第一次作业第 2、3 题

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。

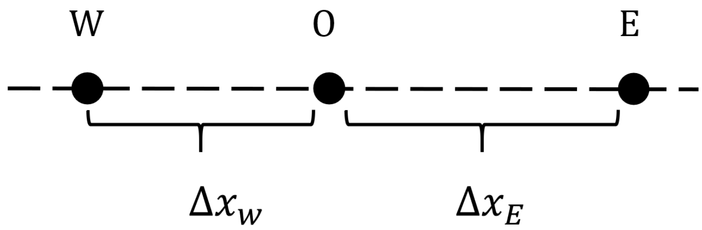


图 **1** 网格示意图

对 O 点左右两点 W 和 E 做泰勒展开，可得

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

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

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

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

其中，

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

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

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

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

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

(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，

$$\begin{aligned} \left(\frac{d^2\phi}{dx^2}\right)\bigg|_{x=O}= & \frac{2\phi_E}{\Delta x_E(\Delta x_E+\Delta x_W)}-\frac{2\phi_O}{\Delta x_E\Delta x_W}+\frac{2\phi_W}{\Delta x_W(\Delta x_E+\Delta x_W)} \\ & -\left(\frac{d^3\phi}{dx^3}\right)\bigg|_{x=O}\frac{\Delta x_E-\Delta x_W}{3}+O(\Delta x^2) \end{aligned} \tag{9}$$

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

$$\left(\frac{d^2\phi}{dx^2}\right)\bigg|_{x=O} = \frac{2\phi_E}{S(1+S){\Delta x_W}^2} - \frac{2\phi_O}{S{\Delta x_W}^2} + \frac{2\phi_W}{(1+S){\Delta x_W}^2} = R_S \tag{10}$$

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：

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

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

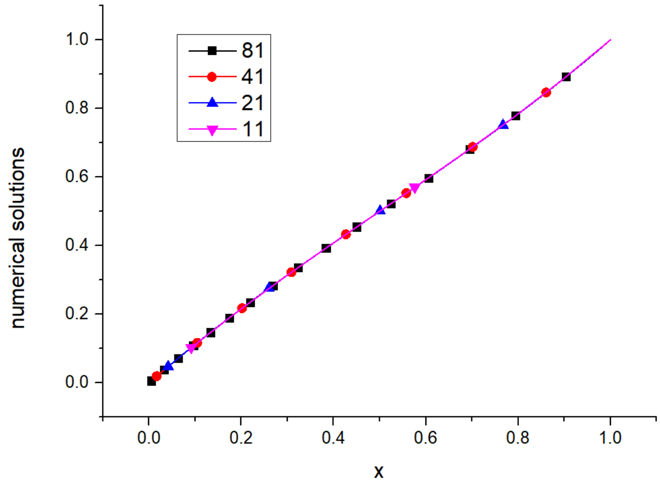


图 **2**　数值解

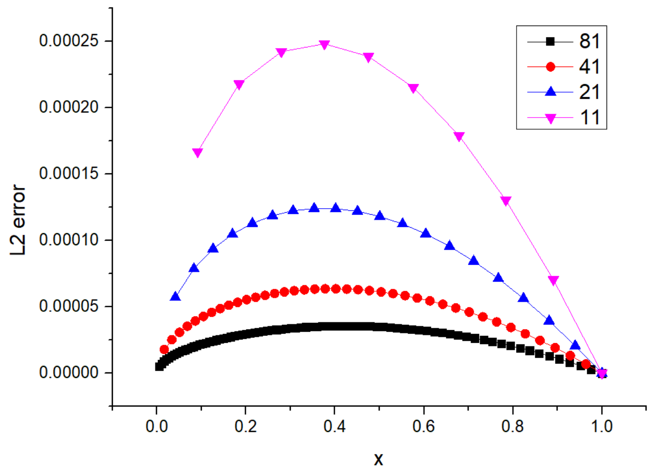


图 **3**　相对误差 (二范数)

