Time Series Final Exam

Alex Salo

December 9, 2015

1. Investigating Estimators of the Coefficients of an AR(p) process

Let us simulate 500 realizations of AR(3) process $X_t - 0.3X_{t-1} + 0.8X_{t-3} = \varepsilon_t$ and try to estimate the coefficients using four different methods while using 50, 100, 250 or 500 of the generated realizations:

```
rvar = 1                 # variance of white noise
nreps = 500              # number of generated realizations of the process in simulation
n = c(50, 100, 250, 500) # number of realizations actually used in simulation
alpha = c(-0.3, 0, 0.8)  # coefficients of AR(3) process
# arsim = ar_est(alpha=alpha, rvar=rvar, nreps=nreps, seed=0)
arsim = readRDS("arsim.rds")
```

A. Numerical Description of the estimates

Let us make a summary statistics for each n for each coefficient for each estimate using summary function in R:

```
estimate_names = vector()
AR_est_names = c("YW", "OLS", "MLE", "Burg")
for (name in AR_est_names)
  for (num in n)
    estimate_names = c(estimate_names, paste(name, num, sep='_'))

sum_names = names(summary(arsim$yw.est[1,4,]))
summary_table = array(0, dim = c(length(estimate_names), 6, length(alpha)),
                      dimnames=list(estimate_names,       # estimate method
                                    sum_names,            # summary headers
                                    c('a1', 'a2', 'a3'))) # coefficients
for (i in 1:length(n))
  for (name in AR_est_names)
    for (coef in 1:length(alpha)) {
      est_name = paste(name, n[i], sep='_')               # ~YW_50
      arsim_name = paste(tolower(name),'est', sep='.')    # ~arsim$yw.est
      summary_table[est_name, ,coef] = unname(summary(arsim[[arsim_name]][coef,i,]))
    }
summary_table
```

```
## , , a1
##
##             Min. 1st Qu.  Median    Mean 3rd Qu.     Max.
## YW_50    -0.7889 -0.4171 -0.3349 -0.3474 -0.2713  0.02030
## YW_100   -0.5751 -0.3744 -0.3168 -0.3271 -0.2742 -0.14320
## YW_250   -0.4459 -0.3394 -0.3111 -0.3120 -0.2817 -0.19090
## YW_500   -0.3972 -0.3269 -0.3025 -0.3052 -0.2853 -0.22130
## OLS_50   -0.5811 -0.3499 -0.2899 -0.2949 -0.2372  0.02761
```

```
## OLS_100  -0.5298 -0.3365 -0.2928 -0.2983 -0.2582 -0.15040
## OLS_250  -0.4432 -0.3226 -0.2974 -0.2995 -0.2736 -0.18710
## OLS_500  -0.3830 -0.3176 -0.2958 -0.2988 -0.2798 -0.22050
## MLE_50   -0.6096 -0.3464 -0.2967 -0.2976 -0.2384  0.02550
## MLE_100  -0.5310 -0.3339 -0.2932 -0.2988 -0.2566 -0.15170
## MLE_250  -0.4400 -0.3239 -0.2984 -0.2995 -0.2726 -0.18940
## MLE_500  -0.3831 -0.3181 -0.2963 -0.2988 -0.2794 -0.22020
## Burg_50  -0.6141 -0.3558 -0.2944 -0.3018 -0.2402  0.01305
## Burg_100 -0.5466 -0.3353 -0.2928 -0.2992 -0.2562 -0.14170
## Burg_250 -0.4405 -0.3239 -0.2974 -0.2996 -0.2722 -0.19110
## Burg_500 -0.3867 -0.3172 -0.2967 -0.2989 -0.2790 -0.22110
##
## , , a2
##
##              Min.   1st Qu.       Median      Mean 3rd Qu.   Max.
## YW_50    -0.23680 -0.008402  0.06402000 0.0757100 0.15490 0.5711
## YW_100   -0.16220 -0.014310  0.03415000 0.0428100 0.08949 0.3258
## YW_250   -0.11340 -0.014800  0.01589000 0.0178000 0.04870 0.1694
## YW_500   -0.06202 -0.015770  0.00706600 0.0088240 0.03105 0.1072
## OLS_50   -0.24740 -0.057520  0.00103100 0.0083820 0.07085 0.3608
## OLS_100  -0.19900 -0.041040  0.00258800 0.0060040 0.04486 0.2330
## OLS_250  -0.11500 -0.026960  0.00089350 0.0018830 0.02933 0.1648
## OLS_500  -0.07176 -0.020390 -0.00127500 0.0008779 0.02151 0.1067
## MLE_50   -0.24320 -0.051590 -0.00156000 0.0104800 0.07152 0.4400
## MLE_100  -0.19560 -0.038680  0.00194700 0.0062470 0.04646 0.2349
## MLE_250  -0.12100 -0.026370  0.00008209 0.0019040 0.02954 0.1605
## MLE_500  -0.07035 -0.020940 -0.00059780 0.0008814 0.02036 0.1052
## Burg_50  -0.25060 -0.051690  0.00409200 0.0143200 0.07360 0.4253
## Burg_100 -0.16980 -0.040320  0.00337800 0.0067270 0.04678 0.2431
## Burg_250 -0.12000 -0.026070  0.00091550 0.0020280 0.02817 0.1619
## Burg_500 -0.07424 -0.020670 -0.00072400 0.0008835 0.02081 0.1054
##
## , , a3
##
##            Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## YW_50    0.3198  0.6075 0.6725 0.6652  0.7322 0.9120
## YW_100   0.4852  0.6907 0.7309 0.7255  0.7717 0.8731
## YW_250   0.6305  0.7440 0.7698 0.7686  0.7972 0.8714
## YW_500   0.6906  0.7653 0.7842 0.7839  0.8052 0.8495
## OLS_50   0.4000  0.7027 0.7712 0.7622  0.8329 0.9593
## OLS_100  0.5296  0.7373 0.7804 0.7772  0.8222 0.9164
## OLS_250  0.6344  0.7684 0.7930 0.7903  0.8164 0.8839
## OLS_500  0.7022  0.7763 0.7953 0.7946  0.8133 0.8608
## MLE_50   0.3908  0.7138 0.7750 0.7643  0.8312 0.9259
## MLE_100  0.5295  0.7440 0.7832 0.7784  0.8216 0.9000
## MLE_250  0.6287  0.7684 0.7944 0.7905  0.8158 0.8781
## MLE_500  0.7017  0.7768 0.7953 0.7946  0.8136 0.8633
## Burg_50  0.3876  0.7050 0.7647 0.7552  0.8206 0.9239
## Burg_100 0.5356  0.7415 0.7788 0.7758  0.8218 0.8953
## Burg_250 0.6336  0.7677 0.7929 0.7898  0.8152 0.8831
## Burg_500 0.7039  0.7768 0.7955 0.7944  0.8129 0.8605
```

In fact, the desirable information is in 4d array, but since humans can only perceive 2d tables, we condense one dimension (increase of n) into additional rows (while marking as est_n), and print three tables for each

coefficient separately since we don't really need to compare the stats between the coefficients.

It appears that each estimator makes a very reasonable estimates of the $\alpha_1, \alpha_2, \alpha_3$; also there seem to be no significant difference between the estimators, except for α_2, where Yuler-Walker estimate significantly under-performs other three estimates. For example, in case of $n == 500$ we see that:

1. Yuler-Walker gives a mean and median estimates that are worse than MLE and Burg by the entire order (time ten, or extra zero in accuracy).
2. OLS seems to be twice worse than MLE and Burg.

That observation seems to be present only in case of $\alpha_2 = 0$, which suggest that YW maybe has some problems with this extreme value.

B. Histograms

Let us make a function to produce the histograms of α_a estimations for increasing number of considered realizations for a given a and estimation method:

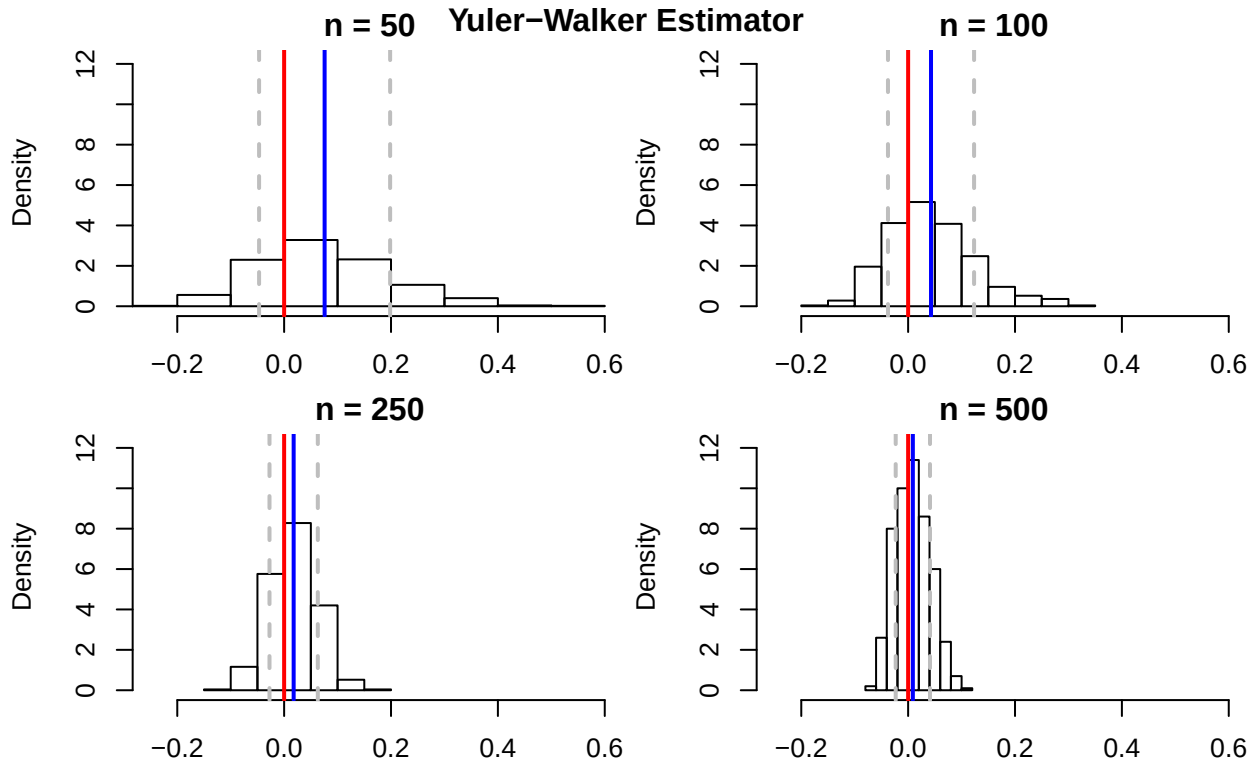
```
plot_hists = function(a, xlim, ylim, main, est_data) {
  par(mfrow=c(2,2), mar = c(2,4,1.5,0.5) + 0.1)
  for (num in 1:length(n)) {
    data = est_data[a,num,]
    tmp = hist(data, probability = TRUE, main=paste("n =", n[num]),
         xlim=xlim, ylim=ylim,
         xlab = expression(paste("Estimation of ", alpha)))
    abline(v=alpha[a], col='Red', lwd=2)
    data_mean = mean(data); data_std = sd(data)
    abline(v=data_mean, col='Blue', lwd=2)
    abline(v=c(data_mean-data_std, data_mean+data_std), col='Grey', lwd=2, lty=2)
  }
  title(paste(main, "Estimator"), line = -1, outer = TRUE)
}
```

Now we can see what each of the four different estimation methods produce for $alpha_2$. Note the following about the plots:

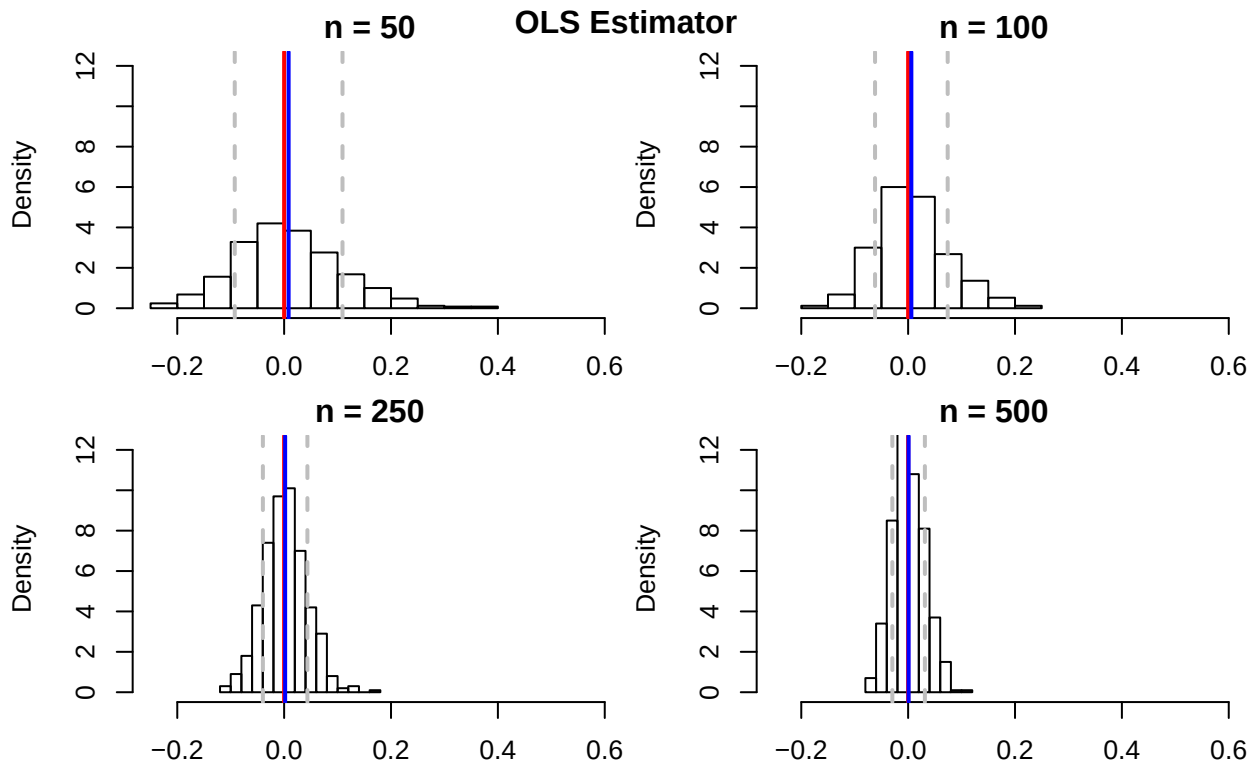
1. Red line shows true value of α_2, which is 0.
2. Blue line shows the mean value of $\hat{\alpha}_a$ across all replications.
3. Grey lines shows $\pm$ one standard error of $\hat{\alpha}_a$

```
# coef alpha_2
a = 2

# min\max accross all estimators and all realizations for given coef
xlim = c(min(summary_table[,,a]), max(summary_table[,,a]))

# from zero to the max height of the best estimator at max n
ylim = c(0, max(hist(arsim$mle.est[a,4,], plot=FALSE)$density))
```

