Nonparametric Statistics: Homework 8

蒋翌坤 20307100013

Solution to Problem 1

The risk of `MLE` is

$$R(\hat{\theta}_{\mathrm{MLE}}, \theta) = \sum_{i=1}^{n} \mathbb{E}(\hat{\theta}_{\mathrm{MLE},i} - \theta_i)^2 = \sum_{i=1}^{n} \mathbb{E}_\theta(Z_i - \theta_i)^2 = \sum_{i=1}^{n} \mathrm{V}(Z_i) = n = 1000$$

The risk of the estimator $\tilde{\theta} = (bZ_1, bZ_2, \ldots, bZ_n)$ is

$$\begin{aligned} R(\tilde{\theta}, \theta) &= \sum_{i=1}^{n} \mathbb{E}(\tilde{\theta}_i - \theta_i)^2 = \sum_{i=1}^{n} \mathbb{E}_\theta(bZ_i - \theta_i)^2 = \sum_{i=1}^{n} b^2\mathrm{V}(Z_i) + (1-b)^2\theta_i^2 \\ &= nb^2 + (1-b)^2 \sum_{i=1}^{n} \frac{1}{i^4} = 1000b^2 + (1-b)^2 \sum_{i=1}^{1000} \frac{1}{i^4} \end{aligned}$$

Plot the risk of $\tilde{\theta}$ as a function of b, we get Figure 1.1.

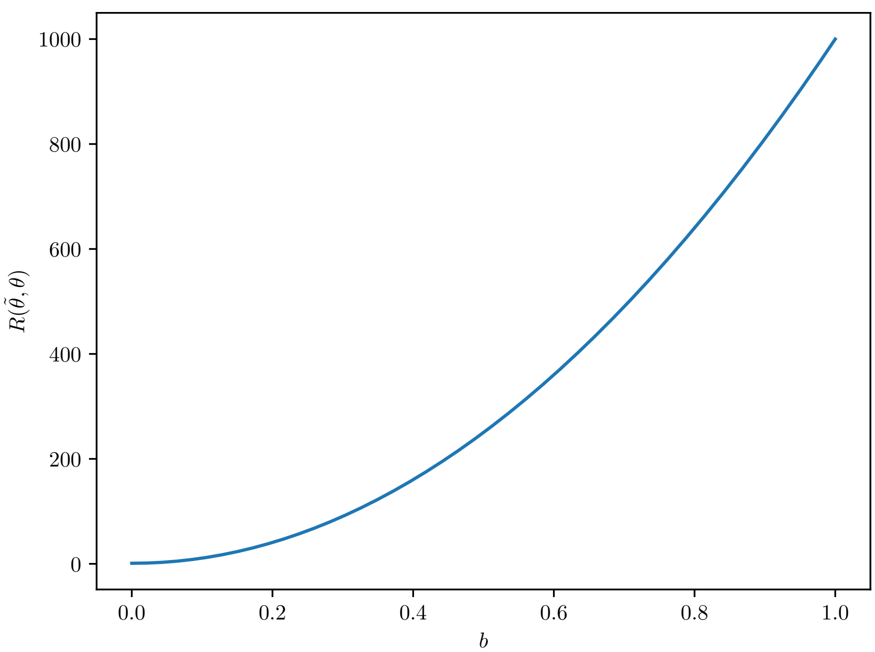

Figure 1.1: Risk of $\tilde{\theta}$ as a function of b

To find optimal value b_*, we take partial derivative of b.

$$\frac{\partial R(\tilde{\theta}, \theta)}{\partial b} = 2000b + 2(b-1)\sum_{i=1}^{1000} \frac{1}{i^4} = 0 \quad \Rightarrow \quad b_* = \frac{\sum_{i=1}^{1000} \frac{1}{i^4}}{1000 + \sum_{i=1}^{1000} \frac{1}{i^4}} \approx 1.081 \times 10^{-3}$$

In the simulation, we repeat the experiment for 10000 times. Mean of $\hat{b}$ is 1.699×10^{-2}, which is more than 10 times higher to b_*. However, standard deviation of $\hat{b}$ is 2.463×10^{-2}. More than Half (5006) of $\hat{b}$ is 0, which is less than b_*. The histogram of $\hat{b}$ is shown in Figure 1.2.

