod(i,g)==1) then
p(i)=p(i)+1
endif

    enddo
```

Implementing the off-diagonal terms is trivial and can be viewed in the provided source code. The rhs was constructed as follows:

```
    !generate rhs vector b
!initialize values to 0
do i=1,n
rhs(i)=0
enddo

    !calcuate values of b for different boundary conditions
do i=1,n
if (mod(i,g)==0) then
rhs(i)=-2
endif

    if (i¿n-g) then
rhs(i)=-2
endif
```

```
    if (i==n) then
rhs(i)=-4
endif
enddo
```

Here n is the total number of grid points, n=g*g.

Solvers

The choice of solver is dependent on the problem being solved and on the properties of the matrix being used by the solver. The matrix representing the grid is a symmetric matrix ($A = A^t$) which was stored diagonally to save memory and lower computation requirements. The matrix is also diagonally dominant as it satisfies

$$|a_{ii}| \leq \sum_{i \neq j} |a_{ij}| \text{ for all } i.$$

It can also be shown using the Gershgorin circle theorem that the matrix is positive definite. Let

$$R_i = \sum_{i \neq j} |a_{ij}|$$

be the sum of the absolute values of non-diagonal elements in the i-th row.

$$\{D_i = z \, \epsilon \, \mathbb{C} : |z - a_{ii}| \leq R_i\}$$

Where D_i is a Gershgorin disc. For each entry on the main diagonal the disc will never reach the negative domain and as such the matrix is positive definite.

We have shown our matrix to be symmetric, positive definite, and diagonally dominant and as such the conjugate gradient and biconjugat gradient stabilized methods make for good solvers. As we will see below these methods give similar results, and differ slightly in number of iterations and CPU time.

Results

We use the Conjugate gradient method as it converges more quickly than BiCGSTAB, later we will compare the results of using the BiCGSTAB method and other methods.
Below is a table outlining the CPU time and Number of iterations needed to solve the system for different grid sizes (different g), and different values of parameter k.

