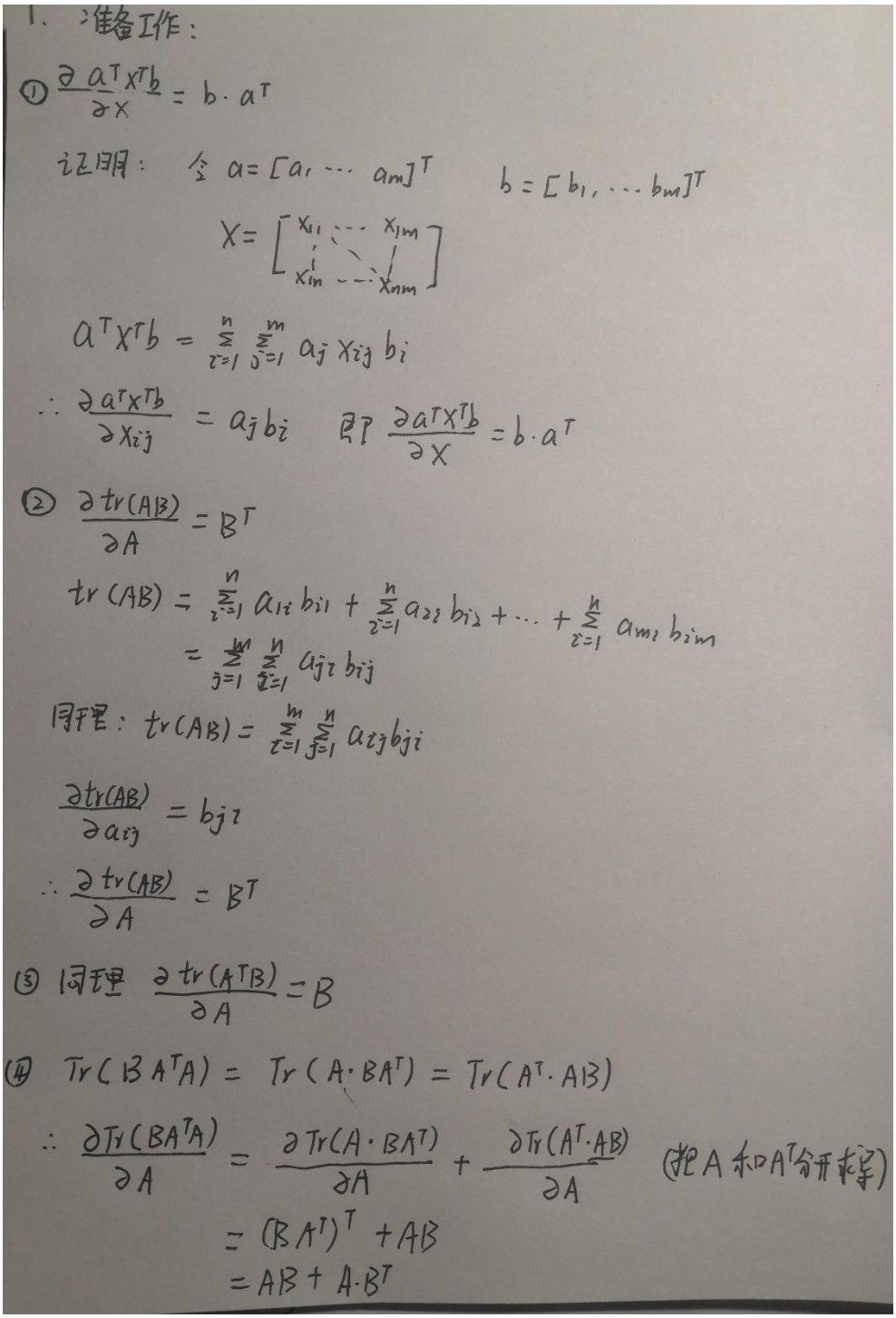
1. 证明



1. 准备工作：

① $\frac{\partial a^T X^T b}{\partial X} = b \cdot a^T$

证明：令 $a = [a_1 \cdots a_m]^T$ $b = [b_1, \cdots b_m]^T$

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

$a^T X^T b = \sum_{i=1}^{n} \sum_{j=1}^{m} a_j x_{ij} b_i$

$\therefore \frac{\partial a^T X^T b}{\partial x_{ij}} = a_j b_i$ 即 $\frac{\partial a^T X^T b}{\partial X} = b \cdot a^T$

