Homework 7, Dili Maduabum and Josh Bailey

Problem 1

Part 1

Exogenous

$\breve{X}$ is an exogenous instrument for $\tilde{X}$ if $Cov(\tilde{U}, \breve{X}) = 0$, where $\tilde{U}$ is gotten from:

$$Y = \tilde{X}B + U \implies Y = XB + \underbrace{V^1B + U}_{\tilde{U}}$$

$$\therefore Y = XB + \tilde{U}$$

$$\begin{aligned} Cov(\tilde{U}, \breve{X}) &= Cov(V^1B + U, \breve{X}) \\ &= Cov(V^1B + U, X + V^2) \\ &= Cov(V^1B, X) + Cov(V^1B, V^2) + Cov(U, X) + Cov(U, V^2) \end{aligned}$$

Term by term:

$$Cov(V^1B, X) = BCov(V^1, X) = B(\underbrace{E[XV^1]}_{0} - \underbrace{E[X]\underbrace{E[V^1]}_{0}}_{0}) = 0$$

$$Cov(V^1B, V^2) = BCov(V^1, V^2) = B(\underbrace{E[V^1V^2]}_{0} - \underbrace{E[V^1]\underbrace{E[V^2]}_{0}}_{0}) = 0$$

$Cov(U, X) = 0$ by assumption

$$Cov(U, V^2) = \underbrace{E[UV^2]}_{0} - \underbrace{E[U]\underbrace{E[V^2]}_{0}}_{0} = 0$$

$$\therefore Cov(\tilde{U}, \breve{X}) = 0 + 0 + 0 + 0 = 0///$$

Relevance

To show relevance, it's enough to show that $Cov(\tilde{X}, \breve{X}) \neq 0$

$$\begin{aligned} Cov(\tilde{X}, \breve{X}) &= Cov(X + V^1, X + V^2) \\ &= Cov(X, X) + Cov(X, V^2) + Cov(V^1, X) + Cov(V^1, V^2) \end{aligned}$$

Term by term:

$Cov(X, X) = Var(X)$

$Cov(X, V^2) = \underbrace{E[XV^2]}_{0} - \underbrace{E[X]\underbrace{E[V^2]}_{0}}_{0}) = 0$

$Cov(V^1, X) = 0$ (see the calculations in `Exogenous`)

$Cov(V^1, V^2) = 0$ (see the calculations in `Exogenous`)

$$\therefore Cov(\tilde{X}, \breve{X}) = Var(X) + 0 + 0 + 0 = Var(X) > 0///$$

Part 2

$$Q^s(P) = \alpha^s + P\beta^s + Z\gamma + U^s$$
$$Q^d(P) = \alpha^d + P\beta^d + U^d$$

a) We expect β^s to be positive (upward sloping slope curve) and β^d to be negative (downward sloping demand curve). Thus $\beta^d - \beta^s$ should be negative.

b) $Q^s(P) = Q^d(P)$

$$\alpha^s + P\beta^s + Z\gamma + U^s = \alpha^d + P\beta^d + U^d$$
$$P\beta^d - P\beta^s = \alpha^s - \alpha^d + Z\gamma + U^s - U^d$$
$$\implies P^{equilibrium} = \frac{\alpha^s - \alpha^d + Z\gamma + U^s - U^d}{\beta^d - \beta^s}$$

c) To show that the equilibrium price is endogenous in the model for demand, it is enough to show that : $Cov(P^{equilibrium}, U^d) \neq 0$

$$\implies Cov(\frac{\alpha^s - \alpha^d + Z\gamma + U^s - U^d}{\beta^d - \beta^s}, U^d)$$
$$= \frac{1}{\beta^d - \beta^s} Cov(\alpha^s - \alpha^d + Z\gamma + U^s - U^d, U^d)$$
$$= \frac{1}{\beta^d - \beta^s}(\underbrace{Cov(\alpha^s - \alpha^d, U^d)}_{\text{0, covariance with constant}} + \gamma \underbrace{Cov(Z, U^d)}_{\text{0, since } E[ZU^d]=0} + \underbrace{Cov(U^s, U^d)}_{0} - \underbrace{Cov(U^d, U^d)}_{Var(U^d)})$$
$$= -\frac{1}{\beta^d - \beta^s} Var(U^d) \neq 0 \quad \text{under the assumption that} \quad Var(U^d) \neq 0$$

