作业 5：计算 2D Poisson 方程数值解

一、学生信息

姓名：柳建国
学号：2022Z8017782089
专业：电子信息
所部：数字所

二、问题描述

如图 1 所示，计算 2D Poisson 方程数值解。

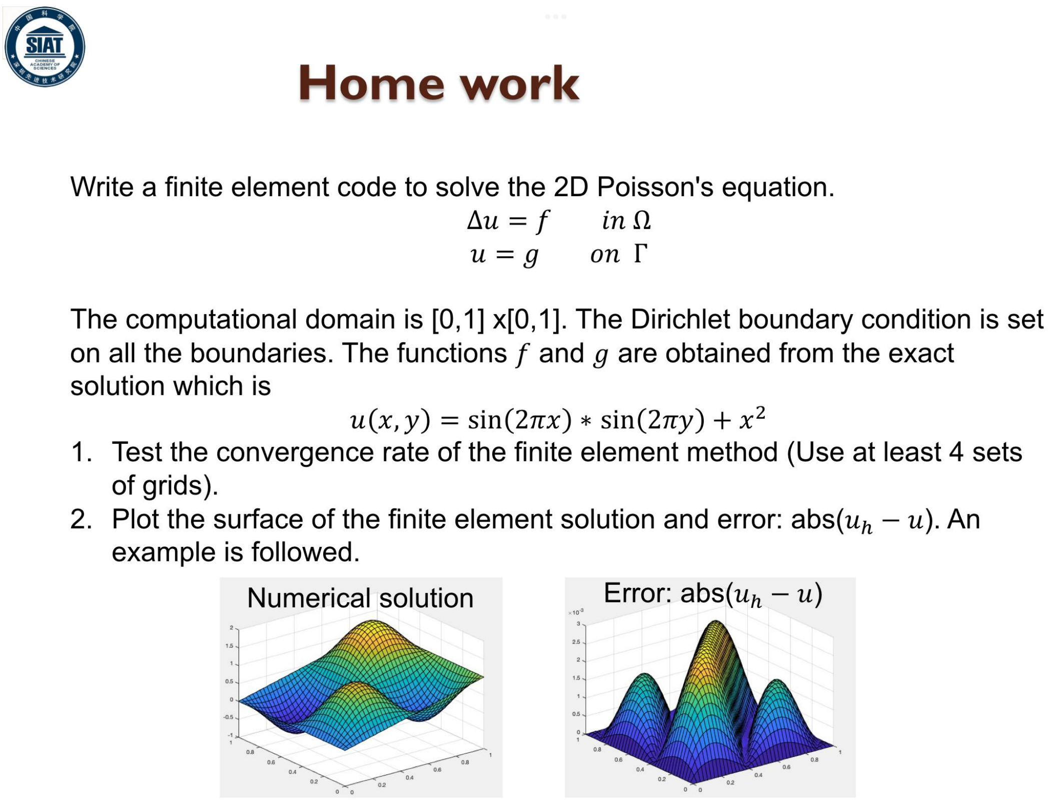

图 1 2D Poisson 方程

三、实验过程

1. 实验原理

本次实验使用 Matlab 进行实现，相关代码见附录。

设网格数目为 nx,ny，根据题目有：

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

$$\frac{du}{dx} = 2\pi cos(2\pi x) * sin(2\pi y) + 2x$$

$$\frac{du}{dy} = 2\pi sin(2\pi x) * cos(2\pi y)$$

$$\frac{d^2u}{dx^2} =- 4\pi^2 sin(2\pi x)sin(2\pi y) + 2$$

$$\frac{d^2u}{dy^2} =- 4\pi^2 sin(2\pi x)sin(2\pi y)$$

$$\Delta \mathrm{u} =- \frac{d^2u}{dx^2} - \frac{d^2u}{dy^2} = 8\pi^2 sin(2\pi x)sin(2\pi y) - 2$$

