第一次作业 and 边界条件处理 and 迭代求解器

作者: ZhangYu HIT

一、第二题

1.1 原始方程和解析解

考虑如下 Poisson 方程和边界条件

$$\frac{d^2\phi}{dx^2} = 2x - 1 \tag{1}$$

$$\phi|_{x=0} = 0, \qquad \phi|_{x=1} = 1 \tag{2}$$

有解析解

$$\phi = \frac{1}{3}x^3 - \frac{1}{2}x^2 + \frac{7}{6}x \tag{3}$$

1.2 网格和数值离散

这是一个一位椭圆形方程，考虑一般性，离散格式采用非均匀网格，如图1。

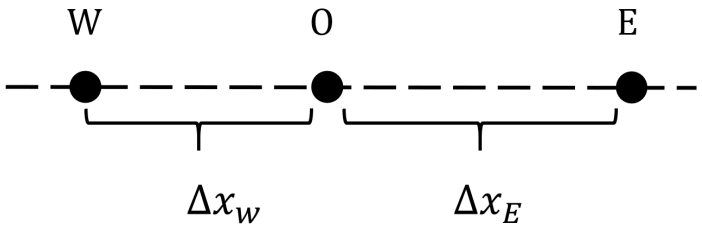


图 **1**　网格示意图

对 O 点左右两点 W 和 E 做泰勒展开，可得

$$\begin{aligned}\phi_W = \phi_O &+ \left(\frac{d\phi}{dx}\right)\Big|_{x=O}\frac{(-\Delta x_W)}{1!} + \left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O}\frac{(-\Delta x_W)^2}{2!} \\ &+ \left(\frac{d^3\phi}{dx^3}\right)\Big|_{x=O}\frac{(-\Delta x_W)^3}{3!} + \left(\frac{d^4\phi}{dx^4}\right)\Big|_{x=O}\frac{(-\Delta x_W)^4}{4!} + O(\Delta x^5)\end{aligned} \tag{4a}$$

$$\begin{aligned}\phi_E = \phi_O &+ \left(\frac{d\phi}{dx}\right)\Big|_{x=O}\frac{(\Delta x_E)}{1!} + \left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O}\frac{(\Delta x_E)^2}{2!} \\ &+ \left(\frac{d^3\phi}{dx^3}\right)\Big|_{x=O}\frac{(\Delta x_E)^3}{3!} + \left(\frac{d^4\phi}{dx^4}\right)\Big|_{x=O}\frac{(\Delta x_E)^4}{4!} + O(\Delta x^5)\end{aligned} \tag{4b}$$

为了得到二阶导的离散格式，需要将一阶导消去，采用 $\Delta x_E \times$ 式 (4a) $+\Delta x_W \times$ 式 (4b)，可得

$$\Delta x_E \phi_W + \Delta x_W \phi_E = (\Delta x_E + \Delta x_W)\phi_O + \left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O} \frac{\Delta x_E \Delta x_W (\Delta x_E + \Delta x_W)}{2} + O(\Delta x^4) \tag{5}$$

其中，

$$\begin{aligned} O(\Delta x^4) = & \left(\frac{d^3\phi}{dx^3}\right)\Big|_{x=O} \frac{\Delta x_E \Delta x_W (\Delta x_E{}^2 - \Delta x_W{}^2)}{6} \\ & + \left(\frac{d^4\phi}{dx^4}\right)\Big|_{x=O} \frac{\Delta x_E \Delta x_W (\Delta x_E{}^3 + \Delta x_W{}^3)}{24} + O(\Delta x^5)(\Delta x_E + \Delta x_W) \\ = & \left(\frac{d^3\phi}{dx^3}\right)\Big|_{x=O} \frac{\Delta x_E \Delta x_W (\Delta x_E - \Delta x_W)(\Delta x_E + \Delta x_W)}{6} \\ & + \left(\frac{d^4\phi}{dx^4}\right)\Big|_{x=O} \frac{\Delta x_E \Delta x_W (\Delta x_E + \Delta x_W)(\Delta x_E{}^2 - \Delta x_E \Delta x_W + \Delta x_W{}^2)}{24} \\ & + O(\Delta x^5)(\Delta x_E + \Delta x_W) \end{aligned} \tag{6}$$

根据式 (5) 和式 (6)，通过简单的代数变换，把 $\left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O}$ 用其他量表示，可以得到

$$\begin{aligned} \left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O} = & \frac{2\phi_E}{\Delta x_E(\Delta x_E + \Delta x_W)} - \frac{2\phi_O}{\Delta x_E \Delta x_W} + \frac{2\phi_W}{\Delta x_W(\Delta x_E + \Delta x_W)} \\ & - \left(\frac{d^3\phi}{dx^3}\right)\Big|_{x=O} \frac{\Delta x_E - \Delta x_W}{3} \\ & - \left(\frac{d^4\phi}{dx^4}\right)\Big|_{x=O} \frac{\Delta x_E{}^2 - \Delta x_E \Delta x_W + \Delta x_W{}^2}{12} + O(\Delta x^3) \end{aligned} \tag{7}$$

至此，可得推论

(1) 如果 $\Delta x_W = \Delta x_E = \Delta x$，即均匀网格，则一阶误差消失，该离散变成二阶精度

$$\left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O} = \frac{\phi_E}{\Delta x^2} - \frac{2\phi_O}{\Delta x^2} + \frac{\phi_W}{\Delta x^2} \underbrace{- \left(\frac{d^4\phi}{dx^2}\right)\Big|_{x=O} \frac{\Delta x^2}{12} + O(\Delta x^3)}_{o(\Delta x^2)} \tag{8}$$

(2) 如果 $\Delta x_{_W} \neq \Delta x_{_E}$，即非均匀网格，则二阶误差不能消除，该离散变成一阶精度，

$$\left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O} = \frac{2\phi_{_E}}{\Delta x_{_E}(\Delta x_{_E}+\Delta x_{_W})} - \frac{2\phi_{_O}}{\Delta x_{_E}\Delta x_{_W}} + \frac{2\phi_{_W}}{\Delta x_{_W}(\Delta x_{_E}+\Delta x_{_W})} \underbrace{-\left(\frac{d^3\phi}{dx^3}\right)\Big|_{x=O}\frac{\Delta x_{_E}-\Delta x_{_W}}{3}+O(\Delta x^2)}_{o(\Delta x^1)} \tag{9}$$

1.3 求解

对于该题，网格处于非均匀情况，网格间距以等比数列形式（公比为 S）增加，此时，只需令 $\Delta x_{_E} = S\Delta x_{_W}$，将其带入式 (9)，可得

$$\left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O} = \frac{2\phi_{_E}}{S(1+S)\Delta {x_{_W}}^2} - \frac{2\phi_{_O}}{S\Delta {x_{_W}}^2} + \frac{2\phi_{_W}}{(1+S)\Delta {x_{_W}}^2} = R_S \tag{10}$$

为了简化系数矩阵，将式 (10) 转化为如下形式

$$\phi_{_E} - (1+S)\phi_{_O} + S\phi_{_W} = R_S(1+S)S\frac{\Delta {x_{_W}}^2}{2} \tag{11}$$

其中，R_S 代表原 Poisson 方程源项，注意这里忽略了二阶误差。注意原始方程四阶导为零，所以如果采用均匀网格，误差应为零。

由于原方程给定第一类边界条件，容易构造如下有限差分系数矩阵，并写成线性方程组形式：

$$\begin{bmatrix} 1 & \dots & & & & & 0 \\ S & -(1+S) & 1 & & & & \\ & S & -(1+S) & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & S & -(1+S) & 1 & \\ 0 & & & & \dots & & 1 \end{bmatrix} \begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \end{bmatrix} = \begin{bmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{N-2} \\ R_{N-1} \end{bmatrix} \tag{12}$$

编程实现完全按照式 (12)，可采用矩阵求解器解得 $[\Phi]$，或者根据式 (11) 构造显式或半隐格式迭代求解。

$$\text{显式迭代:} \qquad \phi_{_O}^{n+1} = \frac{\phi_{_E}^{n} + S\phi_{_W}^{n}}{(1+S)} - R_S\frac{\Delta {x_{_W}}^2}{2}S \tag{13}$$

$$\text{半隐式迭代:} \qquad \phi_{_O}^{n+1} = \frac{\phi_{_E}^{n} + S\phi_{_W}^{n+1}}{(1+S)} - R_S\frac{\Delta {x_{_W}}^2}{2}S \tag{14}$$

