Caleb Logemann
AER E 546 Fluid Mechanics and Heat Transfer I
Homework 4

1. The laser heating in the second problem of homework 3 should probably be modeled as a localized hot-spot in two dimensions - assuming that the slab is thin, so it is heated uniformly across its thickness. Solve the (non-dimensional) diffusion equation

$$\frac{\partial T}{\partial t} = \nabla^2 T \qquad \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

in the square region $0 \leq x, y \leq 1$. Form the semi discrete equations and integrate in time by RK2. Choose a suitable Δt. Submit only the algorithm part of your code.

The walls are kept cool, i.e. $T = 0$ on the boundaries. The initial hot spot is represented by

$$T(x, y) = \frac{1}{\pi\sigma^2} e^{-r^2/\sigma^2} \qquad r^2 = (x - 0.25)^2 + (y - 0.25)^2, \sigma = 0.1$$

- Let $Nx = Ny = 201$. Provide a plot of contour lines of the solution at $t = 0.02$ and $t = 0.1$.
- How long (t =??) must you wait for the maximum temperature to drop to 1% of its intial value?

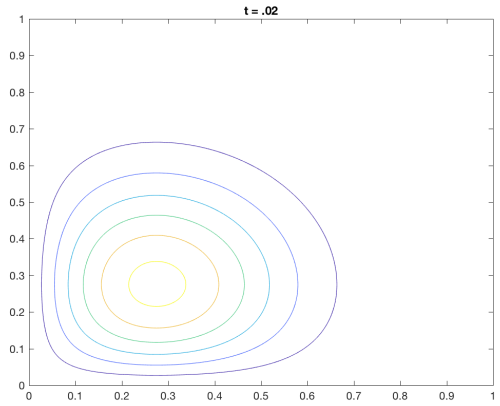
The following function runs the second order Runge-Kutta method for any given right hand side operator. This is the same method that was used in an earlier homework.

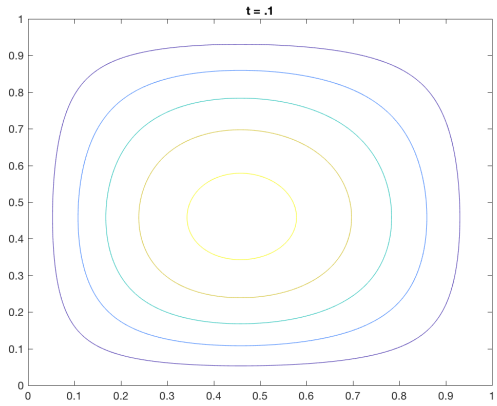
```
function [result] = RK2(RHSFunc, x0, nTimeSteps, deltaT)
    n = length(x0);
    result = zeros(nTimeSteps+1, n);
    result(1,:) = x0;

    for i = 1:nTimeSteps
        t = (i-1)*deltaT;
        temp = result(i,:) + 0.5*deltaT*RHSFunc(t, result(i,:));
        result(i+1, :) = result(i,:) + deltaT*RHSFunc(t + 1/2*deltaT, temp);
        disp(i/nTimeSteps);
    end
end
```

The following function implements the right hand side for the 2D heat operator.

```
function [result] = HeatRHS2D(t, T0, Nx, Ny, deltaX, deltaY, reshapeFunc, nFunc)
    result = zeros(1,Nx*Ny);
    T = reshapeFunc(T0);
    for i = 2:Nx-1
        for j = 2:Ny-1
            result(nFunc(i, j)) = (T(i+1,j) - 2*T(i, j) + T(i-1,j))/deltaX^2 + (T(i, j
                ↪ +1) - 2*T(i, j) + T(i,j-1))/deltaY^2;
        end
    end
end
```

Using both of these function the following images can be produced. These show the temperature profile at $t = .02$ and .1. Note that the overall temperature is much less at $t = .1$ because most of the energy has diffused out through the boundaries.





Taking the maximum after each time step we see that at $t = 0.092$ the maximum value just drops past 1% of the initial maximum value.