根据上述公式修改相应代码，即可以得到数值解，相关代码见附录。

2. 测试结果

分别测试了网格为 10×10、20×20、50×50、100×100 情况下的数值解和相应误差，得到结果如下：

(1) 当网格点数为 nx=ny=10 时，得到的数值解绘制图像如图 2 所示，误差情况如图 3 所示。

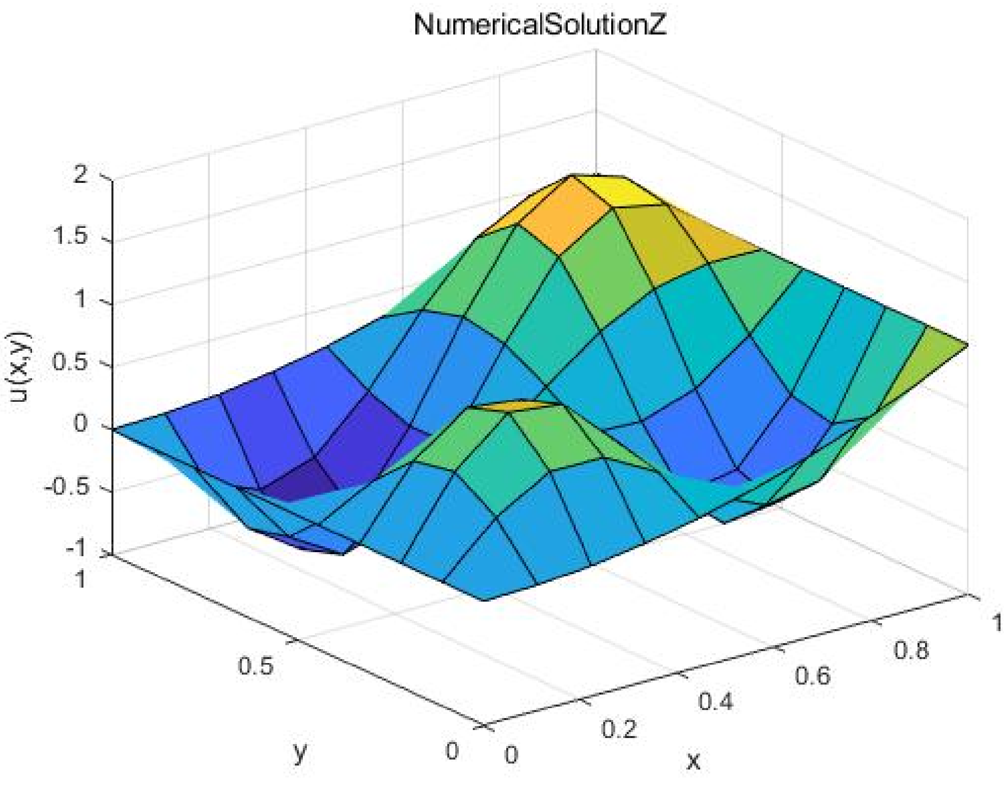

图 2 当网格为 nx=ny=10 时，得到的数值解结果图

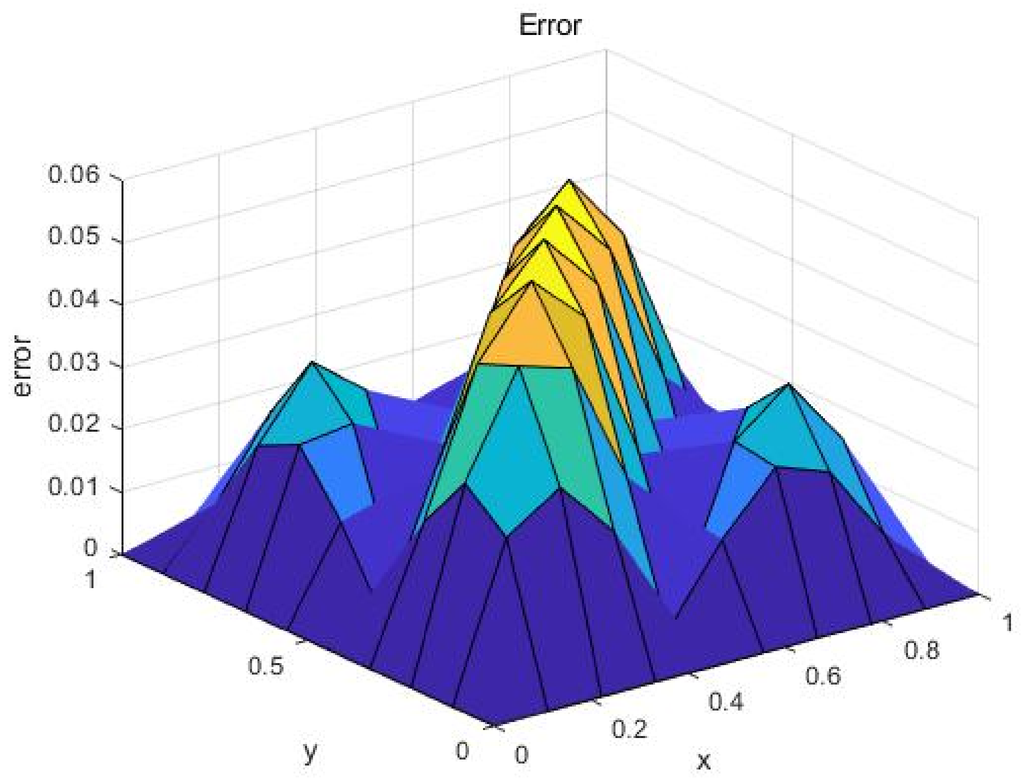

图 3 当网格为 nx=ny=10，数值解与精确解误差图

(2) 当网格点数为 nx=ny=20 时，得到的数值解绘制图像如图 4 所示，误差情况如图 5 所示。

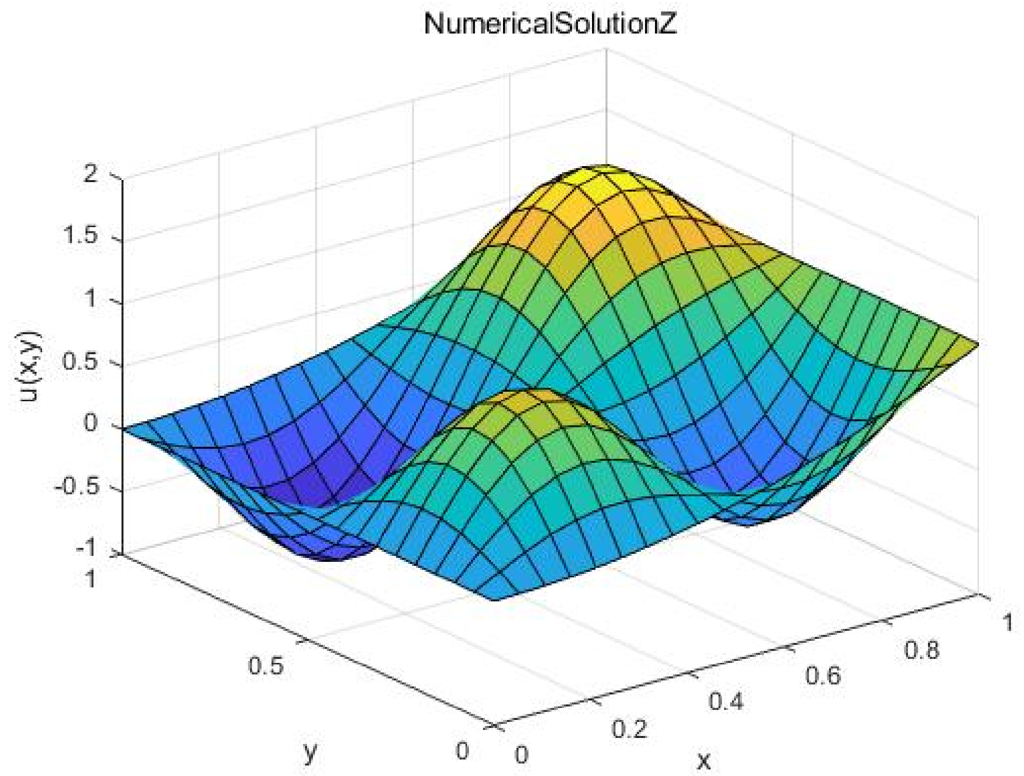

图 4 当网格为 nx=ny=20 时，得到的数值解结果图。

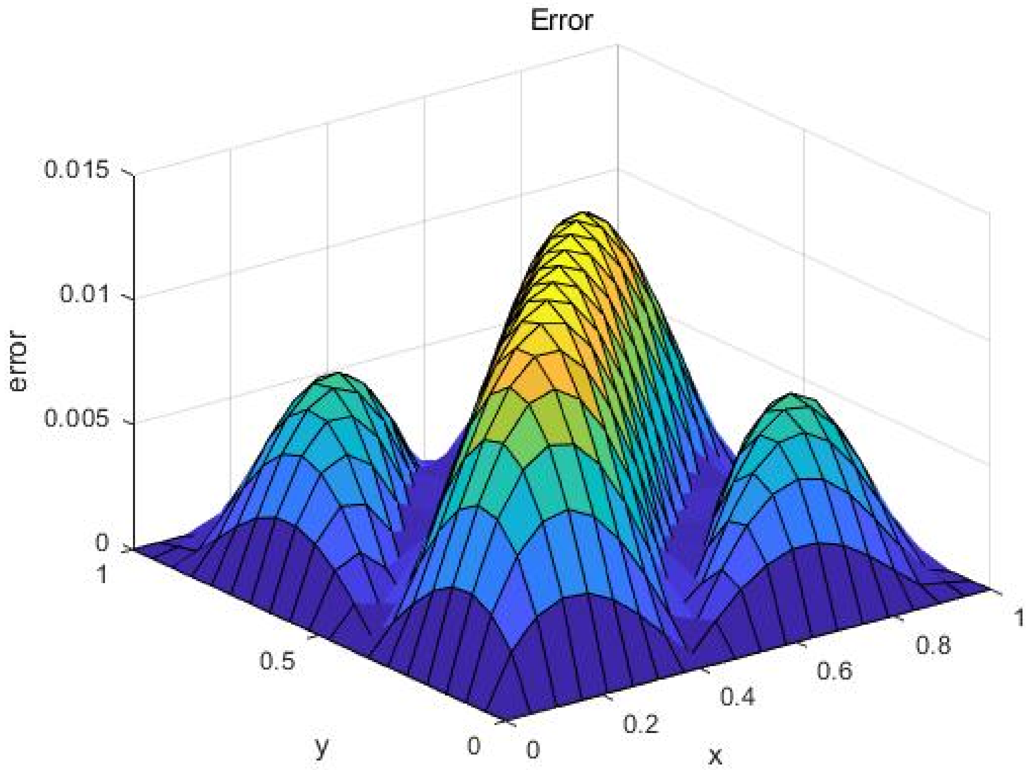

