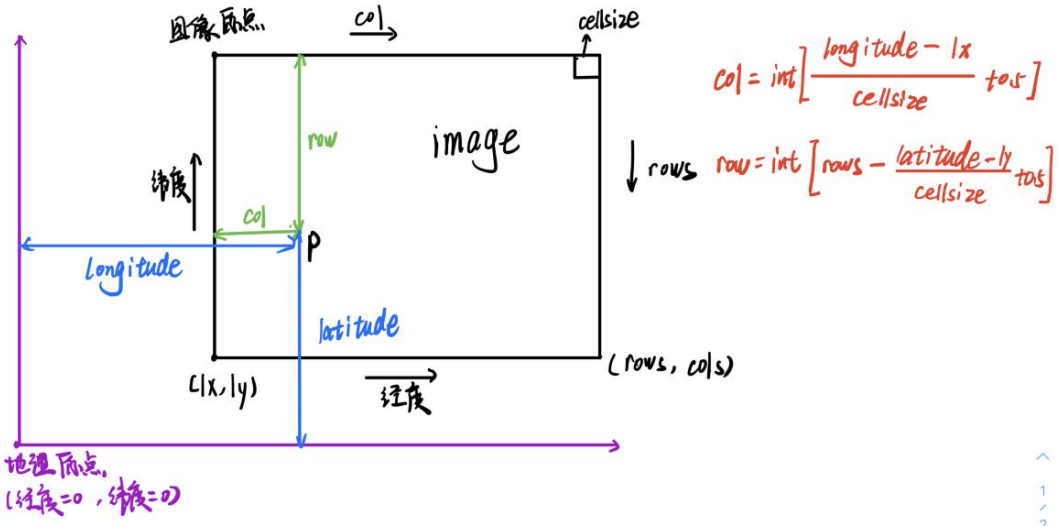
作业三

一．改造淹没分析代码，对 srtm.asc 数据进行淹没分析计算

（1）编写坐标转换函数 geo2image，实现地理经纬度和图像行列号坐标的转换。采用的公式如下：




```
def geo2image(longitude,latitude,cols, rows, lx, ly, cellsize):#坐标转换
    '''longitude:经度 latitude:纬度
    cols:图像最大列数 rows:图像最大行数 cellsize:单元格的经纬跨度
    lx:图像左下角经度 ly:图像左下角纬度'''
    col=int((longitude-lx)/cellsize+0.5)#计算列号
    row=int(rows-(latitude-ly)/cellsize+0.5)#计算行号
    return [row,col]#返回行列号
```

（2）然后编写淹没函数 floodFill。Filled 存储已经填充的单元格，fill 存储待填充的单元格，首先把种子点的行列号加入到待填充的单元格集合当中，然后就可以开始遍历了。

```
def floodFill(r,c, mask):#淹没函数
    '''r:种子点行号 c:种子点列号 mask:经过淹没条件二值化后的图形蒙版'''
    filled = set()# 已经填充的单元格
    fill = set()# 待填充的单元格
    fill.add((r,c))#填充种子点
    width = mask.shape[1]-1#计算行列数
    height = mask.shape[0]-1
    # 输出淹没数组
    flood = np.zeros_like(mask, dtype=np.int8)
```

当待填充集合不为空时，从里面取出一个，如果满足淹没条件，就进行填充，然后加入到已填充集合当中。然后将其上下左右四个单元格全部加入待填充集合当中，循环处理，直到处理完全部单元格。返回淹没好的图像数组。

```
    # 遍历并且修改需要检查的单元格，填充一个时，将其加入集合中
    while fill:
        x, y = fill.pop()#获取一个单元格
        if x == height or y == width or x < 0 or y < 0:#防止越界
            continue
        if mask[x][y] == 1:#如果这个单元格淹没条件
            flood[x][y] = 1#填充
            filled.add((x, y))#加入已填充集合
            # 检查四周单元格
            west = (x, y-1)
            east = (x, y+1)
            north = (x-1, y)
            south = (x+1, y)
            if west not in filled:#如果没有被填充
                fill.add(west)#加入待填充集合，等待下一次填充
            if east not in filled:#如果没有被填充
                fill.add(east)#加入待填充集合，等待下一次填充
            if north not in filled:#如果没有被填充
                fill.add(north)#加入待填充集合，等待下一次填充
            if south not in filled:#如果没有被填充
                fill.add(south)#加入待填充集合，等待下一次填充
    return flood#返回填充后的数组
```

（3)编写保存淹没图像的函数 saveFloodImage。首先使用 numpy 库里的 loadtxt 函数打开图片（跳过前面的头文信息）

```
def saveFloodImage(src,target,longitude,latitude,above):#保存淹没图片
    '''src为原图像路径,target为目标图像路径
    longitude,latitude为经纬度
    above为最大淹没高度(相对种子点而言)'''
    img = np.loadtxt(src, skiprows=6)#跳过开头6行，打开图片
```

将头文件中关于图像的信息提取出来

```
#头文件信息提取
hdr = [getline(source, i) for i in range(1, 7)]
values = [float(h.split(" ")[-1].strip()) for h in hdr]#取出头文件的数据，去掉\n,\t
cols, rows, lx, ly, cell, nd = values
```

使用 geo2image 函数将经纬度转换为图像行列号，并且将种子点的相关信息输出

```
#坐标转换
[row,col]=geo2image(longitude,latitude,cols,rows,lx,ly,cell)#得到行列号
height=img[row,col]#原始点高度
print('The initial point---\nlongitude:'
    +str(longitude)+'  latitude:'+str(latitude)+'\nheight:'+str(height)+'\nrow,col:'+str([row,col]))
```

使用 numpy 库的 where 函数，对图像进行二值处理，小于淹没高度的设置为 1，否则设置为 0. 然后调用 floodFill 函数开始淹没。

```
a = np.where(img < height+above, 1, 0)#高程满足条件输出1，否则输出0
fld = floodFill(row,col, a)#开始淹没
```

将头文件组合起来，打开输出图片，进行写入，保存图片。

```
header = ""
for i in range(6):#组合头文件
    header += hdr[i]

# 打开输出文件，加入头文件，保存数组
with open(target, "wb") as f:
    f.write(bytes(header, 'UTF-8'))
    np.savetxt(f, fld, fmt="%1i")
```

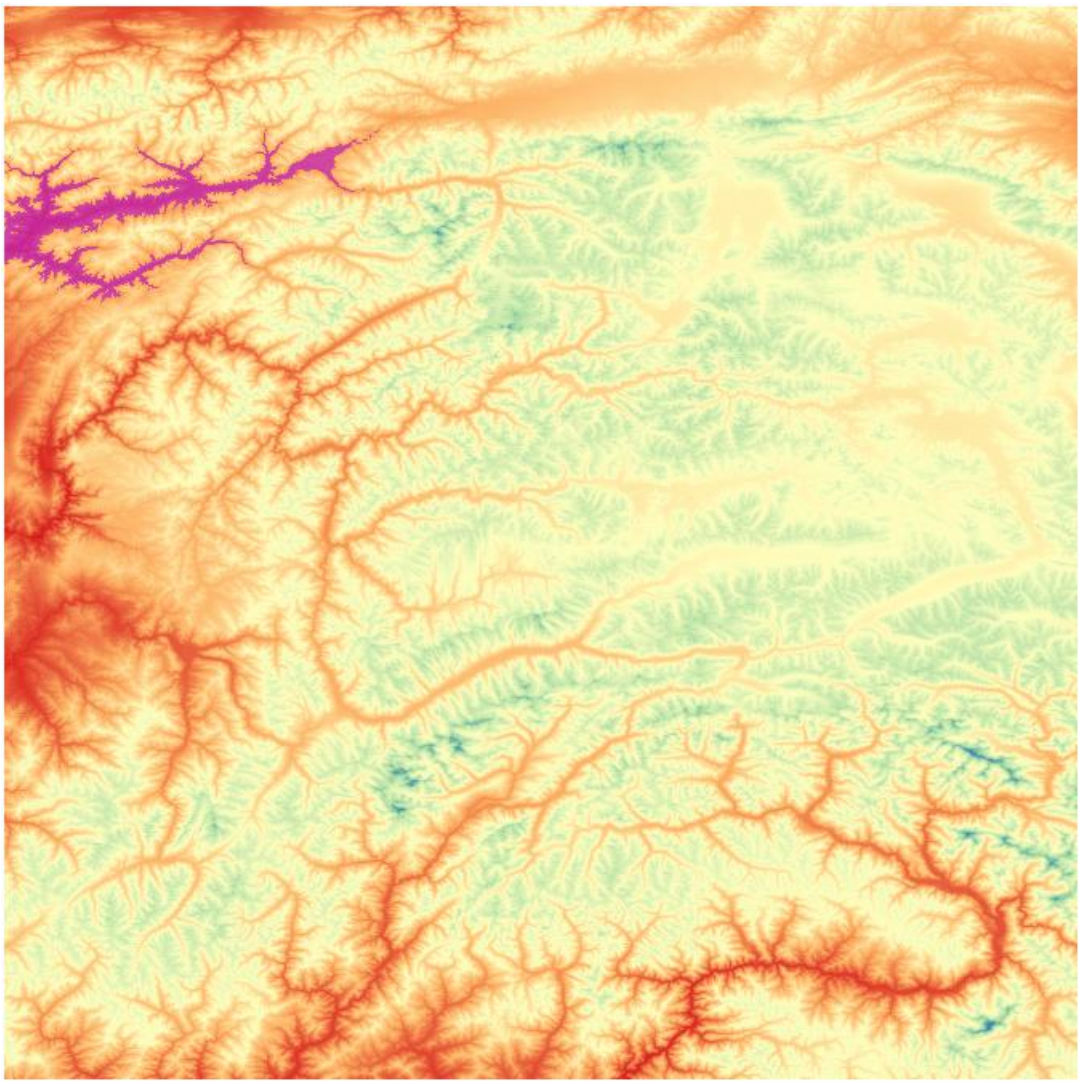
（4）调用函数，进行试验：

```
source = ".\\flood\\srtm.asc"#原始图片
saveFloodImage(source, ".\\flood\\flood1.asc", 71.475, 39.262, 200)
saveFloodImage(source, ".\\flood\\flood2.asc", 71.475, 39.262, 400)
saveFloodImage(source, ".\\flood\\flood3.asc", 74.404, 35.892, 200)
saveFloodImage(source, ".\\flood\\flood4.asc", 71.806,35.830,300)
saveFloodImage(source, ".\\flood\\flood5.asc", 71.806,35.830,500)
```

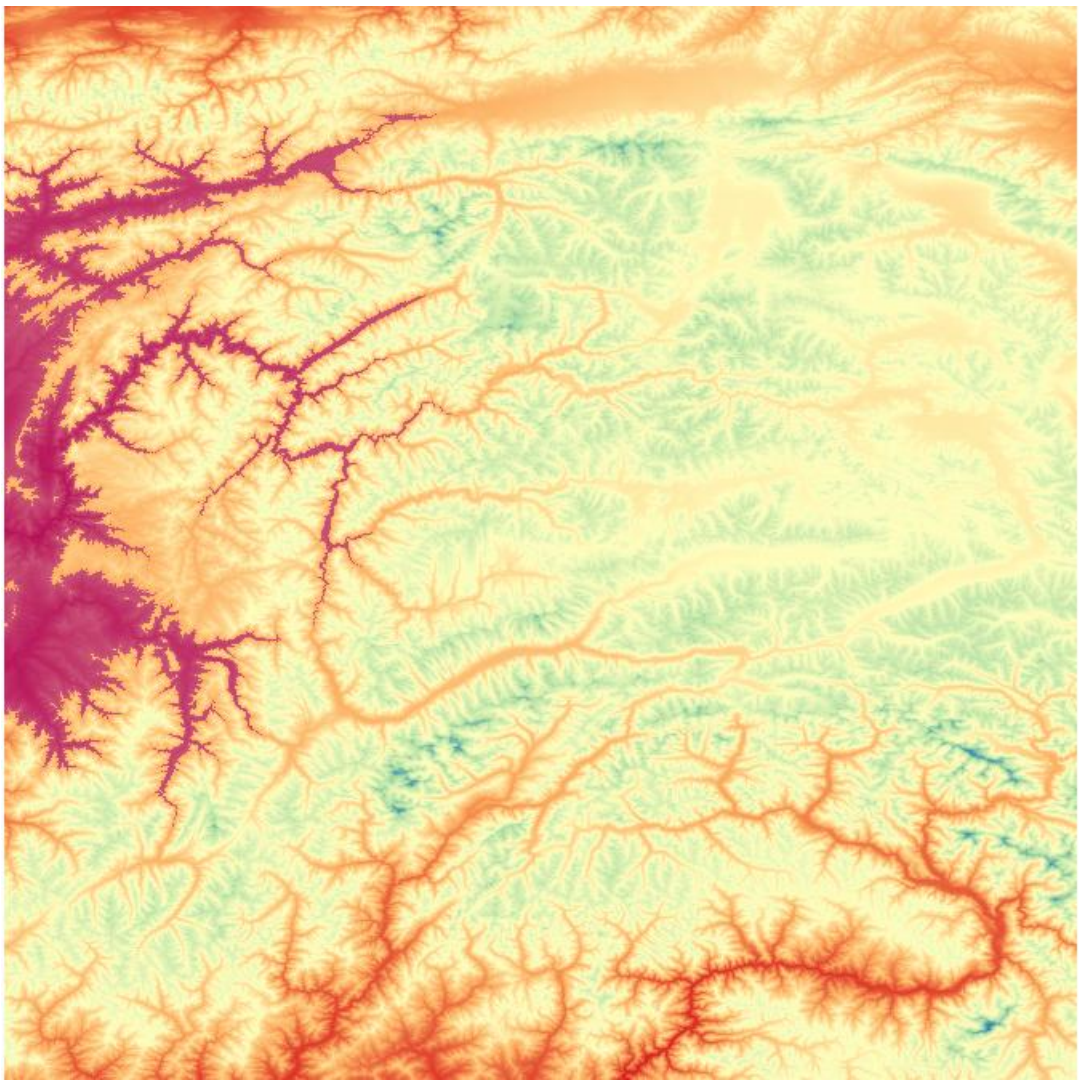
（5）结果展示：

第一组：相同位置不同的淹没高度

```
The initial point---
longitude:71.475  latitude:39.262
height:1958.0
row,col:[886, 1770]
above:200
```

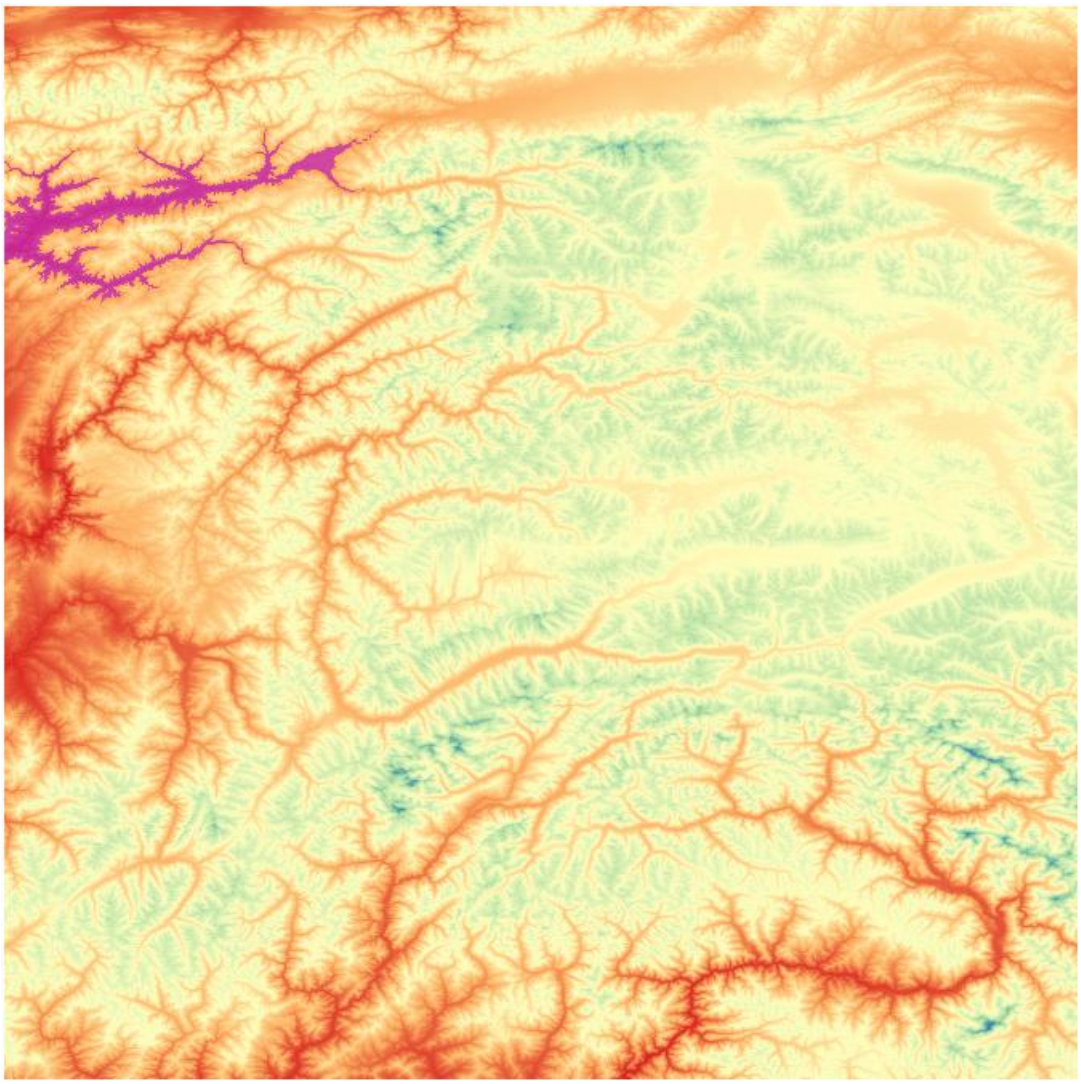


```
The initial point---
longitude:71.475  latitude:39.262
height:1958.0
row,col:[886, 1770]
above:400
```

