Numerical Experiment A

Xinzhi Li

Student ID: **2022211084**

School of Physics Science and Technology, ShanghaiTech University, Shanghai 201210, China

Email address: lixzh2022@shanghaitech.edu.cn

October 30, 2022

I. INTRODUCION

Poisson' equation is an elliptic partial differential equation of broad utility in theoretical physics. For example, the solution to ***Poisson's equation*** is the potential field caused by a given electric charge or mass density distribution; with the potential field known, one can then calculate electrostatic or gravitational field. It is a generalization of *Laplace's equation*, which is aslo frequently seen in physics. However, it is quite complicated to obtain the exact solution. Sometimes, we can only get the formal solution through *Green's function*. In practice, we translate the continuous equation into a series of linear equations, and then perform a numerical method to calculate the approximate solution. In the following the section, we will use three different iterations, **Jacobi**, **Gauss-Seidel** and **SOR** to sovle a concrete problem and make some analyses.

II. PROBLEM

Considering a specific *Poisson' equation*:

$$\begin{cases} -\left(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}\right)=f(x,y), 0<x,y<1 \\ u(0,y)=u(1,y)=u(x,0)=u(x,1)=0 \end{cases} \tag{1}$$

Set $f(x,y)=2\pi^2\sin(\pi x)\sin(\pi y)$, then the exact solution of Eq (1) becomes

$$u^*(x,y)=\sin(\pi x)\sin(\pi y) \tag{2}$$

Set $h=\dfrac{1}{N}, N\in\mathcal{N}^+, x_i=ih, y_j=jh, u_{i,j}\approx u(x_i,y_j), f_{i,j}=f(x_i,y_j)$.

$$\begin{cases} \left.\dfrac{\partial^2 u}{\partial x^2}\right|_{x=x_i,y=y_j} = \dfrac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2} \\ \left.\dfrac{\partial^2 u}{\partial y^2}\right|_{x=x_i,y=y_j} = \dfrac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{h^2} \end{cases} \tag{3}$$

Then we can translate Eq (1) into the linear equations:

$$\begin{cases} -u_{i-1,j}-u_{i,j-1}+4u_{i,j}-u_{i+1,j}-u_{i,j+1}=h^2 f_{i,j} \\ u_{i,0}=u_{0,j}=u_{i,N}=u_{N,j}=0, \qquad i,j=1,2,\cdots,N-1 \end{cases} \tag{4}$$

It can be also written as

$$L_h u^h = h^2 f^h \tag{5}$$

with

$$u_j^h=\begin{vmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{N-1,j}\end{vmatrix} \quad f_j^h=\begin{vmatrix} f_{1,j} \\ f_{2,j} \\ \vdots \\ f_{N-1,j}\end{vmatrix} \quad u^h=\begin{vmatrix} u_1^h \\ u_2^h \\ \vdots \\ u_{N-1}^h\end{vmatrix} \quad f^h=\begin{vmatrix} f_1^h \\ f_2^h \\ \vdots \\ f_{N-1}^h\end{vmatrix}$$

$$C=\begin{bmatrix} 0 & 1 & \cdots & & & \\ 1 & 0 & 1 & & & \\ \vdots & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix}_{(N-1)\times(N-1)} \qquad L_h=\begin{bmatrix} 4I-C & -I & \cdots & & \\ -I & 4I-C & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & 4I-C & -I \\ & & & -I & 4I-C \end{bmatrix} \tag{6}$$

III. NUMERICAL RESULTS

A. Experiment 1

Set $h = 0.1$ and use three iterations to obtain the solution of the Eq (5), under the condition $\|u^{(k+1)} - u^{(k)}\|_\infty < \epsilon, \epsilon = 10^{-6}$. Evaluate the norm $\|u^{(k+1)} - u^*\|_\infty$. For the **SOR** iteration, $\omega = 1.2, 1.3, 1.9, 0.9$. The result is shown in Table 1.

Table 1: Iteration Results in Experiment 1

Method	Iterations	$\|u^{(k+1)} - u^*\|_\infty$	Radius of convergence ρ
Jacobi iteration	217	0.008247	0.951057
Gauss-Seidel iteration	116	0.008256	0.904508
SOR iteration($\omega = 1.2$)	79	0.008260	0.855750
SOR iteration($\omega = 1.3$)	63	0.008262	0.818687
SOR iteration($\omega = 1.9$)	126	0.008266	0.900000
SOR iteration($\omega = 0.9$)	140	0.008254	0.921804
SOR iteration($\omega = 1.6$)	29	0.008266	0.600000
SOR iteration($\omega = 1.7$)	41	0.008265	0.700000

B. Experiment 2

Use Jacobi iteration to solve the equation, with different steps $h = 0.1, 0.05, 0.02, 0.01$. Known that $\rho(J) = 1 - 2\sin^2 \dfrac{\pi h}{2} \approx 1 - \dfrac{\pi^2 h^2}{2}$. The result is shown in Table 2.

IV. ANALYSIS AND REMARK

Table 1 shows that, under the same accuracy requirement, the speed of Gauss-Seidel iteration is roughly twice of the Jacobi's. This is consistent with the theory for the tridiagonal matrix:

$$\begin{cases} \rho(G) \approx \rho(J)^2 \\ R(G) = -\ln \rho(G) \approx -\ln \rho(J)^2 = -2\ln \rho(J) = 2R(J) \end{cases} \tag{7}$$

Moreover, one can find that, for the **SOR** iteration, the iterations can be much less than the Jacobi iteration. The optimal ω is about 1.6, which is very similar to the theoretical value:

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - (\rho(J))^2}} \tag{8}$$

Shown in Table 2, the iterations becomes larger and larger when the length of step goes smaller. One of reasons could be the speed of the iteration is slower with the increasing of the radius of convergence ρ (since $\rho \approx 1 - \dfrac{\pi^2 h^2}{2}$). The accuray increases with the smaller h. However, when $h = 0.01$, the $\|u^{(k+1)} - u^*\|_\infty$ is larger than that in $h = 0.02$. This is because the speed of convergence is too slow to get the more accurate result.

