# Machine Learning Homework 1.2

专业：软件工程　　　　　姓名：沈金龙　　　　　学号：18214806

## 1. 实验题目

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

>What objective does the centroid of the points optimize?

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

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

注：In mathematics and physics, the **centroid** or geometric center of a plane figure is the arithmetic mean position of all the points in the figure. Informally, it is the point at which a cutout of the shape could be perfectly balanced on the tip of a pin.

## 2. 实验要求

1）编程语言不限。

2）作业包含一份报告（word 或pdf格式）及代码加注释，并打包到.zip，其中zip文件的命名格式为学号_姓名。

3）不允许使用梯度下降相关的库函数。

4）禁止抄袭。

## 3. 实验过程及代码

1）本实验采用 python3.6 完成，首先使用 rand 生成 2000*2 大小的随机数矩阵。由于质心的定义是所有点的算术平均位置，以此计算出质心的坐标，并在二维单元图中进行标示。

2）点的质心优化的目标是：求质心点到其余点的最短的距离和，即找到一个点 C（x，y）使得 f（x，y）最小，那么本题的优化目标就是 min f（x，y），此 f（x，y）即为损失函数 cost，其表示如下：

$$\min f(x,y) = \min \sum_{i=0}^{m} \sqrt{(x-x_i)^2 + (y-y_i)^2}$$

3）使用梯度下降算法来计算质心：

>定义损失函数，优化目标就是最小化损失函数。C 表示待求点 centroid，all_points 表示生成所有的点。下面的返回值就是表示该点到所有点距离和。

```
def cost(c, all_points):
    return sum(sum((c - all_points) ** 2, axis=1) ** 0.5)
```

>求梯度向量，根据损失函数求出 x,y 的偏导数，再求出梯度向量 array([dx, dy])

$$\nabla f(x,y) = \begin{bmatrix} \sum_i \frac{x-x_i}{\sqrt{(x-x_i)^2+(y-y_i)^2}} \\ \sum_i \frac{y-y_i}{\sqrt{(x-x_i)^2+(y-y_i)^2}} \end{bmatrix}$$

```python
def mygradient(c, all_points):
    dx = sum((c[0] - all_points[:, 0]) / sum((c - all_points) ** 2, axis=1) ** 0.5)  # 求x偏导数
    dy = sum((c[1] - all_points[:, 1]) / sum((c - all_points) ** 2, axis=1) ** 0.5)  # 求y偏导数
    s = (dx ** 2 + dy ** 2) ** 0.5
    dx = dx/s
    dy = dy/s
    return array([dx, dy])  # 得到梯度向量
```

>设置参数值。设置出发点从左上角（0，1）开始，初始化学习率，设置最大迭代次数和阈值。

```python
x = array([0, 1])  # 出发点
theta = 0.03  # 学习率
loop_max = 1000  # 最大迭代次数(防止死循环)
epsilon = 1e-6  # 设置阈值，当小于此值时停止迭代
xb = x
```

>开始循环。

cost1：表示梯度更新前的损失函数值，costi： 表示梯度更新后的损失函数值。

用更新前的损失函数值 cost1 减去更新后的 costi，差值大于阈值说明还没收敛，继续循环。当 costi-cost1>阈值，说明步长过长，已经越过最佳值，需要缩小学习率，一直到收敛。

```python
for i in range(loop_max):  # range() 函数可创建一个整数列表
    cost1 = cost(x, all_points)  # 梯度更新前的损失函数值
    xi = x - theta * mygradient(x, all_points)  # 梯度更新后的新的点
    costi = cost(xi, all_points)  # 更新后的损失函数值
    if cost1 - costi > epsilon:  # 更新前损失函数值减去更新后的差大于阈值，继续循环
        x = xi
        cost1 = costi
    elif costi - cost1 > epsilon:  # 更新后损失函数值减去更新前的差大于阈值，说明步长过大，需要调小
        theta = theta * 0.3
    else:
        break
    xb = vstack((xb, x))
```

