SML HW5

29-176004 奥村 恭平 [*,†]

June 5, 2018

宿題1: $\widetilde{M}$ が直交行列になることを示せ

$$\widetilde{M}^\top \widetilde{M} = I_d$$

を示せば十分．$\widetilde{M} = C^{-\frac{1}{2}} M$ より，

$$\begin{aligned}
\widetilde{M}^\top \widetilde{M} &= M^\top C^{-1} M \\
&= M^\top \left(\frac{1}{n} \sum_i x_i x_i^\top \right)^{-1} M \\
&= M^\top \left(\frac{1}{n} X X^\top \right)^{-1} M \quad (X := (x_1, \ldots, x_n) \text{ とした}) \\
&= n M^\top \left(X X^\top \right)^{-1} M
\end{aligned}$$

ここで，$M^\top \left(XX^\top \right)^{-1} M = (M^{-1} X X^\top (M^\top)^{-1})^{-1} = (SS^\top)^{-1} = (\sum_i s_i s_i^\top)^{-1}$ (ただし，$S = (s_1, \ldots, s_n)$ とした) であること，および，$\frac{1}{n} \sum_i s_i s_i^\top = I_d$(原信号に対する仮定) より，

$$\begin{aligned}
\widetilde{M}^\top \widetilde{M} &= n M^\top \left(X X^\top \right)^{-1} M \\
&= n \left(\sum_i s_i s_i^\top \right)^{-1} \\
&= \left(\frac{1}{n} \sum_i s_i s_i^\top \right)^{-1} \\
&= I_d
\end{aligned}$$

となる． □

*E-mail: kyohei.okumura@gmail.com
†東京大学大学院 経済学研究科 M2

宿題 2

$g(s) = s^3$ に対する射影追跡の近似ニュートンアルゴリズムを実装した．言語は python．以下はシミュレーション結果の図．非ガウス方向を検出できていることがわかる．

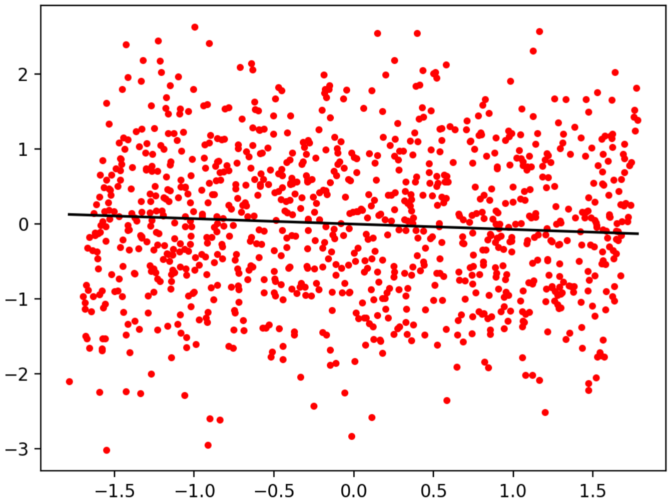

Figure 1: シミュレーション実行結果

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


def centering_sphering(X):
    '''
    X: d x n matrix
    '''
    n = X.shape[1]
    H = np.eye(n) - np.ones((n,n))/n
    XH = np.dot(X, H)
    temp = sqrtm(np.linalg.inv(np.dot(XH, XH.T)/n))
    X_tilde = np.dot(temp, XH)
    return X_tilde


def approx_newton(X, Nlim=50):
    '''
    X should be normalized.
    X: d x n matrix
    '''
    n = X.shape[1]
    b = np.array([1,0])
    threshold = 1e-08
    diff = np.inf
    n_loop = 1

    while n_loop < Nlim:
        #print(b)
        b_prev = b
        sum = 0
        for i in range(n):
            sum += X[:, i] * (np.dot(b, X[:, i]) ** 3)
        b = 3 * b - sum/n
        b = b / np.linalg.norm(b)
```

```python
        diff = np.linalg.norm(b - b_prev)
        if (diff < threshold):
            break
        else:
            n_loop += 1

    if n_loop == Nlim:
        print('may not be converged')

    return b


def line(b, X):
    x_min = np.min(X[0])
    x_max = np.max(X[0])
    x = np.linspace(x_min, x_max, 1000)
    return [x, (b[1]/b[0])*x]


def plot(x1, line=None):
    x = x1[0]
    y = x1[1]
    plt.plot(x, y, 'ro', ms=3, label='class1')

    if not (line is None):
            plt.plot(line[0], line[1], 'k-', ms=5)

    #plt.xlim(np.min(x)-1, np.max(x)+1)
    #plt.ylim(np.min(y)-1, np.max(y)+1)

    plt.show()

# simulation
N = 1000

## original sigmal
s_gauss = np.random.randn(N)*2 + 3
s_uniform = np.random.rand(N) * 3 - 2
S = np.array([s_gauss, s_uniform])

## transformation matrix
M = np.array([[1,3],[5,1]])

## observed sigmal
X = np.dot(M, S)

X_tilde = centering_sphering(X)
b = approx_newton(X_tilde)

plot(X_tilde, line(b, X_tilde))
```