Data Science Assignment

Group 1

Morris, Pierri, Rosa, Valentini

Accademic Year 2024/2025

In [1]:
```python
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from scipy.stats import norm, uniform

from scipy.optimize import minimize

np.random.seed(42)
```

Exercise 1: Finite sample properties of Ridge estimator

Given a linear model $Y_i = X_i\beta_0 + \varepsilon_i$ *with* $X_i \in \mathbb{R}^k$ *and the ridge estimator* $\hat{\beta}_n^R = \min_{\beta\in\mathbb{R}^k}\left\{\frac{1}{2n}\|Y - \beta X\|_2^2 + \frac{\lambda_n}{2}\|\beta\|_2^2\right\}$ *where* n *is the number of observations and* $\lambda_n \in \mathbb{R}_+$ *the penalty term*

1.1 Monte Carlo simulation of OLS and Ridge estimators

Simulate observations for $\{X_i, Y_i\}$ *following the previous linear model with* $X_i \sim N(0, I_k)$, $\varepsilon_i \sim N(0, \sigma^2 I_k)$, $\sigma^2 = 0.5$, $n = 50$, $k = 5$ *and* $\beta_0 = [0.1, 0.05, 0.2, 0.9, 0.5]$

- *For various simulations compute the OLS and Ridge estimators* $\hat{\beta}_n^{OLS}$, $\hat{\beta}_n^R$ *given that* $\lambda_n = 0.1 \cdot n^{\frac{1}{3}}$

In [2]:
```python
# Settings

n=50
K=5
sigma2 = 0.5

X = np.random.randn(n, K)
eps = np.random.normal(0, np.sqrt(0.5), n)
beta0 = np.array([0.1, 0.05, 0.2, 0.9, 0.5])

Y = np.dot(X, beta0) + eps #X*beta + eps

alpha = 0.1
l = alpha * (n**(1/3))
```

In [3]:
```python
def simulate_data(real_beta, n_data = 50, n_feature = 5, mean_error = 0, var_error = 0.5):
  """
  Function to simulate data X, Y, epsilon
  """
  X = np.random.randn(n_data, n_feature)
  eps = np.random.normal(0, np.sqrt(var_error), n_data)
  Y = np.dot(X, real_beta) + eps

  return Y, X, eps
```

In [4]:
```python
def desMatrix_ (X):
  """
  Function to compute the inverse of matrix (X'X)
  """
  return np.linalg.inv(np.dot(X.T, X))

def leastSquare(X, Y):
  """
  Function to compute the OLS (X^t * X)^(-1) * X^t * y
  """
  temp = desMatrix_(X)
  temp = np.dot(temp, X.T)
  return np.dot(temp, Y)

def ridgeLeastSquare(X, Y, n, K, lambdan):
  """
  Function computing the Ridge estimator
  """
  Q_ = desMatrix_(X)*n      # Q_ is the inverse of Qn = (X'X)/n
  I = np.eye(K)
  Wn = np.linalg.inv(I + lambdan * Q_)
  beta_ols = leastSquare(X,Y)

  return np.dot(Wn, beta_ols)
```

In [5]:
```python
# Simulations
max_dim = 10000
beta_ols = np.zeros((max_dim, K))
beta_ridge = np.zeros((max_dim, K))
data_YXe = []
tol = 1e-9

np.random.seed(42)
for i in range(max_dim):

  Y, X, eps = simulate_data(beta0)
  data_YXe.append([Y, X, eps])

  # Regression
  beta_ols[i] = leastSquare(X, Y)
  beta_ridge[i] = ridgeLeastSquare(X, Y, n, K, l)
```

•• *Show that the two estimators satisfy the following relation:*

$\hat{\beta}_n^R = W_n(\lambda_n) \cdot \hat{\beta}_n^{OLS}$ *where* $W_n(\lambda_n) = (I_k + \lambda_n \cdot Q_n^{-1})^{-1}$, $Q_n = \frac{X^T \cdot X}{n}$ *and* I_k *is the identity matrix of dimensions* $k \times k$

Proof:

The first-order condition for the solution of our minimization problem is given by:

$$\frac{1}{2n}\left(-2X^T\right)\left(Y - X\hat{\beta}_n^R\right) + \frac{\lambda_n}{2}2\hat{\beta}_n^R = 0$$

which leads to:

$$(Q_n + \lambda_n I_K)\hat{\beta}_n^R - \frac{X^TY}{n} = 0$$

where $Q_n = \frac{X^TX}{n}$ and I_k is the identity matrix of dimensions $k \times k$.

Therefore, since matrix $Q_n + \lambda_n I_K$ is positive definite, we can invert it and obtain:

$$\hat{\beta}_n^R = (Q_n + \lambda_n I_K)^{-1}\frac{X^TY}{n}.$$

Since $Q_n\hat{\beta}_n^{OLS} = \frac{X^TX}{n}(X^TX)^{-1}X^TY = \frac{X^TY}{n}$,

$$\hat{\beta}_n^R = (Q_n + \lambda_n I_K)^{-1}Q_n\hat{\beta}_n^{OLS}.$$

And finally, with

$$W_n(\lambda_n) := (Q_n + \lambda_n I_K)^{-1}Q_n = \left(I_K + \lambda_n Q_n^{-1}\right)^{-1}$$

we obtain

$$\hat{\beta}_n^R = W_n(\lambda_n)\hat{\beta}_n^{OLS}.$$

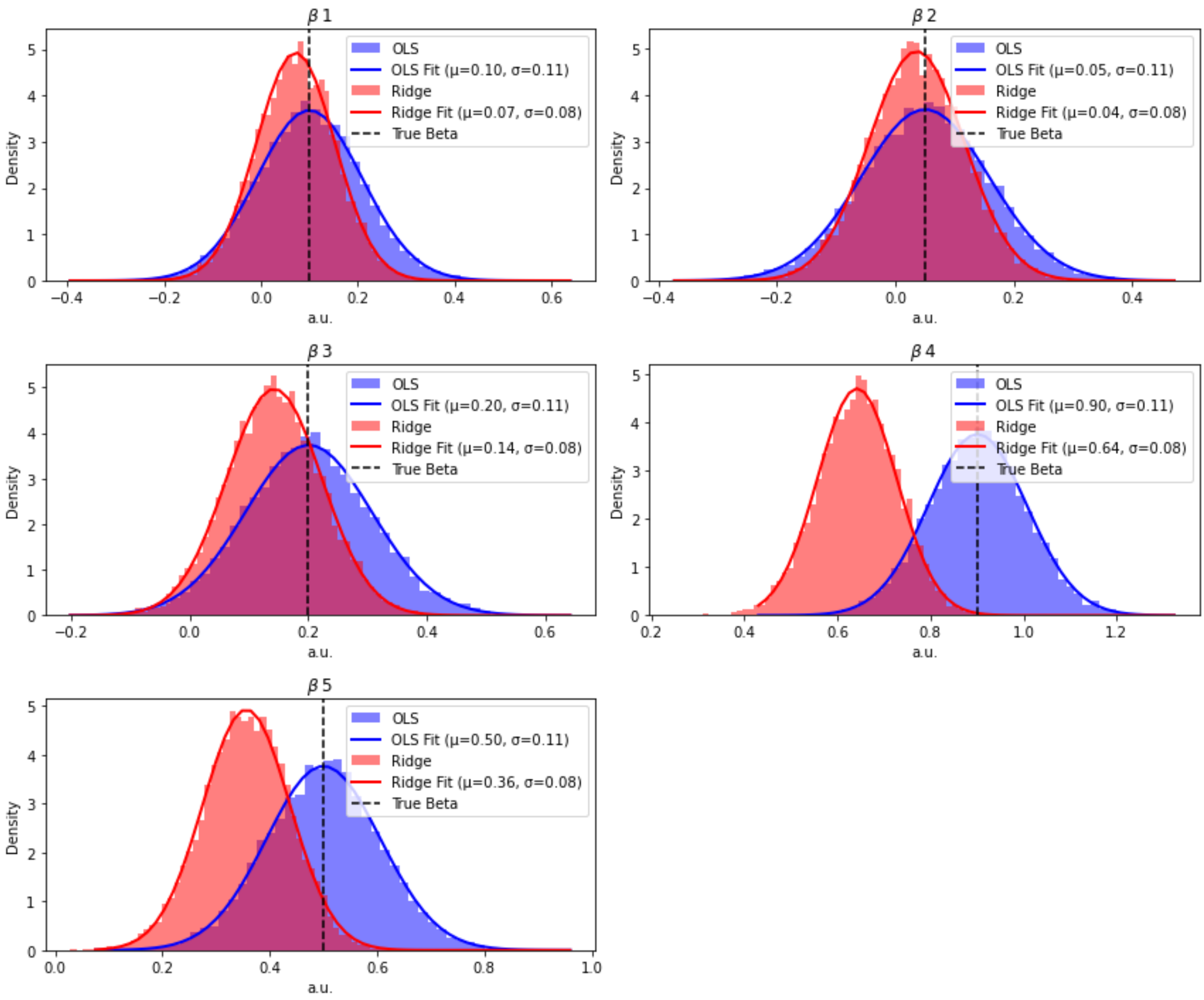
In [6]:
```
np.random.seed(42)
for i in range(len(data_YXe)):

  Y = data_YXe[i][0]
  X = data_YXe[i][1]



  Q_ = desMatrix_(X)*n       # Q_ is the inverse of Qn = (X'X)/n
  I = np.eye(K)
  Wn = np.linalg.inv(I + l * Q_)

  wrong_beta = []


  for j in range(len(beta_ridge[i])):

    temp = Wn@beta_ols[i]

    if np.abs(beta_ridge[i,j] - temp[j]) > tol:
        wrong_beta.append([i,j])


print('Number of "different" betas: ',len(wrong_beta))
  #if i%100 == 0:
      #print(beta_ridge[i],'\n', Wn@beta_ols[i])
```

```
Number of "different" betas:  0
```

••• *Plot the histogram of the OLS and the Ridge estimators in your simulation. What can you observe about the bias and variance of the estimators?*

In [7]:

```
# Plot coefficient histogram for every dimension of beta

n_dimensions = beta_ols.shape[1]

fig, axes = plt.subplots(3, 2, figsize=(12, 10))
axes = axes.ravel()

n_bins = 56

for i in range(n_dimensions):
  ax = axes[i]

  # OLS
  data_ols = beta_ols[:, i]
  ax.hist(beta_ols[:, i], bins=50, color='blue', alpha=0.5, density= True, label='OLS')
  mean_ols, std_ols = norm.fit(data_ols)
  x_vals = np.linspace(data_ols.min(), data_ols.max(), n_bins)
  pdf_ols = norm.pdf(x_vals, mean_ols, std_ols)
  ax.plot(x_vals, pdf_ols, 'blue', lw=2, label=f'OLS Fit (μ={mean_ols:.2f}, σ={std_ols:.2f})')

  # Ridge
  data_ridge = beta_ridge[:, i]
  ax.hist(data_ridge, bins = n_bins, color='red', alpha=0.5,density = True, label='Ridge')
  mean_ridge, std_ridge = norm.fit(data_ridge)
  pdf_ridge = norm.pdf(x_vals, mean_ridge, std_ridge)
  ax.plot(x_vals, pdf_ridge, 'red', lw=2, label=f'Ridge Fit (μ={mean_ridge:.2f}, σ={std_ridge:.2f})')

  ax.axvline(x=beta0[i], color='black', linestyle='--', label='True Beta')

  ax.set_title(r'$\beta \;$'+f'{i+1}')
  ax.set_xlabel('a.u.')
  ax.set_ylabel('Density')
  ax.legend()



axes[-1].axis('off')
fig.tight_layout()
plt.show()
```



Before starting it is important to remember the *Bias-Variance Decomposition*

$$MSE = \mathbb{E}\left[(\hat{\beta} - \beta_0)^2\right] = \left(\mathbb{E}\left[\hat{\beta}\right] - \beta_0\right)^2 + \mathbb{E}\left[\left(\hat{\beta} - \mathbb{E}\left[\hat{\beta}\right]\right)^2\right] + \sigma_\varepsilon^2 = Bias(\hat{\beta}) + Var(\hat{\beta}) + \sigma_\varepsilon^2$$

where the bias is the distance of the expected value of the estimator from the true value, while the variance tells us how "spread" is the distribution (σ_ε is the variance of the error of the model).

Now, that said:

1. The OLS estimators fits best the true parameters, as we expected from the generation process.

2. The *Bias - Variance Decomposition* is satisfied, in fact, nearer is the mean of the estimator to the true parameter, higher it is the variance since, we know that by increasing the bias we have a reduction of the variance, and viceversa. In our case, the first scenario is exactly what happens.

3. The Ridge estimator tends to shrink the parameters. Therefore β_0 which are near to zero are better estimated. This is due to the fact the bigger are the parameters, the bigger is the shrinkage.

•••• *Show that the Ridge estimator has lower variance than the OLS estimator, i.e.,*

$\mathbb{V}ar(\hat{\beta_n^{OLS}}|X) \succ \mathbb{V}ar(\hat{\beta_n^{R}}|X)$

Proof:

Using the expression for the Ridge estimator found before and using the covariance matrix expression of the OLSE:

$$\mathrm{Var}\left(\hat\beta^{R}_n|X\right) = W_n(\lambda_n)\mathrm{Var}\left(\hat\beta^{OLS}_n|X\right)W_n(\lambda_n) = \sigma^2_0 W_n(\lambda_n)(X'X)^{-1}W_n(\lambda_n)$$

We know that $W_n(\lambda_n) = \left(I_K + \lambda_n Q_n^{-1}\right)^{-1}$ and that $Q_n = \frac{X\cdot X^T}{n}$, so we can compute:

$$\begin{aligned}\frac{\mathrm{Var}\left(\hat\beta^{OLS}_n|X\right) - \mathrm{Var}\left(\hat\beta^{R}_n|X\right)}{\sigma^2_0/n} &= \frac{\sigma^2_0(X'X)^{-1} - \sigma^2_0 W_n(\lambda_n)(X'X)^{-1}W_n(\lambda_n)}{\sigma^2_0/n}\\ &= \frac{(X'X)^{-1}}{1/n} - W_n(\lambda_n)\frac{(X'X)^{-1}}{1/n}W_n(\lambda_n)\\ &= Q_n^{-1} - W_n(\lambda_n)Q_n^{-1}W_n(\lambda_n)\\ &= W_n(\lambda_n)\left(W_n(\lambda_n)^{-1}Q_n^{-1}W_n(\lambda_n)^{-1} - Q_n^{-1}\right)W_n(\lambda_n).\end{aligned}$$

Where the latter can be obtained by multiplying the first Q_n^{-1} by $W_n(\lambda_n)\cdot W_n(\lambda_n)^{-1}$ on the left and by $W_n(\lambda_n)^{-1}\cdot W_n(\lambda_n)$ on the right, thanks to we can regroup.

Analyzing the matrix in the middle:

$$\begin{aligned}W_n(\lambda_n)^{-1}Q_n^{-1}W_n(\lambda_n)^{-1} - Q_n^{-1} &= \left(I_K + \lambda_n Q_n^{-1}\right)Q_n^{-1}\left(I_K + \lambda_n Q_n^{-1}\right) - Q_n^{-1}\\ &= Q_n^{-1} + 2\lambda_n Q_n^{-2} + \lambda_n^2 Q_n^{-3} - Q_n^{-1}\\ &= 2\lambda_n Q_n^{-2} + \lambda_n^2 Q_n^{-3},\end{aligned}$$

it is positive definite whenever $\lambda_n > 0$, since $2\lambda_n Q_n^{-2} + \lambda_n^2 Q_n^{-3}$ is positive definite (under the assumption of X full rank).

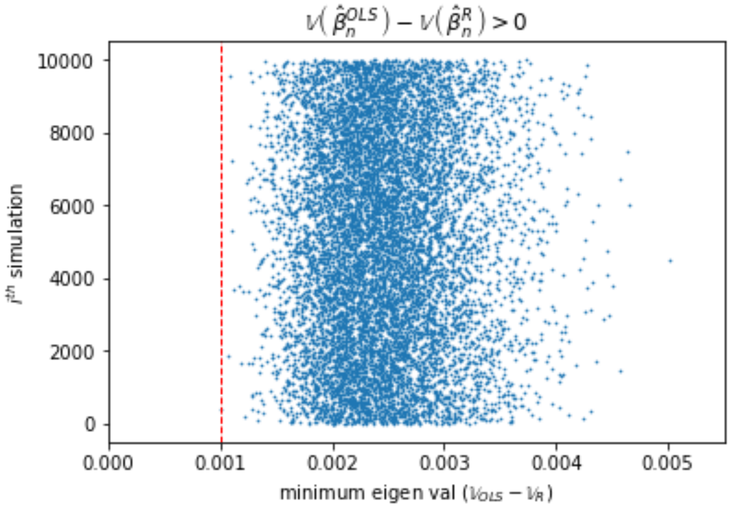
Therefore, since $W_n(\lambda_n)$ is symmetric and positive definite (under the assumption of X full rank)

$$\frac{n}{\sigma^2_0}\left(\mathrm{Var}\left(\hat\beta^{OLS}_n|X\right) - \mathrm{Var}\left(\hat\beta^{R}_n|X\right)\right) \succ 0$$

whenever $\lambda_n > 0$.

