# 1 问题描述

已知在区域Ω= [0, 1] × [0, 1]上，二维椭圆方程的Dirichlet边值问题如下定义:

其中真实解。试完成以下题目:

(1) 请写出上述问题的变分形式，以及相应有限元的离散变分形式;

(2) 选取分片线形连续的有限元空间，依照(1)问中的变分形式计算上述问题的有限元解并与真实解比较. 你需要作出三张二维热力图(参考matplotlib中的imshow方法)，分别对应真实解，有限元解和绝对误差，以此来验证你的结果;

(3) 理论上(2)中有限元解L2范数的收敛阶为O(h2)，试验证之.

# 2 解答

## 2.1 变分形式

考虑空间V：

令

有

同时乘上v得，

同时积分得，

根据格林公式，

由于在空间V中，

式可以改写为，

问题转化为变分形式：

其中，

## 2.2 有限元的离散变分形式

定义Vh为V的一个n维子空间，

求近似解，

任取一个单元

其中，

利用上式将积分式离散化，

整理顺序，得到刚度矩阵的表达式：

## 2.3 有限元代码实现

### 2.3.1 单元划分

对区域Ω= [0, 1] × [0, 1]，采用线性三角形单元，将区域Ω划分成2Nx×Ny的三角形单元。每个三角形单元由三个节点组成，每个节点有单元内的局部坐标和整体坐标，分别储存在points类和triangleelement类中。

|  |
| --- |
| import numpy as np  #将区域Ω的x边划分成Nx份，y边划分成Ny份  Nx = 10  Ny = 10  class Points:  x\_space = np.linspace(0, 1, Nx + 1, endpoint=True)  y\_space = np.linspace(0, 1, Ny + 1, endpoint=True)  yy, xx = np.meshgrid(y\_space, x\_space)  xy = np.stack([xx, yy], axis=2)  @staticmethod  def get(i):  return Points.xy[i[0], i[1]]  @staticmethod  def global\_idx(i):  return i[0] \* (Nx + 1) + i[1]  class TriangleElement:  #存储点的全局指标，点的全局指标转换成坐标由Points全权负责  def \_\_init\_\_(self, i, j, k):  self.i = i  self.j = j  self.k = k  @property  def p1(self):  return Points.get(self.i)  @property  def p2(self):  return Points.get(self.j)  @property  def p3(self):  return Points.get(self.k)  @property  def area(self):  #均匀网格，可以简单地计算每个三角形单元的面积  return (self.p2[0] \* self.p3[1] - self.p2[1] \* self.p3[0] +  self.p3[0] \* self.p1[1] - self.p3[1] \* self.p1[0] +  self.p1[0] \* self.p2[1] - self.p1[1] \* self.p2[0]) / 2 |

### 2.3.2 刚度矩阵

计算每个单元的局部刚度矩阵，并将局部刚度送到总刚度矩阵。其中是有显式表达式，而没有，需要采用数值积分的办法。

|  |
| --- |
| def add\_local\_stiffness(self, global\_stiffness: np.array): #对这一项的刚度进行计算  gi = Points.global\_idx(self.i)  gj = Points.global\_idx(self.j)  gk = Points.global\_idx(self.k)  sub\_idx = [gi, gj, gk]  i, j, k = Points.get(self.i), Points.get(self.j), Points.get(self.k)  local\_stiff = np.sum(np.array(  [[(j - k) \*\* 2, (i - k) \* (k - j), (i - j) \* (j - k)],  [(i - k) \* (k - j), (i - k) \*\* 2, (k - i) \* (i - j)],  [(i - j) \* (j - k), (k - i) \* (i - j), (i - j) \*\* 2]]), axis=2) / self.area / 4  global\_stiffness[np.ix\_(sub\_idx, sub\_idx)] += local\_stiff #送总刚度矩阵  def add\_local\_stiffness2(self, global\_stiffness: np.array): #对u这一项的刚度进行计算  gi = Points.global\_idx(self.i)  gj = Points.global\_idx(self.j)  gk = Points.global\_idx(self.k)  sub\_idx = [gi, gj, gk]  i, j, k = Points.get(self.i), Points.get(self.j), Points.get(self.k)  def compose\_fun2(local\_idx1,local\_idx2):  if local\_idx1 == 0:  li, lj, lk = i, j, k  if local\_idx1 == 1:  lk, li, lj = i, j, k  if local\_idx1 == 2:  lj, lk, li = i, j, k  if local\_idx2 == 0:  mi, mj, mk = i, j, k  if local\_idx2 == 1:  mk, mi, mj = i, j, k  if local\_idx2 == 2:  mj, mk, mi = i, j, k  def \_fun2(x, y):  \_r2= ((lj[1]-lk[1])\*(mj[1]-mk[1])\*x\*x + (lj[1]-lk[1])\*(mk[0]-mj[0])\*x\*y + (lj[1]-lk[1])\*(mj[0]\*mk[1]-mk[0]\*mj[1])\* x +  (lk[0]-lj[0])\*(mj[1]-mk[1])\*y\*x + (lk[0]-lj[0])\*(mk[0]-mj[0])\*y\*y + (lk[0]-lj[0])\*(mj[0]\*mk[1]-mk[0]\*mj[1])\* y +  (lj[0]\*lk[1]-lk[0]\*lj[1])\*(mj[1]-mk[1])\*x + (lj[0]\*lk[1]-lk[0]\*lj[1])\*(mk[0]-mj[0])\*y + (lj[0]\*lk[1]-lk[0]\*lj[1])\*(mj[0]\*mk[1]-mk[0]\*mj[1]) )/ 4 / self.area / self.area  return \_r2  return \_fun2    for \_i in range(0,3):  for \_j in range(0,3):    local\_stiff2 = np.zeros((3, 3))    local\_stiff2[np.ix\_([\_i], [\_j])] += self.quadrature(compose\_fun2(\_i,\_j))  def quadrature(self, q: callable):    return (q((self.p1[0] + self.p2[0]) / 2, (self.p1[1] + self.p2[1]) / 2) +  q((self.p2[0] + self.p3[0]) / 2, (self.p2[1] + self.p3[1]) / 2) +  q((self.p3[0] + self.p1[0]) / 2, (self.p3[1] + self.p1[1]) / 2)) / 3 \* self.area  global\_stiffness[np.ix\_(sub\_idx, sub\_idx)] += local\_stiff2 #送总刚度矩阵 |

### 2.3.3 荷载向量

荷载向量也不具有显式表达式，需要采用数值积分计算。

|  |
| --- |
| def add\_local\_load(self, f: callable, global\_load: np.array):  gi = Points.global\_idx(self.i)  gj = Points.global\_idx(self.j)  gk = Points.global\_idx(self.k)  i, j, k = Points.get(self.i), Points.get(self.j), Points.get(self.k)  def compose\_fun(local\_idx):  if local\_idx == 0:  li, lj, lk = i, j, k  if local\_idx == 1:  lk, li, lj = i, j, k  if local\_idx == 2:  lj, lk, li = i, j, k  def \_fun(x, y):  \_r = (x \* lj[1] - lj[0] \* y + lj[0] \* lk[1] + lk[0] \* y - x \* lk[1] - lk[0] \* lj[1]) / 2 / self.area  return \_r \* f(x, y)  return \_fun  global\_load[gi] += self.quadrature(compose\_fun(0))  global\_load[gj] += self.quadrature(compose\_fun(1))  global\_load[gk] += self.quadrature(compose\_fun(2))  return self.quadrature(compose\_fun(0)), self.quadrature(compose\_fun(1)), self.quadrature(compose\_fun(2))  def quadrature(self, q: callable):    return (q((self.p1[0] + self.p2[0]) / 2, (self.p1[1] + self.p2[1]) / 2) +  q((self.p2[0] + self.p3[0]) / 2, (self.p2[1] + self.p3[1]) / 2) +  q((self.p3[0] + self.p1[0]) / 2, (self.p3[1] + self.p1[1]) / 2)) / 3 \* self.area |

### 2.3.4 组装单元，计算数值解

调用上述类和函数，将区域Ω划分单元，组装局部刚度矩阵和荷载向量，并送至总刚度矩阵和总荷载向量中。根据Dirichlet边值的第一类边值条件，划去已知边界条件的行和列。最后，计算线性方程组得到二维椭圆方程的Dirichlet边值问题的数值解。

|  |
| --- |
| import math  pai=math.pi  def q(xx, yy):  return 2 \*(pai\*\*2+1)\*math.sin(pai\*xx)\*math.sin(pai\*yy)  element\_list = []  for ix in range(Nx):  for iy in range(Ny):  element\_list.append(TriangleElement((ix, iy), (ix + 1, iy), (ix, iy + 1)))  element\_list.append(TriangleElement((ix, iy + 1), (ix + 1, iy), (ix + 1, iy + 1)))  global\_stiffness = np.zeros(((Nx + 1) \* (Ny + 1), (Nx + 1) \* (Ny + 1)))  [te.add\_local\_stiffness(global\_stiffness) for te in element\_list]  print (global\_stiffness)  [te.add\_local\_stiffness2(global\_stiffness) for te in element\_list]  print (global\_stiffness)  interior\_idx = (Nx + 1) \* np.arange(1, Nx).reshape(Nx - 1, 1) + np.arange(1, Ny).reshape(1, Ny - 1)  interior\_idx = interior\_idx.ravel()  global\_stiffness = global\_stiffness[np.ix\_(interior\_idx, interior\_idx)]  global\_load = np.zeros((Nx + 1) \* (Ny + 1))  [te.add\_local\_load(q, global\_load) for te in element\_list]  global\_load = global\_load[interior\_idx]  res = np.linalg.solve(global\_stiffness, global\_load)  res = res.reshape(Nx - 1, Ny - 1) |