采用高斯消元法求解，得到数值解和相对误差

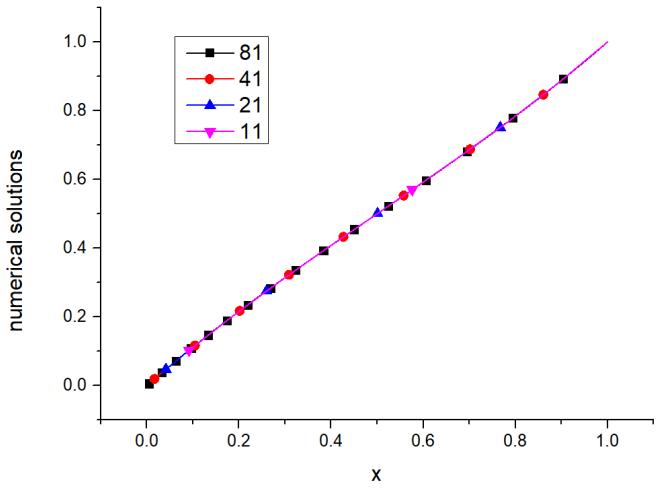


图 **2**　数值解

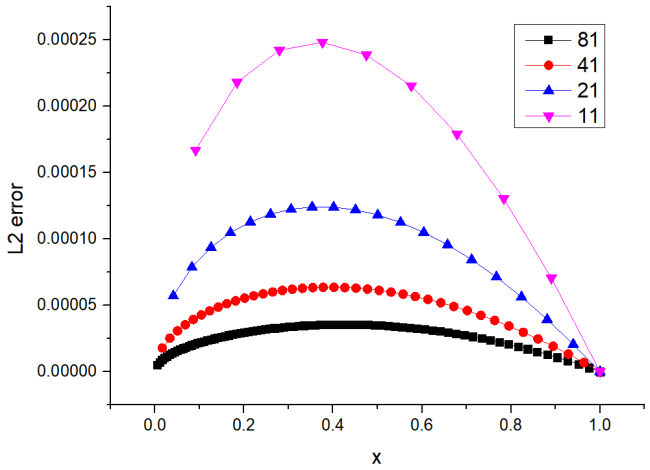


图 **3**　相对误差 **(**二范数**)**

1.4 C 语言源程序

```
  hello;
  clc;
  clear;
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int coordinate(const int x, const int y, const int N){
```

```
  return(x + y * N);
}

void Gaussian_Elimination(double* A, double* B, double* res, const int N){
  for(int j = 0; j < N - 1; ++j){
    for(int i = j + 1; i < N; ++i){
      double f_eli = A[coordinate(j, i, N)] / A[coordinate(j, j, N)];

      B[i] = B[i] - f_eli * B[j];
      for(int k = 0; k < N; ++k){
        A[coordinate(k, i, N)] = A[coordinate(k, i, N)] - f_eli * A[coordinate(k, j, N)];
      }
    }
  }

  for(int j = N - 1; j > -1; --j){
    res[j] = B[j];
    for(int i = j + 1; i < N; ++i){
      if(i != j){
        res[j] -= A[coordinate(i, j, N)] * res[i];
      }
    }
    res[j] = res[j] / A[coordinate(j, j, N)];
  }
}

int main(){
  const double L = 1.0;
  const int N = 21;

  // ******* Exponential mesh**************************
  const double S = 1.02;
  const double x0 = L * (1 - S) / (1 - pow(S, N - 1));

  //// ******* Uniform mesh**************************
  //const double S = 1.00;
  //const double x0 = L / (N - 1);

  printf("The first delta_x is : x0 = %f\n", x0);

  double* Coefficient_Matrix = (double*) malloc(N * N * sizeof(double));

  double* Right_Term = (double*) malloc((N ) * sizeof(double));
  double* X_Posi   = (double*) malloc((N ) * sizeof(double));
  double* Delta_x  = (double*) malloc((N - 1) * sizeof(double));

  double* phi_num  = (double*) malloc((N ) * sizeof(double));
```

```
double* phi_ana = (double*) malloc((N ) * sizeof(double));

//**********We need get delta_x, position of each node,***********
Delta_x[0] = x0;
phi_ana[0] = 0;
X_Posi[0] = 0;
Right_Term[0] = 2 * X_Posi[0] - 1;

for(int i = 1; i < N; ++i){

   if(i < N - 1){
      Delta_x[i] = S * Delta_x[i - 1];

   }

   X_Posi[i] = Delta_x[i - 1] + X_Posi[i - 1];

   phi_ana[i] = 1.0 / 3.0 * X_Posi[i] * X_Posi[i] * X_Posi[i]
   - 1.0 / 2.0 * X_Posi[i] * X_Posi[i]
   + 7.0 / 6.0 * X_Posi[i];

   Right_Term[i] = 2 * X_Posi[i] - 1;
}


//*********** Constract Coeffieient Matrix of Dirichlet Boundary condiction
for(int j = 1; j < N - 1; ++j){
   for(int i = 0; i < N ; ++i){
      const int coordinate_ = coordinate(i, j, N);
      Coefficient_Matrix[coordinate_] = 0;
      if(i == j){
         Coefficient_Matrix[coordinate_] = -(1.0 + S);
      }
      if(i + 1 == j){
         Coefficient_Matrix[coordinate_] = S;
      }
      if(i - 1 == j){
         Coefficient_Matrix[coordinate_] = 1.0;
      }
   }
}
for(int i = 0; i < N; ++i){
   const int coordinate_T = coordinate(i, 0 , N);
   const int coordinate_B = coordinate(i, N - 1, N);
   Coefficient_Matrix[coordinate_T] = 0;
   Coefficient_Matrix[coordinate_B] = 0;
}
```

```
Coefficient_Matrix[coordinate(0 , 0 , N)] = 1;
Coefficient_Matrix[coordinate(N - 1, N - 1, N)] = 1;
//***End of constracting Coeffieient Matrix of Dirichlet Boundary condiction

//***** Reconstract Right hand term of linear algebraic equations
for(int i = 1; i < N - 1; ++i){
   Right_Term[i] *= Delta_x[i - 1] * Delta_x[i - 1] * S * (1 + S) / 2.0;
}
Right_Term[0  ] = 0;
Right_Term[N - 1] = 1;
//**End of reconstracting Right hand term of linear algebraic equations

Gaussian_Elimination(Coefficient_Matrix, Right_Term, phi_num, N);



//*** Output results*********************
// std::ostringstream name;
// name << N << "_nodes_results" << ".dat";
// std::ofstream fout(name.str().c_str( ) );

// double* abs_err = (double*) malloc((N) * sizeof(double));
// double* rela_err = (double*) malloc((N) * sizeof(double));

// fout << "i Position Right_Term Analysitic Numerical Abs_error Relative_error" << endl;
// for(int i = 0; i < N; ++i){
   //    abs_err[i] = fabs(phi_ana[i] - phi_num[i]);
   //    rela_err[i] = sqrt((phi_ana[i] - phi_num[i]) * (phi_ana[i] - phi_num[i]) /
       phi_ana[i]);
   //    //fout << std::setw(4) << i << " "
   //    fout << i << " "
   //         << std::fixed << std::setprecision(10)
   //         << X_Posi[i] << " "
   //         << Right_Term[i] << " "
   //         << phi_ana[i] << " "
   //         << phi_num[i] << " "
   //         << abs_err[i] << " "
   //         << rela_err[i] << " "
   //         << endl;
   // }
// fout.close();

// free(abs_err);
// free(rela_err);
free(Coefficient_Matrix);
free(Right_Term );
free(X_Posi   );
free(Delta_x  );
```

```
  free(phi_num );
  free(phi_ana );
}
```

二、第三题

2.1 网格离散

网格离散示意图如4,

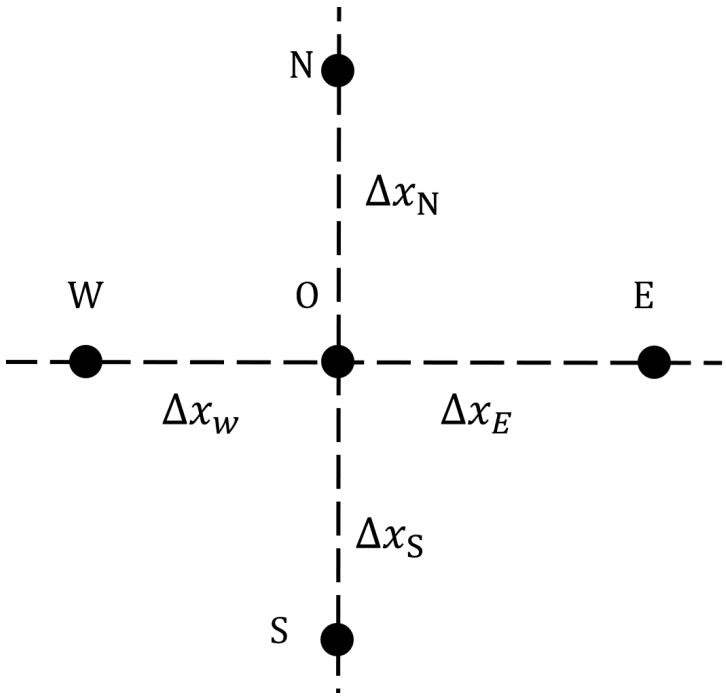


图 **4**　网格示意图

根据式 (7)，忽略误差项，加入 y 方向节点，二维情况下二阶导离散变成

$$\left(\frac{d^2\phi}{dx^2}\right)\Big|_{x=O} = \frac{2\phi_E}{\Delta x_E(\Delta x_E+\Delta x_W)} - \frac{2\phi_O}{\Delta x_E\Delta x_W} + \frac{2\phi_W}{\Delta x_W(\Delta x_E+\Delta x_W)} + \frac{2\phi_N}{\Delta x_N(\Delta x_N+\Delta x_S)} - \frac{2\phi_O}{\Delta x_N\Delta x_S} + \frac{2\phi_S}{\Delta x_S(\Delta x_N+\Delta x_S)} \tag{15}$$

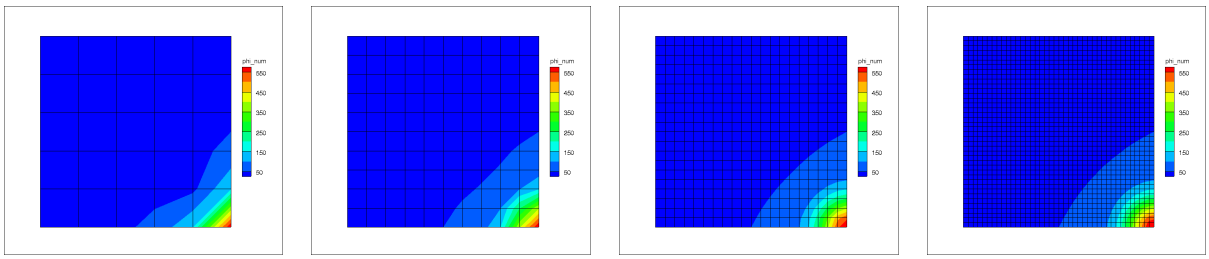
此时，原二维 Poisson 方程内节点离散格式为

$$\phi_E\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)} + \phi_W\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)} + \phi_N\frac{2}{\Delta x_N(\Delta x_N+\Delta x_S)} + \phi_S\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)} - \phi_O\left(\frac{2}{\Delta x_E\Delta x_W}+\frac{2}{\Delta x_N\Delta x_S}\right) = S_R \tag{16}$$