In [8]:
```
tol = 1e-7
dif_cov_min = np.zeros(len(data_YXe))

for i in range(len(data_YXe)):

  X = data_YXe[i][1]

  var_betaOls = sigma2 * desMatrix_(X)
  W = np.linalg.inv( np.eye(K) + l * desMatrix_(X)*n )

  var_betaR =  W@var_betaOls@W

  dif = var_betaOls - var_betaR
  eig_cov_dif = np.linalg.eigvals(dif)
  dif_cov_min[i] =  min(eig_cov_dif)

  if dif_cov_min[i] < tol:
      raise ValueError(f"The i-th min eigenvalue is too near 0: {i}")
```

In [9]:
```
plt.scatter( dif_cov_min, range(len(dif_cov_min)) ,s = 0.5)
padding = 0.01
plt.axvline(min(dif_cov_min), color='red', linestyle='--',linewidth=1)


plt.xlim( 0, 0.0055)
plt.ylabel('$i^{th}$ simulation')
plt.xlabel('minimum eigen val $(\mathbb{V}_{OLS} - \mathbb{V}_{R})$')
plt.title(r'$ \mathbb{V}\, \left(\,\hat\beta^{OLS}_n \,\right) - \mathbb{V}\, \left(\, \hat\beta^{R}_{n
plt.show()
```



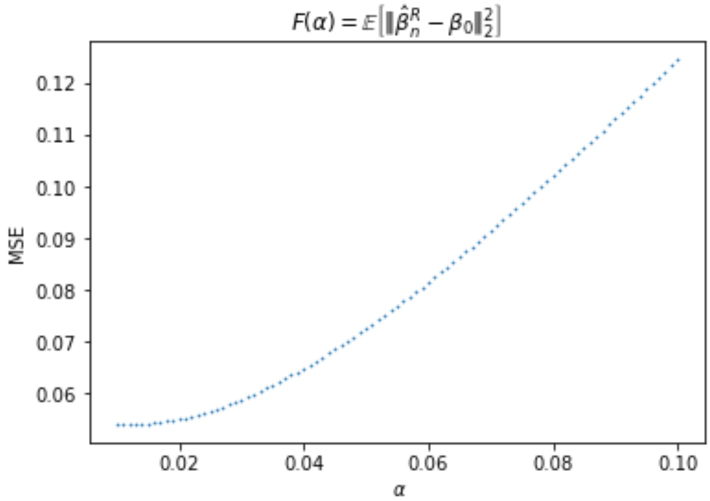
1.2 Penalty parameter choice by minimizing the mean square error

- *Define* $F(\alpha) = \mathbb{E}\left[\left\Vert \hat\beta_n^{R} - \beta_0 \right\Vert_2^2\right]$, *where* $\hat\beta_n^{R}$ *is the Ridge estimator with penalty parameter* $\lambda_n = \alpha \cdot n^{\frac{1}{3}}$. *Using similar simulation specifications as above produce a plot of the function* F *for values of* α *between 0.01 and 0.1.*

In [10]:
```
def square_norm(beta0, alpha, Y, X, n = 50, K = 5):
    """
    Function to compute the square norm of the difference
    between the Ridge estimator and the true parameter
    """
    l = alpha * (n**(1/3))



    beta_ridge = ridgeLeastSquare(X, Y, n, K, l)
    diff_b = (beta_ridge - beta0)

    return np.linalg.norm(diff_b, 2)**2


def mse_alpha(beta0, alpha, data, n = 50, K = 5):
    """
    Function computing F_alpha
    """
    temp = []
    for i in range(len(data)):
        Y = data[i][0]
        X = data[i][1]
        temp.append(square_norm(beta0, alpha, Y, X))
    return np.mean(temp)
```

In [11]:
```
# Computing F with alpha in range [0.01, 0.1)
F = [mse_alpha(beta0,alpha, data_YXe) for alpha in np.arange(0.01, 0.1, 0.001)]

# Compute the minimum of F to use it later
min_lambda_F = F[np.argmin(F)]
```

In [12]:
```
plt.scatter(np.arange(0.01, 0.1, 0.001), F, s = 0.5)
plt.ylabel('MSE')
plt.xlabel(r' $\alpha$')
plt.title(r'$ F(\alpha) = \mathbb{E} \left[ \Vert \hat\beta_n^{R} - \beta_0 \Vert^{2}_{2} \right] $')
plt.show()
```

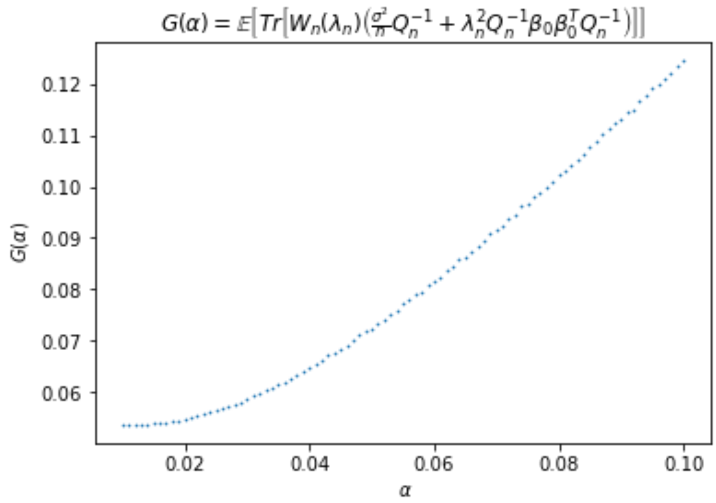


We can observe that the smaller mean square error is obtained with a smaller alpha, as we expected from the previous point.

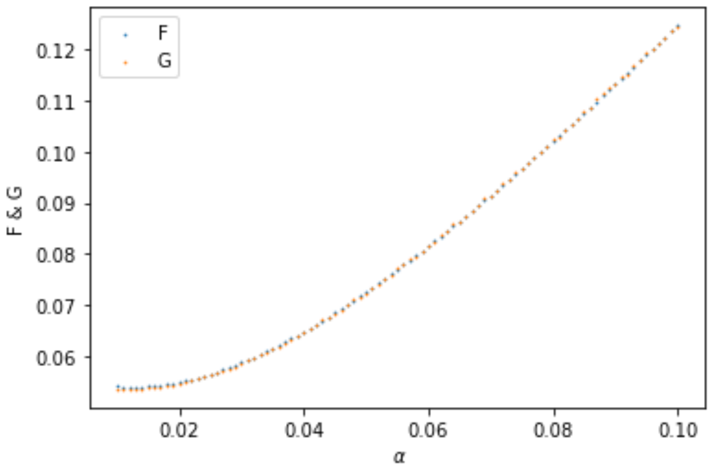
•• *Let us define now the function:* $G(\alpha) = \mathbb{E}\left[Tr\left[W_n(\lambda_n)\left(\frac{\sigma^2}{n}Q_n^{-1} + \lambda_n^2 Q_n^{-1}\beta_0\beta_0^T Q_n^{-1}\right)W_n(\lambda_n)\right]\right]$. *Compute by simulation function* G, *plot it and show that it is equal to the mean square function* F.

In [13]:

```
np.random.seed(42)
def compute_G(alpha, beta0, sigma2, data, n = 50, K =  5):
  """
  Function to compute G
  """
  G = []

  for i in range(len(data)):

    _, X, _ = simulate_data(beta0)#data[i][1]

    l = alpha * (n**(1/3))
    Q_ = desMatrix_(X)*n # Q_ is the inverse of the matrix (X'X)/n

    W = np.linalg.inv( np.eye(K) + l * Q_ )


    part_1 = (sigma2/n) * Q_

    term_1 = (l**2) * Q_
    term_2 = np.outer(beta0,beta0)
    term_3 = term_1 @ term_2
    part_2 = term_3 @ Q_

    midTerm = part_1 + part_2

    mat = W@midTerm@W

    tr = np.trace(mat)
    G.append(tr)

  return np.mean(G)


# Computing G with alpha in range [0.01, 0.1)
G_alpha = [compute_G(alpha, beta0, sigma2, data_YXe) for alpha in np.arange(0.01, 0.1, 0.001)]

# Compute the minimum of G to use it later
min_lambda_G = G_alpha[np.argmin(G_alpha)]

plt.scatter(np.arange(0.01, 0.1, 0.001), G_alpha, s=0.5)
plt.ylabel(r' $G(\alpha)$ ')
plt.xlabel(r' $\alpha$')
plt.title(r'$G(\alpha) = \mathbb{E}\left[Tr\left[W_n(\lambda_n)\left(\frac{\sigma^2}{n}Q_n^{-1}+\lambda
plt.show()
```



In [14]:
```
plt.scatter(np.arange(0.01, 0.1, 0.001), F, s = 0.5, label = 'F')
plt.scatter(np.arange(0.01, 0.1, 0.001), G_alpha, s=0.5, label = 'G')
plt.legend()
plt.ylabel('F & G')
plt.xlabel(r' $\alpha$')
plt.show()
```



Proof

We want to show that $G(\alpha) = F(\alpha)$. To begin we remember the definition of F and observe that it can be rewritten using the trace:

$$F(\alpha) = \mathbb{E}\left[\left\|\hat{\beta}_n^R - \beta_0\right\|_2^2\right] = Tr\left(\mathbb{E}\left[\left(\hat{\beta}_n^R - \beta_0\right)\cdot\left(\hat{\beta}_n^R - \beta_0\right)^T\right]\right)$$

.

The latter expression holds since

$$\mathbb{E}\left[\left\|\hat{\beta}_n^R - \beta_0\right\|_2^2\right] = \mathbb{E}\left[\sum_i\left(\hat{\beta}_{i,n}^R - \beta_{i,0}\right)^2\right] = \sum_i\mathbb{E}\left[\left(\hat{\beta}_{i,n}^R - \beta_{i,0}\right)\right] = Tr\left(\mathbb{E}\left[\left(\hat{\beta}_n^R - \beta_0\right)\cdot\left(\hat{\beta}_n^R - \beta_0\right)^T\right]\right)$$

Now, by definition of Bias and Variance of an estimator:

$$\frac{\mathbb{E}\left[\left(\hat{\beta}_n^R - \beta_0\right)\cdot\left(\hat{\beta}_n^R - \beta_0\right)^T\right]}{\sigma_0^2/n} = \frac{\mathbb{V}ar\left(\hat{\beta}_n^R\right) + \mathbb{E}\left[\left(\hat{\beta}_n^R - \mathbb{E}\left[\hat{\beta}_n^R\right]\right)\cdot\left(\hat{\beta}_n^R - \mathbb{E}\left[\hat{\beta}_n^R\right]\right)^T\right]}{\sigma_0^2/n}$$

where

$$\frac{\mathbb{V}ar(\hat{\beta}_n^R)}{\sigma_0^2} = W_n(\lambda_n)Q_n^{-1}W_n(\lambda_n)$$
$$Q_n^{-1} = \frac{\mathbb{E}\left[\left(\hat{\beta}_n^{OLS} - \beta_0\right)\cdot\left(\hat{\beta}_n^{OLS} - \beta_0\right)\right]}{\sigma_0^2/n}$$

and

$$\begin{aligned}\mathbb{E}\left[\left(\hat{\beta}_n^R - \mathbb{E}\left[\hat{\beta}_n^R\right]\right)\cdot\left(\hat{\beta}_n^R - \mathbb{E}\left[\hat{\beta}_n^R\right]\right)^T\right] &= (W_n(\lambda_n) - I_k)\,\frac{\beta_0\cdot\beta_0^T}{\sigma_0^2/n}(W_n(\lambda_n) - I_k)\\ &= \lambda_n^2 W_n(\lambda_n)Q_n^{-1}\frac{\beta_0\cdot\beta_0^T}{\sigma_0^2/n}Q_n^{-1}W_n(\lambda_n).\end{aligned}$$

And since $I_k - W_n(\lambda_n)^{-1} = -\lambda_n Q_n^{-1}$, we get that:

$$\frac{\mathbb{E}\left[\left(\hat{\beta}_n^R - \beta_0\right)\cdot\left(\hat{\beta}_n^R - \beta_0\right)^T\right]}{\sigma_0^2/n} = W_n(\lambda_n)\left(Q_n^{-1} + \lambda_n^2 Q_n^{-1}\frac{\beta_0\cdot\beta_0^T}{\sigma_0^2/n}Q_n^{-1}\right)W_n(\lambda_n)$$

Therefore, we have

$$\begin{aligned}F(\alpha) = \mathbb{E}\left[\left(\hat{\beta}_n^R - \beta_0\right)\cdot\left(\hat{\beta}_n^R - \beta_0\right)^T\right] &= W_n(\lambda_n)\left(\frac{\sigma_0^2}{n}Q_n^{-1} + \lambda_n^2 Q_n^{-1}(\beta_0\cdot\beta_0^T)Q_n^{-1}\right)W_n(\lambda_n)\\ &= \mathbb{E}\left[Tr\left(W_n(\lambda_n)\left(\frac{\sigma_0^2}{n}Q_n^{-1} + \lambda_n^2 Q_n^{-1}(\beta_0\cdot\beta_0^T)Q_n^{-1}\right)W_n(\lambda_n)\right)\right] = G(\alpha)\end{aligned}$$

••• *Substitute the parameters β_0 and σ^2 in the function G with the OLS estimator $\hat{\beta}_n^{OLS}$ and the error variance estimator $\hat{\sigma}_n^2$, in order to optimize the square error in each simulation by choosing the optimal penalty parameter. For this purpose, in each simulation compute:*

$$\hat{\lambda}_n^{opt} = \min_{\lambda \geq 0} Tr\left[W_n(\lambda)\left(\frac{\hat{\sigma}_n^2}{2}Q_n^{-1} + \lambda^2 Q_n^{-1}\hat{\beta}_n^{OLS} \cdot \hat{\beta}_n^{OLS^T} Q_n^{-1}\right)W_n(\lambda)\right]$$

where

$$\hat{\sigma}_n^2 = \frac{\left\| Y - \hat{\beta}_n^{OLS} X \right\|_2^2}{n-k}.$$

Using such optimal penalty parameter compute the Ridge estimator and show that it delivers a mean square error similar to the minimum of functions F and G.

In [15]:
```
#np.random.seed(42)

def fun_lambda(l, X, Y, n= 50, K = 5):
  """
  Object function which we'll use to minimize the optimal lambda
  """

  Q_ = desMatrix_(X)*n
  W = np.linalg.inv( np.eye(K) + l * Q_ )

  beta_hat = leastSquare(X,Y)
  eps_hat = Y - (X@beta_hat)
  sigma_hat = (np.linalg.norm(eps_hat)**2) / (n-K)

  # As G(alpha) calculation
  part_1 = (sigma_hat / n) * Q_

  term_1 = l**2 * Q_
  term_2 = np.outer(beta_hat, beta_hat)
  term_3 = term_1 @ term_2

  part_2 = term_3 @ Q_

  midTerm = part_1 + part_2

  mat = W@midTerm@W

  # Trace of the matrix
  tr = np.trace(mat)

  return tr
```

In [16]:
```
def mse_opt_lambda( beta0, data, n = 50, K =  5):
  """
  Function computing the mean square error with the optimized lambda
  """
  MSE = []
  for i in range(len(data)):

    Y = data[i][0]
    X = data[i][1]

    l = minimize(fun_lambda, x0=0.0, args=(X, Y), bounds = [(0.0, None)])
    beta_ridge_hat = ridgeLeastSquare(X, Y, n, K, lambdan = l.x[0])

    mse = np.linalg.norm(beta_ridge_hat - beta0)**2
    MSE.append(mse)
  return np.mean(MSE)

optimal_lambda_mse = mse_opt_lambda(beta0, data_YXe)
```

In [17]:
```
print('Optimal lambda of F by varying alpha: ', round(min_lambda_F,3),'\n')
print('Optimal lambda of G by varying alpha: ', round(min_lambda_G,3),'\n')
print('Optimal lambda by minimization ', round(optimal_lambda_mse,3))
```

```
Optimal lambda of F by varying alpha:  0.054

Optimal lambda of G by varying alpha:  0.054

Optimal lambda by minimization  0.057
```

In conclusion, we find an MSE similar to the minima of F and G. The discrepancy may is product of the step size convergence of the minimize function or due to the number of simulations.

Exercise 2: Nonlinear regression with measurement errors

A researcher considers the (nonlinear) regression model:

$$y_i = h(x_i^*, \beta_0) + \varepsilon_i,$$

where β_0 is a scalar parameter and h is a given function.

The explanatory variable is observed with a measurement error:

$$x_i = x_i^* + u_i.$$

The available data for the researcher are $(x_i, y_i)_{i=1,...,n}$ and we assume that

$$(x_i^*, \varepsilon_i, u_i)' \sim^{i.i.d.} N\left[\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_{x^*}^2 & 0 & 0 \\ 0 & \sigma_\varepsilon^2 & 0 \\ 0 & 0 & \sigma_u^2 \end{pmatrix}\right].$$

2.1 Misspecified estimation

Suppose first that the researcher neglects the measurement errors. We want to study the consequences of this choice.

To simplify, let us assume in this part of the exercise that $h(x_i^*, \beta_0) = x_i^* \cdot \beta_0$ ($x_i = x_i^*$), that is, the regression model is linear. The researcher proposes the estimator:

$$\hat{\beta} = \arg\min_\beta \sum_{i=1}^n (y_i - x_i\beta)^2.$$

- *Compute*

$$\beta_0^* = \arg\min_\beta \mathbb{E}_0\left[(y_i - x_i\beta)^2\right]$$

where $\mathbb{E}_0[\cdot]$ denotes the expected value with respect the real underlying distribution. Is the estimator $\hat{\beta}$ consistent for β_0?

β_0^* computation

$$\begin{aligned}\mathbb{E}_0[(y_i - x_i\beta)^2] &= \mathbb{E}_0[y_i^2 - 2y_ix_i\beta + x_i^2\beta^2] = \mathbb{E}_0[(x_i\beta_0 + \varepsilon_i)^2 - 2(x_i\beta_0 + \varepsilon_i)x_i\beta + x_i^2\beta^2] \\ &= \mathbb{E}_0[x_i^2\beta_0^2 + \varepsilon^2 + 2\varepsilon x_i\beta_0 - 2x_i^2\beta\beta_0 - 2\varepsilon_ix_i\beta + x_i^2\beta^2] \\ &= \mathbb{E}_0[x_i^2(\beta_0^2 + \beta^2) + 2\varepsilon_ix_i(\beta_0 - \beta) - 2x_i^2\beta\beta_0 + \varepsilon^2]\end{aligned}$$

Now, given that $\mathbb{E}[x_i] = \mathbb{E}[\varepsilon_i] = 0$ and that they are independent (which means that $\mathbb{E}[x_i\varepsilon_i] = \mathbb{E}[x_i]\mathbb{E}[\varepsilon_i]$).

