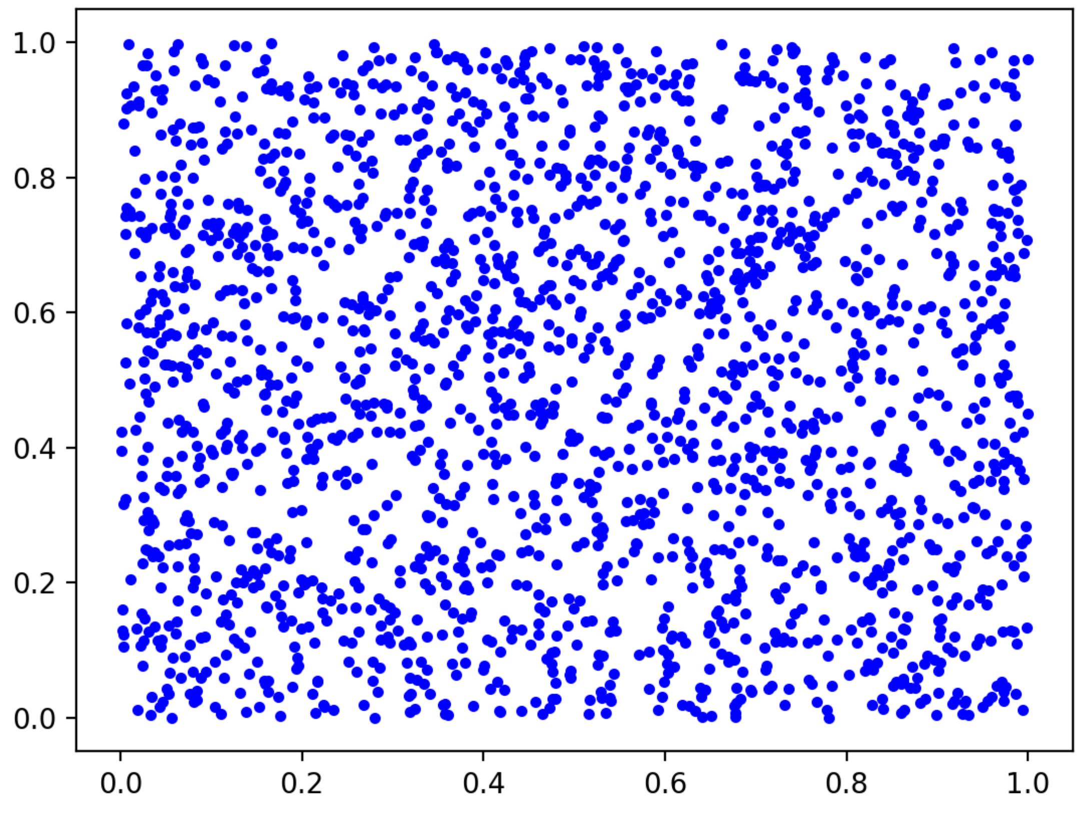
Homework 1.2

1. Generate n = 2,000 points uniformly at random in the two-dimensional unit square. Which point do you expect the centroid to be?

首先，通过numpy包的random库随机生成2000个点（0<=x,y<=1），代码和生成的点图如下：

```
points = np.random.rand(2000,2)
plt.plot(points[:,0],points[:,1],'.',color='b')
```



使用numpy.random.rand所生成的数据满足在[0,1]区间上的均匀分布，且质心可以定义为

$$(\frac{\sum_{i=1}^{N} x_i}{N}, \frac{\sum_{i=1}^{N} y_i}{N})$$

即x，y坐标上的均值，所以期望的质心应该为$(0.5,0.5)$。

2. What objective does the centroid of the points optimize?

因为质心可以理解为与所有的点距离和最小的点，因此对于求质心的目标函数可以定义为质心点到所有点的欧式距离和。即：

$$f(x^*, y^*) = \sum_{i=1}^{N} \sqrt{(x^* - x_i)^2 + (y^* - y_i)^2}$$

代码实现为：

```
# What objective does the centroid of the points optimize
def cost(centroid):
    sum = 0
    for i in range(2000):
        sum += ((centroid[0]-points[i,0])**2+(centroid[1]-points[i,1])**2)**0.5
    return sum
```

3. Apply gradient descent (GD) to find the centroid.

梯度下降方法即从最开始**随机初始化一个质心点**，然后**循环一定次数**，**每次往梯度下降的方向计算新的质心点的值，直到达到最大循环次数或者到已经收敛**。而梯度下降方法的梯度计算即：将cost对质心x*和y*分别求偏导，然后将所有点的x,y值带入计算，得到在x方向上的梯度和在y方向上的梯度。得到梯度之后，**需要用一个学习率lambda去进行更新质心的x*,y***。在这题中，我是固定学习率lambda进行计算。同时在我实验时，由于梯度计算出来很大，所以导致梯度更新不稳定，经常跳动很大，因此我还在计算梯度时使用了一个normalize，

