Project 1: 求解Poisson方程五点差分格式的快速算法及数值比较实验

纪经纬 5141209098

April 10, 2018

1 问题的提出

考虑带Dirichlet边界条件的Poisson调和方程：

$$\begin{cases} -\Delta u = f \quad in\ \Omega \\ u = \alpha \quad on\ \partial\Omega \end{cases} \tag{1.1}$$

式中$\Omega = (0,a)\times(0,b)$,$f$是定义在$\Omega$上的函数，$\alpha$是定义在$\partial\Omega$ 上的函数。为利用差分法进行数值求解$u(x,y)$，首先将区域Ω进行网格剖分，在x方向和y方向分别进行$I+1$和$J+1$等分，则x方向和y方向的步长分别是

$$h = \frac{a}{I+1}, k = \frac{b}{J+1} \tag{1.2}$$

那么内部点就是$x_i = ih, y_j = jk. 0 < i < I+1, 0 < j < J+1.$

然后进行五点差分，格式如下：

$$\begin{cases} -\Delta_h u_{ij} = f_{ij}, \quad (x_i,y_j)\in\Omega_h \\ u_{ij} = \alpha_{ij}, \quad (x_i,y_j)\in\partial\Omega_h \end{cases} \tag{1.3}$$

式中

$$\Delta_h u_{ij} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i,j-1}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2} \tag{1.4}$$

不失一般性，假设在单位正方形区域考虑，$a = b = 1$。那么这个问题的离散形式可以用矩阵方程表示：

$$AU + UB = F \tag{1.5}$$

其中，

$$A = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}_{I\times I}, \quad B = \frac{1}{k^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}_{J\times J},$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,J} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,J} \\ \cdots & \cdots & \cdots & \cdots \\ u_{I,1} & u_{I,2} & \cdots & u_{I,J} \end{bmatrix}_{I\times J},$$

$$F = \begin{bmatrix} f_{1,1} + \frac{1}{h^2}u_{0,1} + \frac{1}{k^2}u_{1,0} & f_{1,2} + \frac{1}{h^2}u_{0,2} & \cdots & f_{1,J-1} + \frac{1}{h^2}u_{0,J-1} & f_{1,J} + \frac{1}{h^2}u_{0,J} + \frac{1}{k^2}u_{1,J+1} \\ f_{2,1} + \frac{1}{k^2}u_{2,0} & f_{2,2} & \cdots & f_{2,J-1} & f_{2,J} + \frac{1}{k^2}u_{2,J+1} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ f_{I-1,1} + \frac{1}{k^2}u_{I-1,0} & f_{I-1,2} & \cdots & f_{I-1,J-1} & f_{I-1,J} + \frac{1}{k^2}u_{I-1,J+1} \\ f_{I,1} + \frac{1}{h^2}u_{I+1,1} + \frac{1}{k^2}u_{I,0} & f_{I,2} + \frac{1}{h^2}u_{I+1,2} & \cdots & f_{I,J-1} + \frac{1}{h^2}u_{I+1,2} & f_{I,J} + \frac{1}{h^2}u_{I+1,J} + \frac{1}{k^2}u_{I,J+1} \end{bmatrix}_{I\times J}$$

$$= [f_{i,j}]_{I\times J} + \frac{1}{h^2}\begin{bmatrix} u_{0,1} & u_{0,2} & \cdots & u_{0,J-1} & u_{0,J} \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ u_{I+1,1} & u_{I+1,2} & \cdots & u_{I+1,J-1} & u_{I+1,J} \end{bmatrix}_{I\times J} + \frac{1}{k^2}\begin{bmatrix} u_{1,0} & 0 & \cdots & 0 & u_{1,J+1} \\ u_{2,0} & 0 & \cdots & 0 & u_{2,J+1} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ u_{I-1,0} & 0 & \cdots & 0 & u_{I-1,J+1} \\ u_{I,0} & 0 & \cdots & 0 & u_{I,J+1} \end{bmatrix}_{I\times J}$$

