Least Squares and *Total* Least Squares (Principal Components)

1. Use the function `np.random.multivariate_normal()` to create samples $(z_i, y_i) \in \mathbb{R}^2$ for $i = 1, \ldots, 100$ from a 2-dimensional multivariate normal distribution with mean $\mu := (0, 0)$ and covariance

$$\Sigma := \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}.$$

2. Solve the least squares problem

$$\min_{c \in \mathbb{R}} \sum_{i=1}^{100} (cz_i - y_i)^2$$

   and plot the model $f(z) = cz$ and the data points into one plot.

3. Now consider the 2×100 matrix $A = [v_1, \cdots, v_n]$ with columns $v_i := (z_i, y_i)^\top \in \mathbb{R}^2$ and compute an SVD $A = U\Sigma V^\top$ of it. Draw the lines through the vectors u_1 and u_2 into the same plot. These are the principal components that explain most of the variance.

Some Background

A column in A contains the measured features (e.g., age and height) for a particular sample (e.g., a person) and a row contains all measured values for a particular feature. Thus, let $a_i \in \mathbb{R}^n$ denote a row of A, then by assuming that the mean $a_i^\top \mathbf{1}$ is zero (without loss of generality, otherwise center the data) for all features, we have

$$\frac{1}{n-1} a_i^\top a_j = \begin{cases} \text{“statistical variance in feature } i\text{”} & i = j \\ \text{“statistical covariance between feature } i \text{ and } j\text{”} & i \neq j. \end{cases}$$

Furthermore we observe that the matrix

$$\frac{1}{n-1} AA^\top = \frac{1}{n-1} \left(a_i^\top a_j \right)_{ij}$$

contains these covariances (it is therefore often called *sample covariance matrix*) and using the SVD $A = U\Sigma V^\top$ we find

$$\frac{1}{n-1} AA^T = \frac{1}{n-1} U \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_r^2 \end{pmatrix} U^T = \frac{1}{n-1} \sum_{i=1}^{r} \sigma_i^2 u_i u_i^T.$$

The first few summands explain most of $\frac{1}{n-1} AA^T$, i.e., the sample covariance matrix, and the singular vectors $u_1, \ldots, u_r$ are called principal components.

Geometrically we have the following interpretation:

$$A = \begin{array}{c} m \text{ feats} \\ \downarrow \end{array} \overset{\xrightarrow{n \text{ samples}}}{\begin{pmatrix} | & & | & & | \\ a_1 & \cdots & a_i & \cdots & a_n \\ | & & | & & | \end{pmatrix}} = U\Sigma V^T = \underbrace{\begin{pmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{pmatrix}}_{\text{orthonormal basis}} \underbrace{(\Sigma V^T)}_{\text{coordinates of } a_i \text{ in terms of this basis}}$$

Thus, each sample $a_i \in \mathbb{R}^m$ is a linear combination of $u_1, \ldots, u_m$ with coefficients $(\Sigma V^T)_i$.

Relation to “total least squares”: The least squares problem

$$\min_{x \in \mathbb{R}} \|Ax - b\|^2,$$

where we want to minimize the error in the *dependent* variables, can be reformulated as the constrained optimization problem

$$\min_{r,x\in\mathbb{R}} \|r\|^2$$
$$s.t.\ \ r = Ax - b.$$

If we also want to encounter errors in the *independent* variables we arrive at the problem

$$\min_{r,s,x\in\mathbb{R}} \| \begin{pmatrix} r \\ s \end{pmatrix} \|^2$$
$$s.t.\ \ (A+s)x = b + r.$$

This problem is called the total *least squares problem* and errors on both dependent and independent variables are considered. One can show that the solution of this problem is the low-rank approximation which we yield from cropping the singular value decomposition. See for details: `https://eprints.soton.ac.uk/263855/1/tls_overview.pdf`

Solution:

```python
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt



if __name__ == "__main__":
    # generate random sample (multivariate normal distribution)
    mean = [0, 0]; cov = [[1, .7], [.7, 1]]
    Nrandom = 100
    x = np.random.multivariate_normal(mean, cov, Nrandom).T

    # PCA: compute svd
    U, sigma, V = linalg.svd(x)

    # least squares
    c, residuals, rank, singular_val = np.linalg.lstsq(x[0][np.newaxis].T,
                                                         x[1], rcond=None)

    # PLOT
    fig, ax = plt.subplots()

    # plot random sample as dots
    plt.plot(x[0], x[1], "o", alpha=.6, zorder=1)

    # plot least squares fit: f(z) = t * c
    t = np.linspace(-3,3,20)
    plt.plot(t, t*c, zorder=4)
    print("slope of the least squares fit:", c)

    # plot first principal component = line spanned by first column of U
    # each point (x,y) on this line is orthogonal (-U[0,1],U[0,0])
    plt.plot(t, t*(U[0,1]/U[0,0]), zorder=4, color='red')
    print("slope of PCA line:", U[0,1]/U[0,0])

    # plot styling
    plt.grid( alpha = 0.25)
    plt.xlim(xmin=-3, xmax=3)
    plt.ylim(ymin=-3, ymax=3)
```

```python
    ax.set_aspect('equal')
    plt.legend(["Samples", "Linear Least Squares", "First Principal Component"])
    plt.show()
```