第七章作业 王晨曦

Created	@March 31, 2022 10:46 AM

姓名：王晨曦

班级：计算机96

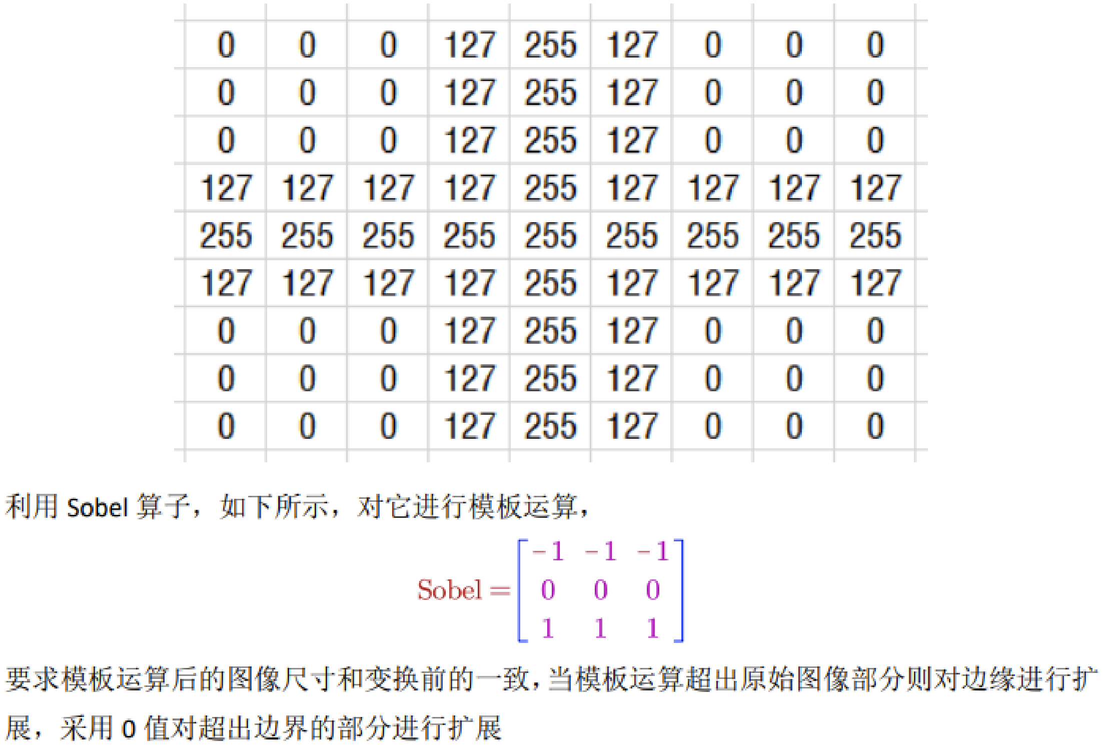
学号：2196123413

得分：

声明：

由于考虑到手算过于冗余，故本人编写程序对数据进行统一处理，本作业中所有程序均为本人根据上课所学理论知识自己编写，无任何上网copy行为

- 给定一幅图像如下所示：



0	0	0	127	255	127	0	0	0
0	0	0	127	255	127	0	0	0
0	0	0	127	255	127	0	0	0
127	127	127	127	255	127	127	127	127
255	255	255	255	255	255	255	255	255
127	127	127	127	255	127	127	127	127
0	0	0	127	255	127	0	0	0
0	0	0	127	255	127	0	0	0
0	0	0	127	255	127	0	0	0

