Allen Nguyen

1191857

MECE 5397

Problem code: B02-4

Scientific Computing

for Mechanical Engineering

Dr. Andrea Prosperetti

Professor Amit R Amrikar

Numerical solutions of Two-Dimensional Heat Diffusion Equation

* **Abstract**

This project will produce Matlab codes using two different discretization methods: Alternating Direction Implicit (ADI) method and Explicit method to solve numerically the two-dimensional diffusion equation. The numerical solutions of the two-dimensional equation give approximations of the temperature distribution in a squared object of the size 2π by 2π at steady state. Furthermore, by employing two different discretization methods, the validity of the two methods in solving the second order partial differential equation can be verified by one another.

* **Problem Statement: 2D Heat Diffusion Equation**

Domain of interest:

Boundary conditions:

Initial condition:

* **Discretization methods**

Two methods are chosen to discretize the 2D heat diffusion equation: To make the

+ Explicit discretization method:

\* and discretized using the central difference approximation.

\*Different than the ADI method, the explicit method discretized the heat diffusion equation at one whole time step.

As a result, the discretized heat diffusion equation using the explicit method has the form:

\*

And after few manipulations and set

Although the explicit method is less stable than the ADI method since it is conditionally stable while ADI is unconditionally stable, the setup and iteration process of the explicit are very straight forward does not use lots of memory.

+ Alternating direction implicit (ADI) method:

\* and discretized using the central difference approximation

\* is discretized using the forward approximation at half of a time step

From the discretization above, the discretized heat diffusion equation using ADI method has the forms:

First half of a time step

\*

After some manipulations:

(1)

Second half of a time step

\*

After the same manipulations as the first half of a time step:

(2)

The equations (1) and (2) produce 2 huge diagonal matrices, each for each half time step, with which the Thomas algorithm can be used to solve for the distribution of u along x and y axes. Instead of iterating over huge diagonal matrices, notice each of the huge diagonal matrix can be partitioned into multiple smaller diagonal matrices. Iteration over each of these small diagonal matrices produce values of u along a row for the first half of a time step. With the same

[ (1+) 0 0

(1+) 0

0 (1+) c =

b a

method, each of the smaller diagonal matrices give values of u along a column for the second half of the time step. With this setup, lots of memory can be saved and extra number of iterations over a, b, c using for loop can be omitted because a, b and c are always constant. The only the thing change is the right side of the matrix equation. They can be divided into block as the result of the partitions of the huge diagonal matrix and arranged in to a matrix whose each column contains the right side for each of the small diagonal matrices. The same is done for the iterations of the second half of the time step. The only different is the first c value for each of the small diagonal matrices

[ (1+) 0 0

(1+) 0

0 (1+) c =

b a

Below is pseudo code which gives a clear insight into the what was described above

Value of a

Value of b

Value of c

while the average difference in u values between the old and new time step is not close to 0

Increase current time

For the first column to the last column of the matrix that contains the right-side values for each of the small diagonal matrix.

Iterate using u values at the current time step to find the values of right side of the matrix equation

Iterate using these right-side values to find values of u at one half of a time step newer.

U at current time step = U at old time step

For the first column to the last column of the matrix that contains the right-side values for each of the small diagonal matrix.

Iterate using u values at a half newer time step to get the right-side values of the matrix equation

Iterate using these right-side values to get values of u at newer time step.

U at current time step= U at newer time step

Difference=u at current time step-u at the old-time step.

* **Result Discussion**