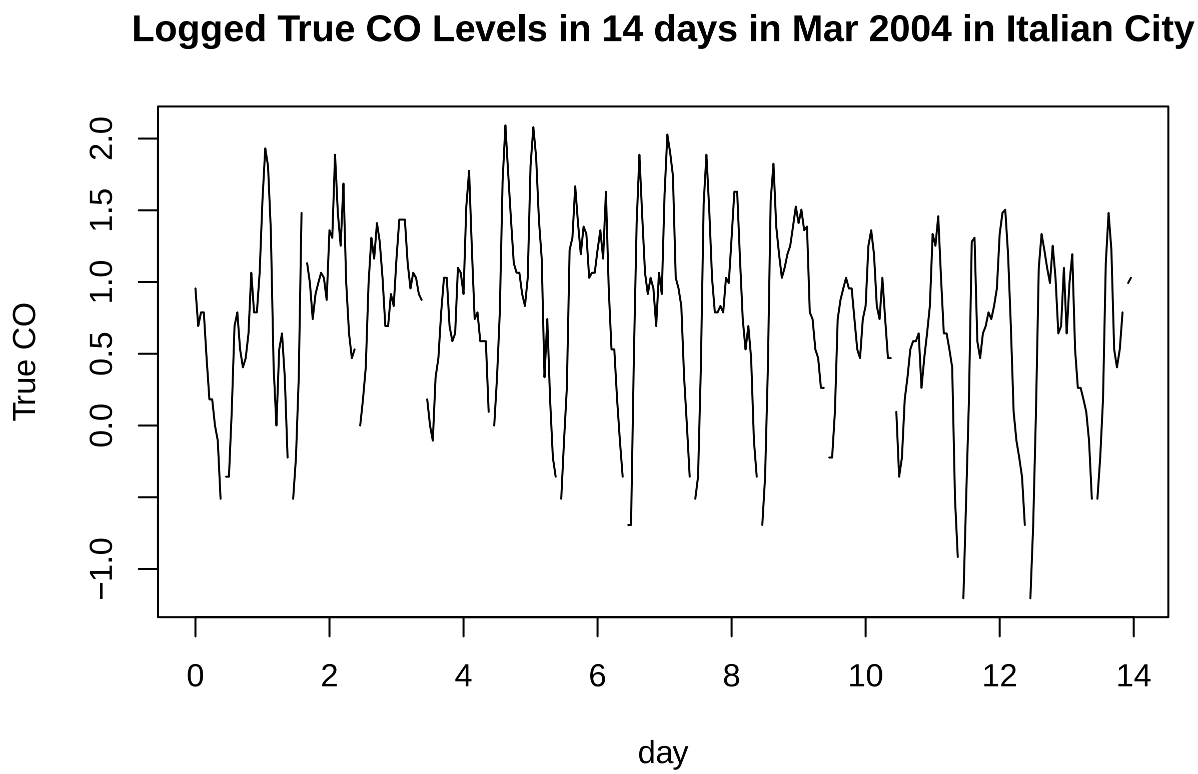
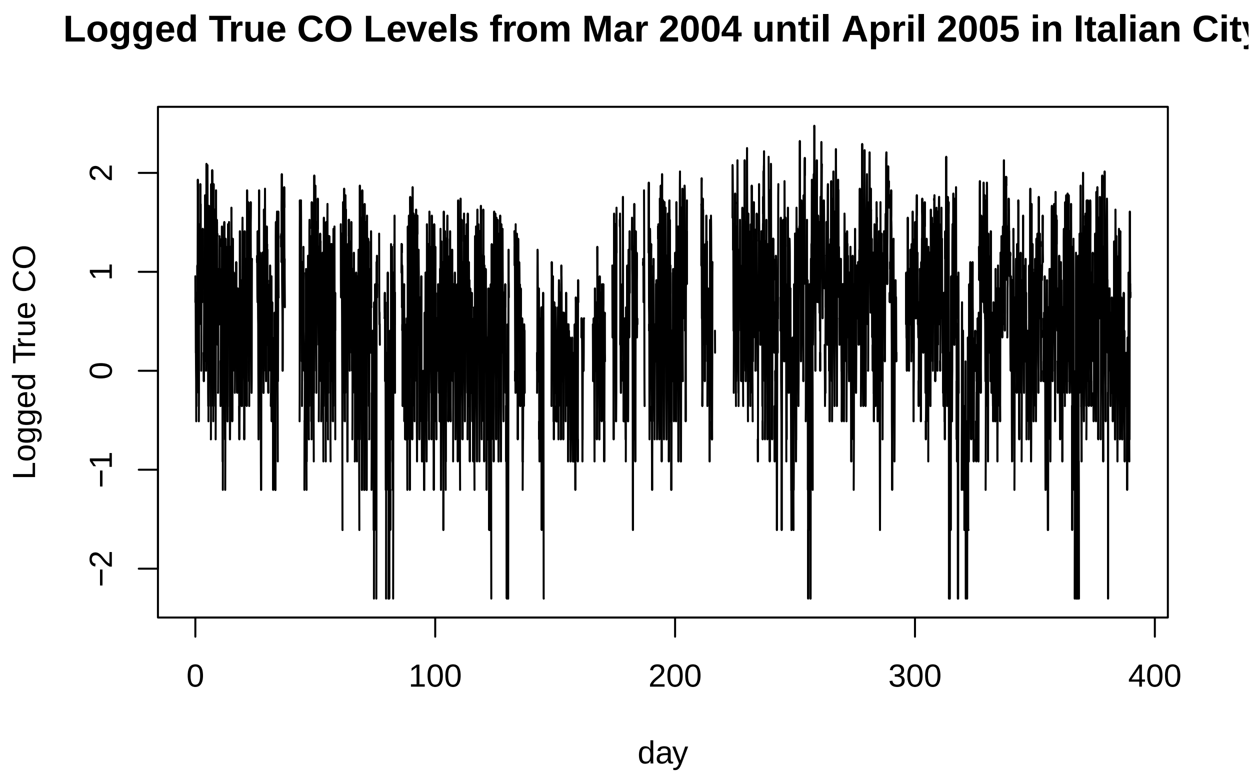
STSCI4550Project

```r
####################################
# State-Space Model for Air Data
####################################

library(MARSS)
load('~/Downloads/Air.RData')
#time(Air)

# constructing a time plot of the data
# days range from first 14 days in March 2004
plot( x = time(Air)[1:336],
      y = log( as.data.frame(Air)[1:336,"True CO"] ),
      xlab='day',
      ylab = "True CO",
      main='Logged True CO Levels in 14 days in Mar 2004 in Italian City',
      type = "l")
```




```r
# constructing a time plot of the data
# days range from March 2004 until April 2005
plot( x = time(Air),
      y = log( as.data.frame(Air)[,"True CO"] ),
      xlab='day',  ylab = "Logged True CO",
```

```r
     main='Logged True CO Levels from Mar 2004 until April 2005 in Italian City',
     type = "l")
```




```r
# Create a 1 x 336 matrix containing hourly measurements of the
# log of True CO over the first 2 weeks (336/24=14):

ind <- 1:336
T <- length(ind)
y <- t(as.matrix(log(Air[ind,1])))



# Center/scale the exogenous variables:

temp <- scale(Air[ind, 3])
rhum <- scale(Air[ind, 4])
hum <- scale(Air[ind, 5])
```

```r
### State space model specification in MARSS format:

# I'll provide some code which will help you to set up the
# state space model in MARSS format. You will have to fill
# in the rest.

# Constructing B (Phi in S&S notation):

# We first create a matrix of zeros (but we need to use a list
# because the matrix will not just contain numbers).

Blist <- list()
```

```r
for(i in 1:(24^2)){
  Blist[[i]] <- 0
}
B <- matrix(Blist, nrow=24)
B[1,1] <- 'phi' # Top left entry is 'phi'
B[2,] <- c(0, rep(-1, 23)) # Filling in the second row
# We then fill in the remaining rows:
for(i in 3:24){
  B[i,i-1] <- 1
}

# You need to construct Q. You can do this in much the
# same way we constructed B.
Qlist <- list()
for(i in 1:(24^2)){
  Qlist[[i]] <- 0
}
Q <- matrix(Qlist, nrow=24)
Q[1,1] <- "q11" # Top left entry is 'phi'
Q[2,2] <- "q22"


# You need to construct x0 and V0
# (mu0 and Sigma0 in S&S notation, respectively).
x0 <- as.matrix( rep(0,24) )
V0 <- diag(10,24)


# U has no equivalent in S&S notation and the MARSS
# default value is not zero, so you need to set U equal
# to a zero vector of appropriate dimension.
U <- as.matrix( rep(0,24) )

# You need to construct the measurement matrix Z
# (A in S&S notation).
Z <- rep(0,24)
Z[1:2] <- 1
Z <- t(Z)

# Constructing D (Gamma in S&S notation):
D <- matrix(c('beta0', 'beta1', 'beta2', 'beta3'), nrow=1)

# Constructing d:

# We need to pass in the values of the exogenous variables at
# every point in time, so we use an array object as follows:

d <- array(rep(0, 4*length(ind)), dim=c(4,1,length(ind)))
for(t in ind){
  d[,,t] <- c(1, temp[t], rhum[t], hum[t])
}

# You need to construct R (this is also R in S&S notation).
```

```r
R <- as.matrix( "r11" )

# A has no equivalent in S&S notation and the MARSS
# default value is not zero, so you need to set A equal
# to a zero vector of appropriate dimension.
A <- as.matrix( 0 )

# Construct the model list as follows:
model.list <- list(B=B, Q=Q, x0=x0, V0=V0, U=U,
                   Z=Z, D=D, d=d, R=R, A=A, tinitx=0)

### Next you need to fit the model using the MARSS function.

# As we did in notes5code.R (e.g. with the biomarker data),
# you will need to first run the EM algorithm using the option
# method='kem' to get initial estimates and then pass these
# as initial values with method='BFGS'. During the first MARSS
# call (with method='kem') you should also pass in the argument
# control=list(minit=2, maxit=2) so that the EM algorithm only
# runs for a small number of iterations (even 2 iterations is fine).
fit_kem <- MARSS(y=y, model=model.list, control=list(minit=2, maxit=2), method='kem')
```

```
## Warning! Reached maxit before parameters converged. Maxit was 2.
##  abstol not reached and no log-log test info since maxit less than min.iter.conv.test.
##
## MARSS fit is
## Estimation method: kem
## Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
## WARNING: Abstol convergence only no info on log-log convergnece.
##  No log-log convergence info because maxit (=2) < min.iter.conv.test (=15).
##  The likelihood and params might not be at the ML values.
##  Try setting control$maxit higher.
## Log-likelihood: -178.5309
##
##
##          Estimate
## R.r11     0.03672
## B.phi     0.96602
## Q.q11     0.03847
## Q.q22     0.02577
## D.beta0   0.00063
## D.beta1   0.00145
## D.beta2   0.00457
## D.beta3   0.00393
## Initial states (x0) defined at t=0
##
## Standard errors have not been calculated.
## Use MARSSparamCIs to compute CIs and bias estimates.
##
## Convergence warnings
##  No convergence testing performed.
```

```r
fit_bfgs_with_kem <- MARSS(y=y, model=model.list, method = "BFGS",
                           inits = fit_kem)
```