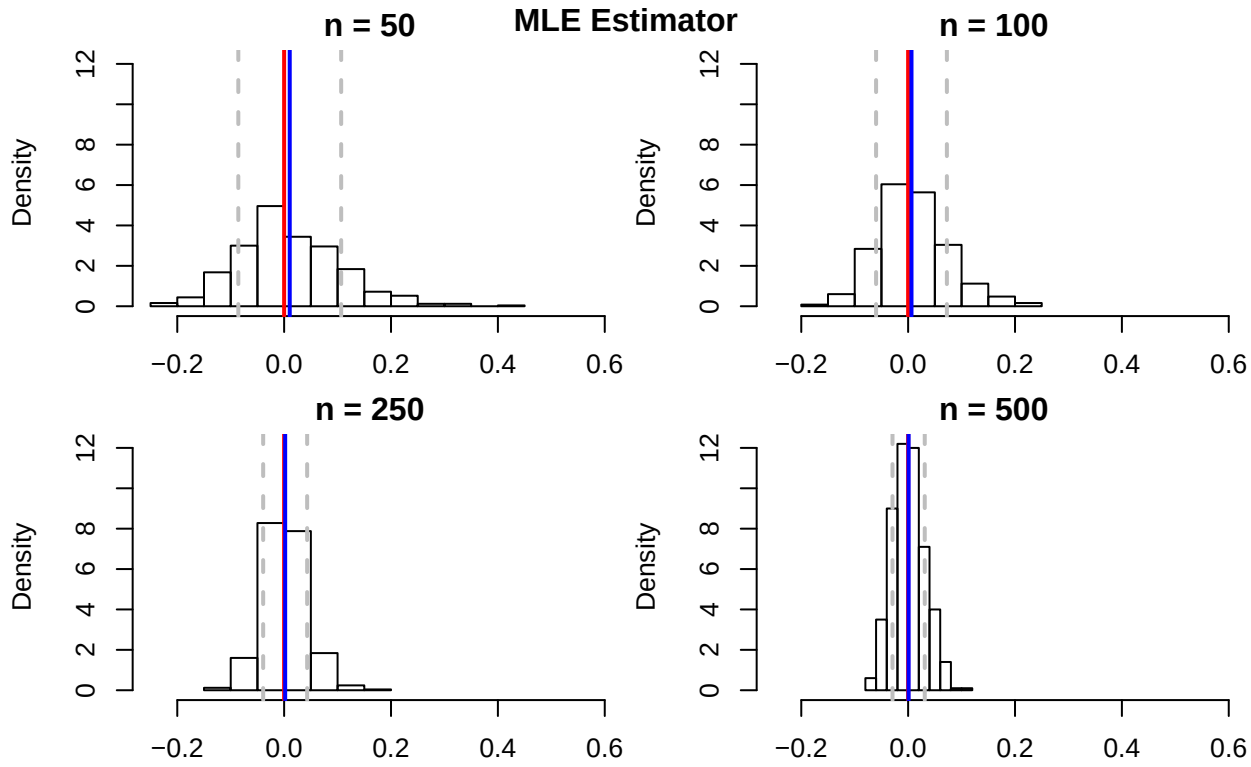
```
plot_hists(a, xlim, ylim, main="Yuler-Walker", est_data=arsim$yw.est)
```



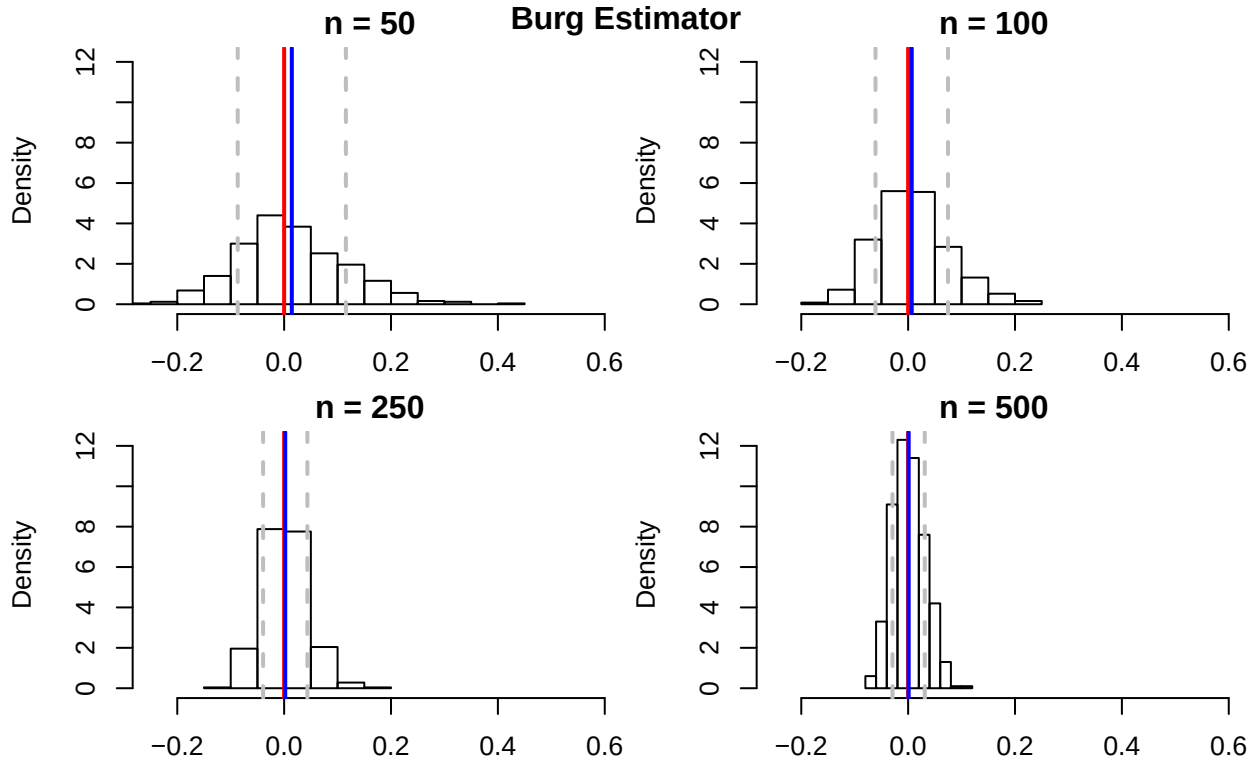

```
plot_hists(a, xlim, ylim, main="OLS", est_data=arsim$ols.est)
```



```
plot_hists(a, xlim, ylim, main="MLE", est_data=arsim$mle.est)
```




```
plot_hists(a, xlim, ylim, main="Burg", est_data=arsim$burg.est)
```



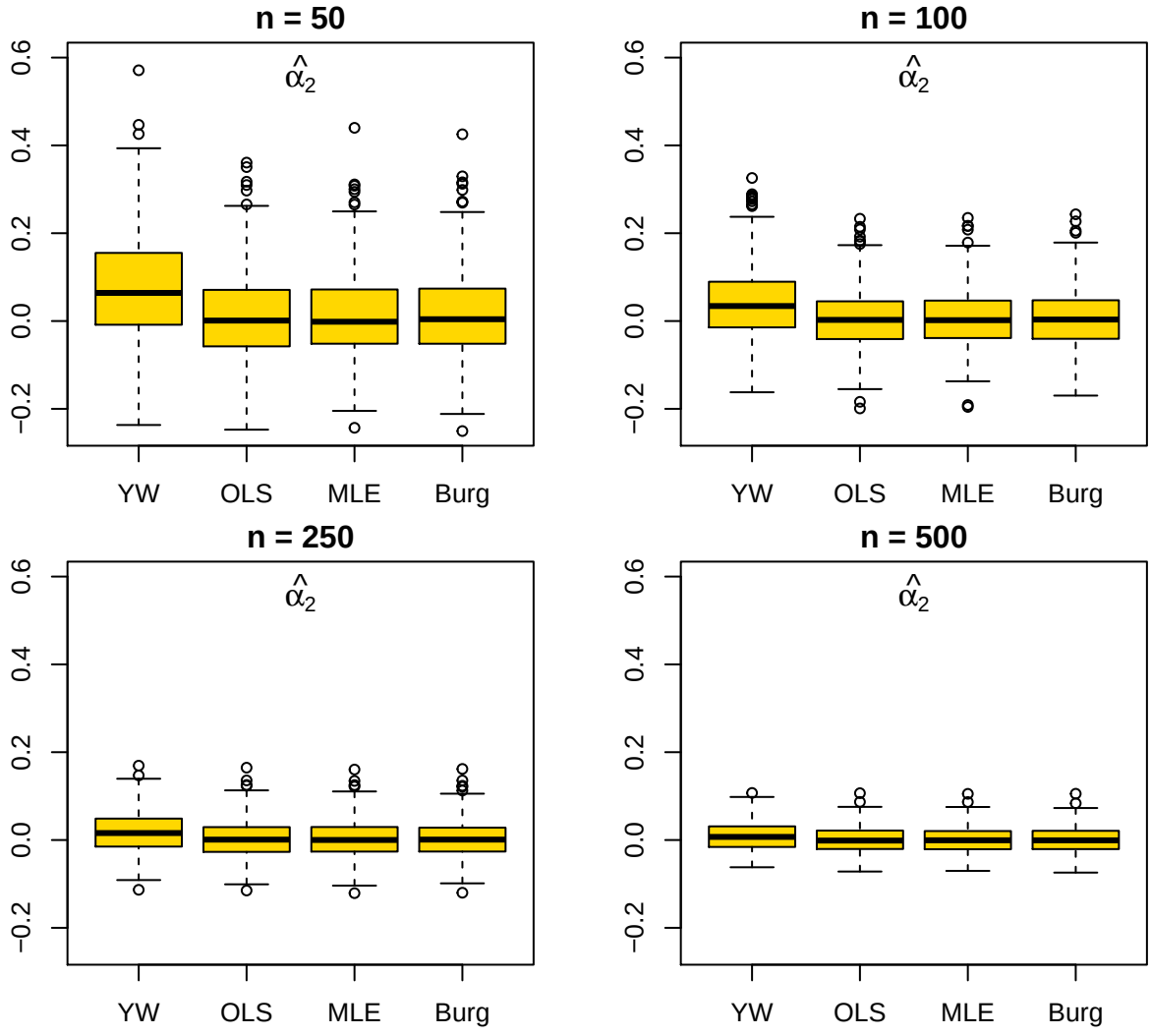
C. Boxplots

Let us also make a function to plot boxplots for each estimator and increasing number of considered realizations for a given a:

```
plot_boxplots = function(a) {
  par(mfrow=c(2,2), mar = c(2,4,1.5,0.5) + 0.1)
  for (num in 1:length(n)) {
    boxplot(arsim$yw.est[a,num,], arsim$ols.est[a,num,],
            arsim$mle.est[a,num,], arsim$burg.est[a,num,],
            names=AR_est_names, col="gold",
            main=paste("n =", n[num]), pars=list(ylim=c(-0.25,0.6)))
    title(bquote(hat(alpha[.(a)])), line=-1)
  }
}
```

Which for $\hat{\alpha}_2$ gives:

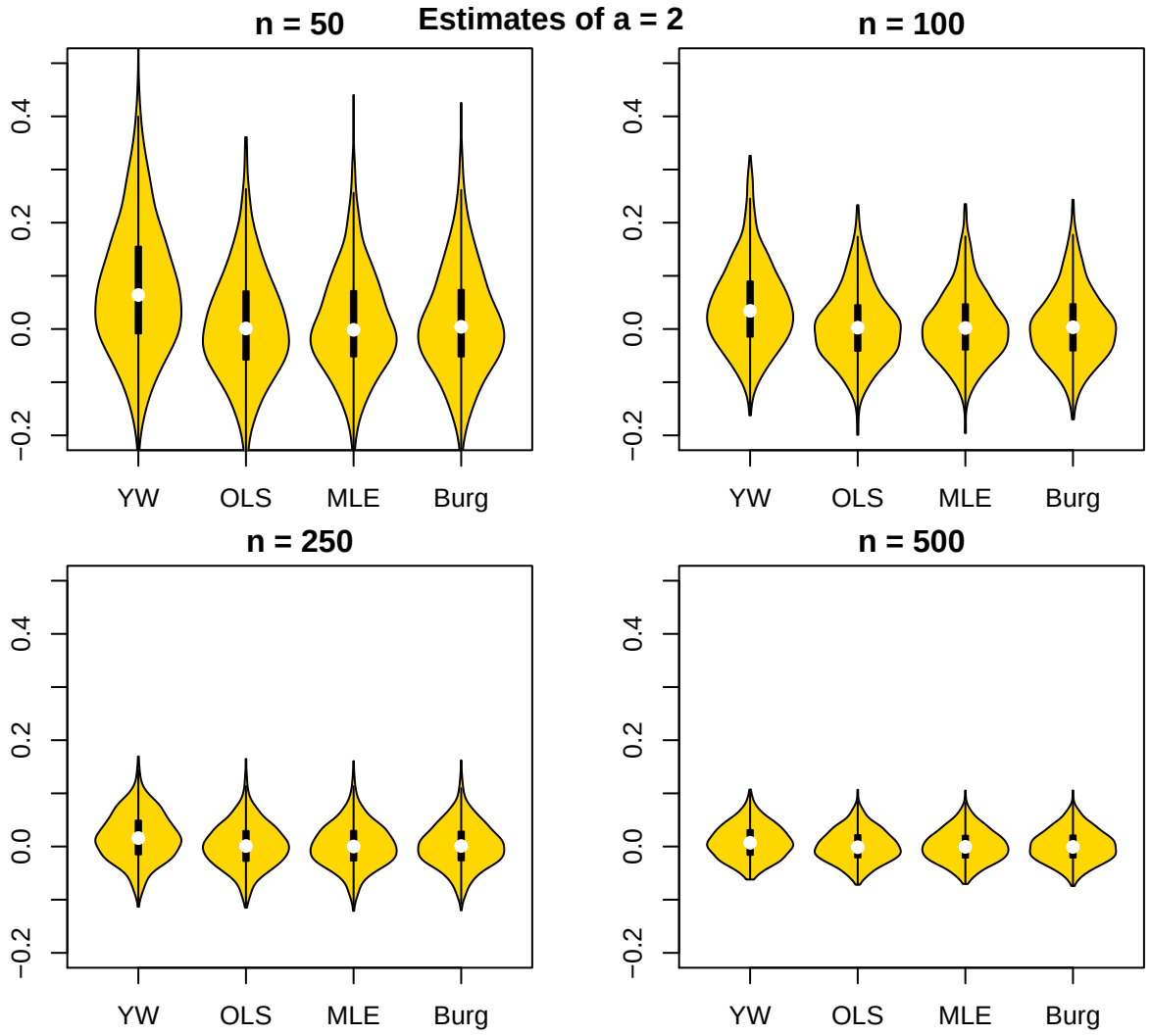
```
plot_boxplots(a=2)
```



D. Another Graphical Summaries

Consider Violin plots which is just combination of a boxplot and kernel density plot.