目标函数分别对x*,y*求偏导结果如下：

$$\frac{\partial f(x^*, y^*)}{\partial x^*} = \sum_{i=1}^{N} \frac{x^* - x_i}{\sqrt{(x^* - x_i)^2 + (y^* - y_i)^2}}$$

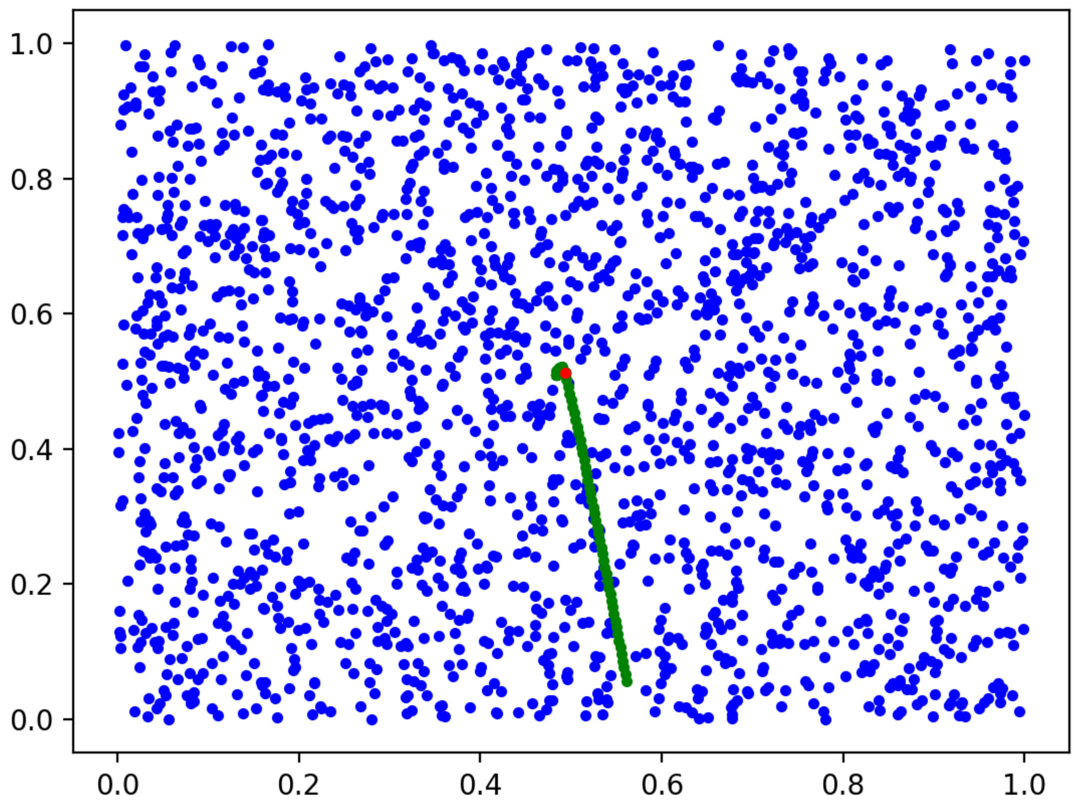
$$\frac{\partial f(x^*, y^*)}{\partial y^*} = \sum_{i=1}^{N} \frac{y^* - y_i}{\sqrt{(x^* - x_i)^2 + (y^* - y_i)^2}}$$

计算代码如图：

```python
# Apply gradient descent (GD) to find the centroid
def gradient_descent(centroid):
    sum_dx = 0
    sum_dy = 0
    for i in range(2000):
        divided = ((centroid[0]-points[i,0])**2+(centroid[1]-points[i,1])**2)**0.5
        sum_dx += ((centroid[0]-points[i,0])/divided)
        sum_dy += ((centroid[1]-points[i,1])/divided)
    sum = (sum_dx**2+sum_dy**2)**0.5
    dx = sum_dx/sum
    dy = sum_dy/sum
    return dx, dy

centroid = np.random.rand(2)
theta = 0.01
max_loop = 100
plt.figure(2)
plt.plot(points[:,0],points[:,1],'.',color='b')
for i in range(max_loop):
    print("Centroid is:", centroid)
    print("Cost is:", cost(centroid))
    plt.plot(centroid[0],centroid[1],'.', color='g')
    dx, dy = gradient_descent(centroid)
    centroid[0] = centroid[0] - theta * dx
    centroid[1] = centroid[1] - theta * dy
```

进行梯度下降时每一步计算出来的点连成的折线图如下：



下面是程序梯度下降算法算出来最后的质心值，约等于（0.5，0.5）：

```
Final Centroid is: [0.50461496 0.50568666]
Final Cost is: 765.5069680793836
```