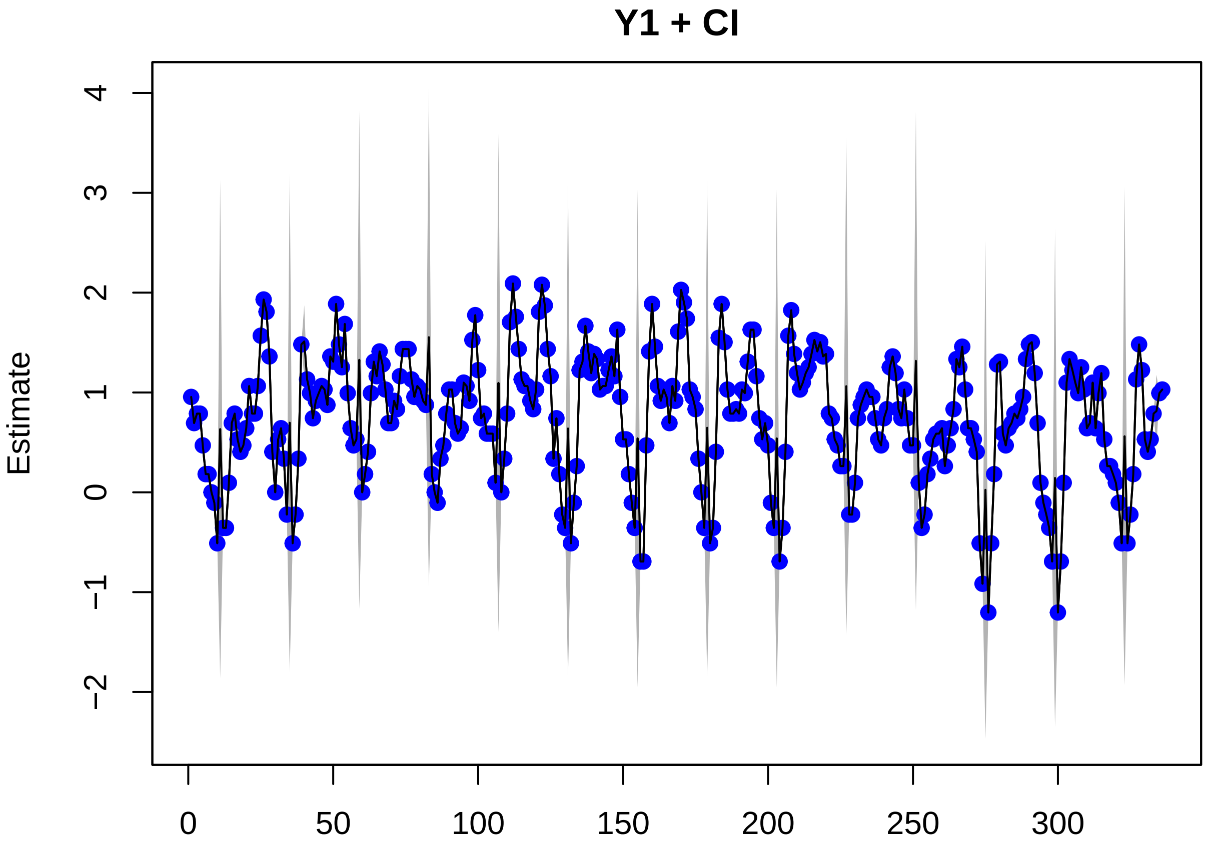
```
## Success! Converged in 60 iterations.
## Function MARSSkfas used for likelihood calculation.
##
## MARSS fit is
## Estimation method: BFGS
## Estimation converged in 60 iterations.
## Log-likelihood: -64.08381
## AIC: 144.1676   AICc: 144.6306
##
##          Estimate
## R.r11   9.15e-12
## B.phi   7.83e-01
## Q.q11   5.13e-02
## Q.q22   3.12e-13
## D.beta0 7.55e-01
## D.beta1 2.59e-02
## D.beta2 6.63e-02
## D.beta3 2.55e-02
## Initial states (x0) defined at t=0
##
## Standard errors have not been calculated.
## Use MARSSparamCIs to compute CIs and bias estimates.
```

```r
# confidence intervals
MARSSparamCIs(fit_bfgs_with_kem)
```

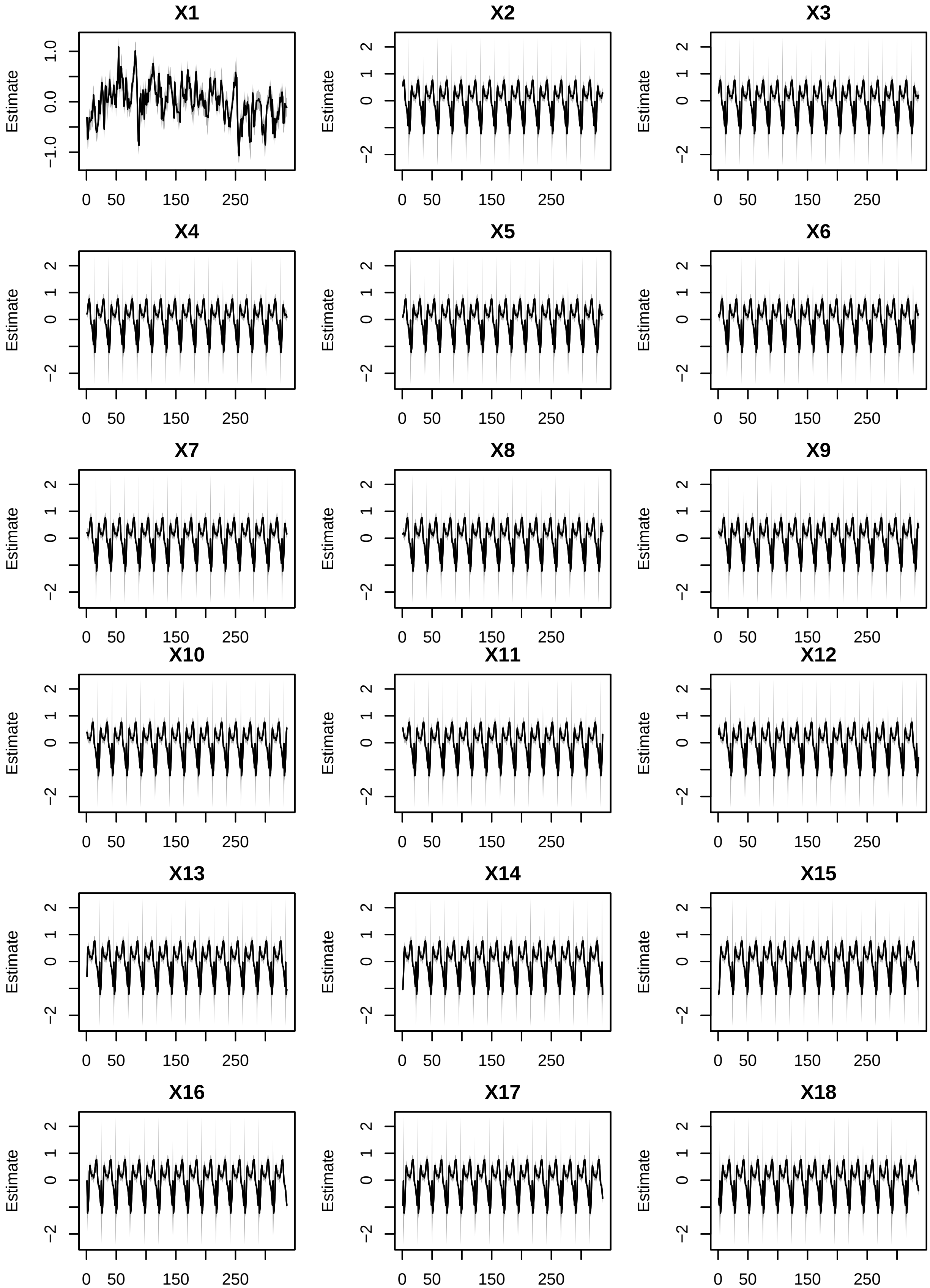
```
##
## MARSS fit is
## Estimation method: BFGS
## Estimation converged in 60 iterations.
## Log-likelihood: -64.08381
## AIC: 144.1676   AICc: 144.6306
##
##           ML.Est  Std.Err    low.CI    up.CI
## R.r11   9.15e-12 0.005062 -0.009922 0.009922
## B.phi   7.83e-01 0.045043  0.694973 0.871537
## Q.q11   5.13e-02 0.008757  0.034089 0.068417
## Q.q22   3.12e-13 0.000159 -0.000312 0.000312
## D.beta0 7.55e-01 0.146444  0.467846 1.041897
## D.beta1 2.59e-02 0.140957 -0.250404 0.302136
## D.beta2 6.63e-02 0.135973 -0.200172 0.332831
## D.beta3 2.55e-02 0.063844 -0.099626 0.150639
## Initial states (x0) defined at t=0
##
## CIs calculated at alpha = 0.05 via method=hessian
```

