NUMERICAL SCHEME FOR HEAT EQUATION WITH FIRST BOUNDARY CONDITION ON SQUARE DOMAIN

YIPING LU

1. Numercial Schemes For Heat Equation With First Boundary Condition

In this report we consider the following problem:

$$u_t = \Delta u = u_{xx} + u_{yy}, u|_{\partial D\times(0,1]} = 0$$

Here $D = (0,1)\times(0,1)$ is a square domain on the plane.

If we give the initial condition $u(x,y,0) = \sin(\pi x)\sin(\pi y)$, this PDE can be easily solved with solution $u(x,y,t) = e^{-2\pi^2 t}\sin(\pi x)\sin(\pi y)$

1.1. Numerical Methods.

In this report we use several numerical methods to solve the PDE. We test the **explicit scheme, implicit scheme** and the **Crank-Nicolson Scheme**.

1.1.1. *Explicit Scheme.*

The forward explicit scheme can be simply written as

$$\frac{U_{j,k}^{m+1} - U_{j,k}^{m}}{h_t} = \frac{U_{j+1,k}^{m} - 2U_{j,k}^{m} + U_{j-1,k}^{m}}{h_x^2} + \frac{U_{j,k+1}^{m} - 2U_{j,k}^{m} + U_{j,k-1}^{m}}{h_y^2}$$

Equally

$$U_{j,k}^{m+1} = [1-2(\mu_x+\mu_y)]U_{j,k}^{m} + \mu_x(U_{j+1,k}^{m} + U_{j-1,k}^{m}) + \mu_y(U_{j,k+1}^{m} + U_{j,k-1}^{m})$$

Then the numerical error has the formula

$$e_{j,k}^{m+1} = [1-2(\mu_x+\mu_y)]e_{j,k}^{m} + \mu_x(e_{j+1,k}^{m} + e_{j-1,k}^{m}) + \mu_y(e_{j,k+1}^{m} + e_{j,k-1}^{m}) - T_{j,k}^{m}h_t$$

Here $T_{j,k}^{m} = Tu(x_j, y_k, t_m)$ is the local truncation error and it is easy to prove that $Tu(x,y,t) = \frac{1}{2}u_t t h_t - \frac{1}{12}(u_{xxxx}(x,y,t)h_x^2 + u_{yyyy}(x,y,t)h_y^2) + O(h_t^2 + h_x^4 + h_y^4)$

Consider the maximal principle which can be formatted as following:

Theorem(Maximal Principle) For the finite difference scheme L_h defined as

$$L_h U_j = \sum_{i\in J\backslash\{j\}} c_{i,j}U_i - c_jU_j$$

satisfies

- $J_D \neq \emptyset$ and $J = J_\Omega \cup J_D$ is connected
- for every $j \in J_\Omega$ we have $c_j, c_{i,j} > 0$ and $c_j \geq \sum_{i \in D_{L_h}(j)} c_{i,j}$

Then

$$\max_{i \in J_\Omega} U_i \leq \max\{\max_{i \in J_D} U_i, 0\}$$

To satisfy the Maximal Principle we need $\mu_x + \mu_y \leq \frac{1}{2}$

Next we apply Fourier analysis method to analysis the stability of the scheme. Let the Fourier wave as

$$U_{j,k}^m = \lambda_\alpha^m e^{i(\alpha_x x_j + \alpha_y y_k)}, \alpha = (\alpha_x, \alpha_y)$$

Then $\lambda_\alpha = 1 - 4(\mu_x \sin^2 \frac{\alpha_x h_x}{2} + \mu_y \sin^2 \frac{\alpha_y h_y}{2})$

1.1.2. *Implicit scheme.* Here we consider the implicit scheme has the following formula which contains the Crank-Nicolson Scheme as a special case

$$\begin{aligned}
\frac{U_{j,k}^{m+1} - U_{j,k}^m}{h_t} &= (1-\theta)(\frac{\delta_x^2}{h_x^2} + \frac{\delta_y^2}{h_y^2})U_{j,k}^m + \theta(\frac{\delta_x^2}{h_x^2} + \frac{\delta_y^2}{h_y^2})U_{j,k}^{m+1} \\
&= (1-\theta)(\frac{U_{j+1,k}^m - 2U_{j,k}^m + U_{j-1,k}^m}{h_x^2} + \frac{U_{j,k+1}^m - 2U_{j,k}^m + U_{j,k-1}^m}{h_y^2}) + \\
&\theta(\frac{U_{j+1,k}^{m+1} - 2U_{j,k}^{m+1} + U_{j-1,k}^{m+1}}{h_x^2} + \frac{U_{j,k+1}^{m+1} - 2U_{j,k}^{m+1} + U_{j,k-1}^{m+1}}{h_y^2})
\end{aligned}$$