2. Explain why ADI is called approximate factorization - i.e. make sure you understand the formal basis for the method.

   Use the ADI scheme with Euler implicit time-stepping to solve the same heat equation, as in the previous problem, in the same square, $0 \leq x, y \leq 1$. But now, initially $T = 0$ everywhere, except on the lower wall. The boundary conditions are

$$T = \sin([)2]2\pi x \text{ on } y = 0; 0 \leq x \leq 1$$
$$T = 0 \text{ on } x = 0 \text{ and } x = 1; 0 \leq y \leq 1$$
$$T = 0 \text{ on } y = 1; 0 \leq x \leq 1$$

   Integrate to $t = 0.25$ and plot temperature contours at $t = 0.002, 0.01, 0.04$ and $t = 0.1$. Provide a line plot of $T(1/2, 1/2)$ versus t; has the solution converged to steady state? If so, what equation has been solved? Provide the algorithm part of your code - with brief comment lines, showing how you implemented ADI.

   ADI is called an approximate factorization as it is making the following approximation

$$\left(1 - \frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2}\right) \approx \left(1 - \frac{\partial^2}{\partial x^2}\right)\left(1 - \frac{\partial^2}{\partial y^2}\right)$$

   This operator does not exactly factor like this, but it is accurate to Δx^4 so the factorization is approximately correct.

   The following function now implements the ADI method.

```
function [u] = ADI(u0, deltaX, deltaY, Nx, Ny, deltaT, nTimeSteps, g)
    % to change from natural row wise ordering to natural
    % column wise ordering or vice versa
    % uCol = uRow(rowToCol) or uRow = uCol(colToRow)
    rowToCol = repmat(0:Ny:Nx*Ny-Ny, 1, Ny) + kron(1:Ny,ones(1, Nx));
    colToRow = repmat(0:Nx:Nx*Ny-Nx, 1, Nx) + kron(1:Nx,ones(1, Ny));

    % matrix to represent diffusion in row/x direction
    % Dx*uRow = Dx2*uRow
    e = ones(Nx, 1);
    tridiagonal = 1/(deltaX^2)*spdiags([e, -2*e, e], [-1, 0, 1], Nx, Nx);
    Dx = kron(speye(Ny), tridiagonal);
```