We have that

$$\mathbb{E}_0[(y_i - x_i\beta)^2] = \sigma_x^2(\beta_0^2 + \beta^2) - 2\sigma_x^2\beta\beta_0 + \sigma_\varepsilon^2$$

where $\sigma_x^2 = Var(x_i)$, $\sigma_\varepsilon^2 = Var(\varepsilon_i)$ $\forall i$ (since the mean is 0 for both x_i and ε_i).

Now we want to differentiate with respect to beta and find the minimum. Therefore,

$$\frac{\partial}{\partial\beta}\mathbb{E}_0[(y_i - x_i\beta)^2] = 2\sigma_x^2\beta - 2\sigma_x^2\beta_0 = 2\sigma_x^2(\beta - \beta_0) = 0 \iff \beta = \beta_0$$

As so, we get that $\beta_0^* = \beta_0$

The proof of **consistency** is the following:

$$
\begin{aligned}
\sum_{i=1}^{n}(y_i - x_i\beta)^2 &= \sum_{i=1}^{n}(x_i\beta_0 + \varepsilon_i - x_i\beta)^2 \\
&= \sum_{i=1}^{n} \beta_0^2 x_i^2 + \varepsilon_i^2 + \beta^2 x_i^2 + 2\beta_0 x_i \varepsilon_i - 2\beta\beta_0 x_i^2 - 2\beta x_i \varepsilon_i \\
&= \beta_0^2 \sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} \varepsilon_i^2 + \beta^2 \sum_{i=1}^{n} x_i^2 + 2\beta_0 \sum_{i=1}^{n} x_i\varepsilon_i - 2\beta\beta_0 \sum_{i=1}^{n} x_i^2 - 2\beta \sum_{i=1}^{n} x_i \varepsilon_i
\end{aligned}
$$

Multiplying and dividing by n, we get

$$
n\beta_0^2 \frac{\sum_{i=1}^{n} x_i^2}{n} + n\frac{\sum_{i=1}^{n} \varepsilon_i^2}{n} + n\beta^2 \frac{\sum_{i=1}^{n} x_i^2}{n} + 2n\beta_0 \frac{\sum_{i=1}^{n} x_i\varepsilon_i}{n} - 2n\beta\beta_0 \frac{\sum_{i=1}^{n} x_i^2}{n} - 2n\beta \frac{\sum_{i=1}^{n} x_i\varepsilon_i}{n}
$$

So, by setting the partial derivative of this quantity with respect to β to zero and simplifying, we obtain:

$$
2\hat{\beta}\frac{\sum_{i=1}^{n} x_i^2}{n} - 2\beta_0 \frac{\sum_{i=1}^{n} x_i^2}{n} - 2\frac{\sum_{i=1}^{n} x_i\varepsilon_i}{n} = 0
$$

We can now send $n \to +\infty$ and get the expectations:

$$
\begin{aligned}
\frac{1}{n}\sum_i x_i^2 &\to \mathbb{E}_0[x_i^2] = \sigma_x^2 \\
\frac{1}{n}\sum_i x_i\varepsilon_i &\to \mathbb{E}_0[x_i\varepsilon_i] = 0
\end{aligned}
$$

Thus:

$$
2\hat{\beta}\sigma_x^2 - 2\beta_0\sigma_x^2 = 0 \iff \hat{\beta} = \beta_0
$$

Then the estimator $\hat{\beta}$ is consistent for β_0.

•• *Derive the asymptotic distribution of the M-estimator $\hat{\beta}$.*

For the **asymptotic distribution** of the estimator $\hat{\beta}$: if we call $\psi(\beta) = \frac{\varepsilon_i^2}{2}$, we have

$$\hat{\beta} = \arg\min_{\beta} \mathbb{E}_0\left[\psi(\beta)\right].$$

The partial derivatives are:

$$\frac{\partial\varepsilon}{\partial\beta} = -x \qquad \frac{\partial^2\varepsilon}{\partial\beta\partial\beta'} = 0$$
$$\frac{\partial\psi(\beta)}{\partial\beta} = \varepsilon\frac{\partial\varepsilon}{\partial\beta} = -x\varepsilon \qquad \frac{\partial^2\psi(\beta)}{\partial\beta\partial\beta'} = x^2$$

So the matrices:

$$J_0 = \mathbb{E}_0\left[-\frac{\partial^2\psi(\beta)}{\partial\beta\partial\beta'}\right] = \mathbb{E}_0[-x^2] = -\sigma_x^2$$
$$I_0 = \mathbb{E}_0\left[\frac{\partial\psi(\beta)}{\partial\beta}\frac{\partial\psi(\beta)}{\partial\beta'}\right] = \mathbb{E}_0[x^2\varepsilon^2] = \sigma_x^2\sigma_\varepsilon^2 = -\sigma_\varepsilon^2 J_0$$

Finally the asymptotic distribution of the estimator $\hat{\beta}$ is:

$$\sqrt{n}(\hat{\beta} - \beta_0) \xrightarrow{d} \mathcal{N}(0, \Sigma)$$

where $\Sigma = J_0^{-1} I_0 J_0^{-1} = \frac{\sigma_\varepsilon^2}{\sigma_x^2}$

2.2 NLS estimation

Suppose now that the researcher properly accounts for measurement errors. Let us assume that

$$h(x_i^*, \beta_0) = (x_i^* + \beta_0)^2.$$

• *Show that*

$$\mathbb{E}_0[y_i|x_i] = (x_i^* + \beta_0)^2 + \sigma_u^2$$

We can compute

$$\begin{aligned}\mathbb{E}_0\left[y_i \mid x_i\right] &= \mathbb{E}_0\left[(x_i^* + u_i + \beta_0)^2 + \varepsilon_i \mid x_i\right] \\ &= \mathbb{E}_0\left[x_i^{*2} + u_i^2 + \beta_0^2 + 2x_i^* u_i + 2x_i^*\beta_0 + 2u_i\beta_0 + \varepsilon_i \mid x_i\right] \\ &= (x_i + \beta_0)^2 + \sigma_u^2\end{aligned}$$

Thus, the equation is proven.

•• *Propose a consistent NLS estimator of β_0, called $\hat{\beta}_{NLS}$. Derive the asymptotic distribution of $\hat{\beta}_{NLS}$.*

A **consistent estimator** of β_0 is:

$$\hat{\beta}_{NLS} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - (x_i^* + \beta)^2\right)^2.$$

The consistency can be shown as follows:

$$\begin{aligned}
\sum_{i=1}^{n} \left(y_i - (x_i^* + \beta)^2\right)^2 &= \sum_{i=1}^{n} \left((x_i^* + \beta_0)^2 + \varepsilon_i - (x_i^* + \beta)^2\right)^2 \\
&= \sum_{i=1}^{n} \left(x_i^{*2} + 2x_i^*\beta_0 + \beta_0^2 + \varepsilon_i - x_i^{*2} - 2x_i^*\beta - \beta^2\right)^2 \\
&= \sum_{i=1}^{n} \left((\beta_0^2 - \beta^2) + 2x_i^*(\beta_0 - \beta) + \varepsilon_i\right)^2 \\
&= \sum_{i=1}^{n} (\beta_0^2 - \beta^2)^2 + 4x_i^{*2}(\beta_0 - \beta)^2 + \varepsilon_i^2 + 4x_i^*(\beta_0^2 - \beta^2)(\beta_0 - \beta) + \\
&\quad + 2\varepsilon_i(\beta_0^2 - \beta^2) + 4x_i^*\varepsilon_i(\beta_0 - \beta) \\
&= n(\beta_0^2 - \beta^2)^2 + 4(\beta_0 - \beta)^2 \sum_{i=1}^{n} x_i^{*2} + \sum_{i=1}^{n} \varepsilon_i^2 + 4(\beta_0^2 - \beta^2)(\beta_0 - \beta) \sum_{i=1}^{n} x_i^* + \\
&\quad + 2(\beta_0^2 - \beta^2) \sum_{i=1}^{n} \varepsilon_i + 4(\beta_0 - \beta) \sum_{i=1}^{n} x_i^*\varepsilon_i
\end{aligned}$$

Let's call that function $\alpha(\beta)$. It can be written as

$$\begin{aligned}
\alpha(\beta) = n(\beta_0^2 - \beta^2)^2 + 4n(\beta_0 - \beta)^2 \frac{\sum_{i=1}^{n} x_i^{*2}}{n} + n\frac{\sum_{i=1}^{n} \varepsilon_i^2}{n} + 4n(\beta_0^2 - \beta^2)(\beta_0 - \beta)\frac{\sum_{i=1}^{n} x_i^*}{n} + \\
+ 2n(\beta_0^2 - \beta^2)\frac{\sum_{i=1}^{n} \varepsilon_i}{n} + 4n(\beta_0 - \beta)\frac{\sum_{i=1}^{n} x_i^*\varepsilon_i}{n}
\end{aligned}$$

Setting the partial derivative with respect to β to zero:

$$\hat{\beta}(\beta_0^2 - \hat{\beta}^2) + 2(\beta_0 - \hat{\beta})\frac{\sum_{i=1}^{n} x_i^{*2}}{n} + 2\hat{\beta}(\beta_0 - \hat{\beta})\frac{\sum_{i=1}^{n} x_i^*}{n} + (\beta_0^2 - \hat{\beta}^2)\frac{\sum_{i=1}^{n} x_i^*}{n} + \hat{\beta}\frac{\sum_{i=1}^{n} \varepsilon_i}{n} + \frac{\sum_{i=1}^{n} x_i^*\varepsilon_i}{n} = 0$$

As $n \to +\infty$, we take expectations:

$$\begin{aligned}
\frac{1}{n}\sum_i x_i^{*2} &\to \mathbb{E}_0[x_i^{*2}] = \sigma_{x^*}^2 \\
\frac{1}{n}\sum_i \varepsilon_i^2 &\to \mathbb{E}_0[\varepsilon_i^2] = \sigma_{\varepsilon}^2 \\
\frac{1}{n}\sum_i x_i^*\varepsilon_i &\to \mathbb{E}_0[x_i^*\varepsilon_i] = 0 \\
\frac{1}{n}\sum_i x_i^* &\to \mathbb{E}_0[x_i^*] = 0 \\
\frac{1}{n}\sum_i \varepsilon_i &\to \mathbb{E}_0[\varepsilon_i] = 0
\end{aligned}$$

Thus:

$$\hat{\beta}(\beta_0^2 - \hat{\beta}^2) + 2\sigma_{x^*}^2(\beta_0 - \hat{\beta}) = 0$$

Further reduction:

$$(\beta_0 - \hat{\beta})\left[\hat{\beta}(\beta_0 + \hat{\beta}) + 2\sigma_{x^*}^2\right] = 0$$

Hence, one solution is:

$$\hat{\beta}_{NLS} = \beta_0.$$

It can be shown that this is the minimum, while the other two solutions are maxima.

Then, for the **asymptotic distribution** of $\hat{\beta}_{NLS}$, we call $\psi(\beta) = \frac{\varepsilon_i^2}{2} = \frac{\left(y_i - (x_i^* + \beta)^2\right)^2}{2}$:

$$\hat{\beta}_{NLS} = \arg\min_{\beta} \mathbb{E}_0 \left[\psi(\beta)\right].$$

The partial derivatives are:

$$\frac{\partial \varepsilon}{\partial \beta} = -2(x^* + \beta) \qquad \frac{\partial^2 \varepsilon}{\partial \beta \partial \beta'} = -2$$

$$\frac{\partial \psi(\beta)}{\partial \beta} = \varepsilon \frac{\partial \varepsilon}{\partial \beta} = -2\varepsilon(x^* + \beta) \qquad \frac{\partial^2 \psi(\beta)}{\partial \beta \partial \beta'} = 4(x^* + \beta)^2 - 2\varepsilon$$

So the matrices are:

$$J_0 = \mathbb{E}\left[-\frac{\partial^2 \psi(\beta)}{\partial \beta \partial \beta'}\bigg|_{\beta=\beta_0} \right] = -2\,\mathbb{E}[2(x^* + \beta_0)^2 - \varepsilon] = -4\,\mathbb{E}[x^{*2} + \beta_0^2 + 2\beta_0 x^*] = -4(\sigma_{x^*}^2 + \beta_0^2)$$

$$I_0 = \mathbb{E}\left[\frac{\partial \psi(\beta)}{\partial \beta} \frac{\partial \psi(\beta)}{\partial \beta'}\bigg|_{\beta=\beta_0} \right] = 4\,\mathbb{E}[\varepsilon^2(x^* + \beta_0)^2] = 4\,\mathbb{E}[\varepsilon^2 x^{*2} + 2\varepsilon^2 x^* \beta_0 + \varepsilon^2 \beta_0^2] = 4(\sigma_\varepsilon^2 \sigma_{x^*}^2 + \sigma_\varepsilon^2 \beta_0^2) = -\sigma_\varepsilon^2 J_0$$

Finally the asymptotic distribution of the estimator $\hat{\beta}_{NLS}$ is:

$$\sqrt{n}(\hat{\beta}_{NLS} - \beta_0) \xrightarrow{d} \mathcal{N}(0, \Sigma)$$

where $\Sigma = J_0^{-1} I_0 J_0^{-1} = \frac{\sigma_\varepsilon^2}{4(\sigma_{x^*}^2 + \beta_0^2)}$.

Exercise 3: PML estimation in a duration model

The positive variables (durations) y_i, are generated by the model:

$$y_i = \beta_0 x_i \varepsilon_i \quad i = 1, \ldots, n. \quad \beta_0 > 0$$

where,

regressors x_i and errors ε_i are positive and such that (x_i, ε_i) are i.i.d, where x_i and ε_i are independent with $\mathbb{E}[\varepsilon_i] = 1$;

$\beta_0 > 0$ is an unknown scalar parameter.

Consider different pseudo-models for a PML estimation of β

3.1 Conditional expectation

Prove that

$$\mathbb{E}_0[y_i|x_i] = \beta_0 x_i$$

where $\mathbb{E}_0[\cdot]$ denotes expectation under the true model.

Proof

$$\begin{aligned}
\mathbb{E}_0[y_i|x_i] &= \mathbb{E}_0[\beta_0 x_i \varepsilon_i | x_i] \\
&= \beta_0 x_i \mathbb{E}_0[\varepsilon_i] \\
&= \beta_0 x_i
\end{aligned}$$

3.2 PML with exponential density

The econometrician considers the following parametric family of pseudo-densities:

$$f(y \mid x; \beta) = \frac{1}{\beta x} e^{-\frac{y}{\beta x}}, \quad y \geq 0, \quad \beta > 0$$

- *Compute the pseudo-true value defined by*

$$\beta_0^* = \arg\max_{\beta} \mathbb{E}_0 \left[\log f(y_i \mid x_i; \beta)\right]$$

and show that $\beta_0^* = \beta_0$.

Proof

$$\begin{aligned}
\mathbb{E}_0 \left[\log f(y \mid x, \beta)\right] &= \mathbb{E}_0 \left[-\log(\beta x) - \frac{y}{\beta x}\right] \\
&= -\log(\beta) - \mathbb{E}_0 \left[\log(x)\right] - \mathbb{E}_0 \left[\frac{\beta_0 x \varepsilon}{\beta x}\right] \\
&= -\log(\beta) - \mathbb{E}_0 \left[\log(x)\right] - \frac{\beta_0}{\beta}
\end{aligned}$$

Applying the First Order Condition:

$$\left.\frac{\partial \mathbb{E}_0 \left[\log f(y \mid x, \beta)\right]}{\partial \beta}\right|_{\beta = \beta_0^*} = -\frac{1}{\beta_0^*} + \frac{\beta_0}{\beta_0^{*2}} = 0$$

$$\implies \beta_0^* = \beta_0$$

•• *Is this result surprising? Explain it using the general PML theory derived in the course!*

This result is not surprising. Indeed, we can see that the parametric family we are considering is exponential linear:

$$\log f(y \mid m) = -\log(m) - \frac{1}{m} y = A(m) + B(y) + C(m) y$$

where $m = \beta x$, $A(m) := -\log(m)$, $B(y) = 0$ and $C(m) := -\frac{1}{m}$.

By assumption, the conditional mean m is correctly specified. From that, the PML theory guarantees $\beta_0^* = \beta_0$.

••• *Compute the PML estimator of β based on the exponential family:*

$$\hat{\beta} = \arg\max_{\beta} \sum_{i=1}^{n} \log f(y_i \mid x_i; \beta).$$

and show that

$$\hat{\beta} = \frac{1}{n} \sum_{i=1}^{n} \frac{y_i}{x_i}$$

Proof

It follows by applying the First order Condition:

$$\frac{\partial}{\partial \beta} \left(\sum_{i=1}^{n} \log f(y_i \mid x_i, \beta) \right) \Bigg|_{\beta = \hat{\beta}} = \frac{\partial}{\partial \beta} \left(\sum_{i=1}^{n} - \log(\beta x_i) - \frac{y_i}{\beta x_i} \right) \Bigg|_{\beta = \hat{\beta}} = 0$$

$$\implies \sum_{i=1}^{n} \left(-\frac{x_i}{\hat{\beta} x_i} + \frac{y_i}{\hat{\beta}^2 x_i} \right) = 0$$

$$\implies \hat{\beta} = \frac{1}{n} \sum_{i=1}^{n} \frac{y_i}{x_i}$$

Is the estimator $\hat{\beta}$ consistent?

According to the PML theory, assuming that β_0 is first-order identified, the PML estimator is consistent if (and only if) the pseudo-true conditional densities $f(y \mid m)$ is exponential linear.