1.4 C 语言源程序

```
#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,

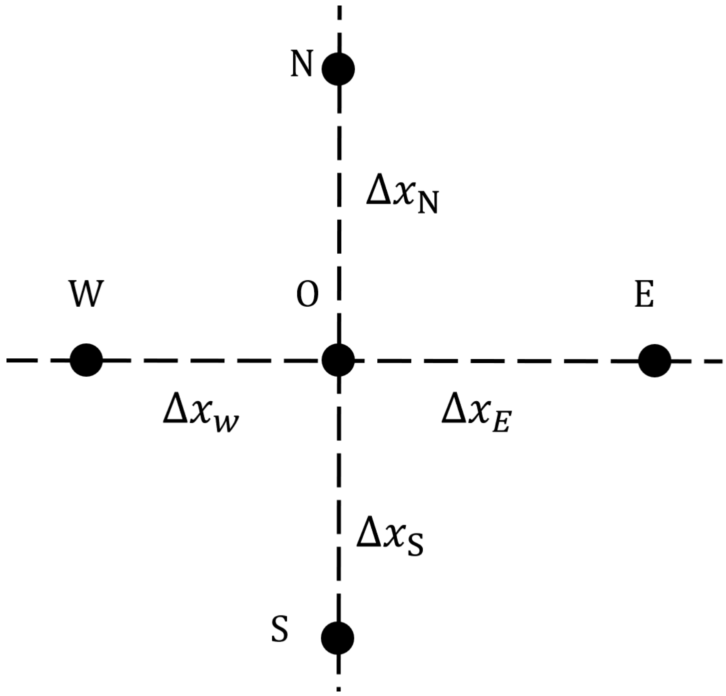


图 **4**　网格示意图

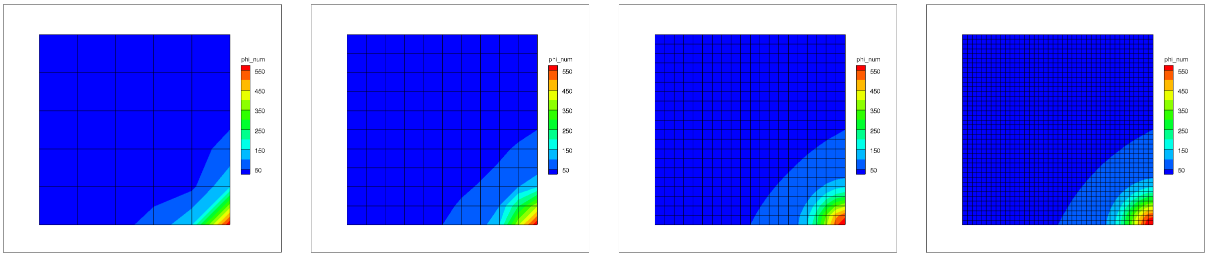
根据式 (7)，忽略误差项，加入 y 方向节点，二维情况下二阶导离散变成

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

此时，原二维 Poisson 方程内节点离散格式为

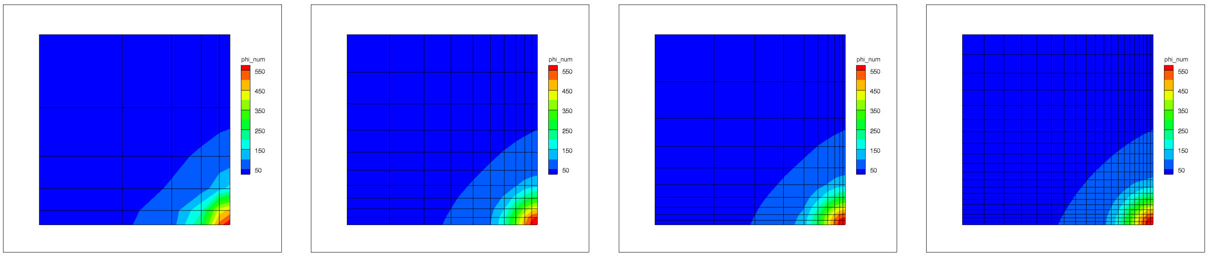
$$\begin{aligned}&\phi_E\frac{2}{\Delta x_E(\Delta x_E+\Delta x_W)} + \phi_W\frac{2}{\Delta x_W(\Delta x_E+\Delta x_W)} \\ &+ \phi_N\frac{2}{\Delta x_N(\Delta x_N+\Delta x_S)} + \phi_S\frac{2}{\Delta x_S(\Delta x_N+\Delta x_S)} - \phi_O\left(\frac{2}{\Delta x_E\Delta x_W} + \frac{2}{\Delta x_N\Delta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

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。



(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

2.2 计算结果

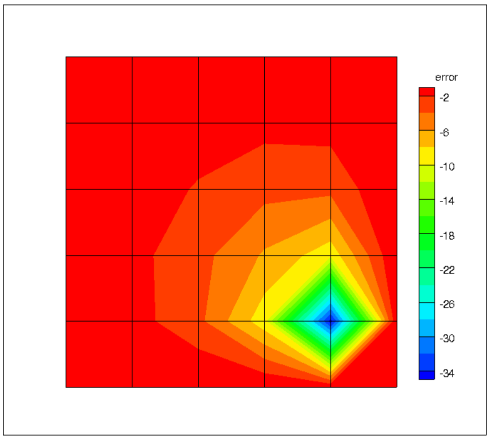