图 5 当网格为 nx=ny=20，数值解与精确解误差图

(3) 当网格点数为 nx=ny=50 时，得到的数值解绘制图像如图 6 所示，误差情况如图 7 所示。

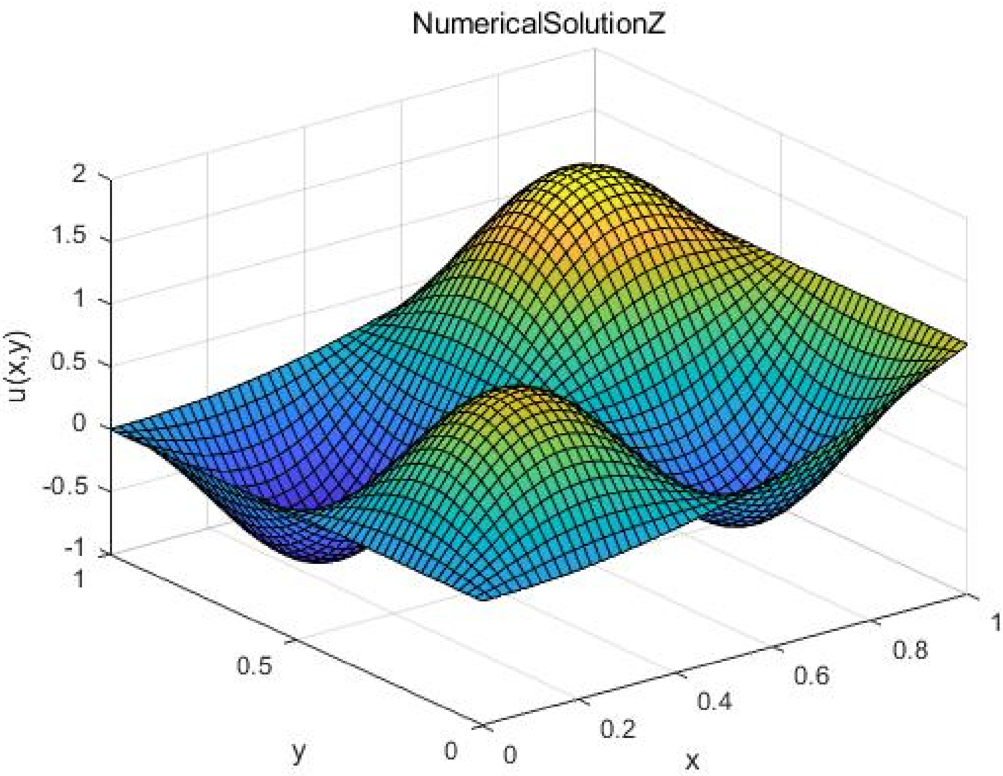

图 6 当网格为 nx=ny=50 时，得到的数值解结果图

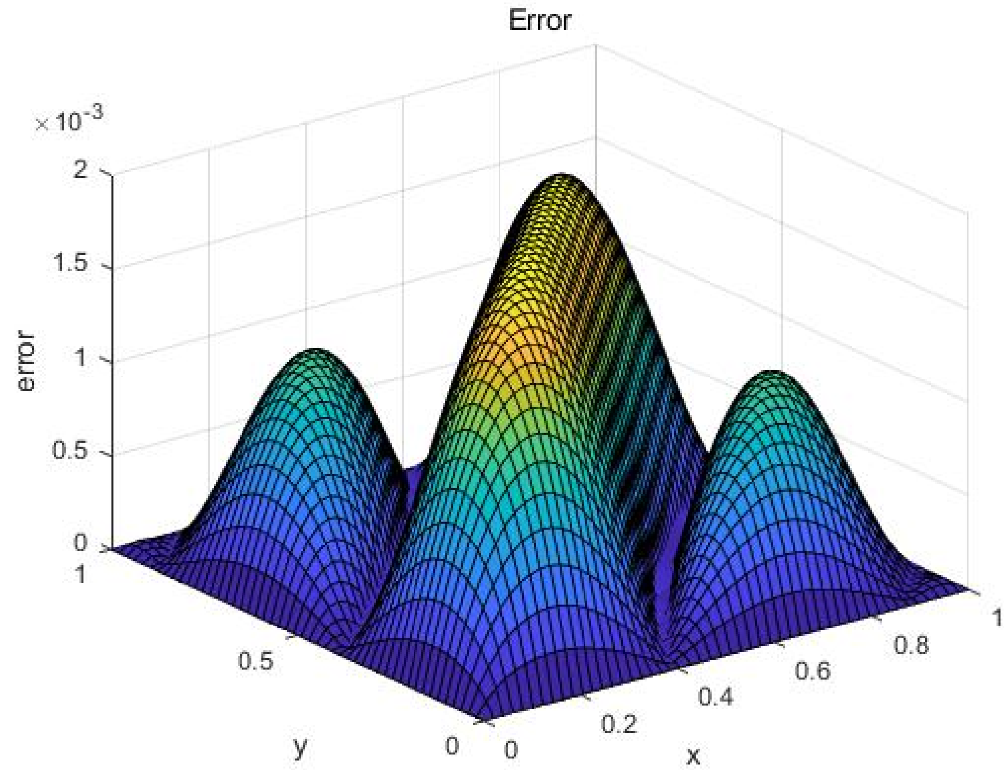

图 7 当网格为 nx=ny=50，数值解与精确解误差图

(4) 当网格点数为 nx=ny=100 时，得到的数值解绘制图像如图 8 所示，误差情况如图 9 所示。

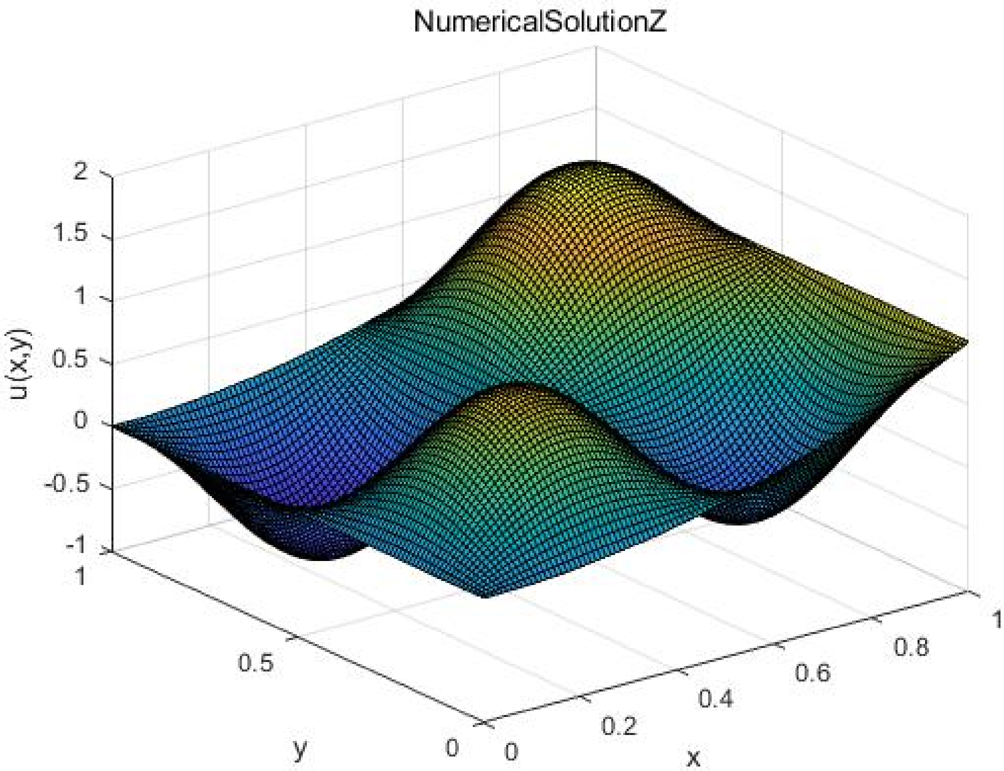

图 8 当网格为 nx=ny=100 时，得到的数值解结果图

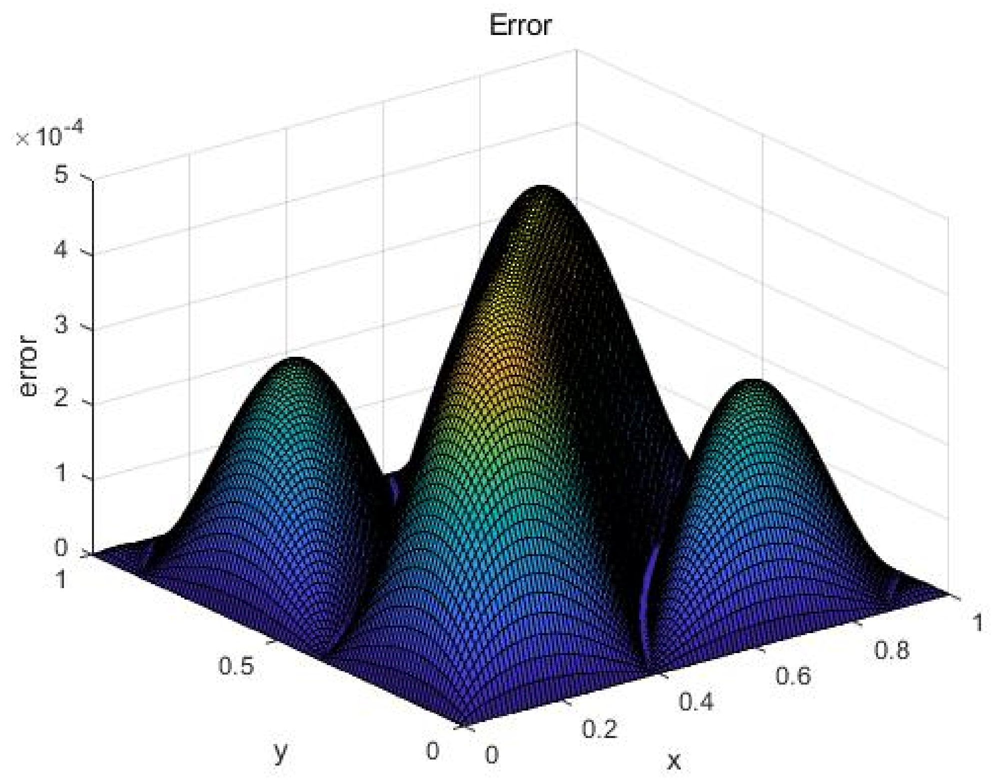

图 9 当网格为 nx=ny=100，数值解与精确解误差图

四、源代码-Matlab 实现

1. 主函数：MyFEM_homework5.m