本文就是基于这个问题，对五点差分格式的DST快速算法、Jacobi迭代法、G-S迭代法进行比较实验，分别从计算的耗时、精确程度去衡量算法的优劣。为了进行数值实验，选取

$$u(x,y) = \sin(2\pi nx) + sin(2\pi ny) + x^2 \tag{1.6}$$

那么，

$$-\Delta u(x,y) = 4n^2\pi^2[\sin(2\pi nx) + \sin(2\pi ny)] - 2 = f(x,y) \tag{1.7}$$

其中n越大，原函数震荡的频率越高，对网格剖分的精细程度也就越高。对应的Matlab代码为：

```
function z = u( X, Y, n )
% u(x,y), and n is a parameter to control the frequency
    z = sin(2*pi*n.*X)+ sin(2*pi*n.*Y) + X.*X  ;
end
```

和

```
function z = f( X, Y, n )
%F the function on the right hand side of (1.1)
    z = 4 * n*n*pi*pi*(sin(2*n*pi.*X)+sin(2*n*pi.*Y) )  - 2  ;
end
```

先观察一下解的形状：

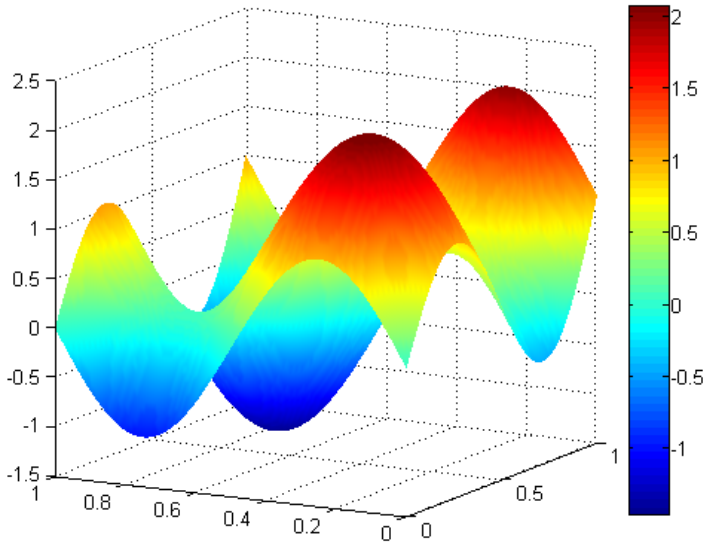

Figure 1: 曲面$u(x,y) = \sin(2\pi nx) + sin(2\pi ny) + x^2,\ n = 1$

2 求解的算法

2.1 DST快速算法

记$P=[\sin(ij\pi h)]_{I\times I}, Q=[\sin(ij\pi k)]_{J\times J}$.对于矩阵$H_N=\begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & 1 \\ & & & & 1 & 0 \end{bmatrix}$,有特征值$\lambda_k=2\cos(\frac{k\pi}{N+1}), k=1,2,...,N$.那么$A,B$矩阵分别有特征值

$$\begin{aligned} \lambda_k(A) &= \frac{1}{h^2}[2-2\lambda_k(H_I)] \\ &= \frac{4}{h^2}\sin^2[\frac{k\pi}{2(I+1)}], \quad k=1,2,...,I \end{aligned} \tag{2.1}$$

和

$$\mu_k(B)=\frac{4}{k^2}\sin^2[\frac{k\pi}{2(J+1)}], \quad k=1,2,...,J \tag{2.2}$$

根据课堂上的推导，对$u_{ij}\in\Omega_h$:

$$U=PWQ \tag{2.3}$$

其中，$w_{ij}=\frac{4hk}{\lambda_i+\mu_j}[PFQ]_{ij}$.