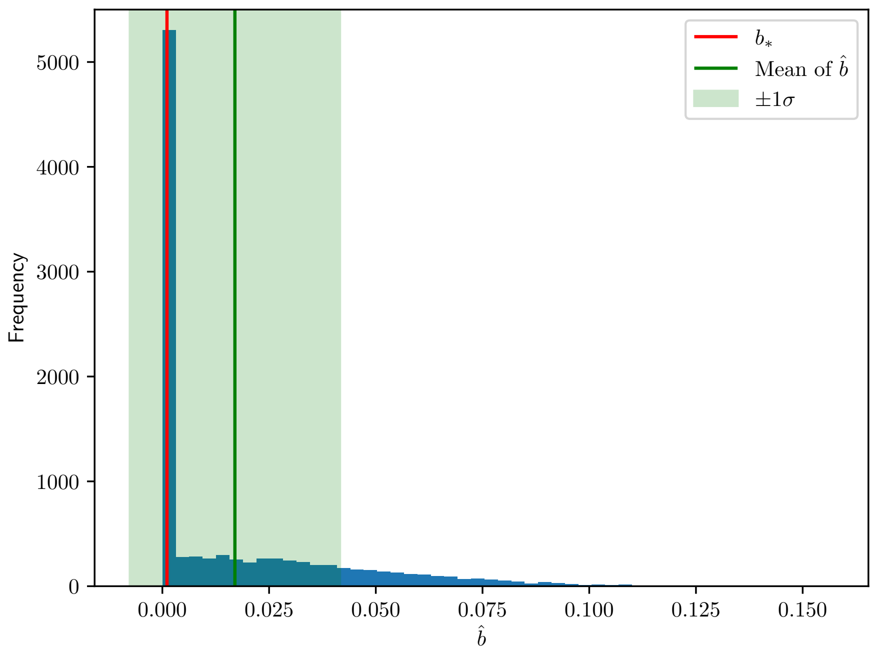

Figure 1.2: histogram of $\hat{b}$

Pinsker bound is

$$\frac{\sigma^2 c^2}{\sigma^2 + c^2} = \frac{1000 \times \sum_{i=1}^{1000} \frac{1}{i^4}}{1000 + \sum_{i=1}^{1000} \frac{1}{i^4}} \approx 1.081$$

The risk of `MLE` is 1000 which is almost 1000 times higher than Pinsker bound. The risk of the James-Stein estimator from simulation has mean of 1.942 which is close to Pinsker bound and standard deviation of 1.900.

Solution to Problem 2

a

The risk of this estimator is

$$R(\hat{\theta},\theta)=\mathbb{E}(\hat{R})=\mathbb{E}\Big(n\sigma_n^2+2\sigma_n^2\sum_{i=1}^n D_i+\sum_{i=1}^n(\hat{\theta}_i-Z_i)^2\Big) \tag{1}$$

Let $p_{i1}=P(Z_i<-\lambda)=\Phi\Big(\dfrac{-\lambda-\theta_i}{\sigma_n}\Big)$, $p_{i2}=P(-\lambda\le Z_i\le\lambda)=\Big[\Phi\Big(\dfrac{\lambda-\theta_i}{\sigma_n}\Big)-\Phi\Big(\dfrac{-\lambda-\theta_i}{\sigma_n}\Big)\Big]$, $p_{i3}=P(Z_i>\lambda)=\Big[1-\Phi\Big(\dfrac{\lambda-\theta_i}{\sigma_n}\Big)\Big]$, we have

$$D_i=\frac{\partial(\hat{\theta}_i-Z_i)}{\partial Z_i}=\begin{cases}-2(Z_i+\lambda)-1 & \text{if } Z_i<-\lambda\\ -1 & \text{if } -\lambda\le Z_i\le\lambda\\ 2(Z_i-\lambda)-1 & \text{if } Z_i>\lambda\end{cases}=2(|Z_i|-\lambda)I(|Z_i|>\lambda)-1$$

