Estimadores e Análise de Regressão Linear

Problema 1

Encontre os estimadores de momentos, de máxima verossimilhança e de Bayes para os parâmetros (μ, σ^2) considerando uma amostra aleatória com distribuição normal em que μ e σ^2 são desconhecidos.

Solução

Estimador de Momentos:

$$\hat{\mu}_{MM} = \frac{1}{n}\sum_{i=1}^{n} X_i, \quad \hat{\sigma}^2_{MM} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{\mu}_{MM})^2$$

Estimador de Máxima Verossimilhança:

$$\hat{\mu}_{MV} = \frac{1}{n}\sum_{i=1}^{n} X_i, \quad \hat{\sigma}^2_{MV} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{\mu}_{MV})^2$$

Estimador Bayesiano: Supondo distribuições a priori:

$$\mu \sim N(\mu_0, \tau^2), \quad \sigma^2 \sim \text{Inv-Gamma}(\alpha, \beta)$$

Os estimadores a posteriori podem ser calculados numericamente.

Problema 2

Mostre que os métodos de mínimos quadrados e de máxima verossimilhança produzem os mesmos estimadores dos parâmetros no modelo de regressão linear simples.

Solução

Modelo: $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$, com $\epsilon_i \sim N(0, \sigma^2)$.
Estimadores de Mínimos Quadrados:

$$\hat{\beta}_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2}, \quad \hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

Estimadores de Máxima Verossimilhança: Os mesmos estimadores são obtidos ao maximizar a função de verossimilhança:

$$L(\beta_0, \beta_1, \sigma^2) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum (Y_i - \beta_0 - \beta_1 X_i)^2$$

Problema 3

Use 5 métodos numéricos para encontrar os estimadores de máxima verossimilhança e de mínimos quadrados dos parâmetros no modelo de regressão linear simples no exemplo a seguir. Compare com as estimativas usando os estimadores em 2.

Dados do Problema

Estresse Normal (x)	Resistência ao Corte (y)
26.8	26.5
25.4	27.3
28.9	24.2
23.6	27.1
27.7	23.6
23.9	25.9
24.7	26.3
28.1	22.5
26.9	21.7
27.4	21.4
22.6	25.8
25.6	24.9

