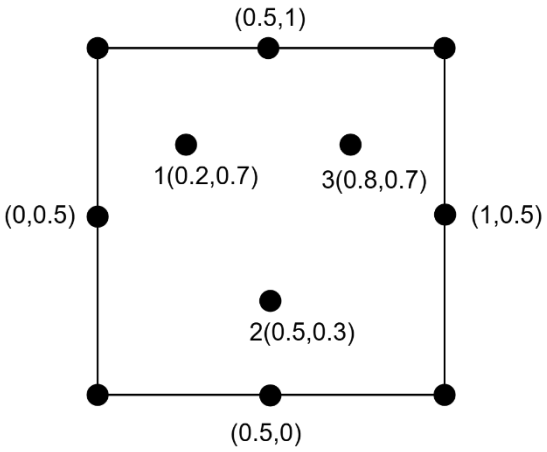
1. 考虑如图所示的正方形区域($0\leqslant x\leqslant 1, 0\leqslant y\leqslant 1$)，：

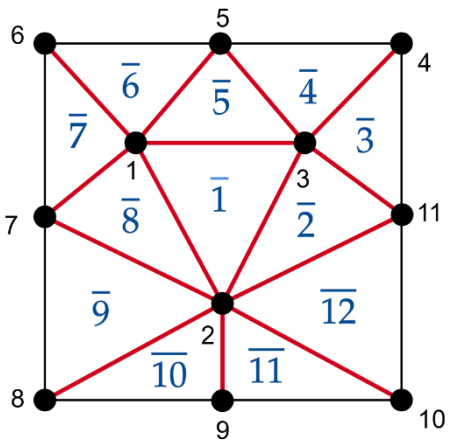


（1）给出一种合理的三角形剖分

（2）列方程并求解$\varphi(x,y)$在点1,2,3 处的值，其中$\varphi(x,y)$满足如下边界条件的拉普拉斯方程

$$\begin{cases} \nabla^2\varphi(x,y)=0 \\ \varphi(x,0)=\varphi(x,1)=4\left(x-\dfrac{1}{2}\right)^2,\ \varphi(0,y)=\varphi(1,y)=1 \end{cases}$$

（1）



（2）①依据编号存储每个点的坐标；

②记录每个三角形的顶点信息，这里我选择从最小的顶点编号开始逆时针顺序存储：如图中第 3 个三角形顶点信息为[3, 11, 4]，分别对应三角形内第 1, 2, 3 点；

③遍历每个三角形，计算b和d

$$\begin{cases} b_{i1}=y_{i2}-y_{i3} \\ b_{i2}=y_{i3}-y_{i1} \\ b_{i3}=y_{i1}-y_{i2} \end{cases},\quad \begin{cases} d_{i1}=x_{i3}-x_{i2} \\ d_{i2}=x_{i1}-x_{i3} \\ d_{i3}=x_{i2}-x_{i1} \end{cases}$$

其中i是三角形编号，1,2,3 是三角形内部顶点编号

④同时计算三角形面积$\Delta_i=\frac{1}{2}(b_{i1}d_{i2}-b_{i2}d_{i1})$；

⑤计算三角形内矩阵元

$$k^{T_i}=\frac{1}{4\Delta_i}\begin{bmatrix} b_{i1}^2+d_{i1}^2 & b_{i1}b_{i2}+d_{i1}d_{i2} & b_{i1}b_{i3}+d_{i1}d_{i3} \\ b_{i2}b_{i1}+d_{i2}d_{i1} & b_{i2}^2+d_{i2}^2 & b_{i2}b_{i3}+d_{i2}d_{i3} \\ d_{i3}d_{i1}+b_{i3}b_{i1} & b_{i3}b_{i2}+d_{i3}d_{i2} & b_{i3}^2+d_{i3}^2 \end{bmatrix}$$

并将每个矩阵元素根据顶点的全局编号加到大矩阵 K 中，K 应为11×11的矩阵；

⑥处理边界，考虑线性方程组

$$\begin{bmatrix}(K_{11}) & (K_{12}) \\ (K_{21}) & (K_{22})\end{bmatrix}\begin{bmatrix}(\Phi_1) \\ (\Phi_2)\end{bmatrix}=0$$