式 (16) 适用于正交网格的均匀或非均匀情况，对于均匀网格，应为二阶精度，对于非均匀网格，类似于式 (9)，离散三阶导无法消除，变成一阶精度。

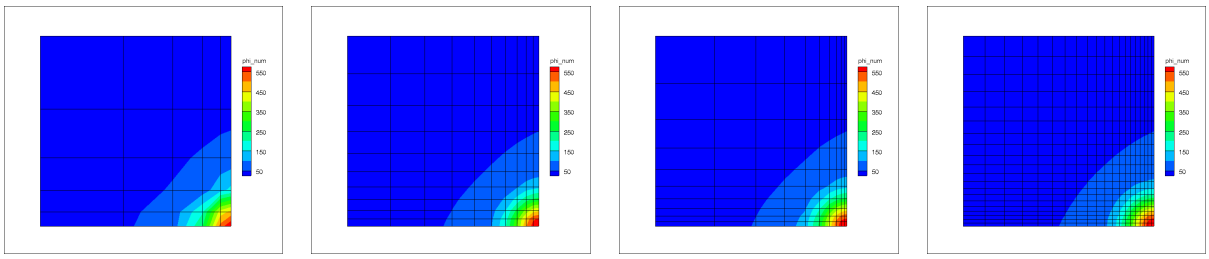
2.2 计算结果

图5和图6给出了离散节点为 6，11，21 和 41 的均匀网格数值解和离散节点为 6，11，21 三种情况的数值解，其中，非均匀网格 x、y 方向网格离散尺度关系与上一题相同，公比为 S_x 分别为 S_y，不同节点数目的结果公比可能不同。



(a) 6 节点　(b) 11 节点　(c) 21 节点　(d) 41 节点

图 **5**　均匀网格数值解



(a) 6 节点　(b) 11 节点 (1)　(c) 11 节点 (2)　(d) 21 节点

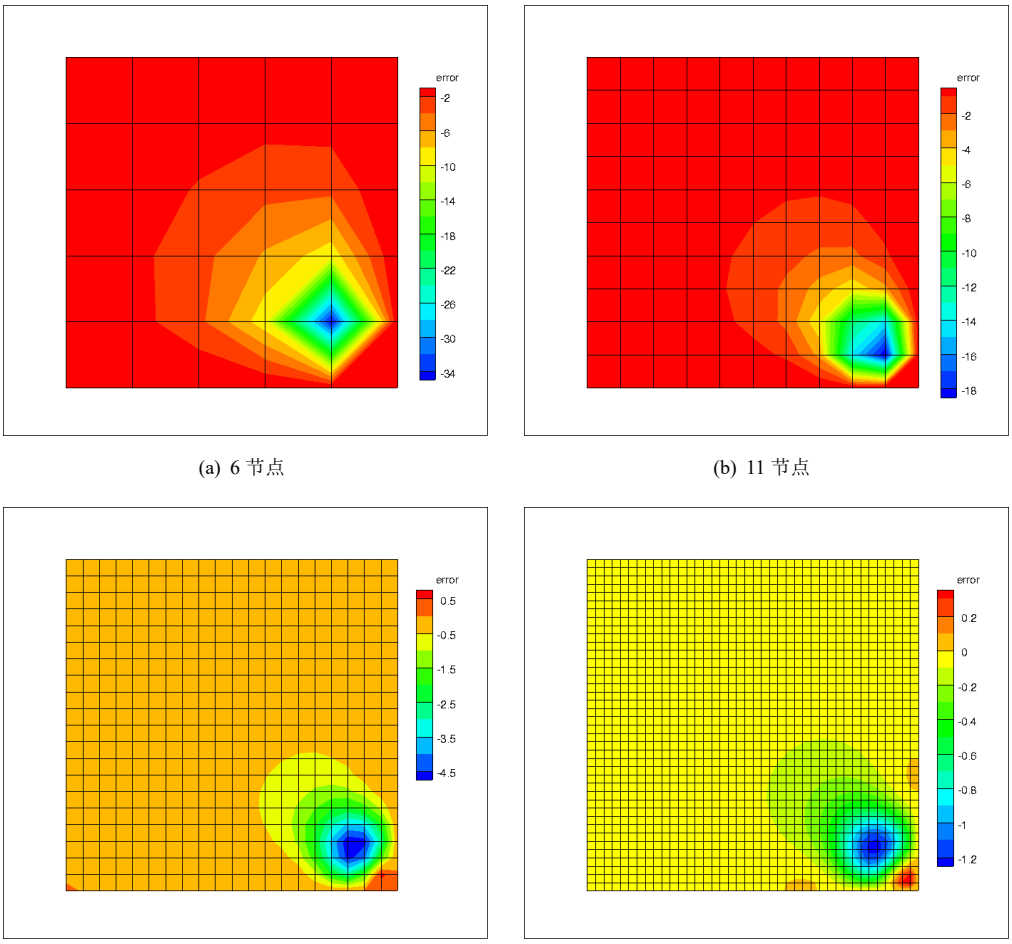
图 **6**　非均匀网格数值解

图7和图8给出了精确解和数值解的误差云图，由于计算相对误差时出现了奇点，在此只给出精确解与数值解的绝对误差，分别计算了不同网格节点数目的离散误差，并且对比了均匀网格与非均匀网格，虽然非均匀网格只有一阶精度，但是由于此题目在一个角点区域梯度很大，网格数目较少时，非均匀网格离散可以得出较为准确的结果（由于绘图结果问题，此处应关注 colorbar）。

$$L_{error} = \phi_{num} = \phi_{ana} \tag{17}$$

2.3 C++ 源程序

```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
```



(a) 6 节点　　(b) 11 节点

(c) 21 节点　　(d) 41 节点

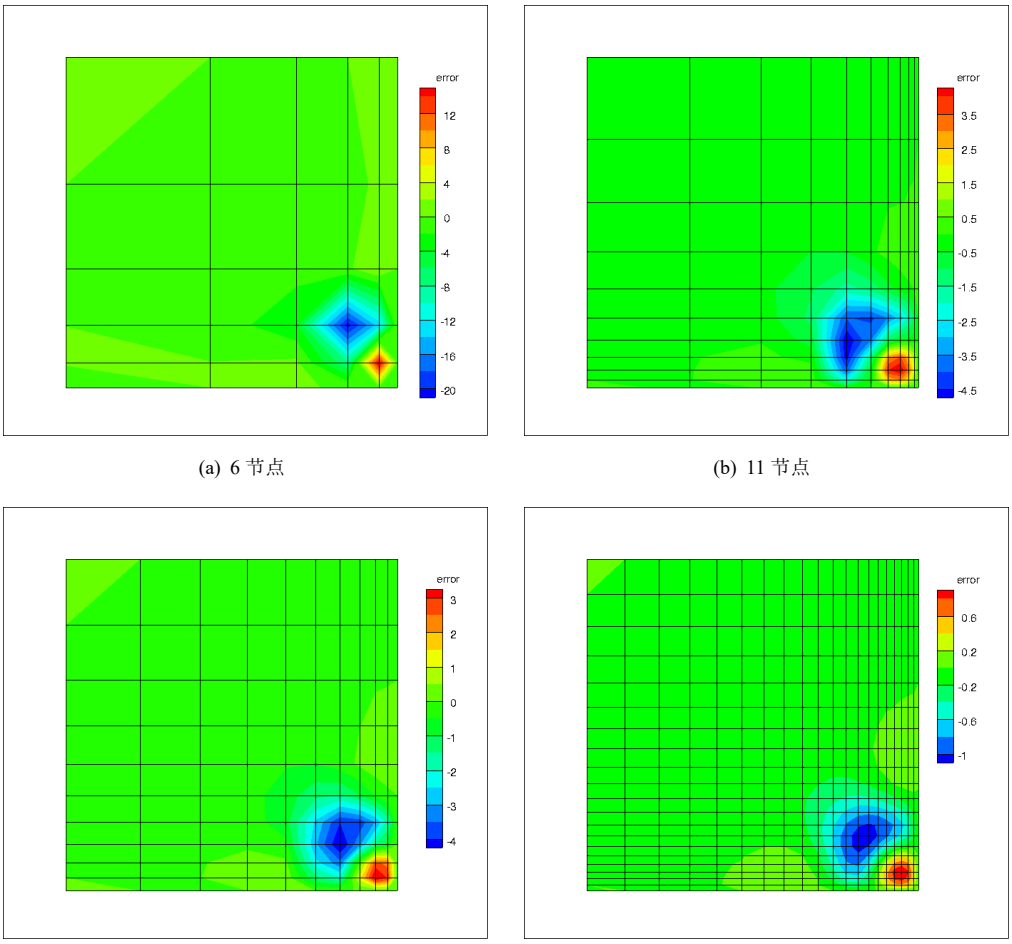
图 7　均匀网格离散误差

```
#include <string>
#include <Eigen/Dense>


using std::cout;
using std::endl;
using std::vector;
using namespace Eigen;

int coordinate(const int x, const int y, const int NX){
   return(x + y * NX);
}

template<typename T>
T p2(const T x){
   return(x * x);
}
```