功能：绘调用 fem2d_poisson_rectangle_linear 函数得到数值解，绘制数值解图像，绘制精确解图像，计算数值解与精确解之间的误差，并绘制误差图像。

```
clear;clc;close all;
% 设置网格大小
nx=100;
ny=100;

% 调用函数得到数值解结果
NumericalSolution=fem2d_poisson_rectangle_linear ( nx, ny );

% 对数值解结果进行格式修改，用于绘图和求误差
k = 0;
for j = 1 : ny
    for i = 1 : nx
      k = k + 1;
      NumericalSolutionZ(j,i)=NumericalSolution(k);
    end
```

```
end

% 为绘制数值解，定义网格
x = linspace(0,1,nx);
y = linspace(0,1,ny);
[X,Y] = meshgrid(x,y);

% 绘制数值解图
figure;
surf(X,Y,NumericalSolutionZ);
xlabel('x');
ylabel('y');
zlabel('u(x,y)');
title('NumericalSolutionZ');

% 定义精确解函数
u = @(x,y) sin(2*pi*x) .* sin(2*pi*y) + x.^2;

% 在网格上面计算精确解结果
ExactSolutionZ = u(X,Y);

% 绘制精确解结果图
figure;
surf(X,Y,ExactSolutionZ);
xlabel('x');
ylabel('y');
zlabel('u(x,y)');
title('ExactSolutionZ');

% 计算误差
Error=abs(ExactSolutionZ-NumericalSolutionZ);
% 绘制误差图
figure;
surf(X,Y,Error);
xlabel('x');
ylabel('y');
zlabel('error');
title('Error')
```