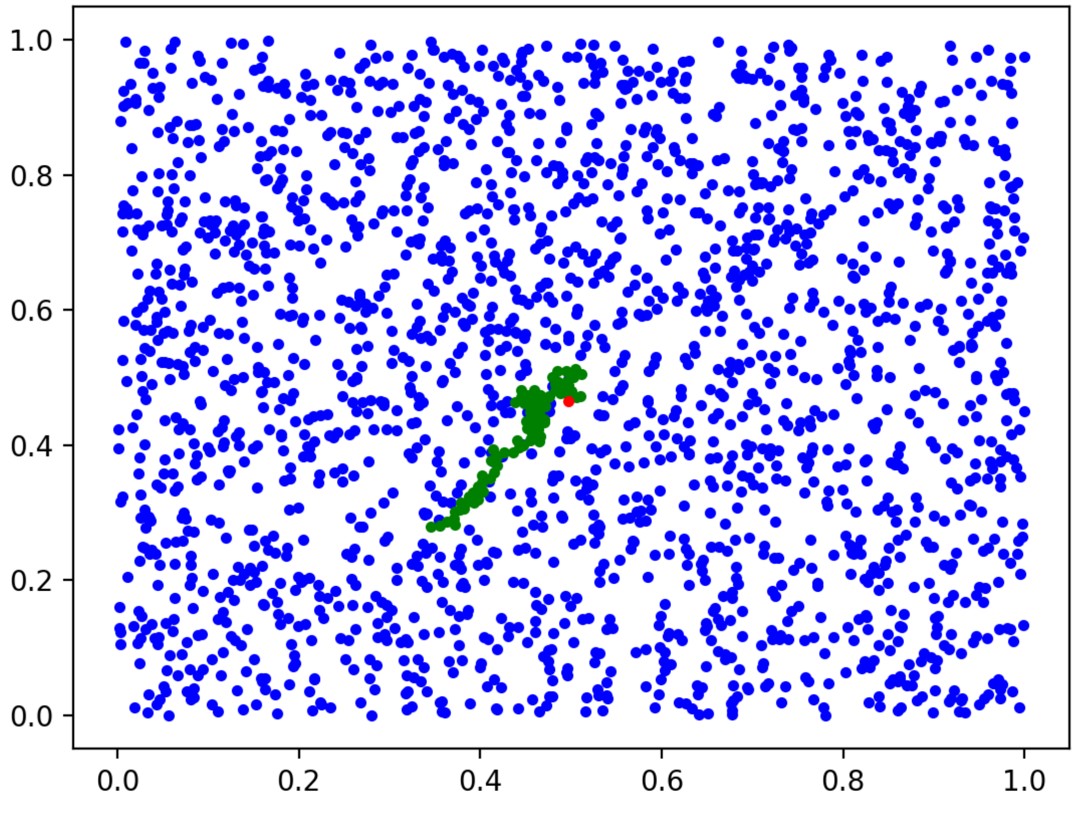
4. Apply stochastic gradient descent (SGD) to find the centroid. Can you say in simple words, what the algorithm is doing?

随机梯度下降和梯度下降类似，只有一个区别就是随机梯度下降是**每次只随机选择一个点求梯度进行梯度下降**，而**梯度下降算法是每次对所有的点进行计算求梯度**。简单来说，这个算法就是每次随机选择一个点，将该点带入原本算好的梯度公式中，得到在该点处的梯度值用来更新质心的x*,y*（对于梯度更新和计算梯度等整个算法过程请看第三问回答）。而随机梯度因为每次只选择一个点来更新，所以容易“走弯路”（从下面第二张图的绿色曲线可以看出），也同时有可能走进局部最优点，没有找到全局最优。但是它相对于梯度下降来说，由于每次更新只需要计算一个点，因此**计算更快，收敛更快**。

最后计算出来的质心值也是约等于（0.5，0.5）：

```
Final Centroid is: [0.49705053 0.46468097]
Final Cost is: 770.419007503627
```

随机梯度下降每一步计算出来的质心点连成的折线图如下：



随机梯度下降算法代码如下：

```python
# Apply stochastic gradient descent (SGD) to find the centroid
from random import choice
def stochastic_gradient_descent(centroid):
    point = choice(points)
    divided = ((centroid[0]-point[0])**2+(centroid[1]-point[1])**2)**0.5
    sum_dx = ((centroid[0]-point[0])/divided)
    sum_dy = ((centroid[1]-point[1])/divided)
    sum = (sum_dx**2+sum_dy**2)**0.5
    dx = sum_dx/sum
    dy = sum_dy/sum
    return dx, dy

centroid = np.random.rand(2)
theta = 0.01
max_loop = 100
plt.figure(3)
plt.plot(points[:,0],points[:,1],'.',color='b')
for i in range(max_loop):
    print("Centroid is:", centroid)
    print("Cost is:", cost(centroid))
    plt.plot(centroid[0],centroid[1],'.', color='g')
    dx, dy = stochastic_gradient_descent(centroid)
    centroid[0] = centroid[0] - theta * dx
    centroid[1] = centroid[1] - theta * dy
```