Project B- Diffusion

Scientific Computing

Miguel De Armas

1351046

**Abstract**

The diffusion equation is a parabolic PDE that shows how something spreads more widely. It requires boundary conditions and an initial condition for it to be solved. This report shows how the diffusion equation can be solved using two methods: the explicit method and the Alternating Direction Implicit (ADI) method. The report provides detailed explanations on how to solve each method, furthermore; the report contains also MATLAB codes. Grid convergence and error analysis is provided as well. The diffusion equation that is being solved is in equation (1). It is in 2D form which proves to be more of a challenge than 1D. The boundary conditions are in equation (2),(3),(4). Finally, the initial condition is in equation (5).

Boundary Conditions

(2) (3)

Initial Conditions

(5)

**Discretization**

Discretization is an important tool when trying to solve PDEs. It involves using Taylor Series and approximates values to some degree. The approach to the problem is important though, there may be times where the problem because unstable and may blow up the results. The methods used in this report is the Explicit Method along with the ADI method. It is interesting because without the Explicit method, the ADI method would not exist.

**Explicit Method Discretization**

As cited in the book, the explicit method uses the forward difference for the first derivative with an order of.

For the side with respect with x and y, the second derivative centered difference is used with an error of and.

Since the partitions are equal in this case (=, the stability of the problem becomes

After simplification, the discretization becomes,

**ADI Method**

This method makes life so much easier since other methods require a penta-diagonal matrix which is something that is very confusing. The ADI method reduces the matrix into two tridiagonal matrices which were dealt with all semester. This method is unconditionally stabile and second order in time and space. The first step (n+1/2) is explicit in y and implicit in x, as follows,

Since (=, the method reduces to

The second step (n+1) is explicit in x and implicit in y,

**Algorithm and Code**

Explicit Method

First chose the number of nodes. Remember, the more nodes, the more accurate the results. The downside is the increase in computations.

N=50

Make partitions according to the given domains.

The time domain can be calculated as

Now the time vector and space vectors can be created.

Plug in the vectors x and y into the boundary conditions to get the right hand side values.

&

Now plug in the initial condition

For the loop, the y index starts at 1 while the x index starts at 2.

A more detailed look will be in the results area.

ADI Method

Same beginning as the Explicit method, chose N and make your partitions. However the time domain will be different.

Now establish your time and space vectors like the explicit method.

Insert the y vector into the boundary conditions to compute a portion of the right hand side of the equation. The initial condition will be imposed on the interior points on the grid.

To make computations faster and cleaner, we will define our constants before putting everything together,

Now the tridiagonal function will be used to find solutions implicit to x for the first half step.

e=[-a\*ones(1,N-2) 0];

f=[ 1 b\*ones(1, N-2) 1];

g=[ 0 -a\*ones(1, N-2)];

The Dirichlet BC are input in the diagonals of the matrix.

x=tridiag(e,f,g,r)

Now implicit in y for the second half step.

e2= [-a\*ones(1, N-2) -2\*a];

f2= b\*ones(N);

g2= [0 -a\*ones(1,N-2)];

r2= zeros(1,N);

The Neumann BC are input in the diagonals of the matrix.

tridiag(e2,f2,g2,r2)

Now we take explicit in y, then in x.

for q=1:lt %time loop

for j=1:lx %goes through loop and takes explicit in y with proper boundary conditions

for i= 2:lx-1

if j==1

r(i)=c\*U(i,1)+2\*a\*U(i,2); %the ghost node doubles the j+1 value

elseif j==lx

r(i)= a\*U(i,j-1) +c\*U(i,j);

else

r(i)=a\*U(i,j-1)+c\*U(i,j)+a\*U(i,j+1);

end

end

r= [U(1,j) r(2:lx-1) U(lx,j)]; %The Dirichlet boundary conditions are imposed

x= tridiag(e,f,g,r);

U(:,j)=x; %U(n+1/2) is dtermined by combining explicit in y and implicit in x

end

%%Now comes the second half step, explicit in x and implicit in y

for i=2:lx-1

for j=1:lx

r2(j)=a\*U(i-1,j)+c\*U(i,j)+a\*U(i+1,j);%explicit solution in x

end

U(n+1/2) is determined by combining implicit x and explicit y.

U(:,j)=x; %U(n+1/2) is dtermined by combining explicit in y and implicit in x

U(n+1) is determined by combining implicit y and explicit x.

