# 統計的機械学習レポート

工学部電子情報工学科 3 年　03190449 堀 紡希

5 月 21 日

宿題 1 $\tilde{M}$ が直交行列になることを示す

観測信号の球状化により

$$
\begin{aligned}
I_d &= \frac{1}{n}\sum_{i'=1}^{n} \tilde{x}_{i'}\tilde{x}_{i'}^T \\
&= \frac{1}{n}\sum_{i'=1}^{n} C^{-\frac{1}{2}} x_{i'} x_{i'}^T C^{-\frac{1}{2}T} \\
&= \frac{1}{n}\sum_{i'=1}^{n} C^{-\frac{1}{2}} M s_{i'} s_{i'}^T M^T C^{-\frac{1}{2}T} \\
&= \frac{1}{n}\sum_{i'=1}^{n} \tilde{M} s_{i'} s_{i'}^T \tilde{M}^T \\
&= \tilde{M}(\frac{1}{n}\sum_{i'=1}^{n} s_{i'} s_{i'}^T)\tilde{M}^T \\
&= \tilde{M}\tilde{M}^T
\end{aligned}
$$

また原信号に対する仮定より

$$
\begin{aligned}
I_d &= \frac{1}{n}\sum_{i'=1}^{n} s_{i'} s_{i'}^T \\
&= \frac{1}{n}\sum_{i'=1}^{n} M^{-1} x_{i'} x_{i'}^T M^{-1T} \\
&= \frac{1}{n}\sum_{i'=1}^{n} M^{-1} C^{\frac{1}{2}} \tilde{x_{i'}} \tilde{s_{i'}^T} C^{-\frac{1}{2}T} M^{-1T} \\
&= \tilde{M}^{-1}(\frac{1}{n}\sum_{i'=1}^{n} s_{i'} s_{i'}^T)\tilde{M}^{-1T} \\
&= \tilde{M}^{-1}\tilde{M}^{-1T}
\end{aligned}
$$

逆行列を取ると

$$
I_d = \tilde{M}^T \tilde{M}^{-1}
$$

となり直交行列であることが示された。

宿題 2 Python で $g(s) = s^3, g(s) = \tanh(s)$ に対する射影追跡のニュートンアルゴリズムを実装した。

```
import numpy as np
from scipy.linalg import sqrtm

def generate_data(n=1000):
    x = np.concatenate([np.random.rand(n, 1), np.random.randn(n, 1)], axis=1)
    x[0, 1] = 6   # outlier
    x = (x - np.mean(x, axis=0)) / np.std(x, axis=0)  # Standardization
    M = np.array([[1, 3], [5, 3]])
    x = x.dot(M.T)
    x = np.linalg.inv(sqrtm(np.cov(x, rowvar=False))).dot(x.T).T
    return x

def generize(X):
    n = len(X)
    d = len(X[0])
    X = np.reshape(X, [2, 1000])
    H = np.identity(n)-np.ones((n, n))*(1/n)
    A = (1/n)*np.dot(np.dot(X, H), np.dot(H, X.T))
    A = np.linalg.inv(power05(A))
    A = np.dot(A, np.dot(X, H))
    return A
def power05(A):
    l, P = np.linalg.eig(A)
    D = np.dot(np.dot(np.linalg.inv(P), A), P)
    n = len(D)
    for i in range(0, n):
        D[i][i] = D[i][i]**0.5
    return np.dot(np.dot(P, D), np.linalg.inv(P))

def newton(g, m, Xt):
    Xt = Xt.T
    n = len(Xt)
    d = len(Xt[0])
    b = np.ones(d)
    B = np.zeros((n, d))
    for k in range(0, m):
        sum1 = 0
        sum2 = np.zeros(d)
        for i in range(0, n):
            sum1 += diff(g, (np.dot(b, Xt[i])))
            sum2 += (1/n)*g(np.dot(b, Xt[i]))*np.array(Xt[i])
        sum1 /= n
        b += sum1 * b - sum2
        sum3 = np.zeros(d)
        for i in range(0, k-1):
            sum3 += np.dot(b, B[i])*B[i]
        b -= sum3
```

```
        b = b/np.linalg.norm(b, ord = 2)
        B[k] = b
    return b

def diff(f, x):
    eps = 0.0001
    return (f(x+eps)-f(x))/eps
def cubic(x):
    return x**3
def tanh(x):
    return np.tanh(x)

import matplotlib.pyplot as plt
X = generate_data()
b = newton(cubic, 100, generize(X))
print(b)
for x in generize(X).T:
    plt.scatter(x[0], x[1])
x = np.linspace(-5, 5, 100)
plt.plot(x, x*(b[1]/b[0]))
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.show()

import matplotlib.pyplot as plt
X = generate_data()
b = newton(tanh, 100, generize(X))
print(b)
for x in generize(X).T:
    plt.scatter(x[0], x[1])
x = np.linspace(-5, 5, 100)
plt.plot(x, x*(b[1]/b[0]))
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.show()
```