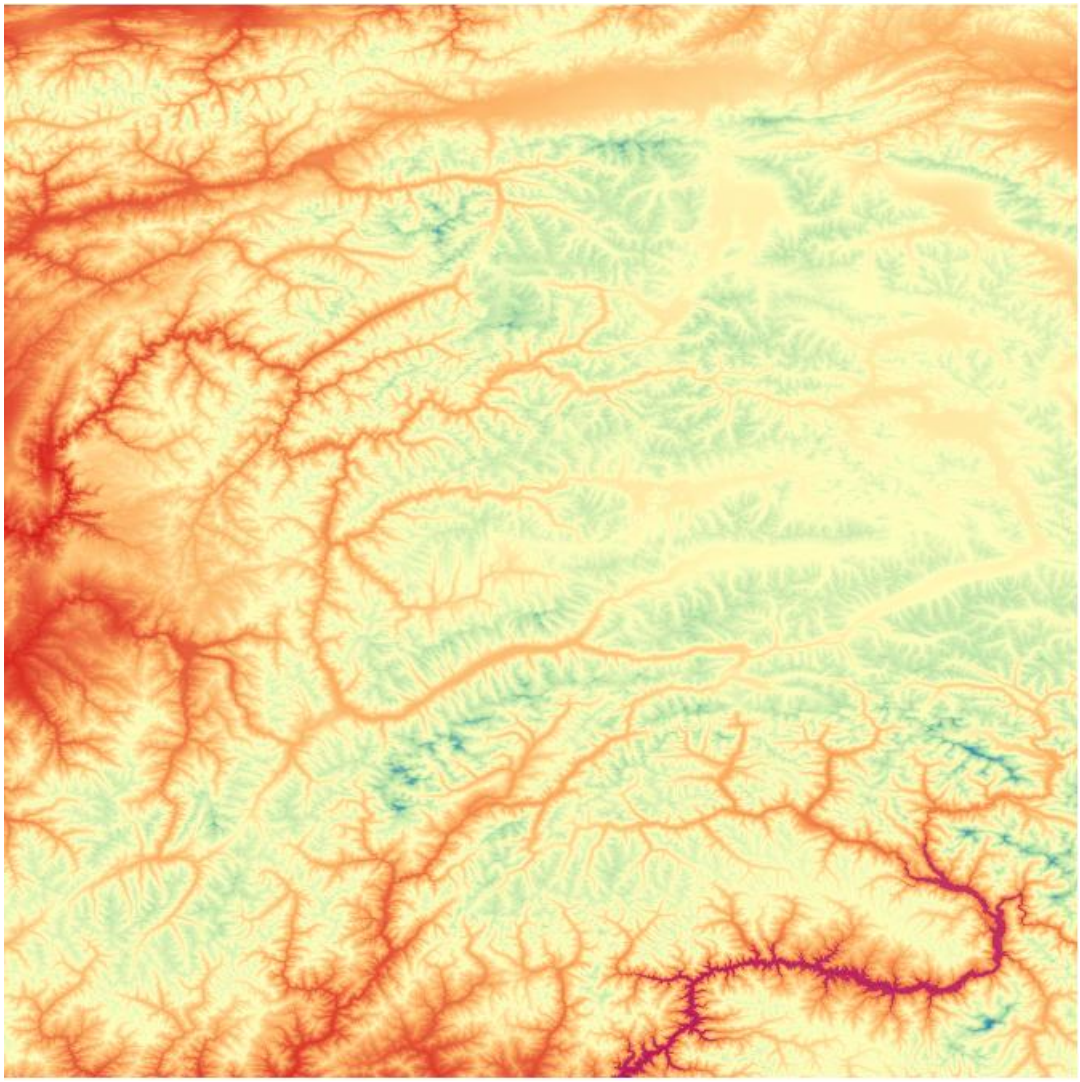


第二组：不同位置相同淹没高度

```
The initial point---
longitude:71.475  latitude:39.262
height:1958.0
row,col:[886, 1770]
above:200
```

