Solving 2D Heat Conduction Equation using FD Schemes

Garvit Mathur (Mechanical Engineer 3rd Year)
b16096@students.iitmandi.ac.in

AIM

The aim is to solve the 2D heat conduction equation using iterative schemes such as Alternating-Direction Implicit (ADI) Method, Alternating-Direction Explicit (ADE) Method, and Hopscotch Method and compare the result to get the most efficient scheme for this case.

Introduction

Consider a Rectangular plate of dimension a x b made up of some conductive material, the plate is kept in an atmosphere where all of its sides are exposed to some different temperature. To study the effect of these temperatures on the rectangular plate after a long time i.e. after a steady state reach is done using the 2D Heat conduction equation:

$$\frac{\partial u}{\partial t} = k\nabla^2 u \tag{1}$$

Where u is Temperature variable and k is material conductivity.

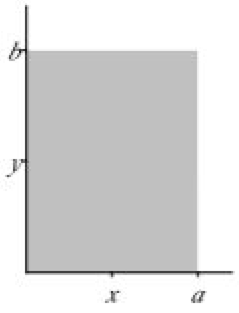

Fig1. Rectangular Plate of dimension a x b

Method

The domain considered in this experiment has a=w=1. And the Dirichlet boundary conditions are T1=100k, T2=200k, T3=300K and T4=400K. As shown in fig 2.

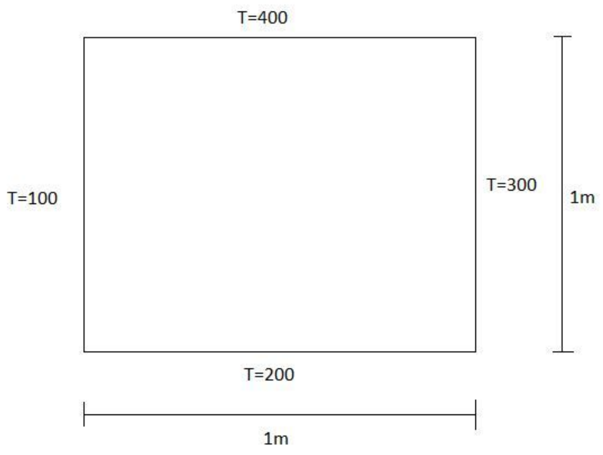

Fig 2. Domain & Boundary Conditions

The eq'n 1 is discretized using Finite difference schemes to solve the problem and to calculate the result computationally.

Finite Difference Schemes Used to solve the problem are:

Alternating-Direction Implicit (ADI) Method,

Alternating-Direction Explicit (ADE) Method and

Hopscotch Method

Each scheme is discussed in details below.

Alternating-Direction Implicit (ADI) Method

The ADI method only results in tridiagonal equations if it is applied along the dimension that is implicit. Thus, on the first step (a), it is applied along the one dimension and, on the second step (b), along another dimension as depicted in fig 3.

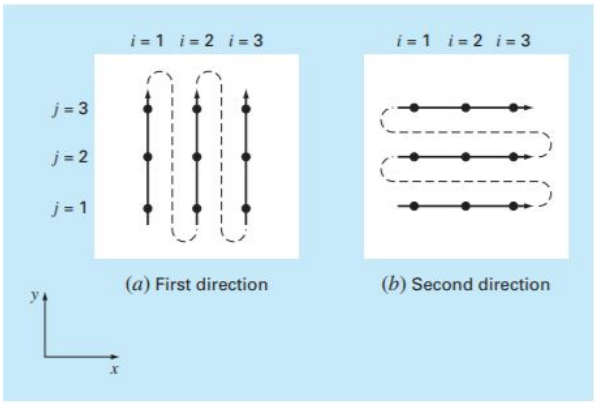

Fig 3. ADI Scheme

The discretized eq'n used is :

$$\textbf{Step 1:} \qquad \frac{u_{i,j}^{n+1/2} - u_{i,j}^{n}}{\Delta t/2} = \alpha\left(\hat{\delta}_x^2 u_{i,j}^{n+1/2} + \hat{\delta}_y^2 u_{i,j}^{n}\right)$$

$$\textbf{Step 2:} \qquad \frac{u_{i,j}^{n+1} - u_{i,j}^{n+1/2}}{\Delta t/2} = \alpha\left(\hat{\delta}_x^2 u_{i,j}^{n+1/2} + \hat{\delta}_y^2 u_{i,j}^{n+1}\right)$$

