AE2 Assignment 1 Heijden_Steenstra_Kullberg FINAL

January 8, 2021

0.1 Programming Assignment - Bootstrap Methods - Adv. Econometrics 2

Deadline: Friday 17:00 hours, 8 January 2021

Nr	**Name**	**Student ID**	**Email**
1.	Dante van der Heijden	11020075	dantesean@gmail.com
2.	Wietse Steenstra	11004487	wietse161@live.nl
3.	Willem Kullberg	11041544	wkullberg@live.nl

Declaration of Originality

We whose names are given under 1., 2. and 3. above declare that:

1. These solutions are solely our own work.
2. We have not made (part of) these solutions available to any other student.
3. We shall not engage in any other activities that will dishonestly improve my results or dishonestly improve or hurt the results of others.

0.2 Instructions for completing and submitting the assignment

1. Submit your work in the form of (i) a Jupyter Notebook and (ii) PDF-file via Canvas assuming basic econometric knowledge, before the deadline. Your notebook should not give errors when executed with `Run All`.
2. Complete the table with the info of your group members. By submitting the Jupyter Notebook, you agree with the included declaration of originality. Do not copy work of others. This will be considered as fraud!
3. Clarify your code with comments.

0.3 Hints

- Only use the paired bootstrap
- Work with Numpy vectors or matrices as much as possible, e.g. `np.quantile(tB_OLS,[0.05,0.95],axis=0)` returns two quantiles for the whole vector of OLS estimates
- When coding, you can reduce the running time by setting `BOOTREP=99` and reduce the number of simulations. For the final execution, please return to original values!
- For a progress bar, please install `conda install -c conda-forge tqdm` or if you don't use anaconda use can just execute `pip install tqdm`

- If you want to use plotly, please install `conda install -c plotly plotly`
- Below, you can find Python code for generating the data and doing a simulation using multicores. To use multicores, you have to install `multiprocess`: `conda install -c conda-forge multiprocess`. Otherwise, execute `pip install multiprocess`
- The idea behind multiprocess is that each CPU core receives a sample, executes the resampling and returns the results. These results will be stored in one big list, which can be analyzed after the simulation.

0.4 Assignment

The purpose of this assignment is for you to gain practical experience with resampling methods. You will investigate several bootstrap confidence intervals for OLS and LASSO estimators. The DGP is given by:

- $X_i \sim N(0, \Sigma)$, $\Sigma = (\sigma_{ij}) \in \mathbb{R}^{p \times p}$ with $\sigma_{ij} = \rho^{|i-j|}$, $\beta_j = 0$ for $1 \leq j \leq p-15$, $\beta_j = 0.5$ for $p-14 \leq j \leq p-10$, $\beta_j = 1.5$ for $p-9 \leq j \leq p-5$ $\beta_j = 2.5$ for $p-4 \leq j \leq p$.
- $\varepsilon_1, ..., \varepsilon_n \sim N(0,1)$
- $y = X\beta + \varepsilon$

Let $\hat{\beta} = (X'X)^{-1}X'y$ denote the OLS estimator, while $\breve{\beta}$ denote the LASSO estimator based on minimizing

$$\sum_{i=1}^{n}(y_i - b'X_i)^2 + \alpha \sum_{j=1}^{p}|b_j|.$$

Only consider `lasso = linear_model.Lasso(alpha=0.02)`, so keep the amount of regularization fixed!

Please, briefly answer all questions below using graphs and if necessary tables.

0.5 Import packages

[1]:
```
import numpy as np
from sklearn import linear_model
# import plotly.express as px        # uncomment if you want to use plotly.
 ↪express
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
import pandas as pd
```

0.6 Generate Samples

[2]:
```
REP = 1000                          # numer of Monte Carlo simulations
BOOTREP = 999                       # number of bootstrap replications
n = 50
p = 25
rho = 0.6
mu = np.zeros(p)
Sigma = np.identity(p)
```

```
for i in range(p):
    for j in range(p):
        Sigma[i,j] = rho**abs(i - j)
beta = np.zeros(p)
beta[(p - 15):(p - 10)] = 0.5
beta[(p - 10):(p - 5)] = 1.5
beta[(p - 5):] = 2.5
arglist=[]
for r in tqdm(range(REP)):
    X = np.random.multivariate_normal(mean=mu, cov=Sigma, size=n)
    eps = np.random.normal(size=n)
    y = X@beta + eps
    arglist.append((r,BOOTREP,y,X))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0),␣
 ↪HTML(value='')))
```