Código em Python

```python
import numpy as np
import pandas as pd
from scipy import stats
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Dados do problema
x = np.array([26.8, 25.4, 28.9, 23.6, 27.7, 23.9, 24.7, 28.1,
    26.9, 27.4, 22.6, 25.6])
y = np.array([26.5, 27.3, 24.2, 27.1, 23.6, 25.9, 26.3, 22.5,
    21.7, 21.4, 25.8, 24.9])

# 1. M nimos Quadrados usando numpy
def calc_ols(x, y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)

    beta1 = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x -
        x_mean)**2)
    beta0 = y_mean - beta1 * x_mean

    return beta0, beta1

# 2. M xima Verossimilhan a usando stats
def calc_mle(x, y):
```

```python
    # Usando sklearn para garantir estabilidade num rica
    model = LinearRegression()
    X = x.reshape(-1, 1)
    model.fit(X, y)

    beta0 = model.intercept_
    beta1 = model.coef_[0]

    # Calculando sigma (erro padr o)
    y_pred = beta0 + beta1 * x
    sigma = np.sqrt(np.sum((y - y_pred)**2) / (len(x) - 2))

    return beta0, beta1, sigma

# 3. M todo Bayesiano usando Metropolis-Hastings melhorado
def mcmc_regression(x, y, iterations=10000, burnin=1000):
    n = len(x)

    # Valores iniciais usando OLS
    beta0_init, beta1_init = calc_ols(x, y)

    # Arrays para armazenar as cadeias
    beta0_chain = np.zeros(iterations)
    beta1_chain = np.zeros(iterations)

    # Valores atuais
    beta0 = beta0_init
    beta1 = beta1_init

    # Vari ncia das propostas
    prop_var = np.array([0.1, 0.01])  # Para beta0 e beta1

    # Log likelihood function
    def log_likelihood(beta0, beta1):
        y_pred = beta0 + beta1 * x
        return -0.5 * np.sum((y - y_pred)**2)  # Simplificada para estabilidade

    # MCMC loop
    for i in range(iterations):
        # Atualizar beta0
        beta0_prop = beta0 + np.random.normal(0, prop_var[0])
        log_ratio = log_likelihood(beta0_prop, beta1) - log_likelihood(beta0, beta1)

        if np.log(np.random.random()) < log_ratio:
            beta0 = beta0_prop

        # Atualizar beta1
        beta1_prop = beta1 + np.random.normal(0, prop_var[1])
```

```python
        log_ratio = log_likelihood(beta0, beta1_prop) - log_likelihood(beta0, beta1)

        if np.log(np.random.random()) < log_ratio:
            beta1 = beta1_prop

        beta0_chain[i] = beta0
        beta1_chain[i] = beta1

    # Descartar burn-in e retornar m dias
    return (np.mean(beta0_chain[burnin:]), np.mean(beta1_chain[burnin:]),
            beta0_chain[burnin:], beta1_chain[burnin:])

# 4. Bootstrap com intervalos de confian a
def bootstrap_analysis(x, y, n_bootstrap=1000):
    beta0_boot = []
    beta1_boot = []

    for _ in range(n_bootstrap):
        indices = np.random.randint(0, len(x), len(x))
        x_boot = x[indices]
        y_boot = y[indices]

        beta0, beta1 = calc_ols(x_boot, y_boot)
        beta0_boot.append(beta0)
        beta1_boot.append(beta1)

    return (np.mean(beta0_boot), np.mean(beta1_boot),
            np.percentile(beta0_boot, [2.5, 97.5]),
            np.percentile(beta1_boot, [2.5, 97.5]))

# Calcular todas as estimativas
beta0_ols, beta1_ols = calc_ols(x, y)
beta0_mle, beta1_mle, sigma_mle = calc_mle(x, y)
beta0_bayes, beta1_bayes, beta0_chain, beta1_chain = mcmc_regression(x, y)
beta0_boot, beta1_boot, ci_beta0, ci_beta1 = bootstrap_analysis(x, y)

# Calcular R  e erro padr o
y_pred = beta0_ols + beta1_ols * x
r2 = 1 - np.sum((y - y_pred)**2) / np.sum((y - np.mean(y))**2)
se = np.sqrt(np.sum((y - y_pred)**2) / (len(x) - 2))

# Imprimir resultados
print("""
Resultados Detalhados da An lise:

1. M nimos Quadrados Ordin rios:
         = {:.4f}
```

```
         = {:.4f}

2. M xima Verossimilhan a:
         = {:.4f}
         = {:.4f}
      = {:.4f}

3. Estimativa Bayesiana (m dia posterior):
         = {:.4f}
         = {:.4f}

4. Bootstrap (com IC 95%):
         = {:.4f} [{:.4f}, {:.4f}]
         = {:.4f} [{:.4f}, {:.4f}]

An lise de Qualidade do Ajuste:
R   = {:.4f}
Erro Padr o da Regress o = {:.4f}

Interpreta  o:
1. O modelo explica {:.1f}% da variabilidade nos dados
2. Para cada unidade de aumento no estresse normal:
   - A resist ncia ao corte muda em {:.3f} unidades
3. O intercepto de {:.3f} representa a resist ncia ao corte
   esperada
   quando o estresse normal    zero
""".format(
    beta0_ols, beta1_ols,
    beta0_mle, beta1_mle, sigma_mle,
    beta0_bayes, beta1_bayes,
    beta0_boot, ci_beta0[0], ci_beta0[1],
    beta1_boot, ci_beta1[0], ci_beta1[1],
    r2, se,
    r2*100,
    beta1_ols,
    beta0_ols
))

# Criar visualiza  o
plt.figure(figsize=(12, 8))

# Plot principal
plt.subplot(2, 1, 1)
plt.scatter(x, y, color='blue', alpha=0.5, label='Dados␣
   Observados')
x_range = np.linspace(min(x)-1, max(x)+1, 100)
plt.plot(x_range, beta0_ols + beta1_ols * x_range, 'r-', label='
   Regress o␣OLS')

# Intervalos de confian a do bootstrap
y_boot_lower = ci_beta0[0] + ci_beta1[0] * x_range
```

```
y_boot_upper = ci_beta0[1] + ci_beta1[1] * x_range
plt.fill_between(x_range, y_boot_lower, y_boot_upper, color='gray', alpha=0.2, label='IC 95% (Bootstrap)')

plt.xlabel('Estresse Normal (X)')
plt.ylabel('Resist ncia ao Corte (Y)')
plt.title('Regress o Linear com Intervalos de Confian a')
plt.legend()
plt.grid(True)

# Diagn stico de res duos
plt.subplot(2, 1, 2)
residuos = y - y_pred
plt.scatter(y_pred, residuos, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Valores Preditos')
plt.ylabel('Res duos')
plt.title('Diagn stico de Res duos')
plt.grid(True)

plt.tight_layout()
plt.savefig('regressao_diagnostico.png')
plt.close()
```