2. 函数 fem2d_poisson_rectangle_linear.m

修改参考代码，实现功能：计算 2D Poisson 数值解，并返回数值解结果。

```
function NumericalSolution=fem2d_poisson_rectangle_linear ( nx, ny )

%*****************************************************************************80
%
%% fem2d_poisson_rectangle_linear() solves the Poisson equation in a rectangle.
%
%  Discussion:
%
%    This program solves
%
%      - d2U(X,Y)/dx2 - d2U(X,Y)/dy2 = F(X,Y)
%
%    in a rectangular region in the plane.
%
%    Along the boundary of the region, Dirichlet conditions
%    are imposed:
%
%      U(X,Y) = G(X,Y)
%
%    The code uses continuous piecewise linear basis functions on
%    triangles determined by a uniform grid of NX by NY points.
%
%    u    =      sin ( pi * x ) * sin ( pi * y ) + x
%
%    dudx = pi * cos ( pi * x ) * sin ( pi * y ) + 1
%    dudy = pi * sin ( pi * x ) * cos ( pi * y )
%
%    d2udx2 = - pi * pi * sin ( pi * x ) * sin ( pi * y )
%    d2udy2 = - pi * pi * sin ( pi * x ) * sin ( pi * y )
%
%    rhs  = 2 * pi * pi * sin ( pi * x ) * sin ( pi * y )
%
%  THINGS YOU CAN EASILY CHANGE:
%
%    1) Change NX or NY, the number of nodes in the X and Y directions.
%    2) Change XL, XR, YB, YT, the left, right, bottom and top limits
%       of the rectangle.
%    3) Change the exact solution in the EXACT routine, but make sure you also
%       modify the formula for RHS in the assembly portion of the program.
%
%  HARDER TO CHANGE:
```

```
%
%    4) Change from "linear" to "quadratic" triangles;
%    5) Change the region from a rectangle to a general triangulated region;
%    6) Handle Neumann boundary conditions.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    01 November 2010
%
%  Author:
%
%    John Burkardt
%
%  Input:
%
%    integer NX, NY, the number of nodes in the X and Y directions.
%
%  Local:
%
%    sparse real A(NODE_NUM,NODE_NUM), the finite element system matrix.
%
%    real B(NODE_NUM), the finite element right hand side.
%
%    real C(NODE_NUM), the finite element coefficient vector.
%
%    integer ELEMENT_NODE(3,ELEMENT_NUM), the indices of the nodes
%    that form each element.
%
%    integer ELEMENT_NUM, the number of elements.
%
%    integer NODE_NUM, the number of nodes.
%
%    real NODE_XY(2,NODE_NUM), the X and Y coordinates of each node.
%
%    real XL, the X coordinate of the left boundary.
%
%    real XR, the X coordinate of the right boundary.
%
%    real YB, the Y coordindate of the bottom boundary.
%
```

```
%   real YT, the Y coordinate of the top boundary.
%
  timestamp ( );

  element_order = 3;

  xl = 0.0;
  xr = 1.0;
  yb = 0.0;
  yt = 1.0;

  fprintf ( 1, '\n' );
  fprintf ( 1, 'FEM2D_POISSON_RECTANGLE_LINEAR\n' );
  fprintf ( 1, '  MATLAB/Octave version %s\n', version ( ) );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  Solution of the Poisson equation:\n' );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  - Uxx - Uyy = F(x,y) inside the region,\n' );
  fprintf ( 1, '       U(x,y) = G(x,y) on the boundary of the region.\n' );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  The region is a rectangle, defined by:\n' );
  fprintf ( 1, '\n' );
  fprintf ( 1, ' %g = XL<= X <= XR = %g\n', xl, xr );
  fprintf ( 1, ' %g = YB<= Y <= YT = %g\n', yb, yt );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  The finite element method is used, with piecewise\n' );
  fprintf ( 1, '  linear basis functions on 3-node triangular\n' );
  fprintf ( 1, '  elements.\n' );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  The corner nodes of the triangles are generated by
an\n' );
  fprintf ( 1, '  underlying grid whose dimensions are\n' );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  NX =                       %d\n', nx );
  fprintf ( 1, '  NY =                       %d\n', ny );
%
%  NODE COORDINATES
%
%  Numbering of nodes is suggested by the following 5x10 example:
%
%    J=5 | K=41  K=42 ... K=50
%    ... |
%    J=2 | K=11  K=12 ... K=20
%    J=1 | K= 1  K= 2     K=10
```

```
%        +--------------------
%          I= 1  I= 2 ... I=10
%
  node_num = nx * ny;

  fprintf ( 1, '  Number of nodes =         %d\n', node_num );

  node_xy = zeros(2,node_num);

  k = 0;
  for j = 1 : ny
    for i = 1 : nx

      k = k + 1;

      node_xy(1,k) = ( ( nx - i     ) * xl   ...
                     + (      i - 1 ) * xr ) ...
                     / ( nx     - 1 );

      node_xy(2,k) = ( ( ny - j     ) * yb   ...
                     + (      j - 1 ) * yt ) ...
                     / ( ny     - 1 );

    end
  end
%
%  ELEMENT array
%
%  Organize the nodes into a grid of 3-node triangles.
%  Here is part of the diagram for a 5x10 example:
%
%    |  \ |  \ |  \ |
%    |   \|   \|   \|
%   21---22---23---24--
%    |\ 8 |\10 |\12 |
%    | \  | \  | \  |
%    |  \ |  \ |  \ |  \ |
%    |  7\|  9\| 11\|   \|
%   11---12---13---14---15---16---17---18---19---20
%    |\ 2 |\ 4 |\ 6 |\  8|                |\ 18|
%    | \  | \  | \  | \  |               | \  |
%    |  \ |  \ |  \ |  \ |    ...         |  \ |
%    |  1\|  3\|  5\| 7 \|                |17 \|
%    1----2----3----4----5----6----7----8----9---10
```

```
%
  element_num = 2 * ( nx - 1 ) * ( ny - 1 );

  fprintf ( 1, '  Number of elements =       %d\n', element_num );

  element_node = zeros ( element_order, element_num );

  k = 0;

  for j = 1 : ny - 1
    for i = 1 : nx - 1
%
%      (I,J+1)-
%       |  \
%       |   \
%       |    \  |
%     (I,J)---(I+1,J)
%