Thus, equilibrium price is endogenous in the model for demand

d) Exogenous

To show that Z as instrument for P in the demand model is exogenous, it is enough to show that $Cov(Z, U^d) = 0$:

$$Cov(Z, U^d) = \underbrace{E[ZU^d]}_{\text{0, as given}} - \underbrace{E[Z]\underbrace{E[U^d]}_{0}}_{\implies 0} \implies 0$$

Relevance

To show that Z as instrument for P in the demand model is relevant, it is enough to show that $Cov(P, Z) \neq 0$:

$$Cov(P, Z) = Cov(\frac{\alpha^s - \alpha^d + Z\gamma + U^s - U^d}{\beta^d - \beta^s}, Z)$$
$$= \frac{1}{\beta^d - \beta^s} Cov(\alpha^s - \alpha^d + Z\gamma + U^s - U^d, Z)$$
$$= \frac{1}{\beta^d - \beta^s}(\underbrace{Cov(\alpha^s - \alpha^d, Z)}_{\text{0, covariance with constant}} + \gamma\underbrace{Cov(Z, Z)}_{Var(Z)} + \underbrace{Cov(U^s, Z)}_{0} - \underbrace{Cov(U^d, Z)}_{0})$$

The third term is 0 because $E[U^s, Z] = 0$ & $E[U^s] = 0$ while the fourth term is 0 because $E[U^d, Z] = 0$ & $E[U^d] = 0$

Thus

$$Cov(P, Z) = \frac{1}{\beta^d - \beta^s}\gamma Var(z) > 0$$

e) No, Z is not a valid instrument in the supply model since Z affects supply directly, not just through price.

Problem 2

Part 1

In [1]:
```
import numpy as np

def generate_data(n, beta1, beta2, beta3, beta4, gamma1, gamma2):
    """
    Generates simulated data

    Parameters:
    - n (int): Number of observations to generate.
    - beta1, beta2, beta3, beta4 (float): Coefficients for the model.
    - gamma1, gamma2 (float): Parameters affecting the relationship between x1

    Returns:
    - A tuple containing the generated y, x1, and x2 arrays.
    """

    # Generate n observations of x2 from a normal distribution (mean = 0, vari
    x2 = np.random.normal(0, 1, n)
```

```python
    # Generate n observations of v, the noise for x1, from a normal distributio
    v = np.random.normal(0, 0.1, n)

    # Compute x1 using the given relationship and the generated x2 and v
    x1 = x2 + x2**2 * gamma1 + x2**5 * gamma2 + v

    # Generate n observations of u, the structural error, from a normal distrib
    u = np.random.normal(0, 1, n)

    # Calculate y based on the given formula incorporating x1, x2, their intera
    y = x1 * beta1 + x2 * beta2 + x2**2 * beta3 + np.sin(x2) * beta4 + u

    # Return the generated y, x1, and x2
    return y, x1, x2
```

Part 2

In [2]:
```python
import statsmodels.api as sm

def estimate_unrestricted(y, x1, x2):
    """
    Estimates the unrestricted model and returns the coefficient and standard e
    """
    # Adding a constant term for intercept
    X = sm.add_constant(np.column_stack((x1, x2)))
    model = sm.OLS(y, X).fit()
    # Coefficient and standard error for beta1 (second parameter, hence index 1
    return model.params[1], model.bse[1]

def estimate_restricted(y, x1):
    """
    Estimates the restricted model and returns the coefficient and standard er
    """
    X = sm.add_constant(x1)  # Adding a constant term for intercept
    model = sm.OLS(y, X).fit()
    # Coefficient and standard error for beta1
    return model.params[1], model.bse[1]

def pretest_regression(y, x1, x2, significance_level=0.05):
    """
    Performs pretest regression as per Leeb & Pötscher (2005),
    taking into account the impact of model selection on inference.

    Parameters:
    y (array): Dependent variable.
    x1 (array): Independent variable of interest.
    x2 (array): Additional independent variable for the unrestricted model.
    significance_level (float): Significance level for model selection test.

    Returns:
    beta1_estimate (float): Estimated coefficient for beta1.
    beta1_se (float): Standard error of the beta1 estimate.
    model_selected (str): Indicates whether the 'restricted' or 'unrestricted'
    """
    # Adding constant term for intercept
    X_unrestricted = sm.add_constant(np.column_stack((x1, x2)))
    X_restricted = sm.add_constant(x1)
```

```
    # Fit both unrestricted and restricted models
    unrestricted_model = sm.OLS(y, X_unrestricted).fit()
    restricted_model = sm.OLS(y, X_restricted).fit()

    # Perform t-test on beta2 in the unrestricted model to decide on model sele
    t_test_beta2 = unrestricted_model.t_test("x2 = 0")

    # Model selection based on significance of beta2
    if t_test_beta2.pvalue < significance_level:
        model_selected = 'unrestricted'
        beta1_estimate = unrestricted_model.params[1]
        beta1_se = unrestricted_model.bse[1]
    else:
        model_selected = 'restricted'
        beta1_estimate = restricted_model.params[1]
        beta1_se = restricted_model.bse[1]

    return beta1_estimate, beta1_se, model_selected
```

