Q-1 (a)

$$E[(Y-X'b)^2] = \langle Y - X'b, Y - X'b \rangle$$

$$= \langle U + X'\beta_0 - X'b, U + X'\beta_0 - X'b \rangle$$

$$= \langle U + X'(\beta_0 - b), U + X'(\beta_0 - b) \rangle$$

$$= \langle U, U \rangle + \langle X'(\beta_0 - b), U \rangle + \langle X'(\beta_0 - b), U \rangle + \langle X'(\beta_0 - b), X'(\beta_0 - b) \rangle$$

$$= E[U^2] + 2 \langle X'(\beta_0 - b), U \rangle + \langle X'(\beta_0 - b), X'(\beta_0 - b) \rangle$$

$$= E[U^2] + 2E[(X'(\beta_0 - b)U] + E[(X'(\beta_0 - b))^2]$$

$$= E[U^2] + 2E[((\beta_0 - b)'XU)] + E[X'(\beta_0 - b)X'(\beta_0 - b)]$$

when $X'(\beta_0 - b)U$ is scaler, so its transpose is equal to itself.

$(AB)' = B'A'$

Similarly, $(X'(\beta_0 - b))' = (\beta_0 - b)'X$

$$E[(Y - X'b)] = E[U^2] + 2(\beta_0 - b)'E[XU] + (\beta_0 - b)'E[XX'](\beta_0 - b)$$

$\square$

(b)

We can write quadratic form,

$$(\beta_0 - b)'E[XX'](\beta_0 - b) = E[(\beta_0 - b)'XX'(\beta_0 - b)]$$
$$= E[[(\beta_0 - b)'X]^2$$

It is given $E[((\beta_0 - b)'X)^2] \geq 0$ & is strictly positive if $\beta_0 \neq b$. So,

$$E[(Y - X'b)^2] = E[U^2] + 2(\beta_0 - b)'E[XU] + E[((\beta_0 - b)'X)^2]$$

$$= E[U^2] + 0 + \underbrace{E[((\beta_0 - b)'X)^2]}_{\geq 0}$$

$$\geq E[U^2]$$

As we know, $E[((\beta_0 - b)'X)^2] > 0$ if $\beta_0 \neq b$. So, strict inequality for $\beta_0 \neq b$.

As we see squared error of a linear predictor of Y is $(Y - X'b)^2$ and its expectation is $E[(Y - X'b)^2]$. Also, the expectation is minimized when $b = \beta_0$ i.e when we use population linear predictor.

(c)

$$V(Y) = E[(Y - E[Y])^2]$$

$$= E[(E^*[Y|X] + U - E[Y])^2]$$

$$= E[(E^*[Y|X] - E[Y] + U)^2]$$

$$= E\{(E^*[Y|X] - E[Y])^2 + 2(E^*[Y|X] - E[Y])(U) + U^2\}$$

Since, $E^*[Y|X]$ is projection, so projection error is orthogonal to $E^*[Y|X]$. Also, $E[U] = 0$, so cross term is 0.

Also, $E[Y] = E[E^*[Y|X]]$

$$Var(Y) = E[(E^*[Y|X] - E[E^*[Y|X]])^2] + E[U^2]$$

$$Var(Y) = Var(E^*[Y|X]) + E[U^2]$$

but $E[U^2] = Var(Y - E^*[Y|X])$. So,

$$Var(Y) = Var(E^*[Y|X]) + Var(Y - E^*[Y|X])$$

So, total variance of Y variable due to linear component/linear projection plus the part not explained by linear projection.

(d)

Since X_K is scalar RV,

$$E^*[Y|X_K] = \alpha_K + \beta_K X_K$$

Since X_K, X_ℓ are uncorrelated & $E^*[Y|X_K]$ is linear function of X_K,

$$Cov(E^*[Y|X_K], X_\ell) = 0$$

but since two are uncorrelated, projection of this onto X_ℓ,

$$\hat{E}\left[\hat{E}[Y|X_k]\,|X_\ell\right] = E\left[\hat{E}[Y|X_k]\right] = E[Y]$$

$$E[UX_\ell] = E[YX_\ell] - \sum_{k=1}^{K} E\left[\hat{E}[Y|X_k]\,|X_\ell\right] + (K-1)E[Y]E[X_\ell]$$