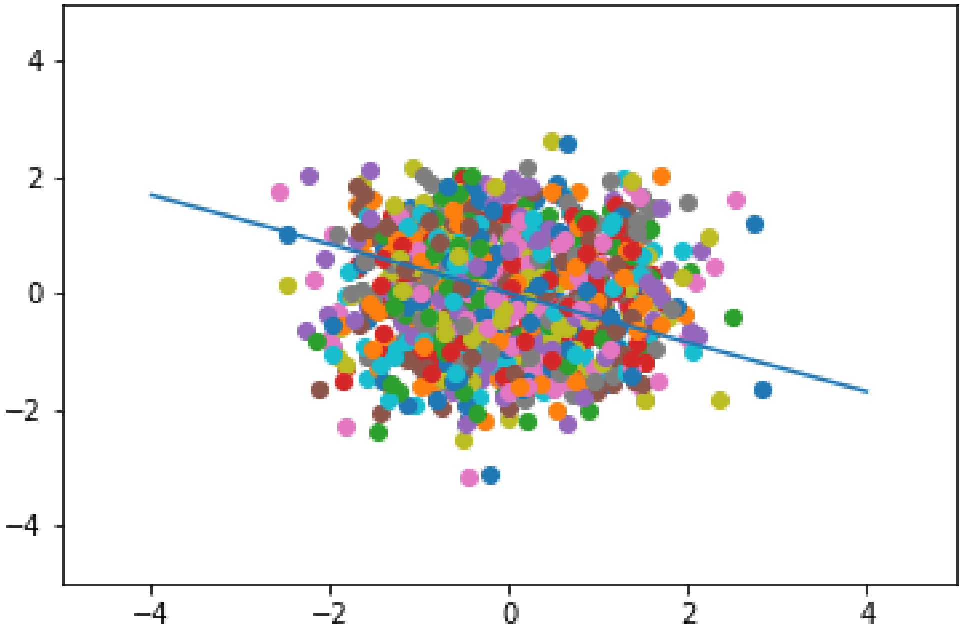


Figure 1: $g(s) = s^3$

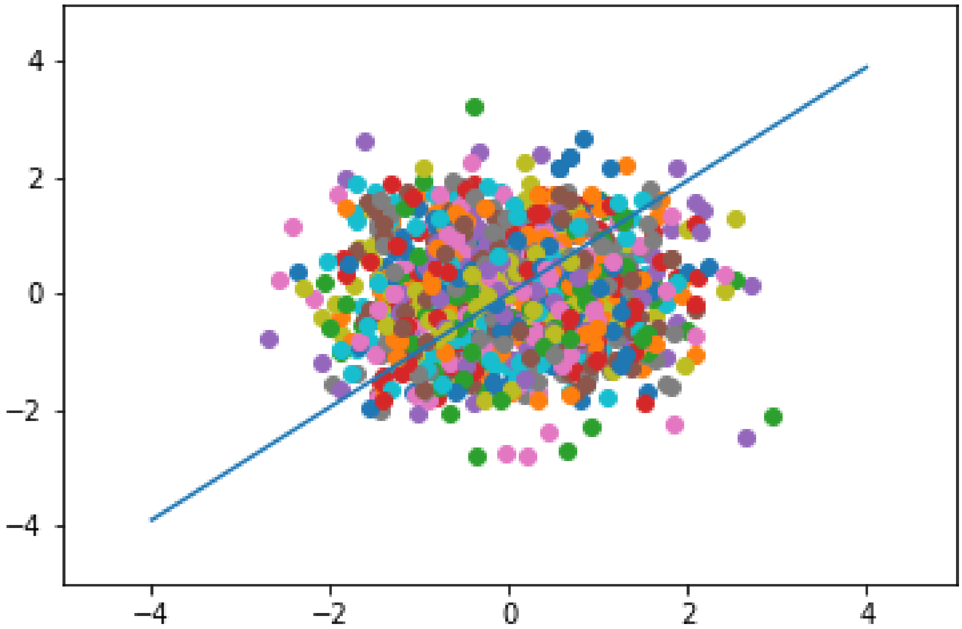


Figure 2: $g(s) = \tanh(s)$

結果　実装するとこのようになった。$\tanh(s)$ の方が外れ値に引きずられておらず、外れ値に対して寛容だと言うことができるだろう。