0.7 Resampling Procedure

[3]:
```
def Bootstrap(args):
    (iter,BOOTREP,y,X)=args

    # Define OLS function
    def OLS(y,X):
        N,p = X.shape                    # number of observations and regressors
        XXi = np.linalg.inv(X.T @ X)
        b_ols = XXi @ (X.T @ y)
        res = y-X @ b_ols
        s2 = (res @ res)/(N-p)
        SE = np.sqrt(s2*np.diag(XXi))
        return b_ols,SE,res

    import numpy as np
    from sklearn import linear_model
    from scipy.stats import norm
    import pandas as pd

    n,p = X.shape
    # Estimates original sample
    lasso = linear_model.Lasso(alpha=0.02)
    lasso.fit(X, y)
    b_LASSO=np.copy(lasso.coef_)
    b_OLS,b_OLS_SE,res = OLS(y,X)
    # initilize bootstrap arrays
    bB_LASSO = np.zeros((BOOTREP,p))
    bB_OLS   = np.zeros((BOOTREP,p))
```

```
    bB_OLS_SE = np.zeros((BOOTREP,p))
    tB = np.zeros((BOOTREP,p))

    np.random.seed(1)
    # balanced bootstap
    index_B=np.random.permutation(np.repeat(np.arange(n),BOOTREP)).
↪reshape((BOOTREP,n))
    for b in range(BOOTREP):
        index = index_B[b,:]  # select the indices
        yB = np.copy(y[index])
        XB = np.copy(X[index,:])
        lasso.fit(XB, yB)
        bB_LASSO[b,:] = np.copy(lasso.coef_)
        bB_OLS[b,:], bB_OLS_SE[b,:], bres = OLS(yB, XB)
        tB[b, :] = (bB_OLS[b]-b_OLS)/bB_OLS_SE[b]

    se_boot_OLS = np.std(bB_OLS, axis=0)
    se_boot_LASSO = np.std(bB_LASSO, axis=0)

    # percentile
    q_bB_LASSO = np.quantile(bB_LASSO,[0.05,0.95],axis=0)
    q_bB_OLS   = np.quantile(bB_OLS,[0.05,0.95],axis=0)
    q_tB_OLS   = np.quantile(tB,[0.05,0.95],axis=0)

    ## Calc median bias
    perc_of_bootreps_smaller_than_b_LASSO = pd.DataFrame(bB_LASSO).lt(b_LASSO).
↪sum()/BOOTREP
    z_0 = norm.ppf(perc_of_bootreps_smaller_than_b_LASSO)

    ## Calc acceleration coefficient

    # Jackknife
    jackknife_values=np.zeros((n,p))
    b_lasso_jack = np.zeros((n,p))
    for i in range(n):
        X_jack=np.delete(X,i,axis=0)
        y_jack = np.delete(y,i)
        lasso.fit(X_jack,y_jack)
        b_lasso_jack[i,:] = np.copy(lasso.coef_)
    avg_b_lasso_jack = np.mean(b_lasso_jack,axis=0)
    dev_b_lasso_jack = avg_b_lasso_jack - b_lasso_jack

    # Acceleration coefficient
    np.seterr(divide='ignore', invalid='ignore')
    acc_coefficient = (dev_b_lasso_jack**3).sum(axis=0) /␣
↪(6*(dev_b_lasso_jack**2).sum(axis=0)**(1.5))
```

```
    # Set nans to zero. Zero is the 'default' acceleration value, so this␣
↪shouldn't affect the results too much.
    acc_coefficient[np.isnan(acc_coefficient)] = 0

    ## Calc upper and lower quantiles
    alpha_1 = norm.cdf(z_0 + (z_0+norm.ppf(0.05))/(1-acc_coefficient*(z_0+norm.
↪ppf(0.05))))
    alpha_2 = norm.cdf(z_0 + (z_0+norm.ppf(0.95))/(1-acc_coefficient*(z_0+norm.
↪ppf(0.95))))


    ## Calc confidence interval
    bca_CI_LASSO = np.zeros((p,2))
    for i in range(p):
        b_LASSO_part = bB_LASSO[:,i]
        bca_CI_LASSO[i,:] = np.quantile(b_LASSO_part,[alpha_1[i],alpha_2[i]])

    argout = [b_LASSO,b_OLS, b_OLS_SE, se_boot_LASSO, se_boot_OLS, q_bB_LASSO,␣
↪q_bB_OLS, q_tB_OLS, bca_CI_LASSO.T    # add more when necessary
              ]
    return(argout)
```

0.8 Execute the Simulation and get Results

[4]:
```
from multiprocess import Pool
pool4 = Pool(processes=4)
result_list = list(tqdm(pool4.imap_unordered(Bootstrap, arglist), total=REP))
pool4.close()
pool4.join()
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0),␣
 ↪HTML(value='')))
```

0.9 Perform the Post-Processing

[5]:
```
nr_methods=4                            # number of methods
                                        # 0 = percentile
                                        # 1 = SE_boot
                                        # 2 = percentile-t
                                        # 3 = BCa


# Initialize arrays
# Bootstrap regressors and SEs
b_LASSO    =np.zeros((REP,p))
b_OLS      =np.zeros((REP,p))
```

```
b_OLS_SE = np.zeros((REP,p))
se_boot_LASSO    =np.zeros((REP,p))
se_boot_OLS      =np.zeros((REP,p))

# Bootstrap upper and lower bounds for CIs
lb_boot_LASSO = np.zeros((REP,p))
ub_boot_LASSO = np.zeros((REP,p))
lb_boot_OLS = np.zeros((REP,p))
ub_boot_OLS = np.zeros((REP,p))
t_lb_boot_OLS = np.zeros((REP,p))
t_ub_boot_OLS = np.zeros((REP,p))

# BCA confidence interval
ub_bca_CI_lasso = np.zeros((REP,p))
lb_bca_CI_lasso = np.zeros((REP,p))

for r in tqdm(range(REP)):
    b_LASSO[r,:]  = result_list[r][0]
    b_OLS[r,:]    = result_list[r][1]
    b_OLS_SE[r,:] = result_list[r][2]
    se_boot_LASSO[r,:] = result_list[r][3]
    se_boot_OLS[r,:] = result_list[r][4]
    lb_boot_LASSO[r,:] = result_list[r][5][0]
    ub_boot_LASSO[r,:] = result_list[r][5][1]
    lb_boot_OLS[r,:] = result_list[r][6][0]
    ub_boot_OLS[r,:] = result_list[r][6][1]
    t_lb_boot_OLS[r,:] = result_list[r][7][0]
    t_ub_boot_OLS[r,:] = result_list[r][7][1]
    lb_bca_CI_lasso[r,:] = result_list[r][8][0]
    ub_bca_CI_lasso[r,:] = result_list[r][8][1]
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1000.0),␣
 ↪HTML(value='')))
```

0.10 Carry Out Analysis

1. Choose $n = 50$, $p = 25$ and $\rho = 0.6$. Determine the bias and RMSE of the OLS and LASSO estimators using 1,000 Monte Carlo replications.