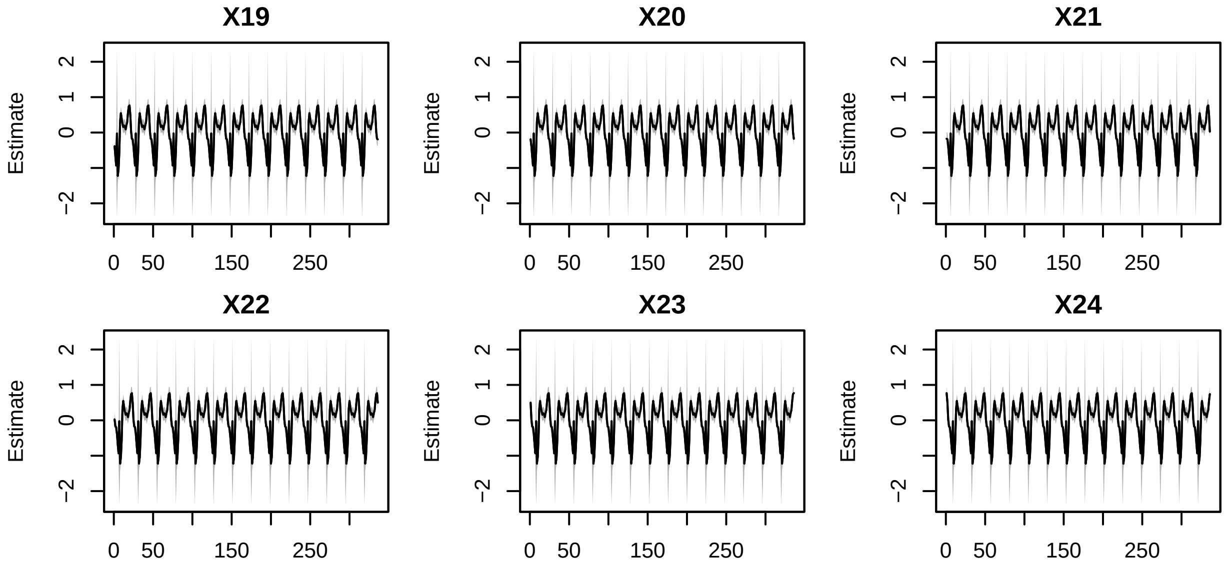
```r
# plot of state estimates
plot(fit_bfgs_with_kem)
```

```
## MARSSresiduals.tt1 reported warnings. See msg element of returned residuals object.
```

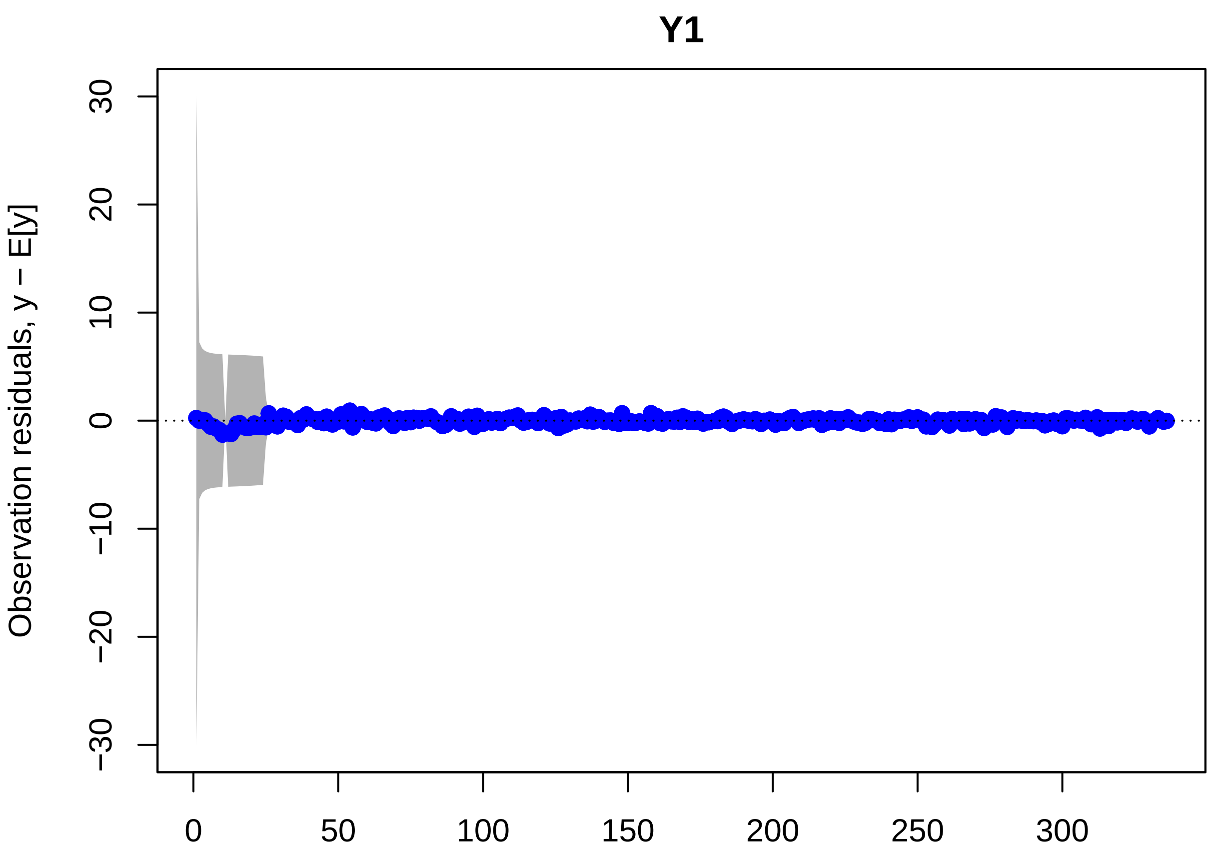



```
## plot type =  fitted.ytT  Observations with fitted values
## Hit <Return> to see next plot (q to exit):
```