利用 Sobel 算子，如下所示，对它进行模板运算，

$$\text{Sobel} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

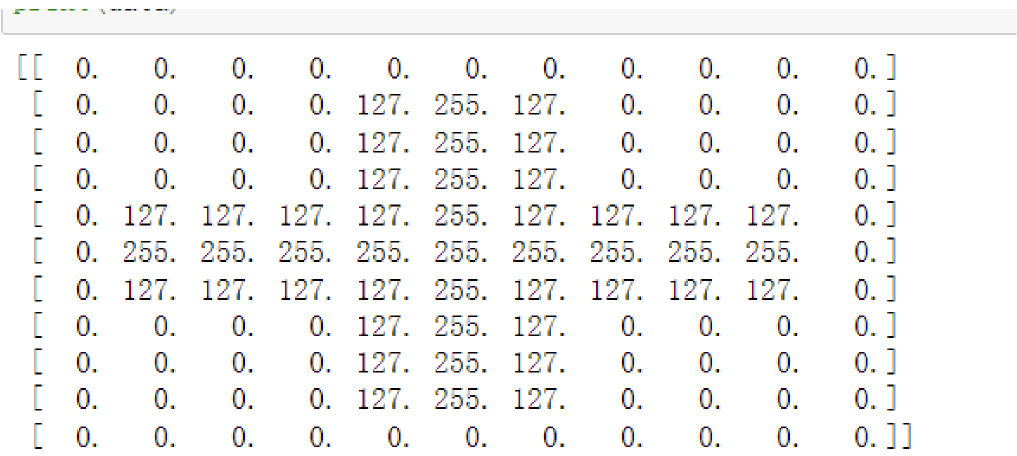
要求模板运算后的图像尺寸和变换前的一致，当模板运算超出原始图像部分则对边缘进行扩展，采用 0 值对超出边界的部分进行扩展

1. **显然该Sobel是在计算y方向的梯度**
2. **求取x方向梯度的模板：**

-1	0	1
-1	0	1
-1	0	1

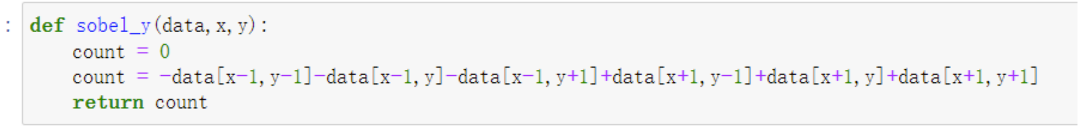
3. **求x和y方向的梯度值**

- 导入图像矩阵，并将边缘填充为0

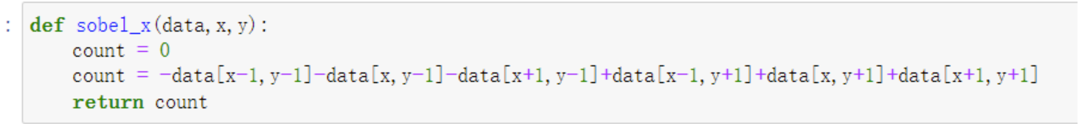


```
print(data)

[[  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.]
 [  0.    0.    0.    0.  127.  255.  127.    0.    0.    0.    0.]
 [  0.    0.    0.    0.  127.  255.  127.    0.    0.    0.    0.]
 [  0.    0.    0.    0.  127.  255.  127.    0.    0.    0.    0.]
 [  0.  127.  127.  127.  127.  255.  127.  127.  127.  127.    0.]
 [  0.  255.  255.  255.  255.  255.  255.  255.  255.  255.    0.]
 [  0.  127.  127.  127.  127.  255.  127.  127.  127.  127.    0.]
 [  0.    0.    0.    0.  127.  255.  127.    0.    0.    0.    0.]
 [  0.    0.    0.    0.  127.  255.  127.    0.    0.    0.    0.]
 [  0.    0.    0.    0.  127.  255.  127.    0.    0.    0.    0.]
 [  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.]]
```