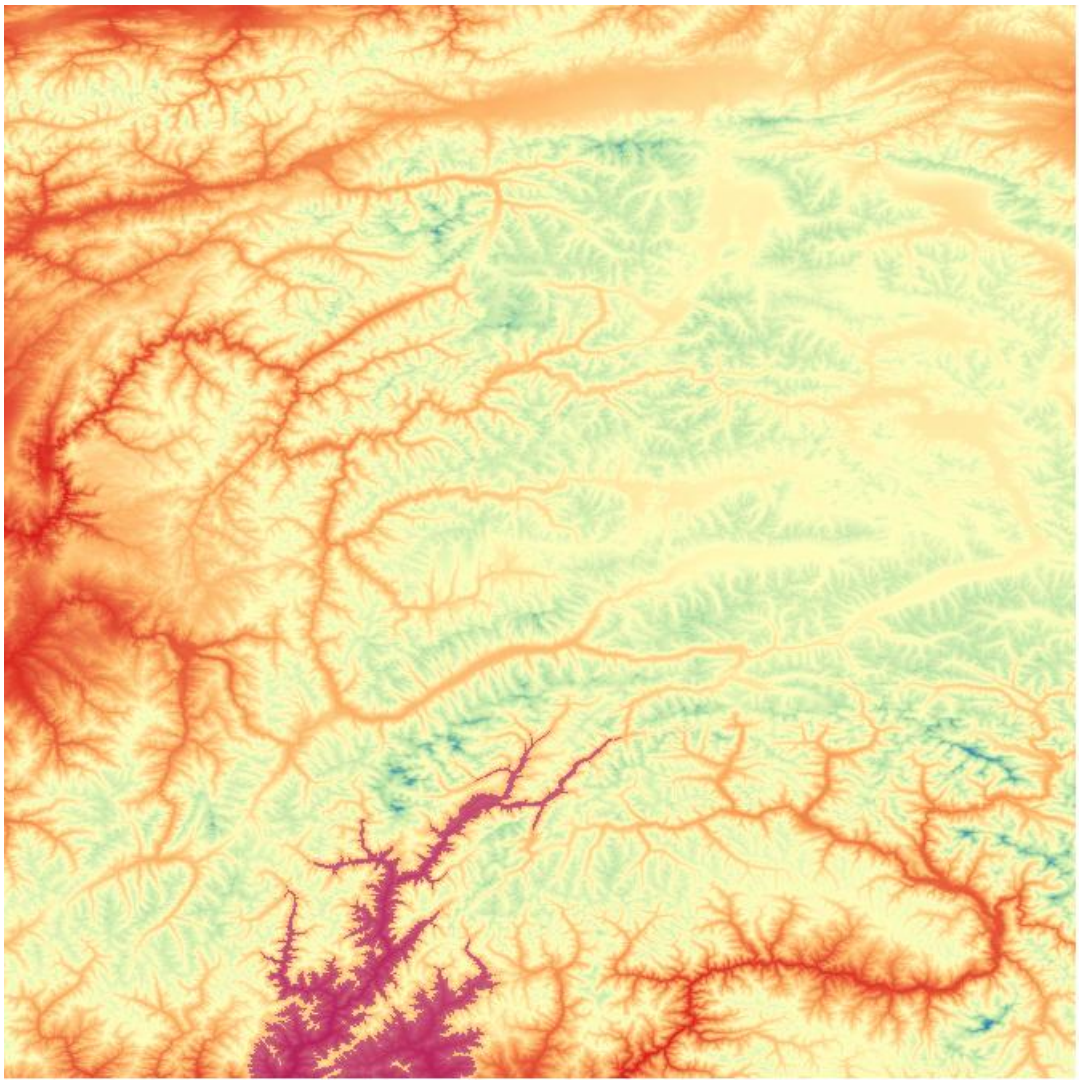


```
The initial point---
longitude:74.404  latitude:35.892
height:1397.0
row,col:[4930, 5285]
above:200
```

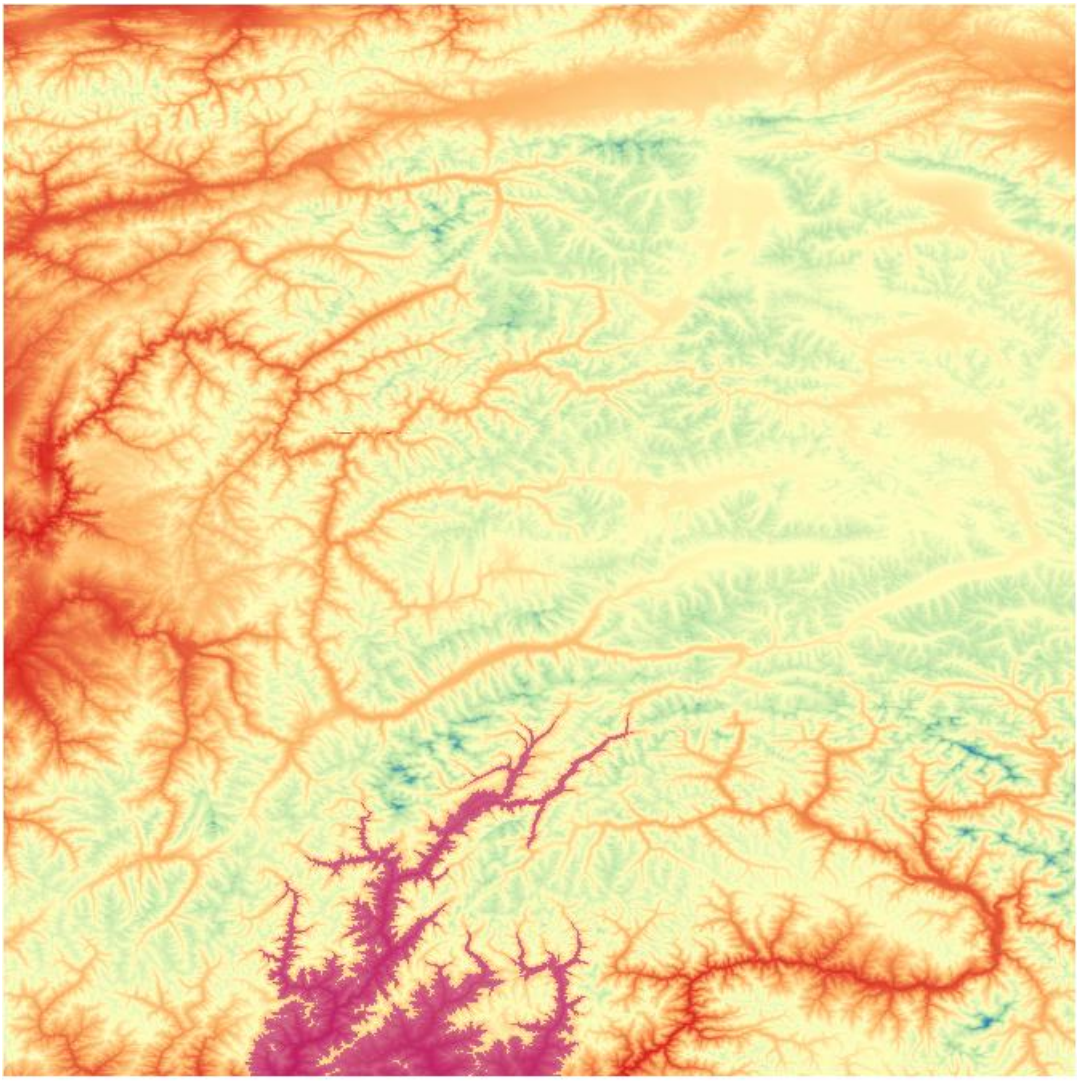


其他试验：

```
The initial point---
longitude:71.806  latitude:35.83
height:2391.0
row,col:[5005, 2168]
above:300
```



```
The initial point---
longitude:71.806  latitude:35.83
height:2391.0
row,col:[5005, 2168]
above:500
```



