**Abstract**

The diffusion equation is a fundamental equation in physics, mathematics, and many other fields such as life science, etc. This equation can model the behavior of micro particles that results due to the random movement of each particle. For example, a specific form of the diffusion equation is the famous heat equation. This very important equation can model the temperature within a surface given the right initial conditions and outside disturbances. This makes solving the diffusion equation a very important matter. However, due to its nature, it is very difficult to find an analytical solution to the diffusion equation. Instead, a discretized version of it is solved. This leads to an approximate solution close to the actual solution. This report documents the steps and results of a project that attempts to model the diffusion equation with a specific set of boundary and initial conditions. The purpose is to see the effect of using two different discretization of the diffusion equations when finding a solution. This includes looking at the computational cost, the accuracy, and the convergence of its discretization. This report will begin by introducing the specific diffusion equation to be solved. Next, the two discretized versions of the equation will be discussed. This will be followed by a description of the numerical method used to solve each version along with a description of the code being used. The code was run on MATLAB. To give the reader a better idea of hardware required to run such simulations, the technical specifications of the computer used to perform the simulations will be included. Finally, the results of the simulations will be shown and discussed.

**Mathematical Statement of the Problem**

The diffusion equation that will be studied in this report is as follows:

**Initial Condition:**

=

**Dirichlet Boundary Conditions:**

**Neumann Boundary Conditions:**

**Discretized version of the equation**

For this study, the two chosen methods were the explicit method discretization using the forward difference in time and the centered difference in space, and the Crank-Nicolson method. These two methods were chosen because their numerical solution approach, as well as criteria for convergence, vary extremely. The purpose is to see the difference between two methods at opposite ends of the spectrum. The explicit method is very easy to implement, but it has a stringent criterion for stability. The Crank-Nicolson method is extremely powerful, but its implementation and computation complexity begins to be non-efficient when applied to higher dimension partial differential equations, such as the 2D diffusion equation being discussed.

**Explicit Method Description and Discretization**

The explicit method discretizes the time domain using the following forward difference approximation:

This gives the time domain an error of order which is not very convenient but still suitable.

However, the spatial dimensions are discretized using the centered difference formula for the second derivative:

This gives both discretization a second order of accuracy.

Once the explicit discretization of each dimension is fit back into the diffusion equation, the solution for the next time step becomes the following:

Looking at the equation, it can be seen that the explicit method is very straightforward. It only has one unknown for each node on the surface at hand. To put it simply: At a specific node, the solution can be found by using the solution around the node, including the node itself, at a time step below it. This makes solving the explicit method a trivial matter if given an initial solution. However, there is a caveat with this method. According to the Von Neumann Stability analysis for this method, the time and spatial step sizes must fit the following requirement:

Any combination of step sizes that do not satisfy that criteria would cause this method to break down.

**Numerical Approach to the Explicit Method**

Implementing the solution to the explicit method in MATLAB is an easy task. First, the step sizes in both the x and y direction were taken to be equal. This made the Von Neumann stability criterion more convenient to meet:

For this study, the number of nodes, N, used changed to see the effects of increasing or decreasing the mesh size. The relationship between the amount of nodes and the step size is:

The boundary conditions had to be carefully placed into the matrix which contains the solution, U. The following psuedocode demonstrates how the boundary conditions were placed, assuming j starts at one, and k starts at one:

U = Zeros(N,N)

U(N,:) =

U(1,:) =

The Neumann boundary condition, due to its nature, was not initialized in the initial vector. Instead, this boundary condition played an important role in the for-loop that would iterate through the solutions at each time step. Instead of only iterating through the inner nodes, the for-loop iterated through the left boundary as well. This is the boundary with the Neumann condition. To deal with this, a ghost node was “created” to the left of the left boundary. But points cannot exist past a boundary, which is why they are called ghost nodes. Instead, the centered difference formula for the first derivative was used to approximate the ghost node in terms of the interior nodes. This leads to the following relationship:

Therefore, the ghost node (which would be at j-1) is approximately:

However, since for this particular case, the approximation simplifies to:

This makes implementing this method even simpler.

Ultimately, the algorithm for finding the solution at a time step, tn+1, follows the following pattern:

if j at Neumann boundary

do

else if j at interior node

do

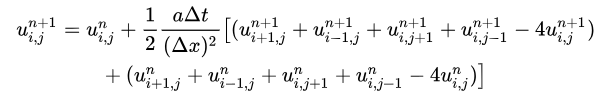
This would run until satisfactory results. The criteria for a satisfactory result is when the next set of solutions is not very different from the current of solutions.

**The Crank-Nicolson Method Description**

The Crank-Nicolson Method is an implicit method in time, and is second order accurate in both space and time. First, the time dimension is discretized at a time step tn+1/2 by the following:

Then, the space dimensions are also found that the same time steps. However, this is done in a not so simple way. The average of the current time step and the next time step are taken to approximate the time step in between them, as follows:

This is repeated for the y dimension. When combined together, and setting the change in x to be the same as the change in y, the discretization becomes:



a = 1 in this case. According to the von Neumann analysis, this discretization will always be stable. This means the time step does not have a stringent relationship with the spatial steps. This is fortunate for those trying to solve a 1D equation. But for a higher dimension system, this method begins to become immensely complicated for computers to solve. Especially with small mesh sizes.