- 分别将求x和y方向导数的模板写成函数sobel_x和sobel_y：

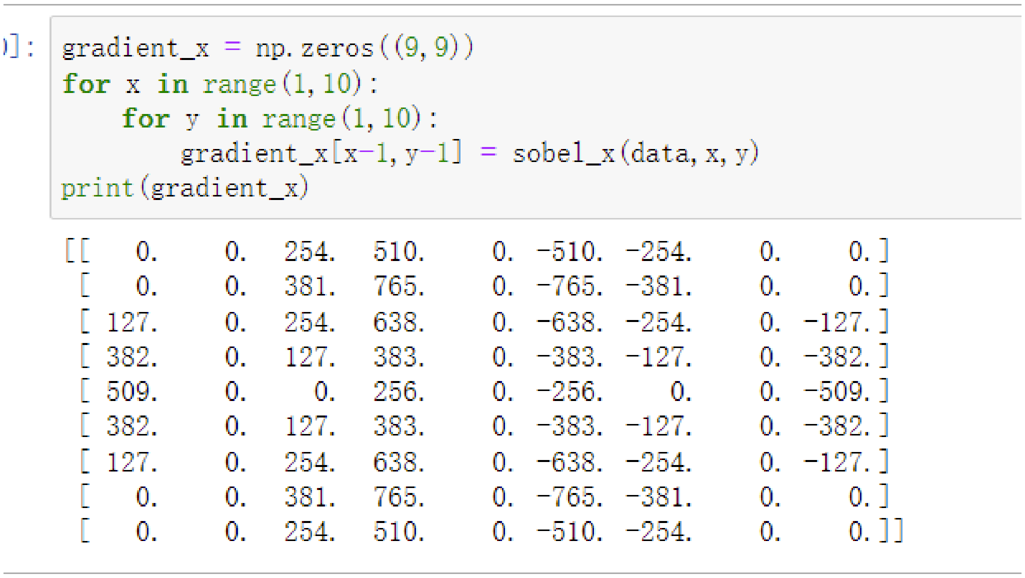


```
def sobel_y(data, x, y):
    count = 0
    count = -data[x-1, y-1]-data[x-1, y]-data[x-1, y+1]+data[x+1, y-1]+data[x+1, y]+data[x+1, y+1]
    return count
```



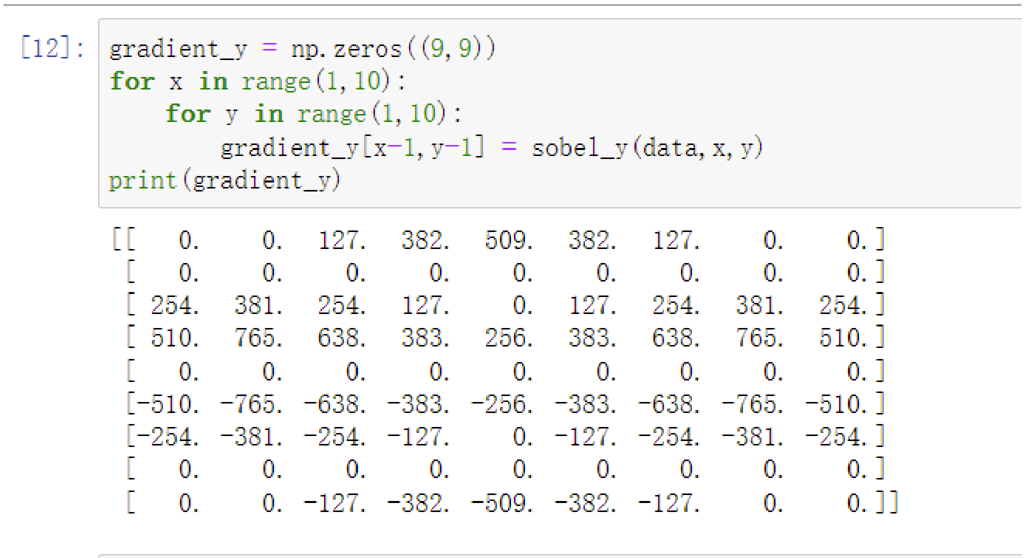
```
def sobel_x(data, x, y):
    count = 0
    count = -data[x-1, y-1]-data[x, y-1]-data[x+1, y-1]+data[x-1, y+1]+data[x, y+1]+data[x+1, y+1]
    return count
```