Part 3

In [3]:

```
import pandas as pd

def generate_data(n, beta1, beta2, gamma1=0, gamma2=0, beta3=0, beta4=0):
    """
    Generates dataset based on specified parameters.
    """
    x2 = np.random.normal(0, 1, n)
    v = np.random.normal(0, 0.1, n)
    x1 = x2 + x2**2 * gamma1 + x2**5 * gamma2 + v
    u = np.random.normal(0, 1, n)
    y = x1 * beta1 + x2 * beta2 + x2**2 * beta3 + np.sin(x2) * beta4 + u
    return y, x1, x2

def estimate_coefficients(y, x1, x2):
    """
    Estimates coefficients for both unrestricted and restricted models.
    """
    X_unrestricted = sm.add_constant(np.column_stack((x1, x2)))
    X_restricted = sm.add_constant(x1)

    unrestricted_model = sm.OLS(y, X_unrestricted).fit()
    restricted_model = sm.OLS(y, X_restricted).fit()

    beta1_unrestricted = unrestricted_model.params[1]
    beta1_restricted = restricted_model.params[1]

    return beta1_unrestricted, beta1_restricted

n_values = [10, 100, 1000, 10000]
S = 1000
beta1 = 2
results = []

for n in n_values:
    beta2 = 3 / np.sqrt(n)
    estimates_n = {'n': [], 'beta1_unrestricted': [], 'beta1_restricted': []}

    for s in range(S):
```

```
        y, x1, x2 = generate_data(n, beta1, beta2)
        beta1_unrestricted, beta1_restricted = estimate_coefficients(y, x1, x2

        estimates_n['n'].append(n)
        estimates_n['beta1_unrestricted'].append(beta1_unrestricted)
        estimates_n['beta1_restricted'].append(beta1_restricted)

    results.append(pd.DataFrame(estimates_n))

# Example to display the results for n=5
print(results[0].head())
```

```
    n  beta1_unrestricted  beta1_restricted
0  10            0.833287          2.072125
1  10            3.088186          2.256175
2  10            2.474973          2.064985
3  10            7.836263          2.982712
4  10            1.945026          2.745197
```