It is second-order accurate with T.E of order O[(delta_t)^2, (delta_x)^2, (delta_y)^2] It is also unconditionally stable with an amplification factor, G

$$G = \frac{[1 - r_x(1 - \cos\beta_x)][1 - r_y(1 - \cos\beta_y)]}{[1 + r_x(1 - \cos\beta_x)][1 + r_y(1 - \cos\beta_y)]}$$

where

$$r_x = \alpha\,\Delta t/(\Delta x)^2 \qquad \beta_x = k_m\,\Delta x$$
$$r_y = \alpha\,\Delta t/(\Delta y)^2 \qquad \beta_y = k_m\,\Delta y$$

Alternating-Direction Explicit (ADE) Method

In this method, step 1 marches the solution from the left boundary to the right boundary. By marching in the direction, u_{j-1}^{n+1} is always known, and consequently, u_j^{n+1} can be determined “explicitly”. In a like manner, step 2 marches the solution from the right boundary to the left boundary, again resulting in an “explicit” formulation, since u_{j+1}^{n+2} is always known

$$\textbf{Step 1:} \qquad \frac{u_j^{n+1} - u_j^n}{\Delta t} = \alpha \frac{u_{j-1}^{n+1} - u_j^{n+1} - u_j^n + u_{j+1}^n}{(\Delta x)^2}$$

$$\textbf{Step 2:} \qquad \frac{u_j^{n+2} - u_j^{n+1}}{\Delta t} = \alpha \frac{u_{j-1}^{n+1} - u_j^{n+1} - u_j^{n+2} + u_{j+1}^{n+2}}{(\Delta x)^2}$$

The above equation can be extended for 2-D also.

Hopscotch Method

It is an unconditionally stable explicit method. It also involves two steps. For step 1, $u_{i,j}^{n+1}$ is computed at each grid point (for which i+j+n is even) by the simple explicit scheme

$$\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} = \alpha\left(\hat{\delta}_x^2 u_{i,j}^n + \hat{\delta}_y^2 u_{i,j}^n\right)$$

For the second step, $u_{i,j}^{n+1}$ is computed at each grid point (for which i+j+n is odd) by the simple implicit scheme

$$\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} = \alpha\left(\hat{\delta}_x^2 u_{i,j}^{n+1} + \hat{\delta}_y^2 u_{i,j}^{n+1}\right)$$

Method

The above three Methods were used to compute the result using *Python* (For code see Appendix).

All the basic data is kept constant for all the three cases as mentioned below:

dx=dy=0.1 (unit)

dt=0.00001s

alpha=0.5

dt_stream = 0.002

epsilon_stream = 1.0e-2 (For convergence)