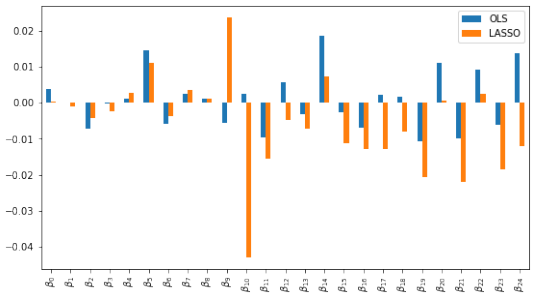
[6]:
```
##### 1.
# Calculate bias for each regressor
MC_mean_OLS = np.mean(b_OLS, axis = 0)
MC_mean_LASSO = np.mean(b_LASSO, axis = 0)
MC_bias_OLS = MC_mean_OLS - beta
MC_bias_LASSO = MC_mean_LASSO - beta
```

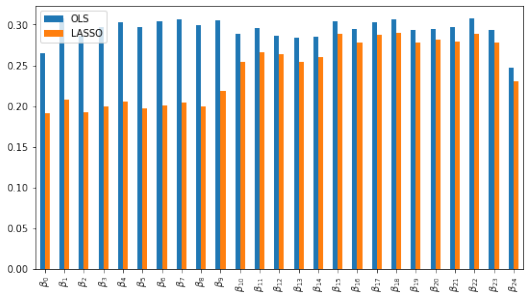
```
# Calculate SE for each regressor
MC_SE_OLS = np.std(b_OLS, axis = 0)
MC_SE_LASSO = np.std(b_LASSO, axis = 0)

# Mean absolute bias
avg_bias_OLS = np.mean(abs(MC_bias_OLS))
avg_bias_LASSO = np.mean(abs(MC_bias_LASSO))

# Mean SE
avg_SE_OLS = np.mean(MC_SE_OLS)
avg_SE_LASSO = np.mean(MC_SE_LASSO)

RMSE_OLS = np.sqrt(np.mean((b_OLS - beta)**2, axis=0))
RMSE_LASSO = np.sqrt(np.mean((b_LASSO - beta)**2, axis=0))

print("Avg OLS bias = {0}, Avg Lasso bias = {1}".
 ↪format(avg_bias_OLS,avg_bias_LASSO))
print("Avg OLS SE = {0}, Avg Lasso SE = {1}".format(avg_SE_OLS,avg_SE_LASSO))
print("Avg OLS RMSE = {0}, Avg Lasso RMSE = {1}".format(np.mean(RMSE_OLS), np.
 ↪mean(RMSE_LASSO)))

# Create DFs for plots
q1_bias = pd.DataFrame({"OLS": MC_bias_OLS, "LASSO": MC_bias_LASSO})
q1_RMSE = pd.DataFrame({"OLS": RMSE_OLS, "LASSO": RMSE_LASSO})
indices = []
for i in range(len(q1_bias.index)):
    indices.append("$ _{" + str(i) + "}$")

q1_bias.index = indices
q1_RMSE.index = indices

# Plot figure
plt.figure(dpi=1200)

fig1, axes1 = plt.subplots(1, 2, figsize=(20,5))
q1_bias.plot.bar(ax=axes1[0])
q1_RMSE.plot.bar(ax=axes1[1])
```

```
Avg OLS bias = 0.006256982844012327, Avg Lasso bias = 0.010104870715248881
Avg OLS SE = 0.2938802575146052, Avg Lasso SE = 0.2433967977682985
Avg OLS RMSE = 0.2939887218161822, Avg Lasso RMSE = 0.2437797966505447
```

[6]: <AxesSubplot:>

<Figure size 7200x4800 with 0 Axes>





Answer: We can see that the average bias is very low, thus the RMSE is mostly caused by the variance. As expected, the OLS estimator has a higher RMSE, especially for the regressors that are equal to zero.

2. Estimate by simulation the coverage probabilities (cov. prob.) of the 90% first-order asymptotic two-sided confidence intervals for the OLS estimator, i.e. the fraction of confidence intervals (CI)

$$[\hat{\beta}_j - 1.645SE(\hat{\beta}_j), \hat{\beta}_j + 1.645SE(\hat{\beta}_j)]$$

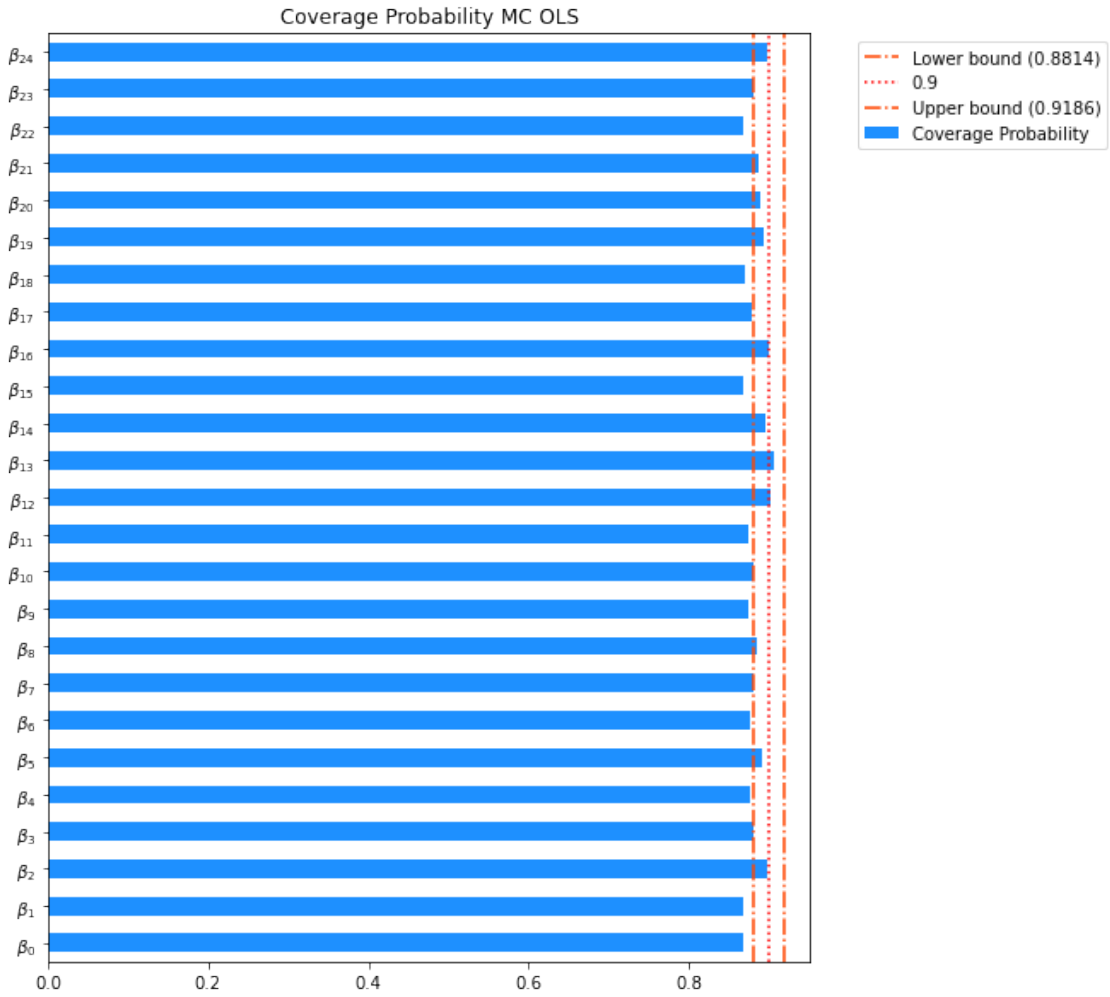
that contains the true parameter β_j. Here $SE(\hat{\beta}_j)$ is the usual (non-robust) standard error based on $s^2(X'X)^{-1}$.

