# Abstract:

**MECE 5397 Scientific Computing**

**Spring 2019**

**Final Project**

**Diffusion Equation – B01-1**



Jonah R. Smith, PSID: 1569818

Professors: A. Amritkar, and A. Prosperetti

Date of Submissions: 05/04/2019

# Problem Statement

The purpose of this project was to solve a simple 2D diffusion equation with no source terms using two discretizations/solution methods, and be able to determine grid convergence and convergence to steady state using these two discretizations. The mathematical statement of the problem is as follows:

Subject to the following constraints and boundary conditions:

That is, there are three Dirichlet conditions on the boundaries of the surface, and one Neumann boundary condition at . For this problem, the following values were given:

For reference, the following figure shows the domain of the problem, and approximately what the Dirichlet boundary conditions look like on an x-y plot, and the Neumann condition is left empty. This plot was made using 64 nodes in both the x and y directions.



Figure 1: Map of Dirichlet Boundary Conditions

# ADI and Explicit Discretizations of Problem

For this project, two different solution methods were used to solve the equation: ADI, and the explicit method.

ADI –

The Alternating Direction Implicit (ADI) scheme is an implicit solution method which is also able to take advantage of the Thomas Algorithm when solving multi-dimensional diffusion equations. The ADI scheme works by breaking down each timestep into two “half-timesteps”. The solution routine takes these half-timesteps alternating between the x and y directions, assuming constant values in the direction about which the timestep is not currently being taken. Given Equation 1, this results in the following discretization:

Equations 4 and 5 give the discretization used to approximate derivatives along the x and y directions respectively. For any given , , this equation uses a stencil as shown below in Figure 2. By using the ADI method, for , , and are solved simultaneously by assuming known value for and from the previous half-timestep’s solution. Then, , , and are solved simultaneously by using the values for and that were just found.

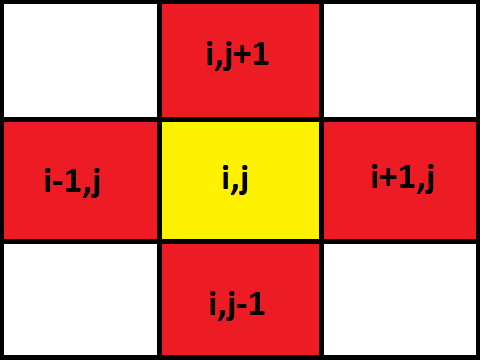


Figure 2: Stencil Used in Discretization

# Description of Numerical Methods

The code created for both the ADI and Explicit methods are extremely similar. The only difference between the two is what happens within the loop that solves for each . The code is setup such that the script Main.m is used to call either ADI.m or Explicit.m, functions for the two solution routines, providing the following inputs:

1. (**ax**) ~ This variable is from the setup of the problem, and should be assigned a value of 0.
2. (**bx**) ~ This variable is from the setup of the problem, and should be assigned a value of 2π.
3. (**ay**) ~ This variable is from the setup of the problem, and should be assigned a value of 0.
4. (**by**) ~ This variable is from the setup of the problem, and should be assigned a value of 2π.
5. (**nodefacx**) ~ This variable defines the level of nodalization in the x direction. The number of internal nodes in the x direction is given by (ceil(2^(nodefacx))-2.
6. (**nodefacy**) ~ This variable defines the level of nodalization in the y direction. The number of internal nodes in the y direction is given by (ceil(2^(nodefacy))-2.
7. (**DTIMEI**) ~ DTIMEI is a control constant that will be referenced many times throughout this report, and is simply the name given to , the size of each timestep. For the ADI routine, each half-timestep is DTIMEI/2.

For both functions, the script begins by either loading in a savefile from a previous run of the function, or by setting up an array for x, y, and u, and creating a simple guess for u by using a linear approximation between the and boundary conditions. The following snippet of code in Figure 3 shows how this initial value for u is derived, and how the Dirichlet boundary conditions are setup.

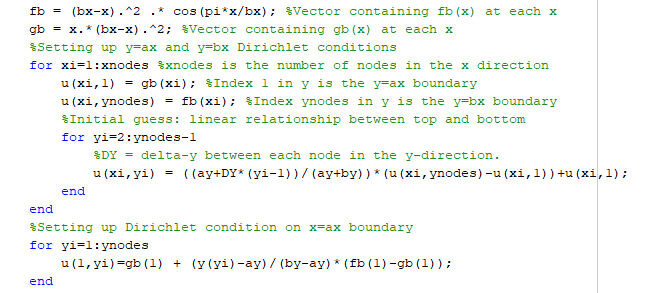


Figure 3: Setting up Dirichlet Boundary Conditions, and Creating an Initial Value for u

The vector **x** is an **xnodes** by 1 array, where , such that and =. The variables **xnodes** and **ynodes** are constant values that give the number of nodes (including 1 node for each boundary) in either the x or y direction. Vectors **gb** and **fb** correspond to and found in the description of the problem, where , and likewise for . This means that the internal nodes within u are given by

u(2:xnodes-1,2:ynodes-1), and that the Neumann boundary is given by u(xnodes,:).

For the ADI method, a half-timestep is taken, and a loop around

# Machine Specifications

# Results