Assignment 3

作业要求

通过给定均值与标准差生成虚拟的大学男女生身高数据共 N个：

$$\mu_M = 176, \sigma_M = 8, \mu_F = 164, \sigma_F = 6$$

其中男女比例为3:2。

（1）用混合高斯模型对大学学生身高进行建模，并推导利用 EM 算法求解的公式（手写）

（2）编程实现 EM 算法对于以上5 个参数的估计并对比正确结果并讨论 EM 算法的优缺点。

作业内容

首先对EM算法进行推导，本质上是通过每部分的高斯结果来对整体的混合高斯进行优化。优化的推导公式如下图所示：

①初始 μ_m σ_m π_m μ_f, σ_f, π_f

②取一身高数据 x_i 属于2个高斯分布概率分别为 $\gamma(x_i, k) = \dfrac{\pi_k N(x_i \mid \mu_k \sigma_k^2)}{\sum_{j=1}^{2} \pi_j N(x_i \mid \mu_k \sigma_k^2)}$ $(k \in \{m, f\})$

③由γ更新参数: $\mu_k^{(t+1)} = \dfrac{\sum \gamma(x_i, k) \times x_i}{\sum \gamma(x_i, k)}$ $\sigma_k^2 = \dfrac{\sum \gamma(x_i, k) \times (x_i - \mu_k)^2}{\sum \gamma(x_i, k)}$ $\pi_k = \dfrac{1}{n} \sum \gamma(x_i, k)$

④重复第②步 至满足停止条件

根据上面的推导过程，编写自己的GMM和EM 优化算法，EM算法编成函数如下

```
def EM(data, max_epoch=1000):
    # 1 步
    n = len(data)
    # 计算中位数
    median_data = statistics.median(data)
    # 初始化参数
    mu1, mu2 = median_data+5, median_data-5   # 随机初始化均值
    sigma1_sq, sigma2_sq = np.random.rand(2) * 10 + 5   # 随机初始化方差
    pi1, pi2 = np.random.rand(2)
    pi1 /= (pi1 + pi2)  # 确保混合权重之和为1

    for _ in range(max_epoch):
        # 2 步
        gamma1 = pi1 * norm.pdf(data, mu1, np.sqrt(sigma1_sq))  #概率密度×混合权重
        gamma2 = pi2 * norm.pdf(data, mu2, np.sqrt(sigma2_sq))
        total_gamma = gamma1 + gamma2   #除以总值
        gamma1 /= total_gamma
        gamma2 /= total_gamma

        # 3 步
        mu1 = np.sum(gamma1 * data) / np.sum(gamma1)    #优化均值
        mu2 = np.sum(gamma2 * data) / np.sum(gamma2)
        sigma1_sq = np.sum(gamma1 * (data - mu1) ** 2) / np.sum(gamma1)     #
        sigma2_sq = np.sum(gamma2 * (data - mu2) ** 2) / np.sum(gamma2)
        pi1 = np.mean(gamma1)    #优化混合权重
```

```
        pi2 = np.mean(gamma2)

        # # 打印每次迭代的结果
        # print(
        #     f"Iteration {_ + 1}: mu1={mu1:.2f}, mu2={mu2:.2f}, sigma1^2=
{sigma1_sq:.2f}, sigma2^2={sigma2_sq:.2f}, pi1={pi1:.2f}")

    return np.array([mu1, mu2]), np.array([sigma1_sq, sigma2_sq]),
np.array([pi1, pi2])
```