$$\mathbb{E}(D_i)=p_{i1}\Big(-2(\theta_i+\lambda)\Big)+p_{i3}\Big(2(\theta_i-\lambda)\Big)-1=2(p_{i3}-p_{i1})\theta_i-(p_{i1}+p_{i3})\lambda-1 \tag{2}$$

$$(\hat{\theta}_i-Z_i)^2=\begin{cases}\Big((Z_i+\lambda)^2+Z_i\Big)^2 & \text{if } Z_i<-\lambda\\ Z_i^2 & \text{if } -\lambda\le Z_i\le\lambda\\ \Big((Z_i-\lambda)^2-Z_i\Big)^2 & \text{if } Z_i>\lambda\end{cases}=\Big((|Z_i|-\lambda)^2I(|Z_i|>\lambda)-|Z_i|\Big)^2$$

$$\begin{aligned}\mathbb{E}\Big((\hat{\theta}_i-Z_i)^2\Big)&=p_{i1}\mathbb{E}\Big((Z_i+\lambda)^2+Z_i\Big)^2+p_{i2}\mathbb{E}\Big(Z_i^2\Big)+p_{i3}\mathbb{E}\Big((Z_i-\lambda)^2-Z_i\Big)^2\\&=(p_{i1}+p_{i3})\Big(\mathbb{E}(Z_i^4)+(6\lambda^2+4\lambda)\mathbb{E}(Z_i^2)+\lambda^4\Big)\\&\qquad+(p_{i1}-p_{i3})\Big(2(2\lambda+1)\mathbb{E}(Z_i^3)+2\lambda^2(2\lambda+1)\mathbb{E}(Z_i)\Big)+\mathbb{E}(Z_i^2)\\&=(p_{i1}+p_{i3})\Big(\theta_i^4+6\theta_i^2\sigma_n^2+3\sigma_n^4+(6\lambda^2+4\lambda)(\sigma_n^2+\theta_i^2)+\lambda^4\Big)\\&\qquad+(p_{i1}-p_{i3})\Big(2(2\lambda+1)(\theta_i^3+3\theta_i\sigma_n^2)+2\lambda^2(2\lambda+1)\theta_i\Big)+(\sigma_n^2+\theta_i^2)\end{aligned} \tag{3}$$

Substitute (2) and (3) into (1), we have

$$\begin{aligned}R(\hat{\theta},\theta)&=n\sigma_n^2+2\sigma_n^2\sum_{i=1}^n\mathbb{E}(D_i)+\sum_{i=1}^n\mathbb{E}(\hat{\theta}_i-Z_i)^2\\&=4\sigma_n^2\sum_{i=1}^n(p_{i3}-p_{i1})\theta_i-2\lambda\sigma_n^2\sum_{i=1}^n(p_{i3}+p_{i1})+\sum_{i=1}^n\theta_i^2\\&\qquad+\sum_{i=1}^n(p_{i1}+p_{i3})\Big(\theta_i^4+6\theta_i^2\sigma_n^2+3\sigma_n^4+(6\lambda^2+4\lambda)(\sigma_n^2+\theta_i^2)+\lambda^4\Big)\\&\qquad+\sum_{i=1}^n(p_{i1}-p_{i3})\Big(2(2\lambda+1)(\theta_i^3+3\theta_i\sigma_n^2)+2\lambda^2(2\lambda+1)\theta_i\Big)\end{aligned} \tag{4}$$

b

To find the optimal λ, we use Equation (4) to find $\arg_\lambda \min R(\hat{\theta}, \theta)$.

From Problem (1), we know $n = 1000, \theta_i = 1/i^2, \sigma_n^2 = 1$, solve Equation (5) numerically, we have $\lambda^* = 10$ and $R(\hat{\theta}, \theta)$=1.082. This risk is smaller than the risk of the James-Stein estimator, which is 1.942.