二. 改造遥感数据分类计算程序，对 GF1.jpg 进行分类计算，分别按照 5,10,15 个类别进行分类，并且适当修改颜色列表参数（可随机）.

（1）导入所用库 gdal_array 和 numpy(这里遇到了一个小问题,就是安装了 gdal 库但是无法导入，后来发现 gdal 和 gdal_array 都放在了 osgeo 库下面）

```
from osgeo import gdal_array
import numpy as np
```

（2）编写 make_rand_lut 函数，实现自动生成包含 num+1 个 rgb 三元组的颜色查找表。这里使用的是 numpy 模块中 random 的 randint 函数实现的，每生成一个就将其加入到 lut 列表中（这个函数输入的参数是 num 即分类的个数，但是要生成 num+1 个颜色，这是由于颜色查找表中的颜色要等于类别数+1，具体原因我们在下面探讨）

```
def make_rand_lut(num):#制造包含num个随机颜色的颜色查找表
    lut=[]#颜色查找表
    for i in range(0,num+1):#这里必须+1，因为颜色查找表中的颜色个数=种类数+1
        lut.append(np.random.randint(0,255,3))#产生随机的RGB颜色存入查找表中
    return lut#返回查找表
```

（3）编写 classify 函数，实现图像的分离计算。其具体参数及编写过程如下：

```
def classify(src,num):#分类函数
    '''src为原图路径,num为类别个数'''
```

①使用 gdal_array 的 LoadFile 方法将原图像加载到数组 srcArr 中存储起来，然后使用 histogram 方法，将图像根据灰度值分为 num 个条带

```
srcArr = gdal_array.LoadFile(src)# 使用gdal库将图片加载到numpy库
classes = gdal_array.numpy.histogram(srcArr, bins=num)[1]#将图像根据灰度分为num个条带
```

②使用之前编写的 make_rand_lut 函数生成包含 num+1 个颜色的颜色查找列表，设置初始灰度为 start=1，并且创建一个用于输出的图像模板，其大小和原图像一致。

```
lut =make_rand_lut(num)#生成颜色查找表 - must be len(classes)+1.
start = 1 #从灰度值等于1开始处理
# 创建一个rgb颜色用于输出的图像
rgb = gdal_array.numpy.zeros((3, srcArr.shape[0],
                              srcArr.shape[1], ), gdal_array.numpy.float32)
```

③循环遍历每一个类，并且给他们赋予颜色。这里 mask 为上一类灰度最大值到这一类灰度最大值之间图像的蒙版。

```
#处理每一个类，并且给他们渲染上颜色
for i in range(len(classes)):
    #mask为上一类分带灰度最大值到这一类灰度最大值之间图像的蒙版
    mask = gdal_array.numpy.logical_and(start <= srcArr, srcArr <= classes[i])
    for j in range(len(lut[i])):
        rgb[j] = gdal_array.numpy.choose(mask, (rgb[j], lut[i][j]))#选择颜色
    start = classes[i]+1#将处理初始灰度值更新到下一个区间
```

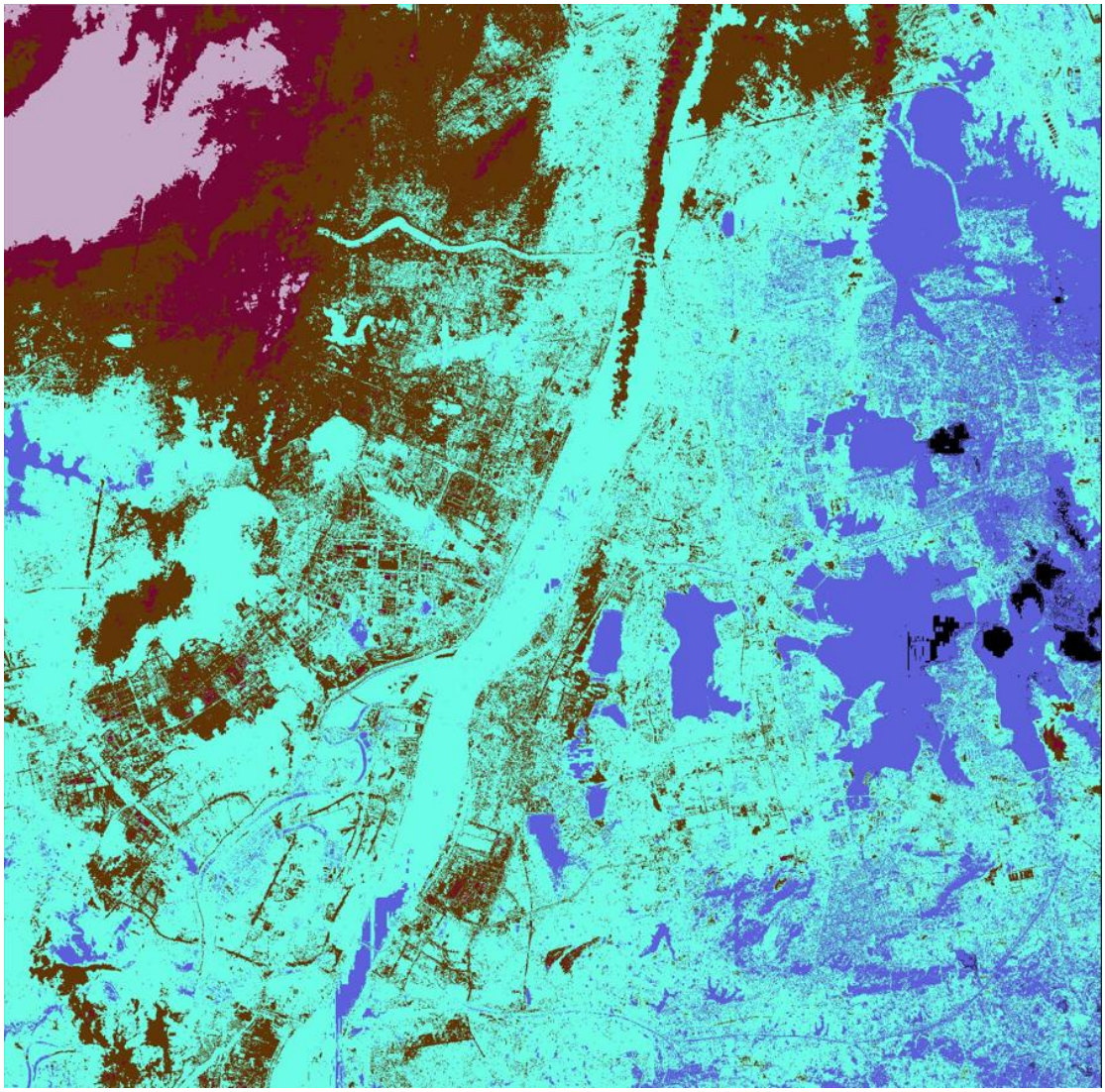
④保存输出图片

```
# 保存图像
tgt = ".\\classify\\classified"#目标图像
output = gdal_array.SaveArray(rgb.astype(gdal_array.numpy.uint8), tgt+'_'+str(num)+'.jpg', format="JPEG")
output = None
```