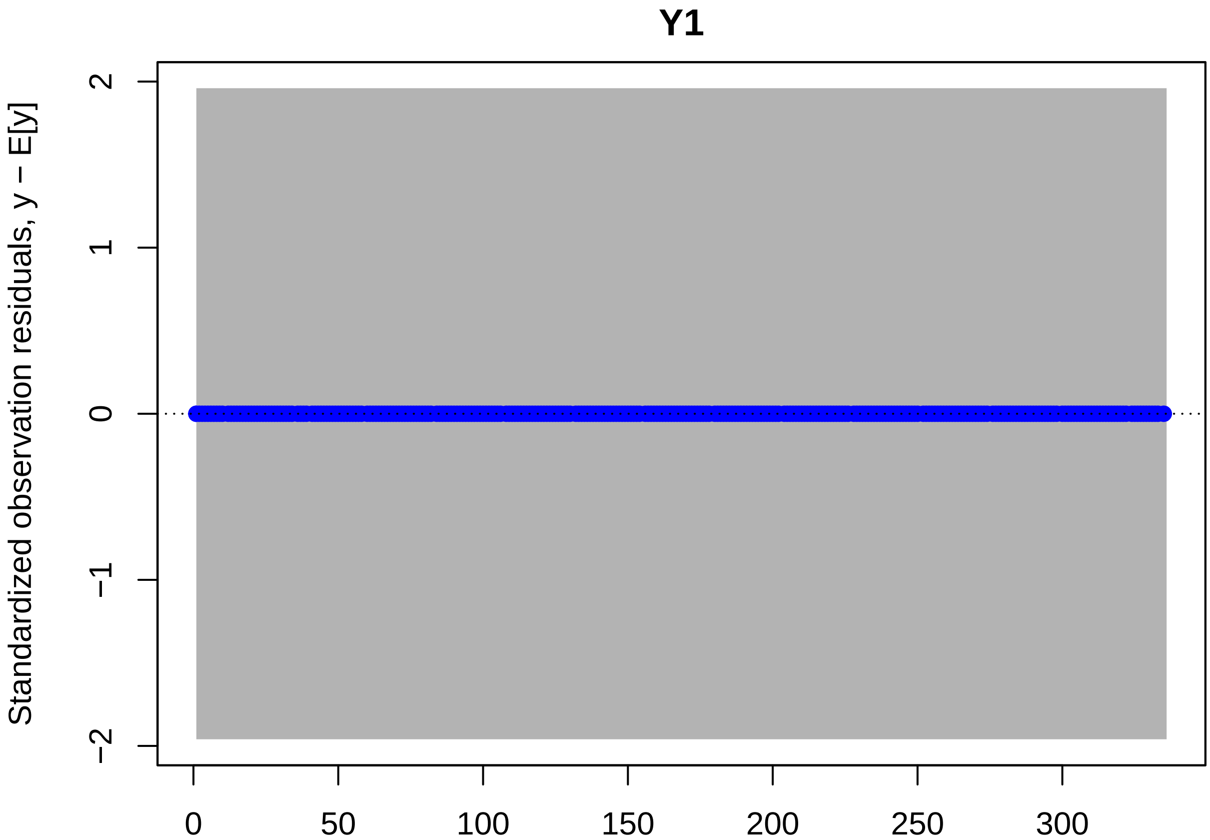

X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
Estimate
0
50
150
250



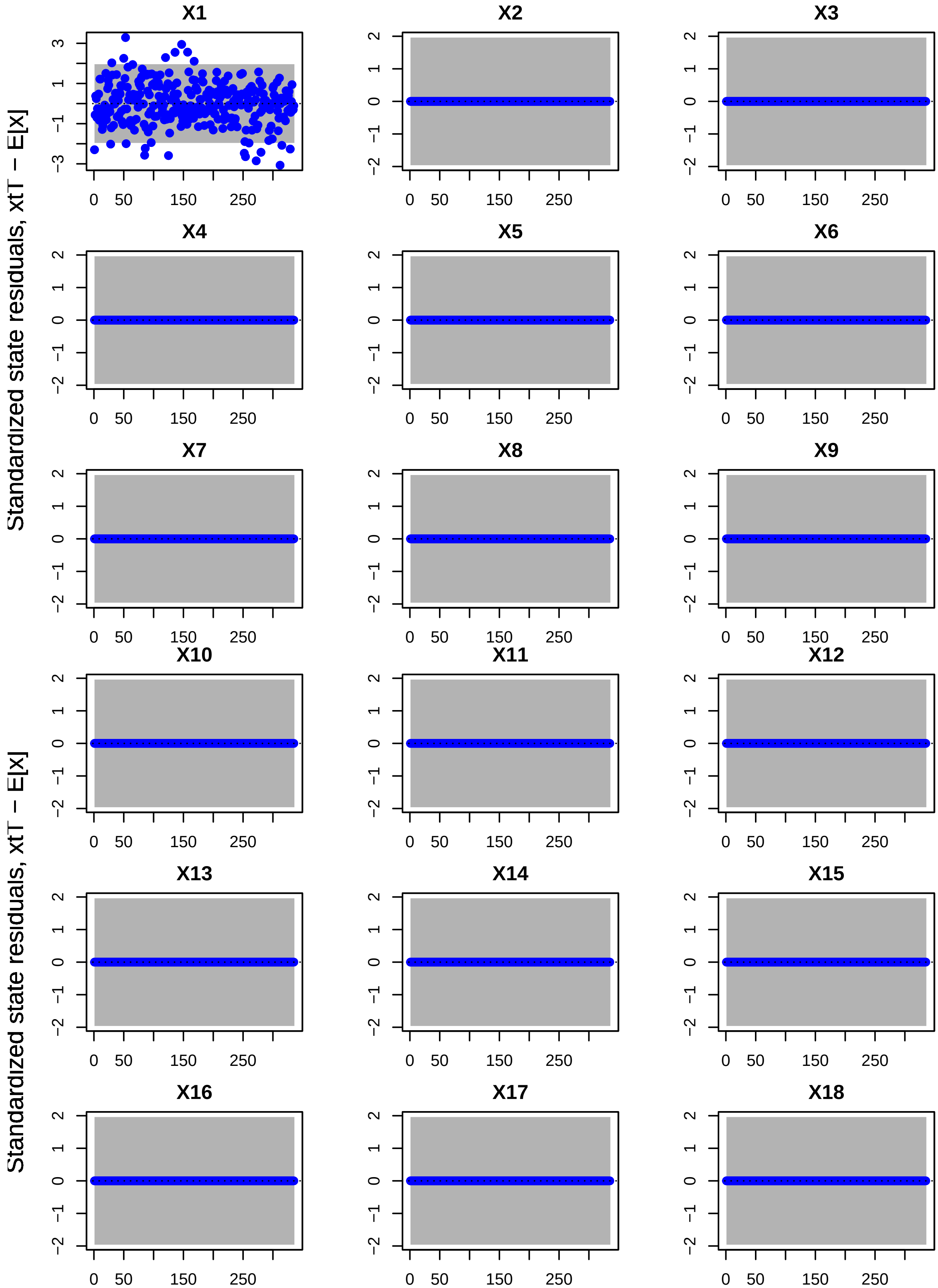
plot type = xtT Estimated states
Hit <Return> to see next plot (q to exit):

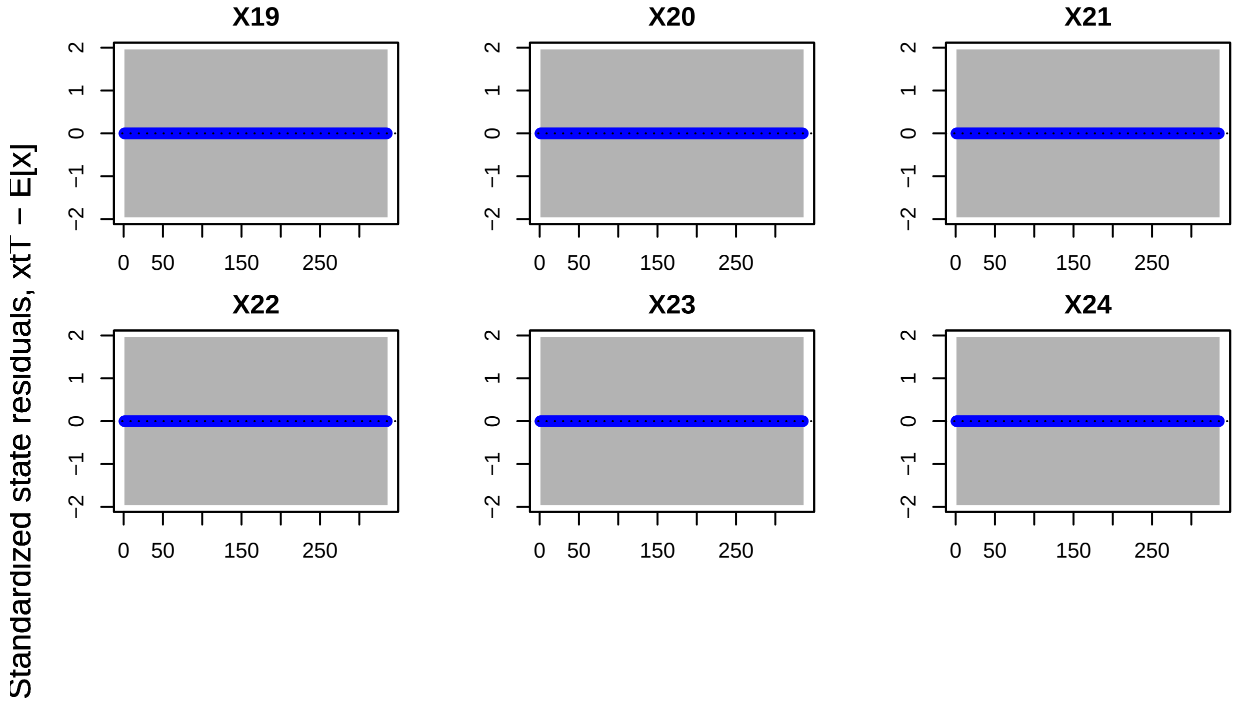


plot type = model.resids.ytt1
Hit <Return> to see next plot (q to exit):