From the previous point, we've verified the two hypothesis, so we can conclude that the PML estimator $\hat{\beta}$ is consistent for estimating β_0.

Despite this, we do the mathematical steps that lead us to verify the consistency:

$$\hat{\beta} = \frac{1}{n}\sum_{i=1}^{n}\frac{y_i}{x_i} = \frac{1}{n}\sum_{i=1}^{n}\frac{\beta_0 x_i \varepsilon_i}{x_i} = \frac{1}{n}\sum_{i=1}^{n}\beta_0\varepsilon_i.$$

Then, knowing that

$$\frac{1}{n}\sum_{i=1}^{n}\varepsilon_i \overset{n\to\infty}{\longrightarrow} \mathbb{E}[\varepsilon_i],$$

we obtain

$$\hat{\beta} = \beta_0\frac{1}{n}\sum_{i=1}^{n}\varepsilon_i \overset{n\to\infty}{\longrightarrow} \beta_0\mathbb{E}[\varepsilon_i] = \beta_0$$

which gives us the consistency of the estimator.

Give its asymptotic distribution.

If we call

$$\psi(\beta) = \log f(y \mid x, \beta) = -\log(\beta x) - \frac{y}{\beta x},$$

its derivatives are

$$\frac{\partial\psi(\beta)}{\partial\beta} = -\frac{1}{\beta} + \frac{y}{\beta^2 x} \qquad \frac{\partial^2\psi(\beta)}{\partial\beta\partial\beta'} = \frac{1}{\beta^2} - 2\frac{y}{\beta^3 x}$$

So the matrices are the following

$$J_0 = \mathbb{E}\left[-\frac{\partial^2\psi(\beta)}{\partial\beta\partial\beta'}\bigg|_{\beta=\beta_0}\right] = \mathbb{E}\left[-\frac{1}{\beta_0^2} + 2\frac{y}{\beta_0^3 x}\right] = -\frac{1}{\beta_0^2} + 2\,\mathbb{E}\left[\frac{\beta_0 x\varepsilon}{\beta_0^3 x}\right] = -\frac{1}{\beta_0^2} + \frac{2}{\beta_0^2} = \frac{1}{\beta_0^2}$$

$$I_0 = \mathbb{E}\left[\frac{\partial\psi(\beta)}{\partial\beta}\frac{\partial\psi(\beta)}{\partial\beta'}\bigg|_{\beta=\beta_0}\right] = \mathbb{E}\left[\frac{1}{\beta_0^2} + \frac{y^2}{\beta_0^4 x^2} - \frac{2y}{\beta_0^3 x}\right] = \frac{1}{\beta_0^2} + \frac{1}{\beta_0^2}\mathbb{E}\left[\varepsilon^2\right] - \frac{2}{\beta_0^2}$$

Now we compute $\mathbb{E}[\varepsilon^2]$. Since $\varepsilon = \frac{y}{\beta_0 x}$:

$$\mathbb{E}\left[\varepsilon^2\right] = \frac{1}{\beta_0^2}\mathbb{E}\left[\frac{y^2}{x^2}\right] = \frac{1}{\beta_0^2}\mathbb{E}\left[\frac{1}{x^2}\mathbb{E}\left[y^2 \mid x\right]\big|_{\beta=\beta_0}\right] = \frac{1}{\beta_0^2}\mathbb{E}[2\beta_0^2] = 2$$

In which it has been used:

$$\mathbb{E}\left[y^2 \mid x\right] = \int_0^{+\infty} y^2 \, f(y \mid x; \beta) \, dy = \int_0^{+\infty} y^2 \, \frac{1}{\beta x} e^{-\frac{y}{\beta x}} dy$$

We call $t = \frac{y}{\beta x}$ and the previous integral is equal to

$$\int_0^{+\infty} t^2 \beta^2 x^2 e^{-t} dt = \beta^2 x^2 \, \Gamma(3) = 2\beta^2 x^2$$

in which we've used the Gamma function:

$$\Gamma(n) = \int_0^{+\infty} x^{n-1} e^{-x} dx = (n-1)!$$

So the final matrices are:

$$\begin{aligned} J_0 &= \frac{1}{\beta_0^2} \\ I_0 &= \frac{1}{\beta_0^2} + \frac{1}{\beta_0^2} \mathbb{E}\left[\varepsilon^2\right] - \frac{2}{\beta_0^2} = -\frac{1}{\beta_0^2} + \frac{1}{\beta_0^2} 2 = \frac{1}{\beta_0^2} = J_0 \end{aligned}$$

Hence:

$$\sqrt{n}(\hat{\beta} - \beta_0) \xrightarrow{d} \mathcal{N}(0, \Sigma)$$

where $\Sigma = J_0^{-1} I_0 J_0^{-1} = \beta_0^2$.

3.3 PML with Weibull density

Let us now consider the density function on $\mathbb{R}^+$:

$$g(\varepsilon) = \frac{2\varepsilon}{c} e^{-\frac{\varepsilon^2}{c}}, \quad \varepsilon \geq 0 \quad c := \frac{4}{\pi}$$

with

$$\int_0^{+\infty} g(\varepsilon) \, d\varepsilon = \int_0^{+\infty} \varepsilon g(\varepsilon) \, d\varepsilon = 1.$$

- *Explain why*

$$\tilde{f}\left(y \mid x; \beta\right) = \frac{2y}{c(\beta x)^2} e^{-\frac{1}{c}\left(\frac{y}{\beta x}\right)^2}$$

with $\beta > 0$, defines a parametric family of conditional densities with correctly specified mean for our problem.

We can substitute $\varepsilon = \frac{y}{\beta x}$ in $g(\varepsilon)$

$$g\left(\frac{y}{\beta x}\right) = \frac{2y}{c\beta x} e^{-\frac{1}{c}\left(\frac{y}{\beta x}\right)^2}.$$

So that

$$\tilde{f}\left(y \mid x; \beta\right) = \frac{1}{\beta x} \, g\left(\frac{y}{\beta x}\right).$$

Firstly, we prove that

$$\int_0^{+\infty} \tilde{f}\,(y \mid x; \beta)\; dy = 1.$$

Indeed,

$$\int_0^{+\infty} \tilde{f}\,(y \mid x; \beta)\; dy = \int_0^{+\infty} \frac{1}{\beta x}\, g\left(\frac{y}{\beta x}\right)\, dy \quad \overset{\varepsilon=\frac{y}{\beta x}}{=} \quad \int_0^{+\infty} \frac{1}{\beta x} g(\varepsilon)\; \beta x\; d\varepsilon = 1$$

With the same substitution, we compute the conditional expectation and we get:

$$\mathbb{E}\left[y \mid x, \beta\right] = \int_0^{+\infty} y \frac{1}{\beta x}\, g\left(\frac{y}{\beta x}\right)\, dy \quad \overset{\varepsilon=\frac{y}{\beta x}}{=} \quad \beta x \int_0^{+\infty} \varepsilon g(\varepsilon)\; d\varepsilon = \beta x$$

which has correctly specified mean for our problem, consistent with what we've found in the previous points of the exercise.

•• *The PML estimator of β based on the family $\tilde{f}\,(y \mid x; \beta)$ is*

$$\tilde{\beta} = \arg\max_{\beta} \sum_i \log f(y_i \mid x_i; \beta).$$

Is $\tilde{\beta}$ a consistent PML estimator for estimating β_0? Explain your answer using the general PML theory.

We want to compute the PML estimator $\tilde{\beta}$:

$$\begin{aligned}\sum_i \left[\log \tilde{f}\,(y_i \mid x_i; \beta)\right] &= \sum_i \left[\log(2y_i) - \log(cx_i^2) - 2\log(\beta) - \frac{1}{c}\left(\frac{y_i}{\beta x_i}\right)^2\right] \\ &= -n\log c - 2n\log(\beta) + \sum_i \left(\log(2\beta_0 x_i \varepsilon_i)\right) - 2\sum_i \left(\log x_i\right) - \frac{1}{c}\frac{\beta_0^2}{\beta^2}\sum_i \left(\varepsilon_i^2\right)\end{aligned}$$

Applying the First Order Condition, we obtain

$$\frac{\partial}{\partial \beta}\sum_i \left[\log \tilde{f}\,(y_i \mid x_i; \tilde{\beta})\right]\Bigg|_{\beta=\tilde{\beta}} = -\frac{2\,n}{\tilde{\beta}} + \frac{2}{c}\frac{\beta_0^2}{\tilde{\beta}^3}\sum_i \varepsilon_i^2 = 0$$

$$\Longrightarrow \tilde{\beta} = \beta_0 \sqrt{\frac{1}{c}\frac{\sum_i \varepsilon_i^2}{n}}$$

As $n \to +\infty$, we take expectations:

$$\frac{1}{n}\sum_i \varepsilon_i^2 \to \mathbb{E}_0[\varepsilon_i^2] = 2$$

So we get

$$\tilde{\beta} = \beta_0 \sqrt{\frac{2}{c}} = \beta_0 \sqrt{\frac{\pi}{2}}$$

Therefore, $\tilde{\beta}$ is equal to β_0 times a constant.

The constant factor arises because the Weibull density is not the true distribution but it's instead an approximation. Specifically, the Weibull distribution approximates the true distribution in a way that introduces a scaling discrepancy captured by the factor $\sqrt{\frac{\pi}{2}}$.

As a result, $\tilde{\beta}$ consistently estimates a biased version of the true parameter β_0 due to model misspecification.

Another way to see the unconsistency of $\tilde{\beta}$ for β_0 is through the same theorem used in one of the previous points: assuming that β_0 is first-order identified, the PML estimator is consistent if (and only if) the pseudo-true conditional densities $\tilde{f}\,(y \mid m)$ is exponential linear.

For this purpose, we can write:

$$\log \tilde{f}\,(y \mid m) = \log(2y) - \log(cm^2) - \frac{1}{c}\left(\frac{y}{m}\right)^2$$

and we can see that:

$$\begin{aligned} &m = \beta x \\ &A(m) = -\log(cm^2) \\ &B(y) = \log(2y) \\ &C(m) = -\frac{1}{c}\frac{y}{m^2} \end{aligned}$$

But the last term depends on both y and m. So we can conclude that the PML estimator $\tilde{\beta}$ is not consistent for estimating β_0

Exercise 4: Jackknife

The aim of this project is to estimate the variance-covariance matrix of the OLSE of the parameters in linear regression by the jackknife method. Let's assume the linear model

$$y_i = x_i'\beta + \varepsilon_i \quad i = 1, \ldots, n$$

where $\beta \in \mathbb{R}^p$, $x_i \in \mathbb{R}^p$ and ε_i are i.i.d. random variables with some distribution F such that $\mathbb{E}[\varepsilon_i] = 0$.

Let $\hat{\beta}$ be the OLSE of β and let $\hat{\varepsilon}_i = y_i - x_i'\hat{\beta}$ be the i^{th} residual.

4.1 Equalities

Show that

$$\hat{\beta}_{(i)} = \hat{\beta} - \left(X'X\right)^{-1} x_i \hat{\varepsilon}_i^m$$

using:

$$\begin{cases} X'_{(i)} y_{(i)} = X'y - x_i y_i \\ \left(X'_{(i)} X_{(i)}\right)^{-1} = (X'X)^{-1} + \frac{(X'X)^{-1} x_i x_i' (X'X)^{-1}}{1 - m_{ii}} \\ m_{ii} = x_i' (X'X)^{-1} x_i \\ \hat{\varepsilon}_i^m = \frac{\hat{\varepsilon}_i}{1 - m_{ii}} \end{cases}$$

Proof

We know that $\hat{\beta} = \left(X'X\right)^{-1} X'y$, therefore, since writing (i) means removing the i-th component, we have that

$$\hat{\beta}_{(i)} = \left(X'_{(i)} X_{(i)}\right)^{-1} X'_{(i)} y_{(i)}$$

which we can expand and get:

$$\begin{aligned} \hat{\beta}_{(i)} &= \left(X'_{(i)} X_{(i)}\right)^{-1} X'_{(i)} y_{(i)} \\ &= \left[\left(X'X\right)^{-1} + \frac{\left(X'X\right)^{-1} x_i x_i' \left(X'X\right)^{-1}}{1 - m_{ii}}\right] \left(X'y - x_i y_i\right) \\ &= \left(\hat{\beta} - \left(X'X\right)^{-1} x_i y_i\right) + \frac{1}{1 - m_{ii}} \left[\left(X'X\right)^{-1} x_i x_i' \hat{\beta} - \left(X'X\right)^{-1} x_i m_{ii} y_i\right] \\ &= \hat{\beta} - \left(X'X\right)^{-1} x_i y_i + \frac{\left(X'X\right)^{-1} x_i}{1 - m_{ii}} \left[x_i' \hat{\beta} - m_{ii} y_i\right] \\ &= \hat{\beta} - \left(X^T X\right)^{-1} x_i y_i + \frac{\left(X'X\right)^{-1} x_i}{1 - m_{ii}} \left[y_i - \hat{\varepsilon}_i - m_{ii} y_i\right] \\ &= \hat{\beta} - \left(X'X\right)^{-1} x_i y_i + \frac{\left(X'X\right)^{-1} x_i}{1 - m_{ii}} \left(y_i - \hat{\varepsilon}_i\right) - \frac{\left(X'X\right)^{-1} x_i}{1 - m_{ii}} m_{ii} y_i \\ &= \hat{\beta} - \frac{(1 - m_{ii}) \left(X'X\right)^{-1} x_i y_i + \left(X'X\right)^{-1} x_i y_i - \left(X'X\right)^{-1} x_i m_{ii} y_i}{1 - m_{ii}} - \left(X'X\right)^{-1} x_i \hat{\varepsilon}_i^m \\ &= \hat{\beta} - \left(X'X\right)^{-1} x_i \hat{\varepsilon}_i^m \end{aligned}$$

4.2 Interpretation

Give the interpretation of the difference $\hat{\beta}_{(i)} - \hat{\beta}$.

$\hat{\beta}_{(i)}$ is the OLS estimator computed without considering the i^{th} measure. While $\hat{\beta}$ is the OLS estimator computed over the complete data-set.

Therefore, the quantity $(\hat{\beta}_{(i)} - \hat{\beta})$ quantify the amount of influence induced by the absence of the x_i measure on the regression result.

For instance, if $(\hat{\beta}_{(i)} - \hat{\beta})$ is small , then the x_i has little influence over the regression , and viceversa.

Therefore, this quantity can be used to identify observations (x_i) that disproportionately affect the regression result, like outliers or high leverage measures (high m_{ii} value)

4.3 Jackknife estimator

Show that the jackknife estimator of the variance-covariance matrix of (the random vector) $\hat{\beta}$ is given by

$$\hat{V}_{\text{Jack}}\left(\hat{\beta}\right) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \left(\hat{\beta}^{*i} - \hat{\beta}^{*\cdot}\right)\left(\hat{\beta}^{*i} - \hat{\beta}^{*\cdot}\right)'$$

where the $\hat \beta^{i}arethepseudo-valuesand\hat \beta^{.}$ their mean.

which can be rewritten as:

$$\hat{V}_{\text{Jack}}(\hat{\beta}) = \frac{n-1}{n}(X'X)^{-1}\left[\sum_{i=1}^{n} x_i x_i' (\hat{\varepsilon}_i^m)^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i \hat{\varepsilon}_i^m\right)\left(\sum_{i=1}^{n} x_i' \hat{\varepsilon}_i^m\right)\right](X'X)^{-1}$$

Proof

We know that every estimator $T^{*i} = T_n + (n-1)\left(T_n - T_{(i)}\right)$. Hence, in our case for $\hat{\beta}^{*i}$ estimator we have

$$\hat{\beta}^{*i} = \hat{\beta} + (n-1)\left(\hat{\beta} - \hat{\beta}_{(i)}\right)$$

$$\hat{\beta}^{*\cdot} = \frac{\sum_i \hat{\beta}^{*i}}{n}$$

and, to semplify the notation, we'll assume $A := (X'X)^{-1}$.