Results & Discussion

Temperature Distribution and Compilation time

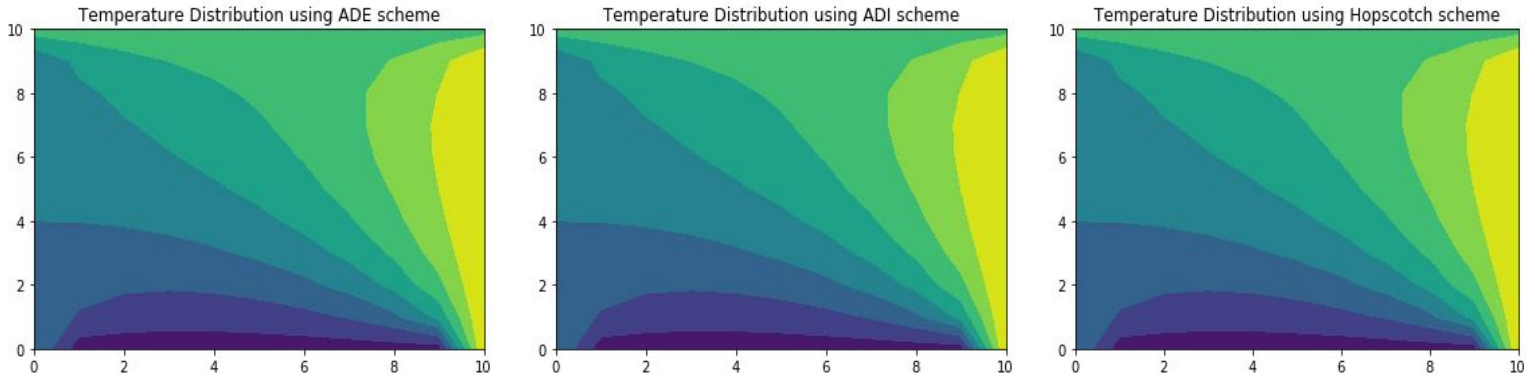

Fig 4. Temperature distribution using different schemes

As clearly seen there is no particular difference in the temperature distribution of all three schemes. All converge to the same result and hence in terms of temperature profile any of the one can be use.

But, the major difference in the scheme is of compilation time as shown in table 1:

Scheme	Compilation Time	Time Step
ADE	36.517s	346
Hopscotch	58.633s	692
ADI	70.382s	693

Table 1: Compilation time and a time step of different schemes

Error Plots

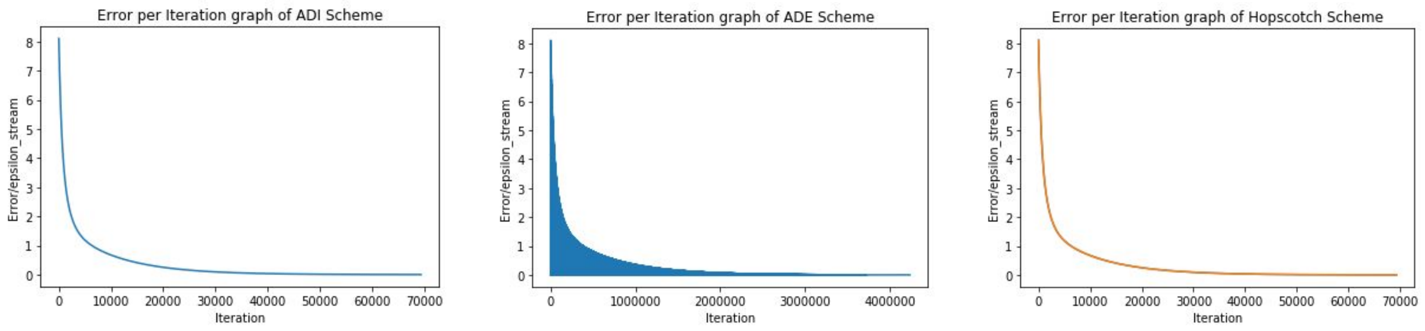

Fig 5: Error per itteration graph of schemes

Fig 5 shows the plot of error/epsilon_stream plotted against the iteration, As seen all three are almost similar, and they are almost exponentially decreasing functions.

Conclusion

From the above results, it can be concluded that the ADE is the fastest scheme as it solves the equations in just 36.517s in this case and hence shows that the explicit scheme is faster than the implicit schemes. Whereas, as we know that explicit schemes become unstable at larger delta_t, implicit schemes are more useful in such cases, but for small delta_t explicit schemes are efficient options.

Future Work

Comparison between more advance schemes can be done to get an efficient scheme even for larger delta_t.

Heat conduction with a crack in the domain can be done.

References

[1] Series in Computational and Physical Processes in Mechanics and Thermal Sciences by W.J. Minkowcycz and E.M. Sparrow

[2] Numerical Methods for Engineers by S.C. Chapra

[3] Dr. Gaurav Bhutani's CFD Class notes.

