question4

June 19, 2021

[1]:

```
from scipy.spatial import Delaunay
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

0.1 Generating the Triangles

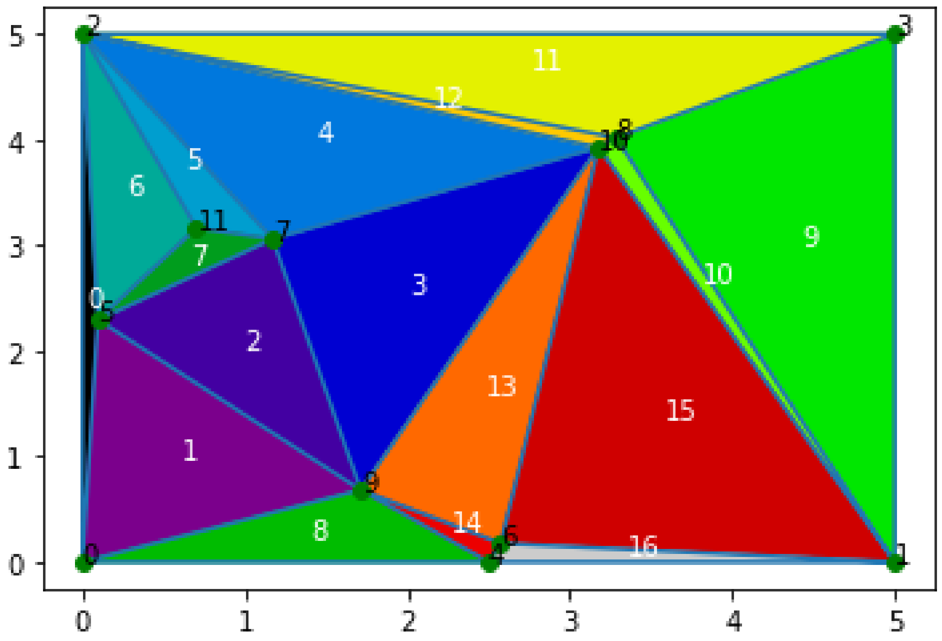
Using the inbuilt function Delaunay available in the scipy package

[2]:

```
def Triangulate(A, B, n):
    # Initializing the Vertices
    vertices = [[0,0], [A,0], [0,B], [A,B]]
    # Generating n different colours
    # NOTE: Although the colors may appear similar to the eye, the RGB Values␣
 ↪are different.
    cmap = plt.get_cmap('nipy_spectral')
    colors = [cmap(i) for i in np.linspace(0, 1, n)]

    # Determing the number of points to obtain n triangles
    if n%2==0:
        k = n/2-1
        locations = vertices
    else:
        k = n//2-1
        locations = vertices
        locations.append([A/2,0])

    # Randomly generating the points
    others_x = np.random.rand(k)*A
    others_y = np.random.rand(k)*B
    for i in range(k):
        locations.append([others_x[i], others_y[i]])
    local = np.array(locations)

    # Using the inbuilt function Delaunay in the scipy library to map the␣
 ↪triangles
    p = plt.figure()
    tri = Delaunay(local)
    p = plt.triplot(local[:,0], local[:,1], tri.simplices)
    for i in range(len(local)):
        p = plt.plot(local[i,0], local[i,1], 'o', color='green')
        p = plt.annotate(str(i), (local[i,0], local[i,1]), color='black')

    # Filling the triangles
    count = 0
    for point in tri.simplices:
        x = [local[point[0]][0], local[point[1]][0], local[point[2]][0]]
        y = [local[point[0]][1], local[point[1]][1], local[point[2]][1]]
        p = plt.fill(x,y, color=colors[count])
        p = plt.annotate(str(count), (np.sum(x)/3, np.sum(y)/3), color='white')
        count+=1
        count = count%len(colors)

    edges = []

    # Determing the vertices and shared edges for each triangle
    for point in tri.simplices:
        data = {"Vetrices": [], "Edges": []}
        data["Vetrices"] = sorted([point[0], point[1], point[2]])
        for vert1 in data["Vetrices"]:
            for vert2 in data["Vetrices"]:
                if vert1!=vert2:
                    if n%2==0:
                        corner = [0, 1, 2, 3]
                    else:
                        corner = [0, 1, 2, 3, 4]
                    if vert1 in corner and vert2  in corner:
                        continue
                    else:
                        if data["Edges"].count(sorted([vert1, vert2]))==0:
                            data["Edges"].append(sorted([vert1, vert2]))
        edges.append(data)

    fname = str(n) + "-Triangulation.png"
    plt.savefig(fname, dpi=600)