Now that $\theta = (10, \ldots, 10, 0, \ldots, 0)$, solve Equation (5) numerically, we have $\lambda^* = 7.31$ and $R(\hat{\theta}, \theta)$=.488.598. This risk is much greater than the risk of the James-Stein estimator, which is 1.942.

Solution to Problem 3

Though the data is not a regular design, using Gram-Schmidt orthogonalization to construct a basis is too complicated. Therefore, we transfrom the data to $[0,1]$: $x = \dfrac{x - \min x}{\max x - \min x}$ and assume the transformed data is a regular design. By calculation, we find $\hat{\sigma}^2 = 0.0285$ and $\hat{J} = 116$ that minimize the risk estimator $\hat{R}(J)$. The function estimation and 95% confidence band is shown in Figure 3.1. When calculating confidence band, we use the James-Stein estimator as $\hat{b}$. The comparison between `REACT` and local linear smoothing is shown in Figure 3.2.

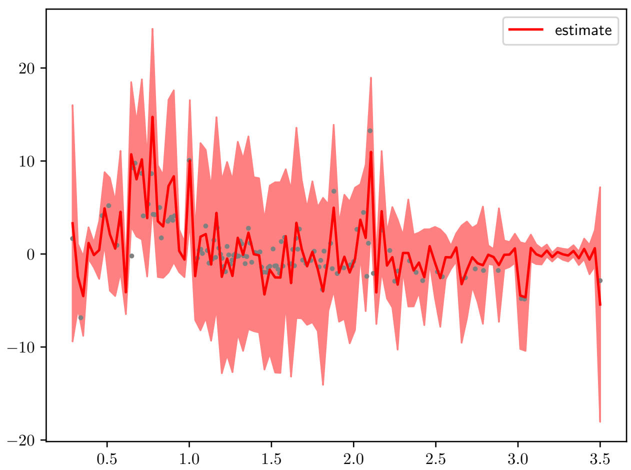

Figure 3.1: `REACT` Regression model and 95% confidence band

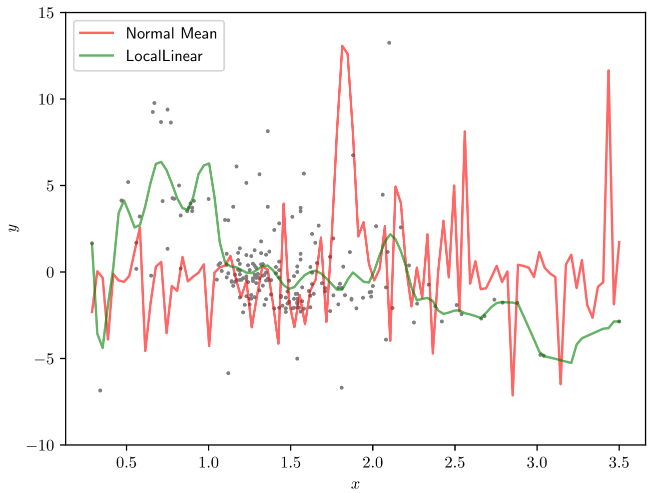

Figure 3.2: Comparison between `REACT` and local linear smoothing

Appendix

Code

The following code can also be found on https://thisiskunmeng.github.io/nonparametric/hw8.html

Code for Problem 1

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
plt.rcParams['text.usetex'] = True

n = 1000

def risk(b):
    global n
    return n * b**2 + (1-b)**2 * np.sum([1/(i + 1)**4 for i in range(n)])

b = np.linspace(0, 1, 10000)
plt.plot(b, risk(b))
plt.xlabel(r'$b$')
plt.ylabel(r'$R(\tilde{\theta}, \theta)$')
plt.savefig('./hw8/1a.pdf')
```

```python
b_opt = np.sum([1/(i + 1)**4 for i in range(n)]) / (n + np.sum([1/(i + 1)**4 for i in range(n)]))
print(b_opt)
```

```
0.0010811530762850031
```

```python
theta = np.array([1/(i + 1)**2 for i in range(n)])

sim_time = 10000
b_hat = np.zeros(sim_time)
for i in range(sim_time):
    np.random.seed(i)
    z = stats.norm.rvs(size = 1000)
    b_hat[i] = max(1 - n/np.sum(z**2), 0)
print(np.mean(b_hat))
print(np.std(b_hat, ddof = 1))
```