APPENDIX 1: ADE Code

```
import numpy as np
import matplotlib.pyplot as plt
import time

start=time.time();

L1=1;
L2=1;
T1=100;
T2=200;
T3=300;
T4=400;

dx=dy=0.1;
dt=0.00001;

alpha=0.5;

dt_stream = 0.002
epsilon_stream = 1.0e-2
rx=alpha*(dt)/dx**2;
ry=alpha*(dt)/dy**2;

N=int(L1/dx)+1;
M=int(L2/dy)+1;

T_n=np.zeros([N,M]);
T_n1=np.zeros([N,M]);
T_n2=np.zeros([N,M]);

# Boundary Condition's

for j in range (0,M):
    i=0;
    T_n[i][j]=T_n1[i][j]=T_n2[i][j]=T1;
for j in range (0,M):
    i=N-1;
    T_n[i][j]=T_n1[i][j]=T_n2[i][j]=T3;
for i in range (0,N-1):
    j=0;
    T_n[i][j]=T_n1[i][j]=T_n2[i][j]=T2;
for i in range (0,N-1):
    j=M-1;
    T_n[i][j]=T_n1[i][j]=T_n2[i][j]=T4;


#Time stepping
er=[];
for t in range (1000):
    for k in range (100):

        #step 1
        for i in range (1,N-1):
            for j in range (1,M-1):
                T_n1[i][j]=(rx*T_n1[i-1][j]+(1-ry-rx)*T_n[i][j]+rx*T_n[i+1][j]/
                    +ry*T_n1[i][j-1]+ry*T_n[i][j+1])/(1+rx+ry);
        #Step 2
        for i in range (N-2,0,-1):
            for j in range (M-2,0,-1):
                T_n2[i][j]=(rx*T_n1[i-1][j]+(1-ry-rx)*T_n1[i][j]+rx*T_n2[i+1][j]/
                    +ry*T_n1[i][j-1]+ry*T_n2[i][j+1])/(1+rx+ry);
        error = 0.0;
        for i in range(0,N):
            for j in range(0,M):
                error = error + (T_n1[i][j] - T_n[i][j])**2
                er.append(error);
                T_n[i][j] = T_n2[i][j]
        error = np.sqrt(error/(M*N))
        print("timestep", t, "error=", error/epsilon_stream, "iteration number", k)
        er.append(error/epsilon_stream);
    if (error/dt_stream)<epsilon_stream:
        break

plt.plot(er)
plt.xlabel('Iteration')
plt.ylabel('Error/epsilon_stream')
plt.title('Error per Iteration graph of ADE Scheme')
plt.show()
plt.contourf(T_n)
plt.title('Temperature Distribution using ADE scheme')


end=time.time();
print("Compilation time is: ")
print(end-start)
```