    return p, pd.DataFrame(edges)
```

Generating for $A = 5$, $B = 5$, $n = 17$

[3]:
```
p, triangles = Triangulate(5,5,17)
```

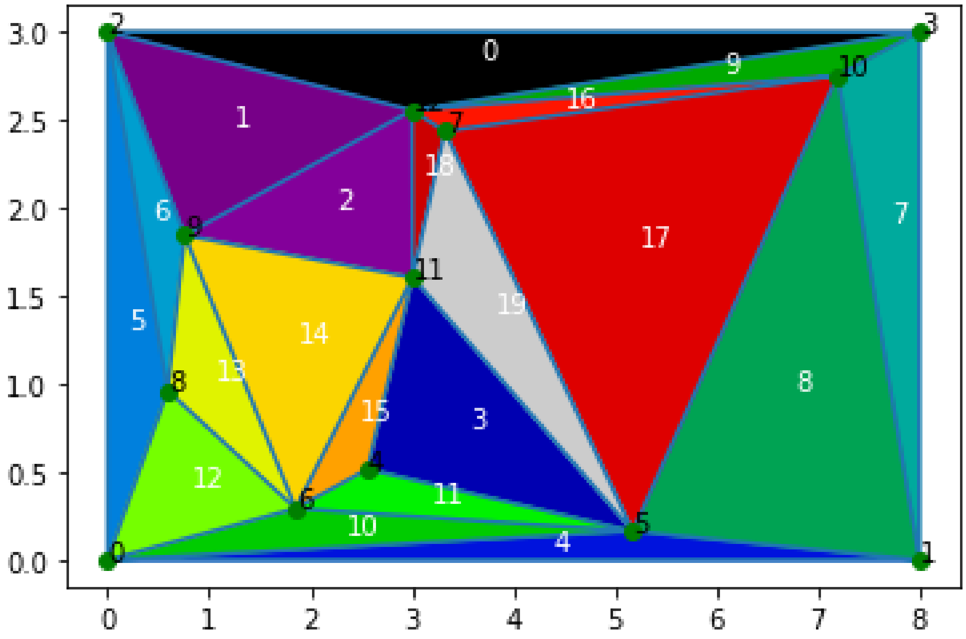


```
[4]: triangles
```

```
[4]:        Vetrices                          Edges
     0     [0, 2, 5]             [[0, 5], [2, 5]]
     1     [0, 5, 9]     [[0, 5], [0, 9], [5, 9]]
     2     [5, 7, 9]     [[5, 7], [5, 9], [7, 9]]
     3    [7, 9, 10]  [[7, 9], [7, 10], [9, 10]]
     4    [2, 7, 10]  [[2, 7], [2, 10], [7, 10]]
     5    [2, 7, 11]  [[2, 7], [2, 11], [7, 11]]
     6    [2, 5, 11]  [[2, 5], [2, 11], [5, 11]]
     7    [5, 7, 11]  [[5, 7], [5, 11], [7, 11]]
     8     [0, 4, 9]             [[0, 9], [4, 9]]
     9     [1, 3, 8]             [[1, 8], [3, 8]]
     10   [1, 8, 10]  [[1, 8], [1, 10], [8, 10]]
     11    [2, 3, 8]             [[2, 8], [3, 8]]
     12   [2, 8, 10]  [[2, 8], [2, 10], [8, 10]]
     13   [6, 9, 10]  [[6, 9], [6, 10], [9, 10]]
     14    [4, 6, 9]     [[4, 6], [4, 9], [6, 9]]
     15   [1, 6, 10]  [[1, 6], [1, 10], [6, 10]]
     16    [1, 4, 6]             [[1, 6], [4, 6]]
```

Generating for $A = 8$, $B = 3$, $n = 20$

```
[5]: p, triangles = Triangulate(8,3,20)
```



[6]:
```
triangles
```

[6]:
```
       Vetrices                         Edges
0    [2, 3, 12]            [[2, 12], [3, 12]]
1    [2, 9, 12]    [[2, 9], [2, 12], [9, 12]]
2   [9, 11, 12]  [[9, 11], [9, 12], [11, 12]]
3    [4, 5, 11]    [[4, 5], [4, 11], [5, 11]]
4     [0, 1, 5]              [[0, 5], [1, 5]]
5     [0, 2, 8]              [[0, 8], [2, 8]]
6     [2, 8, 9]      [[2, 8], [2, 9], [8, 9]]
7    [1, 3, 10]            [[1, 10], [3, 10]]
8    [1, 5, 10]    [[1, 5], [1, 10], [5, 10]]
9   [3, 10, 12]  [[3, 10], [3, 12], [10, 12]]
10    [0, 5, 6]      [[0, 5], [0, 6], [5, 6]]
11    [4, 5, 6]      [[4, 5], [4, 6], [5, 6]]
12    [0, 6, 8]      [[0, 6], [0, 8], [6, 8]]
13    [6, 8, 9]      [[6, 8], [6, 9], [8, 9]]
14   [6, 9, 11]    [[6, 9], [6, 11], [9, 11]]
15   [4, 6, 11]    [[4, 6], [4, 11], [6, 11]]
16  [7, 10, 12]  [[7, 10], [7, 12], [10, 12]]
17   [5, 7, 10]    [[5, 7], [5, 10], [7, 10]]
18  [7, 11, 12]  [[7, 11], [7, 12], [11, 12]]
19   [5, 7, 11]    [[5, 7], [5, 11], [7, 11]]
```

[ ]: