Mathematical and Computational Statistics with a View Towards Finance and Risk Management - Assignment 3

Jaka Golob (20-716-791) Yannik Haller (12-918-645)

04 11 2020

Information

This is our (Jaka Golob and Yannik Haller) solution to Assignment 3 of the course "Mathematical and Computational Statistics with a View Towards Finance and Risk Management". For a better overview, we decided to also write down the questions in blue. Our solutions to these questions are written in black/included in R-Chunks or resulting outputs. Some resulting values from R-calculations are directly inserted into our answer-texts. Comment-symbols from R-Outputs (#) are removed (i.e. summaries are denoted without hashtags to gain readability).

Preparation:

In a first step we need to set the appropriate working directory (this must be done for each machine individually) and load all required packages.

```
# Set working directory
setwd(
   "~/Yannik/UZH/20HS/Mathematical and Computational Statistics with a View Towards Finance and Risk

# Load required packages
pkg_sys <- c("datasets", "foreign", "MASS", "stats","stats4")
pkg     <- c("dplyr","tidyr","ggplot2","stargazer","reshape2","readr","haven","dummies",
             "Hmisc","lmtest","sandwich","doBy","readxl","multiwayvcov","miceadds","car",
             "purrr","knitr","wesanderson","ggvis","shiny","lubridate","reporttools",
             "stringr", "data.table", "devtools","tinytex","rmarkdown","matlib",
             "ggiraphExtra","estimatr","Jmisc","lfe","metRology","QRM","EnvStats",
             "PerformanceAnalytics","SimCorrMix","mixtools","foreach","doParallel",
             "gridExtra","MARSS","forecast","nlme","expm","kableExtra")

invisible(lapply(c(pkg, pkg_sys), library, character.only = TRUE))
rm(list=ls())
```

Note that we assume that the required packages are already installed on your machine. If this is not the case, simply run **lapply(pkg, install.packages, character.only = FALSE)** right after defining the *pkg* variable.

Task a)

Make codes to simulate a linear regression model with stationary AR(p) disturbances, Gaussian innovations, $N(0, \sigma^2)$. For the regressor matrix, you can use a constant and a time trend, and for p, you can take $p = 2$. Use T observations, where T might be 50 or 100. Maybe take $\sigma = 2$, and note that larger values should NOT affect the below results - it is just a scaling parameter, and not a measure of "information" and is not related to estimation accuracy. But do check this, by using different values of sigma.

In the first task we create a code to simulate a linear regression model with stationary AR(2) disturbances and Gaussian innovations which follow $N(0, \sigma^2)$. For the regressor matrix X we use a constant and a time trend as given. We determine the sequence length as $t = 100$, the regression coefficients as $\beta_0 = 0$ and $\beta_1 = 0$, the AR coefficients as $\phi_1 = 0.5$ and $\phi_2 = -0.2$, and we show that these coefficients indeed lead to a stationary model/process as the absolute value of both roots of the polynomial a(L) is greater than 1.

```
## First we define the length t, the number of lags p, the AR parameters phi1 and phi2
## and the standard deviation sigma
t     <- 100
p     <- 2
beta0 <- 0
beta1 <- 0
phi1  <- 0.5
phi2  <- -0.2
sigma <- 2
beta  <- c(beta0,beta1)
phi   <- c(phi1,phi2)

## Next we want to verify that the AR(2) coefficients we chose lead to a stationary
## model, by calculating the p roots of the polynomial a(L)
abs(polyroot(c(1,-phi)))
```

```
[1] 2.236068 2.236068
```

```
# As the absolute value of all p roots of the polynomial a(L) is greater than 1, the
# AR(2) we specify indeed is stationary

## Next we set a seed to ensure reproducibility of our results, simulate the innovations
## by simulating a sequence of Gaussian innovations of length t+p
set.seed(1)
innov <- rnorm(t+p,0,sigma)

## Then we use the generated sequence of innovations to simulate the series of our
## residuals epsilon. As for the first p elements of epsilon not all the p lagged
## values are available, we drop those and only keep the last t elements of the
## series
res <- filter(innov, filter = c(phi1,phi2),method = "recursive")[c((p+1):(t+p))]

## Then we create the regression matrix
X      <- matrix(NA, t, 2)
X[,1]  <- 1
X[,2]  <- c(1:t)

## Next, we use the residuals and the regression matrix to simulate a linear
## regression model
Y <- X%*%beta + res
```

However, as we set $\beta_0 = 0$ and $\beta_1 = 0$ the last step is essentially superfluous as we could directly equate res and Y (which is what we apply in the subsequent tasks)

Task b)

Estimate the parameters (the regression β coefficients, the 2 AR parameters, and σ, from the $N(0, \sigma^2)$ innovations) using iterative least squares.

We start by defining a function which calculates the covariance matrix SigmaInv and use it to define another function which estimates the model parameters by means of iterative least squares (ILS).

```
# First, we define a function to calculate the inverse covariance matrix SigmaInv
SigmaInv <- function(a,t){
  p    <- length(a)
  a    <- -a
  firrowP  <- c(1,rep(0,t-1))
  fircolP  <- c(1,a,rep(0,t-p-1))
  P        <- toeplitz(fircolP,firrowP)
  firrowQ1 <- a[c(seq(p,1,-1))]
  fircolQ1 <- c(a[p],rep(0,p-1))
  Q1       <- toeplitz(fircolQ1,firrowQ1)
  Q        <- rbind(Q1,matrix(0,t-p,p))

  V <- (t(P)%*%P)-(Q%*%t(Q))
  V
}
```

```
## Then, we define a function to estimate the parameters of a VAR(2) model using
## iterative least squares
Var2RegILS <- function(Y,X){
# get length of time series
ylen <- length(Y)
## Start by estimating the beta by OLS and extract the residuals
beta <- inv(t(X)%*%X)%*%t(X)%*%Y
res <- Y-X%*%beta
## Estimate the AR(p) parameters using the OLS residuals
Z   <- cbind(shift(res,-1),shift(res,-2))[c(3:ylen),]
phi <- inv(t(Z)%*%Z)%*%t(Z)%*%res[c(3:ylen)]
## Calculate the inverse of the covariance matrix and use it to estimate beta by gls
SigMatInv <- SigmaInv(phi,ylen)
beta <- inv(t(X)%*%SigMatInv%*%X)%*%t(X)%*%SigMatInv%*%Y
res <- Y-X%*%beta
## Repeat until convergence with maximum tolerated absolute difference between
## the estimates resulting from two consecutive estimations being 1e-4 and
## maximum number of repetitions = 500 (to interrupt loop in case of an occurring
## convergence series)
tol <- 1e-4
i <- 1
runs <- 0
while(i > tol & runs < 5000){
  Z         <- cbind(shift(res,-1),shift(res,-2))[c(3:ylen),]
  phi_new   <- inv(t(Z)%*%Z)%*%t(Z)%*%res[c(3:ylen)]
  SigMatInv <- SigmaInv(phi_new,ylen)
  beta_new  <- inv(t(X)%*%SigMatInv%*%X)%*%t(X)%*%SigMatInv%*%Y
  res       <- Y-X%*%beta_new
  i         <- max(abs(phi-phi_new),abs(beta-beta_new))
  phi       <- phi_new
  beta      <- beta_new
  runs      <- runs + 1
}

## Repeat the estimation of the AR(p) parameters once again to get the estimates
## which correspond to the above loop returns
```

```
Z   <- cbind(shift(res,-1),shift(res,-2))[c(3:ylen),]
phi <- inv(t(Z)%*%Z)%*%t(Z)%*%res[c(3:ylen)]

## Calculate the filtered innovations and their standard deviation
res_fit <- phi[1]*shift(res,-1)[c(3:ylen)] +
  phi[2]*shift(res,-2)[c(3:ylen)]
innov_filt <- res[c(3:ylen)] - res_fit
sigma      <- sd(innov_filt)

## Return a vector containing all estimated parameters
rbind(beta,phi,sigma)
}
```

```
## Finally, apply the function to estimate the parameters
params    <- Var2RegILS(Y,X)
## beta
beta_mle  <- params[c(1,2)]
beta_mle
```

```
[1] 0.108251701 0.002577035
```

```
## phi
phi_mle   <- params[c(3,4)]
phi_mle
```

```
[1]  0.5446523 -0.2609572
```

```
# sigma
sigma_mle <- params[5]
sigma_mle
```

```
[1] 1.777467
```

Note that using ILS this way can (for some simulations) lead to convergence series for all model parameters. We could solve this problem by recording the whole sequence of parameters of this convergence series and then just select the set of parameters which leads to the smallest MSE. However, since R provides well a established function to estimate the model parameters of such models in a very efficient way (i.e. the arima function) and in addition allows to conveniently restrict the estimation to stationary processes, we decide to make use of this function in the subsequent tasks. Especially in the bootstrap, where the estimation process has to be done very often, its efficiency is a great advantage. The arima function is originally programmed to estimate arima models. However, an arima model with all parameters except p being 0 is just an AR(p) model. When this is appropriately specified, the arima function does exactly what we need: it performs a regression of Y on the specified regression matrix X (which we were given) to get starting values and then applies maximum likelihood to iteratively estimate the desired model parameters. Because the estimation is not always successful (in particular when working with distributions for which certain moments diverge as it is the case for the Cauchy distribution), we create the function **Var2Reg**, which in the first step works with the arima function to get estimates for the AR(2) model, checks whether the estimation process was successful and then extracts the estimated parameters and filtered innovations in case the estimation was successful. If the estimation by the arima function was not successful the **Var2Reg** computes "naïve" OLS estimates for the model parameters together with the associated filtered innovations instead (which appears to perform quite well). We keep track of whether the estimation breaks down – respectively, whether we had to use the second "naïve" estimation technique. It turns out that for the case when we simulate the time series with normally distributed innovations it does not break down at all, whereas in the Cauchy case it actually does breaks down, but negligibly seldom – namely less than 0.1% of the time.

```
### In this part we define a function to estimate the MLE model parameters of a VAR(2)
### model
## First, we set up the optimization parameters
opt <- list(maxit = 1e3)
```

```r
# Then, we define the actual function
Var2Reg <- function(Y,X){
  # Get the length of the sequence Y
  ylen <- length(Y)
  # Try to estimate the MLE of the model parameters using the arima function while
  # keeping track of whether the estimation process has failed or not using the
  # "failed" variable.
  # Note that in the arima function all parameters are estimated as desired (the
  # function gets starting values using conditional-sum-of-squares and then applies
  # maximum likelihood)
  AR2 <- try(arima(Y, order = c(2,0,0), xreg = X, include.mean = F,
             transform.pars = T, method = "ML", optim.control = opt), silent = T)
  if(class(AR2) == "try-error"){
    ## If the estimation process failed, we calculate some "naïve" OLS estimates
    ## for the model parameters as well as the associated filtered innovations
    ## (which actually appears to deliver quite accurate estimates)
    # Regression coefficients beta (using "naïve" OLS)
    beta_mle <- inv(t(X)%*%X)%*%t(X)%*%Y
    res <- Y-X%*%beta_mle
    # AR parameters phi (using "naïve" OLS)
    Z   <- cbind(shift(res,-1),shift(res,-2))[c(3:ylen),]
    phi_mle <- inv(t(Z)%*%Z)%*%t(Z)%*%res[c(3:ylen)]
    # Sigma of innovations (using the above mle parameters & assuming they are
    # Gaussian)
    res_fit <- phi_mle[1]*shift(res,-1)[c(3:ylen)] +
      phi_mle[2]*shift(res,-2)[c(3:ylen)]   # get the t-p last fitted residuals
    innov_filt <- res[c(3:ylen)] - res_fit  # get the t-p filtered innovations
    sigma_mle <- sd(innov_filt)             # get their standard deviation
    # Impute a very high value for the sigma_mle if its MLE is not finite
    # (this is needed for the cauchy case)
    sigma_mle[is.na(sigma_mle)] <- 1e10
    # Finally, since the arima function failed we define failed = 1
    failed <- 1

  }else{
    ## If the estimation process was successful, we extract the estimated
    ## parameters and the filtered innovations into variables
    # Regression coefficients beta
    beta_mle  <- as.numeric(AR2$coef)[c(3,4)]
    # AR parameters phi
    phi_mle   <- as.numeric(AR2$coef)[c(1,2)]
    # Sigma of innovations (using the above mle parameters & assuming they are
    # Gaussian)
    innov_filt <- AR2$residuals[c(3:ylen)]  # get the t-p filtered innovations
    sigma_mle <- sd(innov_filt)             # get their standard deviation
    # Impute a very high value for the sigma_mle if its MLE is not finite
    # (this is needed for the cauchy case)
    sigma_mle[is.na(sigma_mle)] <- 1e10
    # Finally, since the arima function was successful we define failed = 0
    failed <- 0
  }
  cbind(t(beta_mle),t(phi_mle),sigma_mle,failed,t(innov_filt))
}
```

Task c)

From your estimation code, you get the filtered innovations, T-p of them, and if the model you estimate is correctly specified (namely, the correct X matrix and use of the correct p for the AR structure) then, those filtered innovations will be approximately IID $N(0, \sigma^2)$. Now apply the bootstrap, both nonparametric and parametric, to get confidence intervals for the model parameters (all of them). Use, say, 90% intervals.