$$
\begin{aligned}
\hat{V}_{\text{Jack}}\left(\hat{\beta}\right) &= \frac{1}{n(n-1)}\sum_{i=1}^{n}\left(\hat{\beta}^{*i}-\hat{\beta}^{*\cdot}\right)\left(\hat{\beta}^{*i}-\hat{\beta}^{*\cdot}\right)' \\
&= \frac{1}{n(n-1)}\sum_{i}\left\{\left[\hat{\beta}+(n-1)\left(\hat{\beta}-\hat{\beta}_{(i)}\right)-\frac{1}{n}\sum_{j}\left(\hat{\beta}+(n-1)\left(\hat{\beta}-\hat{\beta}_{(j)}\right)\right)\right]\cdot\right. \\
&\left.\cdot\left[\hat{\beta}+(n-1)\left(\hat{\beta}-\hat{\beta}_{(i)}\right)-\frac{1}{n}\sum_{j}\left(\hat{\beta}+(n-1)\left(\hat{\beta}-\hat{\beta}_{(j)}\right)\right)\right]'\right\} \\
&= \frac{1}{n(n-1)}\sum_{i}\left\{\left[\hat{\beta}+(n-1)Ax_i\hat{\varepsilon}_i^m-\frac{1}{n}\sum_{j}\left(\hat{\beta}+(n-1)Ax_j\hat{\varepsilon}_j^m\right)\right]\cdot\right. \\
&\left.\cdot\left[\hat{\beta}+(n-1)Ax_i\hat{\varepsilon}_i^m-\frac{1}{n}\sum_{j}\left(\hat{\beta}+(n-1)Ax_j\hat{\varepsilon}_j^m\right)\right]'\right\} \\
&= \frac{1}{n(n-1)}\sum_{i}\left\{\hat{\beta}\hat{\beta}'+(n-1)\hat{\beta}\hat{\varepsilon}_i^m x_i'A'-\hat{\beta}\hat{\beta}'-\frac{(n-1)}{n}\hat{\beta}\sum_{j}\hat{\varepsilon}_j^m x_j'A'+\right. \\
&+(n-1)Ax_i\hat{\varepsilon}_i^m\hat{\beta}'+(n-1)^2Ax_i\hat{\varepsilon}_i^m\hat{\varepsilon}_i^{m'}x_i'A'-(n-1)Ax_i\hat{\varepsilon}_i^m\hat{\beta}'-\frac{(n-1)^2}{n}Ax_i\hat{\varepsilon}_i^m\sum_{j}\hat{\varepsilon}_j^m x_j'A'+ \\
&-\hat{\beta}\hat{\beta}'-\hat{\beta}(n-1)\hat{\varepsilon}_i^m x_i'A'+\hat{\beta}\hat{\beta}'+\frac{(n-1)}{n}\hat{\beta}\sum_{j}\hat{\varepsilon}_j^m x_j'A'-\frac{(n-1)}{n}\sum_{j}Ax_j\hat{\varepsilon}_j^m\hat{\beta}'+ \\
&\left.-\frac{(n-1)^2}{n}\sum_{j}Ax_j\hat{\varepsilon}_j^m\hat{\varepsilon}_i^{m'}x_i'A'+\frac{(n-1)}{n}\sum_{j}Ax_j\hat{\varepsilon}_j^m\hat{\beta}'+\frac{(n-1)^2}{n^2}\sum_{j}Ax_j\hat{\varepsilon}_j^m\sum_{k}\hat{\varepsilon}_k^{m'}x_k'A'\right\} \\
&= \frac{n-1}{n}A\left[\sum_{i}x_ix_i'(\hat{\varepsilon}_i^m)^2-\frac{1}{n}\left(\sum_{i}x_i\hat{\varepsilon}_i^m\right)\left(\sum_{i}x_i'\hat{\varepsilon}_i^m\right)\right]A' \\
&= \frac{n-1}{n}(X'X)^{-1}\left[\sum_{i}x_ix_i'(\hat{\varepsilon}_i^m)^2-\frac{1}{n}\left(\sum_{i}x_i\hat{\varepsilon}_i^m\right)\left(\sum_{i}x_i'\hat{\varepsilon}_i^m\right)\right](X'X)^{-1}
\end{aligned}
$$

4.4 Approximation of the jackknife estimator

Consider an approximation of the Jackknife estimator obtained at point 3. by replacing $1-m_{ii}$ *by* 1. *When is this approximation justified? (Compute the average value of the* m_{ii}*)*

We know the Projection matrix has the form $M=X(X'X)^{-1}X'$, which has the property $Tr\,(M)=rank(M)$

$$
\bar{m}_{ii}=\frac{\sum_{i=1}^{n}m_{ii}}{n}=\frac{\sum_{i=1}^{n}x_i'(X'X)^{-1}x_i}{n}=\frac{Tr\left(X'(X'X)^{-1}X\right)}{n}=\frac{Tr\,(M)}{n}=\frac{rank\,(M)}{n}
$$

Since, $rank(M)=p$ (the design matrix has to be invertible), it means that for $\frac{p}{n}\ll 1$ (large sample-size), the average influence from each x_i over the OLS estimator becomes smaller and smaller. Hence, for large samples

$$
m_{ii}\to 0 \quad \Rightarrow \quad \hat{\varepsilon}_i^m\to\hat{\varepsilon}_i \quad \forall i
$$

4.5 Estimator proposed by H. White

Given the formula of Jackknife estimator of point 3. obtained by replacing $1 - m_{ii}$ *by* 1 *and verify that this estimator is the one proposed by H. White (1980),*
$\text{A Heteroskedasticity consistent Covariance Matrix Estimator and a Direct Test}$
$\text{for Heteroskedasticity, Econometrica}$*, pp. 817-838. Give the exact location in that paper where we find this estimator*

On page 820, line 16, H. White's paper provides an heteroskedasticity-consistent covariance matrix estimator

$$\hat{V}_n(\hat{\beta}) = \frac{1}{n} \sum_i \hat{\varepsilon}_i^2 x_i x_i'$$

$$\Sigma_{White} = \left(\frac{X'X}{n}\right)^{-1} \hat{V}_n(\hat{\beta}) \left(\frac{X'X}{n}\right)^{-1}$$

Where, $\hat{\varepsilon}_i = y_i - x_i \hat{\beta}$ is the empricical OLS squared residual.

The Jackknife estimator captures the variability of $\hat{\beta}$ by leaving out one observation at a time, recomputing the OLS estimator. The formula for the Jackknife Convariant--matrix obtained at point 3. is:

$$\hat{V}_{Jack}(\hat{\beta}) = \frac{n-1}{n} (X'X)^{-1} \left[\sum_i x_i x_i' (\hat{\varepsilon}_i^m)^2 - \frac{1}{n} \left(\sum_i x_i \hat{\varepsilon}_i^m \right) \left(\sum_i x_i' \hat{\varepsilon}_i^m \right) \right] (X'X)^{-1}$$

In large samples $n \gg p$, the influence of any single observation on $\hat{\beta}$ becomes negligible ($m_{ii} \to 0$). As a result, The Jackknife residuals $\hat{\varepsilon}_i^m$ converge to the full-sample OLS residuals $\hat{\varepsilon}_i$.

The second term is negligible due to the orthogonality condition $\mathbb{E}_n[x\hat{\varepsilon}]$ OLS.

Thus, the Jackknife estimator simplifies to:

$$\hat{V}_{Jack}(\hat{\beta}) \approx (X'X)^{-1} \, n \, \hat{V}_n(\hat{\beta}) \, (X'X)^{-1}$$

The Jackknife Covariance-matrix decreases (for consistency) as n grows because the variance of $\hat{\beta}$ scales as $1/n$. In order to obtain the asymptotic Covariance-matrix we multiply by n, making it $\mathcal{O}(1)$

$$n \cdot \hat{V}_{Jack}(\hat{\beta}) = \left(\frac{X'X}{n}\right)^{-1} \hat{V}_n(\hat{\beta}) \left(\frac{X'X}{n}\right)^{-1} = \Sigma_{White}$$

Which is exactly the White's asymptotic Heteroskedasticity Covariance matrix estimator which appear in the uniform convergence for $\hat{\beta}$

$$\sqrt{n} \left(\hat{\beta} - \hat{\beta}_0 \right) \sim \mathcal{N}\left(0, \Sigma\right)$$

What is the relationship between the JackKnife's and White's formulation for the consistent Covariance-matrix estimator?

- Both deal with heteroskedasticity, but with different approaches.
- The jackknife Covariance matrix implicitly captures the variability for $\hat{\beta}$ by leaving out one measure at a time x_i and recomputing the OLS regression. On the other hand White's Covariance Matrix captures explicitly the $\hat{\beta}$ variability by weighting each observation with its squared empirical residual $\hat{\varepsilon}_i^2$ to approximate the heteroskedasticity structure.
- The Jackkinfe asymptotic consistent Covariance-matrix converges to the White's consistent Covariance-matrix because in the limit $n \gg p$, each x_i influence over the $\hat{\beta}$ variability becomes smaller and smaller ($m_{ii} \to 0$). Therefore, $\hat{\varepsilon}_i^m$ converges to the empirical residuals $\hat{\varepsilon}_i$, which is used in the White's consistent Covariance-matrix estimator.

4.6 Summary of the article

Read p. 817, 820 (first half), and p. 821 (first half) of the cited article and summarize in at most one page the problem and the proposed solution.

H. White addresses the problem of estimating the precision of parameter estimates in a linear regression model under the presence of Heteroskedasticity.

$$Y_i = X\beta + \varepsilon_i$$

Where (X_i, ε_i) are i.i.d. and $\mathbb{E}\left[\varepsilon_i\right] = 0$. In this context, $\mathbb{E}\left[\varepsilon_i^2 \mid X\right] = g(X)$. Here, $g(X)$ in unknown and no assumption are done over its nature. Additionally, ε_i is not measurable. In this setting, traditional methods for assessing the precision of parameter estimates (i.e. OLS) fail because they assume homoskedasticity ($\mathbb{E}[\varepsilon_i^2] = \sigma_0^2$), which is not the case here.

H. White propose a consistent Covariance-matrix estimator which allows to asses the precision of the parameter estimates under Heterokedasticity assumption.

$$\hat{V}_n(\hat{\beta}) = \frac{1}{n}\sum_i \hat{\varepsilon}_i^2 x_i x_i'$$

White demonstrates that the estimator $\hat{V}_n\left(\hat{\beta}\right)$ is consistent and that the parameter estimates coverges to

$$\sqrt{n}\left(\hat{\beta} - \hat{\beta}_0\right) \sim \mathcal{N}\left(0, \Sigma_{White}\right)$$

Where, $\Sigma_{White} = \left(\frac{X'X}{n}\right)^{-1} \hat{V}_n(\hat{\beta}) \left(\frac{X'X}{n}\right)^{-1}$

In the special case White's estimator converges to the homoskedasticity Covariance-matrix $\hat{V}_n(\hat{\beta}_n) = \sigma_0^2\left(\frac{X'X}{n}\right)$

Moreover, it can be shown that this estimator is appropriate to construct asymptotic confidence intervals and also to solve the problem of testing linear hypothesis. Specifically, White shows the specific test statistics:

$$\mathcal{H}_0 : \gamma_0 = R\beta_0$$

Where $\gamma_0 \in \mathbb{R}^r$ and R is a finite $r \times K$ full row rank (K is the number of representative variables). Throught the White's estimator it is possible to to perform a $\chi 2$ test

$$n\left(R\hat{\beta}_n - \gamma_0\right)' \left[R\left(\frac{X'X}{n}\right)^{-1} \hat{V}_n(\hat{\beta})\left(\frac{X'X}{n}\right)^{-1} R'\right] \left(R\hat{\beta}_n - \gamma_0\right) \sim \chi_r^2$$

This allows for robust inference even in the presence of heteroskedasticity.

4.7 Jackknife and Bootstrap estimator

Compare the Jackknife estimator derived at point 3. with a bootstrap estimator.

To answer this question, the dataset sales.csv is considered

It has:

- 200×4 data, labeled as follow, $TV, Radio, Newspaper, Sales$
- Sales is the variable to predict by using the other features. We want to achieve this with OLS
- From the EDA we saw that TV and Sales have a parabolic correlation. Hence a quare root trasnformation over TV is considered.
- We assed the presence of conditional Heterokedasticity with a White's test, which indicates that at 10% c.l. the presence of moderate hetherokedasticity ($p_{value} = 0.07$).

Both Bootstrap and Jackknife Covariance-matrices estimators are computed and compared

The following model is considered $Sales = \left[1, \sqrt{TV}, Radio, Newspaper\right] \beta + \varepsilon$.

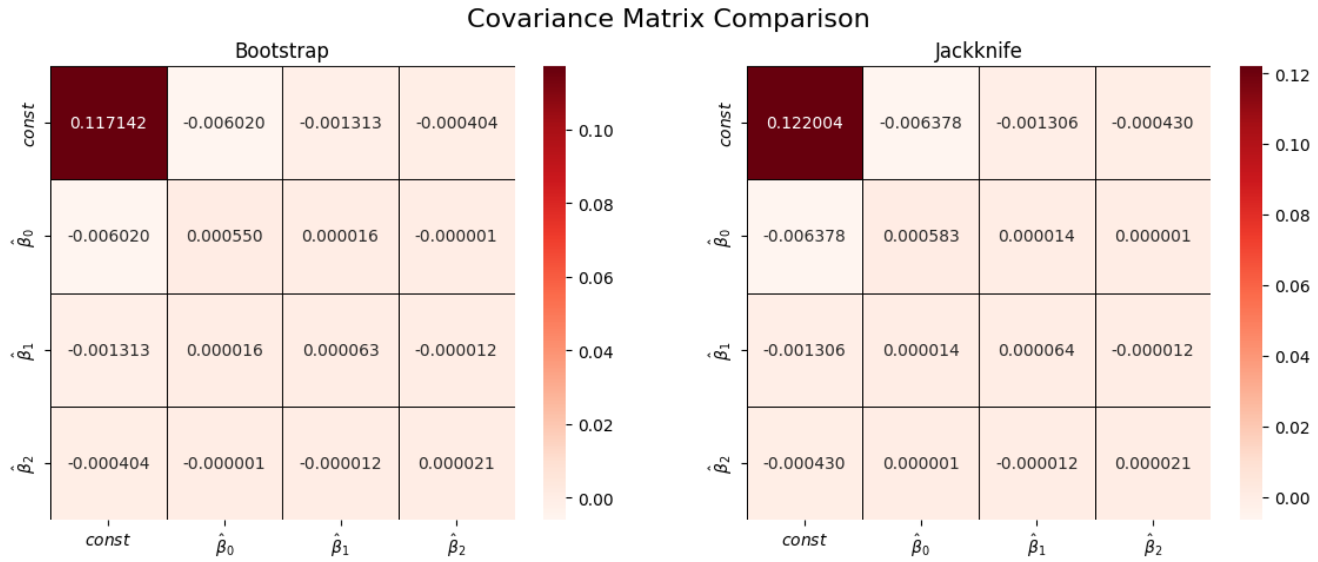
Where, for the Bootstrap we have

$$\hat{V}_{BS}(\hat{\beta}) = \frac{1}{B-1} \sum_{i=1}^{B} \left(\hat{\beta}^{i*} - \hat{\beta}^{*\cdot}\right) \left(\hat{\beta}^{i*} - \hat{\beta}^{*\cdot}\right)'$$

While, for the Jackknife has been used the formula obtained at point 3.

$$\hat{V}_{\text{Jack}}(\hat{\beta}) = \frac{n-1}{n} (X'X)^{-1} \left[\sum_{i=1}^{n} x_i x_i' (\hat{\epsilon}_i^m)^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \hat{\epsilon}_i^m\right) \left(\sum_{i=1}^{n} x_i' \hat{\epsilon}_i^m\right)\right] (X'X)^{-1}$$

Results



Key matrix differences:

- The Covariance-matrix elements for both methods are close, with a percentage difference less than 5%, for the diagonal elements.
- The Jackkinfe Covariance Matrix tends to have slightly larger values on the diagonal elements compared to the Bootstrap Covariance Matrix

Computational differences:

- To compute the Bootstrap were used B (10'000) Bootstrap samples with replacement from the original data, in order estimate an equal B-number of $\hat{\beta}$ estimators, which have been used to obtain the Bootstrap Covariance-matrix.
- The Jackknife covariance matrix was way more straightforward and computationally efficient.
- (in the appendix) It can be shown, that for small datasets (this case), the Jackknife method does not allow to construct accurate confidence intervals.

In conclusion, in this specific contex, with a moderate degree of conditional Heterokedasticity, it is computationally efficient and less intensive to stimate the precision of $\hat{\beta}$ with the Jackknife Covariance-matrix estimator, with less than 5% differece (understimated) with respect to the Bootstrap Covariant-matrix.

Exercise 5. Wild bootstrap

*The aim of this project is to implement the so-called "wild bootstrap" procedure to assess the uncertainty of OLS estimates of a linear model, and compare it with the uncertainty assessments of "paired bootstrap" and "residual bootstrap" procedures. Data containing information on **medical expenses (E)**, **standardized income (I)** and **smoking habit (S, taking value 1 for a smoker and 0 for a non smoker)**.*

Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from statsmodels.stats.diagnostic import het_white
```

Global Variables

In [2]:
```
# number of bootstrap samples
B_samples = 10000
# number of bins for all the histograms
N_bins = 100
# alpha for CI
alpha = 0.05
```

Usefull functions

In [3]:
```
def confInt (true_betas, model_betas, R2_true, R2_model, confidence_level = alpha):
    """
    Parameters:
            true_betas: array with the true coefficients OLS
            model_betas: array with the bootstrap samples of the beta coefficients
            R2_true: array with the true R^2 OLS
            R2_model: array with the bootstrap samples of the adjusted R^2
            confidence_level: confidence level for the confidence interval - Percentile Method
    Return: CI_betas, CI_R2 arrays containing the confidence intervals
    """
    # Left and Right Confidence Intervals (K,2)
    CI_betas = np.zeros( (model_betas.shape[1],2) )
    # Left and Right Confidence Intervals (1,2)
    CI_R2 = np.zeros( (1,2) )

    # Quantiles
    probs = [confidence_level/2, 1-confidence_level/2]

    for i in range(model_betas.shape[1]):
        # Compute Quantiles for betas
        Qs = np.quantile(model_betas[:,i] - true_betas[i], probs)
        CI_betas[i] = (true_betas[i] - Qs[1], true_betas[i] - Qs[0])

    # Compute Quantiles for R2
    Qs = np.quantile(R2_model - R2_true, probs)
    CI_R2 = (R2_true - Qs[1], R2_true - Qs[0])

    return np.array(CI_betas), np.array(CI_R2)
```

In [4]:
```python
def plots (beta_all, beta_OLS, R_2_adj_all, R_2_adj_ols, bins = N_bins, CI = False):
    """
    Parameters:
            beta_all: array with all the bootstrap samples of the beta coefficients
            beta_OLS: array with the beta coefficients from the OLS model
            R_2_adj_all: array with all the bootstrap samples of the adjusted R^2
            R_2_adj_ols: adjusted R^2 from the OLS model
            bins: number of bins for the histograms
            CI: boolean to plot the confidence intervals
    Return: plots and arrays containing the means, standard deviations Normal Fits and CI
    """

    fig, axes = plt.subplots(2,2, figsize = (15, 7.5))

    axes = axes.ravel()

    beta_means = np.zeros(3)
    beta_stds = np.zeros(3)

    R_2_adj_means = np.zeros(1)
    R_2_adj_stds = np.zeros(1)

    beta_names = ['a', 'b', 'c']

