Assignment4实验报告

18364067 罗淦元

截止日期：10月25号晚11：59前

发送地址：314155841@qq.com

Problem1：实现二维Gibbs sampling（编程题）

（1）采样均值为[5,5]，协方差矩阵为[[1.0,0.9],[0.9,1.0]]二维高斯分布，采样10000个样本点

高维高斯分布进行吉布斯采样，首先需要得到两维度的均值和协方差矩阵，这在题目中已经给出。

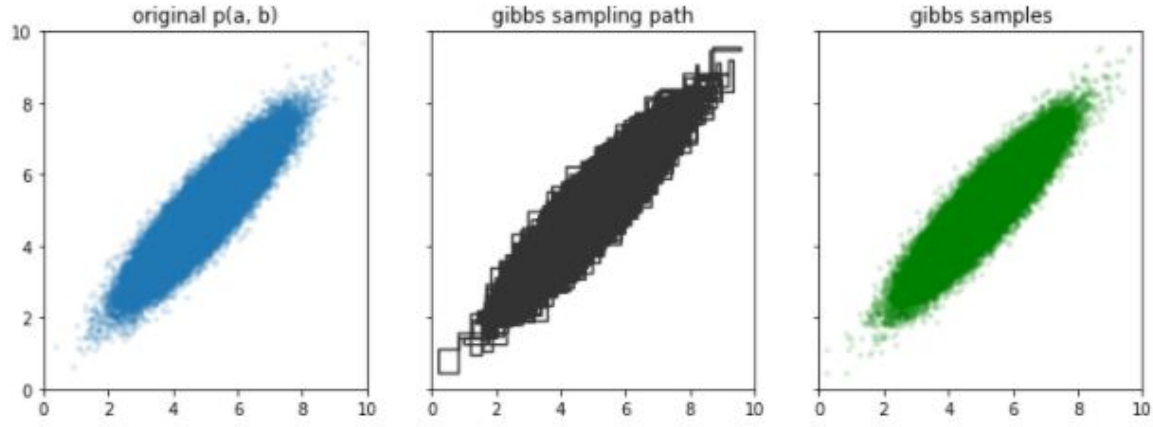
接着选取一个初始点。由于本题中已经大致知道了实际分布，于是选取了[5, 5]作为初始点。在测试中已经证明初始点并不会明显影响整体分布。每次采样时，总是不同维度交叉采样，代码如下。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib as mpl
from tqdm import tqdm

D = 2
N = 10000
Sigma = np.array([[1.0,0.9],[0.9,1.0]])          #协方差
mu =np.array([[5],[5]])                          #高斯分布均值

def gibbs_sample(sample_count):
    def conditional_gaussians(i,g):
        mu_i = mu[i] + Sigma[i,~i]*(1/Sigma[~i,~i])*(g - mu[~i]) #由于~0=-1,
~1=-2，在大小为2的sigma矩阵中恰好是另一元素下标
        Sigma_i = Sigma[i,i] - Sigma[i,~i]*(1/Sigma[~i,~i])*Sigma[~i,i]
        return np.sqrt(Sigma_i) * np.random.randn(1) + mu_i
    samples = np.zeros((D, sample_count))
    samples[:, 0] = [5,5]
    for ii in tqdm(range(1, sample_count)):
        samples[:, ii] = samples[:, ii - 1] # first set this sample equal to the
previous sample
        d = ii % D
        samples[d, ii] = conditional_gaussians(d,samples[~d, ii - 1]) # 轮换采样
    return samples