      k = k + 1;
      element_node(1,k) = i     + ( j - 1 ) * nx;
      element_node(2,k) = i + 1 + ( j - 1 ) * nx;
      element_node(3,k) = i     +   j       * nx;
%
%     (I,J+1)--(I+1,J+1)
%       |  \    |
%           \   |
%            \  |
%          -(I+1,J)
%
      k = k + 1;
      element_node(1,k) = i + 1 +   j       * nx;
      element_node(2,k) = i     +   j       * nx;
      element_node(3,k) = i + 1 + ( j - 1 ) * nx;

    end
  end
%
%  ASSEMBLE THE SYSTEM
%
%  Assemble the coefficient matrix A and the right-hand side B of the
%  finite element equations, ignoring boundary conditions.
%
  b = zeros(node_num,1);
  a = sparse ( [], [], [], node_num, node_num );
```

```
  for e = 1 : element_num

    i1 = element_node(1,e);
    i2 = element_node(2,e);
    i3 = element_node(3,e);

    area = 0.5 * ...
      ( node_xy(1,i1) * ( node_xy(2,i2) - node_xy(2,i3) ) ...
      + node_xy(1,i2) * ( node_xy(2,i3) - node_xy(2,i1) ) ...
      + node_xy(1,i3) * ( node_xy(2,i1) - node_xy(2,i2) ) );
%
%  Consider each quadrature point.
%  Here, we use the midside nodes as quadrature points.
%
    for q1 = 1 : 3

      q2 = mod ( q1, 3 ) + 1;

      nq1 = element_node(q1,e);
      nq2 = element_node(q2,e);

      xq = 0.5 * ( node_xy(1,nq1) + node_xy(1,nq2) );
      yq = 0.5 * ( node_xy(2,nq1) + node_xy(2,nq2) );
      wq = 1.0 / 3.0;
%
%  Consider each test function in the element.
%
      for ti1 = 1 : element_order

        ti2 = mod ( ti1,     3 ) + 1;
        ti3 = mod ( ti1 + 1, 3 ) + 1;

        nti1 = element_node(ti1,e);
        nti2 = element_node(ti2,e);
        nti3 = element_node(ti3,e);

        qi = 0.5 * ( ...
            ( node_xy(1,nti3) - node_xy(1,nti2) ) * ( yq -
node_xy(2,nti2) ) ...
          - ( node_xy(2,nti3) - node_xy(2,nti2) ) * ( xq -
node_xy(1,nti2) ) ) ...
          / area;
        dqidx = - 0.5 * ( node_xy(2,nti3) - node_xy(2,nti2) ) / area;
```

```
        dqidy =   0.5 * ( node_xy(1,nti3) - node_xy(1,nti2) ) / area;

%         rhs = 2.0 * pi * pi * sin ( pi * xq ) * sin ( pi * yq );
        rhs = 8.0 * pi * pi * sin ( 2 * pi * xq ) * sin ( 2 * pi * yq )
- 2;%修改点

        b(nti1) = b(nti1) + area * wq * rhs * qi;
%
%  Consider each basis function in the element.
%
        for tj1 = 1 : element_order

          tj2 = mod ( tj1,     3 ) + 1;
          tj3 = mod ( tj1 + 1, 3 ) + 1;

          ntj1 = element_node(tj1,e);
          ntj2 = element_node(tj2,e);
          ntj3 = element_node(tj3,e);

          qj = 0.5 * ( ...
              ( node_xy(1,ntj3) - node_xy(1,ntj2) ) * ( yq -
node_xy(2,ntj2) ) ...
            - ( node_xy(2,ntj3) - node_xy(2,ntj2) ) * ( xq -
node_xy(1,ntj2) ) ) ...
              / area;
          dqjdx = - 0.5 * ( node_xy(2,ntj3) - node_xy(2,ntj2) ) / area;
          dqjdy =   0.5 * ( node_xy(1,ntj3) - node_xy(1,ntj2) ) / area;

          a(nti1,ntj1) = a(nti1,ntj1) ...
            + area * wq * ( dqidx * dqjdx + dqidy * dqjdy );

        end

      end

    end

  end
%
%  BOUNDARY CONDITIONS
%
%  If the K-th variable is at a boundary node, replace the K-th finite
%  element equation by a boundary condition that sets the variable to U(K).
%
```

```
  k = 0;

  for j = 1 : ny

    for i = 1 : nx

      k = k + 1;

      if ( i == 1 | i == nx | j == 1 | j == ny )

        [ u, dudx, dudy ] = exact ( node_xy(1,k), node_xy(2,k) );

        a(k,1:node_num) = 0.0;
        a(k,k)          = 1.0;
        b(k)            = u;

      end
    end
  end
%
%  SOLVE the linear system A * C = B.
%
  c = a \ b;
%
%  COMPARE computed and exact solutions at the nodes.
%
%  fprintf ( 1, '\n' );
%  fprintf ( 1, '     K     I     J          X           Y        U
U\n' );
%  fprintf ( 1, '                                                 exact
computed \n' );

  k = 0;

  for j = 1 : ny
 %   fprintf ( 1, '\n' );
    for i = 1 : nx

      k = k + 1;

      [ u, dudx, dudy ] = exact ( node_xy(1,k), node_xy(2,k) );

%     fprintf ( 1, '  %4d  %4d  %4d  %10g  %10g  %14g  %14g  %14g\n', ...
%       k, i, j, node_xy(1,k), node_xy(2,k), u, c(k), abs ( u - c(k) ) );
```

```
    end

  end
%
%  Now that the solution has been computed,
%  compute integrals that estimate error.
%
  want_error = true;

  if ( want_error )

    el2 = 0.0;
    eh1 = 0.0;

    for e = 1 : element_num

      i1 = element_node(1,e);
      i2 = element_node(2,e);
      i3 = element_node(3,e);

      area = 0.5 * ...
        ( node_xy(1,i1) * ( node_xy(2,i2) - node_xy(2,i3) ) ...
        + node_xy(1,i2) * ( node_xy(2,i3) - node_xy(2,i1) ) ...
        + node_xy(1,i3) * ( node_xy(2,i1) - node_xy(2,i2) ) );
%
%  Consider each quadrature point.
%  Here, we use the midside nodes as quadrature points.
%
      for q1 = 1 : 3

        q2 = mod ( q1, 3 ) + 1;

        nq1 = element_node(q1,e);
        nq2 = element_node(q2,e);

        xq = 0.5 * ( node_xy(1,nq1) + node_xy(1,nq2) );
        yq = 0.5 * ( node_xy(2,nq1) + node_xy(2,nq2) );
        wq = 1.0 / 3.0;

        uh = 0.0;
        dudxh = 0.0;
        dudyh = 0.0;
```

```
      for tj1 = 1 : element_order

        tj2 = mod ( tj1,     3 ) + 1;
        tj3 = mod ( tj1 + 1, 3 ) + 1;

        ntj1 = element_node(tj1,e);
        ntj2 = element_node(tj2,e);
        ntj3 = element_node(tj3,e);

        qj = 0.5 * ( ...
             ( node_xy(1,ntj3) - node_xy(1,ntj2) ) * ( yq -
node_xy(2,ntj2) ) ...
           - ( node_xy(2,ntj3) - node_xy(2,ntj2) ) * ( xq -
node_xy(1,ntj2) ) ) ...