```
0.016992243506765856
0.024626931833334986
```

```
plt.hist(b_hat, bins = 50)
plt.vlines(b_opt, 0, 5500, colors = 'red', label=r'$b_{*}$')
plt.vlines(np.mean(b_hat), 0, 5500, colors = 'green', label=r'$\textrm{Mean of }\hat{b}$')
plt.fill_between(
    [
        np.mean(b_hat) - np.std(b_hat, ddof = 1),
        np.mean(b_hat) + np.std(b_hat, ddof = 1)
    ],
    0, 5500, color = 'green', alpha = 0.2, label=r'$\pm 1 \sigma$')
plt.ylim(0, 5500)
plt.xlabel(r'$\hat{b}$')
plt.ylabel('Frequency')
plt.legend()
plt.savefig('./hw8/1b.pdf')
```

```
print(np.count_nonzero(b_hat == 0))
r_b = risk(b_hat)
print(np.mean(r_b))
print(np.std(r_b, ddof = 1))
```

```
5006
1.9416713498972331
1.9000687346000968
```

Code for Problem 2

```
from scipy.stats import norm

def risk_t(lam, theta_i, sigma=1):
    p_i3p_i1minus = 1 - norm.cdf((lam-theta_i)/sigma) - norm.cdf((-lam-theta_i)/sigma)
    p_i3p_i1plus = 1 - norm.cdf((lam-theta_i)/sigma) + norm.cdf((-lam-theta_i)/sigma)
    part1 = 4 * sigma**2 * np.sum(p_i3p_i1minus * theta_i)
    part2 = -2 * lam * sigma**2 * np.sum(p_i3p_i1plus)
    part3 = np.sum(theta_i**2)
    part4 = np.sum(
        p_i3p_i1plus *
        (theta_i**4 + 6 * theta_i**2 * sigma**2 + 3 * sigma**4 +
         (6 * lam**2 + 4 * lam) * (sigma**2 + theta_i**2) + lam**4)
    )
    part5 = -np.sum(
        p_i3p_i1minus *
        (2 * (2*lam+1) * (theta_i**3 + 3 * theta_i * sigma**2) +
         2 * lam**2 * (2*lam+1) * theta_i)
    )
```

```
    return part1 + part2 + part3 + part4 + part5

l = np.linspace(5, 100, 500)
y = np.zeros(500)
for i in range(500):
    y[i] = risk_t(l[i], theta)
l_opt = l[np.argmin(y)]
risk_opt = y[np.argmin(y)]
print(l_opt); print(risk_opt)
```

```
10.140280561122244
1.0823232333783048
```

```
theta = np.zeros(1000)
theta[0:10] = 10
l = np.linspace(5, 20, 500)
y = np.zeros(500)
for i in range(500):
    y[i] = risk_t(l[i], theta)
l_opt = l[np.argmin(y)]
risk_opt = y[np.argmin(y)]
print(l_opt); print(risk_opt)
```

```
7.314629258517034
488.59798149054404
```

Code for Problem 3

- Note that code for local linear smoothing is the same as that in homework 7, which is in `estimate` module.

```
from estimate import Estimate
from tqdm import tqdm
class NormalMean(Estimate):
    def __init__(self, y: pd.Series, x: pd.Series):
        super().__init__(y, x)
        self.data = None
        self.method = 'Normal Mean'
        self.process_same_value()
        self.x_old = self.x
        self._scale_x()
        self.z = self.get_z(self.x.values)
        self.b = (1 - self.n / np.sum(self.z**2)) * np.ones(self.n)
        self.jn = int(np.ceil(self.n/4))
        self.j_hat = None
        self.var = self.variance_estimate()
        self.optimal_j_hat()
```

```python

    def process_same_value(self):
        self.data = pd.DataFrame({"x": self.x, "y": self.y})
        self.x = pd.Series(self.data["x"].value_counts().index.sort_values().values)
        self.y = pd.Series([self.data[self.data["x"] == i]["y"].mean() for i in self.x])
        self.n = len(self.x)

    def _scale_x(self):
        self.x = (self.x - self.x.min()) / (self.x.max() - self.x.min())

    @staticmethod
    def cosine_basis(i: int):
        def f(xxx: int | float | np.ndarray) -> np.ndarray:
            if i == 1:
                return np.ones_like(xxx)
            return np.array([np.sqrt(2) * np.cos((i-1) * np.pi * xxx)])
        return f

    def get_z(self, x: np.ndarray) -> np.ndarray:
        z = np.zeros(self.n)
        for j in range(self.n):
            z[j] = 1/self.n * np.sum(self.cosine_basis(j+1)(x) * self.y.values)
        return z

    def estimate(self, x) -> float | np.ndarray:
        if isinstance(x, np.ndarray):
            return np.array([self.estimate(x[i]) for i in range(len(x))])
        s = 0
        for i in range(self.j_hat):
            s += self.z[i] * self.cosine_basis(i+1)(x)
        return s

    def variance_estimate(self) -> float:
        return 1/(self.n - self.jn) * np.sum([self.z[i]**2 for i in range(self.n-self.jn, self.n)])

    def risk(self, j: int):
        part1 = j * self.var / self.n
        part2 = np.sum([np.max([(self.z[j]**2 - self.var/self.n), 0]) for j in range(j, self.n)])
        return part1 + part2

    def optimal_j_hat(self):
        j_hat = np.argmin([self.risk(j) for j in range(1, self.n)])
        self.j_hat = j_hat
        return j_hat

    def l_x(self, x) -> np.ndarray:
        li_x = np.zeros(self.n)
        for i in range(self.n):
            li_x[i] = 1/self.n * np.sum(
                np.array(
                    [float(self.b[j] * self.cosine_basis(j+1)(x) * self.cosine_basis(j+1)(self.x.values[i]))
                     for j in range(self.n)]
                )
            )
        return li_x

    def confidence_band(self):
        self.c = 1.96
        sigma = np.sqrt(self.variance_estimate())
```

```python

        def band(x):
            s = self.c * np.sqrt(sigma) * np.linalg.norm(self.l_x(x))
            return [float(self.estimate(x) - s),
                    float(self.estimate(x) + s),
                    float(self.estimate(x))]

        return band

    def draw_confidence_band(self, flag=True, ax=None):
        if ax is None:
            ax = plt
        band = self.confidence_band()
        x = np.linspace(self.x_old.min(), self.x_old.max(), 100)
        new_x = (x - self.x_old.min()) / (self.x_old.max() - self.x_old.min())
        y = np.array([band(i) for i in tqdm(new_x, desc=self.method + " confidence band")])
        ax.plot(self.x_old, self.y, "o", color="grey", ms=2)
        ax.fill_between(x, y[:, 0], y[:, 1], alpha=.5, color='red')
        ax.plot(x, y[:, 2], color="red", label="estimate")
        ax.legend()
        return [x, y[:, 0], y[:, 1]]

nm = NormalMean(target, x_al)
print(nm.variance_estimate())
print(nm.j_hat)
nm_cb = nm.draw_confidence_band()
plt.savefig('./hw8/3a.pdf')
```

```
0.028482335453141022
116
```

```python
from estimate import LocalLinear

ll = LocalLinear(target, x_al)
ll_cv = ll.plot_cv_score(0.015, 0.1)
ll.set_optimal_parameter(ll_cv[0, np.argmin(ll_cv[1])])

fig, ax = plt.subplots()
ax.set_ylim(-10, 15)
ax.scatter(x_al, target, s=2, color='grey')
ax.plot(nm_cb[0], nm.estimate(nm_cb[0]), color='red', alpha=0.6, label='Normal Mean')
ax.plot(nm_cb[0], ll.estimate(nm_cb[0]), color='green', alpha=0.6, label='LocalLinear')
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.legend()
plt.savefig('./hw8/3b.pdf')
```