(a) 6 节点　　(b) 11 节点

(c) 11 节点　　(d) 21 节点

图 **8**　非均匀网格（指数关系）离散误差

```
//void Right_Term(VectorXd B, X_poi, Y_poi){
  //}

int main(){

  const int N = 10 + 1;
  const double delta = 1.0 / (N - 1);
  //// ******* Uniform mesh**************************
  //const double SX = 1.00;
  //const double SY = 1.00;
  //const double x0 = 1.0 / (N - 1);
  //const double y0 = 1.0 / (N - 1);

  // ******* Non-uniform mesh**************************
  const double SX = 0.7;
  const double SY = 1.3;
  const double x0 = 1.0 * (1 - SX) / (1 - pow(SX, N - 1));
  const double y0 = 1.0 * (1 - SY) / (1 - pow(SY, N - 1));
```

```
// ----------------- spacing of nodes
vector<double> X_delta(N - 1, 0);
vector<double> Y_delta(N - 1, 0);
X_delta[0] = x0;
Y_delta[0] = y0;
for(int i = 1; i < N - 1; ++i){
   X_delta[i] = X_delta[i - 1] * SX;
   Y_delta[i] = Y_delta[i - 1] * SY;
   cout << X_delta[i] << " " << Y_delta[i] << endl;
}

// ----------------- position of nodes
vector<double> X_poi(N, 0);
vector<double> Y_poi(N, 0);
for(int i = 1; i < N; ++i){
   X_poi[i] = X_poi[i - 1] + X_delta[i - 1];
   Y_poi[i] = Y_poi[i - 1] + Y_delta[i - 1];
   cout << X_poi[i] << " " << Y_poi[i] << endl;
}


// ----------------- FD Coeffieient Matrix
MatrixXd A = MatrixXd::Constant(N * N, N * N, 0);

// -----------------[A][X] = [B]
VectorXd B = VectorXd::Constant(N * N, 0);
VectorXd X = VectorXd::Constant(N * N, 0);

VectorXd phi_ana = VectorXd::Constant(N * N, 0);

for(int j = 0; j < N; ++j){
   for(int i = 0; i < N; ++i){
      const int index = coordinate(i, j, N);
      double X_ = X_poi[i];
      double Y_ = Y_poi[j];
      phi_ana(index) = 500 * exp(-50 * (p2(1.0 - X_) + p2(Y_))) + 100 * X_ * (1 - Y_);
   }
}

//Right_Term(B, X_poi, Y_poi); ************************??????????????????
for(int j = 0; j < N; ++j){
   for(int i = 0; i < N; ++i){
      const int index = coordinate(i, j, N);
      // ---------------- Left BC phi(0, y)
      if(i == 0){
         double X_ = 0 * X_poi[i];
```

```
        double Y_ = Y_poi[j];
        B(index) = 500 * exp(-50 * (p2(1.0 - X_) + p2(Y_))) + 100 * X_ * (1 - Y_);
        //B(index) = 500 * exp(-50 * (1.0 + p2(Y_poi[j])));
      }
      // --------------- Right BC phi(1, y)
      else if(i == N - 1){
        double X_ = 1;//X_poi[i];
        double Y_ = Y_poi[j];
        B(index) = 500 * exp(-50 * (p2(1.0 - X_) + p2(Y_))) + 100 * X_ * (1 - Y_);
        //B(index) = 100 * (1 - Y_poi[j]) + 500 * exp(-50 * p2(Y_poi[j]));
      }
      // --------------- Bottom BC phi(x, 0)
      else if(j == 0){
        double X_ = X_poi[i];
        double Y_ = 0 * Y_poi[j];
        B(index) = 500 * exp(-50 * (p2(1.0 - X_) + p2(Y_))) + 100 * X_ * (1 - Y_);
        //B(index) = 100 * X_poi[i] + 500 * exp(-50 * p2(1.0 - X_poi[i]));
      }
      // --------------- Top BC phi(x, 1)
      else if(j == N - 1){
        double X_ = X_poi[i];
        double Y_ = 1;//Y_poi[j];
        B(index) = 500 * exp(-50 * (p2(1.0 - X_) + p2(Y_))) + 100 * X_ * (1 - Y_);
        //B(index) = 500 * exp(-50 * p2(1 - X_poi[i]) + 1);
      }
      else{
        double X_ = X_poi[i];
        double Y_ = Y_poi[j];
        B(index) = (5000000*p2(Y_) + 5000000*p2(X_ - 1) - 100000)*exp(-50*p2(Y_) - 50*p2(1 -
            X_));
        //B(index) = 50000 * exp(-50 * (p2(1.0 - X_poi[i]) + p2(Y_poi[j])))
        //       * (100 * (p2(1.0 - X_poi[i]) + p2(Y_poi[j])) - 2);
      }
    }
  }

  // Driichlet_BC_FD_Matrix(A, X_delta, Y_delta) *************?????????????????
  for(int i = 0; i < p2(N); ++i){
    const int index = coordinate(i, i, p2(N));
    // --------------- Left BC phi(0, y)
    if(i / ((int)N) == 0){
      A(index) = 1.0;
    }
    // --------------- Right BC phi(1, y)
    else if(i / ((int)N) == N - 1){
      A(index) = 1.0;
    }
```

```
    // --------------- Bottom BC phi(x, 0)
    else if((i % (int)N) == 0){
    A(index) = 1.0;
    }
    // --------------- Top BC phi(x, 1)
    else if((i % (int)N) == N - 1){
       A(index) = 1.0;
    }
    else{
       int ij = i / ((int)N);
       int ii = i % ((int)N);
       A(index) = -(2.0 / (X_delta[ii - 1] * X_delta[ii])
       + 2.0 / (Y_delta[ij - 1] * Y_delta[ij]));

       A(index - 1 * p2(N)) = 2.0 / (X_delta[ii - 1] * (X_delta[ii - 1] + X_delta[ii]));
       A(index + 1 * p2(N)) = 2.0 / (X_delta[ii ] * (X_delta[ii - 1] + X_delta[ii]));

       A(index + N * p2(N)) = 2.0 / (Y_delta[ij ] * (Y_delta[ij - 1] + Y_delta[ij]));
       A(index - N * p2(N)) = 2.0 / (Y_delta[ij - 1] * (Y_delta[ij - 1] + Y_delta[ij]));
    }
  }

  X = A.colPivHouseholderQr().solve(B);

  //****************************************fun_tecplot()???????????;
  std::ostringstream name;
  //name << "Phi_" << N << "_.dat";
  name << "No_Phi_" << N << "_.dat";
  std::ofstream out(name.str().c_str( ) );
  out << "Title= \"Poisson_Multi_Blocks\"\n"
  << "VARIABLES = \"X\", \"Y\", \"phi_num\", \"phi_ana\", \"error\", \"rela_error\" \n";
  out << "ZONE T= \"BOX\",I=" << N <<",J="<< N << ", F = POINT" << endl;
  for(int j = 0; j < N; ++j){
    for(int i = 0; i < N; ++i){
       const int index = coordinate(i, j, N);
       out << X_poi[i] << " " << Y_poi[j] << " "
       << X(index) << " "
       << phi_ana(index) << " "
       << X(index) - phi_ana(index) << " "
       << sqrt(p2(X(index) - phi_ana(index)) / p2(1e-30 + phi_ana(index))) << " "
       << endl;
    }
  }
  out.close();
}
```

三、边界格式处理

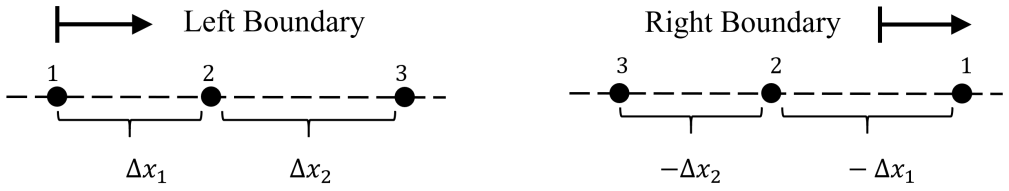
由于第三类边界条件可以看作第一类和第二类的线性组合，所以直接给出第三类边界条件的推导。

对于 Robin 边界条件，或者说第三类边界条件

$$\gamma = \alpha\phi_1 + \beta\frac{\mathrm{d}\phi}{\mathrm{d}x}\bigg|_{x=1} \tag{18}$$

我们有

$$\frac{\mathrm{d}\phi}{\mathrm{d}x}\bigg|_{x=1} = \frac{\gamma - \alpha\phi_1}{\beta} \tag{19}$$



(a) 左边界三点离散　　(b) 右边界三点离散

图 **9**

我们直接给出二阶精度的边界离散格式，二阶精度至少需要边界处三个节点，如图9。

如图9(a)，给出左边界邻近两个节点的泰勒展开

$$\begin{aligned}\phi_3 = \phi_1 + \frac{\mathrm{d}\phi}{\mathrm{d}x}(\Delta x_1 + \Delta x_2) + \frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\frac{(\Delta x_1 + \Delta x_2)^2}{2} \\ + \frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\frac{(\Delta x_1 + \Delta x_2)^3}{6} + \frac{\mathrm{d}^3\phi}{\mathrm{d}x^4}\frac{(\Delta x_1 + \Delta x_2)^4}{24} + o(\Delta x^5)\end{aligned} \tag{20}$$

$$\phi_2 = \phi_1 + \frac{\mathrm{d}\phi}{\mathrm{d}x}\Delta x_1 + \frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\frac{\Delta x_1^2}{2} + \frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\frac{\Delta x_1^3}{6} + \frac{\mathrm{d}^3\phi}{\mathrm{d}x^4}\frac{\Delta x_1^4}{24} + o(\Delta x^5) \tag{21}$$

我们先推导边界节点一阶导的表达式，为了消除三阶导从而得到二阶精度，我们需要对实施如下代数操作

$$\phi_2(\Delta x_1 + \Delta x_2)^2 - \phi_3\Delta x_1^2 \tag{22}$$

此处省略推导过程，直接给出一阶导的表达式

$$\left.\frac{\mathrm{d}\phi}{\mathrm{d}x}\right|_{x=1}=\frac{-\Delta x_1^2\phi_3+\phi_1\left(-2\Delta x_1\Delta x_2-\Delta x_2^2\right)+\phi_2\left(\Delta x_1^2+2\Delta x_1\Delta x_2+\Delta x_2^2\right)}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}+\underbrace{\frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\left(\frac{\Delta x_1^2}{6}+\frac{\Delta x_1\Delta x_2}{6}\right)+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\left(\frac{\Delta x_1^3}{12}+\frac{\Delta x_1^2\Delta x_2}{8}+\frac{\Delta x_1\Delta x_2^2}{24}\right)+o(\Delta x^4)}_{o(\Delta x^2)} \tag{23}$$

如果是均匀网格，则有 $\Delta x_2=\Delta x_1$

$$\left.\frac{\mathrm{d}\phi}{\mathrm{d}x}\right|_{x=1}=\frac{-3\phi_1+4\phi_2-\phi_3}{2\Delta x_1}+\underbrace{\frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\frac{\Delta x_1^2}{3}+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\frac{\Delta x_1^3}{4}+o(\Delta x^4)}_{o(\Delta x^2)} \tag{24}$$

对于 Robin 边界条件，只需要把一阶导的表达式代入式 (23) 或式 (24) 代入 Robin 边界条件式 (18)，整理成为 ϕ_1、ϕ_2、ϕ_3 的代数表达式，并代入差分稀疏矩阵内即可得到 Robin 边界条件的计算式。下边右边界的推导同理。

如果是右边界，那么离散方向刚好与左边界相反，类似于左边界的推导，在此给出简要推导过程。

$$\begin{aligned}\phi_3=\phi_1&-\frac{\mathrm{d}\phi}{\mathrm{d}x}\left(\Delta x_1+\Delta x_2\right)+\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\frac{\left(\Delta x_1+\Delta x_2\right)^2}{2}\\&-\frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\frac{\left(\Delta x_1+\Delta x_2\right)^3}{6}+\frac{\mathrm{d}^3\phi}{\mathrm{d}x^4}\frac{\left(\Delta x_1+\Delta x_2\right)^4}{24}+o(\Delta x^5)\end{aligned} \tag{25}$$

$$\phi_2=\phi_1-\frac{\mathrm{d}\phi}{\mathrm{d}x}\Delta x_1+\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\frac{\Delta x_1^2}{2}-\frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\frac{\Delta x_1^3}{6}+\frac{\mathrm{d}^3\phi}{\mathrm{d}x^4}\frac{\Delta x_1^4}{24}+o(\Delta x^5) \tag{26}$$

同样，利用式 (22)，我们消去二阶导，从而得到二阶精度的右边界节点一阶导表达式

$$\left.\frac{\mathrm{d}\phi}{\mathrm{d}x}\right|_{x=1}=\frac{\Delta x_1^2\phi_3+\phi_1\left(2\Delta x_1\Delta x_2+\Delta x_2^2\right)+\phi_2\left(-\Delta x_1^2-2\Delta x_1\Delta x_2-\Delta x_2^2\right)}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}+\underbrace{\frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\left(\frac{\Delta x_1^2}{6}+\frac{\Delta x_1\Delta x_2}{6}\right)+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\left(-\frac{\Delta x_1^3}{12}-\frac{\Delta x_1^2\Delta x_2}{8}-\frac{\Delta x_1\Delta x_2^2}{24}\right)+o(\Delta x^4)}_{o(\Delta x^2)} \tag{27}$$

对于均匀网格，同样有 $\Delta x_2=\Delta x_1$，

$$\left.\frac{\mathrm{d}\phi}{\mathrm{d}x}\right|_{x=1}=\frac{3\phi_1-4\phi_2+\phi_3}{2\Delta x_1}+\underbrace{\frac{\mathrm{d}^3\phi}{\mathrm{d}x^3}\frac{\Delta x_1^2}{3}-\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\frac{\Delta x_1^3}{4}}_{o(\Delta x^2)} \tag{28}$$

给出左边界 Robin 边界条件通用表达式

$$\alpha\phi_1+\beta\frac{-\Delta x_1^2\phi_3+\phi_1\left(-2\Delta x_1\Delta x_2-\Delta x_2^2\right)+\phi_2\left(\Delta x_1^2+2\Delta x_1\Delta x_2+\Delta x_2^2\right)}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}=\gamma \tag{29}$$

$$\left(\alpha+\beta\frac{\left(-2\Delta x_1\Delta x_2-\Delta x_2^2\right)}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}\right)\phi_1+\beta\frac{-\Delta x_1^2\phi_3+\left(\Delta x_1^2+2\Delta x_1\Delta x_2+\Delta x_2^2\right)\phi_2}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}=\gamma$$

如果是均匀网格

$$\text{左边界均匀网格:}\quad \alpha\phi_1+\beta\frac{-3\phi_1+4\phi_2-\phi_3}{2\Delta x_1}=\gamma \tag{30}$$

$$\left(\alpha-\frac{3\beta}{2\Delta x_1}\right)\phi_1+\beta\frac{4\phi_2-\phi_3}{2\Delta x_1}=\gamma$$

给出右边界 Robin 边界条件通用表达式

$$\alpha\phi_1+\beta\frac{\Delta x_1^2\phi_3+\phi_1\left(2\Delta x_1\Delta x_2+\Delta x_2^2\right)+\phi_2\left(-\Delta x_1^2-2\Delta x_1\Delta x_2-\Delta x_2^2\right)}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}=\gamma \tag{31}$$

$$\left(\alpha+\beta\frac{\left(2\Delta x_1\Delta x_2+\Delta x_2^2\right)}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}\right)\phi_1+\beta\frac{\Delta x_1^2\phi_3+\left(-\Delta x_1^2-2\Delta x_1\Delta x_2-\Delta x_2^2\right)\phi_2}{\Delta x_1^2\Delta x_2+\Delta x_1\Delta x_2^2}=\gamma$$

如果是均匀网格

$$\text{右边界均匀网格:}\quad \alpha\phi_1+\beta\frac{3\phi_1-4\phi_2+\phi_3}{2\Delta x_1}=\gamma \tag{32}$$

$$\left(\alpha+\frac{3\beta}{2\Delta x_1}\right)\phi_1+\beta\frac{-4\phi_2+\phi_3}{2\Delta x_1}=\gamma$$

如果数值求解椭圆形方程式 (33)，我们还可以通过离散直接得到二阶导在边界处的离散。我们只需要消去三阶导，就能得到只剩四阶导和更高阶导数的误差项，从而得到二阶精度边界节点离散式。对于以下的推导，只给出 Robin 边界条件的离散格式，代入系数矩阵时需要稍作化简，将其转化成关于 ϕ_1、ϕ_2、ϕ_3 和右手源项 S 的系数方程，然后代入离散线性代数方程即可。

$$\nabla^2\phi=S \tag{33}$$

这里采用的消元操作为

$$\phi_2(\Delta x_1+\Delta x_3)^3-\phi_3\Delta x_1^3 \tag{34}$$

省略推导过程，直接给出边界的二阶精度 Robin 边界条件的离散格式，如果需要求解 Neumann 边界，只需要令 $\aleph=0$ 即可，如果是 Dirichlet 边界，直接赋值，不用求解边界点的差分方程。

对于左边界，有

$$\left.\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\right|_{x=1} = \frac{\frac{\gamma-\alpha\phi_1}{\beta}a_1 + \phi_3 a_2 + \phi_1 a_3 + \phi_2 a_4}{a_5} \underbrace{+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\left(\frac{\Delta x_1^2}{12} + \frac{\Delta x_1 \Delta x_2}{12}\right)}_{o(\Delta x^2)} \tag{35}$$

其中

$$\begin{aligned}
a_1 &= \left(-4\Delta x_1^3\Delta x_2 - 6\Delta x_1^2\Delta x_2^2 - 2\Delta x_1\Delta x_2^3\right)\\
a_2 &= \qquad -2\Delta x_1^3\\
a_3 &= \left(-6\Delta x_1^2\Delta x_2 - 6\Delta x_1\Delta x_2^2 - 2\Delta x_2^3\right)\\
a_4 &= \left(2\Delta x_1^3 + 6\Delta x_1^2\Delta x_2 + 6\Delta x_1\Delta x_2^2 + 2\Delta x_2^3\right)\\
a_5 &= \left(\Delta x_1^4\Delta x_2 + 2\Delta x_1^3\Delta x_2^2 + \Delta x_1^2\Delta x_2^3\right)
\end{aligned}$$

如果是均匀网格，可简化为

$$\left.\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\right|_{x=1} = \frac{-6\frac{\gamma-\alpha\phi_1}{\beta}\Delta x_1 - 7\phi_1 + 8\phi_2 - \phi_3}{2\Delta x_1^2} \underbrace{+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\frac{\Delta x_1^2}{6}}_{o(\Delta x^2)} \tag{36}$$

对于右边界，有

$$\left.\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\right|_{x=1} = \frac{\frac{\gamma-\alpha\phi_1}{\beta}a_1 + a_2\phi_3 + \phi_1 a_3 + \phi_2 a_4}{a_5} \underbrace{+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\left(\frac{\Delta x_1^2}{12} + \frac{\Delta x_1 \Delta x_2}{12}\right)}_{o(\Delta x^2)} \tag{37}$$

其中

$$\begin{aligned}
a_1 &= \left(4\Delta x_1^3\Delta x_2 + 6\Delta x_1^2\Delta x_2^2 + 2\Delta x_1\Delta x_2^3\right)\\
a_2 &= \qquad -2\Delta x_1^3\\
a_3 &= \left(-6\Delta x_1^2\Delta x_2 - 6\Delta x_1\Delta x_2^2 - 2\Delta x_2^3\right)\\
a_4 &= \left(2\Delta x_1^3 + 6\Delta x_1^2\Delta x_2 + 6\Delta x_1\Delta x_2^2 + 2\Delta x_2^3\right)\\
a_5 &= \left(\Delta x_1^4\Delta x_2 + 2\Delta x_1^3\Delta x_2^2 + \Delta x_1^2\Delta x_2^3\right)
\end{aligned}$$

均匀网格情况可简化为

$$\left.\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\right|_{x=1} = \frac{6\frac{\gamma-\alpha\phi_1}{\beta}\Delta x_1 - 7\phi_1 + 8\phi_2 - \phi_3}{2\Delta x_1^2} \underbrace{+\frac{\mathrm{d}^4\phi}{\mathrm{d}x^4}\frac{\Delta x_1^2}{6}}_{o(\Delta x^2)} \tag{38}$$