② $\frac{\partial tr(AB)}{\partial A} = B^T$

$tr(AB) = \sum_{i=1}^{n} a_{1i} b_{i1} + \sum_{i=1}^{n} a_{2i} b_{i2} + \cdots + \sum_{i=1}^{n} a_{mi} b_{im}$

$= \sum_{j=1}^{m} \sum_{i=1}^{n} a_{ji} b_{ij}$

同理：$tr(AB) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ji}$

$\frac{\partial tr(AB)}{\partial a_{ij}} = b_{ji}$

$\therefore \frac{\partial tr(AB)}{\partial A} = B^T$

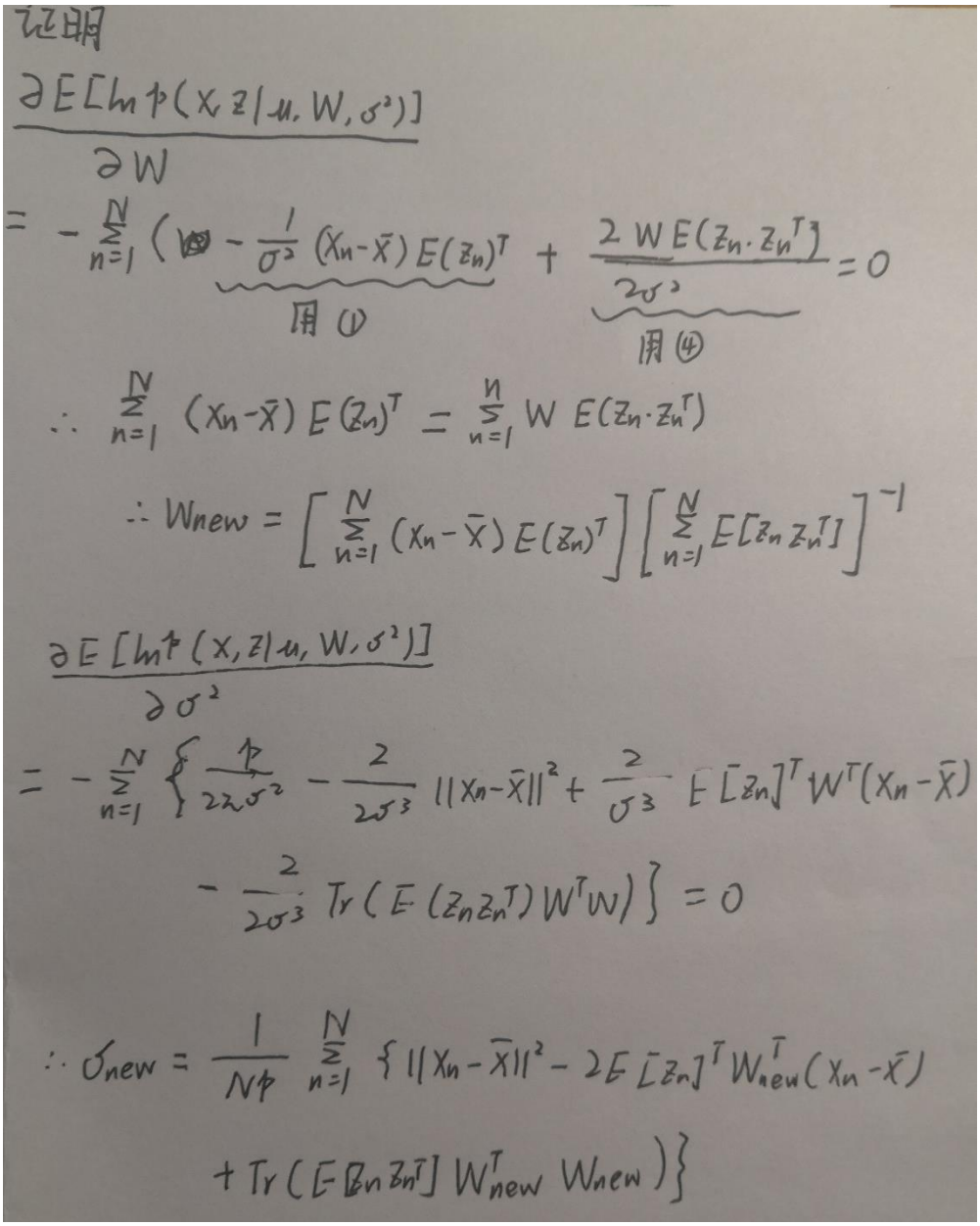
③ 同理 $\frac{\partial tr(A^T B)}{\partial A} = B$