图5和图6给出了离散节点为 6，11，21 和 41 的均匀网格数值解和离散节点为 6，11，21 三种情况的数值解，其中，非均匀网格 x、y 方向网格离散尺度关系与上一题相同，公比为 S_x 分别为 S_y，不同节点数目的结果公比可能不同。

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）。

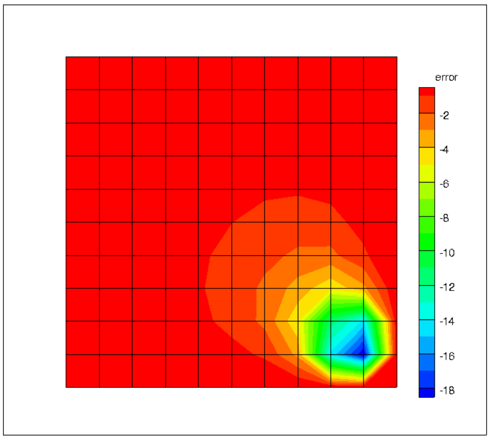
$$L_{error} = \phi_{num} = \phi_{ana} \tag{17}$$

2.3 C++ 源程序

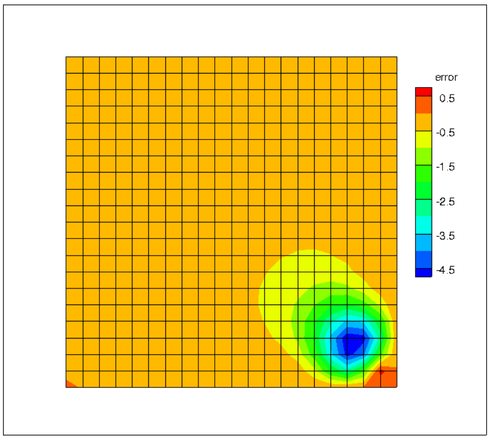
```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
#include <string>
```



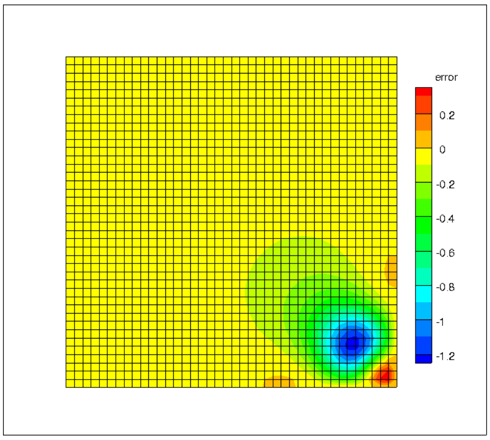
(a) 6 节点



(b) 11 节点



(c) 21 节点



(d) 41 节点

图 **7** 均匀网格离散误差

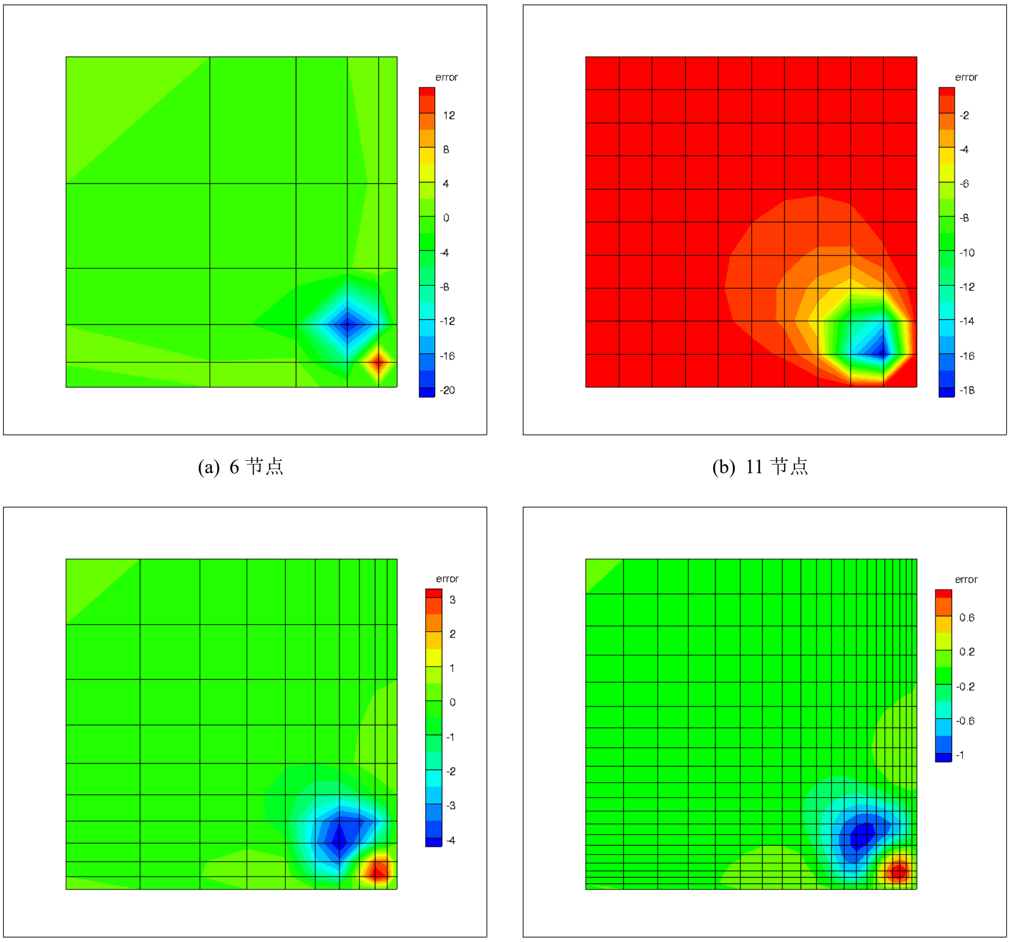
```
#include <Eigen/Dense>


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);
}

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



(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

```
  //}

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();
}
```