- 对图像矩阵使用sobel算子，并用gradient_x和gradient_y来分别存储所求得的梯度矩阵：
    - **x方向梯度值：**



```
]: gradient_x = np.zeros((9,9))
   for x in range(1,10):
       for y in range(1,10):
           gradient_x[x-1,y-1] = sobel_x(data,x,y)
   print(gradient_x)

   [[   0.    0.  254.  510.    0. -510. -254.    0.    0.]
    [   0.    0.  381.  765.    0. -765. -381.    0.    0.]
    [ 127.    0.  254.  638.    0. -638. -254.    0. -127.]
    [ 382.    0.  127.  383.    0. -383. -127.    0. -382.]
    [ 509.    0.    0.  256.    0. -256.    0.    0. -509.]
    [ 382.    0.  127.  383.    0. -383. -127.    0. -382.]
    [ 127.    0.  254.  638.    0. -638. -254.    0. -127.]
    [   0.    0.  381.  765.    0. -765. -381.    0.    0.]
    [   0.    0.  254.  510.    0. -510. -254.    0.    0.]]
```

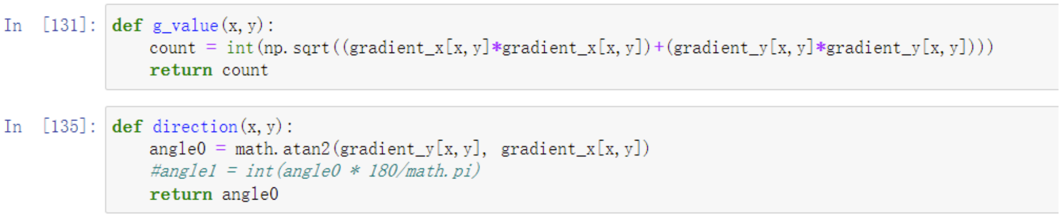
- **y方向梯度值：**



```
[12]: gradient_y = np.zeros((9,9))
      for x in range(1,10):
          for y in range(1,10):
              gradient_y[x-1,y-1] = sobel_y(data,x,y)
      print(gradient_y)

      [[   0.    0.  127.  382.  509.  382.  127.    0.    0.]
       [   0.    0.    0.    0.    0.    0.    0.    0.    0.]
       [ 254.  381.  254.  127.    0.  127.  254.  381.  254.]
       [ 510.  765.  638.  383.  256.  383.  638.  765.  510.]
       [   0.    0.    0.    0.    0.    0.    0.    0.    0.]
       [-510. -765. -638. -383. -256. -383. -638. -765. -510.]
       [-254. -381. -254. -127.    0. -127. -254. -381. -254.]
       [   0.    0.    0.    0.    0.    0.    0.    0.    0.]
       [   0.    0. -127. -382. -509. -382. -127.    0.    0.]]
```

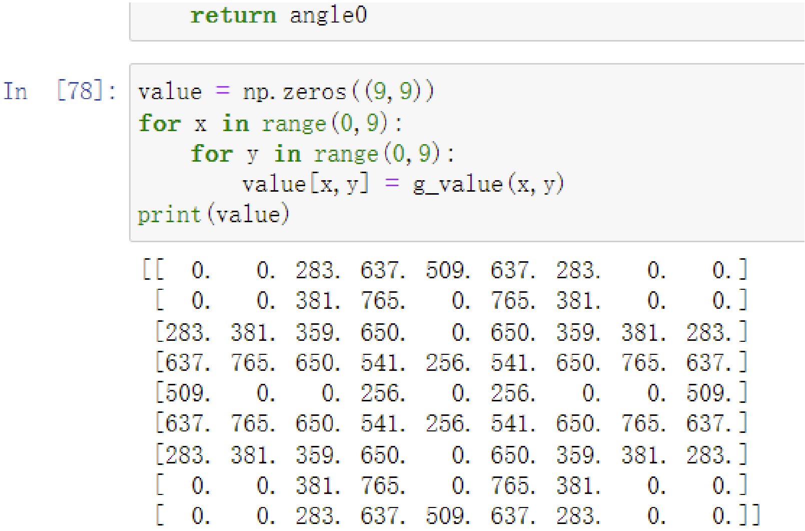
4. **求解梯度的幅度值和方向**

- 写出求解幅度值和方向的函数g_value和direction:



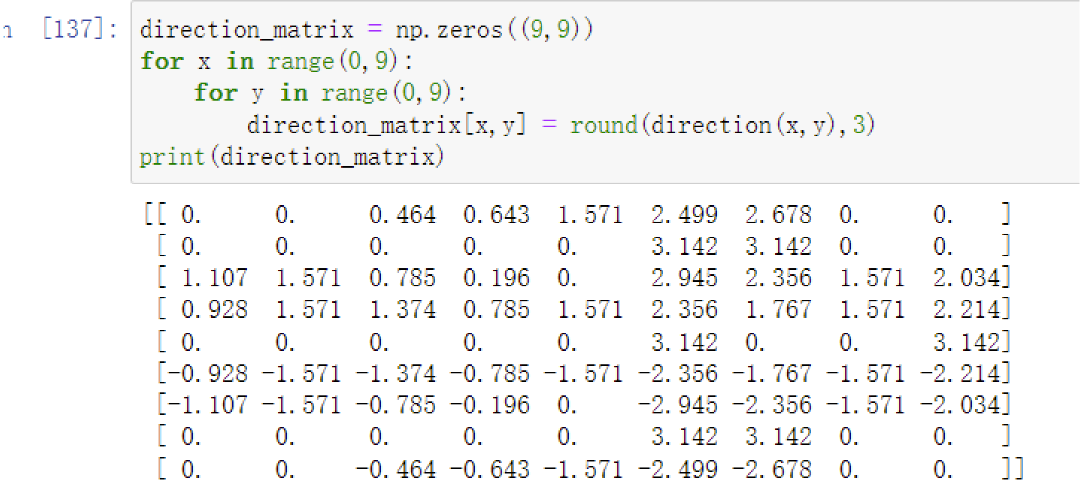
```
In  [131]: def g_value(x, y):
               count = int(np.sqrt((gradient_x[x, y]*gradient_x[x, y])+(gradient_y[x, y]*gradient_y[x, y])))
               return count

In  [135]: def direction(x, y):
               angle0 = math.atan2(gradient_y[x, y], gradient_x[x, y])
               #angle1 = int(angle0 * 180/math.pi)
               return angle0
```

- **梯度的幅度值：**



```
    return angle0

In  [78]: value = np.zeros((9,9))
          for x in range(0,9):
              for y in range(0,9):
                  value[x, y] = g_value(x, y)
          print(value)

[[  0.   0. 283. 637. 509. 637. 283.   0.   0.]
 [  0.   0. 381. 765.   0. 765. 381.   0.   0.]
 [283. 381. 359. 650.   0. 650. 359. 381. 283.]
 [637. 765. 650. 541. 256. 541. 650. 765. 637.]
 [509.   0.   0. 256.   0. 256.   0.   0. 509.]
 [637. 765. 650. 541. 256. 541. 650. 765. 637.]
 [283. 381. 359. 650.   0. 650. 359. 381. 283.]
 [  0.   0. 381. 765.   0. 765. 381.   0.   0.]
 [  0.   0. 283. 637. 509. 637. 283.   0.   0.]]
```