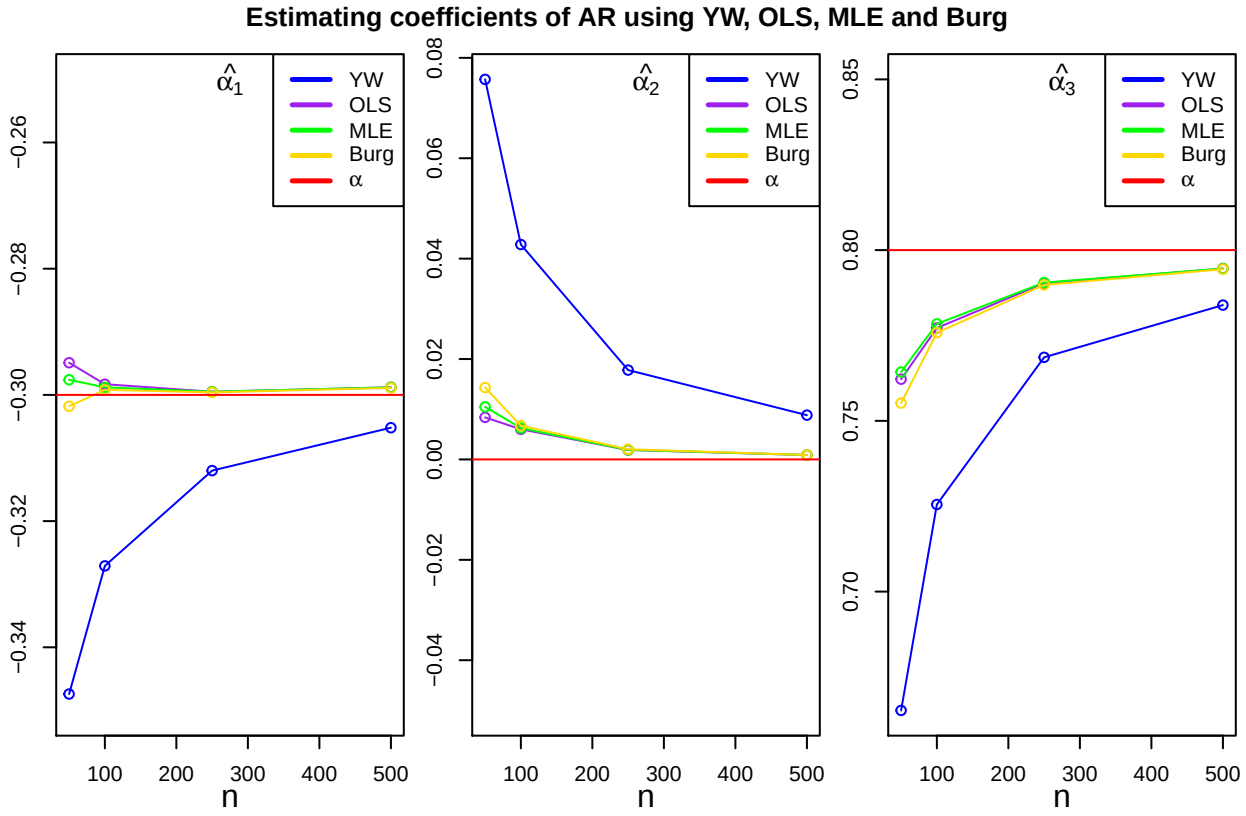
```
# library(vioplot)
plot_violins = function(a) {
  par(mfrow=c(2,2), mar = c(2,4,1.5,0.5) + 0.1)
  for (num in 1:length(n)) {
    vioplot(arsim$yw.est[a,num,], arsim$ols.est[a,num,],
            arsim$mle.est[a,num,], arsim$burg.est[a,num,],
            names=c("YW", "OLS", "MLE", "Burg"), col="gold",
            ylim=c(-0.2,0.5))
    title(paste("n =", n[num]))
  }
  title(main=paste("Estimates of a =", a), line = -1, cex=5, outer = TRUE)
}
plot_violins(a=2)
```



On this plots we can see the information from both previous methods (histograms and boxplots), to some extent, while using only half of the space.

Another thing we can do is to extend the summary table that we calculated. It feels awkward to try manually look into the table for various anomalies and discrepancies; to streamline the process we can plot the mean of the estimates of the coefficients as a function of the number of realizations considered:

```
plot_converge_estimators()
```



Here we can make a several additional observations:

1. Yuler-Walker estimator seemingly significantly underperforms other three estimators.
2. Those other three (OLS, MLE and Burg) produce very similar results.
3. Ultimately every estimator seem to be converging to the true value.
4. YW, OLS and MLE, as we observe in our small sample, only approach the true value from one side and do not oscillate back and forth around the true value. If that observation is a true fact, it would be a significant one since we could put tight **guarantees** on the true values.

Below is the code listing to produce the plots above:

```
# plot convergence of estimates for one alpha
plot_est_converge = function(a) {
  coef = paste('a', a, sep='')
  ylim = c(min(alpha[a] - 0.05, min(summary_table[, 'Mean', coef])),
           max(alpha[a] + 0.05, max(summary_table[, 'Mean', coef])))
  plot(n, summary_table[1:4, 'Mean', coef], type='o', col="Blue", ylim=ylim, xlab="")
  lines(n, summary_table[5:8, 'Mean', coef], type='o', col="Purple")
```

```
  lines(n, summary_table[9:12, 'Mean', coef], type='o', col="Green")
  lines(n, summary_table[13:16, 'Mean', coef], type='o', col="Gold")
  abline(h=alpha[a], col="Red")
  mtext(text="n", line=2, side = 1)
  title(bquote(hat(alpha[.(a)])), line=-1)
  legend('topright', legend=c(AR_est_names, expression(alpha)), lty=c(1,1,1),
         lwd=c(2.5,2.5,2.5),col=c("Blue","Purple","Green", "Gold", "red"))
}
# make plots for each alpha
plot_converge_estimators = function() {
  par(mfrow=c(1,3), mar = c(4,2,3,0.5) + 0.1)
  for (a in 1:length(alpha))
    plot_est_converge(a)
  title('Estimating coefficients of AR using YW, OLS, MLE and Burg', line=-2, outer=TRUE)
}
```

Summary of the Analysis

1. Sampling distributions of the estimators appear to have a bell-shaped and look "normal". Tails appear to shrink with the increase of n (number of realizations considered) while the density around the mean values increases making the estimate more rigid (accurate) with less deviation.

2. Bias in estimators

   (a) Yuler-Walker estimator appears to be biased since the estimation (with the small n) is far from the true value. However, increasing series length appear to shrink the bias and at $n = 500$ the estimation is very close to the true value. On the boxplots, YW estimator stays above the other three at $n = 50$, but then eventually comes close (at the same level) with them.

   (b) OLS, MLE and Burg estimators appear to be unbiased (even with small $n = 50$). These estimators too, however, decrease their bias (distance to the true value of the parameter α_2) with the increase of n, which is nice feature of estimators to have.

3. For every estimator, increasing n reduces the standard error - i.e. the estimation becomes more precise when we see more realizations of the process.

4. To summarize, estimators of AR process do a good job in estimating the true parameters of the process based on its observed realizations. The realizations we observe - the better the estimations are, which is very desirable. YW appears to be biased, while MLE, OLS and Burg to be not. With the increase of n, however, they all seem to converge to the true parameters' values. Standard error shrinks with the increase of n too. Thus all the evidence suggest that the more realizations we observe, the better we can describe the AR process.

5. I would probably choose MLE or Burn estimators for a real application, since they appear to be less biased and give better and consistent results even with smaller n.

2. Investigating Estimators of the coefficients of an MA(q) Process

Let us simulate 500 realizations of MA(3) process $X_t = \varepsilon_t - 0.3X_{t-1} + 0.8X_{t-3}$ and try to estimate the coefficients using either Maximum Likelihood (MLE) or Conditional Sums of Squares (CSS) estimator, while using 50, 100, 250 or 500 of the generated realizations:

```
rvar = 1                  # variance of white noise
nreps = 500               # number of generated realizations of the process in simulation
n = c(50, 100, 250, 500) # number of realizations actually used in simulation
beta = c(-0.3, 0, 0.8)    # coefficients of MA(3) process
# masim = arma_est(alpha=0, beta=beta, rvar=rvar, nreps=nreps, seed=0)
masim = readRDS("masim.rds")
```

A. Numerical Description of the estimates

Let us make a summary statistics for each n for each coefficient for each estimate using summary function in R:

```
estimate_names = vector()
MA_est_names = c("MLE", "CSS")
for (name in MA_est_names)
  for (num in n)
    estimate_names = c(estimate_names, paste(name, num, sep='_'))

sum_names = names(summary(masim$mle.est[1,4,]))
summary_table = array(0, dim = c(length(estimate_names), 6, length(alpha)),
                      dimnames=list(estimate_names,       # estimate method
                                    sum_names,            # summary headers
                                    c('a1', 'a2', 'a3'))) # coefficients
for (i in 1:length(n))
  for (name in MA_est_names)
    for (coef in 1:length(alpha)) {
      est_name = paste(name, n[i], sep='_')               # ~CSS_50
      masim_name = paste(tolower(name),'est', sep='.')    # ~masim$css.est
      summary_table[est_name, ,coef] = unname(summary(masim[[masim_name]][coef,i,]))
    }
summary_table
```

```
## , , a1
##
##             Min. 1st Qu.  Median    Mean 3rd Qu.     Max.
## MLE_50  -0.7734 -0.3659 -0.2895 -0.2887 -0.2036  0.25690
## MLE_100 -0.6055 -0.3515 -0.2952 -0.2972 -0.2479  0.01117
## MLE_250 -0.4578 -0.3307 -0.3041 -0.3019 -0.2728 -0.14090
## MLE_500 -0.4003 -0.3225 -0.3023 -0.3024 -0.2848 -0.18860
## CSS_50  -0.6736 -0.3150 -0.2280 -0.2307 -0.1466  0.20510
## CSS_100 -0.5648 -0.3054 -0.2490 -0.2488 -0.1928  0.05665
## CSS_250 -0.4593 -0.3029 -0.2730 -0.2713 -0.2403 -0.07952
## CSS_500 -0.3909 -0.3055 -0.2839 -0.2842 -0.2651 -0.17600
##
## , , a2
##
```

```
##             Min.  1st Qu.     Median      Mean  3rd Qu.    Max.
## MLE_50  -0.6440 -0.12690 -0.0215300 -0.020000 0.078370 0.47850
## MLE_100 -0.3685 -0.05943 -0.0003142 -0.003254 0.053730 0.34520
## MLE_250 -0.1440 -0.03042  0.0042400  0.001260 0.033720 0.20560
## MLE_500 -0.1468 -0.01765  0.0014730  0.001778 0.020770 0.13100
## CSS_50  -0.5248 -0.15040 -0.0650200 -0.056640 0.039160 0.46520
## CSS_100 -0.3366 -0.09611 -0.0312700 -0.036240 0.021640 0.33610
## CSS_250 -0.1580 -0.05507 -0.0202100 -0.021700 0.009522 0.17960
## CSS_500 -0.1545 -0.03242 -0.0113500 -0.012200 0.005450 0.08091
##
## , , a3
##
##             Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## MLE_50   0.3361  0.7180 0.8137 0.7980  0.8992 1.0000
## MLE_100  0.4954  0.7584 0.8222 0.8134  0.8726 1.0000
## MLE_250  0.6563  0.7800 0.8166 0.8136  0.8467 0.9307
## MLE_500  0.7215  0.7843 0.8039 0.8052  0.8272 0.9456
## CSS_50  -0.0900  0.5870 0.6992 0.6887  0.7930 1.1180
## CSS_100  0.4205  0.6604 0.7342 0.7294  0.7995 0.9825
## CSS_250  0.5696  0.7259 0.7683 0.7629  0.8006 0.9145
## CSS_500  0.6223  0.7527 0.7796 0.7757  0.8016 0.9285
```

Looking at the summary stats table we can observe that while both methods estimate the true value of the coefficients well, on the $\alpha_2 = 0, n = 500$ MLE outperforms CSS by the factor of ten in how close it is to the true value; this factors widens with the increase of n.