Since $\hat{E}[Y|X_K]$ is lin,

$$E\left[\hat{E}[Y|X_k]\,X_\ell\right] = E\left[\hat{E}[Y|X_k]\right]E[X_\ell]$$
$$= E[Y]\,E[X_\ell]$$

for $k=\ell$,

$$E\left[\hat{E}[Y|X_k]\,X_\ell\right] = E[YX_\ell]$$

$\Rightarrow$

$$E[UX_\ell] = E[YX_\ell] - E[YX_\ell] - (K-1)E[Y]E[X_\ell] + (K-1)E[Y]E[X_\ell]$$

$$E[UX_\ell] = 0 + 0 = 0 \quad \forall \ell$$

(e)

Note that when X_K is one regressor

then

$$E^*[Y \mid X_K] = \alpha_K + \beta_K X_K$$

where $\alpha_K = E[Y] - \beta_K E[X_K]$

$$\beta_K = \frac{Cov(Y, X_K)}{V(X_K)}$$

So,

$$E^*[Y \mid X_1, \dots X_N] = \sum_{k=1}^{N} E^*[Y \mid X_K] - (K-1) E[Y]$$

$$= E[Y] + \sum_{k=1}^{N} \left(E^*[Y \mid X_K] - E[Y] \right)$$

$$= E[Y] + \sum_{k=1}^{N} \left(\alpha_K + \beta_K X_K - E[Y] \right)$$

$$= E[Y] + \sum_{k=1}^{N} \left(E[Y] - \beta_k E[X_k] + \beta_k X_k - E[Y] \right)$$

$$= E[Y] + \sum_{k=1}^{K} \left(\beta_k \left(X_k - E[X_k] \right) \right)$$

$$= E[Y] + \sum_{k=1}^{K} \frac{Cov(Y, X_k)}{Var(X_k)} \left(X_k - E[X_k] \right)$$

Now,

$$Var(\hat{E}[Y \mid X_1, \dots X_k]) = Var\left(\sum_{k=1}^{K} \frac{C(Y, X_k)}{Var(X_k)} X_k \right)$$

as variance of expectation is 0.

$$= \sum_{k=1}^{K} \frac{C(Y, X_k)^2}{Var(X_k)^2} Var(X_k) +$$

$$+ \sum_{k \neq i} \frac{Cov(X, X_k) Cov(Y, X_i)}{Var(X_k) Var(X_i)} C(X_k, Y_i)$$

$c(X_k, X_i) = 0$, so,

$$Var(\hat{E}[Y|X_1, \dots X_N]) = \sum_{k=1}^{N} \frac{Cov(Y, X_k)^2}{Var(X_k)}$$

Now, $1 - \frac{Var(U)}{Var(Y)} = \frac{Var(Y) - Var(U)}{Var(Y)}$

we showed in part (c),

$$Var(Y) - Var(U) = Var(\hat{E}[Y|X])$$

$$1 - \frac{Var(U)}{Var(Y)} = \frac{Var(\hat{E}[Y|X_1, \dots X_N])}{Var(Y)}$$

$$= \sum_{k=1}^{N} \frac{Cov(Y, X_k)^2}{Var(Y)\, Var(X_k)}$$

$$= \sum_{k=1}^{N} \rho_K^2 \qquad \square$$

Q-2 (a)

$Y_i \sim N(\mu, \sigma_i^2)$

choose $c = \begin{pmatrix} 1/N \\ 1/N \\ \vdots \\ 1/N \end{pmatrix}$

then $c'Y = \frac{1}{N}Y_1 + \frac{1}{N}Y_2 + \cdots \frac{1}{N}Y_N = \bar{Y}$

So, there exists such a c.

(b)

$MSE(\hat{u}) = Bias^2 + Var(\hat{u})$

$E[\hat{u}] = c' E[Y] = c' \mu \iota_N$

$Var(\hat{u}) = c' Var(Y) c = c' diag(\sigma_1^2, \cdots \sigma_N^2) c$

$Bias(\hat{u}) = \mu c' \iota_N - \mu \iota_N$

$$MSE(\hat{u}) = c' \operatorname{diag}(\sigma_1^2, \cdots \sigma_N^2) c + (\mu c' \iota_N - \mu)$$