In the previous task we get t-p filtered innovations. Our goal is now to get confidence intervals for the estimated model parameters. We achieve this by applying both the nonparametric and parametric bootstrap based on the filtered innovations.

In the nonparametric bootstrap, we generate NB resamples of the t-p filtered innovations (from the estimation of the baseline series Y) with replacement. We then use these samples to generate NB new time series, using the estimated coefficients from the baseline series. Thereafter we use our **Var2Reg** function to estimate the model parameters for the bootstrap series in the same way as in task b. As we repeat this NB times, we get a distribution for all model parameters. We take the 5 percent and 95 percent quantiles of these distributions to determine the boundaries of the corresponding confidence intervals.

The procedure for the parametric bootstrap then is pretty similar. For the t-p filtered innovations we estimate their distribution (which is $N(0, \sigma^2_{mle})$) and then simulate NB samples of Gaussian innovations using σ_{mle} as their standard deviation. We then use these NB innovation series and precede in the same way as in the nonparametric case.

Nonparametric bootstrap

```
## Define the number of resamples (i.e. NB) we want to generate
NB <- 400

## Create a data frame to store the NB resamples of the residuals
NP_Boots_eps <- matrix(NA, t-p, NB)
NP_Boots_eps <- as.data.frame(NP_Boots_eps)

## Create a data frame to store the NB Y series
NP_Boots <- matrix(NA, t-2*p, NB)
NP_Boots <- as.data.frame(NP_Boots)

## Create a data frame to store all mle estimates
NP_Boots_params           <- matrix(NA, NB, 5)
NP_Boots_params           <- as.data.frame(NP_Boots_params)
colnames(NP_Boots_params) <- c("beta0_mle","beta1_mle","phi1_mle",
                               "phi2_mle","sigma_mle")

## Create a data frame to store the resulting confidence intervals
NP_CI           <- matrix(NA, 5, 2)
NP_CI           <- as.data.frame(NP_CI)
colnames(NP_CI) <- c("lower bound","upper bound")
rownames(NP_CI) <- c("NP_CI beta0","NP_CI beta1","NP_CI phi1",
                     "NP_CI phi2","NP_CI sigma")

## Create a variable to keep track of the percentage where naïve estimation has
## been used
naive_NPB <- 0

## Use the code from task a) to generate an AR(p) model with t observations
# Y-sequence
set.seed(123)
epsilon <- rnorm(t+p,0,sigma)
Y <- filter(epsilon,filter = c(phi1,phi2),method = "recursive")[c((p+1):(t+p))]
# X-matrix
```

```r
X      <- matrix(NA, t, 2)
X[,1]  <- 1
X[,2]  <- c(1:t)

## Estimate the model using the code from task b)
AR2 <- Var2Reg(Y,X)

## Extract the mle model parameters and the filtered innovations
beta_mle  <- AR2[c(1,2)]
phi_mle   <- AR2[c(3,4)]
sigma_mle <- AR2[5]
res_filt  <- AR2[c(7:length(AR2))]

## Adjust the X-Matrix to the new length of the Y sequence
X      <- matrix(NA, nrow(NP_Boots), 2)
X[,1]  <- 1
X[,2]  <- c(1:nrow(NP_Boots))

## Generate the NB resamples of the residuals using the non-parametric approach
for(B in c(1:NB)){
  set.seed(B)
  NP_Boots_eps[,B] <- sample(res_filt, (t-p), replace = T)
}

## Generate NB Y series from the NB resamples
for(B in c(1:NB)){
  NP_Boots[,B] <- filter(NP_Boots_eps[,B], filter = c(phi_mle[1],phi_mle[2]),
                         method = "recursive")[c((p+1):(t-p))]
  NP_Boots[,B] <- NP_Boots[,B] + (X %*% as.matrix(beta_mle))
}

## Estimate the model parameters for all NB simulations and store them into the
## NP_Boots_params variable
for(B in c(1:NB)){
  ## Estimate the model coefficients by means of the Var2Reg function
  AR2_NPB <- Var2Reg(NP_Boots[,B],X)
  ## Extract the estimated parameters into variables
  # Regression coefficients beta
  NP_Boots_params[B,c(1,2)] <- AR2_NPB[c(1,2)]
  # AR parameters phi
  NP_Boots_params[B,c(3,4)] <- AR2_NPB[c(3,4)]
  # Sigma of innovations
  NP_Boots_params[B,5] <- AR2_NPB[5]
  ## Check whether "naïve" estimators are used
  naive_NPB <- naive_NPB + AR2_NPB[6]/(NB)
}

## Calculate the CI for each parameter and store them into the NP_CI variable
# beta0
NP_CI[1,] <- as.numeric(quantile(NP_Boots_params[,1], probs = c(0.05,0.95)))
# beta1
NP_CI[2,] <- as.numeric(quantile(NP_Boots_params[,2], probs = c(0.05,0.95)))
# phi1
NP_CI[3,] <- as.numeric(quantile(NP_Boots_params[,3], probs = c(0.05,0.95)))
# phi2
NP_CI[4,] <- as.numeric(quantile(NP_Boots_params[,4], probs = c(0.05,0.95)))
# sigma
```

```
NP_CI[5,] <- as.numeric(quantile(NP_Boots_params[,5], probs = c(0.05,0.95)))

## Take a look at some summary statistics of the resulting parameters
stargazer(NP_Boots_params, type = "text", median = T)
```

```
=================================================================
Statistic  N   Mean  St. Dev.  Min   Pctl(25) Median Pctl(75)  Max
-----------------------------------------------------------------
beta0_mle 400 0.030   0.492   -1.530  -0.285  0.036   0.355   1.576
beta1_mle 400 0.004   0.009   -0.020  -0.002  0.003   0.010   0.027
phi1_mle  400 0.448   0.106   0.125   0.379   0.450   0.508   0.767
phi2_mle  400 -0.238  0.098   -0.554  -0.299  -0.239  -0.177  0.047
sigma_mle 400 1.781   0.127   1.386   1.688   1.789   1.858   2.171
-----------------------------------------------------------------
```

```
## Take a look at the resulting confidence intervals
NP_CI
```

```
            lower bound upper bound
NP_CI beta0 -0.78240309  0.80972749
NP_CI beta1 -0.01104751  0.01790522
NP_CI phi1   0.27769349  0.61735311
NP_CI phi2  -0.39964025 -0.06891440
NP_CI sigma  1.55949486  1.98116520
```

```
## Take a look at the share of estimations which used the naive estimators
naive_NPB
```

```
[1] 0
```