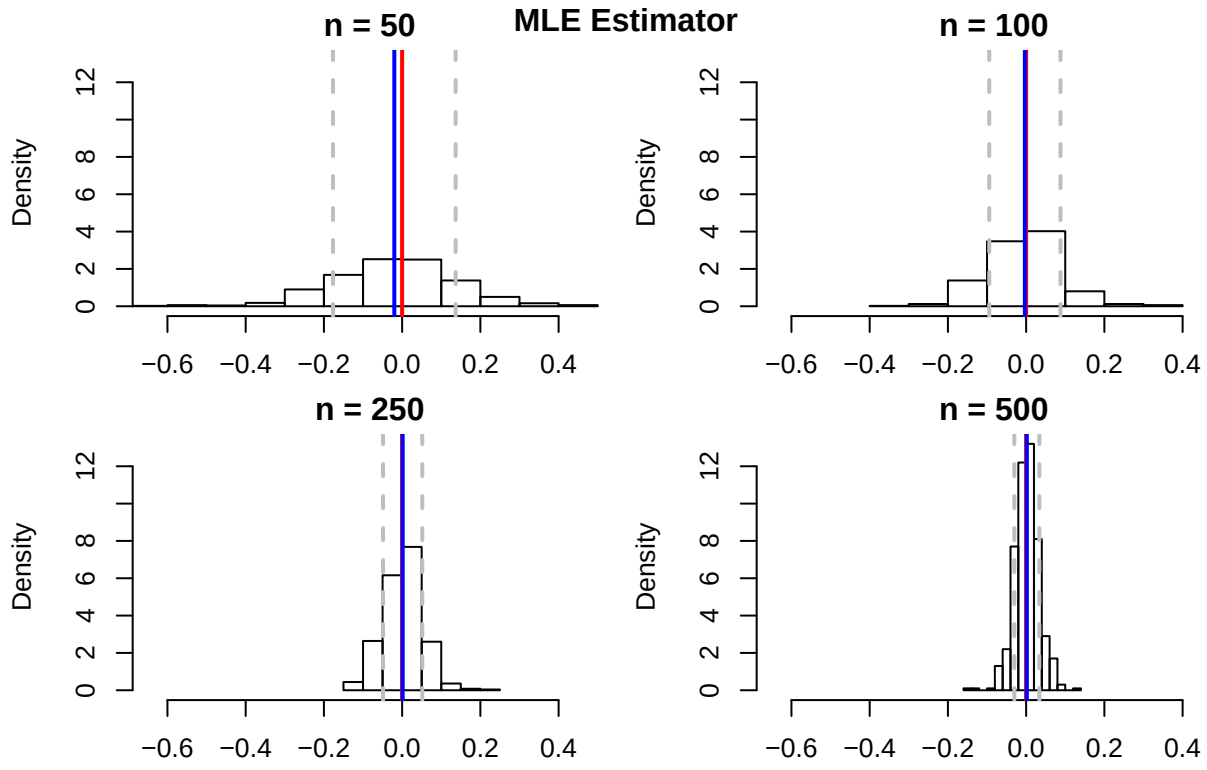
B. Histograms

Using the plot function from the AR section, we produce the histograms of β_a estimations for increasing number of considered realizations for a given a and estimation method. We observe the following:

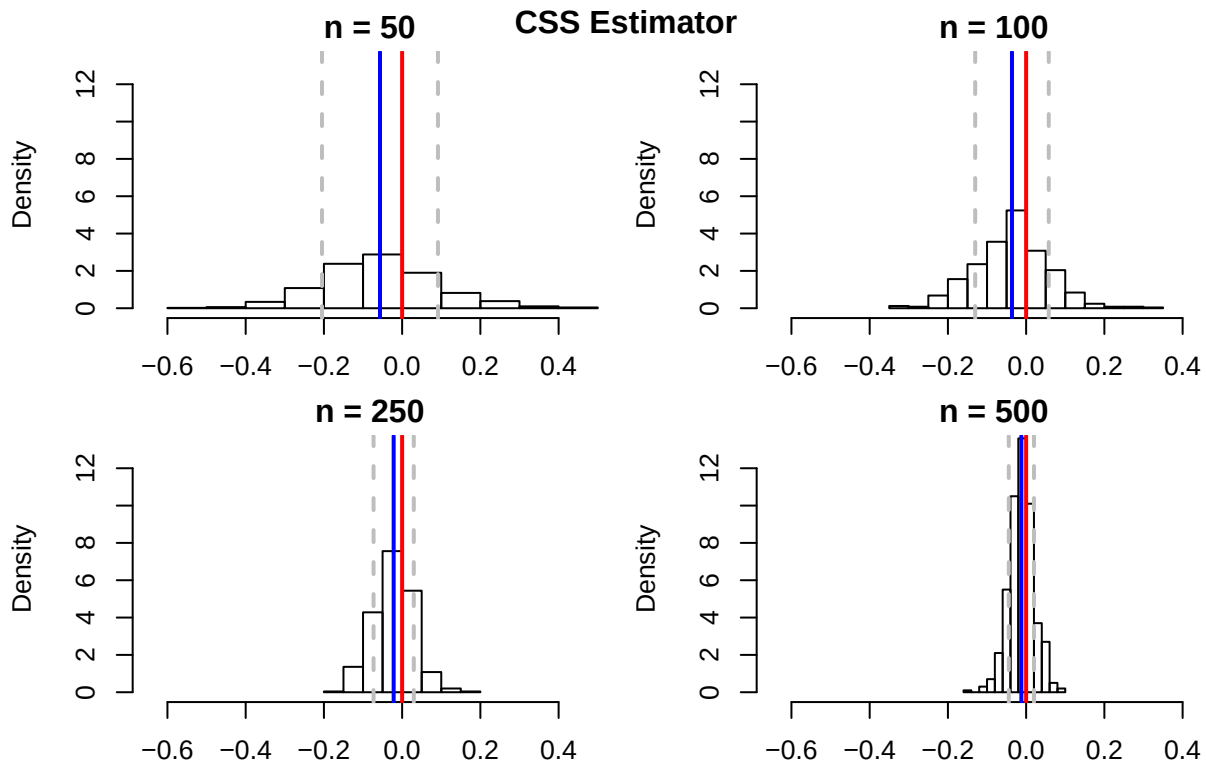
1. Distributions seem to have a "normal" shape.
2. Standard Errors shrink with the increase of n.
3. Both estimator appear to work well and be unbiased - with the increase of n the estimate seems to approach the true value of the parameter.
4. MLE seems to estimate marginally better than CSS and seem to converge faster.

```
a = 2
xlim = c(min(summary_table[,,a]), max(summary_table[,,a]))
ylim = c(0, max(hist(masim$mle.est[a,4,], plot=FALSE)$density))
```

```
plot_hists(a, xlim, ylim, main="MLE", est_data=masim$mle.est)
```



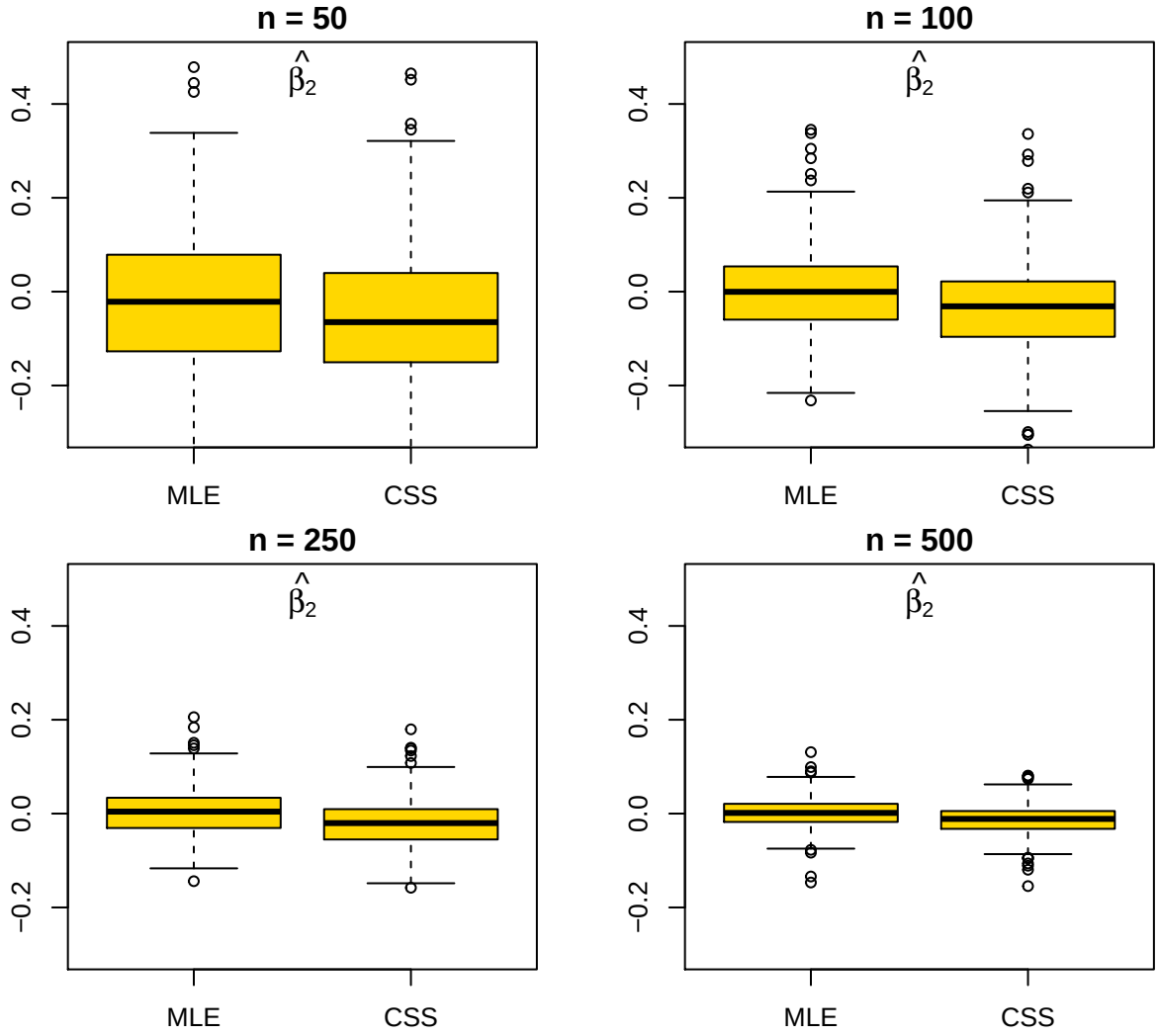

```
plot_hists(a, xlim, ylim, main="CSS", est_data=masim$css.est)
```



C. Boxplots

By slightly adjusting the boxplot function from AR section, we are able to plot boxplots for each estimator and increasing number of considered realizations for a given a:

```
plot_boxplots_ma = function(a) {
  par(mfrow=c(2,2), mar = c(2,4,1.5,0.5) + 0.1)
  for (num in 1:length(n)) {
    boxplot(masim$mle.est[a,num,], masim$css.est[a,num,],
            names=MA_est_names, col="gold",
            main=paste("n =", n[num]), pars=list(ylim=c(-0.3,0.5)))
    title(bquote(hat(beta[.(a)])), line=-1)
  }
}
plot_boxplots_ma(a=2)
```

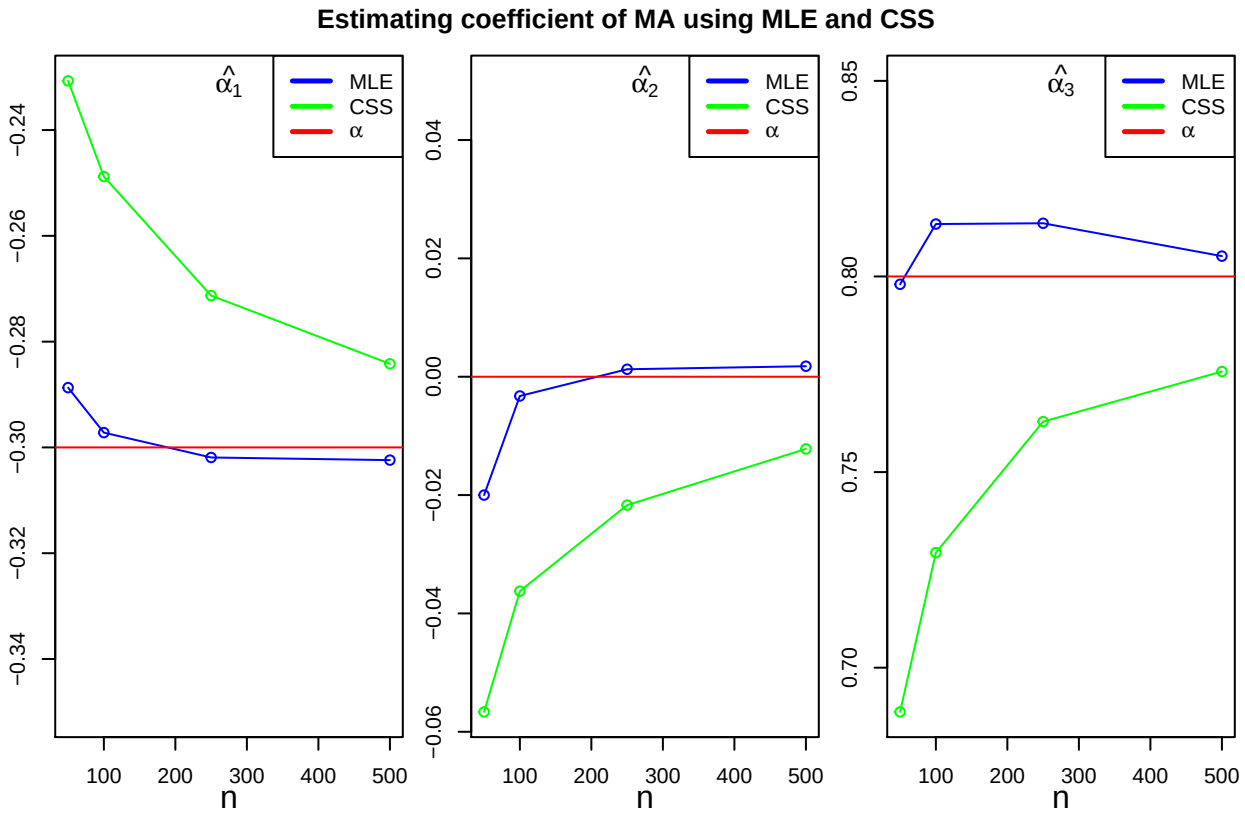


Which for $\hat{\beta}_2$ shows once more that MLE works slightly better.