④ $Tr(BA^TA) = Tr(A \cdot BA^T) = Tr(A^T \cdot AB)$

$\therefore \frac{\partial Tr(BA^TA)}{\partial A} = \frac{\partial Tr(A \cdot BA^T)}{\partial A} + \frac{\partial Tr(A^T \cdot AB)}{\partial A}$ (把A和$A^T$分开来算)

$= (BA^T)^T + AB$

$= AB + A \cdot B^T$




## 2. 上机实验

### 1) 基于 SVD 的 PCA

#### i. 数据中心化处理

```
X = get_data()  # (10304, 400)
X = X.astype(np.float32)
p, m = X.shape
x_mean = np.mean(X, axis=1).reshape((p, 1))
X = X - x_mean
```

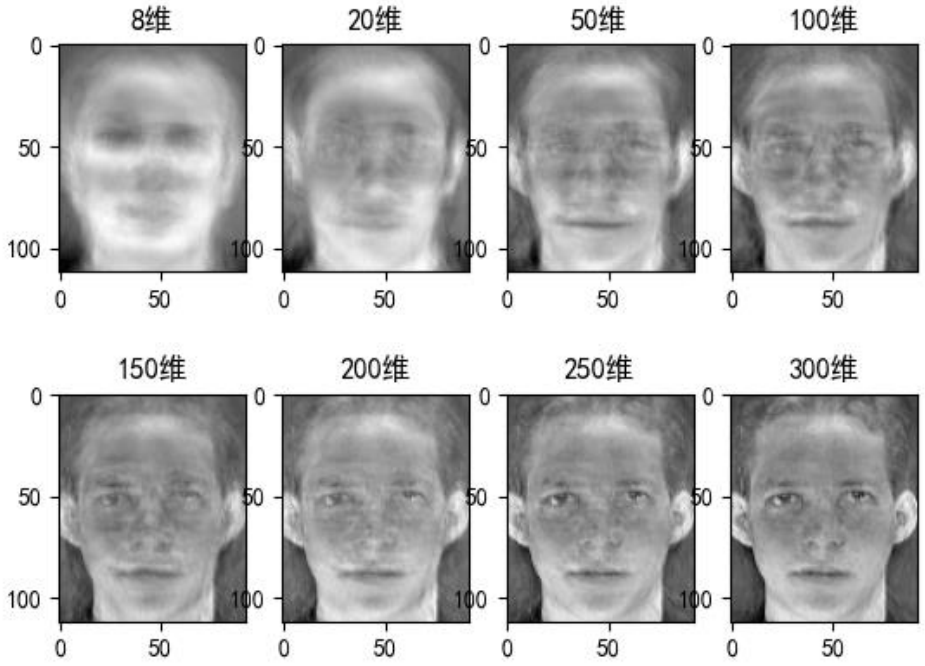
ii. 基于 SVD 计算

```
lamda, V = np.linalg.eig(A)  # A的特征值以列的形式显示
for i in range(m):
    V[:, i:i+1] /= np.dot(V[:, i:i+1].T, V[:, i:i+1])
sorted_indices = np.argsort(-lamda)
chance = [8, 20, 50, 100, 150, 200, 250, 300]  # 降维列表
data = []  # 用来保留降维后重构的数据
for k in chance:
    print("降到{}维，信息量保留为{}".format
          (k, np.sum([lamda[i] for i in range(k)] / np.sum(list(lamda)))))
    U = np.ones((10304, k))
    for i, j in zip(sorted_indices[0:k], range(k)):
        U[:, j:j + 1] = (X @ V[:, i:i + 1]) / np.sqrt(lamda[i])
    Z = U.T @ X
    np.savetxt("result/{}维W值.txt".format(k), U[:, 0:k])
    np.savetxt("result/{}维Z值.txt".format(k), Z)
    X1 = U @ Z + x_mean
    X1 = X1.astype(np.int64)
    data.append(X1[:, 0])
```