注意，以上推导我们虽然只给出了左右边界的格式，但是对于二维或三维情况，采用笛卡尔坐标系，上下或者前后方向，如果采用向上和向前为正方向，则下边界和后边界的离散可以直接参考左边界离散格式，上边界和前边界的离散直接参考右边界的离散格式。

四、第二次作业第一题 (第一次第三题采用迭代方法求解)

偏微分方程

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}=50000\left[100\left((1-x)^2+y^2\right)-2\right]e^{-50\left((1-x)^2+y^2\right)} \tag{39}$$

解析解

$$\phi(x,y)=500e^{-50\left((1-x)^2+y^2\right)}+100x(1-y) \tag{40}$$

这里给出另一种处理边界的系数矩阵构造方法，只适用于第一类边界条件。如果边界都已知，我们需要只求内节点的值即可，可以将边界的值吸收入边界相邻节点的源项中，求出内节点的差分矩阵。给出边界处不满足五点差分格式的点的处理方法：

$$\begin{aligned}\text{左：}\quad &\phi_{_N}\frac{2}{\Delta x_{_N}(\Delta x_{_N}+\Delta x_{_S})}+\phi_{_S}\frac{2}{\Delta x_{_S}(\Delta x_{_N}+\Delta x_{_S})}+\phi_{_E}\frac{2}{\Delta x_{_E}(\Delta x_{_E}+\Delta x_{_W})}\\ &-\phi_{_O}\left(\frac{2}{\Delta x_{_E}\Delta x_{_W}}+\frac{2}{\Delta x_{_N}\Delta x_{_S}}\right)=S_R-\phi_{_W}\frac{2}{\Delta x_{_W}(\Delta x_{_E}+\Delta x_{_W})}\end{aligned} \tag{41a}$$

$$\begin{aligned}\text{右：}&\phi_{_N}\frac{2}{\Delta x_{_N}(\Delta x_{_N}+\Delta x_{_S})}+\phi_{_S}\frac{2}{\Delta x_{_S}(\Delta x_{_N}+\Delta x_{_S})}+\phi_{_W}\frac{2}{\Delta x_{_W}(\Delta x_{_E}+\Delta x_{_W})}\\ &-\phi_{_O}\left(\frac{2}{\Delta x_{_E}\Delta x_{_W}}+\frac{2}{\Delta x_{_N}\Delta x_{_S}}\right)=S_R-\phi_{_E}\frac{2}{\Delta x_{_E}(\Delta x_{_E}+\Delta x_{_W})}\end{aligned} \tag{41b}$$

$$\begin{aligned}\text{下：}&\phi_{_E}\frac{2}{\Delta x_{_E}(\Delta x_{_E}+\Delta x_{_W})}+\phi_{_W}\frac{2}{\Delta x_{_W}(\Delta x_{_E}+\Delta x_{_W})}+\phi_{_N}\frac{2}{\Delta x_{_N}(\Delta x_{_N}+\Delta x_{_S})}\\ &-\phi_{_O}\left(\frac{2}{\Delta x_{_E}\Delta x_{_W}}+\frac{2}{\Delta x_{_N}\Delta x_{_S}}\right)=S_R-\phi_{_S}\frac{2}{\Delta x_{_S}(\Delta x_{_N}+\Delta x_{_S})}\end{aligned} \tag{41c}$$

$$\begin{aligned}\text{上：}&\phi_{_E}\frac{2}{\Delta x_{_E}(\Delta x_{_E}+\Delta x_{_W})}+\phi_{_W}\frac{2}{\Delta x_{_W}(\Delta x_{_E}+\Delta x_{_W})}+\phi_{_S}\frac{2}{\Delta x_{_S}(\Delta x_{_N}+\Delta x_{_S})}\\ &-\phi_{_O}\left(\frac{2}{\Delta x_{_E}\Delta x_{_W}}+\frac{2}{\Delta x_{_N}\Delta x_{_S}}\right)=S_R-\phi_{_N}\frac{2}{\Delta x_{_N}(\Delta x_{_N}+\Delta x_{_S})}\end{aligned} \tag{41d}$$

$$\text{左下：}\phi_N\frac{2}{\Delta x_N(\Delta x_N+\Delta x_S)}+\phi_E\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)}-\phi_O\left(\frac{2}{\Delta x_E\Delta x_W}+\frac{2}{\Delta x_N\Delta x_S}\right)=S_R-\phi_W\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)}-\phi_S\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)} \tag{41e}$$

$$\text{左上：}\phi_S\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)}+\phi_E\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)}-\phi_O\left(\frac{2}{\Delta x_E\Delta x_W}+\frac{2}{\Delta x_N\Delta x_S}\right)=S_R-\phi_W\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)}-\phi_N\frac{2}{\Delta x_N(\Delta x_N+\Delta x_S)} \tag{41f}$$

$$\text{右上：}\phi_S\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)}+\phi_W\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)}-\phi_O\left(\frac{2}{\Delta x_E\Delta x_W}+\frac{2}{\Delta x_N\Delta x_S}\right)=S_R-\phi_N\frac{2}{\Delta x_N(\Delta x_N-\Delta x_S)}-\phi_E\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)} \tag{41g}$$

$$\text{右下：}\phi_N\frac{2}{\Delta x_N(\Delta x_N-\Delta x_S)}+\phi_W\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)}-\phi_O\left(\frac{2}{\Delta x_E\Delta x_W}+\frac{2}{\Delta x_N\Delta x_S}\right)=S_R-\phi_S\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)}-\phi_E\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)} \tag{41h}$$

这种矩阵离散方法只需要求解内部节点，边界点当作已知值最后赋值给数值解即可，对于均匀网格，系数矩阵是对称正定矩阵，有很好的性质，如图14。

对于该问题，采用不同的迭代算法求解器，收敛误差均为残差向量二范数 $r_{L2}<10^{-6}$。图10给出了该问题的数值解、绝对误差和相对误差云图。图11给出了采用 21、41、81、121、161、201 六种不同离散节点数目下的相对误差关系图，可以看出这种二阶离散格式的确具有二阶精度，与所用的迭代方法无关。

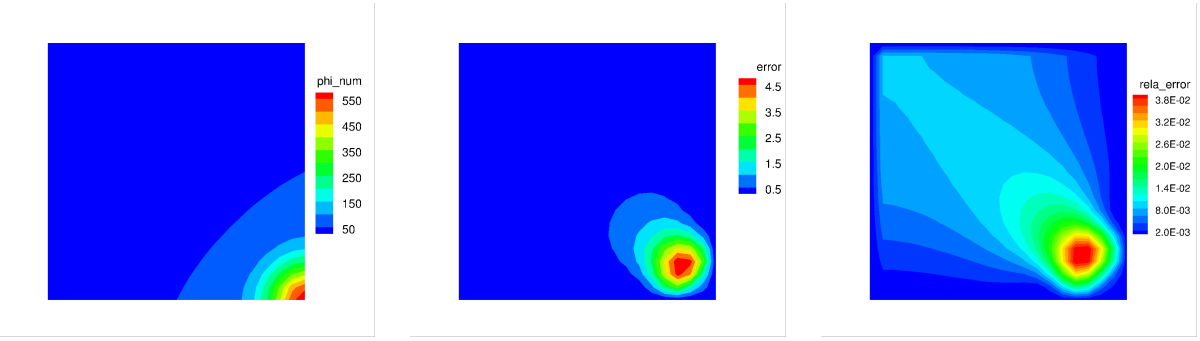
4.1 使用 Jacobi 方法

Jacobi 迭代对于系数矩阵要求较低，采用如示意图12或者图14的离散格式都可。这里采用第一种格式编程实现。

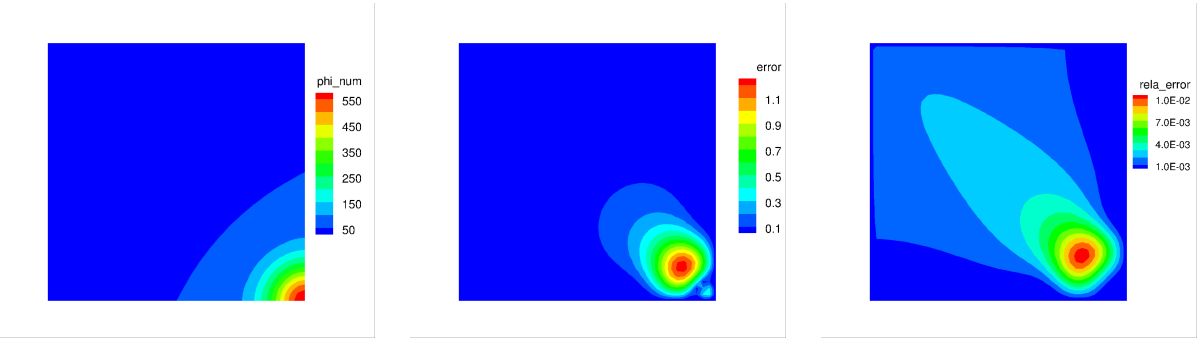
$$\phi_O^{n+1}\left(\frac{2}{\Delta x_E\Delta x_W}+\frac{2}{\Delta x_N\Delta x_S}\right)=S_R+\phi_W^n\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)}+\phi_S^n\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)}+\phi_N^n\frac{2}{\Delta x_N(\Delta x_N+\Delta x_S)}+\phi_E^n\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)} \tag{42}$$