D. Line plot of summary tables

Slightly adjusting the function from AR section we get the following imagery:

```
plot_est_converge_ma = function(a) {
  coef = paste('a', a, sep='')
  ylim = c(min(alpha[a] - 0.05, min(summary_table[, 'Mean', coef])),
           max(alpha[a] + 0.05, max(summary_table[, 'Mean', coef])))
  plot(n, summary_table[1:4, 'Mean', coef], type='o', col="Blue", ylim=ylim, xlab="")
  lines(n, summary_table[5:8, 'Mean', coef], type='o', col="Green")
  abline(h=alpha[a], col="Red")
  mtext(text="n", line=2, side = 1)
  title(bquote(hat(alpha[.(a)])), line=-1)
  legend('topright', legend=c(MA_est_names, expression(alpha)), lty=c(1,1,1),
         lwd=c(2.5,2.5,2.5),col=c("Blue","Green", "Red"))
}
plot_converge_estimators_ma = function() {
  par(mfrow=c(1,3), mar = c(4,2,3,0.5) + 0.1)
  for (a in 1:length(alpha))
    plot_est_converge_ma(a)
  title('Estimating coefficient of MA using MLE and CSS', line=-2, outer=TRUE)
}
plot_converge_estimators_ma()
```



Observations:

1. MLE estimator seemingly significantly underperforms CSS.

2. MLE seems to overestimate at the beginning (small n) which causes it to oscillate about true value of the parameter; it's hard to notice, but it seems that in case of α_1, α_2 MLE actually goes further from the true value on $n = 500$ than on the previous point at $n = 250$ which is not a good sign, in general, yet it is still very close to the true value (closer than CSS anyway).

3. CSS, as oppose to MLE, in our small observation sample, only approach the true value from one side and do not oscillate back and forth around the true value. If that observation is a true fact, allowing one to put tight guarantees on the true values, it could outweigh the benefit of faster convergence of MLE.

This difference in approaching from one side slowly vs. converging faster by overestimating (thus oscillating), if true, is a very important one; it would dictate the method of choice for the concrete real life problem.

That situation seems similar to the problem of Logistic Regression in Machine Learning, where one is to choose between Gradient Descent or Newton Method. Gradient descent works much slower but guarantees the results, while Newton methods works amazingly fast but has some problems on specific datasets since it calculates the Hessian Matrix, which is basically the second partial derivative. [1]

Summary of the Analysis

1. Sampling distributions appears to have "normal shape".

2. Both estimators appear to be unbiased; both converge to the true value of the parameters, but the MLE does so faster than CSS.

3. Increase of n reduces the standard error for both estimators.

4. To summarize, MLE outperforms (converges faster) CSS in estimation the true value of the parameter, yet both appear to be unbiased and work well.

5. I would probably choose MLE since it appears to converge faster and in real life we don't generally have many observations of the time series.

[1] For more details refer to the chapter 3 of this work: http://www.alexsalo.xyz/papers/investigations/machine-learning/mle_wlr_gradient-descent_newthon.pdf

3. Data Analysis

Descplot with smoothed spectrum estimate

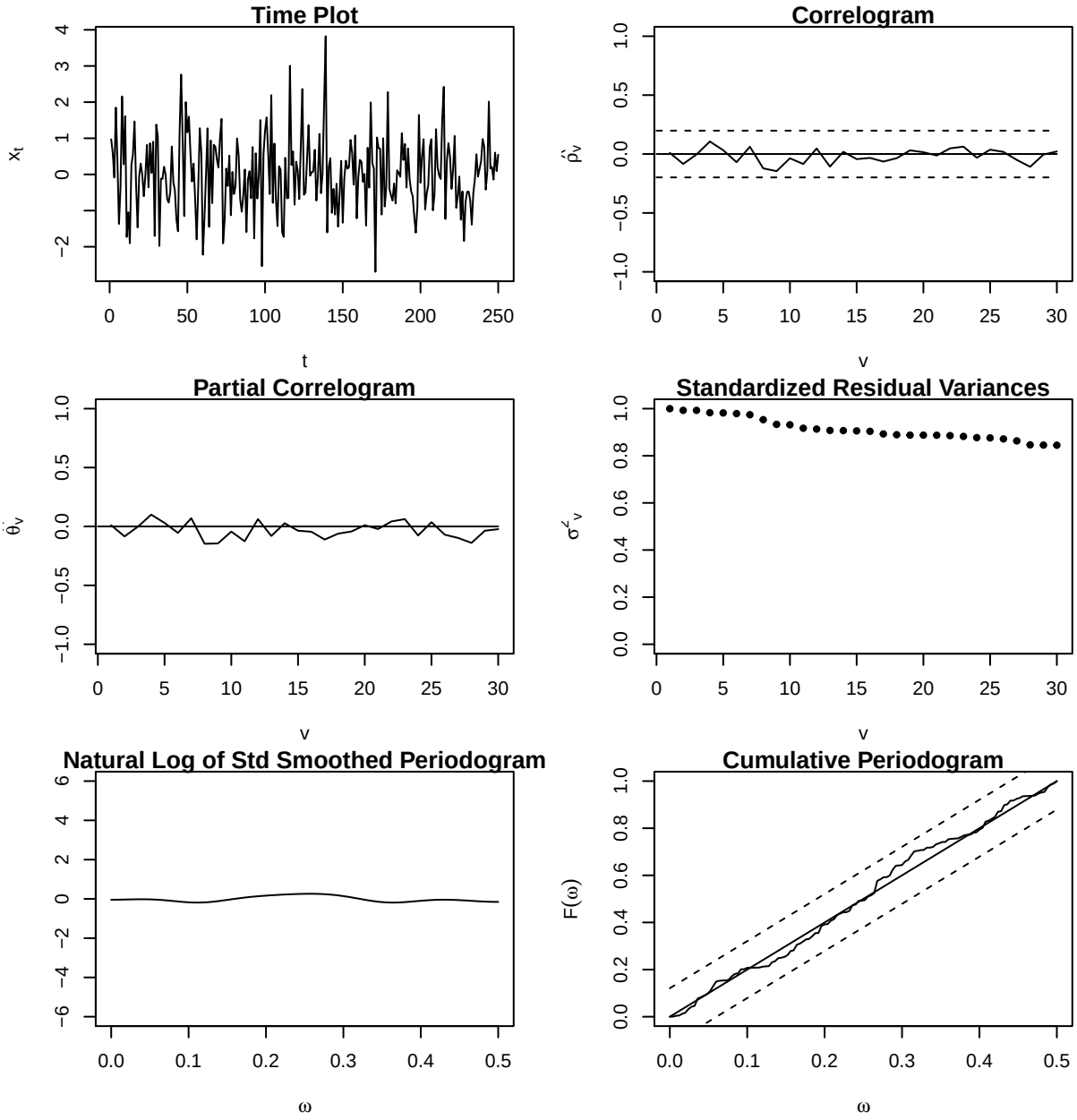
Let us modify the Timeslab's *descplot* function to plot the natural logarithm of the standardized **smoothed periodogram** instead of "raw" periodogram.

```
descplot2 <- function(x, m, alpha = 0.05)
  ...
  # Natural log of Standartized Periodogram
  # Original
  # plot(freqs(n), log(stdf(z$per, tsvar(x), exp(-6), exp(6))),
  #     type = "l", xlab = expression(omega), ylab = "", ylim = c(-6, 6),
  #     main="Natural Log of Standardized Periodogram")

  # Smoothed version
  M = ceiling(4*n^(1/5)/3) # Parzen's suggested optimal value
                           # If (M/n) outside of [0.05, 0.1] range
  if (M/n < 0.05) {        # Ceiling to the nearest int s.t. M/n > 0.05
    M = ceiling(0.05*n)
  }else if (M/n > 0.1) {   # Floor to the nearest int s.t. M/n < 0.1
    M = floor(0.1*n)
  }
  rho = corr(x, M)

  # Smooth with Parzen window of width M
  fhat = windowf(rho=rho$corr, R0=tsvar(x), Q=n, ioptw=4, M=M, n=n)$f
  plot(freqs(n), log(stdf(fhat, tsvar(x), exp(-6), exp(6))),
       type = "l", xlab = expression(omega), ylab = "", ylim = c(-6, 6),
       main="Natural Log of Standardized Smoothed Periodogram")
  ...
}
```

Now we can try to make an educated guess at what process generated given realizations.

Time Series 1

```
descplot2(ts1, m=30)
```



TS1 appears to be generated by a **white noise process** since correlogram and periodograms stay within the white noise boundaries. To estimate process variance, let us use Timeslab's sample autocorrelation function (acf), which outputs the variance estimation if no m is specified:

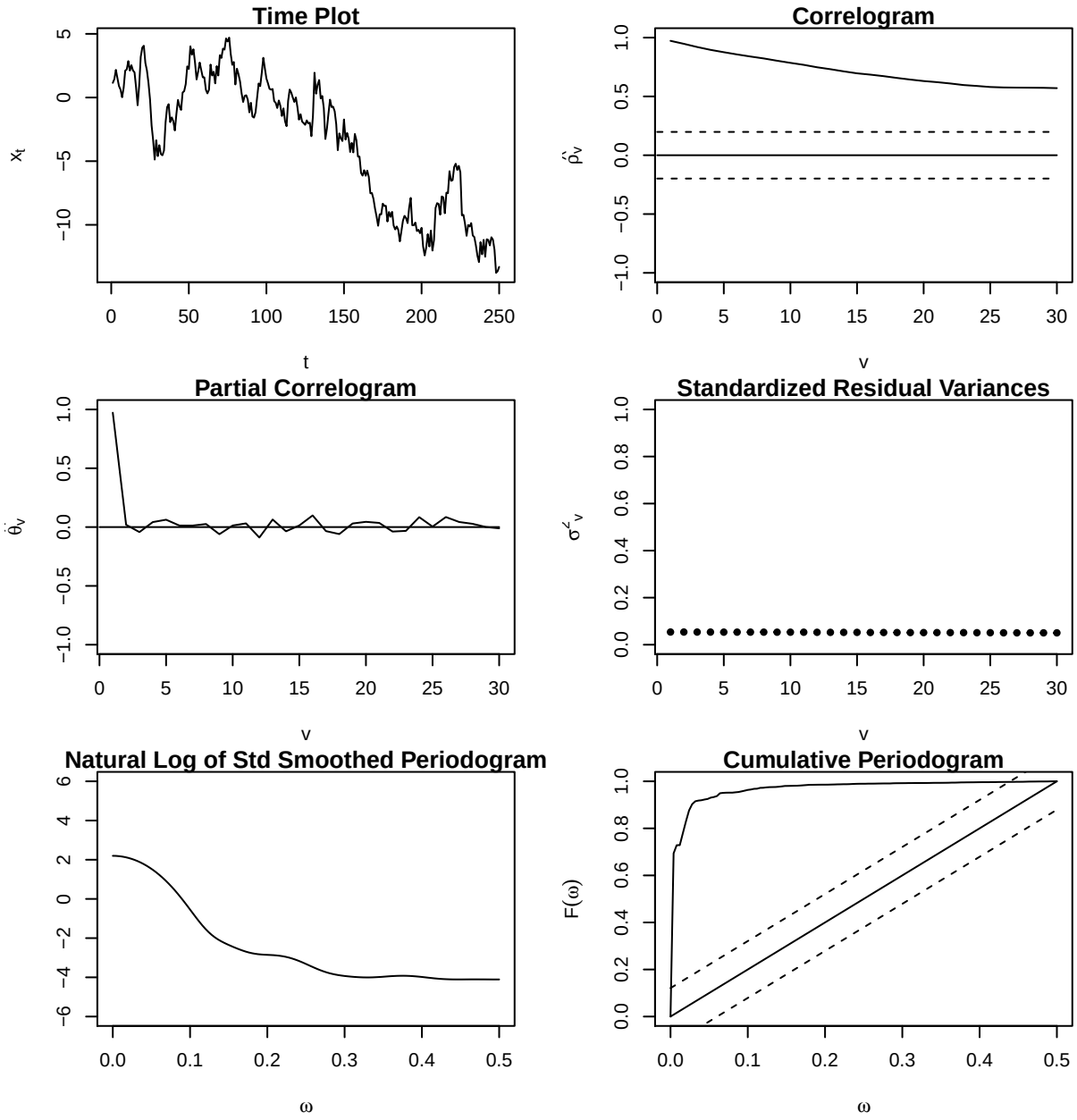
```
corr(ts1)
```

```
## [1] 1.050039
```

Thus we can guess that the process generated TS2 is $X_t^1 = WN(\sigma^2 = 1.05)$

Time Series 2

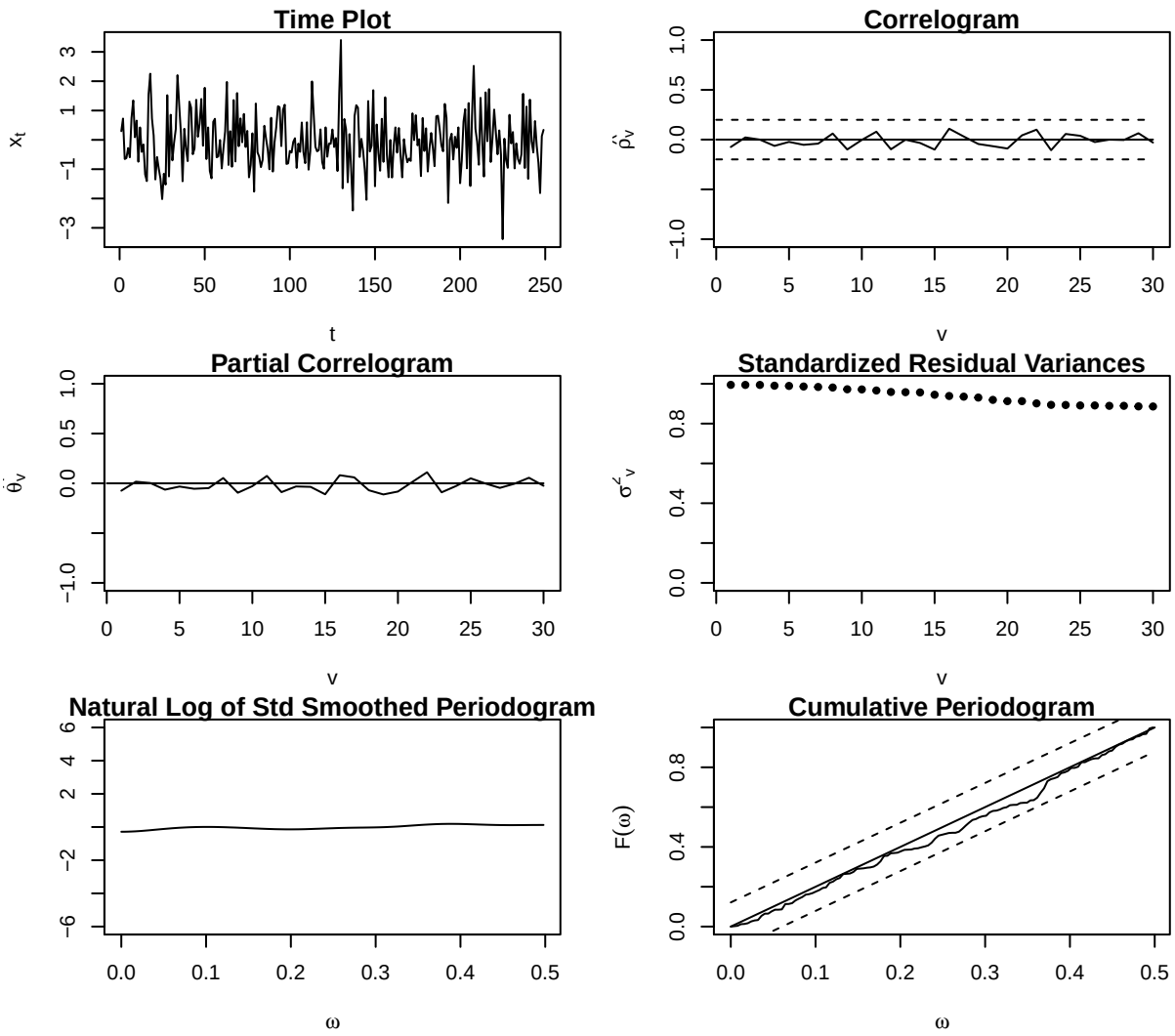
```
descplot2(ts2, m=30)
```



TS2 appears to be generated by a **random walk process** judging by the signature $F(\omega)$ plot (goes up almost as a straight line, then curves fast, then stays close to 1) and zero-after-first-lag PACF. Besides, the time plot looks like a stock market which the way RW plots usually look.