将重构回去的数据保存在 data 中

iii. 结果

```
D:\Python38\python.exe D:/文件仓库/降维/dim
降到8维，信息量保留为0.5618687868118286
降到20维，信息量保留为0.7001464366912842
降到50维，信息量保留为0.8160502314567566
降到100维，信息量保留为0.8958966732025 15
降到150维，信息量保留为0.9285839200019836
降到200维，信息量保留为0.9520465135574341
降到250维，信息量保留为0.9653908014297485
降到300维，信息量保留为0.9785487651824951
```




由图可以看出，图片在降到八维时只保留了 56%的信息量，重构效果不好，在信息量保留 85%以上时，图片可以看得比较清楚，在信息量保留 95%时，图片与原图基本相同。

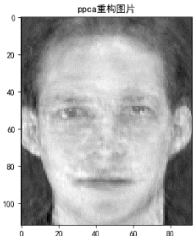
存储的 Z 和 W 看附件。

**2) 最大似然 PCA**

**i. 主要代码**

```python
def p_pca(data, k):
    p, m = data.shape
    mu = np.mean(data, axis=1).reshape((p, 1))
    data = data - mu
    S = (data @ data.T) / m  # 协方差矩阵
    vector_U, value, vector_V = np.linalg.svd(data)
    sort_indices = np.argsort(-value)
    I = np.eye(k)
    sigma2 = sum(value[sort_indices[k:]]) / (p - k)
    diag_sorted = np.diag(value[sort_indices[:k]])
    W = vector_U[:, 0:k] @ ((diag_sorted - sigma2 * I) ** 0.5)
    Z = np.zeros((k, m))
    for i in range(m):
        Z[:, i:i+1] = np.linalg.inv(W.T @ W + sigma2 * I) @ W.T @ (data[:, i:i+1] - mu)
    recon_data = (W @ Z + mu)
    return Z, recon_data
```

ii. 结果




此图是信息量保存为 90%时，可以看出重构效果还不错

3）基于 EM 算法的 PCA

i. 主要代码

```python
def EM_pca(data, k):
    p, m = data.shape
    # 初始化
    W = np.random.randn(p, k)
    Z = np.random.randn(k, m)
    x_mean = np.mean(data, axis=1).reshape(p, 1)
    for epoch in range(50):
        print(epoch)
        # E步
        x_mean = np.mean(data, axis=1).reshape(p, 1)
        data = data - x_mean
        Z = np.linalg.inv(W.T @ W) @ W.T @ data
        # M步
        W = data @ Z.T @ np.linalg.inv(Z @ Z.T)
    recon_data = (W @ Z + x_mean)
    return Z, recon_data
```

ii. 结果

此图也是在信息量保留 90%时重构的结果，可以看出与原图只有很小的差距