On the other hand, the different function in Matlab will influence the accuracy. For example, if one use $A\backslash b$ instead of $inv(A) * b$, the iteration result will be very different. The solution may converge to another point!

Table 2: Iteration Results in Experiment 2

Steps	Iterations	$\|u^{(k+1)} - u^*\|_\infty$	Radius of convergence ρ
$h = 0.1$	217	0.008247	0.951057
$h = 0.05$	762	0.001979	0.987688
$h = 0.02$	3843	0.000176	0.998027
$h = 0.01$	12566	0.001943	0.999507

In conclusion, the SOR iteration is much better than other two iterations, and one should be careful to choose an appropritae step h.

V. CODE

All the experiments are performed by Matlab programs. Here is the code:

```
%------------------------------------Iteration------------------------------------%
%---------------------------------------------------------------------------------%
%------------------------------parameter settings--------------------------------%
h = input('Enter the step h:');
N = 1/h;
e = input('Enter the error e:');

%-----------------------------inital Matrix L_2----------------------------------%
C = zeros(N-1);
for i = 1:N-1
    for j = 1:N-1
        if abs(i-j)==1
            C(i,j)=1;
        end
    end
end
I = eye(N-1);

L = zeros((N-1)*(N-1));
row = (N-1)*ones(1,N-1);
L_1 = mat2cell(L,row,row);

for i = 1:N-1
    L_1{i,i} = 4*I-C;
end

for i = 1:N-1
    for j = 1:N-1
        if abs(i-j)==1
            L_1{i,j}=-I;
        end
    end
end

L_2 = cell2mat(L_1);

%-------------------------------initial x,y,f------------------------------------%
x = zeros(1,N-1);
y = zeros(1,N-1);

for i = 1:N-1
    x(i)=i*h;
```

```
    y(i)=i*h;
end

f = zeros((N-1)^2,1);
for j = 1:N-1
    for i = 1:N-1
        f((j-1)*(N-1)+i) = h^2*2*pi^2*sin(pi*x(i))*sin(pi*y(j));
    end
end

%--------------------------------initial vetor-----------------------------------%
u_0 = zeros((N-1)^2,1);  %initial vector%
u_e = zeros((N-1)^2,1);  %exact solution vector%
for j = 1:(N-1)
    for i = 1:(N-1)
        u_e((j-1)*(N-1)+i) = sin(pi*x(i))*sin(pi*y(j));
    end
end

%-------------------------------matrix D,L,U-----------------------------------%
D = zeros((N-1)^2,(N-1)^2);
for i = 1: (N-1)^2
    D(i,i) = L_2(i,i);
end

L = zeros((N-1)^2,(N-1)^2);
for i = 1:(N-1)^2
    for j = 1:(N-1)^2
        if j<i
            L(i,j) = -L_2(i,j);
        end
    end
end

U = zeros((N-1)^2,(N-1)^2);
for i = 1:(N-1)^2
    for j = 1:(N-1)^2
        if j>i
            U(i,j) = -L_2(i,j);
        end
    end
end

%------------------------------Jacobi iteration --------------------------------%
J = pinv(D)*(L+U);
f_j = pinv(D)*f;
u_1 = u_0;
u_2 = f_j;
n = 1;   %  iteration number  %
while get_norm(u_2,u_1)>=e
     u_1 = J*u_2 + f_j;
     t = u_2;
     u_2 = u_1;
     u_1 = t;
     n = n+1;
end

rhoJ = max(abs(eig(J)));
n_J = n;
nm = get_norm(u_2,u_e);    %||u^(k+1)-u*||%
fprintf('the Jacobi iteration number is %d',n_J);
fprintf('the norm of the error is %f',nm);
fprintf('the radius of the convergence is %f',rhoJ);
```

```

%------------------------Gauss-Seidel iteration------------------------%
G = pinv(D-L)*U;
f_g = pinv(D-L)*f;
u_1 = u_0;
u_2 = f_g;
n = 1;   % iteration number %
while get_norm(u_2,u_1)>=e
      u_1 = G*u_2 + f_g;
      t = u_2;
      u_2 = u_1;
      u_1 = t;
      n = n+1;
end

rhoG = max(abs(eig(G)));
n_G = n;
nm = get_norm(u_2,u_e);    %||u^(k+1)-u*||%
fprintf('the Gauss-Seidel iteration number is %d',n_G);
fprintf('the norm of the error is %f',nm);
fprintf('the radius of the convergence is %f',rhoG);

%--------------------------------SOR--------------------------------%
omega = input('Enter the weight \omega:');
L_omega = pinv(D-omega*L)*((1-omega)*D+omega*U);
f_sor = omega*pinv((D-omega*L))*f;
u_1 = u_0;
u_2 = f_sor;
n = 1;   % iteration number %
while get_norm(u_2,u_1)>=e
      u_1 = L_omega*u_2 + f_sor;
      t = u_2;
      u_2 = u_1;
      u_1 = t;
      n = n+1;
end

rhoS = max(abs(eig(L_omega)));
n_SOR = n;
nm = get_norm(u_2,u_e);    %||u^(k+1)-u*||%
fprintf('the SOR iteration number is %d',n_SOR);
fprintf('the norm of the error is %f',nm);
fprintf('the radius of the convergence is %f',rhoS);
%-----------------------------function-----------------------------%
function norm = get_norm(a,b)
     norm = max(abs(a-b));
end
```