             / area;
        dqjdx = - 0.5 * ( node_xy(2,ntj3) - node_xy(2,ntj2) ) / area;
        dqjdy =   0.5 * ( node_xy(1,ntj3) - node_xy(1,ntj2) ) / area;

        uh    = uh    + c(ntj1) * qj;
        dudxh = dudxh + c(ntj1) * dqjdx;
        dudyh = dudyh + c(ntj1) * dqjdy;

      end

      [ u, dudx, dudy ] = exact ( xq, yq );

      el2 = el2 + ( uh - u )^2 * area;
      eh1 = eh1 + ( ( dudxh - dudx )^2 + ( dudyh - dudy )^2 ) * area;

    end

  end

  el2 = sqrt ( el2 );
  eh1 = sqrt ( eh1 );

  fprintf ( 1, '\n' );
  fprintf ( 1, '*********************************************\n' );
  fprintf ( 1, '*                                           *\n' );
  fprintf ( 1, '*  ERRORS:                                  *\n' );
  fprintf ( 1, '*    L2 error =          %14g     *\n', el2 );
  fprintf ( 1, '*    H1-seminorm error = %14g     *\n', eh1 );
  fprintf ( 1, '*                                           *\n' );
  fprintf ( 1, '*********************************************\n' );
```

```
  end
%
%  WRITE the data to files.
%
  node_filename = 'rectangle_nodes.txt';

  r8mat_write ( node_filename, 2, node_num, node_xy );

  fprintf ( 1, '\n' );
  fprintf ( 1, '  Wrote the node file "%s"\n', node_filename );

  element_filename = 'rectangle_elements.txt';

  i4mat_write ( element_filename, element_order, element_num,
element_node );

  fprintf ( 1, '  Wrote the element file "%s"\n', element_filename );

  value_filename = 'rectangle_solution.txt';

  r8mat_write ( value_filename, 1, node_num, c' );
  NumericalSolution=c;

  fprintf ( 1, '  Wrote the solution value file "%s"\n', value_filename );
%
%  Terminate.
%
  fprintf ( 1, '\n' );
  fprintf ( 1, 'FEM2D_POISSON_RECTANGLE_LINEAR:\n' );
  fprintf ( 1, '  Normal end of execution.\n' );
  fprintf ( 1, '\n' );
  timestamp ( );

  return
end
function [ u, dudx, dudy ] = exact ( x, y )

%**********************************************************************
*********80
%
%% exact() calculates the exact solution and its first derivatives.
%
%  Discussion:
```

```
%
%    The function specified here depends on the problem being
%    solved.  The user must be sure to change both EXACT and RHS
%    or the program will have inconsistent data.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    28 November 2008
%
%  Author:
%
%    John Burkardt
%
%  Input:
%
%    real X, Y, the coordinates of a point
%    in the region, at which the exact solution is to be evaluated.
%
%  Output:
%
%    real U, DUDX, DUDY, the value of
%    the exact solution U and its derivatives dUdX
%    and dUdY at the point (X,Y).
%
%   u    =      sin ( pi * x ) * sin ( pi * y ) + x;
%   dudx = pi * cos ( pi * x ) * sin ( pi * y ) + 1.0;
%   dudy = pi * sin ( pi * x ) * cos ( pi * y );
  u    =      sin ( 2 * pi * x ) * sin ( 2 * pi * y ) + x^2;
  dudx = 2 * pi * cos ( 2 * pi * x ) * sin ( 2 * pi * y ) + 2.0 * x;
  dudy = 2 * pi * sin ( 2 * pi * x ) * cos ( 2 * pi * y );%修改点

  return
end
function i4mat_write ( output_filename, m, n, table )

%***********************************************************************
*********80
%
%% i4mat_write() writes an I4MAT file.
%
```

```
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    09 August 2009
%
%  Author:
%
%    John Burkardt
%
%  Input:
%
%    string OUTPUT_FILENAME, the output filename.
%
%    integer M, the spatial dimension.
%
%    integer N, the number of points.
%
%    integer TABLE(M,N), the points.
%

%
%  Open the file.
%
  output_unit = fopen ( output_filename, 'wt' );

  if ( output_unit < 0 )
    fprintf ( 1, '\n' );
    fprintf ( 1, 'I4MAT_WRITE - Error!\n' );
    fprintf ( 1, '  Could not open the output file.\n' );
    error ( 'I4MAT_WRITE - Error!' );
  end
%
%  Write the data.
%
  for j = 1 : n
    for i = 1 : m
      fprintf ( output_unit, '  %12d', round ( table(i,j) ) );
    end
    fprintf ( output_unit, '\n' );
  end
%
```

```
%  Close the file.
%
  fclose ( output_unit );

  return
end
function r8mat_write ( output_filename, m, n, table )

%*********************************************************************
*********80
%
%% r8mat_write() writes an R8MAT file.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    11 August 2009
%
%  Author:
%
%    John Burkardt
%
%  Input:
%
%    string OUTPUT_FILENAME, the output filename.
%
%    integer M, the spatial dimension.
%
%    integer N, the number of points.
%
%    real TABLE(M,N), the points.
%


%
%  Open the file.
%
  output_unit = fopen ( output_filename, 'wt' );

  if ( output_unit < 0 )
    fprintf ( 1, '\n' );
    fprintf ( 1, 'R8MAT_WRITE - Error!\n' );
```

```
    fprintf ( 1, '  Could not open the output file.\n' );
    error ( 'R8MAT_WRITE - Error!' );
  end
%
%  Write the data.
%
%  For smaller data files, and less precision, try:
%
%     fprintf ( output_unit, '  %14.6f', table(i,j) );
%
  for j = 1 : n
    for i = 1 : m
      fprintf ( output_unit, '  %24.16f', table(i,j) );
    end
    fprintf ( output_unit, '\n' );
  end
%
%  Close the file.
%
  fclose ( output_unit );

  return
end
function timestamp ( )

%*********************************************************************
*********80
%
%% timestamp() prints the current YMDHMS date as a timestamp.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    14 February 2003
%
%  Author:
%
%    John Burkardt
%
  t = now;
  c = datevec ( t );
```

```
  s = datestr ( c, 0 );
  fprintf ( 1, '%s\n', s );

  return
end
```