>画图：

‘g.’表示用绿色的点表示 2000 个随机点

‘r.’ 表示用红色的点表示每一次迭代过程收敛的点

```
pl.plot(all_points[:, 0], all_points[:, 1], 'g.')
pl.plot(xb[:, 0], xb[:, 1], 'r.')
pl.plot(xb[:, 0], xb[:, 1], 'k-')
pl.xlabel('c = (%.3f, %.3f)' % (c[0], c[1]))

pl.show()
```

>最终结果：centroid = c = （0.490，0.51）

4）使用随机梯度下降算法来计算质心：

随机梯度下降和梯度下降的区别就是，梯度下降是所有点都参与梯度更新，而随机梯度下降是每次循环随机选取点进行。

在梯度下降的基础上进行修改：

```
def stochasticgradient(c, r):
    dx = (c[0] - r[0]) / sum((c - r) ** 2) ** 0.5 #求x偏导数
    dy = (c[1] - r[1]) / sum((c - r) ** 2) ** 0.5  #求y偏导数
    s = (dx ** 2 + dy ** 2) ** 0.5
    dx = dx/s
    dy = dy/s
    return array([dx, dy])#得到梯度向量
```

此处 r 表示在所有点中随机选取的一个点，每次迭代用随机抽取点的方式。

from random import choice

r = choice(all_points)

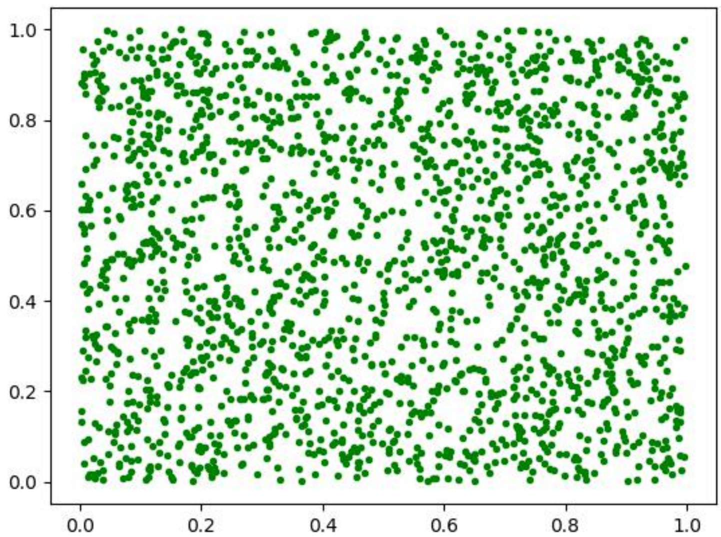
```
for i in range(loop_max):
    from random import choice
    r = choice(all_points)
    cost1 = cost(x, all_points)
    xi = x - theta * stochasticgradient(x, r)
    costi = cost(xi, all_points)
    if cost1 - costi > epsilon:
        x = xi
        cost1 = costi
    elif costi - cost1 > epsilon:
        theta = theta * 0.5
    else:
        break
    xb = vstack((xb, x))
```

可以看到，由于是随机选取的点，所以在找最优值的过程是曲折的。

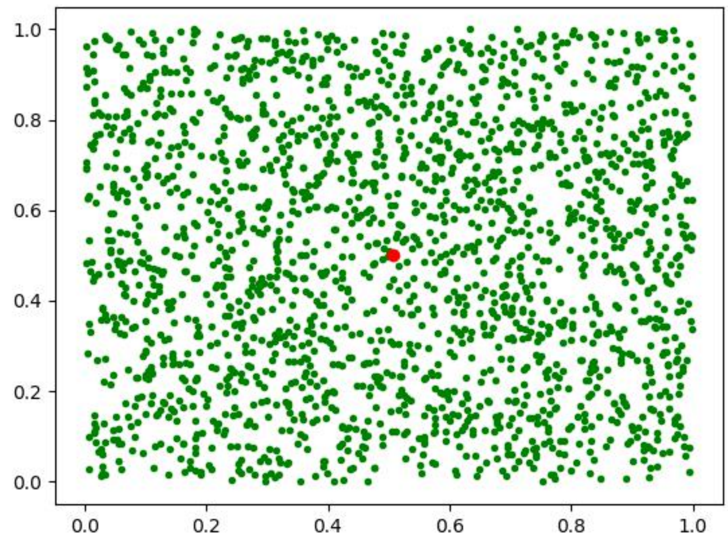
最终收敛：centroid = c = （0.476，0.532）

## 4. 实验结果与分析

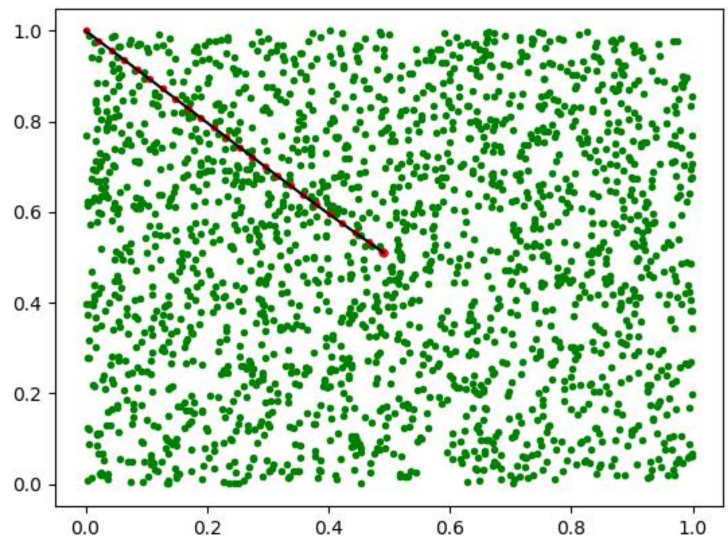
1）在二维单元图中生成 2000 个点，如下图所示：




2）在二维单元图中生成 2000 个点，并按照算术均值的方式求出其质心，如下图所示：

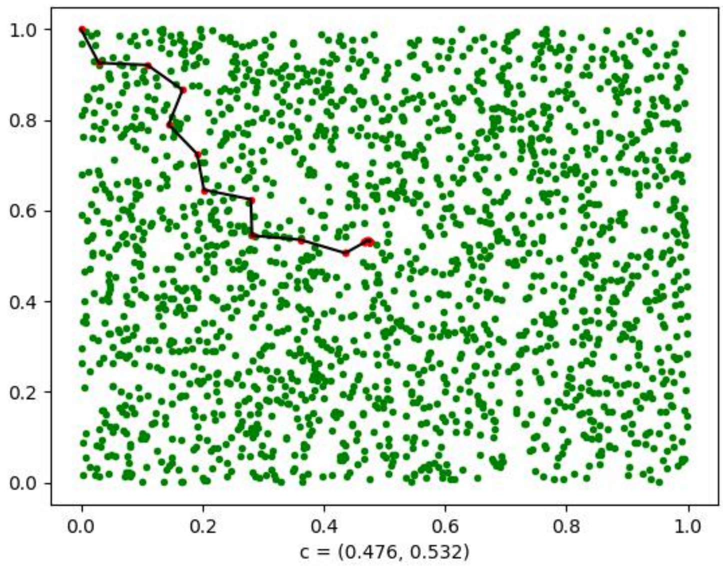



3）按照梯度下降算法求得质心，并绘出寻找路线，如下图所示：




c = (0.490, 0.511)

4）按照随机梯度下降算法求得质心，并绘出寻找路线，如下图所示：




## 5. 总结

本实验回顾了梯度下降算法以及随机梯度下降算法，并使用 python 语言进行了实现，从图示中体验两种算法的异同点，同时加深了对损失函数、学习率等相关概念的理解。