[7]:
```
##### 2.

# Calculate bounds
mc_ols_lb = pd.DataFrame(b_OLS) - 1.645*b_OLS_SE
mc_ols_ub = pd.DataFrame(b_OLS) + 1.645*b_OLS_SE

# Compute betas within bounds and compute average
mc_ols_CI = mc_ols_lb.le(pd.Series(beta).T, axis=1) & mc_ols_ub.ge(pd.
 ↪Series(beta).T, axis=1)
mc_ols_cov_avg_prob = np.mean(mc_ols_CI.sum()/REP)

print("Average coverage probability = {}".format(mc_ols_cov_avg_prob))

# Plot
q2_CI_conv = pd.DataFrame({"Coverage Probability": (mc_ols_CI.sum()/REP)})
q2_CI_conv.index = indices
fig2, axes2 = plt.subplots(1, 1, figsize=(8,10))
(q2_CI_conv).plot.barh(ax=axes2, label="Coverage Probability",␣
 ↪color="dodgerblue")
axes2.title.set_text("Coverage Probability MC OLS")
axes2.axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
 ↪8814)')
axes2.axvline(x=0.9, color='red', ls=':', label='0.9')
axes2.axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
 ↪9186)')
```

```
axes2.legend(bbox_to_anchor=(1.05,1))
```

```
Average coverage probability = 0.8837600000000001
```

[7]: <matplotlib.legend.Legend at 0x1454a704850>



Answer: Most coverage probabilities are close to, but slightly lower than 0.9. So in less than 90% of the cases, the CI contains the true β.

3. Estimate by simulation the cov. prob. of the 90% first-order asymptotic two-sided CI for OLS and LASSO using

$$SE_{boot}(\tilde{\beta}),$$

for

$$\tilde{\beta} \in \{\hat{\beta}, \check{\beta}\}.$$

[8]:
```
##### 3.
```

```
# Calculate bootstrap bounds for OLS and Lasso
boot_ols_lb = pd.DataFrame(b_OLS) - 1.645*se_boot_OLS
boot_ols_ub = pd.DataFrame(b_OLS) + 1.645*se_boot_OLS
boot_lasso_lb = pd.DataFrame(b_LASSO) - 1.645*se_boot_LASSO
boot_lasso_ub = pd.DataFrame(b_LASSO) + 1.645*se_boot_LASSO

# Compute betas within bounds and compute average
boot_ols_CI = boot_ols_lb.le(pd.Series(beta).T, axis=1) & boot_ols_ub.ge(pd.
 ↪Series(beta).T, axis=1)
boot_ols_cov_avg_prob = np.mean(boot_ols_CI.sum()/REP)
boot_lasso_CI = boot_lasso_lb.le(pd.Series(beta).T, axis=1) & boot_lasso_ub.
 ↪ge(pd.Series(beta).T, axis=1)
boot_lasso_cov_avg_prob = np.mean(boot_lasso_CI.sum()/REP)

print("Average coverage probability OLS = {}".format(boot_ols_cov_avg_prob))
print("Average coverage probability LASSO = {}".format(boot_lasso_cov_avg_prob))

# Plots
q3_CI_conv_OLS = pd.DataFrame({"Coverage Probability": (boot_ols_CI.sum()/REP)})
q3_CI_conv_LASSO = pd.DataFrame({"Coverage Probability": (boot_lasso_CI.sum()/
 ↪REP)})
q3_CI_conv_OLS.index = indices
q3_CI_conv_LASSO.index = indices
fig3, axes3 = plt.subplots(1, 2, figsize=(12,10))
(q3_CI_conv_OLS).plot.barh(ax=axes3[0], label="Coverage Probability",␣
 ↪color="dodgerblue")
axes3[0].title.set_text("Coverage Probability OLS")
axes3[0].axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
 ↪8814)')
axes3[0].axvline(x=0.9, color='red', ls=':', label='0.9')
axes3[0].axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
 ↪9186)')
axes3[0].get_legend().remove()
(q3_CI_conv_LASSO).plot.barh(ax=axes3[1], label="Coverage Probability",␣
 ↪color="dodgerblue")
axes3[1].title.set_text("Coverage Probability LASSO")
axes3[1].axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
 ↪8814)')
axes3[1].axvline(x=0.9, color='red', ls=':', label='0.9')
axes3[1].axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
 ↪9186)')
axes3[1].legend(bbox_to_anchor=(1.05,1))
```

```
Average coverage probability OLS = 0.9988399999999998
Average coverage probability LASSO = 0.9538800000000001
```

[8]: <matplotlib.legend.Legend at 0x1454ace11f0>

Answer: Both for OLS and LASSO with bootstrapped SE, the coverage probability is significantly higher than 0.9 and for OLS even close to 1, which indicates strong underrejection. It is remarkable that in the case of LASSO, the coverage probabilities for the betas for which $\beta_i = 0.5$ are closer to 0.9 than the others.