Taking matrix derivative,

$$\frac{\partial MSE(\hat{u})}{\partial c} = 2(\operatorname{diag}(\sigma_1^2, \cdots \sigma_N^2)) c + 2\mu \iota_N (\mu c' \iota_N - \mu) = 0$$

$$\operatorname{diag}(\sigma_1^2, \cdots \sigma_N^2) c + \mu^2 \iota_N c' \iota_N - \mu^2 \iota_N = 0$$

$$\left[\operatorname{diag}(\sigma_1^2, \cdots \sigma_N^2) + \mu^2 \iota_N \iota_N'\right] c = \mu^2 \iota_N$$

$$\boxed{c = \left[\operatorname{diag}(\sigma_1^2, \cdots \sigma_N^2) + \mu^2 \iota_N \iota_N'\right]^{-1} \mu^2 \iota_N}$$

(c)

$\sigma_i^2 = \sigma^2$.

Let us call, $D = \mathrm{diag}(\sigma^2, \dots \sigma^2)$

$$\iota_N' D^{-1} \iota_N = \sum_{i=1}^{N} \frac{1}{\sigma^2}$$

$$c = (D + \mathcal{H}^2 \iota_N \iota_N')^{-1} \mathcal{H}^2 \iota_N$$

$$(D + \mathcal{H}^2 \iota_N \iota_N')^{-1} = D^{-1} - \frac{\mathcal{H}^2 D^{-1} \iota_N \iota_N' D^{-1}}{1 + \mathcal{H}^2 \iota_N' D^{-1} \iota_N}$$

(d)

$$\mathrm{Bias}(\hat{\mu}) = \left[\frac{\mu^2}{\frac{\sigma^2}{N}+\mu^2} - 1\right]\mu$$

$$\mathrm{Var}(\hat{\mu}) = \left(\frac{\mu^2}{\frac{\sigma^2}{N}+\mu^2}\right)^2 \cdot \frac{\sigma^2}{N}$$

when $N \to \infty$, $\mathrm{Bias}(\hat{\mu}) \to 0$ as well as $\mathrm{Var}(\hat{\mu}) \to 0$, so it converges in probability to μ.

$$\frac{\mu^2 D^{-1} 1_N 1_N' D^{-1}}{1+\mu^2 \sum_{i=1}^{N} 1/\sigma^2} =$$

Since, $\hat{\mu} = \dfrac{\mu^2}{\sigma^2/N + \mu^2} \bar{Y}$

when $N \longrightarrow \infty$, $\bar{Y} \longrightarrow \mu$, $\sigma^2/N \longrightarrow 0$

so, $\hat{\mu} \xrightarrow{P} \dfrac{\mu^2}{\mu^2} \cdot \mu = \mu$

(e)

Since that estimator relied on true mean μ, here we can replace true mean μ also with $\bar{Y}$. Thus, We can write

$$\hat{\mu} = \frac{\bar{Y}^2}{\frac{\sigma^2}{N} + \bar{Y}^2} \bar{Y}$$

if we thn σ^2/N is very small,

then

$$\hat{\mu} = \frac{\bar{Y}^2 - \sigma^2/N}{\bar{Y}^2} \bar{Y}$$

will be similar (canot think of a good reason to do this).

The MSE of this will be lower as we showed in class MLE is inadmissible in this type of estimators and above shrinkage estimator has uniformly lower maximum MSE.

(f)

Mean is still pretty good estimator and specially in large sample all of these have similar performance.

Also for mean you don't need to know any extra information and it is much easier to calculate.

Problem 3 PS 2

Muhammad Bashir

In [1]:

```
# Load Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:

```
# set paths to working directory
path = '/Users/muhammadbashir/GitHub/MuhammadCourses/Ec240a/Problem Sets'
# load RPS_calorie_data.out data and read only columns Y0tc and X0te.
calories = pd.read_csv(path + '/RPS_calorie_data.out',usecols=['Y0tc','X0te'])
calories.head()
```

Out[3]:

	Y0tc	X0te
0	9.061790	9.917698
1	7.870896	8.276105
2	8.262976	8.036979
3	9.057076	9.212327
4	8.953918	9.826068

Let (Y) denote log calories and (X) denote log expenditure. Assume that [m(x) = \mathbb{E}[Y \mid X = x] = \sum_{k=1}^{K} \alpha_k g_k(x),]

where (g_k(x) = x^{k-1}).

Using the power series basis described above and the Gram-Schmidt algorithm construct a new basis that is orthogonal to the design points (set (K = 12)). Let (W_i) denote the (K \times 1) vector of orthonormal basis functions for the (i)-th household. Compute the least squares fit (m (X_i) = W_i^0 \hat{\theta}) with [\hat{\theta} = \left(\sum_{i=1}^{N} W_i W_i^0 \right)^{-1} \times \left(\sum_{i=1}^{N} W_i Y_i \right).] Plot this function onto a scatter of the unsmoothed data.

In [4]:

```
# First create K =12 g functions/polynomials, where g_k(x) = x^k-1 for the column X0te.
K = 12
```

```
for k in range(1,K+1):
    calories['X0te_g'+str(k)] = calories['X0te']**(k-1)
```

Orthogonalize g functions using Gram-Schmidt algoritm

In [5]:
```
def gram_schmidt_custom_inner_product(X):
    """
    Orthonormalizes the columns of matrix X using the Gram-Schmidt process
    with a custom inner product defined as the mean of the element-wise product.

    Parameters:
        X (numpy.ndarray): Input matrix with shape (m, n), where m >= n.

    Returns:
        numpy.ndarray: Matrix with orthonormalized columns.
    """
    def custom_inner_product(u, v):
        """Custom inner product: mean of element-wise product."""
        return np.mean(u * v)

    # Get the number of rows (m) and columns (n) of X
    m, n = X.shape

    # Initialize an empty matrix to store orthonormal vectors
    Q = np.zeros((m, n))

    for i in range(n):
        # Start with the current column vector
        v = X[:, i]

        # Subtract projections of v onto all previous Q columns
        for j in range(i):
            projection = custom_inner_product(Q[:, j], v) * Q[:, j]
            v -= projection

        # Normalize the vector using the custom norm
        v_norm = np.sqrt(custom_inner_product(v, v))
        if v_norm > 1e-10:  # Check to avoid division by zero
            Q[:, i] = v / v_norm
        else:
            raise ValueError(f"Column {i} is linearly dependent on previous columns.")
```

```python
    return Q


# Apply the Gram-Schmidt process to the columns of X0te
X = calories[['X0te_g' + str(k) for k in range(1, K+1)]].values
Q = gram_schmidt_custom_inner_product(X)
Q.shape
```

Out[5]: (1358, 12)

Compute least square fit of this orthonormal basis

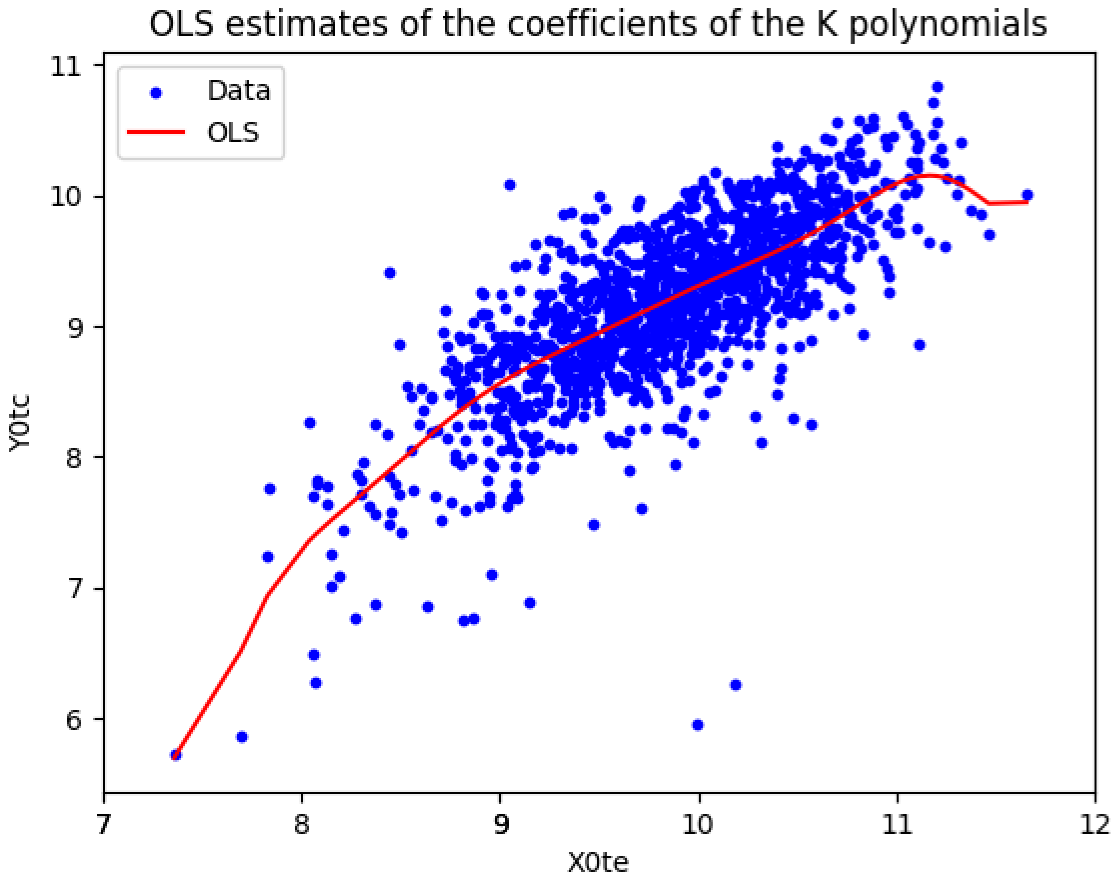
In [6]:
```python
# Compute the OLS estimates of the coefficients of the K polynomials i.,e regress Y0tc on the K polynomials fk.
# First create a list of the columns of the data frame that contain the K polynomials.
# X = calories[['X0te_f'+str(k) for k in range(1,13)]]
# X = np.array(X)
X= Q
Y = calories['Y0tc']
Y = np.array(Y)