由此可以得到以下算法：

1. 在单位正方形区域进行等距划分$h=\frac{1}{I+1}, k=\frac{1}{J+1}$，并且$I=J$[1].
   计算向量$\lambda=[\lambda_i=\frac{4}{h^2}\sin^2[\frac{i\pi h}{2}]]_{I\times 1}$
   计算向量$\mu=[\lambda_i=\frac{4}{k^2}\sin^2[\frac{j\pi k}{2}]]_{J\times 1}$
   根据节1计算矩阵F

2. 利用`dst`函数计算矩阵[2]
   $$V=PFQ$$

3. 计算矩阵$W=[w_{ij}]$,
   $$w_{ij}=\frac{4hkv_{ij}}{\lambda_i+\mu_j}$$

4. 利用`dst`函数计算矩阵$U=PWQ$.

对应的Matlab代码为：

```
function U = DSTPS(N, n)
% Main program. Domain = [0,1]x[0,1]
% Input: N: number of division :N+1 & I=J
%        n: parameter of the function on the RHS
% Output: matrix of U
    tic
    % Step 1:
    h = 1/(N+1);     % step length
    x = h*(1:N);     % vector of interior points
    lambda = 4 * sin(pi*x/2) .* sin(pi*x/2) * (N+1)^2 ;

```

[1]这个假设的目的是因为Matlab中的函数`dst`适用于$I=J$的情形
[2]Matlab命令是`V=dst(dst(F)')'`

```
    [X,Y] = meshgrid(x,x);
    F = f(X,Y, n);
    real_U = u(X,Y, n);  % interior points of U
    h_matrix = zeros(N,N);
    k_matrix = zeros(N,N);
    h_matrix(1,:) = u(x,0,n)';
    h_matrix(N,:) = u(x,1,n)';
    k_matrix(:,1) = u(0,x,n);
    k_matrix(:,N) = u(1,x,n);
    F = F+ (N+1)*(N+1).*(h_matrix+k_matrix);

    % Step 2:
    V = dst(dst(F)')';

    % Step 3:
    W = zeros(N-1);
    for i = 1:N
        for j = 1:N
            W(i,j) = (4*h*h*V(i,j)) / (lambda(i)+lambda(j));
        end
    end

    % Step 4:
    U = dst(dst(W)')';
    toc
    % Compare the computed result with the true values, and then plot
        :
    diff = real_U - U;
    surf(X,Y,diff),shading interp;colorbar
end
```

2.2 Jacobi迭代法

这是求解线性方程组$Ax=b$的基本迭代方法。基本思想是将矩阵A分裂，$A=M-N$，M可逆，进行如下迭代：

$$\begin{cases} \text{Guess } x_0 \in \mathbb{R}^n \\ Mx_{k+1} = Nx_k + b \end{cases} \tag{2.4}$$

希望$\lim_{k\to\infty} x_k = x^*$,$Ax^* = b$.

具体地，Jacobi迭代中取$M=D=diag(a_{11},a_{22},...,a_{nn})$，进行迭代$Dx_{k+1}=(A-D)x_k+b$，那么具体考察左边矩阵的非零项，有

$$a_{ii}x_i^{(k+1)} = -\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + b_i, \quad i=1,2,...,n \tag{2.5}$$

可以看出本质上它就是利用上一次迭代的整个得到的值来计算当前迭代网格中每个点的值。因此，正如课堂上所教授的方法，可以不用写成具体的矩阵形式，只要在每一次迭代中遍历每个内点$(x_i,y_j\in\Omega_h)$,利用以下方程，把k次的值看成已知求解$k+1$次即可：

$$\frac{u_{i+1,j}^{(k)} - 2u_{i,j}^{(k+1)} + u_{i,j-1}^{(k)}}{h^2} + \frac{u_{i,j+1}^{(k)} - 2u_{i,j}^{(k+1)} + u_{i,j-1}^{(k)}}{k^2} = f_{ij} \tag{2.6}$$