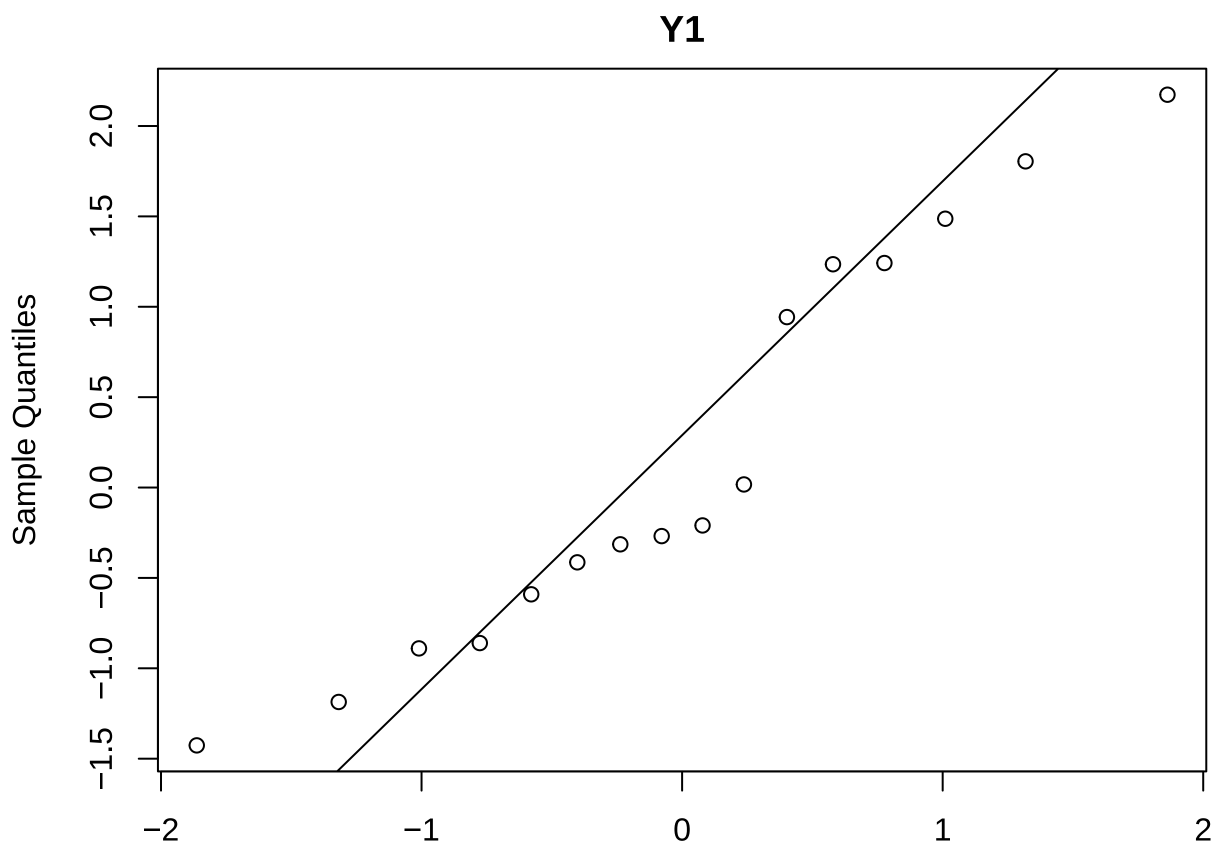



```
## plot type = std.model.resids.ytT
## Hit <Return> to see next plot (q to exit):
```

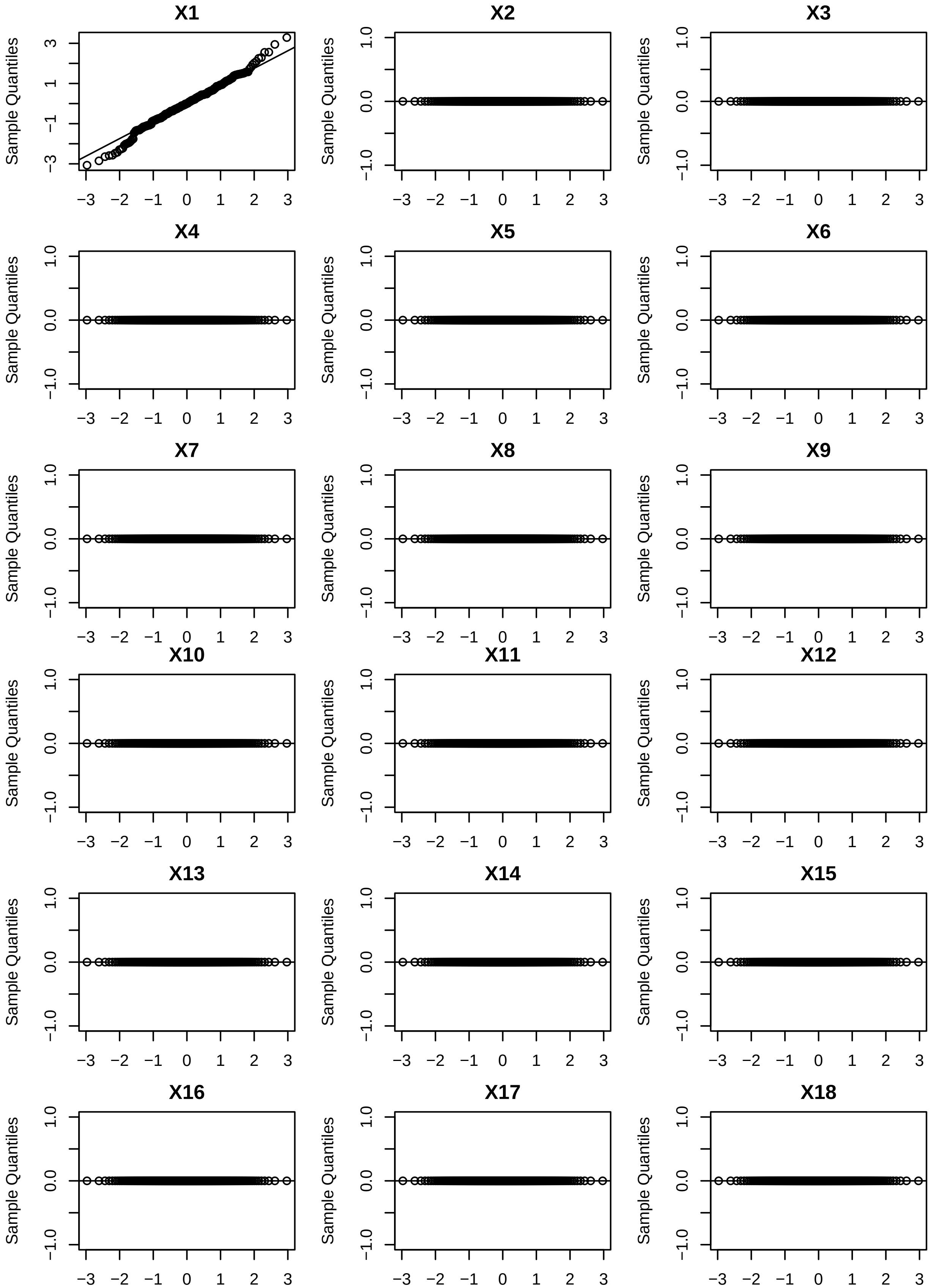

X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
Standardized state residuals, xtT − E[x]
Standardized state residuals, xtT − E[x]
0 50 150 250

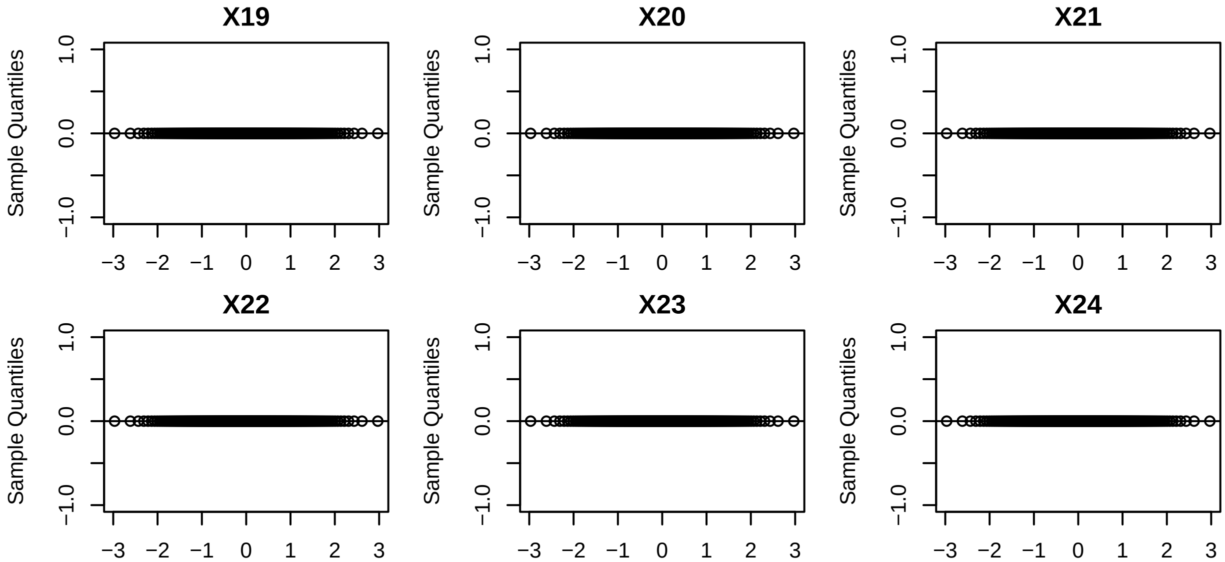



```
## plot type = std.state.resids.xtT
## Hit <Return> to see next plot (q to exit):
```