U(i,:)=x2; %combining explicit in x and implicit in y

**Verification**

To make sure the proper procedure for the method was followed, verification must be done. In this project, verification was completed by a few ways: completing a steady state graph, showing grid convergence, and comparing the both methods used. The steady state plot was completed by using a centralized point and plotted it against time. As shown below, the graph does appear to reach steady state values. The grid convergence was completed by comparing the values of L number of nodes compared to 2L number of nodes. It truncates the minimum difference and determines the best spot for the numerical solutions.

**Results**

**Grid Independence**

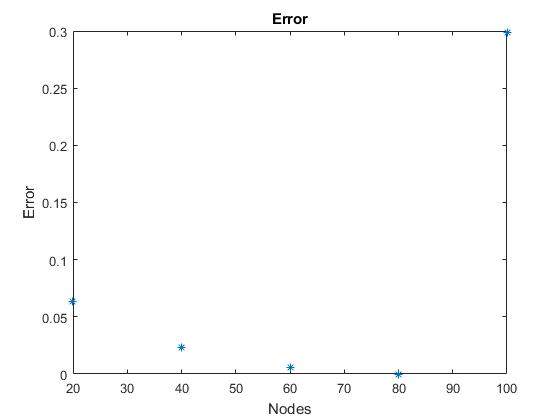
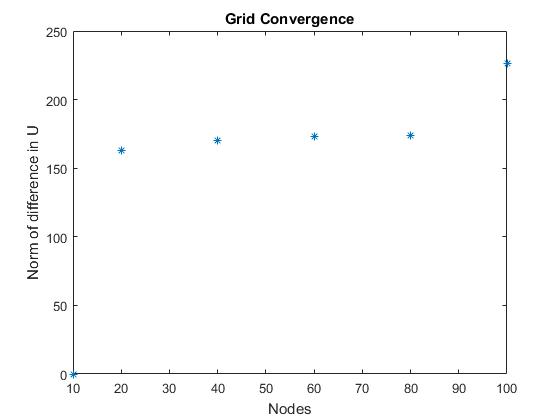


Figure 2: Explicit error

Figure 1: Explicit Grid convergence

**Graphs of Methods**

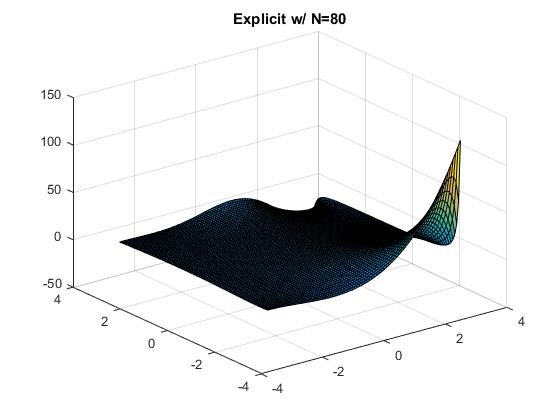
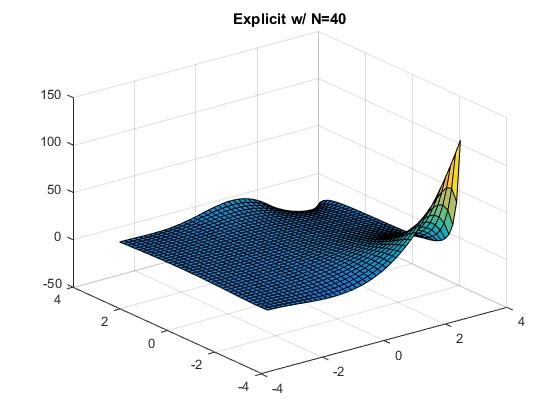


Figure 3 :Explicit Figure 4: Explicit

As above, the more nodes there are, there finer the resolution is.