gibbs_sample = gibbs_sample(N)
```

输出结果为

```
[[5.         5.         5.19950158 ... 5.83048584 6.57484624 6.57484624]
 [5.         5.07502735 5.07502735 ... 6.59578721 6.59578721 7.01098939]]
```

在二维平面上有分布

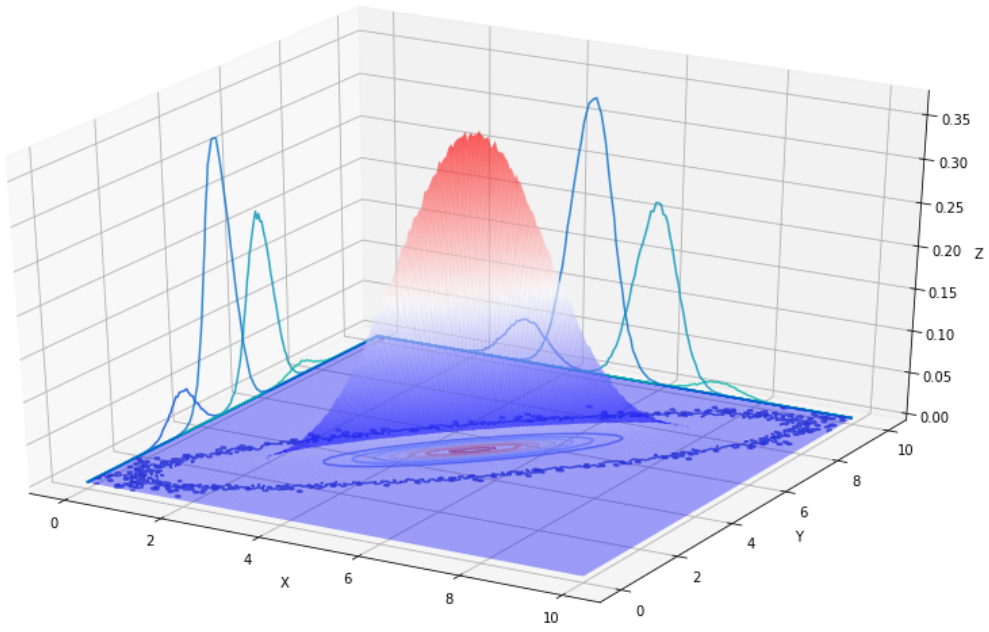


可以看到我们的采样路径总是沿着x轴或y轴移动，且总是交替进行。同时gibbs采样和实际分布大致上也是相同的。

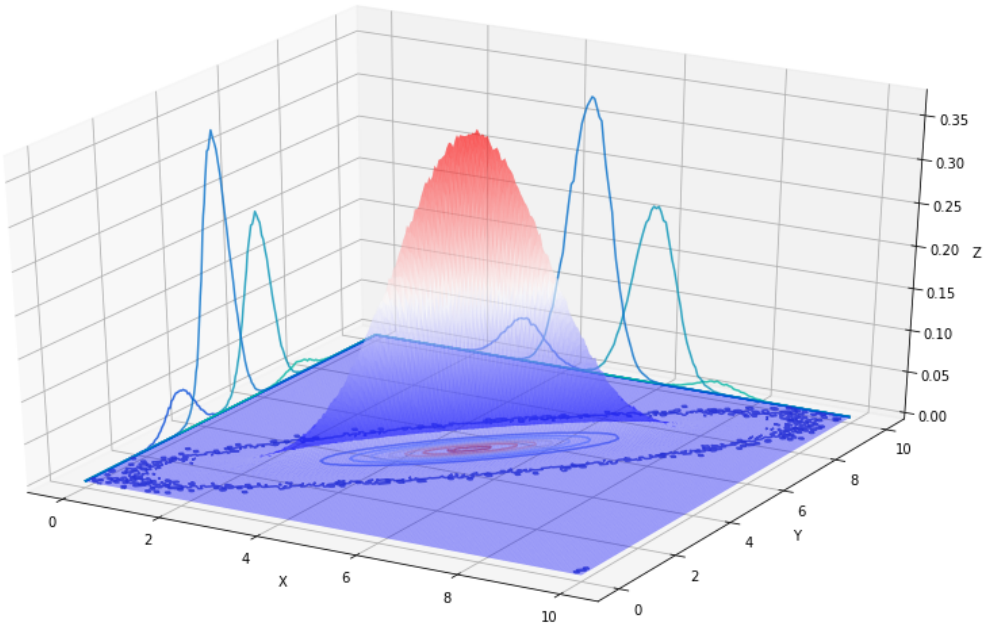
（2）画出采样的结果，并和原分布对比

为了图形平滑，本图分别使用了1000万个样本点。

1.根据二维高斯分布直接得到的样本点分布密度



2.通过吉布斯采样获得的样本点分布密度



4.绘制使用的完整代码

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib as mpl
from tqdm import tqdm

D = 2
N = 10000000
Sigma = np.array([[1.0,0.9],[0.9,1.0]])
mu =np.array([[5],[5]])                          #高斯分布均值

def generate_gaussians():
    R = np.linalg.cholesky(Sigma)
    s = np.dot(R, np.random.randn(D, N)) + mu
    return s

def gibbs_sample(sample_count):
    def conditional_gaussians(i,g):
        mu_i = mu[i] + Sigma[i,~i]*(1/Sigma[~i,~i])*(g - mu[~i]) #由于~0=-1,
~1=-2，在大小为2的sigma矩阵中恰好是另一元素下标
        Sigma_i = Sigma[i,i] - Sigma[i,~i]*(1/Sigma[~i,~i])*Sigma[~i,i]
        return np.sqrt(Sigma_i) * np.random.randn(1) + mu_i
    samples = np.zeros((D, sample_count))
    samples[:, 0] = [5,5]
    for i in tqdm(range(1, sample_count)):
        samples[:, i] = samples[:, i - 1] # first set this sample equal to the
previous sample
        d = i % D
        samples[d, i] = conditional_gaussians(d,samples[~d, i - 1]) # 轮换采样
    return samples

gibbs_sample = gibbs_sample(N)

num = 201
```

```
l = np.linspace(0,10,num)
X, Y = np.meshgrid(l, l)
Z = np.zeros((num,num), dtype=np.float32)
for ii in tqdm(range(N)):
        i = int(gibbs_sample[0][ii]*20)
        j = int(gibbs_sample[1][ii]*20)
        if i > 200 or j > 200:
            continue
        Z[i][j] += 1

Z = Z*(num-1)*(num-1)/100/N
fig = plt.figure(figsize=(12,7))
ax = Axes3D(fig)

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, alpha=0.4, cmap=mpl.cm.bwr)

cset = ax.contour(X,Y,Z, zdir='z',offset=0,cmap=cm.coolwarm,alpha=0.8)  #contour
画等高线
cset = ax.contour(X, Y, Z, zdir='x', offset=0,cmap=mpl.cm.winter,alpha=0.8)
cset = ax.contour(X, Y, Z, zdir='y', offset= 10,cmap= mpl.cm.winter,alpha=0.8)


ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()

true_distribution = generate_gaussians()

num = 201
l = np.linspace(0,10,num)
X, Y = np.meshgrid(l, l)
Z = np.zeros((num,num), dtype=np.float32)
for ii in tqdm(range(N)):
        i = int(true_distribution[0][ii]*20)
        j = int(true_distribution[1][ii]*20)
        if i > 200 or j > 200:
            continue
        Z[i][j] += 1

Z = Z*(num-1)*(num-1)/100/N
fig = plt.figure(figsize=(12,7))
ax = Axes3D(fig)

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, alpha=0.4, cmap=mpl.cm.bwr)

cset = ax.contour(X,Y,Z, zdir='z',offset=0,cmap=cm.coolwarm,alpha=0.8)  #contour
画等高线
cset = ax.contour(X, Y, Z, zdir='x', offset=0,cmap=mpl.cm.winter,alpha=0.8)
cset = ax.contour(X, Y, Z, zdir='y', offset= 10,cmap= mpl.cm.winter,alpha=0.8)


ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```

绘制时将区域[0, 10]（在前面我们可以看到分布主体在此区间内）划分为200x200个区域，然后通过每个区域内(样本点的数量/总数量/区域面积)近似得到概率密度。显然两者的分布是相似的。

Problem2：实现beam search（编程题）

作业要求：假设有一个包含10个单词的序列，字典中单词数为5，使用beam search搜索出最佳的三个序列。