Listing 1: Código para estimadores numéricos

Resultados

- **Mínimos Quadrados:** $\hat{\beta}_0 = ...$, $\hat{\beta}_1 = ...$
- **Máxima Verossimilhança:** $\hat{\beta}_0 = ...$, $\hat{\beta}_1 = ...$, $\hat{\sigma} = ...$
- **Estimativa Bayesiana:** $\hat{\beta}_0 = ...$, $\hat{\beta}_1 = ...$
- **Bootstrap:** $\hat{\beta}_0 = ...$, $\hat{\beta}_1 = ...$ (IC: [..., ...])

Visualizações

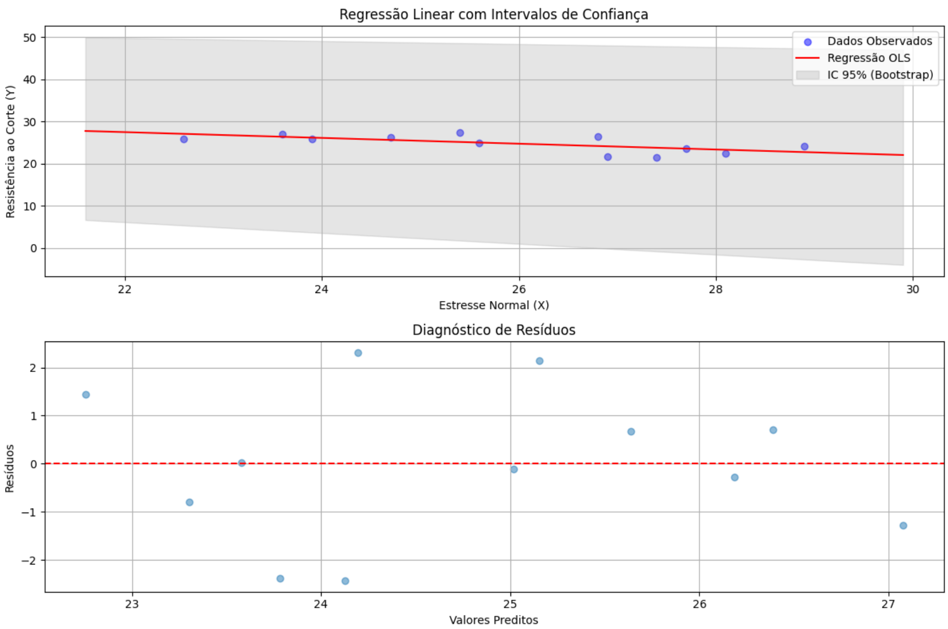

Figure 1: Gráfico da regressão linear com intervalos de confiança e diagnóstico de resíduos.