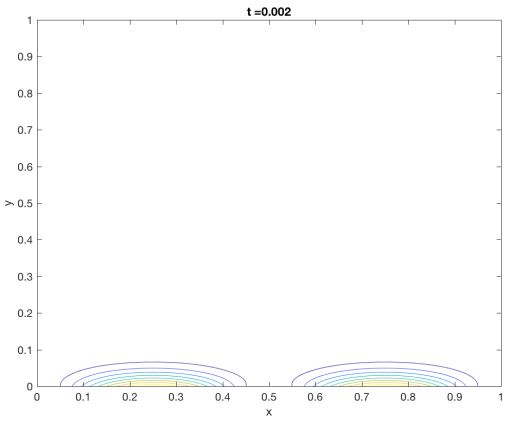
```
% matrix to represent diffusion in col/y direction
% Dy*uCol = Dx2*uCol
e = ones(Ny, 1);
tridiagonal = 1/(deltaY^2)*spdiags([e, -2*e, e], [-1, 0, 1], Ny, Ny);
Dy = kron(speye(Nx), tridiagonal);

I = speye(Nx*Ny);

% solution at all times
% stored in rowwise numbering
u = zeros(nTimeSteps, Nx*Ny);
u(1,:) = u0;

% create index arrays for boundary
% indices for left boundary in row-wise ordering or
zeroIndicesRow = 1:Nx:Nx*Ny;
% indices for right boundary in row-wise ordering or
oneIndicesRow = Nx:Nx:Nx*Ny;
% indices for bottom boundary in column-wise ordering
zeroIndicesCol = 1:Ny:Nx*Ny;
% indices for top boundary in column-wise ordering
oneIndicesCol = Ny:Ny:Nx*Ny;
for n = 1:nTimeSteps
    % First stage
    % (I + deltaT/2 Dy2)*u
    % rhs in column wise ordering
    rhs = (I + deltaT/2*Dy)*u(n, :)';
    rhs = rhs(rowToCol);
    % add boundary conditions for y-direction at time t = tn
    bottomBoundary = deltaT/(2*deltaY^2)*g(deltaT*(n-1), deltaX*(0:Nx-1), 0);
    topBoundary = deltaT/(2*deltaY^2)*g(deltaT*(n-1), deltaX*(0:Nx-1), 1);
    rhs(zeroIndicesCol) = rhs(zeroIndicesCol) + bottomBoundary';
    rhs(oneIndicesCol) = rhs(oneIndicesCol) + topBoundary';

    % change to row-wise ordering
    rhs = rhs(colToRow);
    % add boundary conditions for x-direction at time t = tn + deltaT/2
    leftBoundary = deltaT/(2*deltaX^2)*g(deltaT*(n-1)+deltaT/2, 0, deltaY*(0:Ny-1))
        ↪ ;
    rightBoundary = deltaT/(2*deltaX^2)*g(deltaT*(n-1)+deltaT/2, 1, deltaY*(0:Ny-1)
        ↪ );
    rhs(zeroIndicesRow) = rhs(zeroIndicesRow) + leftBoundary';
    rhs(oneIndicesRow) = rhs(oneIndicesRow) + rightBoundary';
    % solve for uStar which approximates u at t = tn + k/2
    % uStar is in row-wise ordering
    uStar = (I - deltaT/2*Dx)\rhs;

    % second stage
    % rhs row-wise ordering
    rhs = (I + deltaT/2*Dx)*uStar;
    % add boundary conditions for x-direction at time t = tn + k/2
    % left and right boundaries same as before
    rhs(zeroIndicesRow) = rhs(zeroIndicesRow) + leftBoundary';
    rhs(oneIndicesRow) = rhs(oneIndicesRow) + rightBoundary';
    % change to column-wise ordering
    rhs = rhs(rowToCol);
    % add boundary conditions for y-direction at time t = tn + k
    bottomBoundary = deltaT/(2*deltaY^2)*g(deltaT*n, deltaX*(0:Nx-1), 0);
    topBoundary = deltaT/(2*deltaY^2)*g(deltaT*n, deltaX*(0:Nx-1), 1);
    rhs(zeroIndicesCol) = rhs(zeroIndicesCol) + bottomBoundary';
```

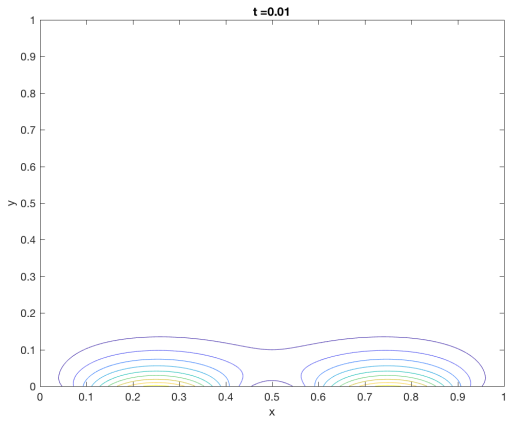
```
        rhs(oneIndicesCol) = rhs(oneIndicesCol) + topBoundary';
        % solve for u at time t = tn + k
        % u is in row wise ordering
        temp = ((I - deltaT/2*Dy)\rhs);
        u(n+1,:) = temp(colToRow)';
    end
end
```