其中Φ_1待求的三个点的值，应为3×1矩阵，Φ_2是边界点的值，均已给出，则可用这些已知的值Φ_0代替，不难得出

$$\Phi_1=(K_{11})^{-1}[-K_{12}\Phi_0]$$

依据此方法求得的三个点的值应为（我们同时给出利用内置函数求解的值作为参考）

	(0.2,0.7)	(0.5,0.3)	(0.8,0.7)
有限元	0.7174	0.4986	0.7174
内置函数	0.7067	0.5070	0.7067

我们也给出K_{11}和$-K_{12}\Phi_0$的值供大家自己检验

$$K_{11}=\begin{bmatrix} 4.1399 & -0.5893 & -0.1458 \\ -0.5839 & 3.9881 & -0.5893 \\ -0.1458 & -0.5893 & 4.1399 \end{bmatrix}$$

$$-K_{12}\Phi_0=\begin{bmatrix}2.5714 \\ 1.1429 \\ 2.5714\end{bmatrix}$$

代码我们作为参考放在最后的附录中。

附录：

MATLAB 代码：

```
clear;

% 格点编号
Sites = zeros(11, 2);
Sites(1, :) = [0.2, 0.7];
Sites(2, :) = [0.5, 0.3];
Sites(3, :) = [0.8, 0.7];
Sites(4, :) = [1, 1];
Sites(5, :) = [0.5, 1];
Sites(6, :) = [0, 1];
Sites(7, :) = [0, 0.5];
Sites(8, :) = [0, 0];
Sites(9, :) = [0.5, 0];
Sites(10, :) = [1, 0];
Sites(11, :) = [1, 0.5];

% 三角形编号
Tri = zeros(12, 3);
Tri(1, :) = [1, 2, 3];
Tri(2, :) = [2, 11, 3];
Tri(3, :) = [3, 11, 4];
Tri(4, :) = [3, 4, 5];
Tri(5, :) = [1, 3, 5];
Tri(6, :) = [1, 5, 6];
Tri(7, :) = [1, 6, 7];
Tri(8, :) = [1, 7, 2];
Tri(9, :) = [2, 7, 8];
Tri(10, :) = [2, 8, 9];
Tri(11, :) = [2, 9, 10];
Tri(12, :) = [2, 10, 11];

% 边界值
Phi0 = [1; 0; 1; 1; 1; 0; 1; 1];

% K 矩阵
K = zeros(11);
for i = 1 : 12
    % 取出第 i 个三角形的顶点编号
    T = Tri(i, :);

    % 取出各顶点的坐标
```

```
    site1 = Sites(T(1), :);
    site2 = Sites(T(2), :);
    site3 = Sites(T(3), :);

    b1 = site2(2) - site3(2);
    b2 = site3(2) - site1(2);
    b3 = site1(2) - site2(2);

    d1 = site3(1) - site2(1);
    d2 = site1(1) - site3(1);
    d3 = site2(1) - site1(1);

    % 三角形面积
    S = 1/2 * (b1 * d2 - b2 * d1);

    K(T(1), T(1)) = K(T(1), T(1)) + (b1^2 + d1^2)/(4*S);
    K(T(2), T(2)) = K(T(2), T(2)) + (b2^2 + d2^2)/(4*S);
    K(T(3), T(3)) = K(T(3), T(3)) + (b3^2 + d3^2)/(4*S);
    K(T(1), T(2)) = K(T(1), T(2)) + (b1 * b2 + d1 * d2)/(4*S);
    K(T(1), T(3)) = K(T(1), T(3)) + (b1 * b3 + d1 * d3)/(4*S);
    K(T(2), T(3)) = K(T(2), T(3)) + (b2 * b3 + d2 * d3)/(4*S);
    K(T(2), T(1)) = K(T(2), T(1)) + (b2 * b1 + d2 * d1)/(4*S);
    K(T(3), T(1)) = K(T(3), T(1)) + (b3 * b1 + d3 * d1)/(4*S);
    K(T(3), T(2)) = K(T(3), T(2)) + (b3 * b2 + d3 * d2)/(4*S);

end

% 处理边界
K11 = K(1:3, 1:3);
K12 = K(1:3, 4:11);
Phi1 = K11 \ (- K12 * Phi0);
disp(Phi1)
```