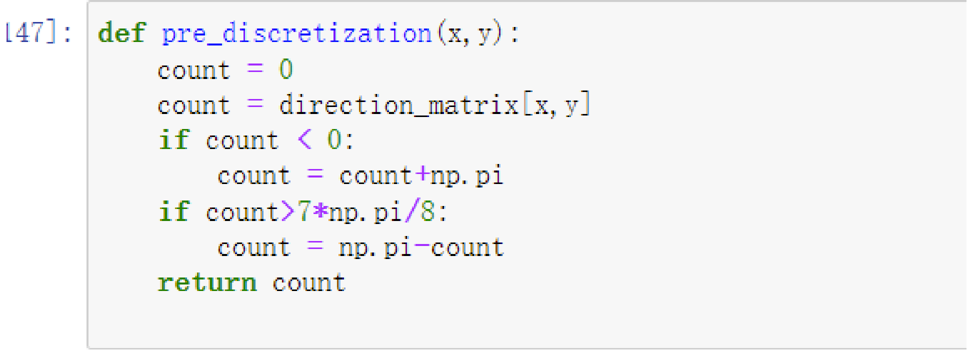
- **梯度的方向（弧度制）：**



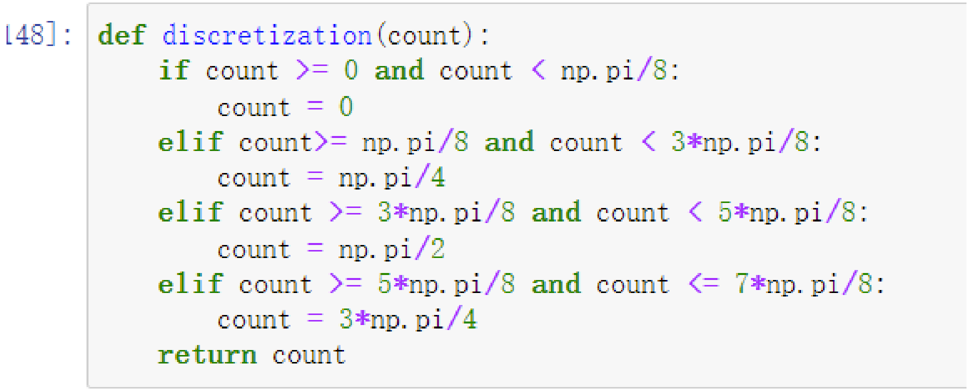
```
n  [137]: direction_matrix = np.zeros((9,9))
          for x in range(0,9):
              for y in range(0,9):
                  direction_matrix[x, y] = round(direction(x, y), 3)
          print(direction_matrix)

[[ 0.     0.     0.464  0.643  1.571  2.499  2.678  0.     0.   ]
 [ 0.     0.     0.     0.     0.     3.142  3.142  0.     0.   ]
 [ 1.107  1.571  0.785  0.196  0.     2.945  2.356  1.571  2.034]
 [ 0.928  1.571  1.374  0.785  1.571  2.356  1.767  1.571  2.214]
 [ 0.     0.     0.     0.     0.     3.142  0.     0.     3.142]
 [-0.928 -1.571 -1.374 -0.785 -1.571 -2.356 -1.767 -1.571 -2.214]
 [-1.107 -1.571 -0.785 -0.196  0.    -2.945 -2.356 -1.571 -2.034]
 [ 0.     0.     0.     0.     0.     3.142  3.142  0.     0.   ]
 [ 0.     0.    -0.464 -0.643 -1.571 -2.499 -2.678  0.     0.   ]]
```

5. 求离散化后的梯度方向

- 写出离散化的函数discretization：

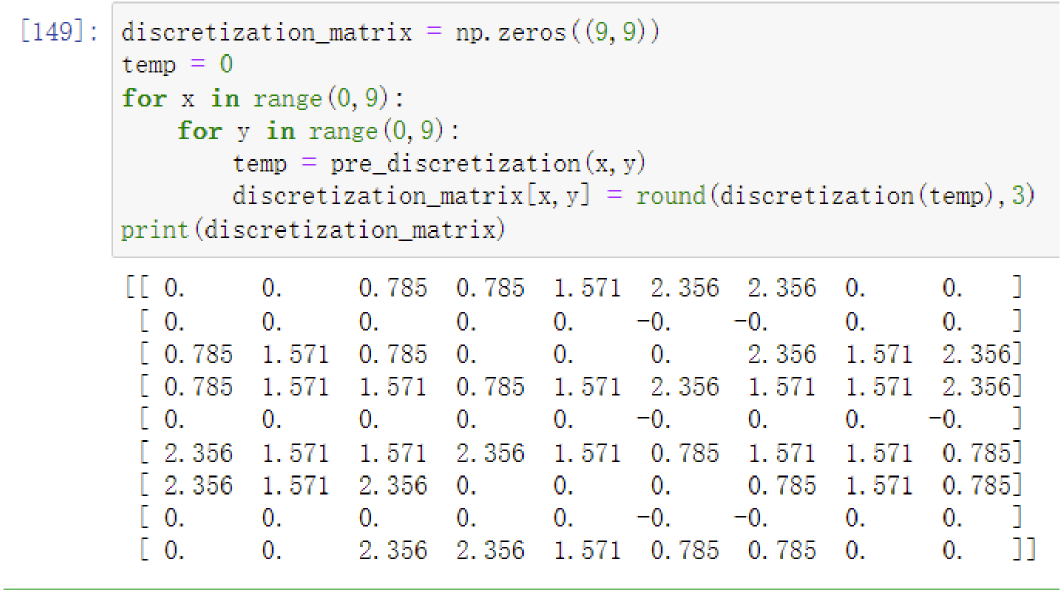


```
147]:  def pre_discretization(x,y):
           count = 0
           count = direction_matrix[x,y]
           if count < 0:
               count = count+np.pi
           if count>7*np.pi/8:
               count = np.pi-count
           return count
```



```
148]:  def discretization(count):
           if count >= 0 and count < np.pi/8:
               count = 0
           elif count>= np.pi/8 and count < 3*np.pi/8:
               count = np.pi/4
           elif count >= 3*np.pi/8 and count < 5*np.pi/8:
               count = np.pi/2
           elif count >= 5*np.pi/8 and count <= 7*np.pi/8:
               count = 3*np.pi/4
           return count
```