# Compute the OLS estimates of the coefficients of the K polynomials i.,e regress Y0tc on the K polynomials fk.
beta = (np.linalg.inv(X.T @ X) @ X.T) @ Y
# predict mean by multiplying the estimated coefficients by the polynomials.
calories['Y0tc_hat'] = X @ beta
```

In [7]:
```python
# Plot this function onto a scatter of the unsmoothed data.
plt.scatter(calories['X0te'], calories['Y0tc'], label='Data', color='blue', s=10)
sorted_indices = np.argsort(calories['X0te'])
plt.plot(calories['X0te'].iloc[sorted_indices], calories['Y0tc_hat'].iloc[sorted_indices], label='OLS', color='red')
plt.xlabel('X0te')
plt.ylabel('Y0tc')
plt.legend()
plt.title('OLS estimates of the coefficients of the K polynomials')

# Add 10 equally spaced xticks
xticks = np.linspace(np.floor(calories['X0te'].min()), np.ceil(calories['X0te'].max()), 8)
xticks = xticks.astype(int)
plt.xticks(xticks)

plt.show()
```



3. Now use the shrinkage estimator of Efromovich (1999) as described in lecture to estimate m (Xi). Plot this function onto a scatter of the unsmoothed data. Comment on your findings.

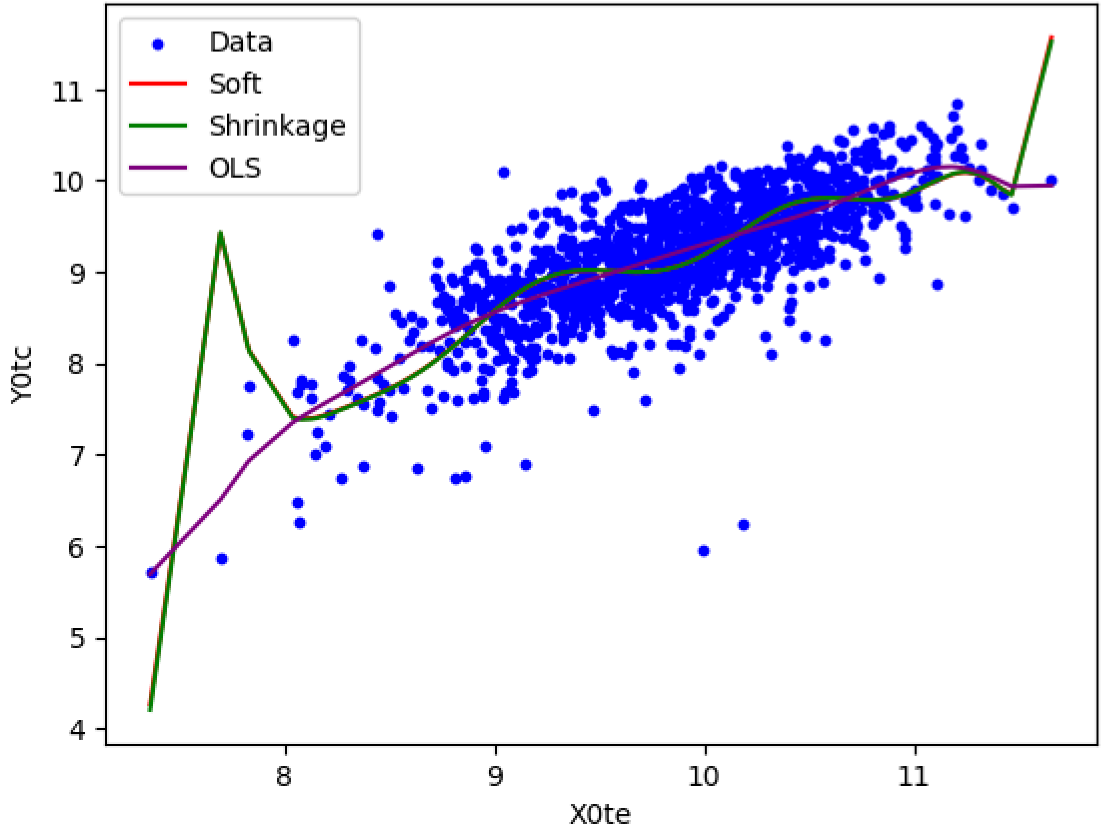
In [8]:
```
# Define Shrinkage constant c as 1-(Sigma^2)/(N*beta^2)
sigma2 = np.var(Y - X @ beta)
N = len(Y)
c = np.zeros(K)
theta = np.zeros(K)
for i in range(K):
    c[i] = 1 - (sigma2 / N) / beta[i]**2
    theta[i] = c[i] * beta[i]
calories['Y0tc_hat_shrink'] = np.dot(X, theta)
```

Soft Threshold Estimator

In [9]:
```python
# First let us choose lambda by minimizing (K/N)*Sigma^2-2(sigma^2/N)Sum1toK(1(abs(theta_k))<=lmabda)+Sum1toK(theta_k^2,lambda^2)
# Define the function that computes the value of the objective function for a given value of lambda.
def objective_function(lmbda, theta, sigma2,K,N):
