ADI Method for 2 Dimensional Heat Equation

Final Project for Numerical Methods II — Option 4

Sixian Liu

Instructor: Leslie Greengard

Contents

1 Matrix System for the 2d ADI **1**

2 Boundary Conditions **2**

3 Numerical Solution vs. True Solution **3**

4 Errors and Order of Accuracy **4**

5 Stability of ADI **4**

6 Matlab Code for ADI Function **6**

7 Matlab Code for Order of Accuracy **8**

1 Matrix System for the 2d ADI

The ADI method contains the following two steps.

$$U_{ij}^{*} = U_{ij}^{n} + \frac{k}{2}(D_y^2 U_{ij}^{n} + D_x^2 U_{ij}^{*})$$

$$U_{ij}^{n+1} = U_{ij}^{*} + \frac{k}{2}(D_x^2 U_{ij}^{*} + D_y^2 U_{ij}^{n+1})$$

Using the 3 points approximation for the second derivative, we have the corresponding equations.

$$U_{i,j}^{*} - r_x(U_{i-1,j}^{*} - 2U_{i,j}^{*} + U_{i+1,j}^{*}) = U_{i,j}^{n} + r_y(U_{i,j-1}^{n} - 2U_{i,j}^{n} + U_{i,j+1}^{n})$$

$$U_{i,j}^{n+1} - r_y(U_{i,j-1}^{n+1} - 2U_{i,j}^{n+1} + U_{i,j+1}^{n+1}) = U_{i,j}^{*} + r_x(U_{i-1,j}^{*} - 2U_{i,j}^{*} + U_{i+1,j}^{*})$$

where $r_x = k/2h_x^2$ and $r_y = k/2h_y^2$.

Then, we use the vector to store the values for 2d.

$$u = \begin{bmatrix} u^{[1]} \\ \cdot \\ \cdot \\ \cdot \\ u^{[m]} \end{bmatrix}, \text{ with } u^{[j]} \begin{bmatrix} u_{1j} \\ \cdot \\ \cdot \\ \cdot \\ u_{mj} \end{bmatrix}$$

In terms of the matrix system, we have

$$(I - r_x D_x^2)U^* = (I + r_y D_y^2)U^n + r_x gstar(t) + r_y gn(t)$$
$$(I - r_y D_y^2)U^{n+1} = (I + r_x D_x^2)U^* + r_y gnp(t) + r_x gstar(t)$$

where I is the $m_x m_y * m_x m_y$ identity matrix, and D_x^2, D_y^2 in the following form.

$$D_x^2 = \begin{bmatrix} T & & & \\ & T & & \\ & & \ddots & \\ & & & T \end{bmatrix}, T = \begin{bmatrix} -2 & 1 & & \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix},$$

$$D_y^2 = \begin{bmatrix} -2L & L & & \\ L & -2L & \ddots & \\ & \ddots & \ddots & L \\ & & L & -2L \end{bmatrix}, L = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

T is of size $m_x * m_x$. L is the $m_x * m_x$ identity matrix.

2 Boundary Conditions

The boundary conditions are

$$gstar(t) = g(t + k/2), \text{ at } x = 0, x = 1$$

$$gn(t) = g(t), \text{ at } y = 0, y = 1$$

$$gnp(t) = g(t + k), \text{ at } y = 0, y = 1$$

where $g(t)$ is the heat equation evaluated at time t. However, in this problem, the heat equation contain $\sin(\pi x)$ and $\sin(\pi y)$. the boundary conditions are simply zeros at integer values.

Notes: In the Matlab code, I commented out the initial conditions gained by assigning the true solution to the boundary values (line 50-64). Instead, I set the boundary values to zeros directly. This is because I observed that "assigning" the values will lose some accuracy, i.e. the zeros in the true solution become nonzeros (about $1.0e-17$) in $g(t)$.

3 Numerical Solution vs. True Solution

The following plots are the numerical solution and true solution of the heat equation with $mx = my = 159$, tfinal $= 0.1$.

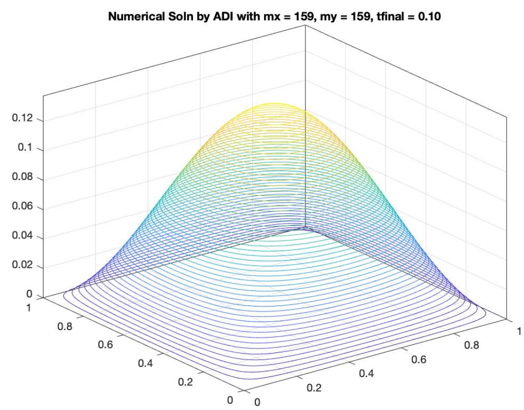

Figure 1: Numerical Solution to the heat equation $\exp(-2\pi^2 t)\sin(\pi x)\sin(\pi y)$ by ADI Method with $m_x = m_y = 159$ and tfinal $= 0.1$