As it can be seen, this method will have five unknown nodes, all located at the next time step. This is the reason why it’s called an implicit technique. The solution requires a system of equations to be solved first in order for the next time step to be solved. This contrasts with the explicit method, which was discussed in the previous section. The system of equations, however, leads to a 5 band matrix. Having a banded matrix is normally a good thing, but only when it’s tri-diagonal. In the case of a tri-diagonal system, the Thomas algorithm can efficiently solve for the unknown nodes. However, increasing the diagonals to 5 makes the system much more difficult to solve.

**The Crank-Nicolson Method Algorithm**

To be able to implement the Crank-Nicolson method in MATLAB, first two things have to be initialized. The coefficient matrix, which will be called LHS, and the solution matrix, which will be called RHS.

The coefficient matrix requires some knowledge about the general patterns for the coefficients of this specific diffusion equation. For a 5 band matrix of this nature, having three Dirichlet boundary conditions at the bottom, top, and right edge, and having a Neumann condition on the left edge, the following pattern was found when developing LHS. First, assume the mesh has N nodes. The matrix will be a square matrix of size (N-1)\*(N-2). The main diagonal of the matrix will contain the following coefficient:

Next, at every row, the column that is N-1 columns away from the main diagonal on every side will have the following value:

But this only begins to occur after the Nth row has been reached.

Next, the Dirichlet boundary conditions must be taken into account. At every (N-1)’th row, the super diagonal (which is the diagonal above the main diagonal) will have a 0. This is to take into account that one of the values on the left hand side, which are usually unknown, will be known. The following code implements these rules:

%This sets up with 5 Band System

for j = 1:(N-2)\*(N-1) %rows

for k = 1:(N-1)\*(N-2) %columns

if (j == k) %begins creating the diagnols

LHS(j,k) = r; %Creates the main diagnol. R = dt/dx^2

if (j+N-1) <= (N-1)\*(N-2)%Checks to see if it can add to diagnol N-1 ahead of main diagnol

LHS(j,k+N-1) = -lam;

end

if (k ~=1) && k-(N-1) >0 %checks to see if it can add to diagnol N-1 behing main diagnol

LHS(j,k-(N-1)) = -lam;

end

if (k ~=1)

LHS(j,j-1) = -lam;

end

if k < (N-1)\*(N-2)

LHS(j,j+1) = - lam;

end

end

%This part takes into account Neumann conditions on left edge by

%checking if the code is at the left edge, which would modify the

%coefficients around the main diagnol

if j == 1+check\*N-1

LHS(j,j+1) = -2\*lam;

if j > 1

LHS(j,j-1) = 0;

LHS(j-1,j) = 0;

end

check = check + 1; %increments the check value once a Neumann row has been found

end

end

end

Once LHS has been created, RHS needs to be determined. RHS contains all of the known values associated with a node. This is normally the five values at a time step below the current nodes. But at every node near a boundary point, the boundary condition must be taken into account. Therefore:

If row == row above lower boundary

Add lower boundary to RHS (along with normal set of solutions)

If column == column before right boundary

Add right boundary condition

If row == row below upper boundary

Add upper boundary condition

If row == corner point

Add boundaries associated with corner point to RHS

With this pattern, a for-loop can easily be created to take all of the requirements into account.

Once both RHS and LHS are created, MATLAB is able to solve the matrices using the linsolve function. As previously stated, five band matrices do not have clean solutions to them. In fact, the point of choosing this method was to see how complicated it would be to implement it, along with how long it would take to run as compared to the explicit method with its stability criterion met. It was indeed seen throughout this project that the Nicolson method was not only much harder to implement, it takes much longer. This is largely due to the size of the coefficient matrix. As previously mentioned, for a grid of size N, the coefficient matrix comes out to be (N-1)\*(N-2)x(N-1)\*(N-2) square matrix. This is a huge memory waster since most of the elements in that matrix are zero.

**Technical Specification of Computer Used to Run Simulations**

The following specs are the specs of the computers used to run the simulation:



**Results**

The results from each discretization method came out to be similar to one another. This gave hope that the methods, as well as the implementation, were right. Of course, human bugs could always still be present, but when two different methods give the same results, it implies either both methods are right, are both happen to be wrong. Before moving on to the actual results, first the parameters, effects of different numbers of nodes, the CFL number and its effect on the Explicit method, and the grid convergence will be discussed.

**Parameters**

For the explicit method, the time step was made so that it fit the Von Neumann criterion:

The amount of nodes that each method went through was: 10, 20, 30, 40, 50, and 60. Past 60 nodes, the time it took to run each simulation became much longer without a worthwhile improvement in the results. This was especially true for the Crank Nicolson method.

The time duration for each method was set at 5 seconds. This number was chosen empirically. By this point, the solution had already converged to a very close approximate solution and any future iterations were seen as inefficient.