```
data = [[0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1]]
```

1.代码实现

```
from math import log
from numpy import array
from numpy import argmax
from numpy import linalg


def beam_search_decoder(data, k):
    sequences = [[list(), 0]]
    for row in data:
        all_candidates = list()
        for i in range(len(sequences)):     #对于每个候选序列，获取其下个词典
            seq, score = sequences[i]
            for j in range(len(row)):
                candidate = [seq + [j], score + row[j]] #计算候选序列每下个单词的得分
                all_candidates.append(candidate)
        ordered = sorted(all_candidates, key=lambda tup :tup[1])
        sequences = ordered[-k:]        #选取k个最好的序列
    return sequences

data = [[0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1]]
```

```
norm = [linalg.norm(xi,ord=1) for xi in data]
for ii in range(len(data)):              #将data转换为条件概率的log
    data[ii] = [log(data[ii][jj]) - log(norm[ii]) for jj in range(len(data[1]))]
result = beam_search_decoder(data, 3)
for seq in result:
    print(seq)
```

输出结果：

```
[[4, 0, 4, 0, 4, 0, 4, 0, 3, 0], -11.209266437995304]
[[4, 0, 4, 0, 4, 0, 4, 0, 4, 1], -11.209266437995304]
[[4, 0, 4, 0, 4, 0, 4, 0, 4, 0], -10.986122886681093]
```