Figure	g	k	Iterations	CPU Time (sec)
1	20	0.01	83	0.9000000E-02
2	20	1	83	0.1200000E-01
3	20	10	78	0.1800000E-01
4	20	100	55	0.1800000E-01
5	20	1000	21	0.4000000E-02
6	100	0.01	368	0.3780000E+00
7	100	1	364	0.4140000E+00
8	100	10	344	0.3500000E+00
9	100	100	237	0.2460000E+00
10	100	1000	86	0.9700000E-01
11	500	10	1520	0.4561400E+02

Visualizations

Below are the visualizations of the grids corresponding to the above computations.

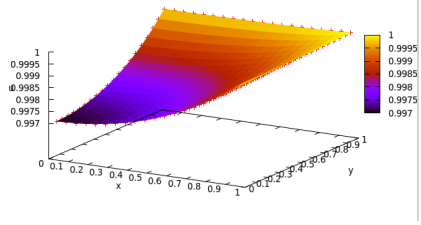


Figure 1: g=20, k=0.01

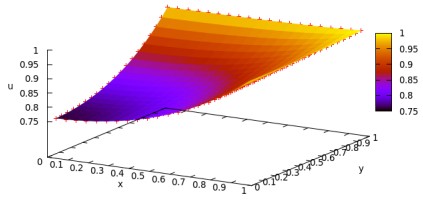


Figure 2: g=20, k=1

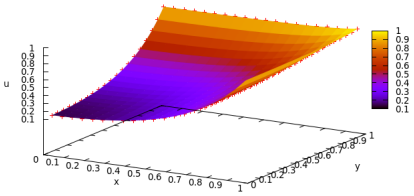


Figure 3: g=20, k=10

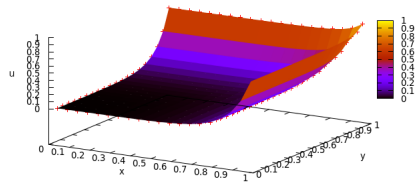


Figure 4: g=20, k=100

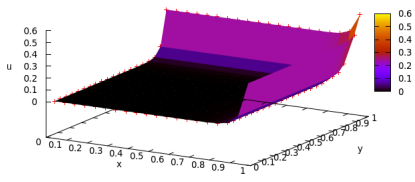


Figure 5: g=20, k=1000

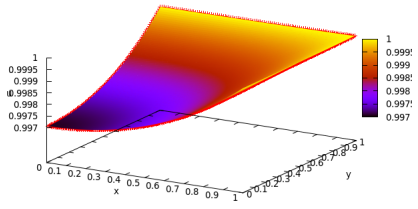


Figure 6: g=100, k=0.01

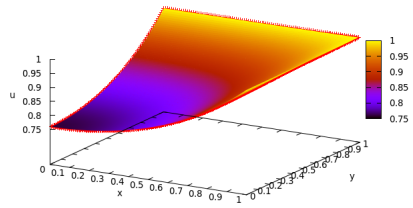


Figure 7: g=100, k=1

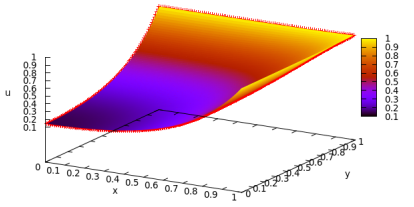


Figure 8: g=100, k=10

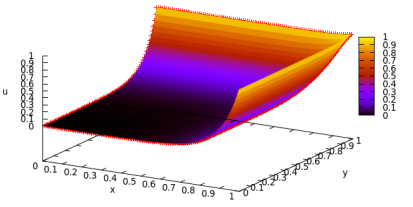


Figure 9: g=100, k=100

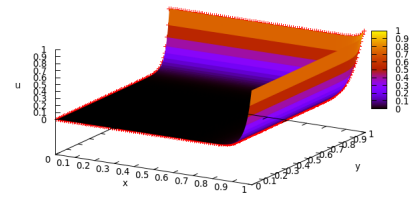


Figure 10: g=100, k=1000

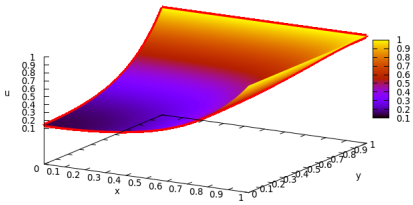


Figure 11: g=500, k=10

The lower density grid (g=20) lacked much of the detail present in the finer grid. The CPU time is obviously shortest for the lowest density grid, but it lacks detail. The computation time however for the finest grid (g=500) is very high and the detail is not much more than the g=100 grid, which has a fast computation time. As such a grid with g=100 is the best choice and will be used throughout.

Convergence

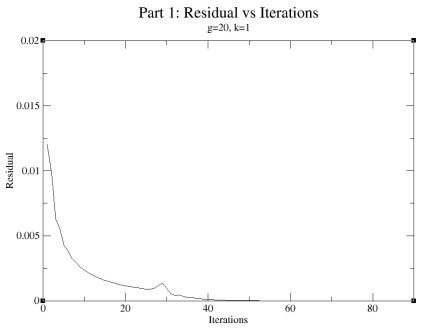
Convergence is tested by observing the behaviour of the update and residual as the solver iterates. The difference between the k-th and k+1 iteration should become small until reaching a defined tolerance at which point it causes the iteration to stop and a solution is reached. The residual is defined as

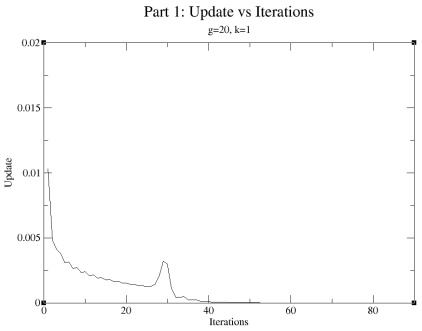
$$r_k := b - Ax_k$$

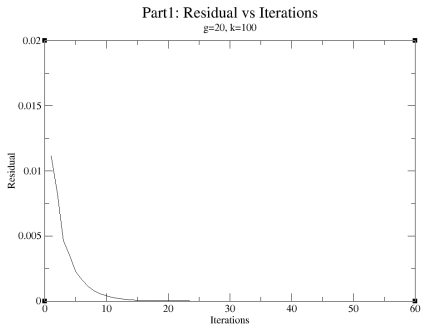