To check our guess, let's take the first difference and make a descplot:

```
descplot2(diff(ts2, 1), m=30)
```



From theory we know that taking first difference converts the random walk process back into white noise process, since RW was made by doing the reverse in the first place:

$$X_t = X_{t-1} + \varepsilon_t$$
$$X_t - X_{t-1} = \varepsilon_t$$

By looking at the descplot we can confirm that taking the first difference transformed the TS back into white noise (since both correlogram and cumulative periodogram are withing the white noise boundaries).

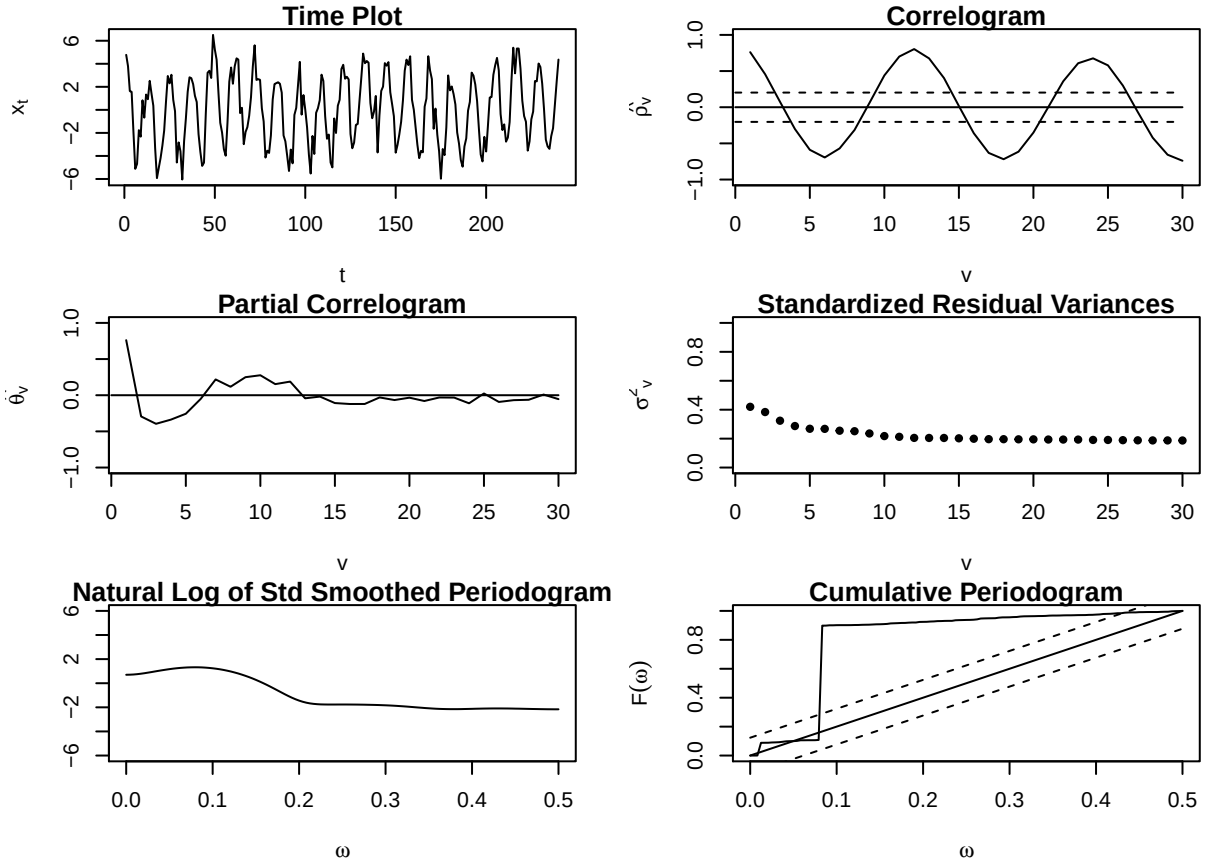
```
corr(diff(ts2, 1))
```

```
## [1] 0.9281478
```

Thus we can guess that the process generated TS2 is $X_t^2 = RW(\sigma^2 = 0.93)$

Time Series 3

```
descplot2(ts3, m=30)
```



TS3 appears to be generated by a **harmonic process** since the correlogram looks like a sinusoid and cumulative periodogram has distinct "jumps". Since we see two such distinct jumps we can guess that harmonic consists of two sinusoids. To determine the frequency components and estimate the corresponding amplitudes, we use the sinusoidal decomposition theorem and the formula of natural frequencies $\omega_n = \frac{(k-1)}{n}$.

```
n = length(ts3)
# Decompose TS into frequency domain via FFT
ts3.freqs = round(Mod(fft(ts3))[1:(n/2+1)] * 2 / n, 6)

# Weed out small values which are likely due to the noise
ts3.freqs[ts3.freqs < 1] = 0

n / (-1 + which(ts3.freqs > 0))  # Find frequencies
```

```
## [1] 80 12
```

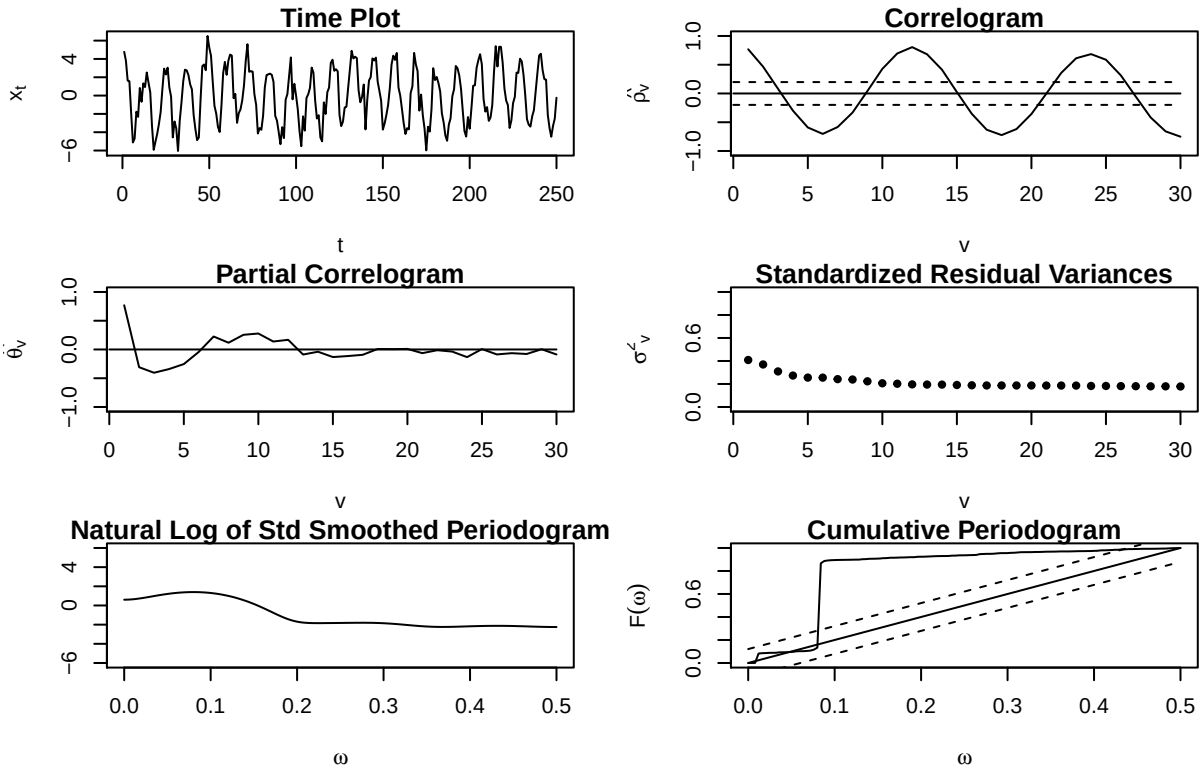
```
ts3.freqs[ts3.freqs > 0]          # Find amplitudes
```

```
## [1] 1.232620 3.699379
```

Thus we can guess that the process generated TS3 is $X_t^3 = 1.2cos\frac{2\pi(t-1)}{80} + 3.7cos\frac{2\pi(t-1)}{12}$

Time Series 4

```
descplot2(ts4, m=30)
```



TS4 appears to be generated by a **harmonic process** since it looks very similar to the TS3. However, on TS4 descplot's cumulative preiodogram we notice that two distinct "jumps" are **not so sharp** as in TS3. That is an evidence of the fact that periodogram didn't "catch" natural frequencies, i.e. **frequencies are not of the form** $\omega_n = \frac{(k-1)}{n}$.

Calculations performed with TS3 are still valid for this case, although we would probably get less precise estimate and we probably want to round the frequencies to the nearest values:

```
n = length(ts4)
ts4.freqs = round(Mod(fft(ts4))[1:(n/2+1)] * 2 / n, 6)
ts4.freqs[ts4.freqs < 1] = 0
ts4.frecuencies = n / (-1 + which(ts4.freqs > 0))          # Find frequencies
ts4.roundedfreq = round(n / (-1 + which(ts4.freqs > 0)))   # Round frequencies
print(rbind(ts4.frecuencies, ts4.roundedfreq))
```

```
##                      [,1]     [,2]
## ts4.frecuencies 83.33333 11.90476
## ts4.roundedfreq 83.00000 12.00000
```

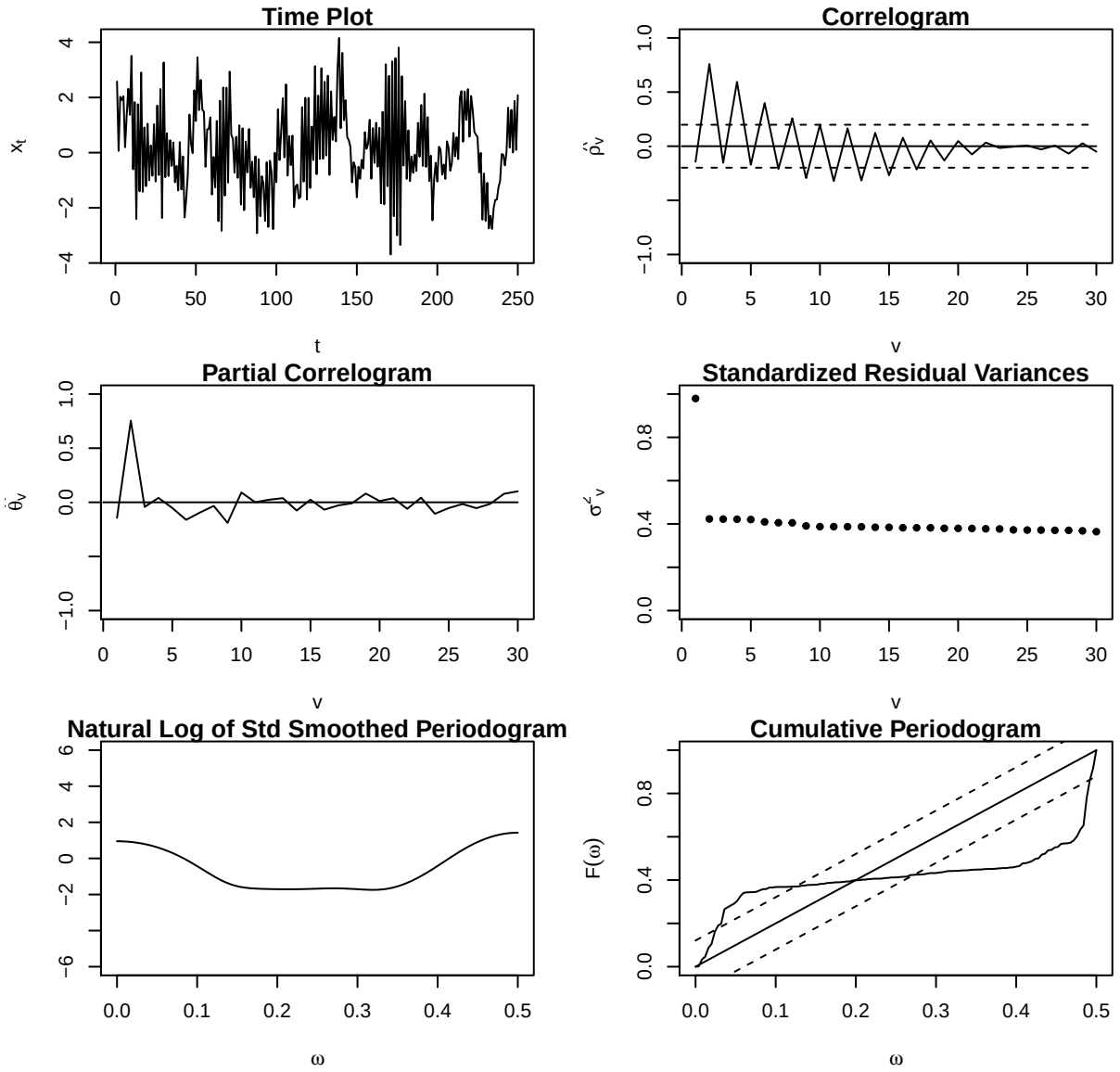
```
ts4.freqs[ts4.freqs > 0]                                   # Find amplitudes
```

```
## [1] 1.193429 3.565754
```

Thus we can guess that the process generated TS4 is $X_t^4 = 1.2cos\frac{2\pi(t-1)}{83} + 3.6cos\frac{2\pi(t-1)}{12}$

Time Series 5

```
descplot2(ts5, m=30)
```



TS5 appears to be generated by a **auto-regressive process** judging by the signature correlogram (evading saw-tooth-like), PACF (peak than almost-flat), periodogram (inverse bump) and $F(\omega)$.