Part 4

In [4]:

```
# Repeat functions from above
def generate_data(n, beta1, beta2, gamma1=0, gamma2=0, beta3=0, beta4=0):
    x2 = np.random.normal(0, 1, n)
    v = np.random.normal(0, 0.1, n)
    x1 = x2 + x2**2 * gamma1 + x2**5 * gamma2 + v
    u = np.random.normal(0, 1, n)
    y = x1 * beta1 + x2 * beta2 + x2**2 * beta3 + np.sin(x2) * beta4 + u
    return y, x1, x2

def estimate_coefficients(y, x1, x2):
    X_unrestricted = sm.add_constant(np.column_stack((x1, x2)))
    X_restricted = sm.add_constant(x1)
    unrestricted_model = sm.OLS(y, X_unrestricted).fit()
    restricted_model = sm.OLS(y, X_restricted).fit()
    return unrestricted_model.params[1], restricted_model.params[1], unrestric

n_values = [10, 100, 1000, 10000]
S = 1000
beta1 = 2
results = []

for n in n_values:
    beta2 = 3 / np.sqrt(n)
    summary_stats = {'n': [], 'beta1_unrestricted': [], 'beta1_restricted': []
                     'se_unrestricted': [], 'se_restricted': []}

    for s in range(S):
        y, x1, x2 = generate_data(n, beta1, beta2)
        beta1_unrestricted, beta1_restricted, se_unrestricted, se_restricted =

        summary_stats['n'].append(n)
        summary_stats['beta1_unrestricted'].append(beta1_unrestricted)
        summary_stats['beta1_restricted'].append(beta1_restricted)
        summary_stats['se_unrestricted'].append(se_unrestricted)
        summary_stats['se_restricted'].append(se_restricted)

    results.append(pd.DataFrame(summary_stats))

def analyze_results(results):
```

```
    for df in results:
        n = df['n'].iloc[0]

        # Compute mean and variance
        df['mean_unrestricted'] = df['beta1_unrestricted'].mean()
        df['var_unrestricted'] = df['beta1_unrestricted'].var()
        df['mean_restricted'] = df['beta1_restricted'].mean()
        df['var_restricted'] = df['beta1_restricted'].var()

        # Compute CI coverage and average length
        df['ci_coverage_unrestricted'] = df.apply(lambda x: x['beta1_unrestric
        df['ci_coverage_restricted'] = df.apply(lambda x: x['beta1_restricted']
        df['ci_length_unrestricted'] = 2 * 1.96 * df['se_unrestricted'].mean()
        df['ci_length_restricted'] = 2 * 1.96 * df['se_restricted'].mean()

        print(f"Results for n = {n}:")
        print(df[['mean_unrestricted', 'var_unrestricted', 'mean_restricted',
                  'ci_coverage_unrestricted', 'ci_coverage_restricted',
                  'ci_length_unrestricted', 'ci_length_restricted']].iloc[0])
        print("\n")

analyze_results(results)
```

```
Results for n = 10:
mean_unrestricted             1.810894
var_unrestricted             16.082085
mean_restricted               2.939840
var_restricted                0.145715
ci_coverage_unrestricted      0.915000
ci_coverage_restricted        0.258000
ci_length_unrestricted       14.861666
ci_length_restricted          1.395591
Name: 0, dtype: float64


Results for n = 100:
mean_unrestricted            2.037744
var_unrestricted             1.066319
mean_restricted              2.301753
var_restricted               0.009736
ci_coverage_unrestricted     0.936000
ci_coverage_restricted       0.145000
ci_length_unrestricted       3.987020
ci_length_restricted         0.394818
Name: 0, dtype: float64


Results for n = 1000:
mean_unrestricted            2.000427
var_unrestricted             0.097567
mean_restricted              2.093245
var_restricted               0.000956
ci_coverage_unrestricted     0.947000
ci_coverage_restricted       0.163000
ci_length_unrestricted       1.242989
ci_length_restricted         0.123523
Name: 0, dtype: float64


Results for n = 10000:
mean_unrestricted            2.000543
var_unrestricted             0.010120
mean_restricted              2.029993
var_restricted               0.000100
ci_coverage_unrestricted     0.949000
ci_coverage_restricted       0.141000
ci_length_unrestricted       0.392001
ci_length_restricted         0.039018
Name: 0, dtype: float64
```

The unrestricted model exhibits a mean estimate that progressively aligns closer to the true β_1 value with increasing sample size (n), signifying consistency and an unbiased nature. However, its variance, though decreasing with n, starts relatively high due to the inclusion of an additional variable (x_2), which, while contributing to model complexity, ensures the model's comprehensiveness.

Thw restricted Model shows significantly lower variance, suggesting a more stable but biased estimate. This model, by omitting x_2, fails to capture the entire data-generating process, leading to a systematic bias as reflected in the deviation of its mean estimates from the true β_1 and its poor performance in confidence interval coverage.

The phenomenon of "model selection effect" is observed, where the unrestricted model, which includes the variable x_2 (whose importance diminishes with n due to $\beta_2 = \frac{3}{\sqrt{n}}$), shows greater variance. As n grows, both models converge in terms of mean estimates towards the true value, but the restricted model does so with lower variance, misrepresenting the importance of x_2.

The fraction of instances where the true model (unrestricted) was selected is consistently high, indicating that the criteria for model selection strongly favor including x_2 at all sample sizes. This preference does not change with n. Ideally, the frequency of correct model selection should increase with n, as larger sample sizes provide more information for accurately evaluating the significance of variables.

Despite β_2 becoming less influential in larger samples ($\beta_2 = \frac{3}{\sqrt{n}}$), the model selection process continues to favor the inclusion of x_2, potentially leading to overfitting in very large samples.

For the unrestricted model, the coverage is close to the expected 95% across all sample sizes, indicating that the confidence intervals are correctly capturing the true value of β_1. The restricted model shows significantly lower coverage, far from the expected 95%, highlighting that omitting x_2 leads to biased estimates of β_1, and the confidence intervals fail to account for this bias.