and the update is defined as:

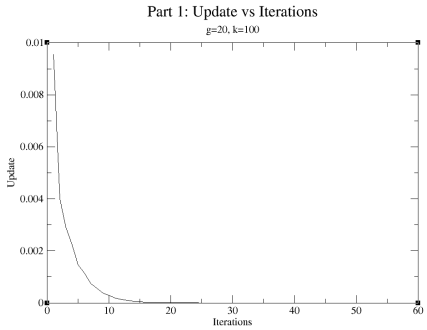
$$update = x_k - x_{k-1}$$

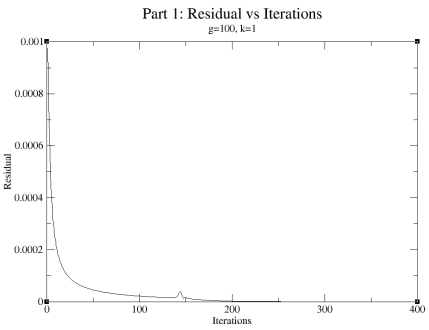
For each case the residual and update tend to zero, indicating that solutions converge. Below are plots for the residual and update for two cases (k=1 and k=100) of each value of g (g=20 and g=100) tested. The other cases behave similarly.

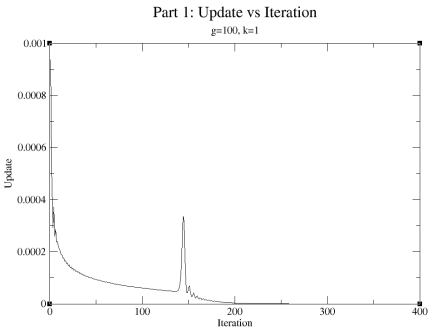


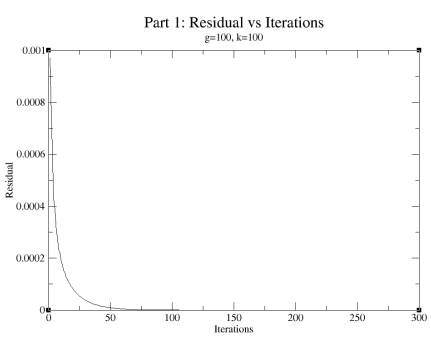


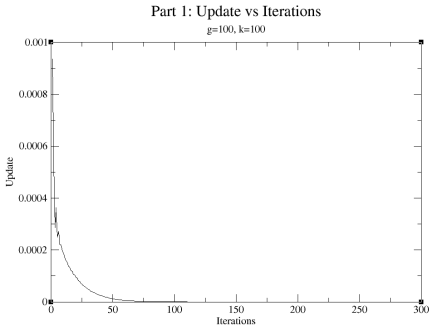










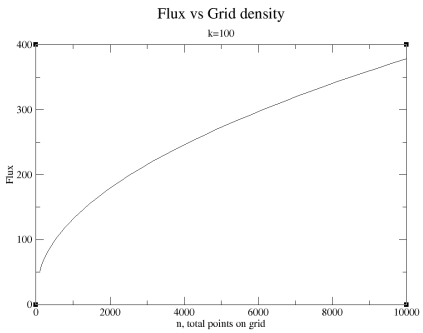


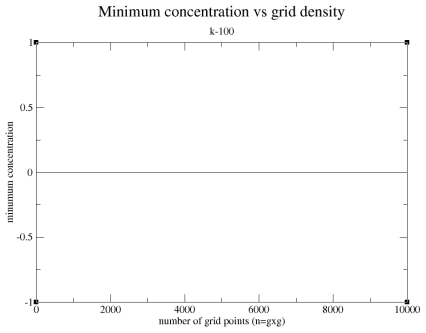
Flux and Concentration

The flux was measured on the exterior boundaries, those abiding to Dirichlet Conditions where u=1. We choose these boundaries as we know the flux is zero on the interior boundary which follow Neumann Condition. The flux was computed using the following:

$$flux = 4 \sum_{i=1}^{n} 1 - u(i)$$

on those Dirichlet boundaries, where i represents a grid point. The following figures were created by setting k=100 and increasing the density of the mesh (increasing n where n=gxg) where g went from g=10 to g=100. We also plot the minimum concentration in the grid as a function of the grid point density for k=100. Unsurprisingly this value remains constant for increasing density of grid points on the domain.





We see the flux increases as the grid becomes finer and there are more points in the domain. This trend continues as g continues to grow, and the mesh becomes more fine.

Part 2: Semilinear Elliptic Boundary Value Problem

The following semilinear elliptic boundary value problem

$$\Delta u = \frac{k_1 u}{k_2 + u} \tag{2}$$

with boundary conditions

$$\begin{gathered} \frac{\partial u}{\partial x} = 0 \text{ at x=0} \\ \frac{\partial u}{\partial y} = 0 \text{ at y=0} \\ \text{u=1 at x=1} \\ \text{u=1 at y=1} \end{gathered}$$

was solved with a finite difference approximation.

Compiling

Enter the following in the terminal to compile the code:

```
gfortran -fdefault-real-8 MA6020-P2-VincentEmond.f90
```

Numerical Methods:Finite Difference Approximation

Taking equation 1 and equation 2 we write

$$\Delta u = \frac{k_1 u}{k_2 + u} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j-1} + u_{i,j+1} - 4ui, j}{h^2} \tag{3}$$

From this we are able to equate each grid point with a corresponding quadratic equation which can be approximated and the approximation can be iterated until a stopping criteria is reached. The stopping criteria used was the norm of the update.

$$||x_k - x_{k-1}|| \leq tol||x_k||$$

It is worth noting that there are other methods by which this problem could be solved (eg. Newton's Method, Non Linear Gauss Seidel).

Algorithm

In our case we use a direct solver to arrive at the solution. The values of u at each grid point is approximated using the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

and equation 2. The algorithm assigns the values of a, b, and c for each point, and then solves each and iterates through this solution until stopping criteria is reached. The values of a,b,c were generalized for interior points, and special cases were made for those points lying on a boundary and those in corners of the grid. Below is some sample code for the evaluation of the grid.

```
!Here we calculate the values of the grid points which depend on boundary conditions
!Assign values of a,b,c which are used with quadratic equation

do i=1,n

!inner points
if (i¿g .and. mod(i,g)/=0 .and. mod(i,g)/=1 .and. i¡n-g) then
a(i)=-4.
b(i)=x(i+1)+x(i-1)+x(i+g)+x(i-g)-4*k2-h*h*k1
c(i)=(x(i+1)+x(i-1)+x(i+g)+x(i-g))*k2
x(i)=(-b(i)-sqrt(b(i)**2-4*a(i)*c(i)))/(2*a(i))
end if

!i=1, bottom left bound
a(1)= -6
b(1)=x(2)+x(1+g)-2*k2-h*h*k1
c(1)=(x(2)+x(1+g))*k2
x(1)=(-b(1)-sqrt(b(1)**2-4*a(1)*c(1)))/(2*a(1))

!top left bound i=g
a(g)= -4
b(g)=2+x(g-1)+x(g+g)-4*k2-h*h*k1
c(g)=(x(g+g)+x(g-1)+2)*k2
x(g)=(-b(g)-sqrt(b(g)**2-4*a(g)*c(g)))/(2*a(g))

!bottom right i=n-g+1
a(n-g+1)= -4
b(n-g+1)=2+x(n-g+2)+x(n-g+1-g)-4*k2-h*h*k1
c(n-g+1)=(x(n-g+2)+x(n-g+1-g)+2)*k2
x(n-g+1)=(-b(n-g+1)-sqrt(b(n-g+1)**2-4*a(n-g+1)*c(n-g+1)))/(2*a(n-
g+1))

!top right i=n
a(n)= -6
b(n)=4+x(n-1)+x(n-g)-6*k2-h*h*k1
c(n)=(x(n-1)+x(n-g)+4)*k2
x(n)=(-b(n)-sqrt(b(n)**2-4*a(n)*c(n)))/(2*a(n))

!top row B.C.
if (mod(i,g)==0 .and. i/=g .and. i/=n) then
a(i)=-5
b(i)=2+x(i-1)+x(i+g)+x(i-g)-5*k2-h*h*k1
c(i)=(x(i-1)+x(i+g)+x(i-g)+2)*k2
x(i)=(-b(i)-sqrt(b(i)**2-4*a(i)*c(i)))/(2*a(i))
endif

!bottom row B.C.
```

```
if (mod(i,g)==1 .and. i/=n-g+1 .and. i/=1) then
a(i)=-3
b(i)=x(i+1)+x(i+g)+x(i-g)-3*k2-h*h*k1
c(i)=(x(i+1)+x(i+g)+x(i-g))*k2
x(i)=(-b(i)-sqrt(b(i)**2-4*a(i)*c(i)))/(2*a(i))
endif