4. Note that the cov. prob. can be interpreted as the number of successes in 1,000 trials. Hence, the estimated cov. prob. is not significantly different (at the 95% confidence level) from 90% if its value is contained in the interval

$$0.90 \pm 1.96\sqrt{(0.90 \times 0.10/1000)} = [0.8814, 0.9186].$$

Check if the estimated cov. prob. of questions 2 & 3 are significantly different from 90%. What is your conclusion?

Answer: As can be seen in the figures above, most coverage probabilities for question 2 are within the interval of $[0.8814, 0.9186]$, thus they are not significantly different from 90%. This is also displayed seen in the table below.

For question 3, with OLS all coverage probabilities are significantly higher than 0.9 and even approach 1. This means that the true beta is almost always contained in the confidence interval and that the confidence interval itself should be more narrow. If our test is specified to have a certain alpha (type I error) we want to maximize the power for this given alpha and not have a lower size in practice, which also leads to lower power. When the amount of observations is

increased, this problem disappears and we see that the coverage probabilities actually go towards 0.9.

For most parameters of the bootstrapped LASSO coverage probabilities, we also have coverage probabilities outside the given interval. This indicates the same problem as with OLS, but to a slightly lesser degree. The difference between OLS and LASSO could be explained to the degree of overfitting, since our dataset is quite small compared to the number of regressors. As LASSO includes some regulariziation and penalizes the parameters more, it will overfit less than OLS and produces a smaller error in rejection probability.

[9]:
```
table_1 = pd.DataFrame({"Coverage Probability Q2": (mc_ols_CI.sum()/REP),␣
 ↪"Coverage Probability Q3 OLS": (boot_ols_CI.sum()/REP), "Coverage␣
 ↪Probability Q3 LASSO": (boot_lasso_CI.sum()/REP)})
table_1.index = indices
display(table_1)
```

	Coverage Probability Q2	Coverage Probability Q3 OLS	\
$ _{0}$	0.869	0.998	
$ _{1}$	0.869	0.998	
$ _{2}$	0.897	0.997	
$ _{3}$	0.882	1.000	
$ _{4}$	0.876	0.995	
$ _{5}$	0.891	1.000	
$ _{6}$	0.877	1.000	
$ _{7}$	0.881	0.998	
$ _{8}$	0.885	0.999	
$ _{9}$	0.874	0.998	
$ _{10}$	0.880	0.999	
$ _{11}$	0.874	0.999	
$ _{12}$	0.902	0.999	
$ _{13}$	0.907	1.000	
$ _{14}$	0.895	0.999	
$ _{15}$	0.868	1.000	
$ _{16}$	0.900	0.999	
$ _{17}$	0.879	1.000	
$ _{18}$	0.871	0.999	
$ _{19}$	0.893	0.999	
$ _{20}$	0.889	0.999	
$ _{21}$	0.887	0.998	
$ _{22}$	0.869	1.000	
$ _{23}$	0.882	0.999	
$ _{24}$	0.897	0.999	

	Coverage Probability Q3 LASSO
$ _{0}$	0.956
$ _{1}$	0.963
$ _{2}$	0.974
$ _{3}$	0.960

```
$ _{4}$                         0.963
$ _{5}$                         0.966
$ _{6}$                         0.966
$ _{7}$                         0.959
$ _{8}$                         0.956
$ _{9}$                         0.945
$ _{10}$                        0.899
$ _{11}$                        0.916
$ _{12}$                        0.919
$ _{13}$                        0.931
$ _{14}$                        0.937
$ _{15}$                        0.959
$ _{16}$                        0.966
$ _{17}$                        0.951
$ _{18}$                        0.948
$ _{19}$                        0.973
$ _{20}$                        0.967
$ _{21}$                        0.964
$ _{22}$                        0.966
$ _{23}$                        0.971
$ _{24}$                        0.972
```

5. Estimate by simulation the cov. prob. for OLS and LASSO of 90% equal-tailed two-sided percentile bootstrap confidence intervals:

$$(\tilde{\beta}^*_{95\%}, \tilde{\beta}^*_{5\%}),$$

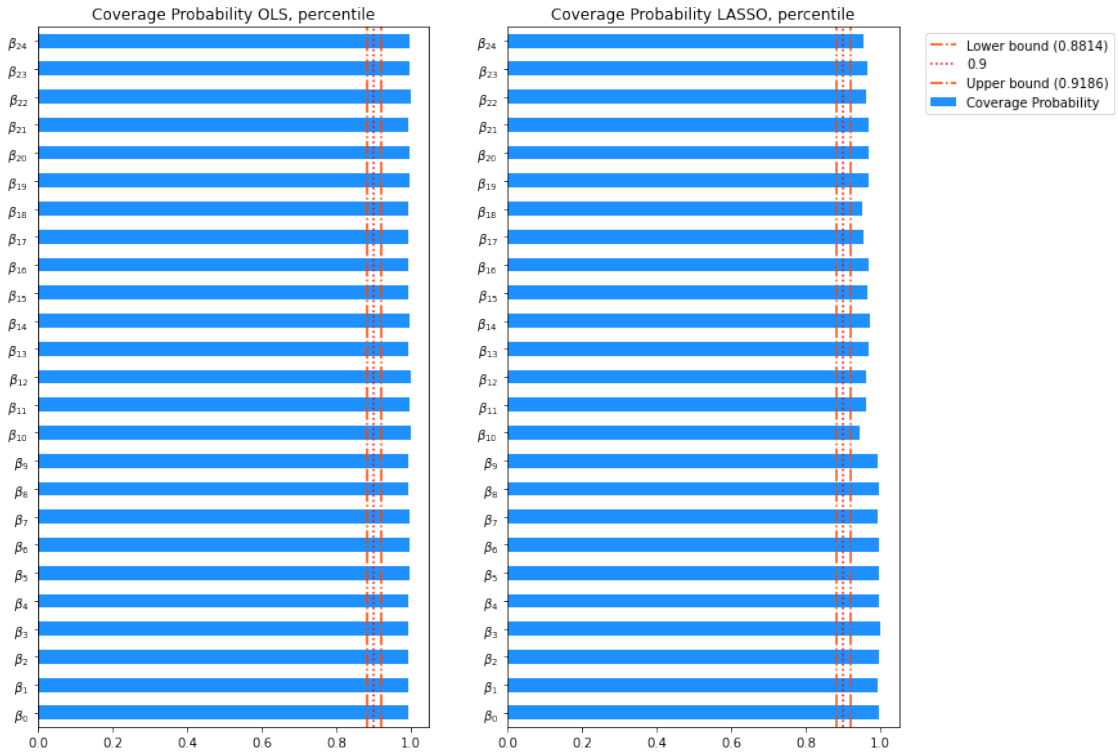
where $\mathbb{P}_*[\tilde{\beta}^* > \beta^*_{5\%}] = 5\%$ and $\tilde{\beta} \in \{\hat{\beta}, \breve{\beta}\}$.

[10]:
```
##### 5.

# Create confidence intervals based on bootstrap
boot_ols_CI = pd.DataFrame(lb_boot_OLS).le(pd.Series(beta).T, axis=1) & pd.
 ↪DataFrame(ub_boot_OLS).ge(pd.Series(beta).T, axis=1)
boot_lasso_CI = pd.DataFrame(lb_boot_LASSO).le(pd.Series(beta).T, axis=1) & pd.
 ↪DataFrame(ub_boot_LASSO).ge(pd.Series(beta).T, axis=1)

# Calculate coverage probabalities
boot_ols_cov_avg_prob = np.mean(boot_ols_CI.sum()/REP)
boot_lasso_cov_avg_prob = np.mean(boot_lasso_CI.sum()/REP)

print("Average coverage probability of OLS = {}".format(boot_ols_cov_avg_prob))
print("Average coverage probability LASSO = {}".format(boot_lasso_cov_avg_prob))

# Plots
q5_CI_conv_OLS = pd.DataFrame({"Coverage Probability": (boot_ols_CI.sum()/REP)})
q5_CI_conv_LASSO = pd.DataFrame({"Coverage Probability": (boot_lasso_CI.sum()/
 ↪REP)})
q5_CI_conv_OLS.index = indices
```

```
q5_CI_conv_LASSO.index = indices
fig4, axes4 = plt.subplots(1, 2, figsize=(12,10))
(q5_CI_conv_OLS).plot.barh(ax=axes4[0], label="Coverage Probability,␣
↪percentile", color="dodgerblue")
axes4[0].title.set_text("Coverage Probability OLS, percentile")
axes4[0].axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
↪8814)')
axes4[0].axvline(x=0.9, color='red', ls=':', label='0.9')
axes4[0].axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
↪9186)')
axes4[0].get_legend().remove()
(q5_CI_conv_LASSO).plot.barh(ax=axes4[1], label="Coverage Probability",␣
↪color="dodgerblue")
axes4[1].title.set_text("Coverage Probability LASSO, percentile")
axes4[1].axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
↪8814)')
axes4[1].axvline(x=0.9, color='red', ls=':', label='0.9')
axes4[1].axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
↪9186)')
axes4[1].legend(bbox_to_anchor=(1.05,1))
```

```
Average coverage probability of OLS = 0.9946399999999997
Average coverage probability LASSO = 0.9757999999999997
```

[10]: <matplotlib.legend.Legend at 0x1454a1d2880>



Answer: Again, most coverage probabilities are close to 1. We see that the results are not very different from those obtained in question 3.

6. Estimate by simulation the cov. prob. for OLS of 90% equal-tailed two-sided percentile-t bootstrap confidence intervals based on the quantiles of the root