## 2.4 数值解和精确解的对比

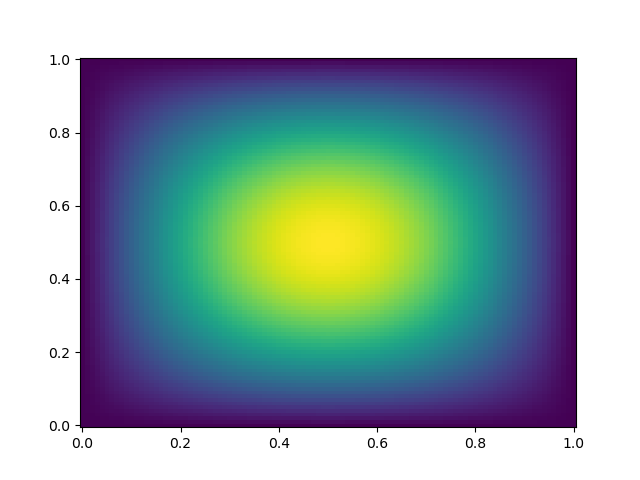


图1 有限元解（100×100）

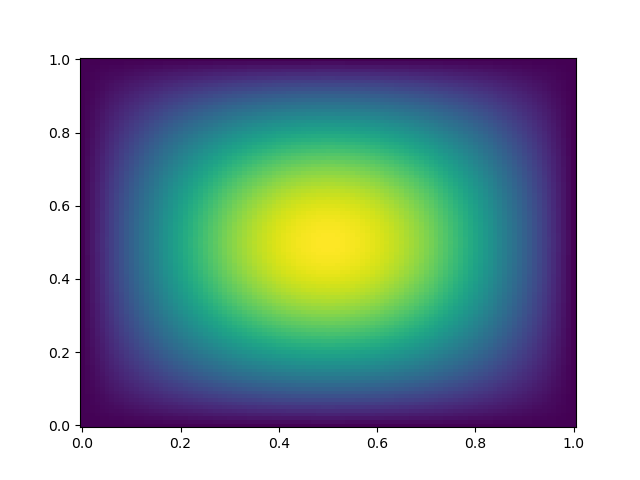


图2 精确解（100×100）

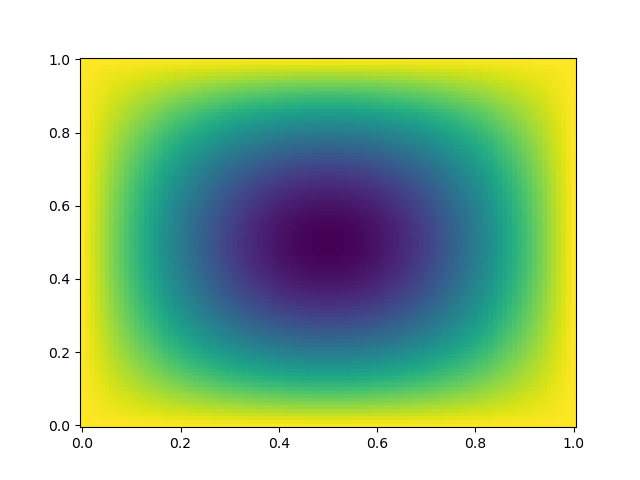


图3 有限元解与精确解的误差（100×100）

## 2.5 有限元解L2范数的收敛阶

采用不同尺寸的网格计算，可以得到误差函数随网格尺寸h的变化关系，如分别采用(Δx,Δt), (Δx,Δt), (Δx,Δt)的步长进行计算，得到,,。

由于，

那么，

因此收敛精度p可以采用下式进行估计

计算所得的误差L2范数列为表1，

表1 误差收敛L2范数

|  |  |  |
| --- | --- | --- |
| 单元划分 | L2 范数 | 收敛阶 |
| 2 | 0.0908 | 1.8492 |
| 4 | 0.0252 | 1.9549 |
| 8 | 0.0065 | 2.0223 |
| 16 | 0.0016 | 1.9707 |
| 32 | 4.0821e-04 | 1.9993 |
| 64 | 1.0210e-04 | - |

因此，(2)中有限元解L2范数的收敛阶为O(h2)。