!left boundary
if (i¡g .and. i/=1) then
a(i)=-3
b(i)=x(i+1)+x(i+g)+x(i-1)-3*k2-h*h*k1
c(i)=(x(i+1)+x(i+g)+x(i-1))*k2
x(i)=(-b(i)-sqrt(b(i)**2-4*a(i)*c(i)))/(2*a(i))
endif

!right boundary
if (i¿n-g+1 .and. i/=n) then
a(i)= -5
b(i)=2+x(i-1)+x(i+1)+x(i-g)-5*k2-h*h*k1
c(i)=(x(i+1)+x(i-1)+x(i-g)+2)*k2
x(i)=(-b(i)-sqrt(b(i)**2-4*a(i)*c(i)))/(2*a(i))
endif
end do
end do
```

As you can see, the coefficients of the quadratic equation were defined and used in calculating the value at each point in the grid.

Results and Visualizations

Below is a table of tests with different values of k_1 and k_2.

Figure	g	k_1	k_2	Iterations	CPU Time (sec)
12	100	10000	10000	64253	0.4805300E+02
13	50	1	100	19469	0.4412000E+01
14	100	100	1	4330	0.3688000E+01
15	50	100	100	16601	0.3494000E+01
16	100	1000	1000	64041	0.47866600E+02
17	100	1	1000	76327	0.6260500E+02
18	100	1000	0.1	156	0.1650000E+00
19	100	1000	10	3912	0.3473000E+01
20	100	10000	0.1	24	0.420000E-01

The visualizations of the above tests are shown below.

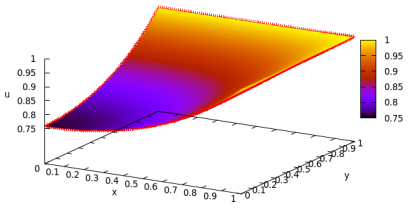


Figure 12: $k_1 = 10000$,$k_2 = 10000$

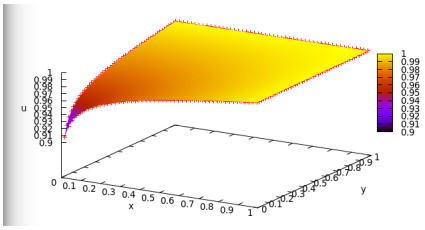


Figure 13: $k_1 = 1$,$k_2 = 100$

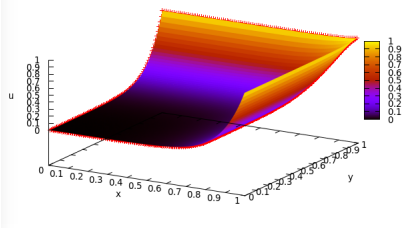


Figure 14: $k_1 = 100$,$k_2 = 1$

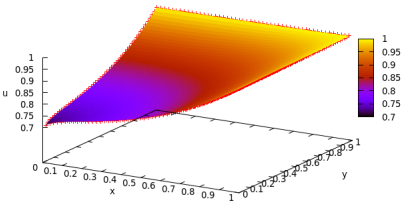


Figure 15: $k_1 = 100$,$k_2 = 100$

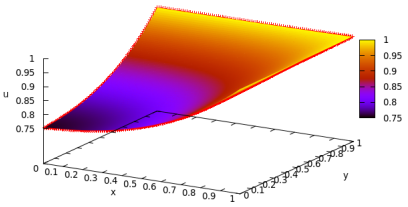


Figure 16: $k_1 = 1000$,$k_2 = 1000$

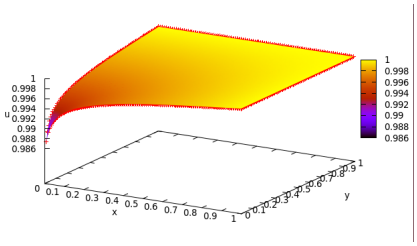


Figure 17: $k_1 = 1$,$k_2 = 1000$

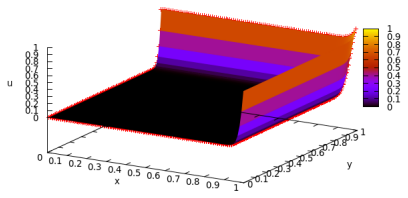


Figure 18: $k_1 = 1000$,$k_2 = 0.1$

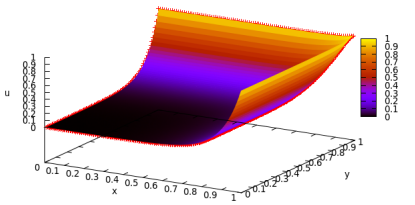


Figure 19: $k_1 = 1000$,$k_2 = 10$

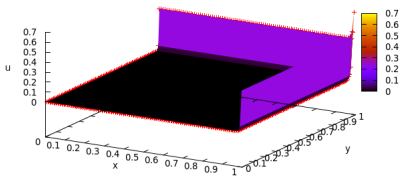
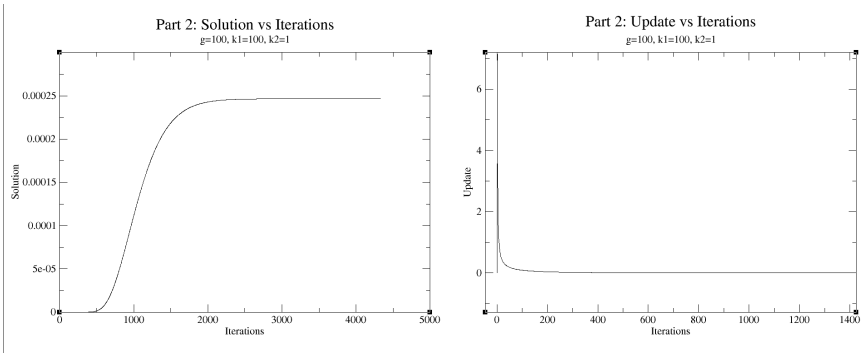


Figure 20: $k_1 = 10000$,$k_2 = 0.1$