Let us determine the optimal order of the model and then fit the coefficients based on the realization that we have. Along with *auto.arima* tests that help to determine the order, we can use the sample ACF and PACF to do the same task. We know, that since AR has the following autocovariance $\sum_{j=0}^{p} \alpha_j R_{j-v} = \delta_v \sigma^2$, we can look for the value at which PACF becomes very close to 0. Our case the value is 2, meaning the AR order is $p = 2$.

Besides, we also have learned that the resid variances of AR realizaiton should drop at the $lag = order$ which again yields $p = 2$.

(i) Optimal Order

Let us demean the TS and use three different methods for determining the optimal order:

```
# library(forecast)          # Contains auto.arima function

ts5.demeaned= ts5-mean(ts5) # Demean the Time Series
auto.arima(x=ts5.demeaned,  # Demeaned Time Series
           d=0,             # Order of first-differencing
           D=0,             # Order of differencing (for arIma)
           max.p=5,         # Max AR order
           max.q=5,         # MAX MA order
           stationary=TRUE, # Restrict to only stationary models
           seasonal=FALSE,  # Allow seasonal models
           ic="aic")        # Akaike's Information Criteria
```

```
## Series: ts5.demeaned
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##           ar1     ar2
##       -0.0345  0.7593
## s.e.   0.0409  0.0408
##
## sigma^2 estimated as 1.044:  log likelihood=-360.99
## AIC=727.97   AICc=728.07   BIC=738.54
```

```
auto.arima(x=ts5.demeaned, d=0, D=0, stationary=T, seasonal=F,
           ic="aicc")       # Corrected AIC
```

```
## Series: ts5.demeaned
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##           ar1     ar2
##       -0.0345  0.7593
## s.e.   0.0409  0.0408
##
## sigma^2 estimated as 1.044:  log likelihood=-360.99
## AIC=727.97   AICc=728.07   BIC=738.54
```

```
auto.arima(x=ts5.demeaned, d=0, D=0, stationary=T, seasonal=F,
           ic="bic")        # Bayesian Information Criteria
```

```
## Series: ts5.demeaned
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##           ar1     ar2
##       -0.0345  0.7593
## s.e.   0.0409  0.0408
##
## sigma^2 estimated as 1.044:  log likelihood=-360.99
## AIC=727.97   AICc=728.07   BIC=738.54
```

Method	AR order	MA order
PACF	2	0
AIC	2	0
AICC	2	0
BIC	2	0
Model	2	0

Table 1: Comparison of the methods for determining optimal ARMA order

(i) Fitting the model

Let us now fir the model using maximum likelihood estimates using the optimal order based on AICC, which is the same as the true model's order:

```
ts5.fit = arima(x=ts5.demeaned,     # Demeaned Time Series
                order=c(2, 0, 0),   # vector (p, d, q)
                include.mean=F,     # whether or not include a mean in the fit
                method = "CSS-ML") # CSS to find initial values, then ML
summary(ts5.fit)
```

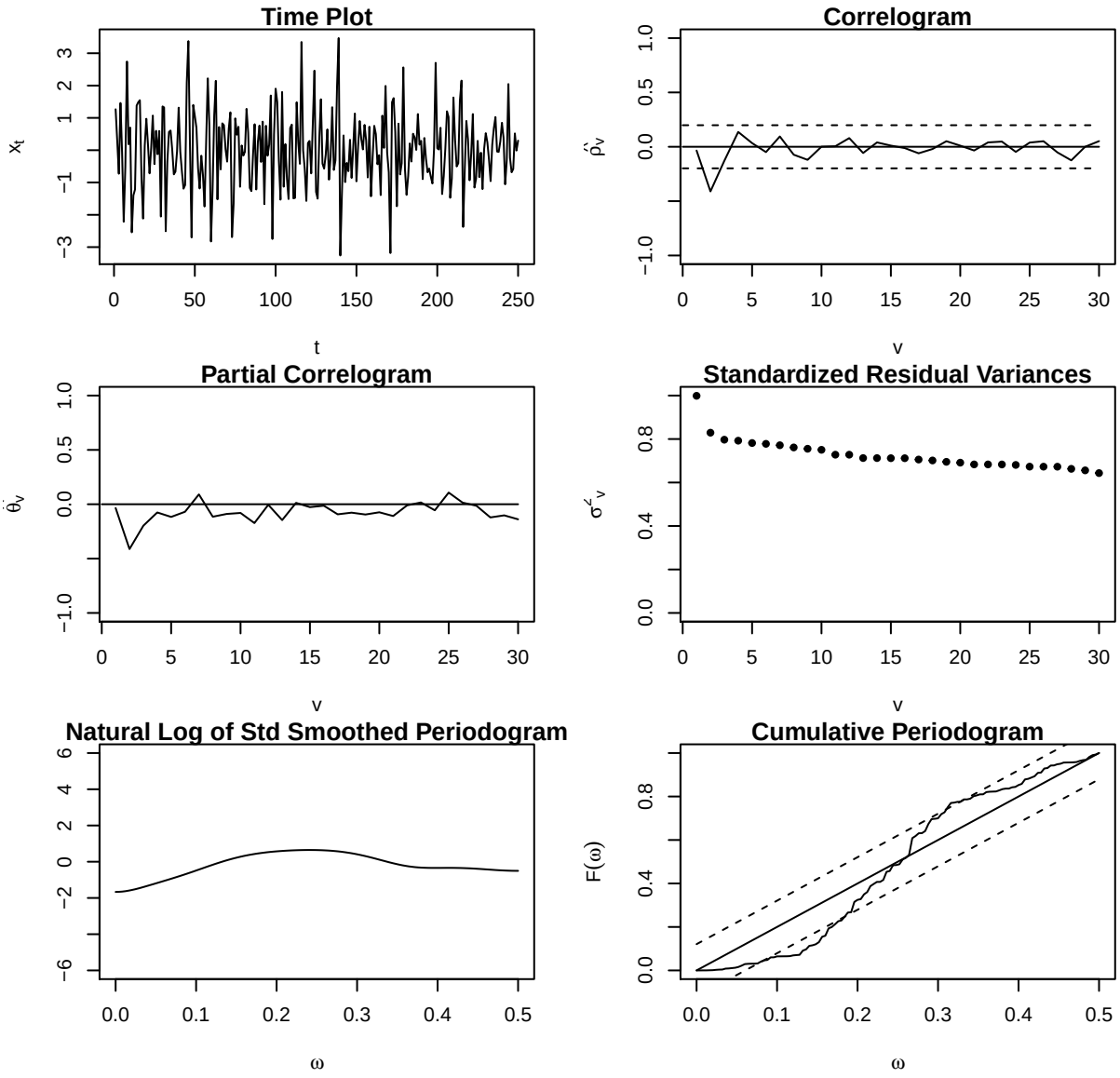
```
##
## Call:
## arima(x = ts5.demeaned, order = c(2, 0, 0), include.mean = F, method = "CSS-ML")
##
## Coefficients:
##           ar1     ar2
##       -0.0345  0.7593
## s.e.   0.0409  0.0408
##
## sigma^2 estimated as 1.044:  log likelihood = -360.99,  aic = 727.97
##
## Training set error measures:
##                       ME     RMSE       MAE       MPE     MAPE      MASE
## Training set 0.002668558 1.021761 0.7946565 -4.798024 143.4605 0.4116807
##                     ACF1
## Training set 0.03359168
```

Thus we can estimate that the AR process generated TS5 is

$$X_t = 0.0345X_{t-1} - 0.7593X_{t-2} + \varepsilon_t$$

Time Series 6

```
descplot2(ts6, m=30)
```



TS6 appears to be generated by a **moving average process** judging by the signature periodogram (bump-like), $F(\omega)$ (S-shaped) and ACF (no particular pattern).

Let us determine the optimal order of the model and then fit the coefficients based on the realization that we have. We know for MA process:

$$R_v = \begin{cases} \sigma^2 \sum_{j=0}^{q-|v|} \beta_j \beta_{j+|v|} & |v| = 1, .., q \\ 0 & |v| = q+1, ... \end{cases}$$

Which also says that PACF of the true MA is zero after the lag=q. On our realizaiton we see that after value $v = 3$ PACF is very close to zero, thus we can guess $q = 3$.

(i) Optimal Order

Let us demean the TS and use three different methods for determining the optimal order:

```
ts6.demeaned= ts6-mean(ts6) # Demean the Time Series
auto.arima(x=ts6.demeaned, d=0, D=0, stationary=T, seasonal=F, ic="aic")
```

```
## Warning in auto.arima(x = ts6.demeaned, d = 0, D = 0, stationary = T,
## seasonal = F, : Unable to fit final model using maximum likelihood. AIC
## value approximated
```

```
## Series: ts6.demeaned
## ARIMA(2,0,4) with non-zero mean
##
## Coefficients:
##           ar1      ar2     ma1      ma2      ma3      ma4  intercept
##       -0.9027  -0.5201  0.6825  -0.2511  -0.8967  -0.4223    -0.0067
## s.e.   0.3667   0.1291  0.3924   0.1129   0.1989   0.1324     0.0033
##
## sigma^2 estimated as 0.9967:  log likelihood=-354.32
## AIC=724.65   AICc=725.25   BIC=752.82
```

```
auto.arima(x=ts6.demeaned, d=0, D=0, stationary=T, seasonal=F, ic="aicc")
```

```
## Series: ts6.demeaned
## ARIMA(2,0,4) with zero mean
##
## Coefficients:
##           ar1      ar2     ma1      ma2      ma3      ma4
##       -1.1766  -0.6107  0.9574  -0.2024  -1.0373  -0.5200
## s.e.   0.1536   0.1307  0.1428   0.0915   0.1016   0.0803
##
## sigma^2 estimated as 0.9941:  log likelihood=-355.12
## AIC=724.24   AICc=724.7   BIC=748.89
```

```
auto.arima(x=ts6.demeaned, d=0, D=0, stationary=T, seasonal=F, ic="bic")
```

```
## Series: ts6.demeaned
## ARIMA(0,0,3) with zero mean
##
## Coefficients:
##           ma1      ma2      ma3
##       -0.2182  -0.5108  -0.2135
## s.e.   0.0639   0.0561   0.0717
##
## sigma^2 estimated as 1.023:  log likelihood=-358.62
## AIC=725.25   AICc=725.41   BIC=739.33
```

Method	AR order	MA order
PACF	0	3
AIC	2	4
AICC	2	4
BIC	0	3
Model	0	3

Table 2: Comparison of the methods for determining optimal ARMA order

(i) Fitting the model. Let's find coefficients based on the AICC's optimal order and the true order, which is estimated correctly by BIC:

```
summary(arima(x=ts6.demeaned, order=c(2, 0, 4), include.mean=F, method = "CSS-ML")) # AICC
```

```
##
## Call:
## arima(x = ts6.demeaned, order = c(2, 0, 4), include.mean = F, method = "CSS-ML")
##
## Coefficients:
##           ar1      ar2     ma1      ma2      ma3      ma4
##       -1.1766  -0.6107  0.9574  -0.2024  -1.0373  -0.5200
## s.e.   0.1536   0.1307  0.1428   0.0915   0.1016   0.0803
##
## sigma^2 estimated as 0.9941:  log likelihood = -355.12,  aic = 724.24
##
## Training set error measures:
##                       ME      RMSE       MAE       MPE    MAPE      MASE
## Training set -0.07299608 0.9970419 0.7789028 -29.08692 298.563 0.5873834
##                     ACF1
## Training set 0.01247662
```

```
summary(arima(x=ts6.demeaned,order=c(0,0,3),include.mean=F,method="CSS-ML")) # True order
```

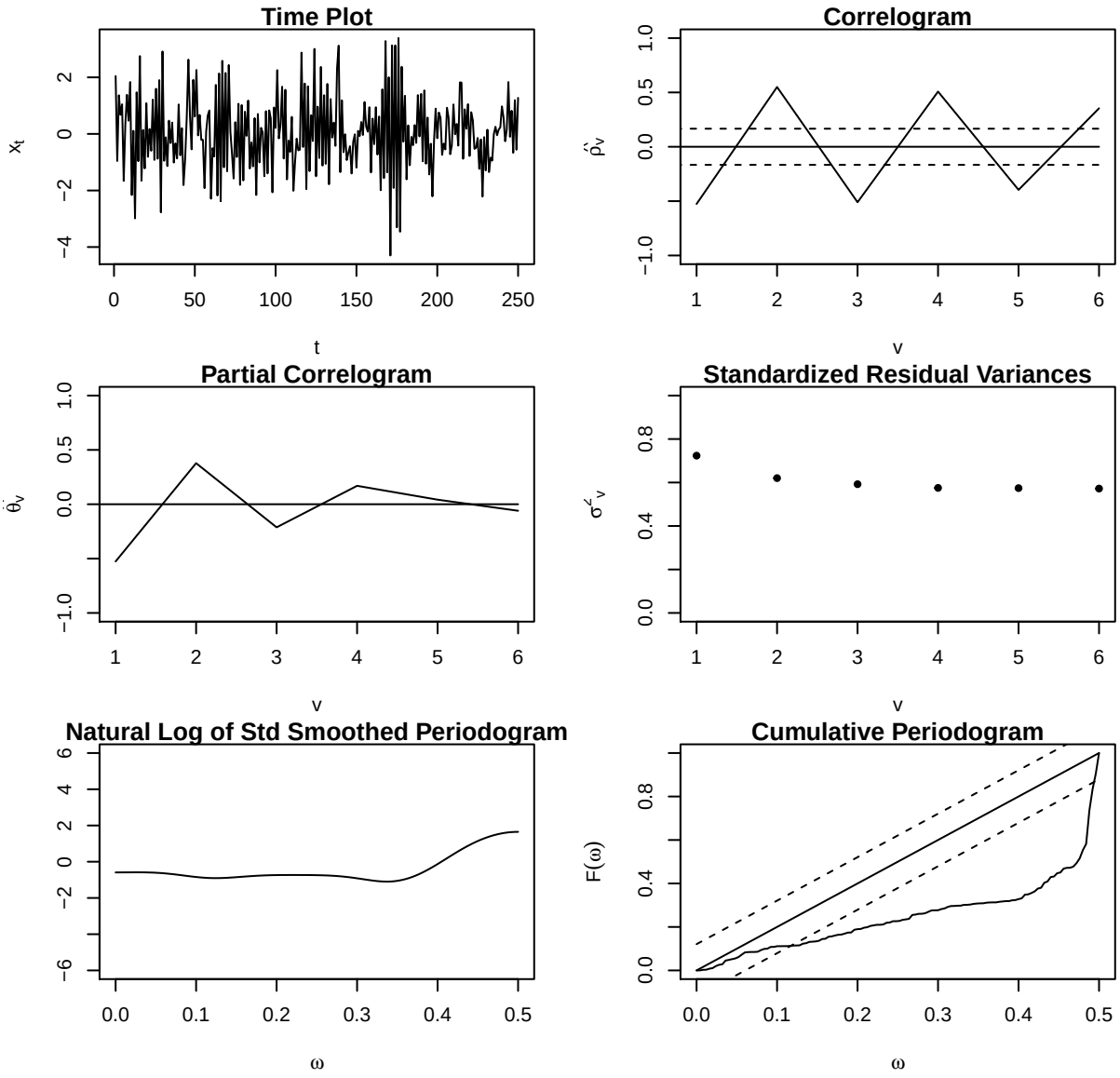
```
##
## Call:
## arima(x = ts6.demeaned, order = c(0, 0, 3), include.mean = F, method = "CSS-ML")
##
## Coefficients:
##           ma1      ma2      ma3
##       -0.2182  -0.5108  -0.2135
## s.e.   0.0639   0.0561   0.0717
##
## sigma^2 estimated as 1.023:  log likelihood = -358.62,  aic = 725.25
##
## Training set error measures:
##                       ME     RMSE       MAE      MPE     MAPE      MASE
## Training set -0.08302501 1.011399 0.7856779 291.4598 432.3955 0.5924926
##                     ACF1
## Training set -0.01483461
```