Parametric bootstrap

To get comparable results from the nonparametric and parametric bootstrap, we decided to restrict the length of the parametric bootstrap resamples of epsilon to the same length as the nonparametric bootstrap resamples of epsilon (i.e. t-p). This therefore leads to NB simulated Y sequences of length t-2p.

```
## Define the number of resamples (i.e. NB) we want to generate
NB <- 400

## Create a data frame to store the NB resamples of the residuals
P_Boots_eps <- matrix(NA, t-p, NB)
P_Boots_eps <- as.data.frame(P_Boots_eps)

## Create a data frame to store the NB Y series
P_Boots <- matrix(NA, t-2*p, NB)
P_Boots <- as.data.frame(P_Boots)

## Create a data frame to store all mle estimates
P_Boots_params           <- matrix(NA, NB, 5)
P_Boots_params           <- as.data.frame(P_Boots_params)
colnames(P_Boots_params) <- c("beta0_mle","beta1_mle","phi1_mle",
                              "phi2_mle","sigma_mle")

## Create a data frame to store the resulting confidence intervals
P_CI           <- matrix(NA, 5, 2)
P_CI           <- as.data.frame(P_CI)
colnames(P_CI) <- c("lower bound","upper bound")
rownames(P_CI) <- c("P_CI beta0","P_CI beta1","P_CI phi1",
                    "P_CI phi2","P_CI sigma")
```

```r
## Create a variable to keep track of the percentage where naïve estimation has
## been used
naive_PB <- 0

## Use the code from task a) to generate an AR(p) model with t observations
# Y-sequence
set.seed(123)
epsilon <- rnorm(t+p,0,sigma)
Y <- filter(epsilon,filter = c(phi1,phi2),method = "recursive")[c((p+1):(t+p))]
# X-matrix
X      <- matrix(NA, t, 2)
X[,1]  <- 1
X[,2]  <- c(1:t)

## Estimate the model using the code from task b)
AR2 <- Var2Reg(Y,X)

## Extract the mle model parameters and the filtered innovations
beta_mle  <- AR2[c(1,2)]
phi_mle   <- AR2[c(3,4)]
sigma_mle <- AR2[5]
res_filt  <- AR2[c(7:length(AR2))]

## Assuming Gaussian residuals, extract their mean and the sigma_mle
P_sigma <- sigma_mle
P_mu    <- mean(res_filt)

# Adjust the X-Matrix to the new length of the Y sequence
X      <- matrix(NA, nrow(P_Boots), 2)
X[,1]  <- 1
X[,2]  <- c(1:nrow(P_Boots))

## Generate the NB resamples of the residuals using the parametric approach
for(B in c(1:NB)){
  set.seed(B)
  P_Boots_eps[,B] <- rnorm((t-p), P_mu, P_sigma)
}

## Generate NB Y series from the NB resamples
for(B in c(1:NB)){
  P_Boots[,B] <- filter(P_Boots_eps[,B], filter = c(phi_mle[1],phi_mle[2]),
                        method = "recursive")[c((p+1):(t-p))]
  P_Boots[,B] <- P_Boots[,B] + (X %*% as.matrix(beta_mle))
}

## Estimate the model parameters for all NB simulations and store them into the
## P_Boots_params variable
for(B in c(1:NB)){
  ## Estimate the model coefficients by means of the Var2Reg function
  AR2_PB <- Var2Reg(P_Boots[,B],X)
  ## Extract the estimated parameters into variables
  # Regression coefficients beta
  P_Boots_params[B,c(1,2)] <- AR2_PB[c(1,2)]
  # AR parameters phi
  P_Boots_params[B,c(3,4)] <- AR2_PB[c(3,4)]
  # Sigma of innovations
  P_Boots_params[B,5] <- AR2_PB[5]
```

```
  ## Check whether "naïve" estimators are used
  naive_PB <- naive_PB + AR2_PB[6]/(NB)
}

## Calculate the CI for each parameter and store them into the P_CI variable
# beta0
P_CI[1,] <- as.numeric(quantile(P_Boots_params[,1], probs = c(0.05,0.95)))
# beta1
P_CI[2,] <- as.numeric(quantile(P_Boots_params[,2], probs = c(0.05,0.95)))
# phi1
P_CI[3,] <- as.numeric(quantile(P_Boots_params[,3], probs = c(0.05,0.95)))
# phi2
P_CI[4,] <- as.numeric(quantile(P_Boots_params[,4], probs = c(0.05,0.95)))
# sigma
P_CI[5,] <- as.numeric(quantile(P_Boots_params[,5], probs = c(0.05,0.95)))

## Take a look at some summary statistics of the resulting parameters
stargazer(P_Boots_params, type = "text", median = T)
```

```
=================================================================
Statistic  N   Mean  St. Dev.  Min   Pctl(25) Median Pctl(75)  Max
-----------------------------------------------------------------
beta0_mle 400 0.029   0.486   -1.751  -0.282  0.022   0.329   1.530
beta1_mle 400 0.004   0.009   -0.020  -0.001  0.004   0.010   0.039
phi1_mle  400 0.454   0.107   0.174   0.379   0.453   0.524   0.750
phi2_mle  400 -0.242  0.101   -0.537  -0.316  -0.248  -0.180  0.073
sigma_mle 400 1.776   0.136   1.442   1.687   1.774   1.861   2.168
-----------------------------------------------------------------
```

```
## Take a look at the resulting confidence intervals
P_CI
```

```
           lower bound upper bound
P_CI beta0 -0.748530815  0.82101661
P_CI beta1 -0.009920915  0.01785681
P_CI phi1   0.280511788  0.62745804
P_CI phi2  -0.392504724 -0.06760494
P_CI sigma  1.549178713  2.01313425
```

```
## Take a look at the share of estimations which used the naive estimators
naive_PB
```

```
[1] 0
```

Task d)

Package all of the above in a FOR loop, and keep track of how many times each true model parameter falls within its bootstrap-based confidence interval, and have separate reporting for the nonparametric and parametric bootstrap. Maybe make a nice table in LaTeX to report results.