Here we see the behaviour of the concentration gradient (the values of the points on the grid) as we change k_1 and k_2. The gradient can be compared to Part 1 through a comparison of equation 1 and equation 3. Figure 3 shows $k_1 = 100$ and $k_2 = 1$, here $k_2 >> k_1$ and as such the behaviour is similar to the k=100 case in part 1. On the other hand when $k_2 >> k_1$ (ie Figure 6) the concentration throughout the grid is very high and most points have values which are near unity. Conversely, with a very large $k_1 >> k_2$ the concentration is nearly 0 throughout the grid.

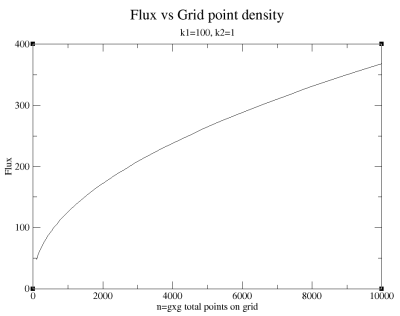
Convergence

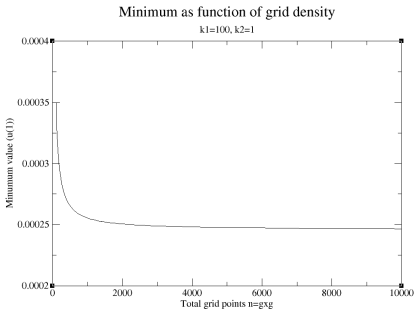
The convergence was confirmed by checking the behaviour of the solution and of the update with respect to the number of iterations. The update was covered in part 1, and the result is similar here, although the update approaches 0 very quickly. The solution also converges asymptotically to one value through the iterations. The plots are shown below for the case of figure 14 where g=100, $k_1 = 100$ and $k_2 = 1$. Other figures follow the same trend.



Flux and Concentration

The flux was calculated in the same fashion as in part 1. Similarly we calculate flux for g ranging from g=10 to g=100, but we use $k_1 = 100$ and $k_2 = 1$. We also observe the minimum concentration over the same g range, and values of k_1 and k_2.





The flux follows a similar trend as seen in part 1. It increases as g increases. The minimum concentration quickly decreases with increasing grid point density and asymptotically approaches a minimum value as g increases.s.

Part 3: Non-dimensional Semilinear Parabolic Boundary Value Problem

For the final part we are solving the nondimensional semilinear parabolic boundary value problem

$$\frac{\partial u}{\partial t} = \Delta u - \frac{k_1 u}{k_2 + u} \tag{4}$$

with the same boundary conditions and on the same domain as previous parts. The grid was initialized with and initial condition of u(0,x,y)=1.

Compiling

The code can be compiled using the following