    CI_betas, CI_R2 = confInt(beta_OLS, beta_all, R_2_adj_ols, R_2_adj_all)

    for i in range(4):

        if i < 3:
            axes[i].hist(beta_all[:,i], bins = bins, range = (min(beta_all[:,i]), max(beta_all[:,i])), alp
            axes[i].set_title(beta_names[i])
            axes[i].set_ylabel('Density')
            axes[i].axvline(beta_OLS[i], color='blue', linestyle='dashed', linewidth=2, label = 'OLS ('+be

            # norm.fit
            mean, std = norm.fit(beta_all[:,i])
            x = np.linspace(min(beta_all[:,i]), max(beta_all[:,i]), 100)
            axes[i].plot(x, norm.pdf(x, mean, std), linewidth=1, linestyle='dashed',color = 'red',label=f'

            beta_means[i] = mean
            beta_stds[i] = std

            if CI :
                axes[i].axvline(CI_betas[i,0], color='black', linestyle='dashed', linewidth=2, label = r'T
                axes[i].axvline(CI_betas[i,1] , color='black', linestyle='dashed', linewidth=2)

            axes[i].legend()

        else:
            axes[i].hist(R_2_adj_all, bins = bins, range = (min(R_2_adj_all), max(R_2_adj_all)), color='re
            axes[i].set_title(r'$R^{2}_{adj}$')
            axes[i].set_ylabel('Density')

            # norm.fit
            mean, std = norm.fit(R_2_adj_all)
            x = np.linspace(min(R_2_adj_all), max(R_2_adj_all), 100)
            axes[i].plot(x, norm.pdf(x, mean, std), linewidth=2, linestyle='dashed',color = 'red', label=f
            axes[i].axvline(R_2_adj_ols, color='blue', linestyle='dashed', linewidth=2, label = r'OLS ($R^

            R_2_adj_means[0] = mean
            R_2_adj_stds[0] = std

            #if CI :
                #axes[i].axvline(CI_R2[0], color='black', linestyle='dashed', linewidth=2, label = r'Two T
                #axes[i].axvline(CI_R2[1] , color='black', linestyle='dashed', linewidth=2)

            axes[i].legend()

    fig.suptitle(r"$E = a + b \cdot I + c \cdot S$", fontsize=16)

    plt.tight_layout()

    if CI:
        return np.array(beta_means), np.array(beta_stds), np.array(R_2_adj_means), np.array(R_2_adj_stds),
    else:
        return np.array(beta_means), np.array(beta_stds), np.array(R_2_adj_means), np.array(R_2_adj_stds)
```

5.1 File reading and basic informations

In [5]:
```
df_data = pd.read_csv('medical.csv')
df_data.head(n=10)
```

Out[5]:

	E	I	S
0	0.966546	-0.439503	0.0
1	2.683663	-0.553907	1.0
2	-0.011327	-0.917397	0.0
3	0.507619	-0.598209	0.0
4	1.570200	-0.292257	0.0
5	4.109109	-0.025166	1.0
6	2.591457	1.678851	0.0
7	0.337843	-0.260459	0.0
8	0.086258	1.332759	0.0
9	3.408870	1.193958	0.0

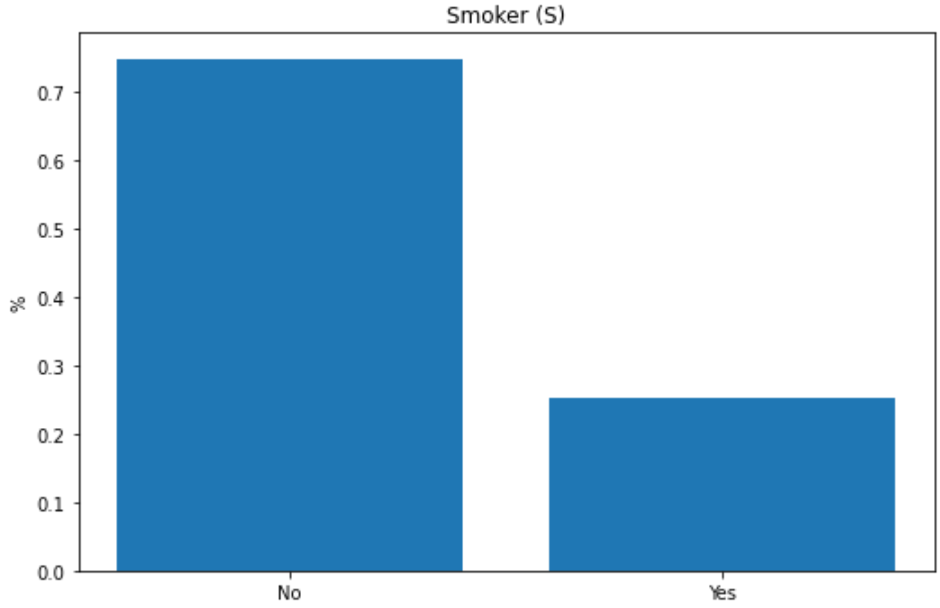
In [6]:
```
df_data.info()
```

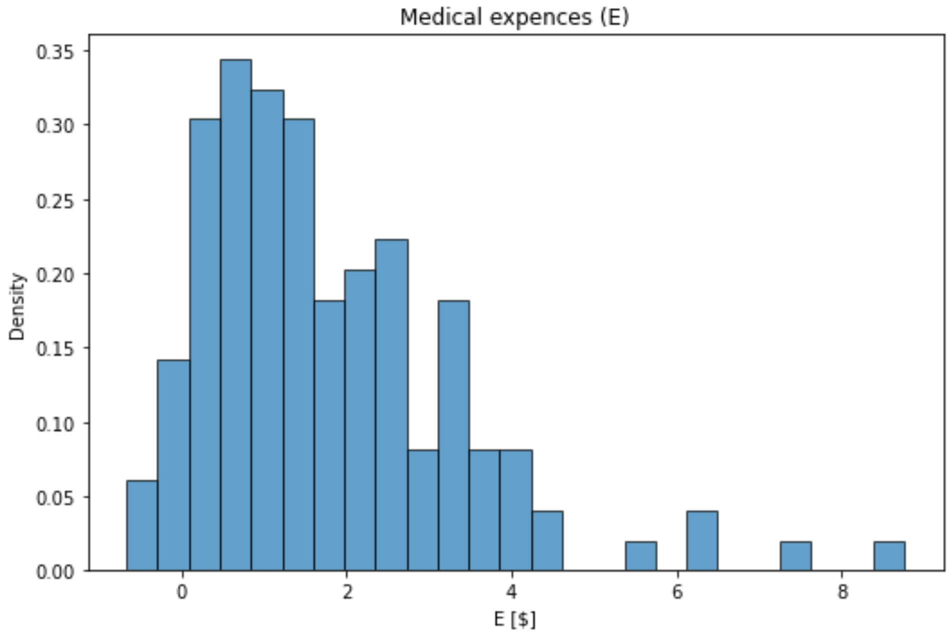
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131 entries, 0 to 130
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   E       131 non-null    float64
 1   I       131 non-null    float64
 2   S       131 non-null    float64
dtypes: float64(3)
memory usage: 3.2 KB
```

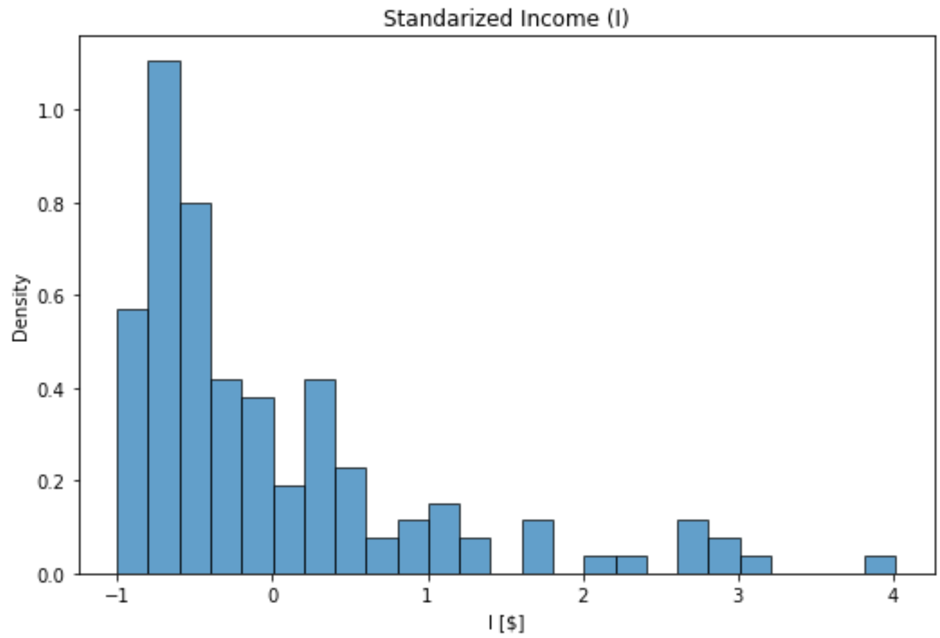
5.2 Summary statistics

In [7]:
```
fig, axes = plt.subplots(3,1, figsize=(7.5, 15))

axes = axes.ravel()
counts = df_data['S'].value_counts(normalize=True)
axes[0].bar(x=['0', '1'], height=counts)
axes[0].set_title('Smoker (S)')
axes[0].set_ylabel('%')
axes[0].set_xticklabels(['No', 'Yes'])

axes[1].hist(df_data['E'], bins=25, range = (min(df_data['E']), max(df_data['E'])), alpha=0.7, rwidth=1, e
axes[1].set_title('Medical expences (E)')
axes[1].set_xlabel('E [$]')
axes[1].set_ylabel('Density')

axes[2].hist(df_data['I'], bins=25, range = (min(df_data['I']), max(df_data['I'])), alpha=0.7, rwidth=1, e
axes[2].set_title('Standarized Income (I)')
axes[2].set_xlabel('I [$]')
axes[2].set_ylabel('Density')

plt.tight_layout()

plt.show()
```

```
/tmp/ipykernel_102/4030781087.py:8: UserWarning: FixedFormatter should only be used together with FixedLo
cator
  axes[0].set_xticklabels(['No', 'Yes'])
```

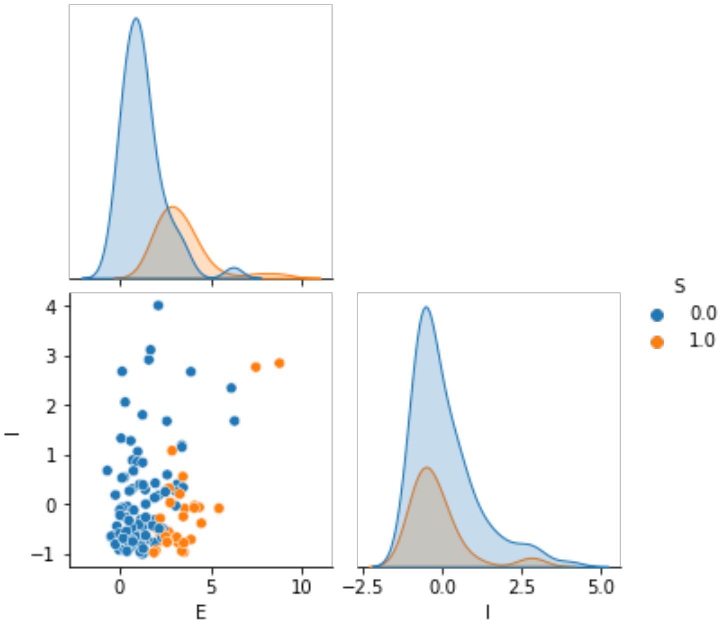






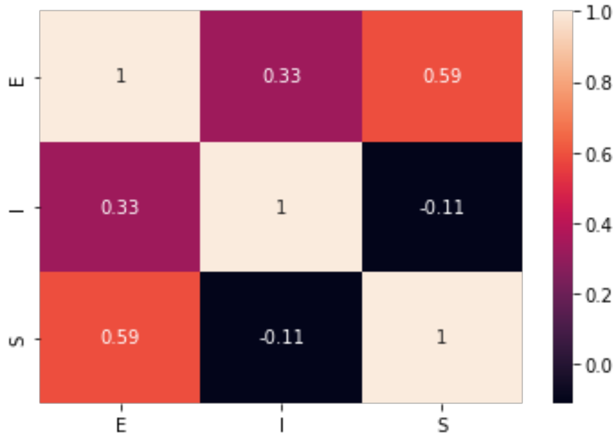
In [8]:
```
sns.pairplot(df_data, hue = 'S', corner=True)
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x7f07e6d19f70>



In [9]:
```
sns.heatmap(df_data.corr(), annot=True)
```

Out[9]: <AxesSubplot:>

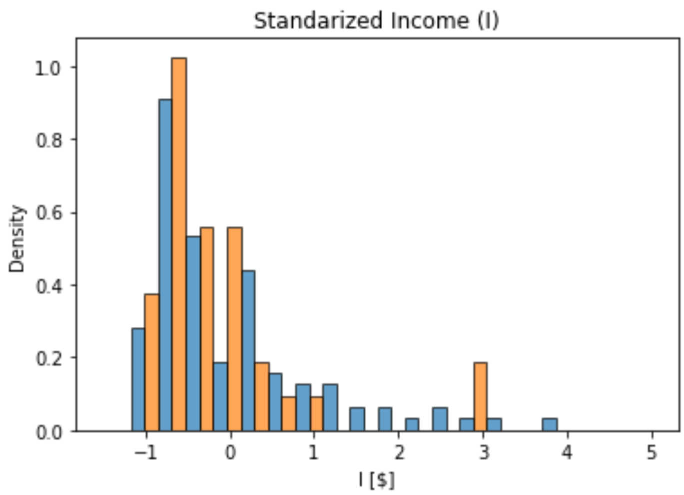


The only significant correlation is between smoking habits and medical expenses.

Regarding the relationship between income and smoking habits, there is no relevant correlation in this dataset.

In [10]:
```
smoker_I = df_data.query('S < 0.5')['I']
no_smoker_I = df_data.query('S > 0.5')['I']
plt.hist([smoker_I, no_smoker_I], bins=20, range = (-1.5, 5), alpha=0.7, rwidth=1, edgecolor="black", dens
plt.title('Standarized Income (I)')
plt.xlabel('I [$]')
plt.ylabel('Density')
```

Out[10]: Text(0, 0.5, 'Density')



The distribution of the income for both smokers and not smokers is almost the same, hence there is no correlation between this parameters.

5.3 OLS

We will use the following model to estimate the medical expenses:

$$E_i = a + bI_i + cS_i + \varepsilon_i \qquad i = 1, \ldots, n$$

In [11]:
```
def OLS (X , Y, const = False):
    """
    X = rapresentative variables (I, S)
    Y = medical expenses (E)
    const = True if we want to include the constant term in the model
    return:
        beta = estimated coefficients
        res = residuals array
        Y_pred = the medical expenses prediction
        R_2_adj = adjusted R^2 from the model
    """

    K_vals = X.shape[1]   # Constant term

    if const:
        X = np.c_[np.ones(X.shape[0]), X]
        k_vals = X.shape[1] + 1

    beta = np.linalg.solve(X.T @ X, X.T @ Y)
    Y_pred = X @ beta
    res = Y - Y_pred

    # R^2_adj
    ss_res = np.sum(res**2)
    ss_tot = np.sum((Y - Y.mean())**2)

    if ss_tot == 0:
        R_2_adj = 0
    else:
        R_2_adj = 1 - (ss_res / ss_tot)*(X.shape[0] - 1) / (X.shape[0] - K_vals)

    return np.array(beta) , np.array(res), np.array(Y_pred), np.array(R_2_adj)
```

In [12]:
```
X = np.array(df_data[['I', 'S']])
Y = np.array(df_data['E'])
beta_OLS, res_OLS, Y_pred, R_2_adj = OLS(X, Y, const = True)

Y_pred = pd.DataFrame(Y_pred, columns = ['E_OLS_pred'])
```

In [13]:
```
print('Beta_OLS: ', np.round(beta_OLS,2))
```

```
Beta_OLS:  [1.19 0.62 2.3 ]
```

In [14]:
```
R_2_ols = 1 - np.sum((df_data['E'] - (beta_OLS[0] + beta_OLS[1]*df_data['I'] + beta_OLS[2]*df_data['S']))*
R_2_adj_ols = 1 - np.sum((df_data['E'] - (beta_OLS[0] + beta_OLS[1]*df_data['I'] + beta_OLS[2]*df_data['S'
```

In [15]:
```
print ('R^2: ', np.round(R_2_ols,5))
print ('R^2_adj: ', np.round(R_2_adj_ols,5))
```