在判断收敛条件时，我采用了老师给的方法，利用矩阵的Frobenius范数：

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{2.7}$$

代码如下：

```
function U = PSJacobi(N, n, stopThreshold)
% Jacobi iteration to solve Poisson equation
% Main program. Domain = [0,1]x[0,1]
% Input: N: number of division :N+1 & I=J
%        n: parameter of the function on the RHS
    tic
    h = 1/N;
    x = h*(0:N);
    [X,Y] = meshgrid(x,x);
    F = h*h* f(X,Y, n);
    real_U = u(X,Y, n);
    U = zeros(N+1);
    U(1,:) = real_U(1,:);
    U(N+1,:) = real_U(1,:);
    U(:,1) = real_U(:,1);
    U(:,N+1) = real_U(:,N+1);
    V = U;
    flag = 1;
    while flag > stopThreshold
        for i = 2:N
            for j = 2:N
                V(i,j) = (U(i-1,j)+U(i+1,j)+U(i,j-1)+U(i,j+1)+F(i,j))
                    /4;
            end
        end
        flag = norm(V-U, 'fro');
        U = V;
    end
    toc
    diff = real_U - U;
    surf(X,Y,diff),shading interp;colorbar
    max_error = max(max(diff))
end
```

2.3 G-S迭代法

类似地，迭代方程

$$\frac{u_{i+1,j}^{(k)} - 2u_{i,j}^{(k+1)} + u_{i,j-1}^{(k+1)}}{h^2} + \frac{u_{i,j+1}^{(k)} - 2u_{i,j}^{(k+1)} + u_{i,j-1}^{(k+1)}}{k^2} = f_{ij} \tag{2.8}$$

如果都是串行实现，那么G-S方法优于Jacobi方法，因为它在新的一次迭代中实时利用了新算得的值，收敛得也更快了。但是它在每一次迭代中建立了这些求解点之间的一个顺序依赖，使得只能串行实现。然而Jacobi方法在一次迭代中没有这样的依赖，可以多线程实现。因此两者各有利弊。按照本文的代码实现，G-S方法应当更胜一筹。

该算法代码只需在Jacobi迭代法的基础上将内循环稍加改动即可；

即改成`V(i,j)=(V(i-1,j)+U(i+1,j)+V(i,j-1)+U(i,j+1)+F(i,j))/4;`。

2.4 二维的紧致差分格式

根据课堂上的推导，紧致差分格式可以进行如下推广：不妨令$I = J = N$

$$AU + UB = WF + FW + G \tag{2.9}$$

其中，

$$A = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}_{N\times N}, \quad B = \frac{1}{k^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}_{N\times N},$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,N} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,N} \\ \cdots & \cdots & \cdots & \cdots \\ u_{N,1} & u_{N,2} & \cdots & u_{N,N} \end{bmatrix}_{N\times N}, \quad F = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,N} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,N} \\ \cdots & \cdots & \cdots & \cdots \\ f_{N,1} & f_{N,2} & \cdots & f_{N,N} \end{bmatrix}_{N\times N},$$

$$W = \frac{1}{12}\begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{bmatrix}_{N\times N}$$

$$G = \begin{bmatrix} \frac{1}{12}f_{0,1} + \frac{1}{h^2}u_{0,1} & \frac{1}{12}f_{0,2} + \frac{1}{h^2}u_{0,2} & \cdots & \frac{1}{12}f_{0,N-1} + \frac{1}{h^2}u_{0,N-1} & \frac{1}{12}f_{0,N} + \frac{1}{h^2}u_{0,N} \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{12}f_{N+1,1} + \frac{1}{h^2}u_{N+1,1} & \frac{1}{12}f_{N+1,2} + \frac{1}{h^2}u_{N+1,2} & \cdots & \frac{1}{12}f_{N+1,N-1} + \frac{1}{h^2}u_{N+1,N-1} & \frac{1}{12}f_{N+1,N} + \frac{1}{h^2}u_{N+1,N} \end{bmatrix}_{N\times N}$$