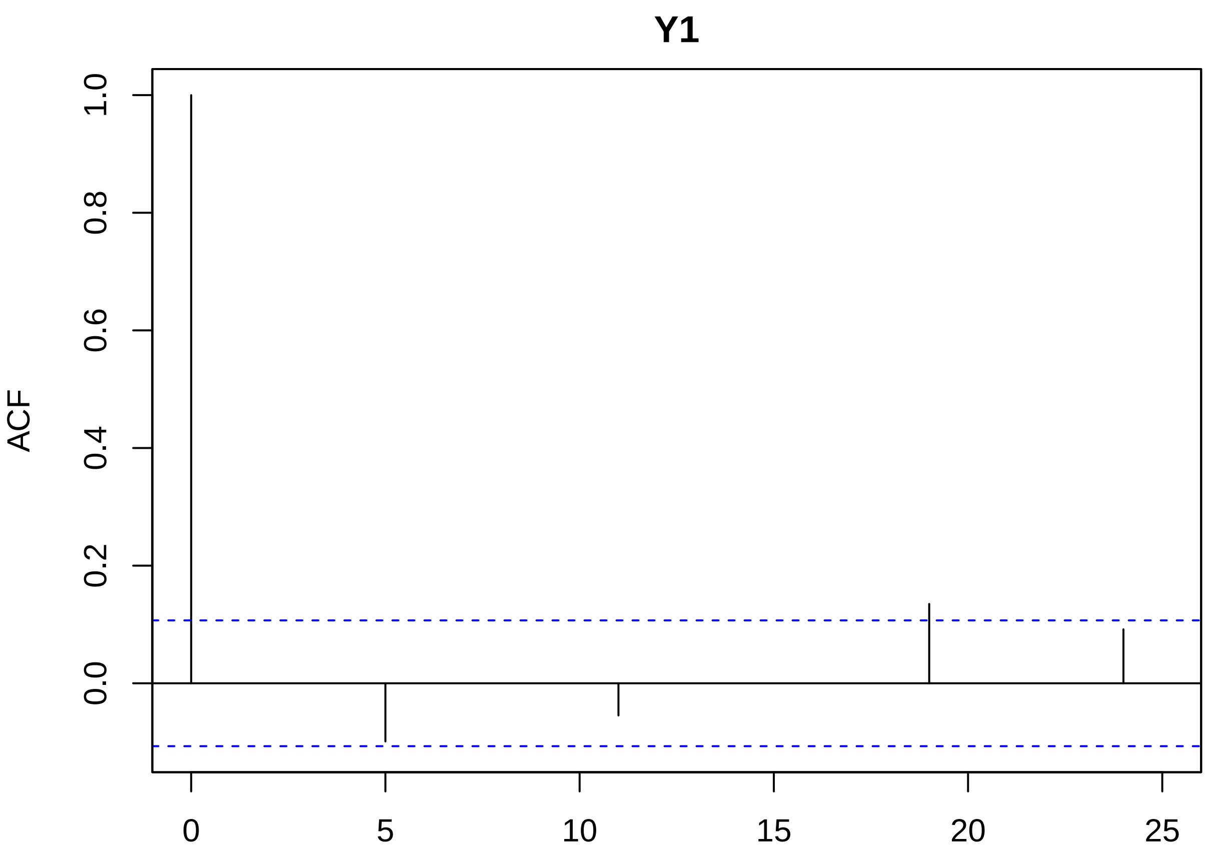



```
## plot type = qqplot.std.model.resids.ytt1
## Hit <Return> to see next plot (q to exit):
```


X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
Sample Quantiles




```
## plot type = qqplot.std.state.resids.xtT
## Hit <Return> to see next plot (q to exit):
```




```
## plot type = acf.std.model.resids.ytt1
```

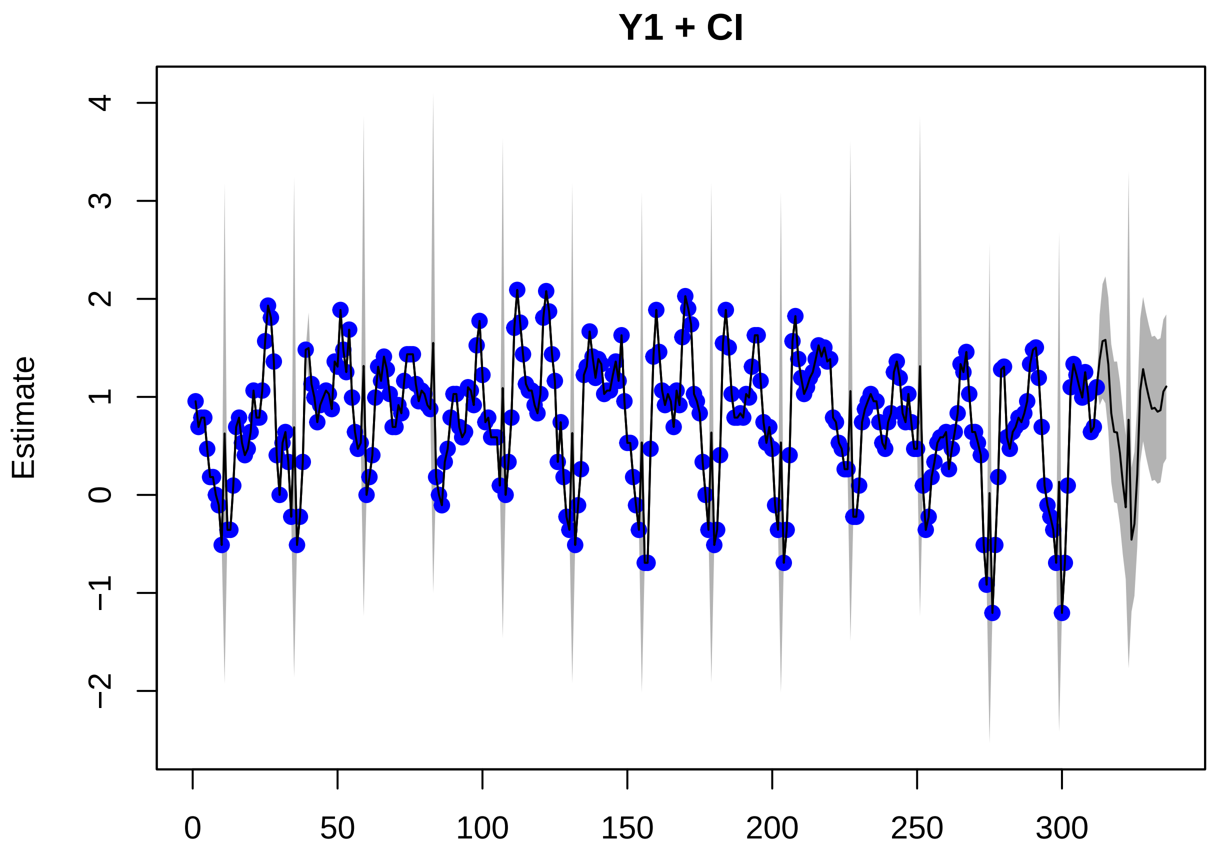
```r
## FORECASTING

# Replace h values at the end of the y series (or the subset
# of the y series we're considering) with NAs  which MARSS
# will then forecast:
```

```r
h <- 24
ywithNAs <- y
ywithNAs[,(T-h+1):T] <- rep(NA, h)

fit_bfgs_with_kem <- MARSS(y=ywithNAs, model=model.list, method = "BFGS",
                           inits = fit_kem)
```