Python 代码：

```
import numpy as np

Dot = np.array([[0.2, 0.7], [0.5, 0.3], [0.8, 0.7], [0, 1], [0, 0.5],
[0, 0], [0.5, 0], [1, 0], [1, 0.5], [1, 1], [0.5, 1]])
# 所有的三角形
TN = np.array([[1, 4, 5], [5, 6, 2], [6, 7, 2], [2, 7, 8], [2, 8, 9],
[3, 9, 10], [3, 10, 11], [4, 1, 11], [1, 5, 2], [1, 2, 3], [3, 2, 9],
[1, 3, 11]])

# 边界 8 个点的赋值
```

```
Pha0 = np.array([1, 1, 1, 0, 1, 1, 1, 0])

# 与顶点1,2,3相关的三角形
TN1 = np.array([[1, 1, 4, 5], [9, 1, 5, 2], [10, 1, 2, 3], [12, 1, 3,
11], [8, 1, 11, 4]])  # 第一个数字对应Ti，第i个三角形
TN2 = np.array([[9, 2, 1, 5], [2, 2, 5, 6], [3, 2, 6, 7], [4, 2, 7, 8],
[5, 2, 8, 9], [11, 2, 9, 3], [10, 2, 3, 1]])
TN3 = np.array([[10, 3, 1, 2], [11, 3, 2, 9], [6, 3, 9, 10], [7, 3, 10,
11], [12, 3, 11, 1]])

# 求解每个元素对应该面积
def Sfunc(N):
    n = len(N)
    S = []
    for i in range(n):
        dot1 = N[i, 0] - 1
        dot2 = N[i, 1] - 1
        dot3 = N[i, 2] - 1
        L = np.ones((3,3))
        L[0, 1] = Dot[dot1, 0]
        L[0, 2] = Dot[dot1, 1]
        L[1, 1] = Dot[dot2, 0]
        L[1, 2] = Dot[dot2, 1]
        L[2, 1] = Dot[dot3, 0]
        L[2, 2] = Dot[dot3, 1]
        det = np.linalg.det(L)
        S.append(2*det)
    return S  # 对应4倍面积
SN = np.array(Sfunc(TN))

def Kll_fun(N):
    n = len(N)
    b = np.zeros(n)
    d = np.zeros(n)
    K = 0
    for i in range(n):
        Ti = N[i, 0] - 1
        dot1 = N[i, 2] - 1
        dot2 = N[i, 3] - 1
        b[i] = Dot[dot1, 1] - Dot[dot2, 1]
        d[i] = Dot[dot2, 0] - Dot[dot1, 0]
        K += (b[i]**2 + d[i]**2)/ SN[Ti]
    return K
```

```
def BD_TN(N):
    n = len(N)
    B = np.zeros((n,3))
    D = np.zeros((n,3))
    for i in range(n):
        dot1 = N[i, 0] - 1
        dot2 = N[i, 1] - 1
        dot3 = N[i, 2] - 1
        # 计算 B1
        b1 = Dot[dot2, 1] - Dot[dot3, 1]
        b2 = Dot[dot3, 1] - Dot[dot1, 1]
        b3 = Dot[dot1, 1] - Dot[dot2, 1]
        B[i, 0] = b1
        B[i, 1] = b2
        B[i, 2] = b3
        # 计算 D1
        d1 = Dot[dot3, 0] - Dot[dot2, 0]
        d2 = Dot[dot1, 0] - Dot[dot3, 0]
        d3 = Dot[dot2, 0] - Dot[dot1, 0]
        D[i, 0] = d1
        D[i, 1] = d2
        D[i, 2] = d3
    return B, D
B_TN, D_TN = BD_TN(TN)


def Klm_fun(l, m):
    indices = [index for index, row in enumerate(TN) if l in row and m
in row]
    N_T = np.array(indices)
    n = len(N_T)
    sum = 0
    for i in range(n):
        Ti = N_T[i]
        for j in range(3):
            if TN[Ti, j] == l:
                bl = B_TN[Ti, j]
                dl = D_TN[Ti, j]
                for k in range(3):
                    if TN[Ti, k] == m:
                        bm = B_TN[Ti, k]
                        dm = D_TN[Ti, k]
```

```
        sum += (bl*bm + dl*dm) / SN[Ti]
    return sum

#print(Klm_fun(1,2))

K = np.zeros((3, 11))
K[0, 0] = Kll_fun(TN1)
K[1, 1] = Kll_fun(TN2)
K[2, 2] = Kll_fun(TN3)
for i in range(3):
    for j in range(11):
        if j != i:
            K[i, j] = Klm_fun(i + 1, j + 1)

print(K)

K11 = K[:, :3]  # 前 3 列
K12 = K[:, 3:]  # 后 8 列

P = np.dot(K12, Pha0)
Pha = np.linalg.solve(K11, -P)
print(Pha)
```