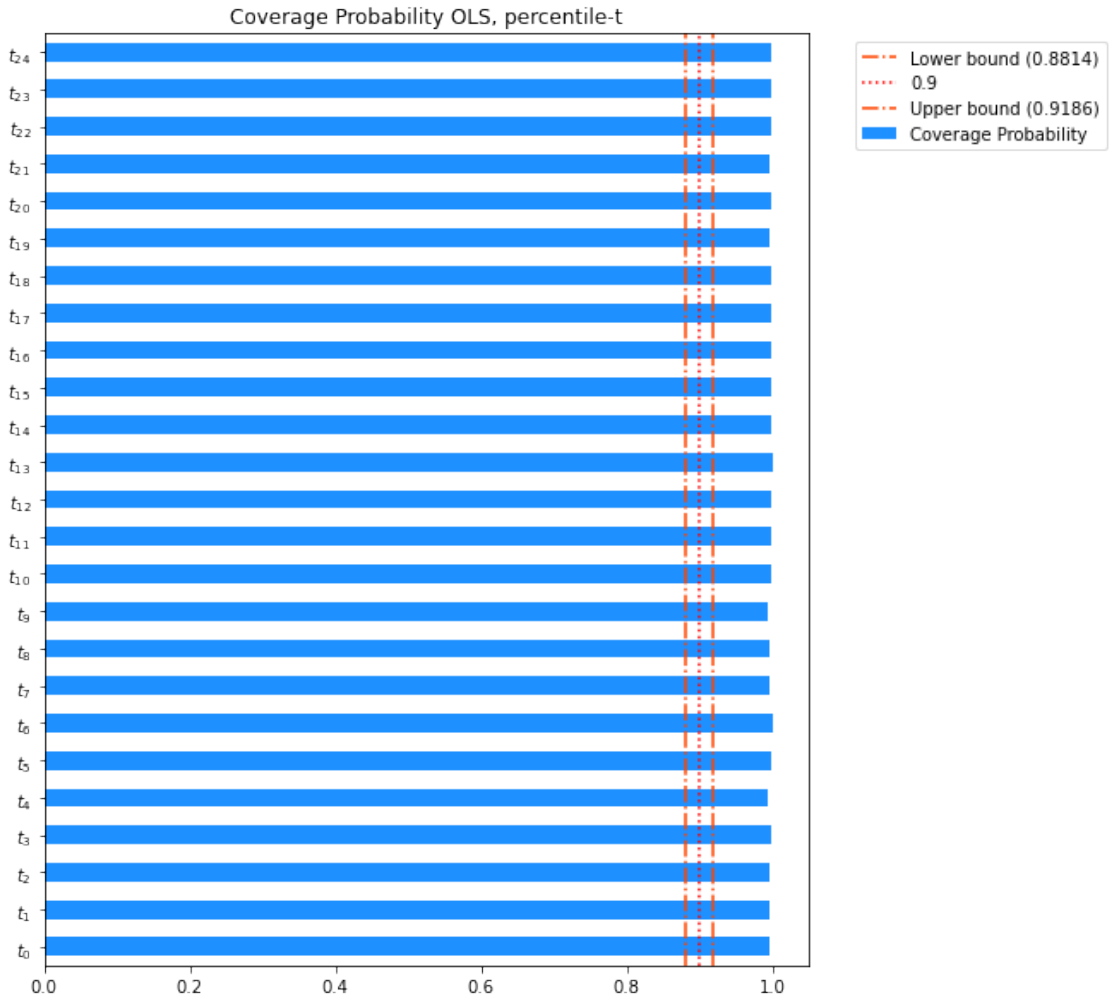
$$(\hat{\beta}^* - \hat{\beta})/SE(\hat{\beta}^*),$$

where $SE(\hat{\beta}^*)$ is based on $s^{*2}(X^{*\prime}X^*)^{-1}$.

[11]:
```
# Percentiles
percentile_t_lb = b_OLS - t_ub_boot_OLS*b_OLS_SE
percentile_t_ub = b_OLS - t_lb_boot_OLS*b_OLS_SE

# Create confidence intervals based on bootstrap
t_boot_ols_CI = pd.DataFrame(percentile_t_lb).le(pd.Series(beta).T, axis=1) &␣
↪pd.DataFrame(percentile_t_ub).ge(pd.Series(beta).T, axis=1)

# Calculate coverage probabalities
t_boot_ols_cov_avg_prob = np.mean(t_boot_ols_CI.sum()/REP)

print("Average coverage probability of OLS = {}".
↪format(t_boot_ols_cov_avg_prob))

# Plot
q6_CI_conv = pd.DataFrame({"Coverage Probability": (t_boot_ols_CI.sum()/REP)})
indices = []
for i in range(len(q6_CI_conv.index)):
    indices.append("$t_{" + str(i) + "}$")
q6_CI_conv.index = indices
fig5, axes5 = plt.subplots(1, 1, figsize=(8,10))
(q6_CI_conv).plot.barh(ax=axes5, label="Coverage Probability, percentile-t",␣
↪color="dodgerblue")
axes5.title.set_text("Coverage Probability OLS, percentile-t")
axes5.axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
↪8814)')
axes5.axvline(x=0.9, color='red', ls=':', label='0.9')
axes5.axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
↪9186)')
axes5.legend(bbox_to_anchor=(1.05,1))
```

```
Average coverage probability of OLS = 0.9975199999999999
```

[11]:
```
<matplotlib.legend.Legend at 0x1454a8bdfd0>
```



7. What problems would you encounter if you wanted to implement the percentile-t intervals for the LASSO? How could you remedy these problems (you don't have to implement this)?

Answer: To calculate the percentile-t intervals, we would need the standard errors of the bootstrap LASSO coefficients, which are very hard to calculate analytically. A way to solve this is to perform a nested bootstrap and estimate the standard errors by simulation. Using these standard errors, we could then calculate the bootstrapped t-values.

8. Estimate by simulation the cov. prob. for LASSO of 90% two-sided bias-corrected and accelerated (BC_a) confidence intervals. For this, use the bootstrap to estimate the (median) bias and the Jackknife for the acceleration constant. In the BC_a method, the quantiles are adjusted:

$$\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z_{\alpha/2})}\right), \alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha}}{1 - \hat{a}(\hat{z}_0 + z_{1-\alpha/2})}\right)$$

with $z_{0.95} = 1.645$. Here

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\sum_{i=1}^{n} \mathbb{1}\{\hat{\theta}^*(b) < \hat{\theta}\}}{B}\right)$$