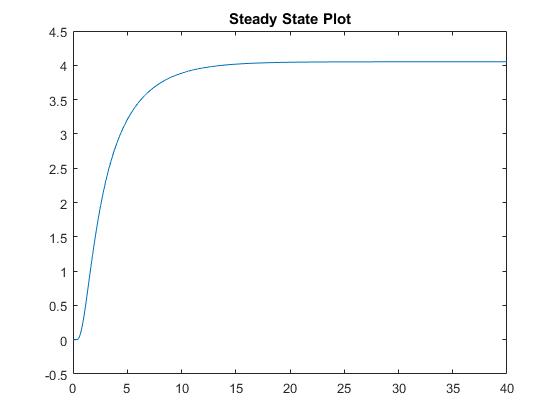
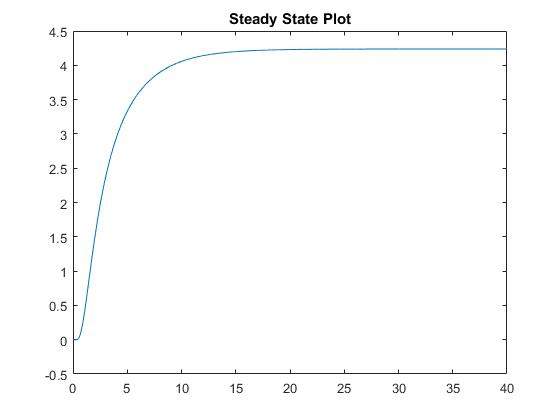


Figure 5 : Explicit Figure 6: Explicit

Both reach a steady state point around the same number.

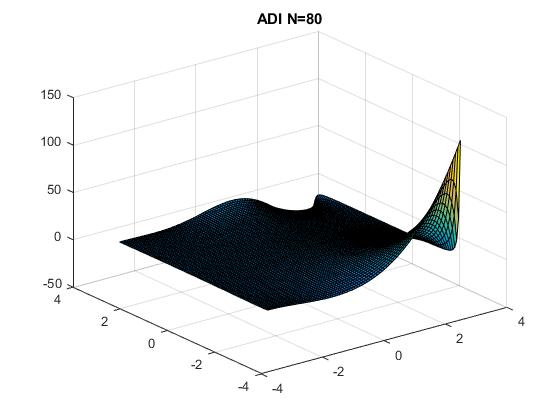
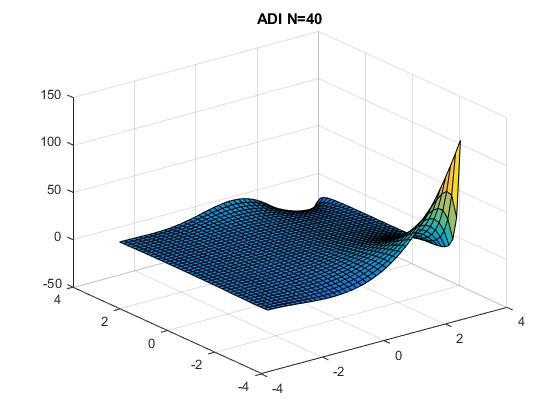


Figure 6: ADI Figure 7: ADI

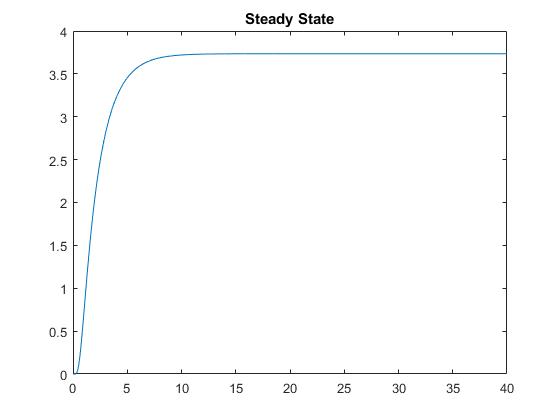
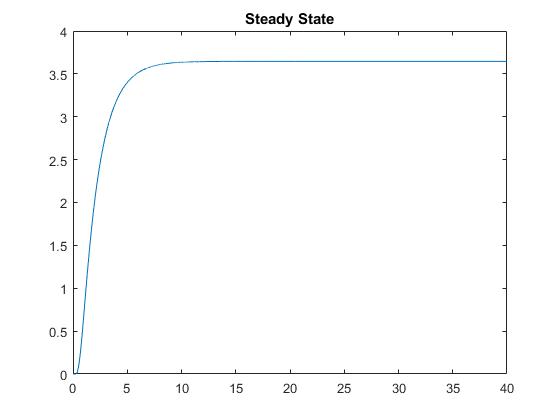
 

Figure 8: ADI Figure 8: ADI