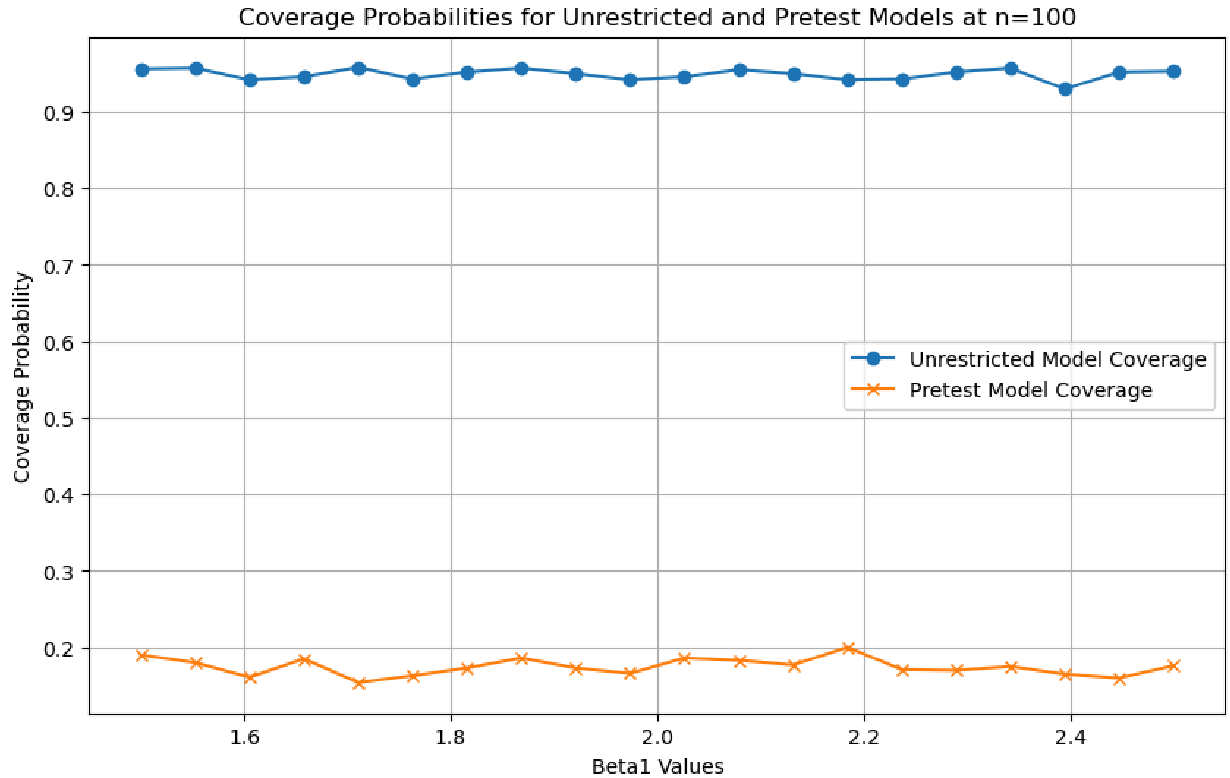
The expected coverage level for a 95% confidence interval is about 95% of the time to include the true parameter value. The unrestricted model performs as expected, demonstrating the reliability of its estimates and confidence intervals. The poor performance of the restricted model in terms of CI coverage highlights the consequences of model misspecification. Despite appearing more precise (shorter CIs), the restricted model's intervals fail to cover the true value due to bias introduced by omitting a relevant variable.