For the Fourier wave $U_{j,k}^m = \lambda_\alpha^m e^{i(\alpha_x x_j + \alpha_y y_k)}$, we have

$$\lambda_\alpha = \frac{1 - 4(1-\theta)(\mu_x \sin^2 \frac{\alpha_x h_x}{2} + \mu_y \sin^2 \frac{\alpha_y h_y}{2})}{1 + 4\theta(\mu_x \sin^2 \frac{\alpha_x h_x}{2} + \mu_y \sin^2 \frac{\alpha_y h_y}{2})}$$

In order to obtain the stability we need

$$2(\mu_x + \mu_y)(1 - 2\theta) \leq 1, 0 \leq \theta \leq \frac{1}{2}$$

Why Crank-Nicolson Scheme Choose $\theta = \frac{1}{2}$

The local truncation error can be proof to be $O(h_t^2 + h_x^2 + h_y^2)$ when $\theta = \frac{1}{2}$ and to be $O(h_t + h_x^2 + h_y^2)$ when θ is other value.

2. Numerical Linear Algebra Methods Used In Solving The Scheme

2.1. **Classical Iterative Methods:G-s and CG.** Analysis see in textbooks.

2.2. **Multi-grid.** Analysis see in appendix.(Which will be updated at later version)

3. Numerical Results

3.1. **Implement Details.** In the Crank-Nicolson Scheme test, we use the **gmres,pcg** in matlab with can transfer a function handler as the parameter.

3.2. **Performance of different schemes.** In this section, all of the results is using **pcg** in order to get fast and robust solutions:

Test	Space step	Time step	CN Scheme	Implicit Scheme	Explicit Scheme
1	1/512	1/50	2.44e-7	6.88e-5	1.39e56
2	1/512	1/500	2.40e-11	5.79e-7	NaN
3	1/512	1/5000	**3.97e-16**	5.674e-9	1.393e56
4	1/128	1/50	2.43e-7	6.88e-5	7.80e32
5	1/128	1/500	1.55e-11	5.81e-7	7.79e32
6	1/128	1/5000	1.04e-12	5.82e-9	7.79e32
7	1/32	1/50	2.28e-7	6.90e-5	2.71e201
8	1/32	1/500	1.32e-10	6.02e-7	5.23e8
9	1/32	1/5000	2.55e-10	8.20e-9	3.62e-9

Table 1. Error at time $t = 0.2$

3.3. **Time that different NLA methods take.** In this part we test several NLA methods in the implicit scheme and plot the result in the table below

Test	Space step	Time step	Cholesky	Multigrid	Gauss-Seidel
1	1/32	1/20	0.08	0.19	0.38
2	1/64	1/20	0.31	0.37	1.48
3	1/128	1/20	5.36	**1.31**	6.45

Table 2. CPU time of different NLA methods while calculating the solution at $t = 0.2$

3.4. **Some interesting observation.**

- Sometimes the pde solver may become faster if the time step becomes smaller. It seems that the iterative number will become bigger, but at the same time the algebra equation $(I - \Delta t\Delta)u_{t+1} = u_t$ becomes easier.
- Stupid methods is faster at easy situation.
- CN scheme is also faster than implicit scheme.

4. Readme

In this section, I will introduce the arrangement of my code. To show the demo, you can run the code in **runme.m**

4.1. Usage of the code.

To get the answer you can use the functions **CN_heat.m,explicit_heat.m,implicit_heat.m**

The choice of **Scheme_tool.method** have the following choices:

For Crank-Nicolson Scheme, Scheme_tool.method can choose

- gmres
- pcg
- cg

For Implicit Scheme, Scheme_tool.method can choose

- gs
- pcg
- cg
- gmres
- test
- multigrid
- chol

4.2. Arangement of the code.

Code in different paths is different usage:

- **matrix_func**: Numerical Linear Algebra functions and the code to calculate the matrix of laplace operator.
- **multigrid**: Functions for multigrid methods including restrict and lifting operator, V cycle and the multigrid method for heat equation.
- **op**:Some useful tools to change the grid to a vector
- **pde_func**: PDE solvers for heat equation and the true answer to test
- **plot_func**: To turn the PDE's solution to a video.
- **tool:** Two matlab classes to show the tol and max_int of different methods