```
R^2:  0.50446
R^2_adj:  0.49672
```

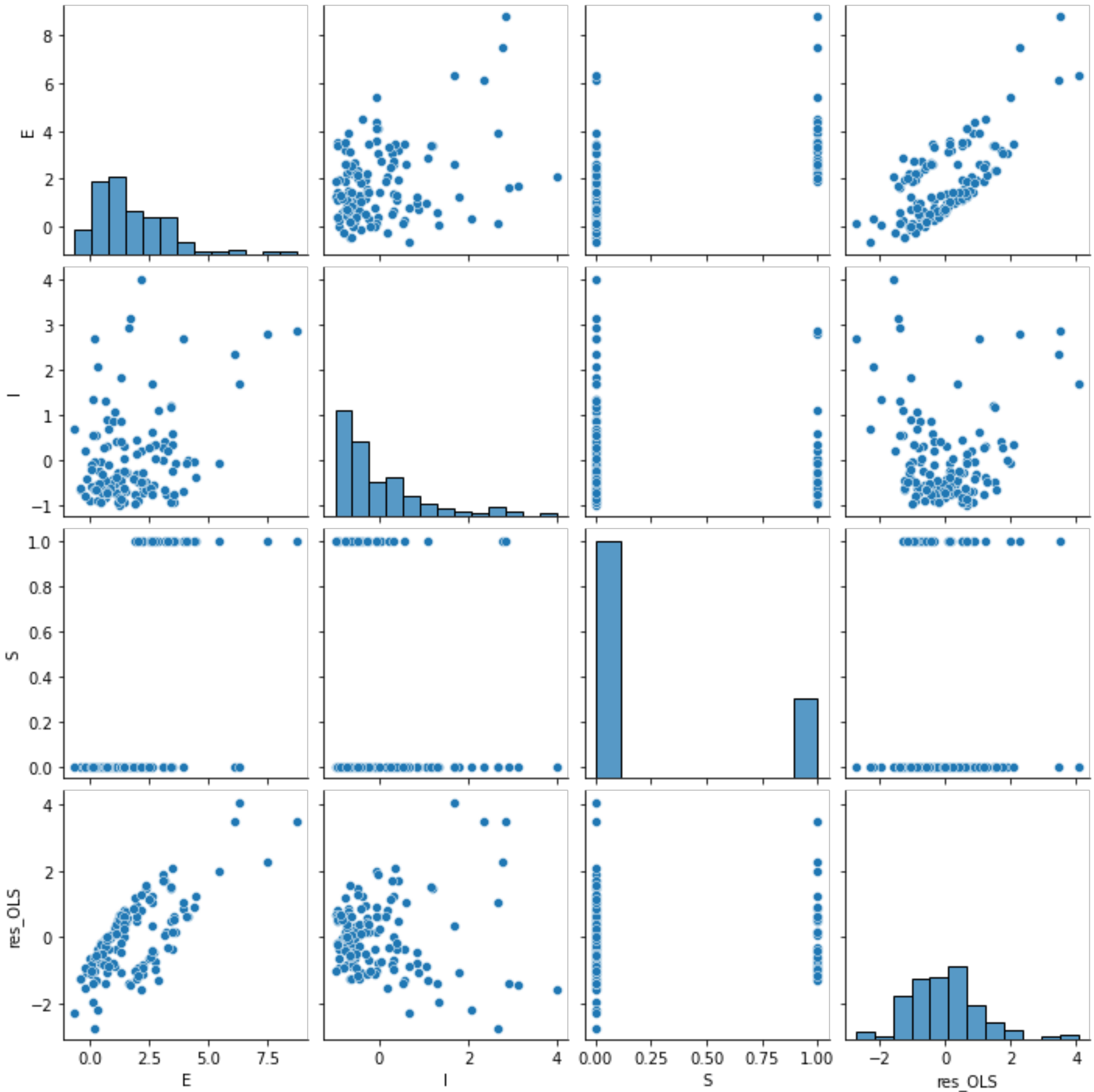
R^2 results are not surprising, because one of the variable is binary (S). Indeed, the OLS is not the best estimator for this kind of model.

In [16]:
```
df_res_OLS = df_data.copy(deep = True)
```

In [17]:
```
df_res_OLS['res_OLS'] = res_OLS
```

5.4 Residual vs explanatory variable X

In [18]:
```
sns.pairplot(df_res_OLS)
plt.show()
```



From this plot, we suppose there is an heteroskedasticity. Indeed, if we look at the plot of I vs residuals, we cannot observe a clear pattern but a clusterization at small value of I.

Similar observations can be done with respect of E, in which the is a more clear structure which suggests presence of heteroskedasticity.

To test this hypothesis, we perform a White test:

In [19]:
```
X = np.c_[np.ones(X.shape[0]), X]
white_test = het_white(res_OLS, X)
```

In [20]:
```
print('p-value: ', white_test[1], '\nIf P-value < 0.05, there is no Homoskedasticity')
```

```
p-value:  4.837759421665738e-09
If P-value < 0.05, there is no Homoskedasticity
```

This p-value confirms the hypothesis of heteroskedasticity.

5.5 Bootstrap estimates of the model coefficients

Paired Bootstrap

Assumptions:

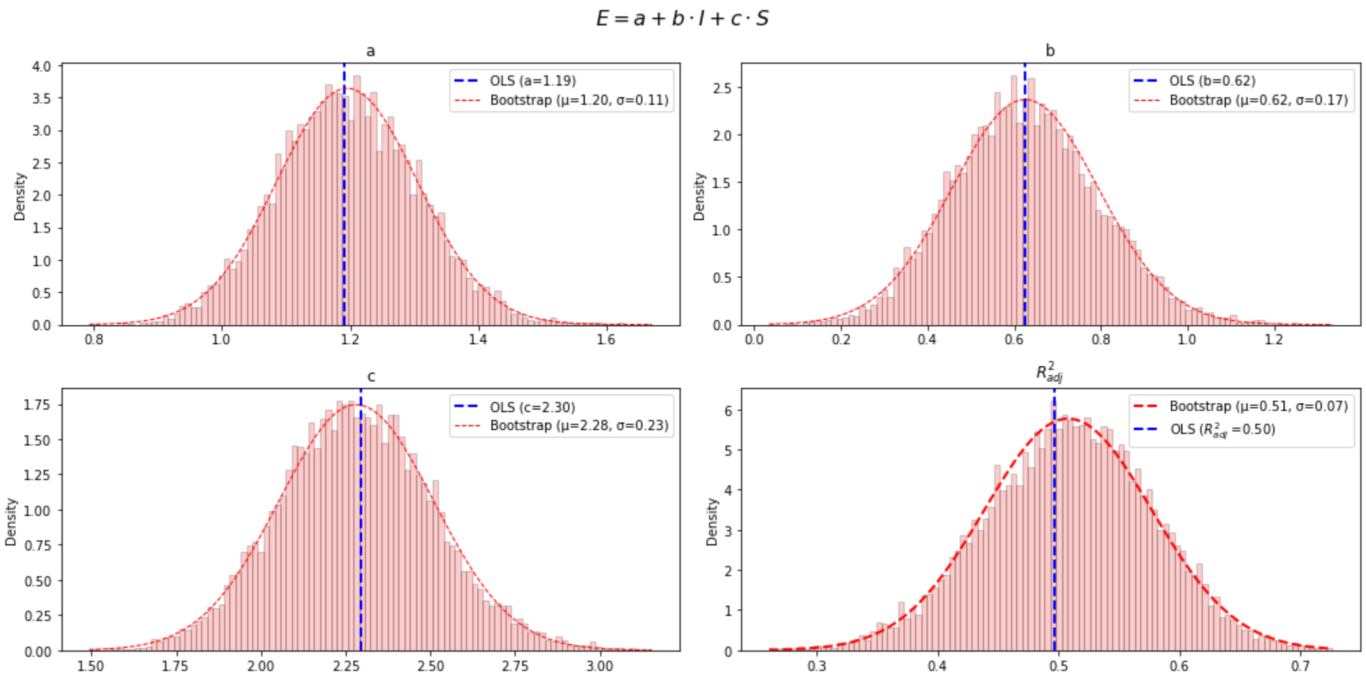
- assumes that there exists a joint probability distribution for the explanatory variables;
- makes no assumptions about the properties of the error terms ε_i.

In this method a sample of n rows $\{(E_i^*, I_i^*, S_i^*)\}$ is randomly extracted from the original dataset.

In [21]:
```
def paired_bootstrap(df = df_data, B = B_samples, K_vals = 3, seed = 13):
    """
    Parameters:
            df : DataFrame containing the representative data
            B : number of bootstrap samples
            K_vals : number of predictors
            seed : seed for the random number generator
    Return: 2 arrays containing all beta and R^2_adj Bootstrap values
    """
    # Fixing the seed
    np.random.seed(seed)

    # Fixing sample size
    sample_size = df.shape[0]

    # Initializing arrays to store the results
    beta_all = np.zeros((B, K_vals))
    R2_adj_all = np.zeros(B)

    # Bootstrapping
    for i in range(B):
        df_BS = df.sample(n=sample_size, replace=True)
        df_BS.reset_index(drop=True, inplace=True)

        # Calculating the OLS and R^2_adj
        beta, _, _, R_2_adj = OLS(np.c_[df_BS['I'], df_BS['S']], df_BS['E'], const=True)

        # Storing the results
        beta_all[i] = beta
        R2_adj_all[i] = R_2_adj

    return np.array(beta_all), np.array(R2_adj_all)
```

In [22]:
```
beta_all_PB, R_2_adj_all_PB  = paired_bootstrap(df = df_data)
```

In [23]:
```
beta_means_PB, beta_stds_PB, R_2_adj_means_PB, R_2_adj_stds_PB = plots(beta_all_PB, beta_OLS, R_2_adj_all_
```



Residual Bootstrap

Assumptions:

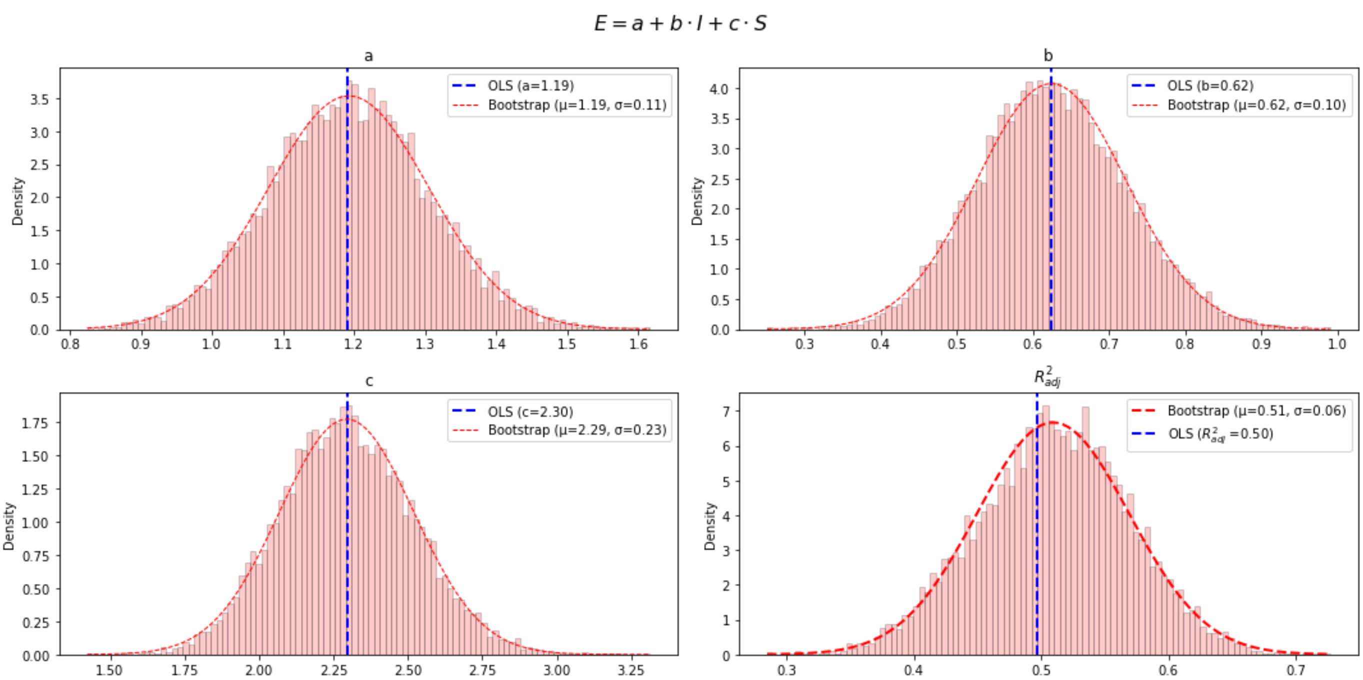
- $\mathbb{E}\left[E_i \mid I_i, S_i\right] = a + bI_i + cS_i$;
- the error terms ε_i are IID and homoskedastic.

In this case the residuals obtained from the OLS is used as a pool to extract an n sample of residuals $\left\{\hat{\varepsilon}_i^*\right\}$.

In [24]:
```
def residual_bootstrap(df = df_data, B = B_samples, K_vars = 3, seed = 13):
    """
    Parameters:
            df : DataFrame containing the reppresentative data
            B : number of bootstrap samples
            K_vars : number of predictors
            seed : seed for the random number generator
    Return: 2 arrays containing all beta and R^2_adj Bootstrap values
    """

    # Fixing the seed
    np.random.seed(seed)

    # Fixing sample size
    sample_size = df.shape[0]

    # Initializing arrays to store the results
    beta_all = np.zeros((B, K_vars))
    R2_adj_all = np.zeros(B)

    _, array_res, E_hat, _ = OLS(np.c_[df['I'], df['S']], df['E'], const=True)

    # Bootstrapping
    for i in range(B):
        e_star = np.random.choice(array_res, size = sample_size, replace = True)

        # Define the new Ei_star BS variables E_i^star = E_i + eps_i^star
        E_star = E_hat + e_star

        # Calculating the OLS and R^2_adj
        beta, _, _, R_2_adj = OLS(np.c_[df['I'], df['S']], E_star, const=True)

        # Storing the results
        beta_all[i] = beta
        R2_adj_all[i] = R_2_adj

    return np.array(beta_all), np.array(R2_adj_all)
```

In [25]:
```
beta_all_RB, R_2_adj_all_RB = residual_bootstrap(df=df_data)
```

In [26]:
```
beta_means_RB, beta_stds_RB, R_2_adj_means_RB, R_2_adj_stds_RB = plots(beta_all_RB, beta_OLS, R_2_adj_all_
```



Wild Bootstrap

Assumptions:

- $\mathbb{E}\left[E_i \mid I_i, S_i\right] = a + bI_i + cS_i$.

Note that it allows the presence of heteroskedasticity.

In this last bootstrap, we start by creating a sample of $\{\hat{\varepsilon}_i^*\}$ of size n where $\{\hat{\varepsilon}_i^*\} = f(\hat{\varepsilon}_i)v_i$, where

$$f(\hat{\varepsilon}_i) = \sqrt{\frac{n}{n-K}}\hat{\varepsilon}_i$$

and

$$v_i = \begin{cases} +1 & \text{, w.p. } \frac{1}{2} \\ -1 & \text{, w.p. } \frac{1}{2} \end{cases}$$

In [27]:

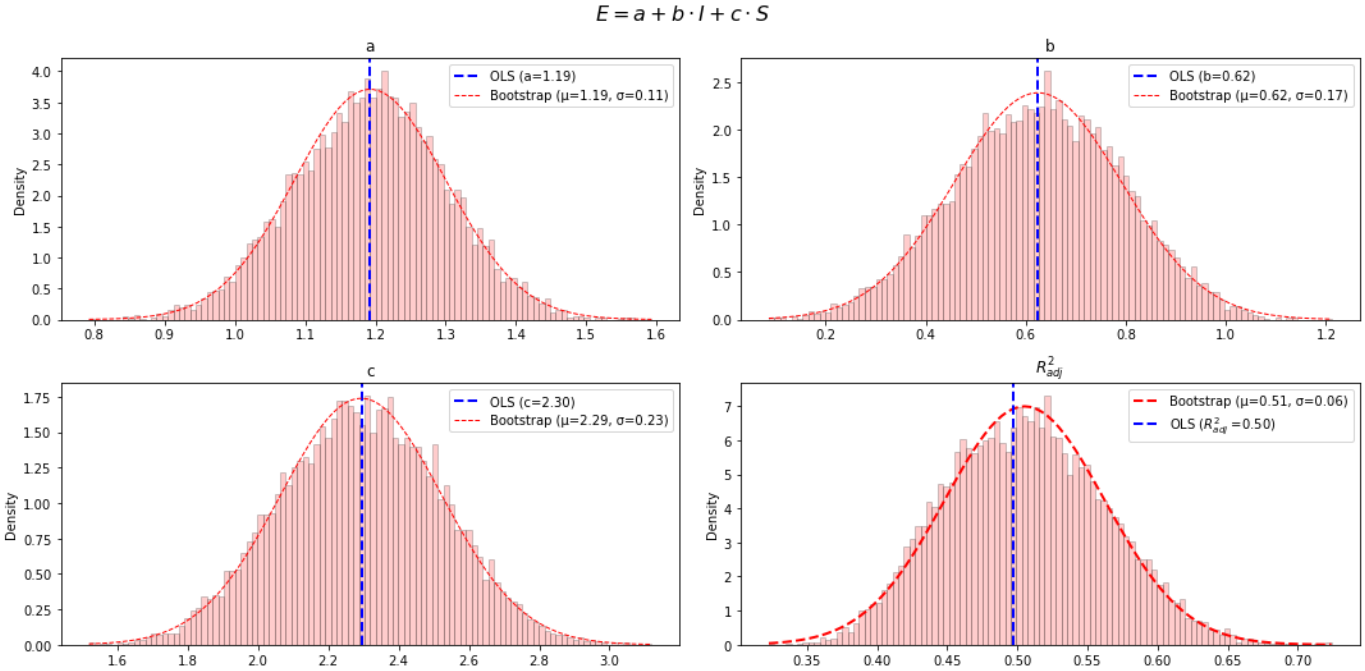
```
def wild_bootstrap(df = df_data , B=B_samples, K_vars=3, seed=13):
    """
    Parameters:
            df_data : DataFrame containing the representative data
            B : number of bootstrap samples
            K_vars : number of predictors
            seed : seed for the random number generator
    Return: beta_all, R2_adj_all
    """
    # Fixing the seed
    np.random.seed(seed)

    # Fixing sample size
    sample_size = df_data.shape[0]

    # Initializing arrays to store the results
    beta_all = np.zeros((B, K_vars))
    R2_adj_all = np.zeros(B)

    _, array_res, E_pred, _ = OLS(np.c_[df['I'], df['S']], df['E'], const=True)

    # Computes f(eps) = sqrt(n/(n+k)) * eps
    f_eps = np.sqrt(sample_size / (sample_size - K_vars)) * array_res

    # Bootstrapping
    for i in range(B):
        # Computes f(eps) * v_i
        v_i = np.random.choice([-1, 1], size=sample_size)
        eps_star = f_eps * v_i  # New residuals