    return (K/N)*sigma2 - 2*(sigma2/N)*np.sum(np.abs(theta) <= lmbda) + np.sum(np.minimum(theta**2, lmbda**2))
# minimize the objective function over lambda
from scipy.optimize import minimize_scalar
result = minimize_scalar(objective_function, args=(theta, sigma2,K,N))
lmbda = result.x
# Soft thresholding
theta_soft = np.sign(theta) * np.maximum(np.abs(theta) - lmbda, 0)
calories['Y0tc_hat_soft'] = X @ theta_soft
```

In [10]:
```python
# Plot the soft thresholding estimates of the coefficients of the K polynomials
plt.scatter(calories['X0te'], calories['Y0tc'], label='Data', color='blue', s=10)
sorted_indices = np.argsort(calories['X0te'])
plt.plot(calories['X0te'].iloc[sorted_indices], calories['Y0tc_hat_soft'].iloc[sorted_indices], label='Soft', color='red')
plt.plot(calories['X0te'].iloc[sorted_indices], calories['Y0tc_hat_shrink'].iloc[sorted_indices], label='Shrinkage', color='green')
plt.plot(calories['X0te'].iloc[sorted_indices], calories['Y0tc_hat'].iloc[sorted_indices], label='OLS', color='purple')
plt.xlabel('X0te')
plt.ylabel('Y0tc')
plt.legend()
plt.title('Soft thresholding and Shrinkage estimates of the coefficients of the K polynomials')
plt.show()
```

Soft thresholding and Shrinkage estimates of the coefficients of the K polynomials



We notice that OLS has overfitting to some extent as it is too smooth. Both Soft threshold and shrinkage estimators are same but different from OLS. These estimators punish overfitting and hence look different from OLS. Notice that in the middle, red/greem line is more flexible.