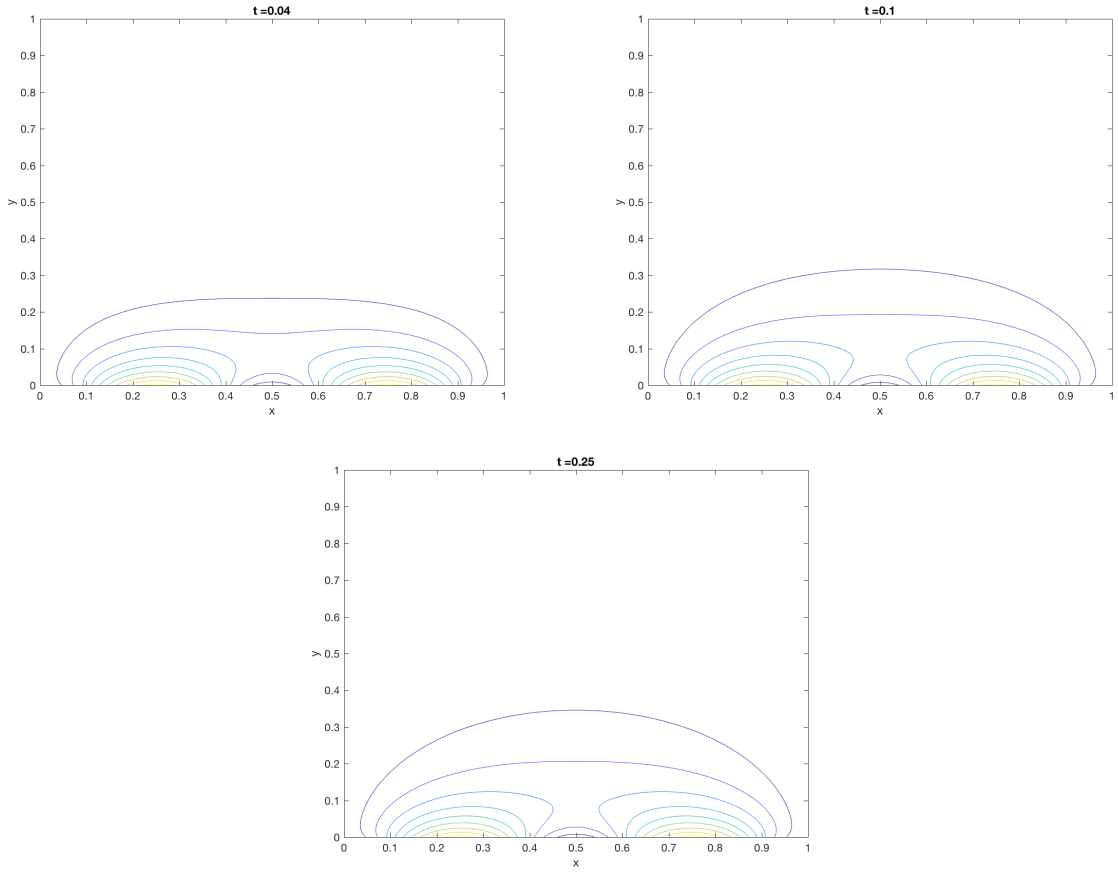
The following script now uses the previous function to solve the heat equation with the given boundary conditions and initial values.

```
% numbered first left to right, then bottom to top
nFunc = @(i, j) (j - 1)*Nx + i;
iFunc = @(n) mod(n, Nx) + Nx*(mod(n, Nx) == 0);
jFunc = @(n) (n - iFunc(n))/Nx + 1;
reshapeFunc = @(u) reshape(u, [Nx, Ny])';

x = linspace(a, b, Nx);
y = linspace(a, b, Ny);
u0func = @(x, y) (y == 0)*sin(2*pi*x)^2;

u0 = zeros(nGridCells, 1);
for i = 1:Nx
    for j = 1:Ny
        u0(nFunc(i, j)) = u0func(x(i), y(j));
    end
end

g = @(t, x, y) (y == 0)*sin(2*pi*x).^2;

adiSol = ADI(u0, deltaX, deltaY, Nx, Ny, deltaT, nTimeSteps, g);

iter = 3;
for t = [0.002, 0.01, 0.04, 0.1, 0.25]
    rk2solMatrix = reshapeFunc(adiSol(t/deltaT+1,:));
    contour(x, y, rk2solMatrix);
```

The following images are produced.







The following is a graph of the temperature at $(0.5, 0.5)$ over the course of the simulation. As you can see the temperature is approaching a steady state. This means that the final solution is actually a solution to the steady state elliptic problem

$$\nabla^2 T = 0.$$