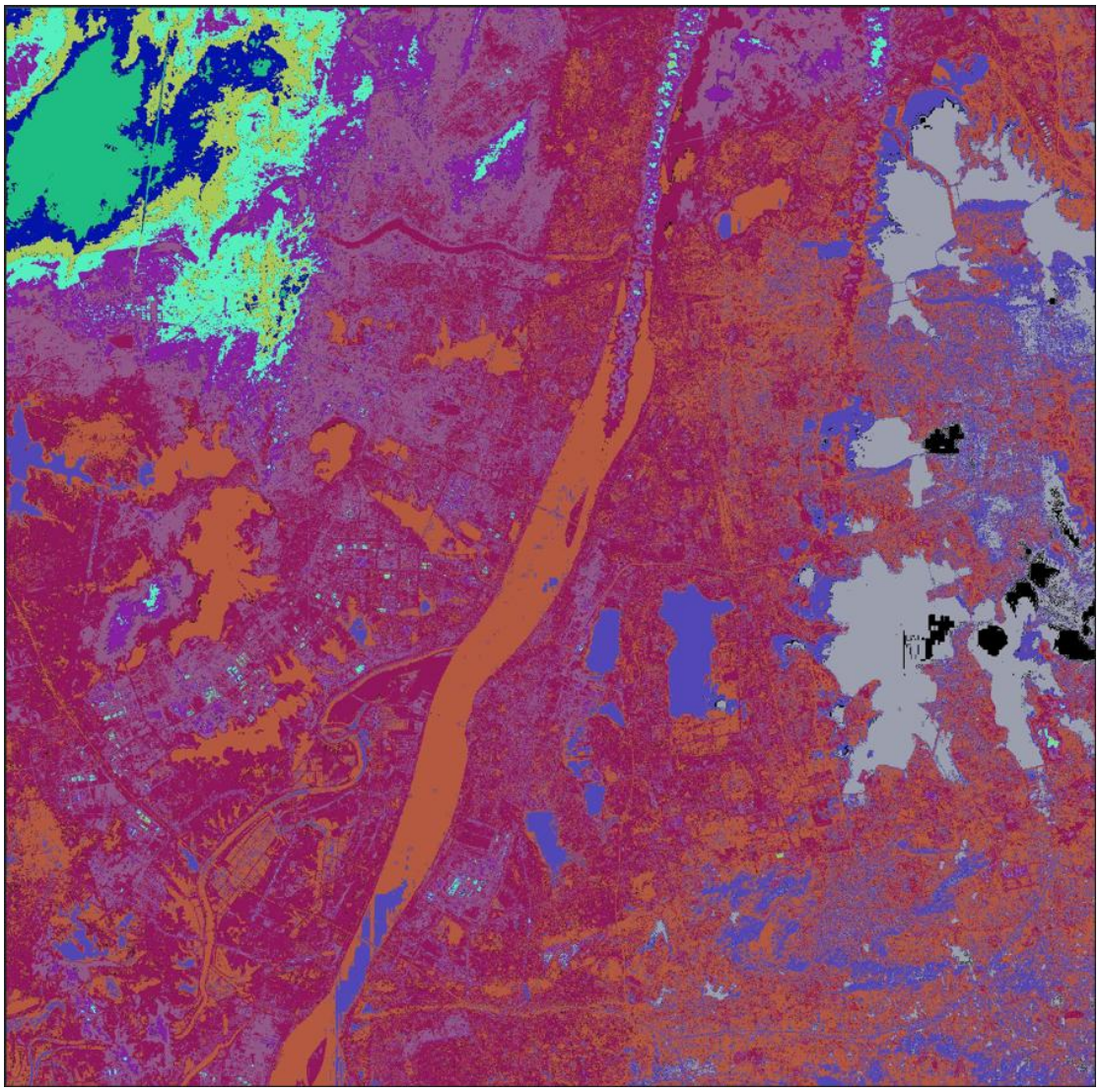
（4）调用函数，进行试验

```
src = ".\\classify\\GF1.jpg"#原图像
classify(src,5)
classify(src,10)
classify(src,15)
```