APPENDIX 2: ADI Code

```python
import numpy as np
import matplotlib.pyplot as plt
import time

start=time.time();

L1=1;
L2=1;
T1=100;
T2=200;
T3=300;
T4=400;

dx=dy=0.1;
dt=0.00001;
er=[];
alpha=0.5;
dt_stream = 0.002
epsilon_stream = 1.0e-2
rx=alpha*(dt/2)/dx**2;
ry=alpha*(dt/2)/dy**2;

N=int(L1/dx)+1;
M=int(L2/dy)+1;

T_n=np.zeros([N,M]);
T_nh=np.zeros([N,M]);
T_n1=np.zeros([N,M]);

# Boundary Condition's

for j in range (0,M):
    i=0;
    T_n[i][j]=T_nh[i][j]=T_n1[i][j]=T1;
for j in range (0,M):
    i=N-1;
    T_n[i][j]=T_nh[i][j]=T_n1[i][j]=T3;
for i in range (0,N-1):
    j=0;
    T_n[i][j]=T_nh[i][j]=T_n1[i][j]=T2;
for i in range (0,N-1):
    j=M-1;
    T_n[i][j]=T_nh[i][j]=T_n1[i][j]=T4;


#Time stepping

for t in range (1000):
    for k in range (100):

        #step 1
        for i in range (1,N-1):
            for j in range (1,M-1):
                T_nh[i][j]=(ry*T_n[i][j+1]+(1-2*ry)*T_n[i][j]+ry*T_n[i][j-1]+rx/
                    *T_nh[i+1][j]+rx*T_nh[i-1][j])/(1+2*rx);
        #step 2
        for j in range (1,M-1):
            for i in range (1,N-1):
                T_n1[i][j]=(rx*T_nh[i+1][j]+(1-2*rx)*T_nh[i][j]+rx*T_nh[i-1][j]/
                    +ry*T_n1[i][j+1]+ry*T_n1[i][j-1])/(1+2*ry);
        error = 0.0;
        for i in range(0,N):
            for j in range(0,M):
                error = error + (T_n1[i][j] - T_n[i][j])**2
                T_n[i][j] = T_n1[i][j]
        error = np.sqrt(error/(M*N))
        er.append(error/epsilon_stream);
        print("timestep", t, "error=", error/epsilon_stream, "iteration number", k)
    if (error/dt_stream)<epsilon_stream:
        break


plt.plot(er)
plt.xlabel('Iteration')
plt.ylabel('Error/epsilon_stream')
plt.title('Error per Iteration graph of ADI Scheme')
plt.show()
plt.contourf(T_n)
plt.title('Temperature Distribution using ADI scheme')

end=time.time();
print("Compilation time is: ")
print(end-start)
```

APPENDIX 3: Hopscotch Code

```
import numpy as np
import matplotlib.pyplot as plt
import time

start=time.time();

L1=1;
L2=1;
T1=100;
T2=200;
T3=300;
T4=400;

dx=dy=0.1;
dt=0.00001;

alpha=0.5;
dt_stream = 0.002
epsilon_stream = 1.0e-2
rx=alpha*(dt)/dx**2;
ry=alpha*(dt)/dy**2;

N=int(L1/dx)+1;
M=int(L2/dy)+1;

T_n=np.zeros([N,M]);
T_n1=np.zeros([N,M]);


# Boundary Condition's

for j in range (0,M):
    i=0;
    T_n[i][j]=T_n1[i][j]=T1;
for j in range (0,M):
    i=N-1;
    T_n[i][j]=T_n1[i][j]=T3;
for i in range (0,N-1):
    j=0;
    T_n[i][j]=T_n1[i][j]=T2;
for i in range (0,N-1):
    j=M-1;
    T_n[i][j]=T_n1[i][j]=T4;

#print(T_n)

#Time stepping
er=[];
for t in range (1000):
    for k in range (100):

        #step 1
        for i in range (1,N-1):
            for j in range (1,M-1):
                if (((i+j)%2)!=0):
                    T_n1[i][j]=T_n[i][j]+rx*(T_n[i+1][j]-2*T_n[i][j]+T_n[i-1][j])+ry/
                    *(T_n[i][j+1]-2*T_n[i][j]+T_n[i][j-1])
        #Step 2
        for i in range (1,N-1):
            for j in range (1,M-1):
                if (((i+j)%2)==0):
                    T_n1[i][j]=T_n[i][j]+rx*(T_n1[i+1][j]-2*T_n1[i][j]+T_n1[i-1][j])+ry/
                    *(T_n1[i][j+1]-2*T_n1[i][j]+T_n1[i][j-1])
        error = 0.0;
        for i in range(0,N):
            for j in range(0,M):
                error = error + (T_n1[i][j] - T_n[i][j])**2
                T_n[i][j] = T_n1[i][j]
        error = np.sqrt(error/(M*N))
        er.append(error/epsilon_stream);
        print("timestep", t, "error=", error/epsilon_stream, "iteration number", k)
    if (error/dt_stream)<epsilon_stream:
        break

plt.plot(er)
plt.xlabel('Iteration')
plt.ylabel('Error/epsilon_stream')
plt.title('Error per Iteration graph of Hopscotch Scheme')
plt.show()
plt.contourf(T_n)
plt.title('Temperature Distribution using Hopscotch scheme')

end=time.time();
print("Compilation time is: ")
print(end-start)
```