同时编写辅助程序完成训练以及图像化显示

```
# 使用自己编写的EM函数
means, sigmas, weights=EM(data)
sigmas = np.sqrt(sigmas)
print("GMM Means:", means)
print("GMM Sigmas:", sigmas)
print("GMM Weights:", weights)
x = np.linspace(140, 210, 1000)
y_m = norm.pdf(x, 176, 8)*0.6
y_f = norm.pdf(x, 164, 6)*0.4
y_m_p = norm.pdf(x, means[0], sigmas[0])*weights[0]
y_f_p = norm.pdf(x, means[1], sigmas[1])*weights[1]
total_pdf = norm.pdf(x, means[0], sigmas[0])*weights[0] + norm.pdf(x, means[1],
sigmas[1])*weights[1]
plt.hist(data, bins=30, density=True, alpha=0.6, color='g')
plt.plot(x, total_pdf, '-k', label='Personal PDF')
plt.plot(x, y_m_p, 'b', label='pre male PDF')
plt.plot(x, y_f_p, 'r', label='pre female PDF')
plt.plot(x, y_m, '--b', label='real male PDF')
plt.plot(x, y_f, '--r', label='real female PDF')
plt.legend()
plt.title("Personal EM Fit")
plt.xlabel("Height (cm)")
plt.ylabel("Density")
plt.show()
```

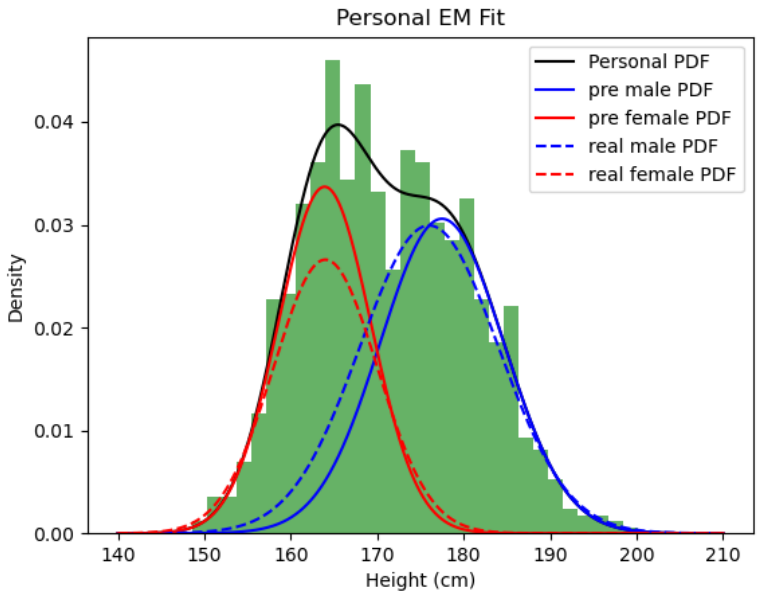
在使用自己的代码进行的同时，利用库函数`GaussianMixture`对GMM进行拟合，与自己编写的代码进行对比。

```
# 使用包装好的GMM
gmm = GaussianMixture(n_components=2, random_state=42)
gmm.fit(data.reshape(-1, 1))

# 获取GMM参数
means = gmm.means_.flatten()
sigmas = np.sqrt(gmm.covariances_.flatten())
weights = gmm.weights_.flatten()

# 可视化GMM拟合结果
x = np.linspace(140, 210, 1000)
logprob = gmm.score_samples(x.reshape(-1, 1))
responsibilities = gmm.predict_proba(x.reshape(-1, 1))
pdf = np.exp(logprob)
pdf_individual = responsibilities * pdf[:, np.newaxis]
```

```
# 将 pdf_individual 打包成元组列表，并按 means 的大小排序
sorted_data = sorted(zip(means, sigmas, weights, pdf_individual.T), key=lambda
x: x[0], reverse=True)
# 解压排序后的元组列表
means_sorted, sigmas_sorted, weights_sorted, pdf_individual_sorted =
zip(*sorted_data)

# 转换为 NumPy 数组
pdf_individual_sorted = np.array(pdf_individual_sorted).T
means = np.array(means_sorted).flatten()
sigmas = np.array(sigmas_sorted).flatten()
weights = np.array(weights_sorted).flatten()

print("GMM Means:", means)
print("GMM Sigmas:", sigmas)
print("GMM Weights:", weights)
y_m = norm.pdf(x, 176, 8)*0.6
y_f = norm.pdf(x, 164, 6)*0.4

plt.hist(data, bins=30, density=True, alpha=0.6, color='g')
plt.plot(x, pdf, '-k', label='GMM PDF')
plt.plot(x, pdf_individual[:, 0], 'b', label='male PDF')
plt.plot(x, pdf_individual[:, 1], 'r', label='female PDF')
plt.plot(x, y_m, '--b', label='real male PDF')
plt.plot(x, y_f, '--r', label='real female PDF')
plt.legend()
plt.title("GMM Fit")
plt.xlabel("Height (cm)")
plt.ylabel("Density")
plt.show()
```