and

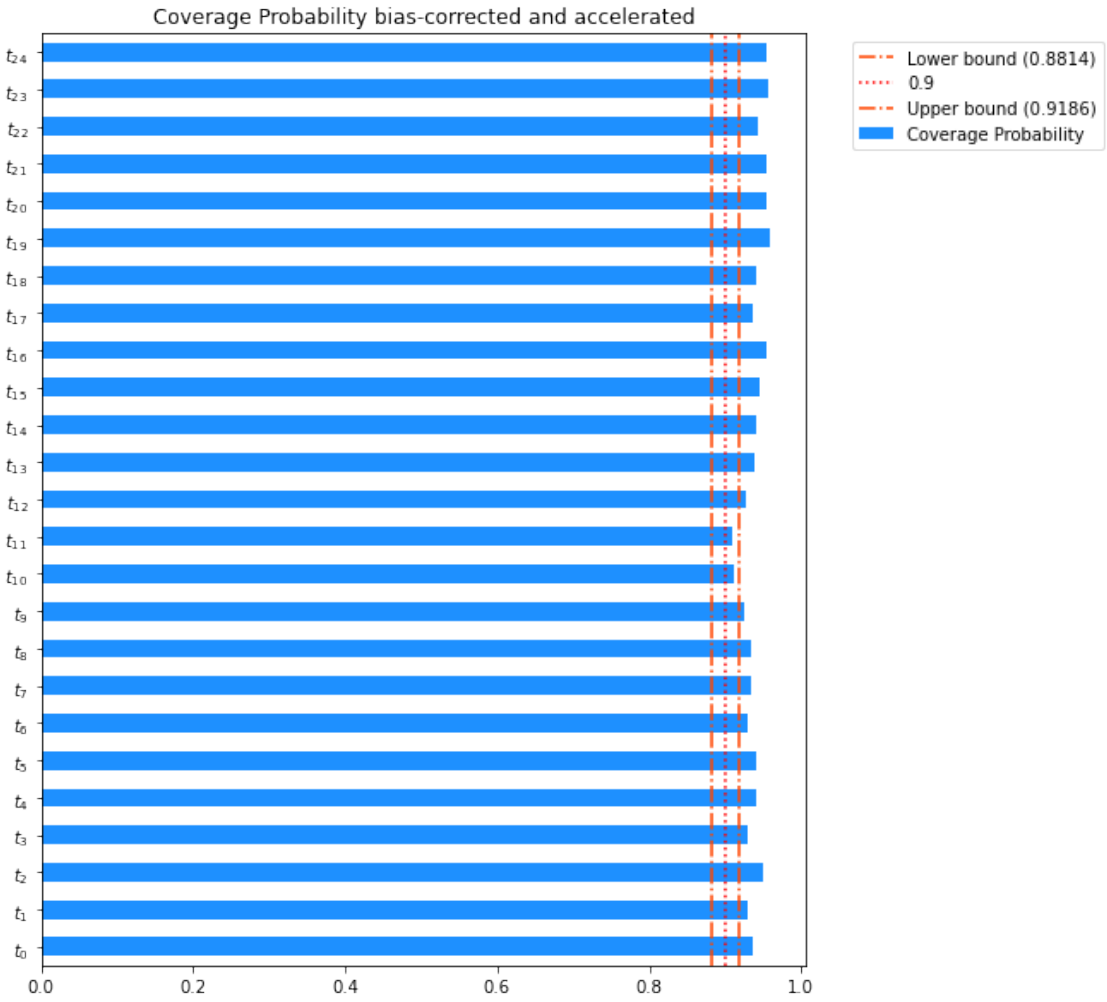
$$\hat{a} = \frac{\sum_{i=1}^{n}(\hat{\theta}_{(\cdot)} - \hat{\theta}_{(i)})^3}{6\{\sum_{i=1}^{n}(\hat{\theta}_{(\cdot)} - \hat{\theta}_{(i)})^2\}^{3/2}}$$

with $\hat{\theta}_{(\cdot)} = \sum_{i=1}^{n}\hat{\theta}_{(i)}/n$; see for more details Section 14.3 of Efron, B., & Tibshirani, R. J. (1994). An introduction to the bootstrap: link).

[12]:
```
# Create confidence intervals based on bootstrap
bca_lasso_CI = pd.DataFrame(lb_bca_CI_lasso).le(pd.Series(beta).T, axis=1) & pd.
 ↪DataFrame(ub_bca_CI_lasso).ge(pd.Series(beta).T, axis=1)

# Calculate coverage probabalities
boot_lasso_cov_avg_prob = np.mean(bca_lasso_CI.sum()/REP)

print("Average coverage probability bias-corrected and accelerated = {}".
 ↪format(boot_lasso_cov_avg_prob))

# Plot
q8_CI_conv = pd.DataFrame({"Coverage Probability": (bca_lasso_CI.sum()/REP)})
indices = []
for i in range(len(q8_CI_conv.index)):
    indices.append("$t_{" + str(i) + "}$")
q8_CI_conv.index = indices
fig6, axes6 = plt.subplots(1, 1, figsize=(8,10))
(q8_CI_conv).plot.barh(ax=axes6, label="Coverage Probability",␣
 ↪color="dodgerblue")
axes6.title.set_text("Coverage Probability bias-corrected and accelerated")
axes6.axvline(x=0.8814, color='orangered', ls='-.', label='Lower bound (0.
 ↪8814)')
axes6.axvline(x=0.9, color='red', ls=':', label='0.9')
axes6.axvline(x=0.9186, color='orangered', ls='-.', label='Upper bound (0.
 ↪9186)')
axes6.legend(bbox_to_anchor=(1.05,1))
```

```
Average coverage probability bias-corrected and accelerated = 0.9386
```

[12]:
```
<matplotlib.legend.Legend at 0x1454abd3700>
```



Answer: Although for most regressors the t-values are still significantly higher than 0.9, these results are a big improvement from those seen in question 3 and question 5. This means that the bias-corrected and accelerated confidence intervals lead to a smaller ERP.

9. Based on all the results of this assignment, which inference procedure would you advise a practitioner that wants to conduct inference in a model described by the DGP? Motivate your recommendation.

Answer: From the coverage probabilities in the previous exercises, we can see that in this small-sample environment, our estimators are likely to significantly overfit on the data. We know that thanks to the added regularization, this problem will be less with the LASSO estimator, which is also shown in the RMSE of question 1. However, constructing confidence intervals with an acceptable ERP proved difficult. If this practicioner has access to enough computing power, we would recommend trying a nested bootstrap to estimate a percentile-t interval, which is likely to be more accurate.

For OLS, the bootstrapped bias-corrected and accelerated confidence intervals are the best performing bootstrap method. However, we see that the usual non-bootstrapped OLS confidence intervals

are actually closer to their nominal size and would be favoured. We also expect that this method works well, because the assumptions required are satisfied in the model. Our recommendation would thus be to use the OLS estimator and construct non-bootstrapped confidence intervals