        # Calculating eps_star and addint it to the empirical E
        E_star = E_pred + eps_star
        # OLS
        beta,_ ,_ , R_2_adj = OLS(np.c_[df['I'], df['S']], E_star, const=True)

        # Storing the results
        beta_all[i] = beta
        R2_adj_all[i] = R_2_adj

    return np.array(beta_all), np.array(R2_adj_all)
```

In [28]:

```
beta_all_WB, R_2_adj_all_WB = wild_bootstrap(df=df_data, B=B_samples)
```

In [29]:

```
beta_means_WB, beta_stds_WB, R_2_adj_means_WB, R_2_adj_stds_WB = plots(beta_all_WB, beta_OLS, R_2_adj_all_
```

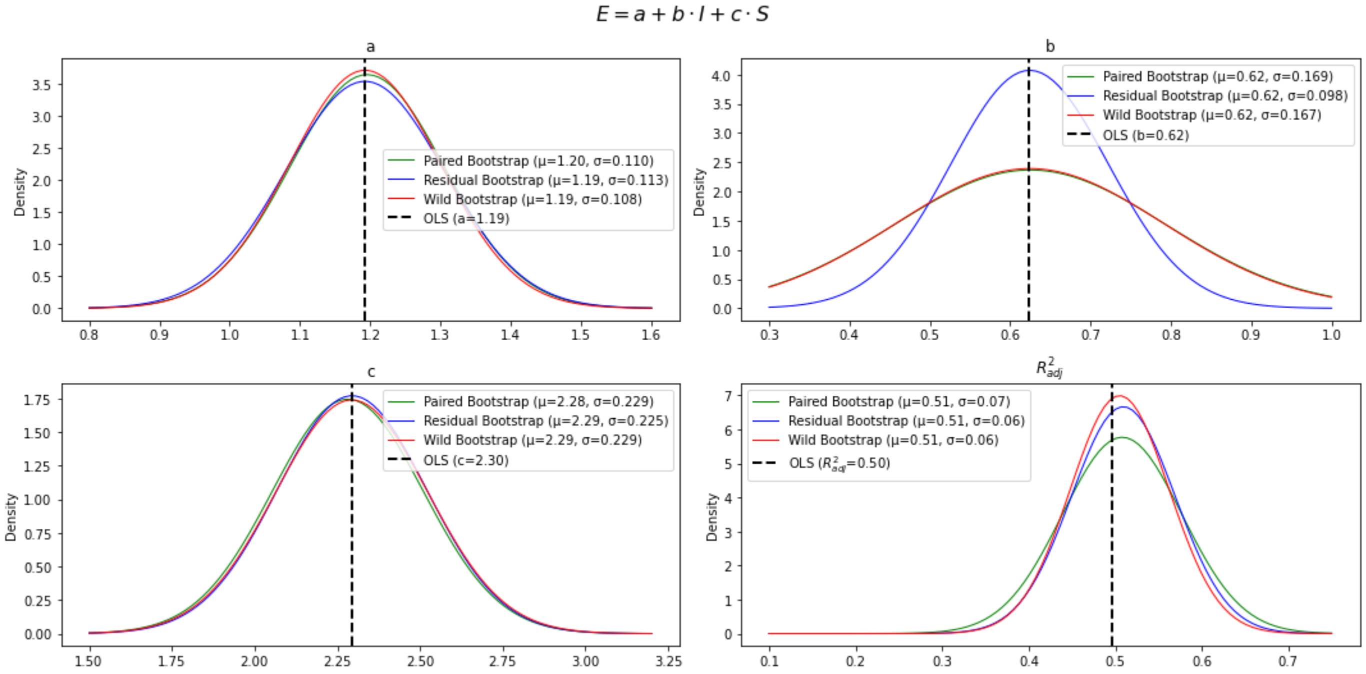


5.6 Parameters comparison

To better visualize the shape of the distributions, we'll use the normal fit for comparison without plotting the histograms.

In [30]:

```
fig, axes = plt.subplots(2,2, figsize = (15, 7.5))

axes = axes.ravel()

bins = 25

xb0 = np.linspace(0.8, 1.6, 100)
xb1 = np.linspace(0.3, 1, 100)
xb2 = np.linspace(1.5, 3.2, 100)

x = np.column_stack([xb0, xb1, xb2])

xR_2 = np.linspace(0.1, 0.75, 100)

beta_names = ['a', 'b', 'c']

for i in range(4):

    if i < 3:

        axes[i].plot(x[:,i], norm.pdf(x[:,i], beta_means_PB[i], beta_stds_PB[i]), color = 'green', linewic
        axes[i].plot(x[:,i], norm.pdf(x[:,i], beta_means_RB[i], beta_stds_RB[i]), color = 'blue', linewidt
        axes[i].plot(x[:,i], norm.pdf(x[:,i], beta_means_WB[i], beta_stds_WB[i]), color = 'red', linewidth

        axes[i].set_title( beta_names[i])
        axes[i].set_ylabel('Density')

        axes[i].axvline(beta_OLS[i], color='black', linestyle='dashed', linewidth=2, label = 'OLS ('+ beta
        axes[i].legend()

    else:
        axes[i].plot(xR_2, norm.pdf(xR_2, R_2_adj_means_PB[0], R_2_adj_stds_PB[0]), color = 'green', linew
        axes[i].plot(xR_2, norm.pdf(xR_2, R_2_adj_means_RB[0], R_2_adj_stds_RB[0]), color = 'blue', linewi
        axes[i].plot(xR_2, norm.pdf(xR_2, R_2_adj_means_WB[0], R_2_adj_stds_WB[0]), color = 'red', linewic

        axes[i].set_title(r'$R^{2}_{adj}$')
        axes[i].set_ylabel('Density')

        axes[i].axvline(R_2_adj_ols, color='black', linestyle='dashed', linewidth=2, label = 'OLS ('+r'$R^
        axes[i].legend()

fig.suptitle(r"$E = a + b \cdot I + c \cdot S$", fontsize=16)
plt.tight_layout()
plt.show()
```



Knowing the assumptions of the three methods, as we expected, the results obtained from Wild and Paired Bootstrap are similar.

While the discrepancy for the parameter b in the Residual Bootstrap is due to violation of the homoskedastic assumption. Moreover it is only observed in this parameter because the other one (S) is binary.

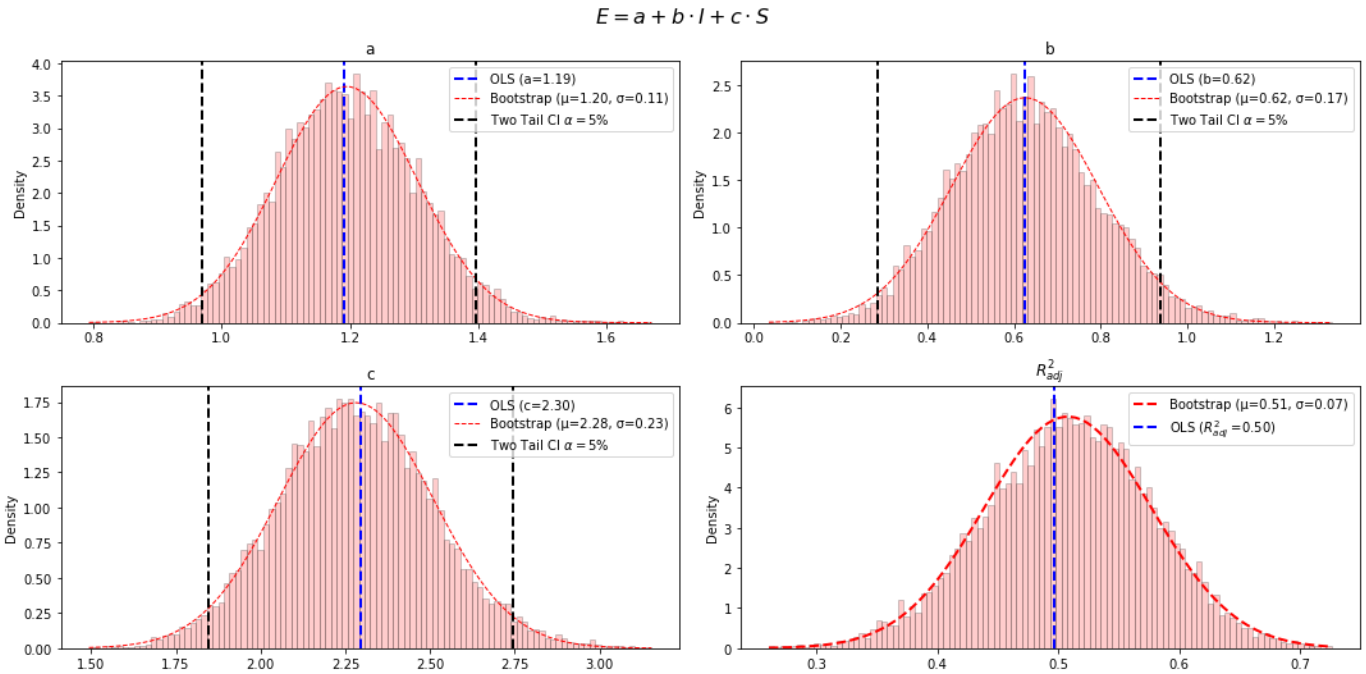
Regarding the R^2, all of them are close to the OLS one. However the Residual R^2 is not to be considered because the violated assumption.

5.7 Confidence intervals

Percentile Method

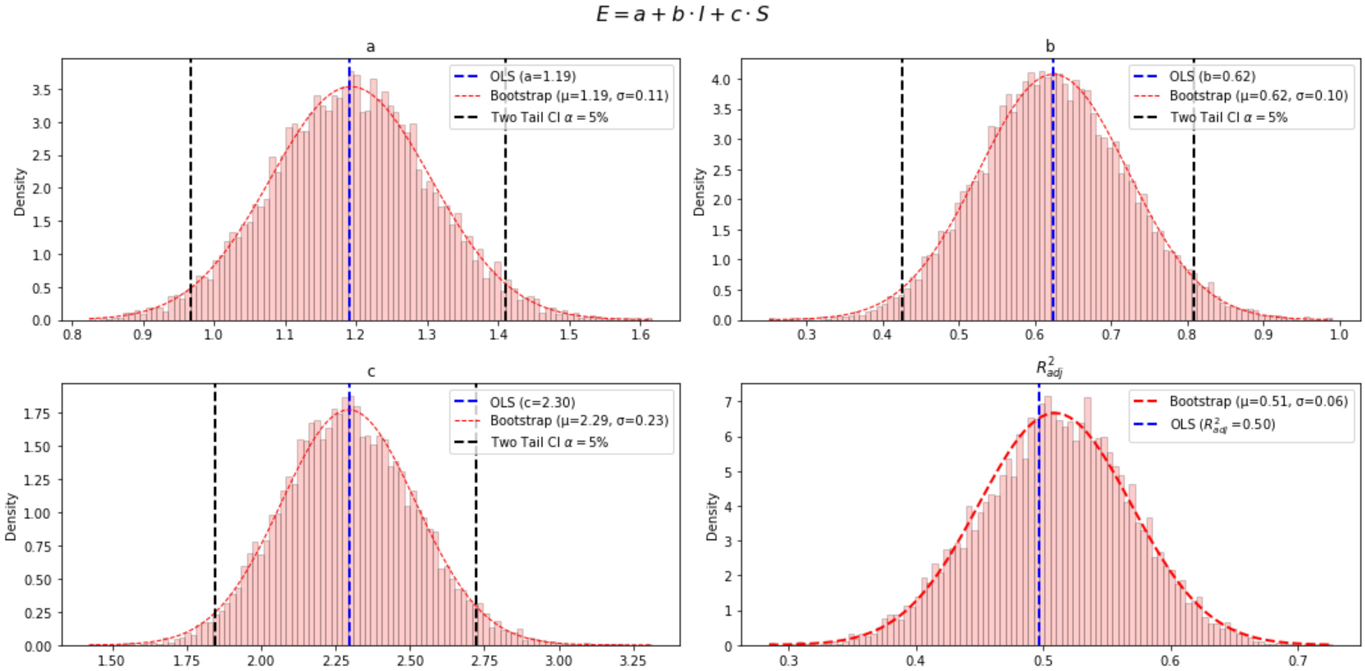
Paired bootstrap CI

In [31]:
```
_,_,_,_, CI_pb = plots(beta_all_PB, beta_OLS, R_2_adj_all_PB, R_2_adj_ols, bins = N_bins, CI = True)
```



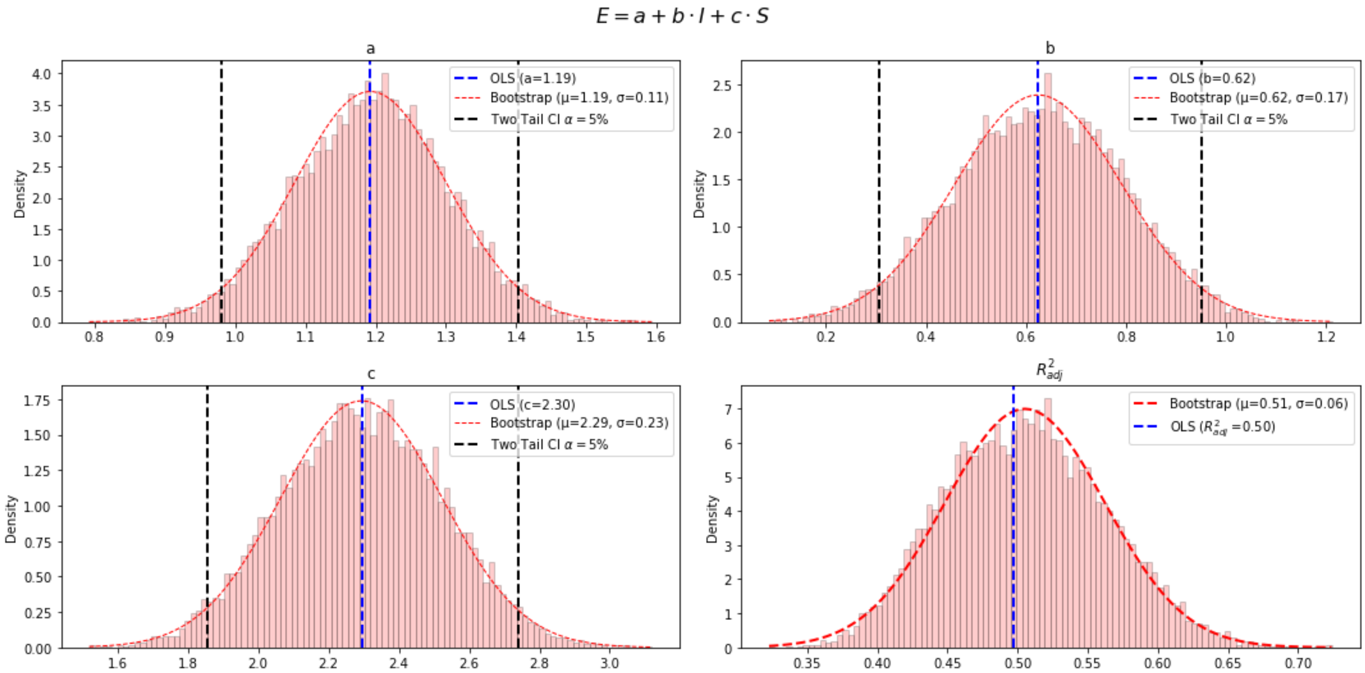
Residual bootstrap CI

In [32]:
```
_,_,_,_, CI_rb = plots(beta_all_RB, beta_OLS, R_2_adj_all_RB, R_2_adj_ols, bins = N_bins, CI = True)
```



Wild Bootstrap CI

In [33]:
```
_,_,_,_, CI_wb = plots(beta_all_WB, beta_OLS, R_2_adj_all_WB, R_2_adj_ols, bins = N_bins, CI = True)
```



In [34]:
```
print('CI width of Paired Bootstrap of a: ', np.round(CI_pb[0, 1] - CI_pb[0, 0], 4))
print('CI width of Paired Bootstrap of b: ', np.round(CI_pb[1, 1] - CI_pb[1, 0], 4))
print('CI width of Paired Bootstrap of c: ', np.round(CI_pb[2, 1] - CI_pb[2, 0], 4))
```

```
CI width of Paired Bootstrap of a:  0.4282
CI width of Paired Bootstrap of b:  0.6525
CI width of Paired Bootstrap of c:  0.9004
```

In [35]:
```
print('CI width of Residual Bootstrap of a: ', np.round(CI_rb[0, 1] - CI_rb[0, 0], 4))
print('CI width of Residual Bootstrap of b: ', np.round(CI_rb[1, 1] - CI_rb[1, 0], 4))
print('CI width of Residual Bootstrap of c: ', np.round(CI_rb[2, 1] - CI_rb[2, 0], 4))
```

```
CI width of Residual Bootstrap of a:  0.443
CI width of Residual Bootstrap of b:  0.3821
CI width of Residual Bootstrap of c:  0.8784
```

In [36]:
```
print('CI width of Wild Bootstrap of a: ', np.round(CI_wb[0, 1] - CI_wb[0, 0], 4))
print('CI width of Wild Bootstrap of b: ', np.round(CI_wb[1, 1] - CI_wb[1, 0], 4))
print('CI width of Wild Bootstrap of c: ', np.round(CI_wb[2, 1] - CI_wb[2, 0], 4))
```

```
CI width of Wild Bootstrap of a:  0.4223
CI width of Wild Bootstrap of b:  0.6453
CI width of Wild Bootstrap of c:  0.8861
```

Despite the Residual method would be the one to be preferred because its CI width, this method cannot be chosen due to the violated assumption mentioned before.

Between the remaining two, it is slightly preferable the Wild Bootstrap method due to the tightest confidence interval.