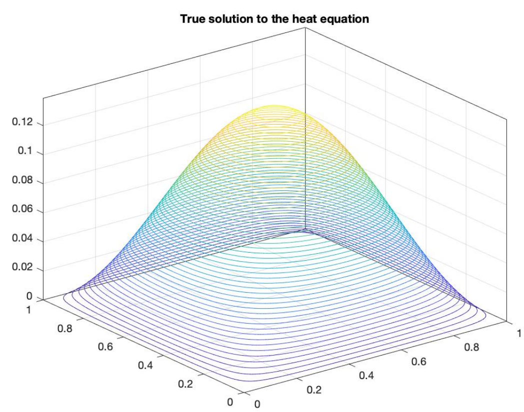

Figure 2: True Solution to the heat equation$\exp(-2\pi^2 t)\sin(\pi x)\sin(\pi y)$

4 Errors and Order of Accuracy

From the **loglog** plot, we see that the slope is approximately 2. The lease square fit of the error vs. h is of second order. We've verified that ADI is second order.

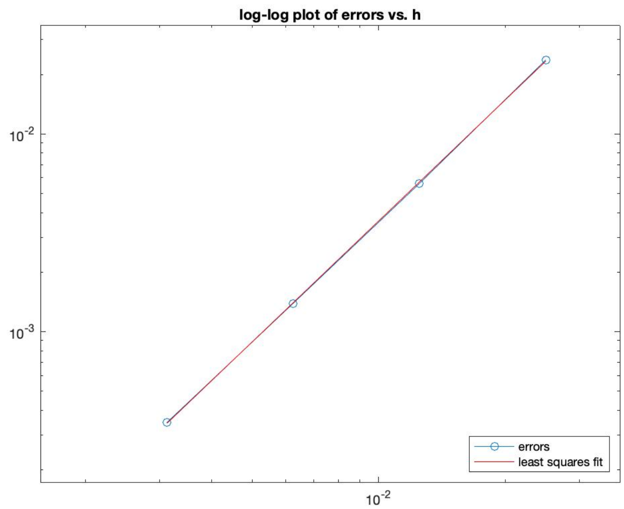

Figure 3: loglog plot of max. error vs. h

```
     h        error       ratio      observed order
  0.02500   2.37314e-02      NaN            NaN
  0.01250   5.62255e-03   4.22075        2.07750
  0.00625   1.38804e-03   4.05072        2.01818
  0.00313   3.45934e-04   4.01243        2.00448


Least squares fit gives E(h) = 42.1026 * h^2.03186
```

Figure 4: Table of the Error and Least Square Fit of Order of Accuracy

5 Stability of ADI

According to section 10.7 in the textbook (we also showed in HW6, the trapezoidal method), an implicit method satisfies CFL for any time step k. However, CFL is a necessary condition for the numerical solution to be convergent.

Experimentally, we set $k = c * h$ and reduce c. We obtain a plot of k vs. the max. error. We observe that as k decreases, the max. error decreases.

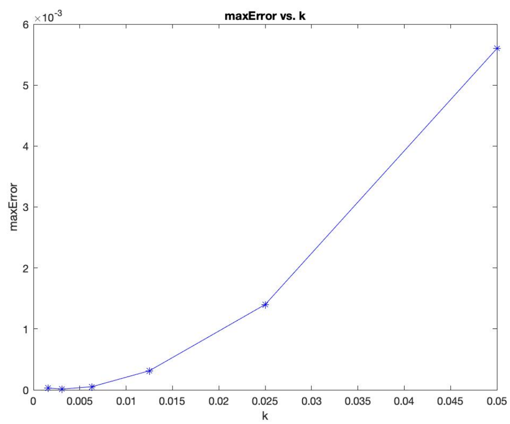

Figure 5: Max. Error vs. the Time Step k

A similar Von Neumann Analysis to Multi-step 2d Method

This analysis is described in Trefethen's *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations* chapter 3 and 4 [1]. The general idea is the following.

- Take the Fourier transform of the 2-vector.
- Instead of the the amplification factor $g(\xi)$ for 1d method, we have the amplification matrix $G_k(\xi)$ obtained from the Fourier transformed system.
- Then, examine the $||G_k(\xi)^n|| \leq C$. By, the lower and upper bound

$$\rho G_k(\xi)^n \leq ||G_k(\xi)^n|| \leq ||G_k(\xi)||^n,$$

Trefethen indicates this theorem. Here, ρ is the spectral radius.