In this task we have to run the previous procedure several times (in our case we run rep repetitions), and check how often the true model parameters actually lie within the corresponding confidence interval we determined. Applying this technique leads to the following coverages:

- Nonparametric Bootstrap: $\beta_0 : 0.88,\ \beta_1 : 0.877,\ \phi_1 : 0.863,\ \phi_2 : 0.88,\ \sigma : 0.79$
- Parametric Bootstrap: $\beta_0 : 0.88,\ \beta_1 : 0.883,\ \phi_1 : 0.887,\ \phi_2 : 0.887,\ \sigma : 0.82$

For this series (which was simulated with normally distributed innovations) we see that all the value are close to the wanted value of 0.9.

```
## Define the number of repetitions we want to apply
rep <- 300

## Define the number of resamples (i.e. NB) we want want to generate
NB <- 400

## Store the true model parameters into variables
beta0 <- 0
beta1 <- 0
phi1  <- 0.5
phi2  <- -0.2
sigma <- 2
```

```
### Create all the needed variables for the nonparametric bootstrap
## Create a data frame to store the NB resamples of the residuals
NP_Boots_eps <- matrix(NA, t-p, NB)
NP_Boots_eps <- as.data.frame(NP_Boots_eps)

## Create a data frame to store the NB Y series
NP_Boots <- matrix(NA, t-2*p, NB)
NP_Boots <- as.data.frame(NP_Boots)

## Create a data frame to store all mle estimates
NP_Boots_params           <- matrix(NA, NB, 5)
NP_Boots_params           <- as.data.frame(NP_Boots_params)
colnames(NP_Boots_params) <- c("beta0_mle","beta1_mle","phi1_mle",
                               "phi2_mle","sigma_mle")

## Create a data frame to store the resulting confidence intervals
NP_CI           <- matrix(NA, 5, 2)
NP_CI           <- as.data.frame(NP_CI)
colnames(NP_CI) <- c("lower bound","upper bound")
rownames(NP_CI) <- c("NP_CI beta0","NP_CI beta1","NP_CI phi1",
                     "NP_CI phi2","NP_CI sigma")

## Create a data frame to store the boolean indicating whether the true
## model parameter falls within its bootstrap-based CI
NP_Coverage <- matrix(NA, rep, 5)
NP_Coverage <- as.data.frame(NP_Coverage)
colnames(NP_Coverage) <- c("beta0","beta1","phi1","phi2","sigma")


### Create all the needed variables for the parametric bootstrap
## Create a data frame to store the NB resamples of the residuals
```

```
P_Boots_eps <- matrix(NA, t-p, NB)
P_Boots_eps <- as.data.frame(P_Boots_eps)

## Create a data frame to store the NB Y series
P_Boots <- matrix(NA, t-2*p, NB)
P_Boots <- as.data.frame(P_Boots)

## Create a data frame to store all mle estimates
P_Boots_params           <- matrix(NA, NB, 5)
P_Boots_params           <- as.data.frame(P_Boots_params)
colnames(P_Boots_params) <- c("beta0_mle","beta1_mle","phi1_mle",
                              "phi2_mle","sigma_mle")

## Create a data frame to store the resulting confidence intervals
P_CI           <- matrix(NA, 5, 2)
P_CI           <- as.data.frame(P_CI)
colnames(P_CI) <- c("lower bound","upper bound")
rownames(P_CI) <- c("P_CI beta0","P_CI beta1","P_CI phi1",
                    "P_CI phi2","P_CI sigma")

## Create a data frame to store the boolean indicating whether the true
## model parameter falls within its bootstrap-based CI
P_Coverage <- matrix(NA, rep, 5)
P_Coverage <- as.data.frame(P_Coverage)
colnames(P_Coverage) <- c("beta0","beta1","phi1","phi2","sigma")

## Create variables to keep track of the percentage where naïve estimation has
## been used
# in the baseline model
naive_glob <- 0
# in the nonparametric bootstrap
naive_NPB <- 0
# in the parametric bootstrap
naive_PB <- 0
```

```
### Set up the phat loop
for(i in c(1:rep)){

## Generate an AR(p) model with t observations and Gaussian innovations
# Y-sequence
set.seed(i)
epsilon <- rnorm(t+p,0,sigma)
Y <- filter(epsilon,filter = c(phi1,phi2),method = "recursive")[c((p+1):(t+p))]
# X-matrix
X     <- matrix(NA, t, 2)
X[,1] <- 1
X[,2] <- c(1:t)

## Estimate the model
AR2 <- Var2Reg(Y,X)

## Extract the mle model parameters and the filtered innovations
beta_mle  <- AR2[c(1,2)]
phi_mle   <- AR2[c(3,4)]
sigma_mle <- AR2[5]
res_filt  <- AR2[c(7:length(AR2))]

## Check whether "naïve" estimators are used
```

```
naive_glob <- naive_glob + AR2[6]/rep



### Nonparametric bootstrap
## Adjust the X-Matrix to the new length of the Y sequence
X      <- matrix(NA, nrow(NP_Boots), 2)
X[,1]  <- 1
X[,2]  <- c(1:nrow(NP_Boots))

## Generate the NB resamples of the residuals using the non-parametric approach
for(B in c(1:NB)){
  set.seed(((i-1)*rep)+B)
  NP_Boots_eps[,B] <- sample(res_filt, (t-p), replace = T)
}

## Generate NB Y series from the NB resamples
for(B in c(1:NB)){
  NP_Boots[,B] <- filter(NP_Boots_eps[,B], filter = c(phi_mle[1],phi_mle[2]),
                         method = "recursive")[c((p+1):(t-p))]
  NP_Boots[,B] <- NP_Boots[,B] + (X %*% as.matrix(beta_mle))
}

## Estimate the model parameters for all NB simulations and store them into the
## NP_Boots_params variable
for(B in c(1:NB)){
  ## Estimate the model coefficients by means of the Var2Reg function
  AR2_NPB <- Var2Reg(NP_Boots[,B],X)
  ## Extract the estimated parameters into variables
  # Regression coefficients beta
  NP_Boots_params[B,c(1,2)] <- AR2_NPB[c(1,2)]
  # AR parameters phi
  NP_Boots_params[B,c(3,4)] <- AR2_NPB[c(3,4)]
  # Sigma of innovations
  NP_Boots_params[B,5] <- AR2_NPB[5]
  ## Check whether "naïve" estimators are used
  naive_NPB <- naive_NPB + AR2_NPB[6]/(rep*NB)
}

## Calculate the CI for each parameter and store them into the NP_CI variable
# beta0
NP_CI[1,] <- as.numeric(quantile(NP_Boots_params[,1], probs = c(0.05,0.95)))
# beta1
NP_CI[2,] <- as.numeric(quantile(NP_Boots_params[,2], probs = c(0.05,0.95)))
# phi1
NP_CI[3,] <- as.numeric(quantile(NP_Boots_params[,3], probs = c(0.05,0.95)))
# phi2
NP_CI[4,] <- as.numeric(quantile(NP_Boots_params[,4], probs = c(0.05,0.95)))
# sigma
NP_CI[5,] <- as.numeric(quantile(NP_Boots_params[,5], probs = c(0.05,0.95)))

## Check whether the true model parameters fall within the constructed CIs
# beta0
NP_Coverage[i,1] <- ((beta0 > NP_CI[1,1]) & (beta0 < NP_CI[1,2]))
# beta1
NP_Coverage[i,2] <- ((beta1 > NP_CI[2,1]) & (beta1 < NP_CI[2,2]))
# phi1
```

```
NP_Coverage[i,3] <- ((phi1  > NP_CI[3,1]) & (phi1  < NP_CI[3,2]))
# phi2
NP_Coverage[i,4] <- ((phi2  > NP_CI[4,1]) & (phi2  < NP_CI[4,2]))
# sigma
NP_Coverage[i,5] <- ((sigma > NP_CI[5,1]) & (sigma < NP_CI[5,2]))



### Parametric bootstrap
## Adjust the X-Matrix to the new length of the Y sequence
X      <- matrix(NA, nrow(P_Boots), 2)
X[,1]  <- 1
X[,2]  <- c(1:nrow(P_Boots))

## Assuming Gaussian residuals, extract their mean and the sigma_mle
P_sigma <- sigma_mle
P_mu    <- mean(res_filt)

## Generate the NB resamples of the residuals using the parametric approach
for(B in c(1:NB)){
  set.seed(((i-1)*rep)+B)
  P_Boots_eps[,B] <- rnorm((t-p), P_mu, P_sigma)
}

## Generate NB Y series from the NB resamples
for(B in c(1:NB)){
  P_Boots[,B] <- filter(P_Boots_eps[,B], filter = c(phi_mle[1],phi_mle[2]),
                        method = "recursive")[c((p+1):(t-p))]
  P_Boots[,B] <- P_Boots[,B] + (X %*% as.matrix(beta_mle))
}

## Estimate the model parameters for all NB simulations and store them into the
## P_Boots_params variable
for(B in c(1:NB)){
  ## Estimate the model coefficients by means of the Var2Reg function
  AR2_PB <- Var2Reg(P_Boots[,B],X)
  ## Extract the estimated parameters into variables
  # Regression coefficients beta
  P_Boots_params[B,c(1,2)] <- AR2_PB[c(1,2)]
  # AR parameters phi
  P_Boots_params[B,c(3,4)] <- AR2_PB[c(3,4)]
  # Sigma of innovations
  P_Boots_params[B,5] <- AR2_PB[5]
  ## Check whether "naïve" estimators are used
  naive_PB <- naive_PB + AR2_PB[6]/(rep*NB)
}

## Calculate the CI for each parameter and store them into the P_CI variable
# beta0
P_CI[1,] <- as.numeric(quantile(P_Boots_params[,1], probs = c(0.05,0.95)))
# beta1
P_CI[2,] <- as.numeric(quantile(P_Boots_params[,2], probs = c(0.05,0.95)))
# phi1
P_CI[3,] <- as.numeric(quantile(P_Boots_params[,3], probs = c(0.05,0.95)))
# phi2
P_CI[4,] <- as.numeric(quantile(P_Boots_params[,4], probs = c(0.05,0.95)))
# sigma
```

```
P_CI[5,] <- as.numeric(quantile(P_Boots_params[,5], probs = c(0.05,0.95)))

## Check whether the true model parameters fall within the constructed CIs
# beta0
P_Coverage[i,1] <- ((beta0 > P_CI[1,1]) & (beta0 < P_CI[1,2]))
# beta1
P_Coverage[i,2] <- ((beta1 > P_CI[2,1]) & (beta1 < P_CI[2,2]))
# phi1
P_Coverage[i,3] <- ((phi1  > P_CI[3,1]) & (phi1  < P_CI[3,2]))
# phi2
P_Coverage[i,4] <- ((phi2  > P_CI[4,1]) & (phi2  < P_CI[4,2]))
# sigma
P_Coverage[i,5] <- ((sigma > P_CI[5,1]) & (sigma < P_CI[5,2]))
}
```

```
## Create a data frame to store the coverages of the nonparametric bootstrap
NP_Cov_Share <- matrix(NA, 5, 1)
NP_Cov_Share <- as.data.frame(NP_Cov_Share)
rownames(NP_Cov_Share) <- c("beta0","beta1","phi1","phi2","sigma")
colnames(NP_Cov_Share) <- "Coverage"

# Extract the coverages of the nonparametric bootstrap
for(i in c(1:5)){
  NP_Cov_Share[i,] <- mean(NP_Coverage[,i])
}

## Create a data frame to store the coverages of the parametric bootstrap
P_Cov_Share <- matrix(NA, 5, 1)
P_Cov_Share <- as.data.frame(P_Cov_Share)
rownames(P_Cov_Share) <- c("beta0","beta1","phi1","phi2","sigma")
colnames(P_Cov_Share) <- "Coverage"

# Extract the coverages of the parametric bootstrap
for(i in c(1:5)){
  P_Cov_Share[i,] <- mean(P_Coverage[,i])
}

## Present the resulting coverages in nice latex tables
# Nonparametric bootstrap
NP_Cov_Share_k <- kable(NP_Cov_Share, format = 'latex', digits = 2, booktabs = T,
                        linesep = "", align=c("c"))
NP_Cov_Share_k <- kable_styling(NP_Cov_Share_k, latex_options = c("striped"))
```