Part 5

Tried this but it doesn't seem right

In [5]:
```
import matplotlib.pyplot as plt

def simulate_data_and_calculate_coverage(n, beta1_values, S=1000):
    coverage_unrestricted = []
    coverage_pretest = []

    for beta1 in beta1_values:
        beta2 = 3 / np.sqrt(n)
        unrestricted_coverage_count = 0
        pretest_coverage_count = 0
```

```
        for _ in range(S):
            y, x1, x2 = generate_data(n, beta1, beta2)
            X_unrestricted = sm.add_constant(np.column_stack((x1, x2)))
            unrestricted_model = sm.OLS(y, X_unrestricted).fit()

            p_value_beta2 = unrestricted_model.t_test("x2 = 0").pvalue.item()

            ci_lower_u = unrestricted_model.params[1] - 1.96 * unrestricted_moc
            ci_upper_u = unrestricted_model.params[1] + 1.96 * unrestricted_moc
            if ci_lower_u <= beta1 <= ci_upper_u:
                unrestricted_coverage_count += 1

            selected_model = 'restricted' if p_value_beta2 > 0.05 else 'unrest

            if selected_model == 'unrestricted':
                if ci_lower_u <= beta1 <= ci_upper_u:
                    pretest_coverage_count += 1
            else:
                X_restricted = sm.add_constant(x1)
                restricted_model = sm.OLS(y, X_restricted).fit()
                ci_lower_r = restricted_model.params[1] - 1.96 * restricted_moc
                ci_upper_r = restricted_model.params[1] + 1.96 * restricted_moc
                if ci_lower_r <= beta1 <= ci_upper_r:
                    pretest_coverage_count += 1

        coverage_unrestricted.append(unrestricted_coverage_count / S)
        coverage_pretest.append(pretest_coverage_count / S)

    return coverage_unrestricted, coverage_pretest

n = 100
beta1_values = np.linspace(1.5, 2.5, 20)
S = 1000

# Calculate coverage probabilities
coverage_unrestricted, coverage_pretest = simulate_data_and_calculate_coverage

# Plot
plt.figure(figsize=(10, 6))
plt.plot(beta1_values, coverage_unrestricted, label='Unrestricted Model Coverag
plt.plot(beta1_values, coverage_pretest, label='Pretest Model Coverage', marke
plt.xlabel('Beta1 Values')
plt.ylabel('Coverage Probability')
plt.title('Coverage Probabilities for Unrestricted and Pretest Models at n=100
plt.legend()
plt.grid(True)
plt.show()
```



Problem 3

Part 1

$$\|y - X\beta\|_2^2 = \sum_{i=1}^{n}(y_i - X_i\beta)^2$$

$$\text{add and subtract } X^*\beta$$

$$= \sum_{i=1}^{n}((y_i - X_i\beta^*) + (X_i\beta^* - X_i\beta))^2$$

$$= \sum_{i=1}^{n}((y_i - X_i\beta^*)^2 + (X_i(\beta^* - \beta))^2 + 2(y_i - X_i\beta^*)(X_i(\beta^* - \beta)))$$

$$= \sum_{i=1}^{n}(y_i - X_i\beta^*)^2 + \sum_{i=1}^{n}(X_i(\beta^* - \beta))^2 + 2\sum_{i=1}^{n}(y_i - X_i\beta^*)(X_i(\beta^* - \beta))$$

$$\text{As given, } Y - X^*\beta = U$$

$$\therefore \|y - X\beta^*\|_2^2 + \|X(\beta^* - \beta)\|_2^2 + 2U^TX(\beta^* - \beta)///$$

Part 2

$f(\hat{\beta}) = \|y - X\hat{\beta}\|_2^2 + \lambda\|\hat{\beta}\|_1$

$f(\beta^*) = \|y - X\beta^*\|_2^2 + \lambda\|\beta^*\|_1$

$$f(\hat{\beta}) \leq f(\beta^*)$$

$$\text{Applying the expression in part 1 on the first term of } f(\hat{\beta}) \text{ and } f(\beta^*)$$

$$\implies \cancel{\|y - X\beta^*\|_2^2} + \|X(\beta^* - \hat{\beta})\|_2^2 + 2U^T X(\beta^* - \hat{\beta}) + \lambda\|\hat{\beta}\|_1 \leq$$

$$\cancel{\|y - X\beta^*\|_2^2} + \cancelto{0}{\|X(\beta^* - \beta^*)\|_2^2} + \cancelto{0}{2U^T X(\beta^* - \beta^*)} + \lambda\|\beta^*\|_1$$

$$\implies \|X(\beta^* - \hat{\beta})\|_2^2 \leq -2U^T X(\beta^* - \hat{\beta}) + \lambda(\|\beta^*\|_1 - |\hat{\beta}\|_1)$$

$$= \|X(\beta^* - \hat{\beta})\|_2^2 \leq 2U^T X(\hat{\beta} - \beta^*) + \lambda(\|\beta^*\|_1 - |\hat{\beta}\|_1)///$$

Part 3`

It was used to conclude the proof that $f(\hat{\beta}) \leq f(\beta^*)$, where β^* is the true beta