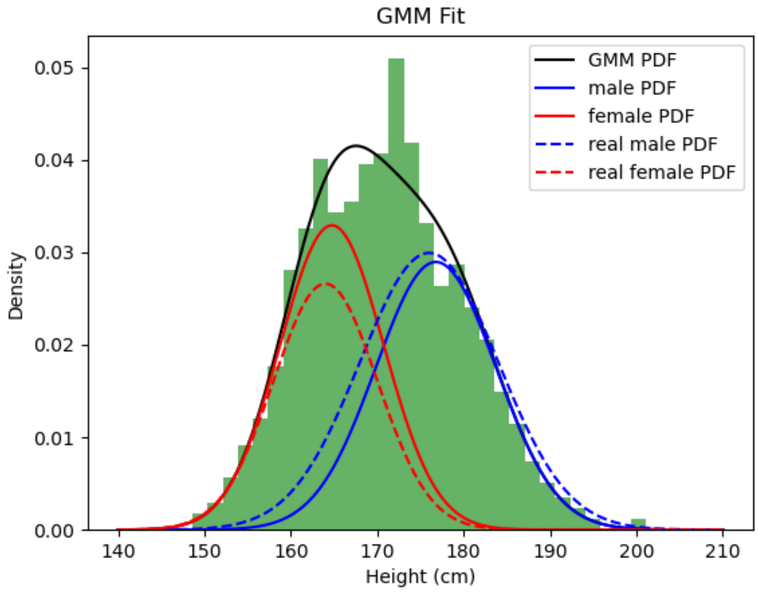
作业结果

分别对两种得到的结果和真实结果对比，自己编写的结果如下所示



```
GMM Means: [177.4857325  163.90880496]
GMM Sigmas: [7.14315643 5.36104504]
GMM Weights: [0.54738934 0.45261066]
```

包装好的GMM函数得到的结果如下所示




```
GMM Means: [176.78418956 164.74666735]
GMM Sigmas: [6.9525257  6.00966337]
GMM Weights: [0.50440486 0.49559514]
```

相较于正确值[176，164]、[8，6]、[0.6，0.4]。两种方法都能够较好的拟合上mean值，方差的拟合上也可以看出趋势，但效果相较于means较差，而在混合权重上的拟合是最差的，仅能展现出大概趋势，并不能准确拟合到对应值。

分析一下EM算法的优缺点：

优点

1. **灵活性高**：EM算法可以应用于各种概率模型，包括高斯混合模型、隐马尔科夫模型等。
2. **易于实现**：EM算法相对简单，易于编程实现，特别是对于混合模型的参数估计。
3. **参数估计精确**：在合适的初始条件和数据集上，EM算法能够提供精确的参数估计。

缺点

1. **收敛到局部最优**：EM算法可能会收敛到局部最优解，而非全局最优解。初始参数选择对结果影响很大。
2. **收敛速度慢**：对于某些数据集，EM算法的收敛速度可能较慢，特别是在接近收敛时，迭代更新步长变得非常小。
3. **需要多次初始化**：为了避免陷入局部最优解，通常需要多次随机初始化参数并选择最优结果，这增加了计算成本。
4. **计算复杂度高**：在处理大型数据集时，EM算法的计算复杂度较高，特别是在E步中计算期望时。
5. **依赖于模型假设**：EM算法假设数据符合某种特定的概率分布模型，如果模型假设不成立，估计结果可能不准确。

总结

EM算法在处理复杂概率模型和缺失数据方面具有显著优势，但其收敛到局部最优解、计算复杂度高以及对初始条件敏感等缺点也限制了其应用。实际应用中，常需要结合具体问题，并结合其他优化方法来改善EM算法的效果。