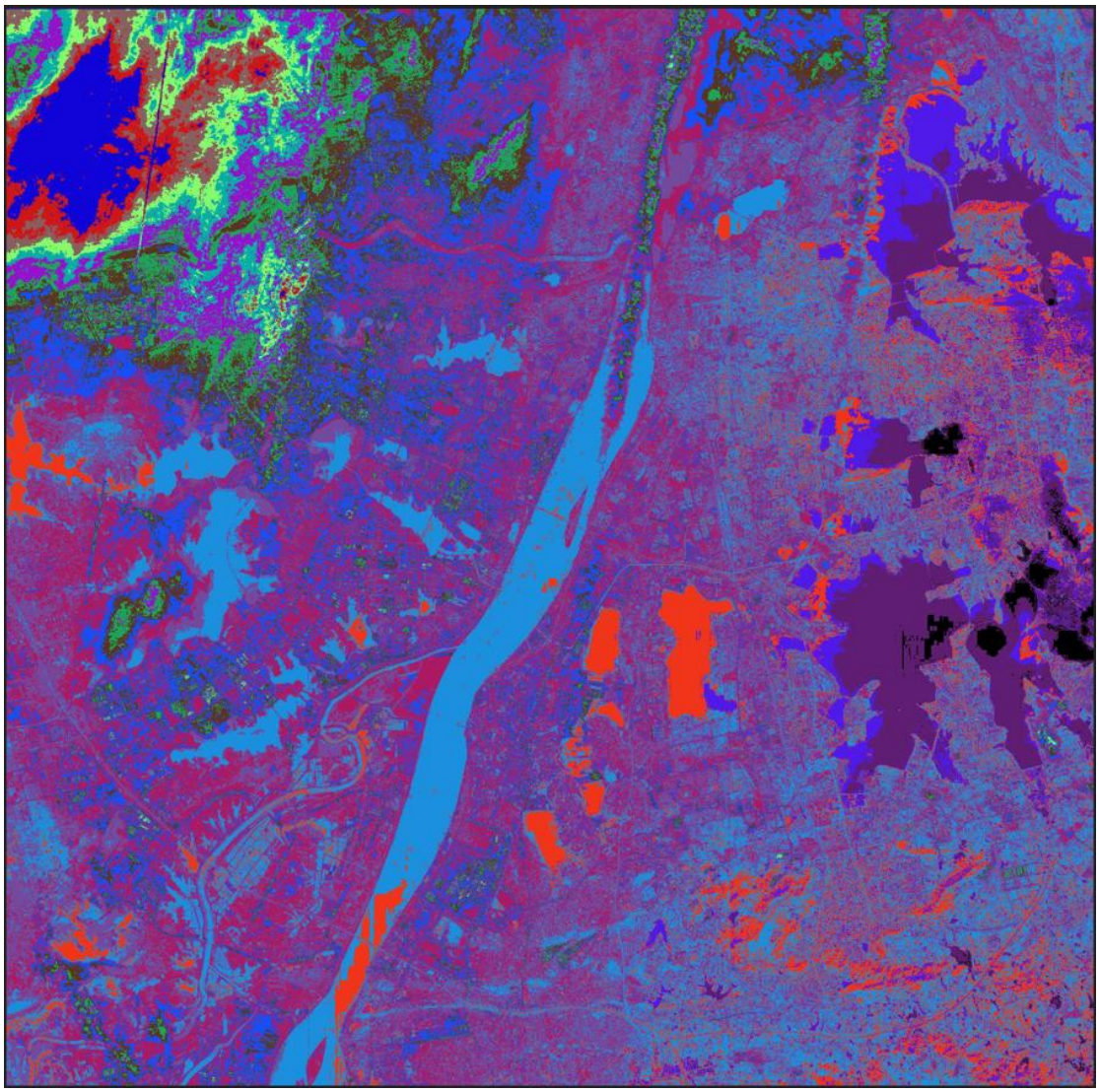
（5）结果展示：



（num=5）



（num=10）



（num=15）

（6）一些思考：在源码的注释里面我注意到一个地方写道“len（lut） must be classes+1”。那么为什么颜色表的颜色个数一定就要比分类个数多一个呢？经过分析我认为原因如下：

```
srcArr:
[[253 254 255 ...  86  87  90]
 [213 215 217 ...  75  76  79]
 [208 210 212 ...  75  75  77]
 ...
 [ 71  71  75 ...  38  43  50]
 [ 70  69  74 ...  26  30  36]
 [ 77  73  74 ...  42  44  45]]
classes:
[  0.  51. 102. 153. 204. 255.]
```

首先我们来看 srcArr 到底是什么组成的，实际上假设图像是 mxn 个像素的话，srcArr 就是一个包含了 m 个列表的列表，其中每个列表又包含了 n 个值，也就是说 srcArr[i][j]代表的就是第 i+1 行第 j+1 列的灰度值。

而 classes 又是什么呢？实际上它是由 num+1 个数值组成的列表，其相邻两

个数值表示了这个类的最小最大灰度值，比如classes[0]和classes[1]代表了第一类的灰度值范围。

那么为什么颜色表lut要包含num+1个颜色呢，num个颜色不是刚刚好，一个类一个吗？实际是因为我们在循环处理的时候生成的蒙版为start<=srcArr<=classes[i]，而我们的初始处理值start=1，classes[0]=0也就是不存在这样的srcArr，所以lut[0]所包含的颜色实际上是浪费了。

```
for i in range(len(classes)):
    #mask为上一类分带灰度最大值到这一类灰度最大值之间图像的蒙版
    mask = gdal_array.numpy.logical_and(start <= srcArr, srcArr <= classes[i])
    for j in range(len(lut[i])):
        rgb[j] = gdal_array.numpy.choose(mask, (rgb[j], lut[i][j]))#选择颜色
    start = classes[i]+1#将处理初始灰度值更新到下一个区间
```