Thus we can estimate that the MA process generated TS6 is

$$X_t = \varepsilon_t - 0.22\varepsilon_{t-1} - 0.51\varepsilon_{t-2} - 0.22\varepsilon_{t-3}$$

Time Series 7

```
descplot2(ts7, m=6)
```



TS6 appears to be generated by a **auto-regressive moving average process** judging by the signature $F(\omega)$ and ACF (evading saw-tooth-like).

Let us determine the optimal order of the model and then fit the coefficients based on the realization that we have. It's harder for the mix of the two models (AR and MA), but we know that for ARMA process:

$$R_v = \begin{cases} \sum_{j=0}^{p} \alpha_j R_{j-v} = \sigma^2 \gamma^2_{-v} & v = 0, .., -q \\ \sum_{j=0}^{p} \alpha_j R_{j-v} = 0 & v = 1, 2, ..., q \\ 0 & |v| = q+1, ... \end{cases}$$

Since resid variances drop at 2 we conclude that AR order is $p = 2$; since PACF is close to zero after lag of 4, I'd conclude that that MA order is $q = 4$, which is slightly off.

(i) Optimal Order

Let us demean the TS and use three different methods for determining the optimal order:

```
ts7.demeaned= ts7-mean(ts7) # Demean the Time Series
auto.arima(x=ts7.demeaned, d=0, D=0, stationary=T, seasonal=F,
           ic="aic")       # Akaike's Information Criteria
```

```
## Warning in auto.arima(x = ts7.demeaned, d = 0, D = 0, stationary = T,
## seasonal = F, : Unable to fit final model using maximum likelihood. AIC
## value approximated
```

```
## Series: ts7.demeaned
## ARIMA(3,0,4) with zero mean
##
## Coefficients:
##          ar1     ar2      ar3      ma1     ma2     ma3     ma4
##       0.4090  0.4018  -0.6882  -0.6607  0.0380  0.3258  0.3056
## s.e.  0.0667  0.0666   0.0639   0.0817  0.0913  0.1100  0.0878
##
## sigma^2 estimated as 0.9746:  log likelihood=-351.51
## AIC=719.36   AICc=719.96   BIC=747.53
```

```
auto.arima(x=ts7.demeaned, d=0, D=0, stationary=T, seasonal=F,
           ic="aicc")       # Corrected AIC
```

```
## Warning in auto.arima(x = ts7.demeaned, d = 0, D = 0, stationary = T,
## seasonal = F, : Unable to fit final model using maximum likelihood. AIC
## value approximated
```

```
## Series: ts7.demeaned
## ARIMA(3,0,4) with zero mean
##
## Coefficients:
##          ar1     ar2      ar3      ma1     ma2     ma3     ma4
##       0.4090  0.4018  -0.6882  -0.6607  0.0380  0.3258  0.3056
## s.e.  0.0667  0.0666   0.0639   0.0817  0.0913  0.1100  0.0878
##
## sigma^2 estimated as 0.9746:  log likelihood=-351.51
## AIC=719.36   AICc=719.96   BIC=747.53
```

```
auto.arima(x=ts7.demeaned, d=0, D=0, stationary=T, seasonal=F,
           ic="bic")        # Bayesian Information Criteria
```

```
## Series: ts7.demeaned
## ARIMA(1,0,1) with zero mean
##
## Coefficients:
##          ar1     ma1
##       -0.9373  0.6449
## s.e.   0.0278  0.0581
##
## sigma^2 estimated as 1.065:  log likelihood=-363
## AIC=732   AICc=732.1   BIC=742.57
```

Method	AR order	MA order
PACF	2	4
AIC	3	4
AICC	3	4
BIC	1	1
Model	2	3

Table 3: Comparison of the methods for determining optimal ARMA order

textbf{(i) Fitting the model} Let us now fit the model using maximum likelihood estimates based on AICC' optimal order and the true order:

```
summary(arima(x=ts7.demeaned, order=c(3, 0, 4), include.mean=F, method = "CSS-ML")) # AICC
```

```
##
## Call:
## arima(x = ts7.demeaned, order = c(3, 0, 4), include.mean = F, method = "CSS-ML")
##
## Coefficients:
##          ar1     ar2      ar3      ma1     ma2     ma3     ma4
##       0.4328  0.4000  -0.6781  -0.7107  0.0699  0.3401  0.3107
## s.e.  0.0613  0.0687   0.0629   0.0821  0.0968  0.1168  0.0980
##
## sigma^2 estimated as 0.9326:  log likelihood = -349.31,  aic = 714.61
##
## Training set error measures:
##                       ME      RMSE       MAE      MPE     MAPE      MASE
## Training set -0.001861458 0.9657177 0.7609637 54.87724 161.0123 0.4015061
##                      ACF1
## Training set -0.006860444
```

```
summary(arima(x=ts7.demeaned,order=c(2,0,3),include.mean=F,method="CSS-ML")) # True order
```

```
##
## Call:
## arima(x = ts7.demeaned, order = c(2, 0, 3), include.mean = F, method = "CSS-ML")
##
## Coefficients:
##          ar1     ar2      ma1      ma2      ma3
##       0.0223  0.8325  -0.2603  -0.5250  -0.2147
## s.e.  0.0444  0.0464   0.0752   0.0793   0.0828
##
## sigma^2 estimated as 1.005:  log likelihood = -357,  aic = 726
##
## Training set error measures:
##                      ME    RMSE      MAE      MPE     MAPE      MASE
## Training set -0.0485148 1.00257 0.777927 56.47421 159.7333 0.4104564
##                     ACF1
## Training set -0.00920898
```

Thus we can estimate that the ARMA process generated TS7 is

$$X_t = 0.02X_{t-1} + 0.83X_{t-2} + \varepsilon_t - 0.26\varepsilon_{t-1} - 0.52\varepsilon_{t-2} - 0.21\varepsilon_{t-3}$$

4. Predicting ARMA(p, q) models

1. Fit ARMA.

We are given the true orders of TS8: $p = 1, q = 2$. Let us fit all but last ten data points:

```
ts8.demeaned = ts8 - mean(ts8)
ts8.trunc = ts8.demeaned[1:(length(ts8.demeaned)-10)]
ts8.fit = arima(x=ts8.trunc, order=c(1, 0, 2), include.mean=F, method = "CSS-ML")
summary(ts8.fit)
```

```
##
## Call:
## arima(x = ts8.trunc, order = c(1, 0, 2), include.mean = F, method = "CSS-ML")
##
## Coefficients:
##          ar1     ma1      ma2
##       -0.7188  0.2156  -0.6419
## s.e.   0.0918  0.0839   0.0556
##
## sigma^2 estimated as 1.058:  log likelihood = -347.75,  aic = 703.5
##
## Training set error measures:
##                       ME     RMSE       MAE      MPE     MAPE     MASE
## Training set -0.02568916 1.028528 0.8047752 137.5164 226.6367 0.521132
##                      ACF1
## Training set 0.0009464212
```

Which estimate that the model looks as:

$$X_t = -0.72X_{t-1} + \varepsilon_t + 0.22\varepsilon_{t-1} - 0.64\varepsilon_{t-2}$$

2. Predict future points.

Using *predict* we now predict the last 10 data points and compute standard error of the prediction for each:

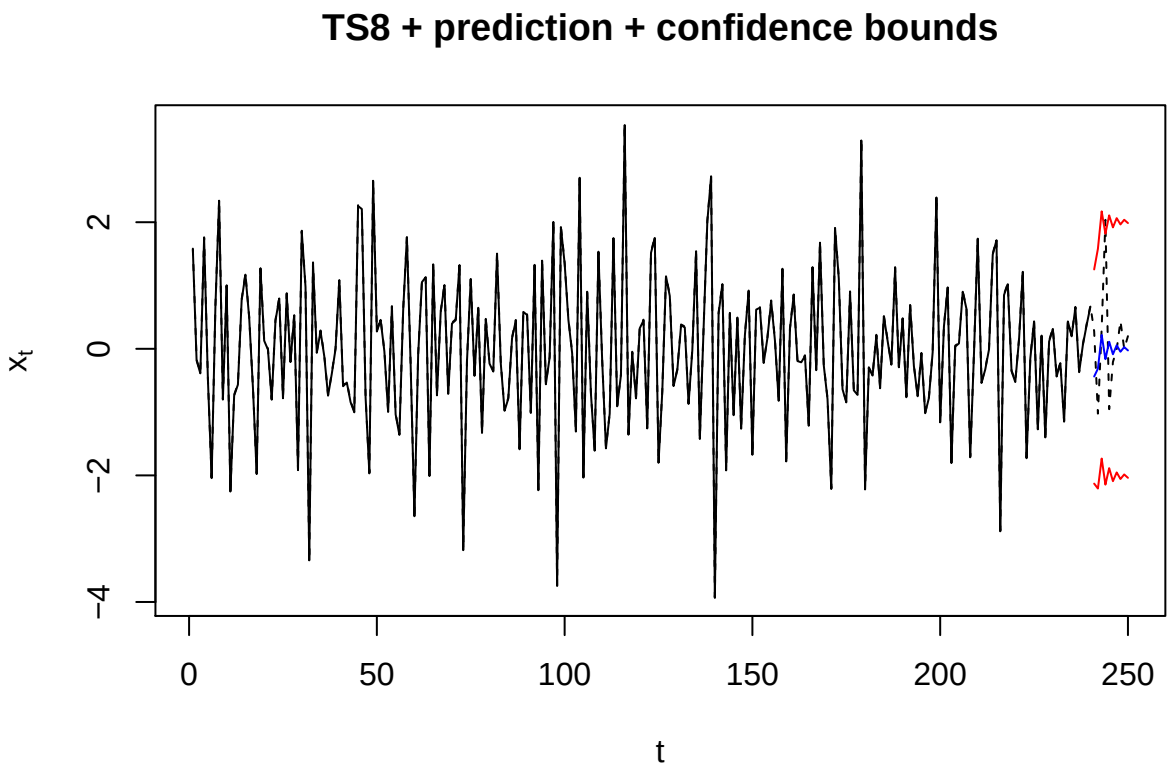
```
ts8.pred = predict(ts8.fit, n.ahead=10)
ts8.pred
```

```
## $pred
## Time Series:
## Start = 241
## End = 250
## Frequency = 1
##  [1] -0.43411574 -0.30960867  0.22255994 -0.15998559  0.11500447
##  [6] -0.08267013  0.05942682 -0.04271854  0.03070791 -0.02207416
##
## $se
## Time Series:
## Start = 241
## End = 250
## Frequency = 1
##  [1] 1.028528 1.151430 1.186923 1.204853 1.214015 1.218722 1.221147
##  [8] 1.222399 1.223045 1.223379
```

3. Plot TS and superimpose prediction.

Last 10 points (not used in fitting) are shown with dashed line.Predicted 10 values are shown as blue line. Red lines are the ±1.645 standard error confidence bounds:

```
# Plot TS
plot(ts8, type="l", lty=2, xlab="t", ylab=expression(x[t]),
     main="TS8 + prediction + confidence bounds")
lines(ts8.trunc, type="l")

# Superimpose predicted values
predicted = ts8.pred$pred + mean(ts8.trunc)
lines(predicted, type="l", col="blue")

# Superimpose +-1.645 Std Error
std_1.645 = 1.645 * ts8.pred$se
lines(predicted + std_1.645, type="l", col="red")
lines(predicted - std_1.645, type="l", col="red")
```



4. Prediction accuracy.

Nine out of ten realized data points are contained within the prediction intervals, as could be seen on the zoomed plot:

```r
# Plot TS
interval = 230:250
plot(interval, ts8[interval], type="o", lty=2, xlab="t", ylab=expression(x[t]),
     ylim=c(min(ts8), max(ts8)), main="TS8 prediction and confidence (zoomed)")
abline(v=240, lty=2)
lines(ts8.trunc, type="l")

# Superimpose predicted values
predicted = ts8.pred$pred + mean(ts8.trunc)
lines(predicted, type="o", col="blue")

# Superimpose +-1.645 Std Error
std_1.645 = 1.645 * ts8.pred$se
lines(predicted + std_1.645, type="o", col="red")
lines(predicted - std_1.645, type="o", col="red")

# Legend
legend('topleft', legend=c("Realization", "Predicted points",
                           "Realized points", "Confidence bounds"),
       lty=c(1,1,2,1), lwd=c(2.5,2.5,2.5,2.5),
       col=c("Black", "Blue", "Black", "Red"))
```