Abstract

The purpose of this code is to solve a two dimensional diffusion equation by discretizing the partial differential equation provided. The discretization is then run through an algorithm designed to generate a solution mathematically via the use of iterations. These iterations are small in nature, but are calculated over large intervals to display steady state behavior. Demonstrated in the source code are two method of using these iterations to solve the equation which has both Dirichlet and Neumann boundary conditions. The bounds of this equation fall on the rectangle of negative pi to pi in both the x and y coordinates. The code provided will generate solutions to this problem using two different methods, one being explicit, and the other being implicit. The explicit code, while being limited by small steps in time, runs quickly and generates good plot to display the results in real time. The implicit code relies on projecting what the next iteration will look like based on previous iterations. In a real world scenario, this code can be used to model the flow of heat through a plate over time.

**Mathematical statement of problem**

The diffusion equation in two dimensions takes a basic form along the lines of:

d2u/dx2+ d2u/dy2 = du/dt

The problems asks us to solve this equation using a computer code. Since the programs available to us to not understand derivatives, we need to discretize this problem.

**Discretized equations**

To solve this equation we will make use of a scheme that discretizes using the central difference method in space, and the forward difference method in time. Individually, these equations look like the following:

d2u/dx2 = (T n i+1,j − 2T n i,j + T n i-1,j)/ /Δx2

d2u/dy2 = (T n i,j+1 − 2T n i,j + T n i,j−1)/Δy2

du/dt = (T n+1 i,j − T n i,j)/Δt

When combined our full diffusion equation looks like the following

(T n+1 i,j − T n i,j)/Δt = (T n i+1,j − 2T n i,j + T n i-1,j)/Δx2 +(T n i,j+1 − 2T n i,j + T n i,j−1)/Δy2

This is the model we will follow to execute our MATLAB code.

**Description of the numerical method**

In the code two methods are used to solve the proposed equation. The first method is ***explicit***. To utilize this method we need to establish a series of matrices and vectors first. The first matrix initialized is the initial conditions matrix. Our problem description has the following specified Dirichlet conditions:

u(x = ax, y) = (y – ay)2sin(pi(y - ay)/2(by – ay)

u(x = bx, y) = cos[pi(y – ay)]cosh(by – y)

where ax = bx = -pi, ay = by = pi

and u(x, y, t = 0) = 0

So we can see that our rectangle has a constant value distribution along the sides of x = ax, bx, and an initial value of zero everywhere else. So we construct this matrix to start with.

Working with 51 nodes, the initial u matrix is generated in the following way

U=zeros(51,51)

u(:,1) = (y – ay)2sin(pi(y - ay)/2(by – ay)

u(:, 51) = cos[pi(y – ay)]cosh(by – y)

Now we build the 5-diagonal coefficient matrix A like so

A = zeros(2601)

Along the main diagonal:

A(i,i)=-2\*(1/dx^2 + 1/dy^2)

As for the other four diagonals, we build our blocks like so:

for i=1:50

for j=1:51

A(i+(j-1)\*51, i+(j-1)\*51+1) = 1/dx^2;

A(i+(j-1)\*51+1, i+(j-1)\*51) = 1/dx^2;

end

end

for i=1:51

for j=1:50

A(i+(j-1)\*51, i+j\*51) = 1/dy^2;

A(i+j\*51, i+(j-1)\*51) = 1/dy^2;

end

end

Now we can begin our iterative process. We begin by specifying a time step, in this case .0025 seconds.

We vectorize our initial values vector u:

U = reshape(u, 2601,1)

and then multiply A\*u to calculate the solution vector M.

M = A\*U

We then reshape M to the same dimensions as the value matrix u, and finally add u +Δt\*M to get our new values. We repeat these steps until we see our values reach a steady state.

u = u +Δt\*M

The second method is ***implicit***.

The initial value matrix is constructed the same way as in the explicit method.

Now our coefficient matrix A is constructed a bit differently. Most of the values rely on a parameter lambda, calculated in the following way:

lambda=dt/dx^2

And A is a five-diagonal matrix constructed like so:

for i = 1:2601

A(i,i)=1+4\*lambda;

end

for i=1:50

for j=1:51

A(i+(j-1)\*51, i+(j-1)\*51+1) = -lambda;

A(i+(j-1)\*51+1, i+(j-1)\*51) = -lambda;

end

end

for i=1:51

for j=1:50

A(i+(j-1)\*51, i+j\*51) = -lambda;

A(i+j\*51, i+(j-1)\*51) = -lambda;

end

end

again, we run through our iterative process with Δt = .0025. but this time, our u matrix is a guess at the values of u at time Δt, and we are solving for the original values to put into vector P.

U=reshape(u,2601,1);

P=A\U;

M=reshape(P,51,51);

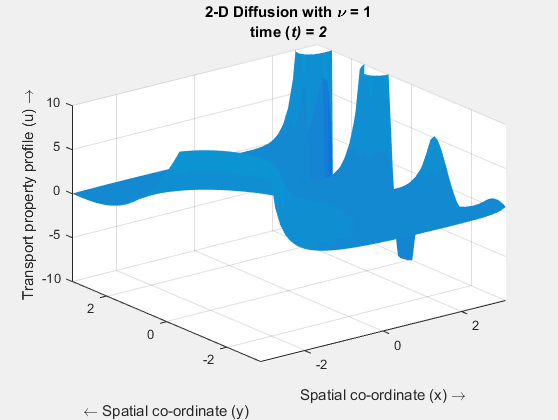
u = u + dt\*M

This way we can guess the values at the next step of Δt based on their current values.

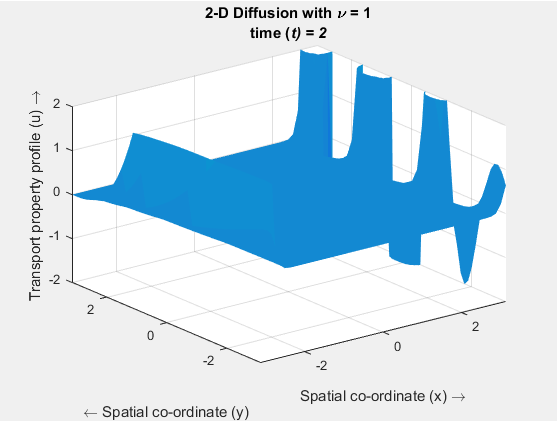
**Results**

results after two seconds (steady state has been reached)

Explicit method:



Implicit method



**Optimization**

The following optimizations were made to decrease run time of the source code:

Vectorization of the dirichlet condition for loops

Initial:

phi=zeros(51,1);

for i=1:51

phi(i)= ((y(i)-pi)^2)\*sin((y(i)-pi)/2);

end

psi=zeros(51,1);

for i=1:51

psi(i)=cos(pi\*(y(i) - pi))\*cosh(pi - y(i));

end

Optimized:

phi = ((y + pi).^2).\*sin((pi\*(y+pi))/(2\*(2\*pi)));

psi = (cos(pi\*(y + pi))).\*cosh(pi - y);