```
gfortran -fdefault-real-8 MA6020-P3-VincentEmondTest.f90
MA6020P3-Solver-VincentEmond.f90
MA6020-P3-output-VincentEmond.f90
```

Numerical Methods

We use a Gauss-Newton algorithm which incorporates Newton's method. We can represent equation 4 in the following way:

$$Mdu = \Phi$$

$$(I - \frac{\Delta t}{2} J)\Delta u = \Delta t(Au - f(u) - b) \tag{5}$$

where M is a matrix, du is the change in u and is being solved for and updated throughout iterations. Φ is a scalar. As such we are solving the matrix problem $Mdu = \Phi$. In matrix M, J is the jacobi and we note that the spectral radius $\rho M < 1$ and as such the solution converges. In Φ matrix A, and rhs b are the same as in part 1.

So the solver will iterate and solve the state of the grid for each time step and we record the state of the system at that time step. The solver continues throughout the time steps until it converges on a solution. The animated visualizations will be sent along with the project files.

Algorithm and Solver

```
    !initialise: set tolerances, max no iterations and initial guess
err1=1e-10; err2=1e-10; nit=n*100

!Assign values of k, k_1 and k_2
k=1000.
k1=100.
k2=0.01
h=1/real(g)

!Assign the time step and initialize time, t=0
dt=h/1000.0
t=0.0

!initialize the grid
dx=1.
x=1.
xold=1.

!v=Φ v=1.

!Set tolerance
tol=1e-12
err3=1

!(1) setup a test case: prescribe solution sol
! and generate a matrix A and rhs, such that A*sol=rhs

!set matrix
call genDIAG(n,ndiag,ioff,A,rhs,sol,g)

rhs=rhs/h**2
A=A/h**2

!initialize counters and number of iterations ctr=0
nitr=0
```

```
do while (stopcrit==0)

!set matrix E
E=A
E(:,3)=E(:,3)-((k1*k2)/(k2+x)**2)

!Set matrix M M=0
M(:,3)=1.
M=M-(dt/2)*E

!set v (capital phi)
call amuxd(n,x,v,A,ndiag,ioff)
v=dt*(v-(k1*x)/(k2+x)-rhs)

!call solver
err1=1e-10
err2=1e-10

call solveLSDIAG(n,ndiag,ioff,M,dx,v,nit,err1,err2,stopcrit)

!update soln
xold=x
x=x+dx
t=t+dt
stopcrit=0
nitr=nitr+1
if (norm2(dx)¡tol) stopcrit=-10
enddo
```

Results and Visualiaztions

Below are results of a few tests performed with varying k, $k_1, and k_2$ values. The grid density was reduced such that g=50 due to long compiling times.

Fig	g	k	k_1	k_2	Iterations	CPU time (sec)
21	50	1000	100	10	79154	110.477
22	50	1000	100	1	13271	18.85999
23	50	1000	1000	0.1	460	0.85

The files relating to these tests are contained in the projet folder. They are named Figure21.gif, Figure22.gif, and Figure23.gif. These were compiled with BiCGSTAB, but compiling with CG yielded the same results differing only in CPU time. Below are the final states to which the solution converges for the above animations available in the project files sent.

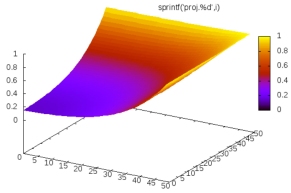


Figure 21: g=50, k=1000, $k_1 = 100$, $k_2 = 10$

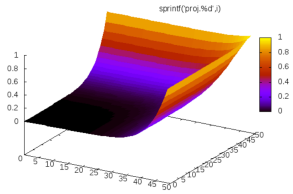


Figure 22: g=50, k=1000, $k_1 = 100$, $k_2 = 1$

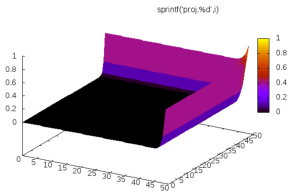


Figure 23: g=50, k=1000, $k_1 = 1000$, $k_2 = 0.1$