	Coverage
beta0	0.88
beta1	0.88
phi1	0.86
phi2	0.88
sigma	0.79

```
# Parametric bootstrap
P_Cov_Share_k <- kable(P_Cov_Share, format = 'latex', digits = 2, booktabs = T,
                       linesep = "", align=c("c"))
P_Cov_Share_k <- kable_styling(P_Cov_Share_k, latex_options = c("striped"))
```

	Coverage
beta0	0.88
beta1	0.88
phi1	0.87
phi2	0.89
sigma	0.82

```
## Take a look at the share of estimations which used the naïve estimators
# in the baseline model
naive_glob
```

```
[1] 0
```

```
# in the nonparametric bootstrap
naive_NPB
```

```
[1] 0
```

```
# in the parametric bootstrap
naive_PB
```

```
[1] 0
```

Task e)

Repeat the above, but replacing the Gaussian innovation sequence in the simulated data with Cauchy observations (and σ is now a scale parameter, so forget the idea that σ^2 is a variance). However, LISTEN CLOSELY: the parametric bootstrap still now (incorrectly) assumes GAUSSIAN! I would thus offhand guess that the nonparametric bootstrap will do much better than the parametric one. But, remember that least squares estimation involves squares of data, and squares of Cauchy do not have finite expectations! So, I have no idea what will happen, even for the nonparametric bootstrap.