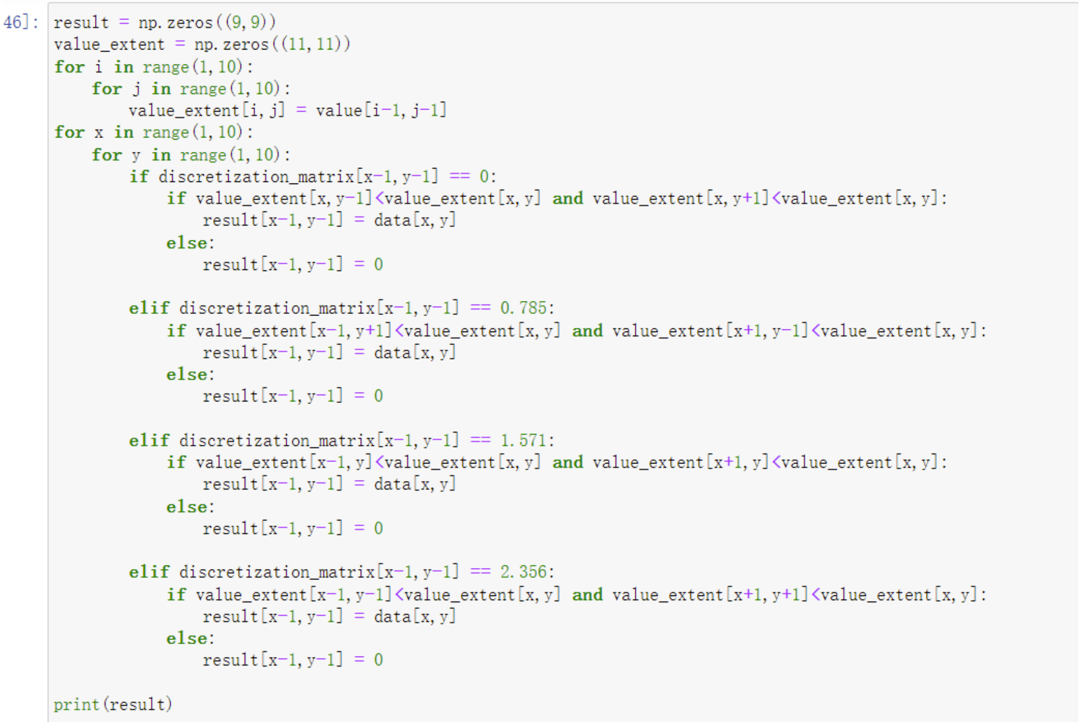
- **离散化后的梯度方向（弧度制）：**



```
[149]:  discretization_matrix = np.zeros((9,9))
        temp = 0
        for x in range(0,9):
            for y in range(0,9):
                temp = pre_discretization(x,y)
                discretization_matrix[x,y] = round(discretization(temp),3)
        print(discretization_matrix)

        [[ 0.     0.     0.785  0.785  1.571  2.356  2.356  0.     0.   ]
         [ 0.     0.     0.     0.     0.    -0.    -0.     0.     0.   ]
         [ 0.785  1.571  0.785  0.     0.     0.     2.356  1.571  2.356]
         [ 0.785  1.571  1.571  0.785  1.571  2.356  1.571  1.571  2.356]
         [ 0.     0.     0.     0.     0.    -0.     0.     0.    -0.   ]
         [ 2.356  1.571  1.571  2.356  1.571  0.785  1.571  1.571  0.785]
         [ 2.356  1.571  2.356  0.     0.     0.     0.785  1.571  0.785]
         [ 0.     0.     0.     0.     0.    -0.    -0.     0.     0.   ]
         [ 0.     0.     2.356  2.356  1.571  0.785  0.785  0.     0.   ]]
```