给出 Jacobi 迭代的程序实现 c++ 子函数代码

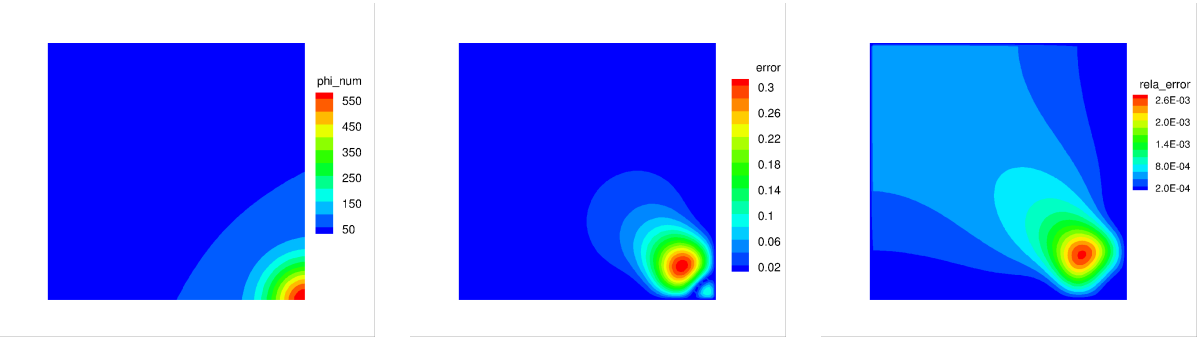
```
void Gauss_Seidel(VectorXd & X, VectorXd & Ac, VectorXd & B, const int N) {
```



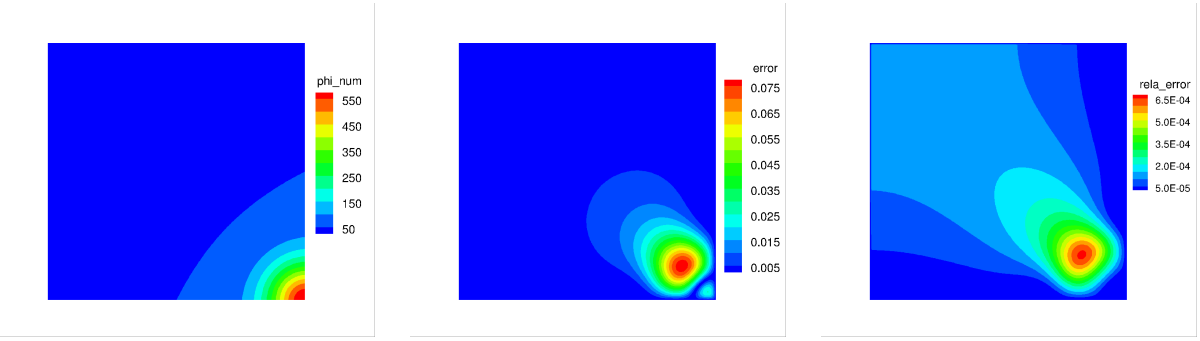
(a) 21 节点数值解　(b) 21 节点绝对误差　(c) 21 节点相对误差



(d) 41 节点数值解　(e) 41 节点绝对误差　(f) 41 节点相对误差



(g) 81 节点数值解　(h) 81 节点绝对误差　(i) 81 节点相对误差



(j) 161 节点数值解　(k) 161 节点绝对误差　(l) 161 节点相对误差

图 **10**　不同网格点数值解、与解析解误差以及相对误差云图

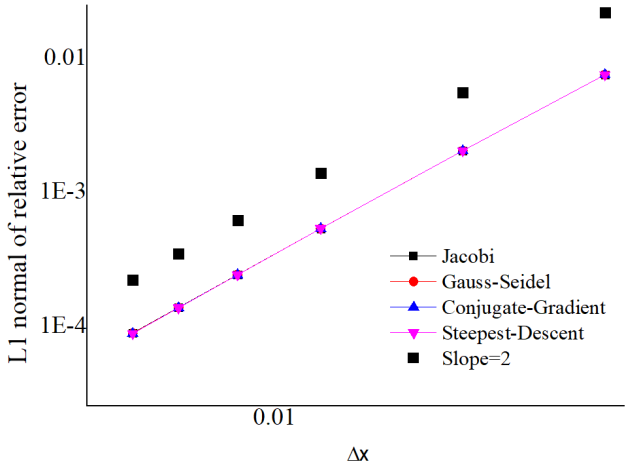


图 **11**　数值格式精度

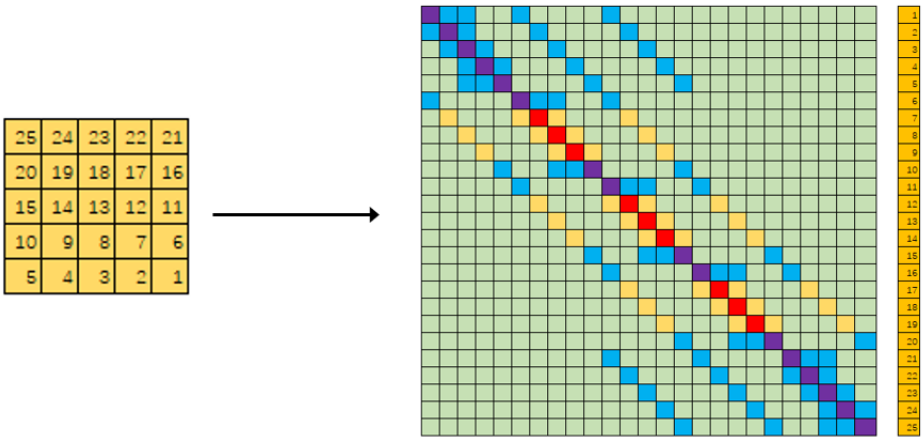


图 **12**　第一种离散格式（通用）

表 **1**　**Jacobi** 迭代结果

节点	迭代步数	计算时间 (ms)	相对误差 L1
21	23172	4.7	0.00749979
41	4775	64.6	0.00203494
81	18211	858	0.000531398
121	39796	4458	0.000240002
161	69258	13831	0.000136424
201	106408	33114	8.81666e-05

```
    for(int j = 0; j < N; ++j){
      for(int i = 0; i < N; ++i){
        const int index = coordinate(i, j, N);
        if(i != 0 && j != 0 && i != N - 1 && j != N - 1){

          const int i_L = coordinate(i - 1, j , N);
          const int i_B = coordinate(i , j - 1, N);
          const int i_R = coordinate(i + 1, j , N);
          const int i_T = coordinate(i , j + 1, N);

          const int im5 = 5 * index;

          X(index) = 1.0 / Ac(im5) * (B(index)
          - (Ac(im5 + 1) * X(i_L) + Ac(im5 + 2) * X(i_B)
          +  Ac(im5 + 3) * X(i_R) + Ac(im5 + 4) * X(i_T)));
        }
        else {
          X(index) = B(index);
        }
      }
    }
  }
```

4.2 使用 Gauss-Seidel 迭代方法

$$\phi_O^{n+1}\left(\frac{2}{\Delta x_E \Delta x_W}+\frac{2}{\Delta x_N \Delta x_S}\right)=S_R+\phi_W^{n+1}\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)} \tag{43}$$
$$+\phi_S^{n+1}\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)}+\phi_N^{n}\frac{2}{\Delta x_N(\Delta x_N+\Delta x_S)}+\phi_E^{n}\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)}$$

给出 Gauss-Seidel 迭代的程序实现 c++ 子函数代码

表 2　**Gauss-Seidel** 迭代结果

节点	迭代步数	计算时间 (ms)	相对误差 L1
21	22524	2.6	0.00749974
41	2501	45.4	0.00203485
81	9556	672	0.000531207
121	20910	3245	0.000239711
161	36428	10461	0.000136034
201	56014	24889	8.76763e-05

```
void Jacobi(VectorXd & X, VectorXd & X_old,
VectorXd & Ac, VectorXd & B, const int N) {
  for(int j = 0; j < N; ++j){
    for(int i = 0; i < N; ++i){
      const int index = coordinate(i, j, N);
      if(i != 0 && j != 0 && i != N - 1 && j != N - 1){
        const int i_L = coordinate(i - 1, j , N);
        const int i_B = coordinate(i , j - 1, N);
        const int i_R = coordinate(i + 1, j , N);
        const int i_T = coordinate(i , j + 1, N);
        const int im5 = 5 * index;
        X(index) = 1.0 / Ac(im5) * (B(index)
        - (Ac(im5 + 1) * X_old(i_L) + Ac(im5 + 2) * X_old(i_B)
        + Ac(im5 + 3) * X_old(i_R) + Ac(im5 + 4) * X_old(i_T)));
      }
      else {
        X(index) = B(index);
      }
    }
  }
}
```

除了给出的部分 Jacobi 和 Gauss-Seidel 子函数外，要完成完整的迭代计算，还需要一个迭代步，给出如下，其中 void get_iter_coeff() 函数是为了得到迭代系数。

```
void Jacobi_Or_Gauss_Seidel_Iteration(int GaussSeidel){
  using Basic::N;
  using Basic::MAX_ERR;
  using Basic::X_delta;
  using Basic::Y_delta;

  using Solve_equ::X;
  using Solve_equ::B;
  using Solve_equ::A;

  // FD coefficient----------------
  VectorXd A_coeff = VectorXd::Constant(p2(N) * 5, 0);
  get_iter_coeff(A_coeff, &X_delta[0], &Y_delta[0], N);
  double err_sum = 1.0;
  int iter_num;

  while(err_sum > MAX_ERR){
    VectorXd X_old = X;
    if(GaussSeidel){
      Gauss_Seidel(X, A_coeff, B, N);
    }
```

```
        else{
            Jacobi(X, X_old, A_coeff, B, N);
        }

        VectorXd error = X - X_old;
        err_sum = error.lpNorm<2>();

        ++iter_num;
    }
    cout << "Iter steps is " << iter_num << ", "
    << "iter max err is " << err_sum << endl;
}
```

4.3 使用最速下降法 Steepest Method

最速下降法和共轭梯度法对于系数矩阵要求很高，需要其对称正定，如果不满足这个条件，则不能得到正确的结果，该方法对于均匀网格、第一类边界条件的 Poisson 问题非常适用。 图14是这种系数矩阵的离散示意图。给出最速下降法的程序实现 c++ 子