$$+ \begin{bmatrix} \frac{1}{12}f_{1,0} + \frac{1}{k^2}u_{1,0} & 0 & \cdots & 0 & \frac{1}{12}f_{1,N+1} + \frac{1}{k^2}u_{1,N+1} \\ \frac{1}{12}f_{2,0} + \frac{1}{k^2}u_{2,0} & 0 & \cdots & 0 & \frac{1}{12}f_{2,N+1} + \frac{1}{k^2}u_{2,N+1} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \frac{1}{12}f_{N-1,0} + \frac{1}{k^2}u_{N-1,0} & 0 & \cdots & 0 & \frac{1}{12}f_{N-1,N+1} + \frac{1}{k^2}u_{N-1,N+1} \\ \frac{1}{12}f_{N,0} + \frac{1}{k^2}u_{N,0} & 0 & \cdots & 0 & \frac{1}{12}f_{N,N+1} + \frac{1}{k^2}u_{N,N+1} \end{bmatrix}_{N\times N}$$

那么，只要将原来算法中的F项替代为$WF + WF + G$即可。代码如下：

```
(* ::Package:: *)

function U = PSCompact(N, n)
% Main program. Domain = [0,1]x[0,1]
% Input: N: number of division :N+1 & I=J
%        n: parameter of the function on the RHS
% Output: matrix of U
    tic
    % Step 1:
    h = 1/(N+1);    % step length
    x = h*(1:N);    % vector of interior points
    lambda = 4 * sin (pi*x/2) .* sin(pi*x/2) * (N+1)^2 ;

```

```
    [X,Y] = meshgrid(x,x);
    F = f(X,Y, n);
    W = 1/12* (4*eye(N)+diag(ones(N-1,1),1)+diag(ones(N-1,1),-1));
    real_U = u(X,Y, n);  % interior points of U
    h_matrix = zeros(N,N);
    k_matrix = zeros(N,N);
    h_matrix(1,:) = 1/12*f (x,0,n)' + (N+1)*(N+1).* u (x,0,n)';  % the first row
    h_matrix(N,:) = 1/12*f (x,1,n)' + (N+1)*(N+1).* u (x,1,n)';  % the last row
    k_matrix(:,1) = 1/12*f(0,x,n) + (N+1)*(N+1).* u(0,x,n);
    k_matrix(:,N) = 1/12*f(1,x,n) + (N+1)*(N+1).* u(1,x,n);
    G = zeros(N,N)+ (h_matrix+k_matrix);
    F = F*W+W*F + G;
    % Step 2:
    V = dst (dst (F)')';

    % Step 3:
    W = zeros(N-1);
    for i = 1:N
        for j = 1:N
            W(i,j) = (4*h*h*V(i,j)) / (lambda(i)+lambda(j));
        end
    end

    % Step 4:
    U = dst (dst (W)')';
    toc
    % Compare the computed result with the true values, and then plot:
    diff = real_U - U;
    surf(X,Y,diff),shading interp;colorbar
    % Output the max absolute error
    max_error = max(max(diff))
    % str=['the value of pi=' num2str(pi)];
    % disp(str);
end
```

3 数值实验[3]