VON NEUMANN CONDITION
FOR VECTOR FINITE DIFFERENCE FORMULAS

Theorem 4.10. *Let $\{S_k\}$ be a linear, constant-coefficient finite difference formula as described above. Then*

(a) $\rho(G_k(\xi)) \leq 1 + O(k)$ *is necessary for stability, and* (4.6.3)

(b) $\|G_k(\xi)\| \leq 1 + O(k)$ *is sufficient for stability.* (4.6.4)

This is somehow beyond the scope of the course. So, I did not perform an analysis on ADI myself.

[1] https://people.maths.ox.ac.uk/trefethen/4all.pdf

6 Matlab Code for ADI Function

```matlab
function [h,k,error] = ADI(mx, my)

    ax = 0;
    bx = 1;
    ay = 0;
    by = 1;
    tfinal = 0.1;
    hx = (bx-ax)/(mx+1);
    hy = (by-ay)/(my+1);
    x = linspace(ax,bx,mx+2);
    y = linspace(ay,by,my+2);
    [X,Y] = meshgrid(x,y);
    X = X';
    Y = Y';

    k = 4 * (hx+hy)/2;    % time step
    nsteps = ceil(tfinal / k);     % number of time steps

    f = @(x,y,t) exp(-2*t*pi.^2).*sin(pi*x).*sin(pi*y);

    % initial condition at t = 0
    u0 = f(X,Y,0);

    % set up the matrices.
    rx = (1/2) * k /(hx^2);
    ry = (1/2) * k /(hy^2);
    e = ones(mx,1);
    I = speye(my);
    S = spdiags([e e],[-1 1],my,my);
    T = spdiags([e -2*e e], [-1 0 1], mx, mx);
    Dx = rx*kron(I, T);  % This is actually (k/2) * Dx^2
    Dy = ry*((kron(I, -2*I) + kron(S, I)));  % This is actually (k/2) * Dy^2
    II = speye(my*mx);

    % initialize u and time
    tn = 0;
    u = u0;

    % main time-stepping loop:

    for n = 1:nsteps
        gstar0 = zeros(mx,my);
```

```
        gn0 = zeros(mx,my);
        gnp0 = zeros(mx,my);

        tnp = tn + k;   % t_{n+1}

        fstar = f(X,Y,(tn + 0.5*k));

        % boundary conditions
%          gstar0(1,:) = fstar(1,2:(m+1));   % x = 0
%          gstar0(m,:) = fstar(m+2,2:(m+1)); % x = 1
%
%         %gstar0
%
%          gn0(:,1) = u(2:(m+1),1);       % y = 0
%          gn0(:,m) = u(2:(m+1),m+2);     % y = 1
%         %gn0
%
         unp = f(X,Y,tnp);
%          gnp0(:,1) = unp(2:(m+1),1);    % y = 0
%          gnp0(:,m) = unp(2:(m+1),m+2); % y = 1
%         %gnp0
%
        uint = u(2:(mx+1),2:(my+1));  % interior points

        % reshape the interior pts and bcs to m*m vector
        uint = reshape(uint,mx*my,1);
        gstar0 = reshape(gstar0,mx*my,1);
        gn0 = reshape(gn0,mx*my,1);
        gnp0 = reshape(gnp0,mx*my,1);


        % solve for the first equation
        rhs1 = (II + Dy)*uint + rx*gstar0 + ry*gn0;
        ustar = (II - Dx)\rhs1;

        % solve for the second equation, uint is U^{n+1} now
        rhs2 = (II + Dx)*ustar + rx*gstar0 + ry*gnp0;
        uint = (II - Dy)\rhs2;

        % add the boundary values, m*m vector unp
        uint = reshape(uint,mx,my);
        u = unp;
        u(2:(mx+1),2:(my+1)) = uint;
```

```
        tn = tnp;

    end       % end of the for loop

    h = (hx+hy)/2;

    ufinal = f(X,Y,tnp);
    error = max(max(abs(u-ufinal)));
    contour3(X,Y,u,50)
    title(sprintf('Numerical Soln by ADI with mx = %3d, my = %3d, tfinal = %2.2f',mx,my,tfinal))

    input('Hit <return> to continue  ');

    contour3(X,Y,ufinal,50)
    title('True solution to the heat equation')
```

7 Matlab Code for Order of Accuracy

```

clc; clear; close all
mxvals = [39 79 159 319];
myvals = [39 79 159 319];
ntest = length(mxvals);
hvals = zeros(ntest,1);  % to hold h values
E = zeros(ntest,1);   % to hold errors

for jtest=1:ntest
    mx = mxvals(jtest);
    my = myvals(jtest);
    [h,k,error] = ADI(mx,my);
    E(jtest) = error;
    hvals(jtest) = h;

end

error_table(hvals, E);   % print tables of errors and ratios
error_loglog(hvals, E);  % produce log-log plot of errors and least squares fit
```