We get the following coverages for the nonparametric and parametric bootstrap when we use Cauchy innovations with a scale parameter equal to $\sigma = 2$:

- Nonparametric Bootstrap: $\beta_0 : 0.817,\ \beta_1 : 0.92,\ \phi_1 : 0.88,\ \phi_2 : 0.92,\ \sigma : 0$
- Parametric Bootstrap: $\beta_0 : 0.857,\ \beta_1 : 0.91,\ \phi_1 : 0.917,\ \phi_2 : 0.937,\ \sigma : 0$

For the series that was simulated using the Cauchy distributed innovations, we see that the coverages for all model parameters except for σ are very close to the target value of 0.9. This is due to the fact that we are determining confidence intervals for a σ, which is assumed to be the standard deviation of Gaussian distributed innovations, whereas in fact we are producing confidence intervals for the variance of a t sequence of Cauchy distributed innovations using a Cauchy scale parameter equal to σ. As the variance of a Cauchy distributed sequence diverges to infinity as its length gets larger, it is somewhat logical that we almost always get too large estimates for the σ of an assumed Gaussian. Thus, this intuitively leads to a coverage of zero for σ in either bootstrap method.

```
## Define the number of repetitions we want to apply
rep <- 300

## Define the number of resamples (i.e. NB) we want want to generate
NB <- 400

## Store the true model parameters into variables
beta0 <- 0
beta1 <- 0
phi1  <- 0.5
phi2  <- -0.2
sigma <- 2
```

```
### Create all the needed variables for the nonparametric bootstrap
## Create a data frame to store the NB resamples of the residuals
NP_Boots_eps <- matrix(NA, t-p, NB)
NP_Boots_eps <- as.data.frame(NP_Boots_eps)

## Create a data frame to store the NB Y series
NP_Boots <- matrix(NA, t-2*p, NB)
NP_Boots <- as.data.frame(NP_Boots)

## Create a data frame to store all mle estimates
NP_Boots_params           <- matrix(NA, NB, 5)
NP_Boots_params           <- as.data.frame(NP_Boots_params)
colnames(NP_Boots_params) <- c("beta0_mle","beta1_mle","phi1_mle",
                               "phi2_mle","sigma_mle")

## Create a data frame to store the resulting confidence intervals
NP_CI           <- matrix(NA, 5, 2)
NP_CI           <- as.data.frame(NP_CI)
colnames(NP_CI) <- c("lower bound","upper bound")
rownames(NP_CI) <- c("NP_CI beta0","NP_CI beta1","NP_CI phi1",
                     "NP_CI phi2","NP_CI sigma")

## Create a data frame to store the boolean indicating whether the true
```

```r
## model parameter falls within its bootstrap-based CI
NP_Coverage <- matrix(NA, rep, 5)
NP_Coverage <- as.data.frame(NP_Coverage)
colnames(NP_Coverage) <- c("beta0","beta1","phi1","phi2","sigma")


### Create all the needed variables for the parametric bootstrap
## Create a data frame to store the NB resamples of the residuals
P_Boots_eps <- matrix(NA, t-p, NB)
P_Boots_eps <- as.data.frame(P_Boots_eps)

## Create a data frame to store the NB Y series
P_Boots <- matrix(NA, t-2*p, NB)
P_Boots <- as.data.frame(P_Boots)

## Create a data frame to store all mle estimates
P_Boots_params           <- matrix(NA, NB, 5)
P_Boots_params           <- as.data.frame(P_Boots_params)
colnames(P_Boots_params) <- c("beta0_mle","beta1_mle","phi1_mle",
                              "phi2_mle","sigma_mle")

## Create a data frame to store the resulting confidence intervals
P_CI           <- matrix(NA, 5, 2)
P_CI           <- as.data.frame(P_CI)
colnames(P_CI) <- c("lower bound","upper bound")
rownames(P_CI) <- c("P_CI beta0","P_CI beta1","P_CI phi1",
                    "P_CI phi2","P_CI sigma")

## Create a data frame to store the boolean indicating whether the true
## model parameter falls within its bootstrap-based CI
P_Coverage <- matrix(NA, rep, 5)
P_Coverage <- as.data.frame(P_Coverage)
colnames(P_Coverage) <- c("beta0","beta1","phi1","phi2","sigma")

## Create variables to keep track of the percentage where naïve estimation has
## been used
# in the baseline model
naive_glob <- 0
# in the nonparametric bootstrap
naive_NPB <- 0
# in the parametric bootstrap
naive_PB <- 0
```

```r
### Set up the phat loop
for(i in c(1:rep)){

## Generate an AR(p) model with t observations and Cauchy innovations
# Y-sequence
set.seed(i)
epsilon <- rcauchy(t+p,0,sigma)
Y <- filter(epsilon,filter = c(phi1,phi2),method = "recursive")[c((p+1):(t+p))]
# X-matrix
X     <- matrix(NA, t, 2)
X[,1] <- 1
X[,2] <- c(1:t)

## Estimate the model
AR2 <- Var2Reg(Y,X)
```

```
## Extract the mle model parameters and the filtered innovations
beta_mle  <- AR2[c(1,2)]
phi_mle   <- AR2[c(3,4)]
sigma_mle <- AR2[5]
res_filt  <- AR2[c(7:length(AR2))]

## Check whether "naïve" estimators are used
naive_glob <- naive_glob + AR2[6]/rep



### Nonparametric bootstrap
## Adjust the X-Matrix to the new length of the Y sequence
X      <- matrix(NA, nrow(NP_Boots), 2)
X[,1]  <- 1
X[,2]  <- c(1:nrow(NP_Boots))

## Generate the NB resamples of the residuals using the non-parametric approach
for(B in c(1:NB)){
  set.seed(((i-1)*rep)+B)
  NP_Boots_eps[,B] <- sample(res_filt, (t-p), replace = T)
}

## Generate NB Y series from the NB resamples
for(B in c(1:NB)){
  NP_Boots[,B] <- filter(NP_Boots_eps[,B], filter = c(phi_mle[1],phi_mle[2]),
                         method = "recursive")[c((p+1):(t-p))]
  NP_Boots[,B] <- NP_Boots[,B] + (X %*% as.matrix(beta_mle))
}

## Estimate the model parameters for all NB simulations and store them into the
## NP_Boots_params variable
for(B in c(1:NB)){