3.1 DST快速算法

为了初步检查算法的逐点收敛性，固定$n=1$，取$N=500$，绘制绝对误差曲面如下：

[3]本节所有实验数据都是清理内存后10次取平均后得到的

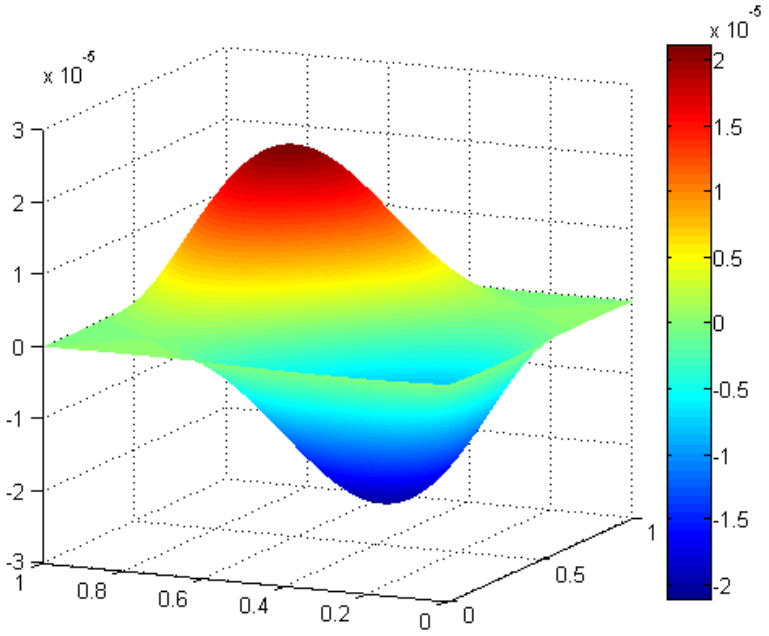

Figure 2: DST算法，取N=500时的绝对误差曲面：$u(x,y) - u_{ij}$；耗时0.114842秒

可以看到对于该函数，绝对误差被控制在了10^{-5}量级。正如我们所预料的，可以直观地看到，由于边界一定是精确解，在边界处误差为零，然后往中心区域误差渐渐累积，在内部某处达到峰值。

Table 1: DST快速算法计算实验：固定$n = 1$，变化N

N	计算时间（秒）	最大绝对误差
10	0.000842	4.380E-02
100	0.004638	5.177E-04
250	0.023669	8.38E-05
500	0.110068	2.10E-05
750	0.251897	9.37E-06
1000	0.690805	5.272E-06
2500	3.351052	8.45E-07
5000	12.36696	2.11E-07
7500	28.32574	9.39E-08
10000	75.75864	5.282E-08

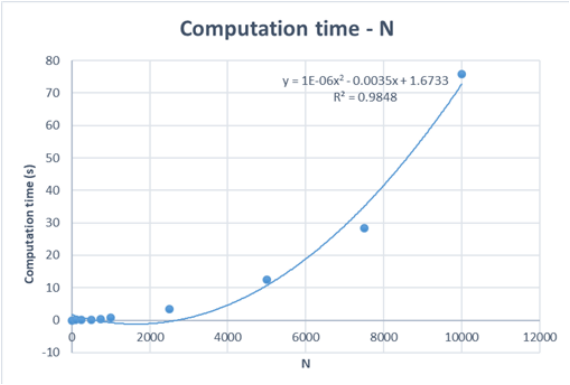

Figure 3: Computation time vs N, for fixed n

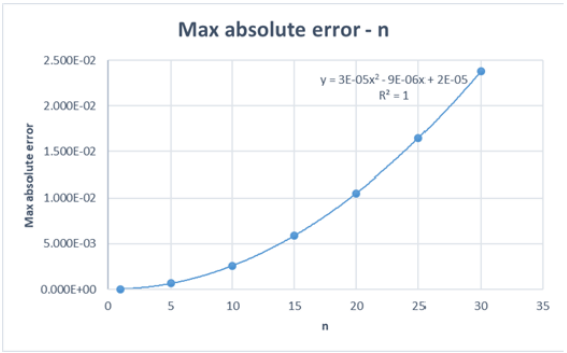

Figure 4: Max absolute error vs n, for fixed N

从表格中看到，正如课堂上给出的，Poisson方程五点差分格式的最大模估计是二阶的。该算法的时间复杂度也是优秀的，DST快速算法将一个$O(n^3)$的矩阵乘法运算大大优化了。按照Matlab的算法，理想状况下大致在$O(n\log(n))$。而且从空间复杂度来说，它也是非常高效的，不用开辟额外的内存空间。这是该算法的强大之处。

从另一个角度考察，由于网格的剖分是固定的，所以函数如果在内部震荡越频繁，其精度应该也越差。我们用实验验证猜想：

Table 2: DST快速算法计算实验：固定$N = 500$，变化n

n	计算时间（秒）	最大绝对误差
1	0.1110	2.105E-05
5	0.1223	6.553E-04
10	0.1105	2.600E-03
15	0.1137	5.900E-03
20	0.1128	1.050E-02
25	0.1158	1.650E-02
30	0.1121	2.380E-02