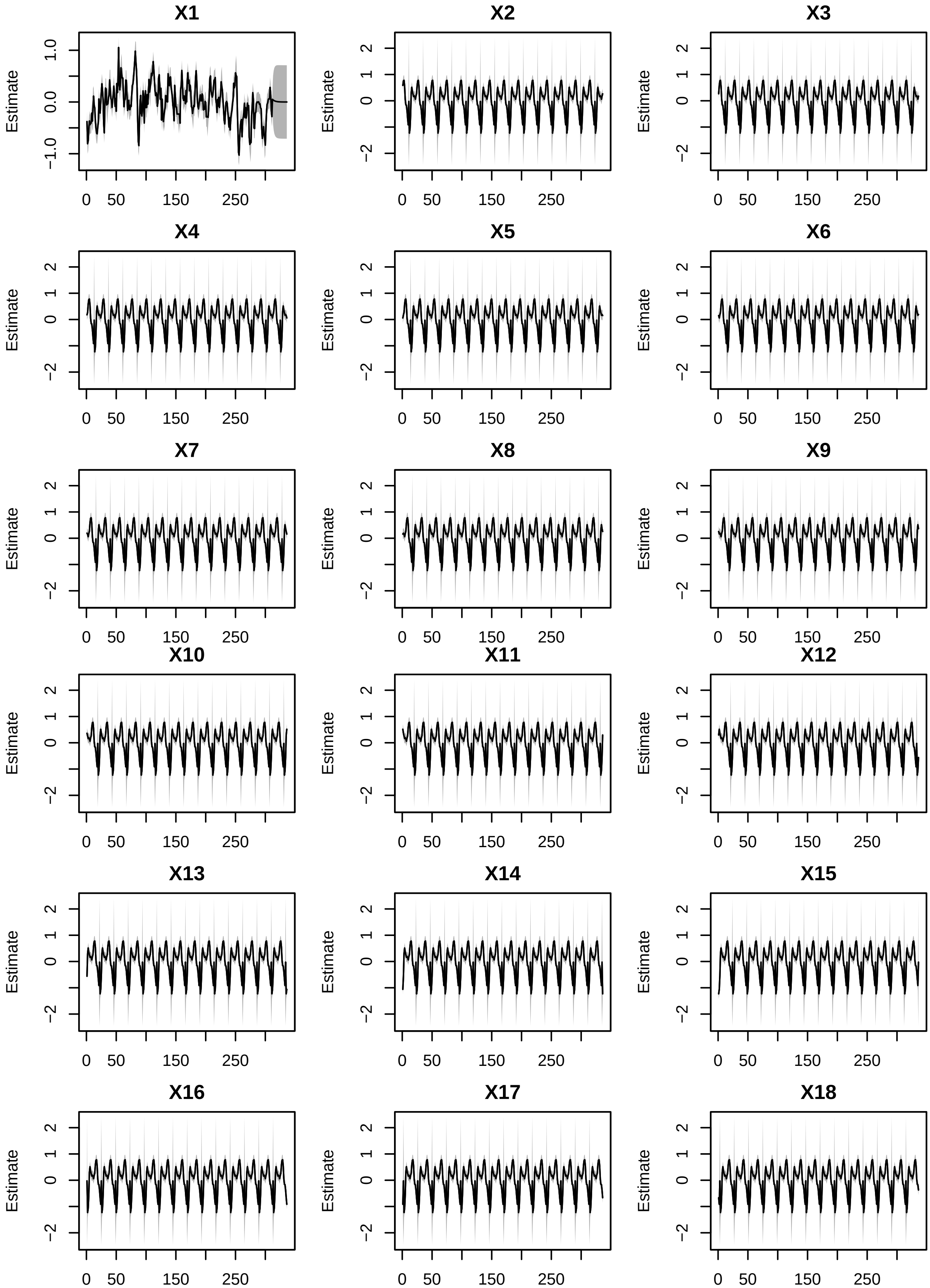
```
## Success! Converged in 131 iterations.
## Function MARSSkfas used for likelihood calculation.
##
## MARSS fit is
## Estimation method: BFGS
## Estimation converged in 131 iterations.
## Log-likelihood: -63.71363
## AIC: 143.4273   AICc: 143.9255
##
##         Estimate
## R.r11   2.18e-12
## B.phi   7.82e-01
## Q.q11   5.11e-02
## Q.q22   6.76e-11
## D.beta0 7.75e-01
## D.beta1 1.33e-02
## D.beta2 4.38e-02
## D.beta3 1.94e-02
## Initial states (x0) defined at t=0
##
## Standard errors have not been calculated.
## Use MARSSparamCIs to compute CIs and bias estimates.
```

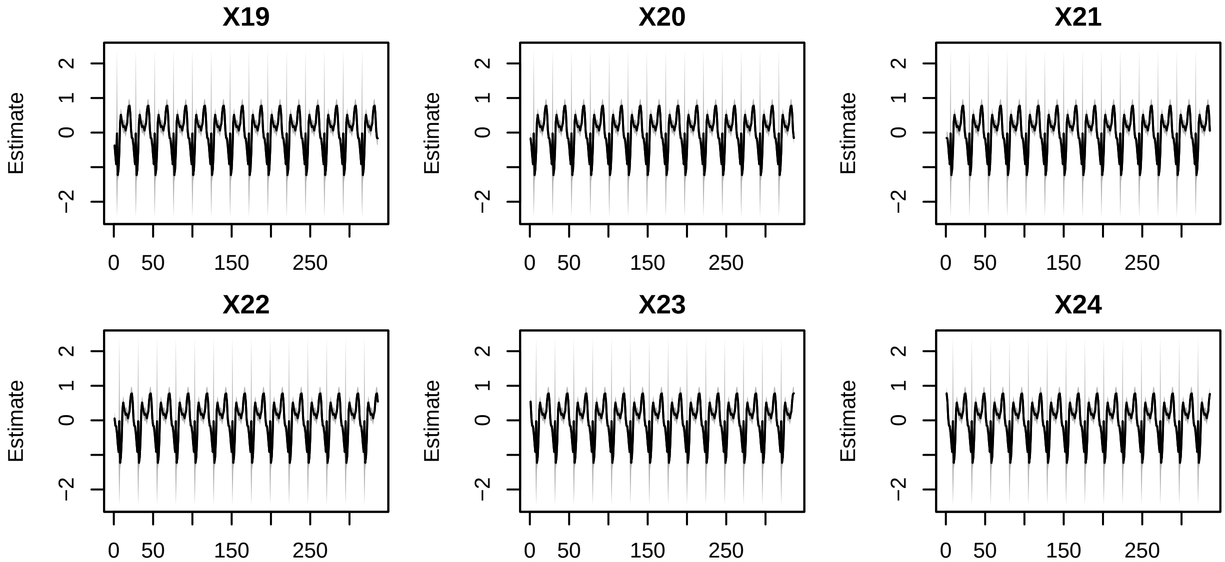
```r
plot(fit_bfgs_with_kem)
```

```
## MARSSresiduals.tt1 reported warnings. See msg element of returned residuals object.
```

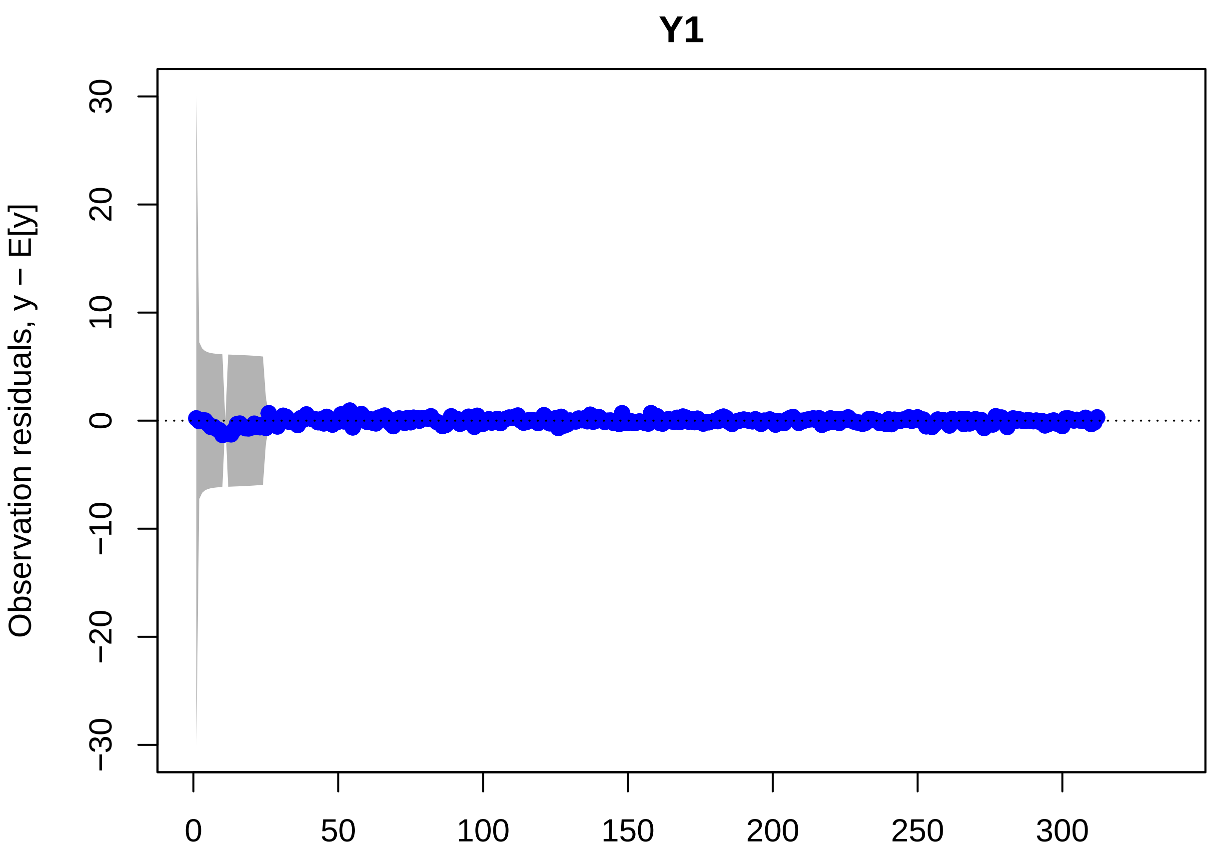



```
## plot type =  fitted.ytT  Observations with fitted values
## Hit <Return> to see next plot (q to exit):
```