**Effect of Amount of Nodes on Discretization**

A low amount of nodes for either method made the solution not very accurate, but the general pattern of the mesh drawn by MATLAB could be seen. And due to the limitation of the computer, it was not possible to have too many nodes for the Crank-Nicolson method. At 150 nodes, the computer began lagging. At 200 nodes, the computer refused to run the code due to space issues. This makes sense due to the matrix being an extremely large matrix once the node reach over 100. The sweet spot for nodes was at the 60 mark.

**Grid Independence**

To determine the grid convergence, the values of the L2 norm at different amounts of nodes was taken to see the general pattern of how the norm value was increasing as the nodes increased.

**CFL number on the Explicit Method**

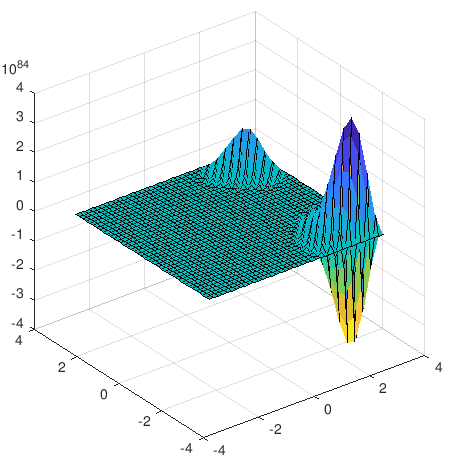


Figure 1

When the Courant-Friedrichs-Lewy condition is not met, the solution to the diffusion equation using the Explicit method breaks down. *Figure 1* shows what happens when the time step is set to equal the spatial step (d\_t = d\_x). The graph “blows up” in a way, with certain areas looking like mountains. Those are the areas where the failure most likely happened, causing the entire system to fail. The max number becomes an insane number.

**Explicit Discretization Solutions**

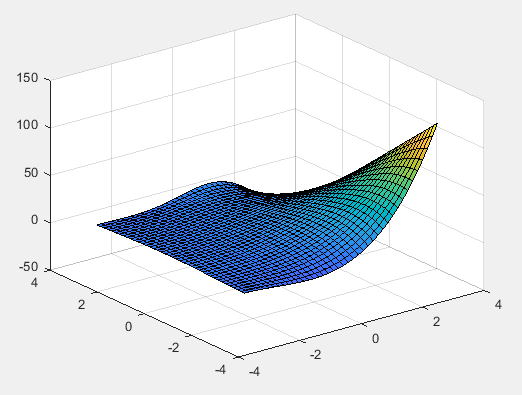
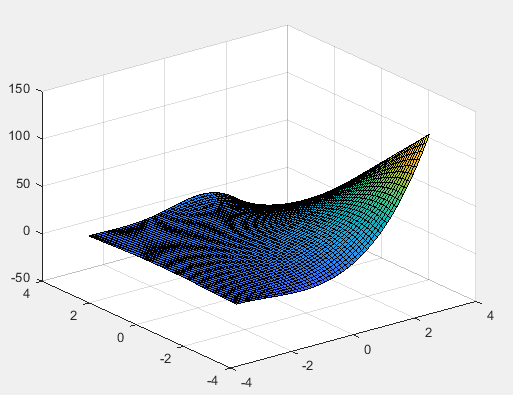


Figure 3-60 Nodes

Figure 2- 40 Nodes

*Figure 2* and *Figure 3* show the difference in results when the number of nodes are 40 and 60, respectively. As shown, apart from the darkened colors due to more nodes being used, the difference in the solutions is not noticeable. This was run for a time interval of 4 seconds. But what important aspect to take note of is the relative error between two solutions at a certain node.

Figure 4- Average Error of Relative errors between two solutions

*Figure 4* shows the average value of the relative difference between the final solution and the second to final solution. As shown, at 40 nodes, the difference between the final solutions is minimal, signifying that the approximate solution is most likely converging to the actual solution. This also shows the spatial accuracy, as doubling the nodes reduces the error by a huge factor.

**Implicit Discretization Solutions**

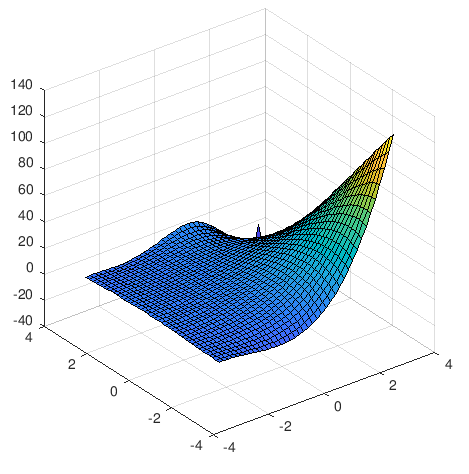


Figure 3-implicit with 40 Nodes

*Figure 3* shows the implicit method with 40 nodes. Despite the angle of the grid being used, the graph looks very similar to what was gained from the explicit method.

**Conclusion**

In conclusion, numerical solutions are powerful. With a good equation, the right tools, and the right initial conditions, nearly any equation can be modeled. When choosing which method to use, weigh the advantages with the disadvantages, make sure the code is implemented correctly and well thought of, and model any PDE with the right givens.