结果符合预期，计算时间不受太大影响，但是精度受到函数震荡频率加剧的显著的削弱；其计算精度和n的关系实验发现呈现较符合二次函数关系。

3.2 紧致差分格式的DST快速算法

事实证明紧致差分的DST快速算法是非常强大的，达到了本文所有格式的最高的四阶精度，而且速度非常快。最后猜测是计算机的浮点数精度所限所以精度回到了二阶。

Table 3: 五点紧致差分快速DST算法

N	计算时间（秒）	最大绝对误差
10	0.000745	7.119E-04
100	0.004999	1.002E-07
1000	0.530032	1.04E-11
10000	125.8643	3.08E-13

3.3 Jacobi迭代法和G-S迭代法

这类方法一部分原因受限于Matlab的循环性能，显得非常低效。Jacobi方法的时间复杂度随

网格剖分数N的增长是令人无法接受的。其最大绝对误差随着精度要求stopThreshold的增加接近线性地更加精确。

Table 4: Jacobi方法计算实验

N	计算时间（秒）	stopThreshold	最大绝对误差
50	0.5711	1.00E-03	2.030E-02
100	2.7641	1.00E-02	2.923E-01
100	7.3579	1.00E-03	4.010E-02
100	14.1271	1.00E-04	4.100E-03
100	21.0041	1.00E-05	7.962E-04
100	27.9219	1.00E-06	5.524E-04
200	95.3443	1.00E-03	7.780E-02
300	∞	1.00E-03	-

对于G－S方法，我们看到结果稍好，但是与Jacobi迭代法没有量级的区别。

Table 5: G-S方法计算实验

N	计算时间（秒）	stopThreshold	最大绝对误差
50	0.3375	1.00E-03	1.040E-02
100	1.9042	1.00E-02	1.634E-01
100	4.7200	1.00E-03	2.010E-02
100	8.0380	1.00E-04	2.200E-03
100	11.6260	1.00E-05	6.531E-04
100	14.9427	1.00E-06	5.400E-04
200	64.0958	1.00E-03	4.010E-02
300	∞	1.00E-03	-

可以预见的是这样的性能是无法应付求解高频波的，在可忍受的时间内算法的收敛性都无法保证。下面简单的实验就可以说明，对于频率稍高的原函数，该方法已经无能为力了。

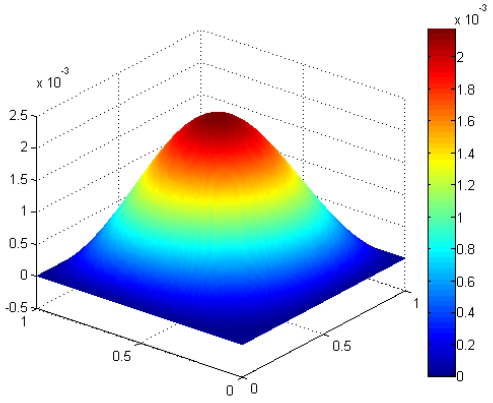

Figure 5: G-S:n=1,stopThreshold=0.0001

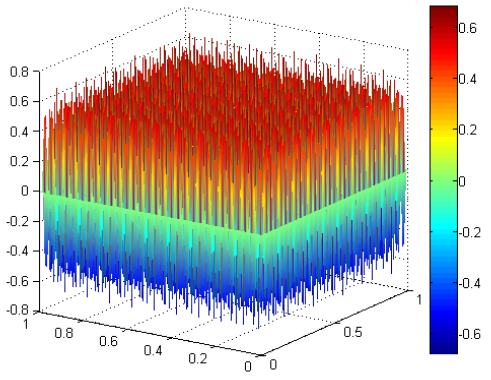

Figure 6: G-S:n=30,stopThreshold=0.0001

3.4 结语

本次作业利用了一个简单的算例，比较了二维非齐次边界条件的Poisson方程五点差分格式的几个算法。综合各个方面，快速DST方法是更胜一筹的；并且紧致差分算法的精度是最高的。