6. **非极大值抑制**

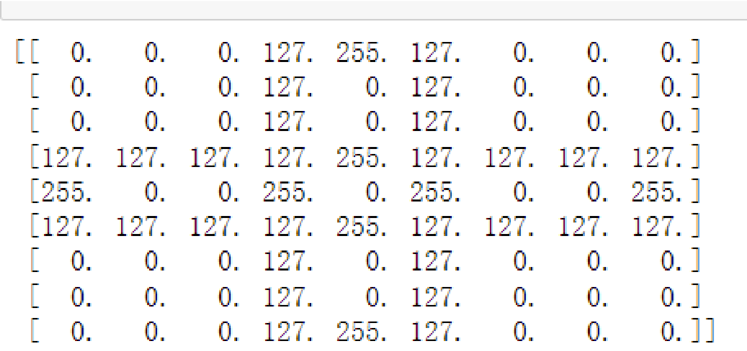
- 非极大值抑制算法：



```
result = np.zeros((9,9))
value_extent = np.zeros((11,11))
for i in range(1,10):
    for j in range(1,10):
        value_extent[i,j] = value[i-1,j-1]
for x in range(1,10):
    for y in range(1,10):
        if discretization_matrix[x-1,y-1] == 0:
            if value_extent[x,y-1]<value_extent[x,y] and value_extent[x,y+1]<value_extent[x,y]:
                result[x-1,y-1] = data[x,y]
            else:
                result[x-1,y-1] = 0

        elif discretization_matrix[x-1,y-1] == 0.785:
            if value_extent[x-1,y+1]<value_extent[x,y] and value_extent[x+1,y-1]<value_extent[x,y]:
                result[x-1,y-1] = data[x,y]
            else:
                result[x-1,y-1] = 0

        elif discretization_matrix[x-1,y-1] == 1.571:
            if value_extent[x-1,y]<value_extent[x,y] and value_extent[x+1,y]<value_extent[x,y]:
                result[x-1,y-1] = data[x,y]
            else:
                result[x-1,y-1] = 0

        elif discretization_matrix[x-1,y-1] == 2.356:
            if value_extent[x-1,y-1]<value_extent[x,y] and value_extent[x+1,y+1]<value_extent[x,y]:
                result[x-1,y-1] = data[x,y]
            else:
                result[x-1,y-1] = 0

print(result)
```

算法思想：遍历图中每个像素点，根据每个点离散后的梯度值，分别与其八邻域中的两个点进行梯度幅度值的比较，若该点大于其他两个点，保留该点像素值，否则置为0.

- **结果**



```
[[  0.   0.   0. 127. 255. 127.   0.   0.   0.]
 [  0.   0.   0. 127.   0. 127.   0.   0.   0.]
 [  0.   0.   0. 127.   0. 127.   0.   0.   0.]
 [127. 127. 127. 127. 255. 127. 127. 127. 127.]
 [255.   0.   0. 255.   0. 255.   0.   0. 255.]
 [127. 127. 127. 127. 255. 127. 127. 127. 127.]
 [  0.   0.   0. 127.   0. 127.   0.   0.   0.]
 [  0.   0.   0. 127.   0. 127.   0.   0.   0.]
 [  0.   0.   0. 127. 255. 127.   0.   0.   0.]]
```

显而易见，这是原图像的边缘。