  ## Estimate the model coefficients by means of the Var2Reg function
  AR2_NPB <- Var2Reg(NP_Boots[,B],X)
  ## Extract the estimated parameters into variables
  # Regression coefficients beta
  NP_Boots_params[B,c(1,2)] <- AR2_NPB[c(1,2)]
  # AR parameters phi
  NP_Boots_params[B,c(3,4)] <- AR2_NPB[c(3,4)]
  # Sigma of innovations
  NP_Boots_params[B,5] <- AR2_NPB[5]
  ## Check whether "naïve" estimators are used
  naive_NPB <- naive_NPB + AR2_NPB[6]/(rep*NB)
}

## Calculate the CI for each parameter and store them into the NP_CI variable
# beta0
NP_CI[1,] <- as.numeric(quantile(NP_Boots_params[,1], probs = c(0.05,0.95)))
# beta1
NP_CI[2,] <- as.numeric(quantile(NP_Boots_params[,2], probs = c(0.05,0.95)))
# phi1
NP_CI[3,] <- as.numeric(quantile(NP_Boots_params[,3], probs = c(0.05,0.95)))
# phi2
NP_CI[4,] <- as.numeric(quantile(NP_Boots_params[,4], probs = c(0.05,0.95)))
# sigma
NP_CI[5,] <- as.numeric(quantile(NP_Boots_params[,5], probs = c(0.05,0.95)))
```

```r
## Check whether the true model parameters fall within the constructed CIs
# beta0
NP_Coverage[i,1] <- ((beta0 > NP_CI[1,1]) & (beta0 < NP_CI[1,2]))
# beta1
NP_Coverage[i,2] <- ((beta1 > NP_CI[2,1]) & (beta1 < NP_CI[2,2]))
# phi1
NP_Coverage[i,3] <- ((phi1  > NP_CI[3,1]) & (phi1  < NP_CI[3,2]))
# phi2
NP_Coverage[i,4] <- ((phi2  > NP_CI[4,1]) & (phi2  < NP_CI[4,2]))
# sigma
NP_Coverage[i,5] <- ((sigma > NP_CI[5,1]) & (sigma < NP_CI[5,2]))



### Parametric bootstrap
## Adjust the X-Matrix to the new length of the Y sequence
X      <- matrix(NA, nrow(P_Boots), 2)
X[,1]  <- 1
X[,2]  <- c(1:nrow(P_Boots))

## Assuming Gaussian residuals, extract their mean and the sigma_mle
P_sigma <- sigma_mle
P_mu    <- mean(res_filt)

## Generate the NB resamples of the residuals using the parametric approach
for(B in c(1:NB)){
  set.seed(((i-1)*rep)+B)
  P_Boots_eps[,B] <- rnorm((t-p), P_mu, P_sigma)
}

## Generate NB Y series from the NB resamples
for(B in c(1:NB)){
  P_Boots[,B] <- filter(P_Boots_eps[,B], filter = c(phi_mle[1],phi_mle[2]),
                        method = "recursive")[c((p+1):(t-p))]
  P_Boots[,B] <- P_Boots[,B] + (X %*% as.matrix(beta_mle))
}

## Estimate the model parameters for all NB simulations and store them into the
## P_Boots_params variable
for(B in c(1:NB)){
  ## Estimate the model coefficients by means of the Var2Reg function
  AR2_PB <- Var2Reg(P_Boots[,B],X)
  ## Extract the estimated parameters into variables
  # Regression coefficients beta
  P_Boots_params[B,c(1,2)] <- AR2_PB[c(1,2)]
  # AR parameters phi
  P_Boots_params[B,c(3,4)] <- AR2_PB[c(3,4)]
  # Sigma of innovations
  P_Boots_params[B,5] <- AR2_PB[5]
  ## Check whether "naïve" estimators are used
  naive_PB <- naive_PB + AR2_PB[6]/(rep*NB)
}

## Calculate the CI for each parameter and store them into the P_CI variable
# beta0
P_CI[1,] <- as.numeric(quantile(P_Boots_params[,1], probs = c(0.05,0.95)))
# beta1
```

```r
P_CI[2,] <- as.numeric(quantile(P_Boots_params[,2], probs = c(0.05,0.95)))
# phi1
P_CI[3,] <- as.numeric(quantile(P_Boots_params[,3], probs = c(0.05,0.95)))
# phi2
P_CI[4,] <- as.numeric(quantile(P_Boots_params[,4], probs = c(0.05,0.95)))
# sigma
P_CI[5,] <- as.numeric(quantile(P_Boots_params[,5], probs = c(0.05,0.95)))

## Check whether the true model parameters fall within the constructed CIs
# beta0
P_Coverage[i,1] <- ((beta0 > P_CI[1,1]) & (beta0 < P_CI[1,2]))
# beta1
P_Coverage[i,2] <- ((beta1 > P_CI[2,1]) & (beta1 < P_CI[2,2]))
# phi1
P_Coverage[i,3] <- ((phi1  > P_CI[3,1]) & (phi1  < P_CI[3,2]))
# phi2
P_Coverage[i,4] <- ((phi2  > P_CI[4,1]) & (phi2  < P_CI[4,2]))
# sigma
P_Coverage[i,5] <- ((sigma > P_CI[5,1]) & (sigma < P_CI[5,2]))
}
```

```r
## Create a data frame to store the coverages of the nonparametric bootstrap
NP_Cov_Share <- matrix(NA, 5, 1)
NP_Cov_Share <- as.data.frame(NP_Cov_Share)
rownames(NP_Cov_Share) <- c("beta0","beta1","phi1","phi2","sigma")
colnames(NP_Cov_Share) <- "Coverage"

# Extract the coverages of the nonparametric bootstrap
for(i in c(1:5)){
  NP_Cov_Share[i,] <- mean(NP_Coverage[,i])
}

## Create a data frame to store the coverages of the parametric bootstrap
P_Cov_Share <- matrix(NA, 5, 1)
P_Cov_Share <- as.data.frame(P_Cov_Share)
rownames(P_Cov_Share) <- c("beta0","beta1","phi1","phi2","sigma")
colnames(P_Cov_Share) <- "Coverage"

# Extract the coverages of the parametric bootstrap
for(i in c(1:5)){
  P_Cov_Share[i,] <- mean(P_Coverage[,i])
}

## Present the resulting coverages in nice latex tables
# Nonparametric bootstrap
NP_Cov_Share_k <- kable(NP_Cov_Share, format = 'latex', digits = 2, booktabs = T,
                        linesep = "", align=c("c"))
NP_Cov_Share_k <- kable_styling(NP_Cov_Share_k, latex_options = c("striped"))
```

	Coverage
beta0	0.82
beta1	0.92
phi1	0.88
phi2	0.92
sigma	0.00

```
# Parametric bootstrap
P_Cov_Share_k <- kable(P_Cov_Share, format = 'latex', digits = 2, booktabs = T,
                       linesep = "", align=c("c"))
P_Cov_Share_k <- kable_styling(P_Cov_Share_k, latex_options = c("striped"))
```

	Coverage
beta0	0.86
beta1	0.91
phi1	0.92
phi2	0.94
sigma	0.00

```
## Take a look at the share of estimations which used the naïve estimators
# in the baseline model
naive_glob
```

```
[1] 0
```

```
# in the nonparametric bootstrap
naive_NPB
```

```
[1] 0.0005416667
```

```
# in the parametric bootstrap
naive_PB
```

```
[1] 0.0002916667
```