表 **3**　**Steepest Descent** 迭代结果

节点	迭代步数	计算时间 (ms)	相对误差 L1
21	1868	14.9	0.0074997
41	7732	171.5	0.00203476
81	31862	2901	0.000531017
121	72852	16397	0.000239421
161	131064	51819	0.000135643
201	206600	134316	8.71861e-05

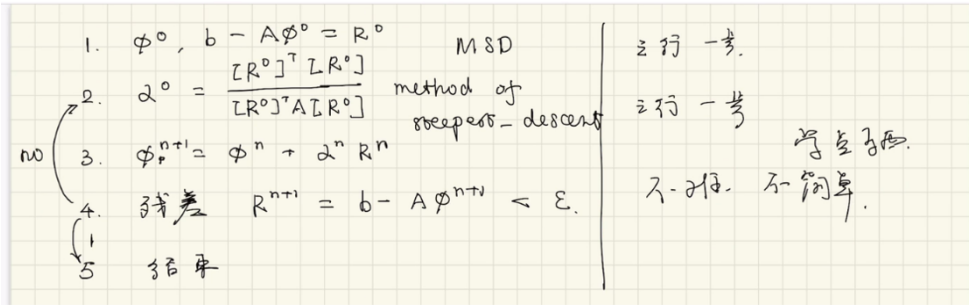


图 **13**　最速下降法算法流程图 **(by WangYang)**

函数代码，其中，void from_X1_to_X(VectorXd & X, VectorXd & X1, int N) 子函数是为了将离散求解后内部节点重构到原始的整个节点区域内。例如图14中浅蓝色的节点区域

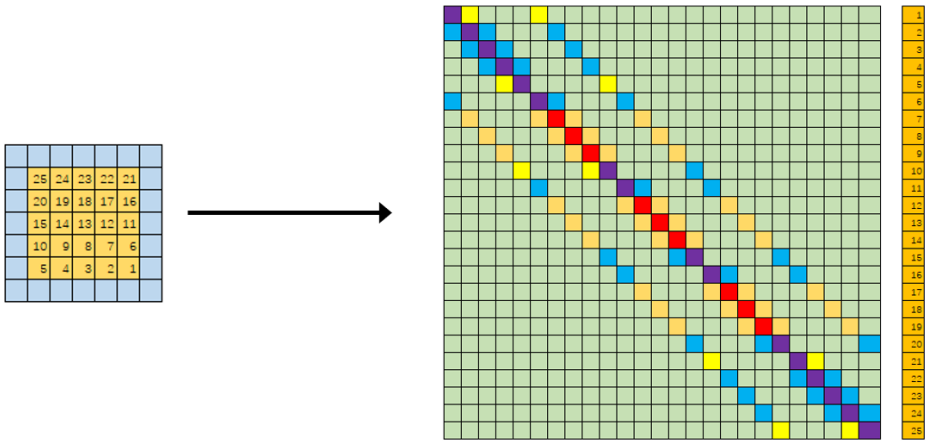


图 **14**　第二种离散格式（**SPD**，性质最好）

是我们没有参与计算的边界节点，经过函数求解之后，我们知道了内部黄色节点的数值解，此时就要利用 from_X1_to_X() 函数将黄色内部节点重新纳入原来的整个区域中。

```
void Steepest_descent_method(){
   using Basic::N;
   using Basic::MAX_ERR;
   using Solve_equ::X;
   using Solve_equ::B1;
   using Solve_equ::A1;

   VectorXd X1 = VectorXd::Constant(p2(N - 2), 0);
   VectorXd r = B1 - A1 * X1;
   for(int k = 0; k < 1000000; ++k){
      if(r.lpNorm<2>() < MAX_ERR){
         cout << "iter steps is " << k << endl;
         break;
      }
      double r_M_rT = r.transpose() * r;
      double ak = (r.transpose() * A1 * r);
      ak = r_M_rT / ak;
      X1 += ak * r;
      r = B1 - A1 * X1;
   }
   from_X1_to_X(X, X1, N);
}
```

4.4 使用共轭梯度法 Conjugate Gradient Method

给出共轭梯度法的程序实现 c++ 子函数代码

```
void Conjugate_Gradient_Method(){
```

表 4　**Conjugate Gradient** 迭代结果

节点	迭代步数	计算时间 (ms)	相对误差 L1
21	75	0.44	0.0074997
41	158	4.04	0.00203476
81	324	34.3	0.000531017
121	494	125	0.000239421
161	665	301	0.000135643
201	837	624	8.71861e-05

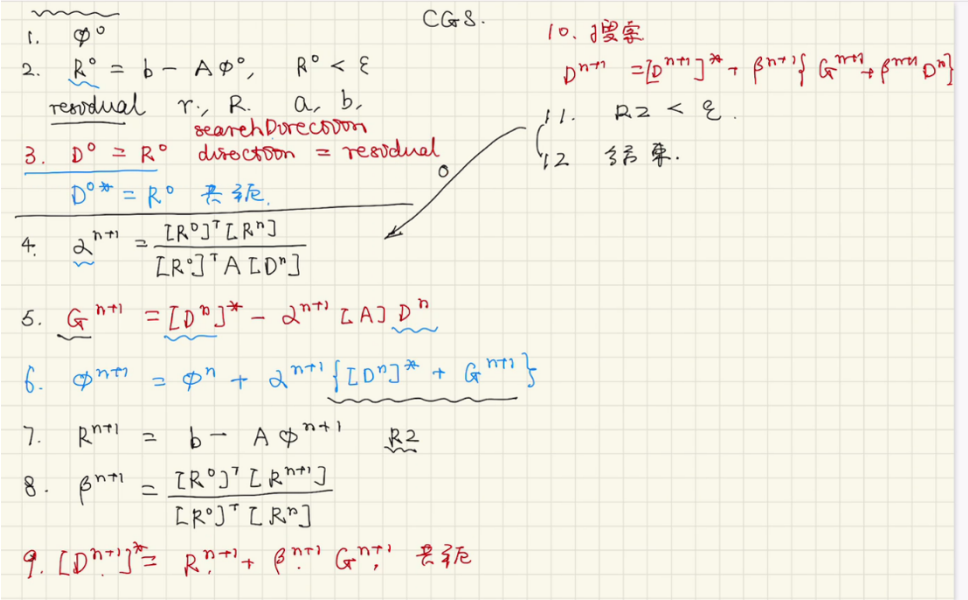


图 **15**　共轭梯度法计算流程图 **((by WangYang))**

```
using Basic::N;
using Basic::MAX_ERR;
using Solve_equ::L2_error;
using Solve_equ::X;
using Solve_equ::B1;
using Solve_equ::A1;

VectorXd X1 = VectorXd::Constant(p2(N - 2), 0);
VectorXd r = B1 - A1 * X1;
VectorXd P = r;

for(int k = 0; k < 1000000; ++k){
   double r_M_rT = r.transpose() * r;
   double ak = (P.transpose() * A1 * P);
   ak = r_M_rT / ak;
   X1 += ak * P;
```

```
    VectorXd r_new = r - ak * A1 * P;
    if(r_new.lpNorm<2>() < MAX_ERR){
       cout << "iter steps is " << k << endl;
       break;
    }
    double betak = (r_new.transpose() * r_new);
    betak = betak / r_M_rT;
    P = r_new + betak * P;
    r = r_new;
  }
  from_X1_to_X(X, X1, N);
}
```

综合表1、表2、表3、表4这四种迭代方法的结果，可以看出，共轭梯度法具有明显的收敛优势，包括收敛速度和算法迭代步数，由于机器误差的出现，共轭梯度法不能以理论的收敛步数完成收敛，但是收敛速度相比于其他三种，仍然是数量级的优势，最速下降法收敛最慢。考虑到这两种算法对系数矩阵的高要求，需要系数矩阵对称正定，能够采用 CG 算法的情况下尽量采用 CG 算法，相比之下，最速下降法则是最不经济的一种迭代算法。高斯塞德尔迭代相比于雅可比迭代方法具有明显的收敛速度优势，但是当节点数很小时（比如 21），其迭代步数反而更多，但是由于矩阵变小，收敛速度仍旧低于雅可比法。但是雅可比法易于并行，且串行情况下，这二者收敛速度并没有数量级的差别，这是雅可比法一个潜在的优势。

4.5 线扫略方法简述

在此只给出适用于 Jacobi 迭代的先扫略方法，将系数矩阵分块求解，可并行。

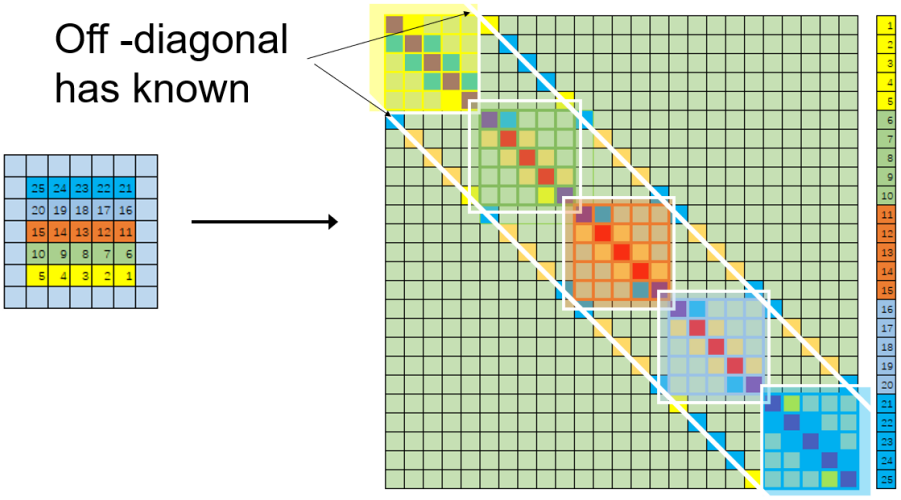


图 **16**　第三种离散格式（可并行）