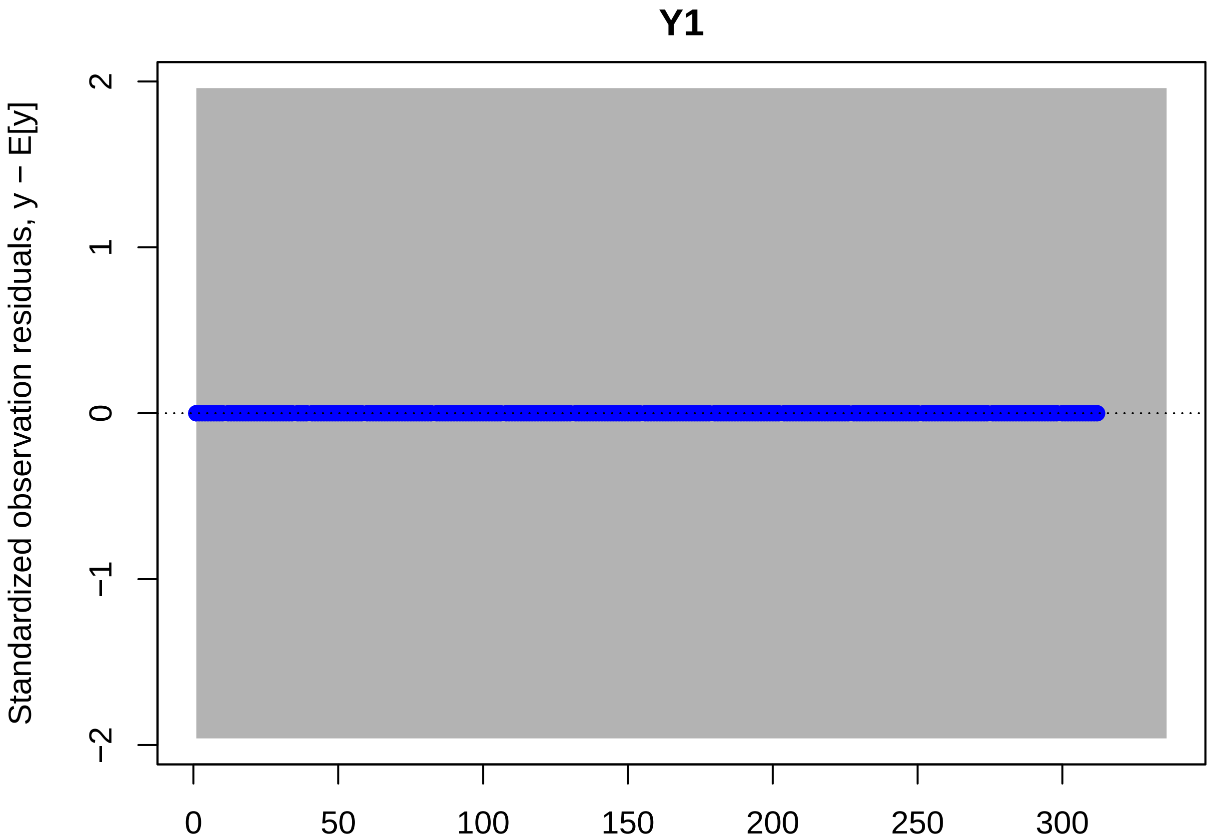

X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
Estimate
0
50
150
250
−2
0
1
2
−1.0
0.0
1.0



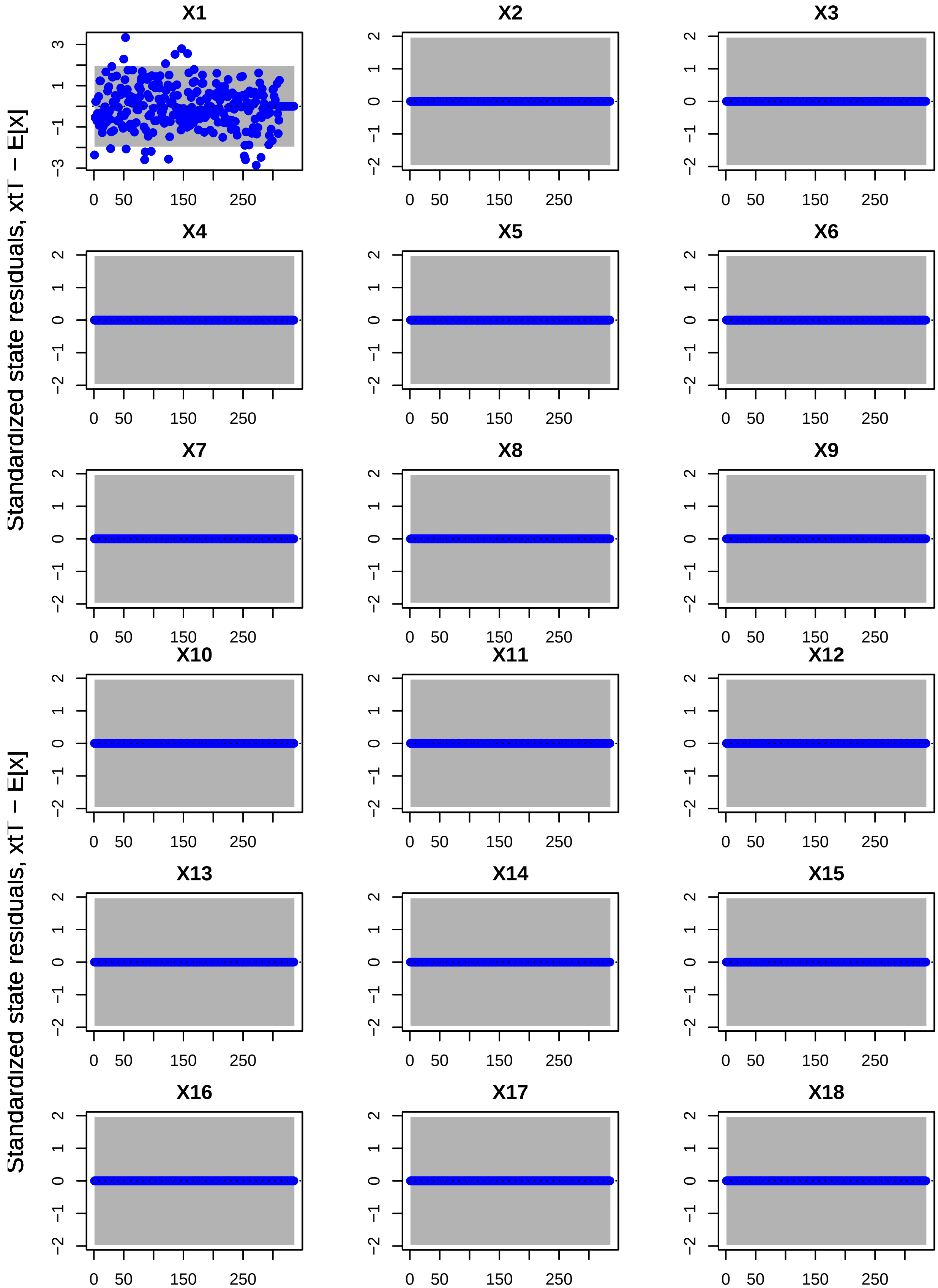

```
## plot type = xtT Estimated states
## Hit <Return> to see next plot (q to exit):
```

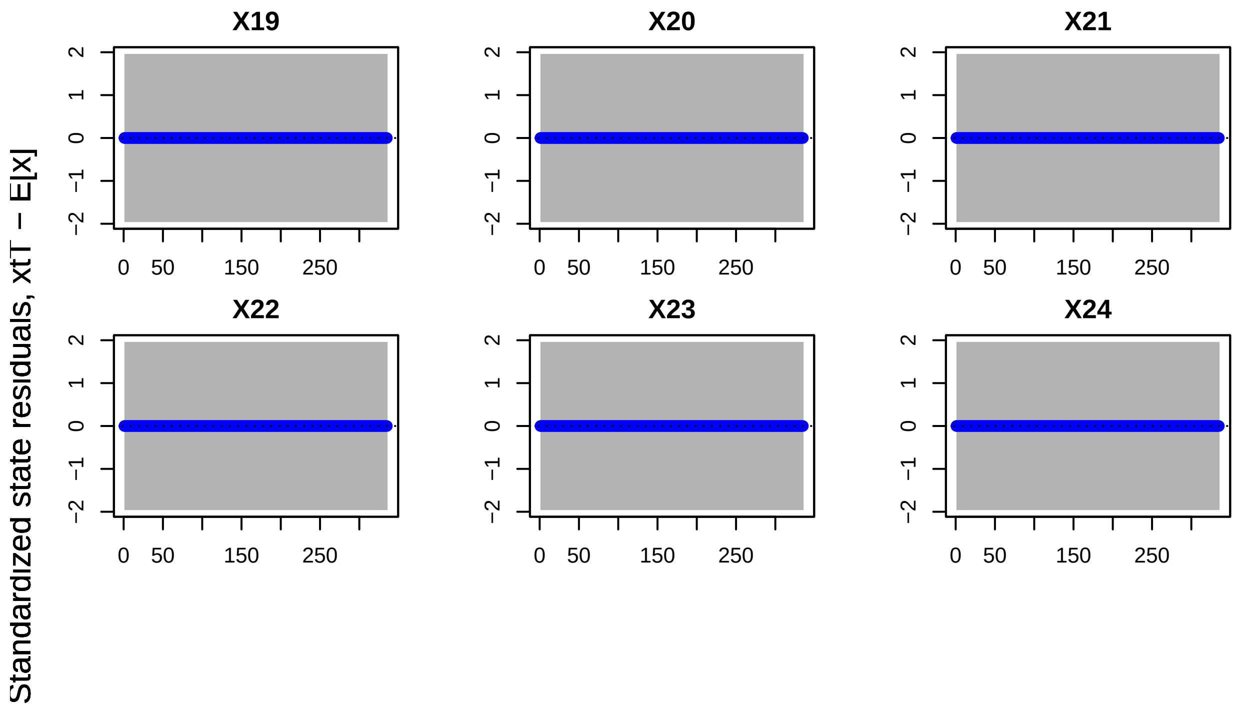



```
## plot type = model.resids.ytt1
## Hit <Return> to see next plot (q to exit):
```

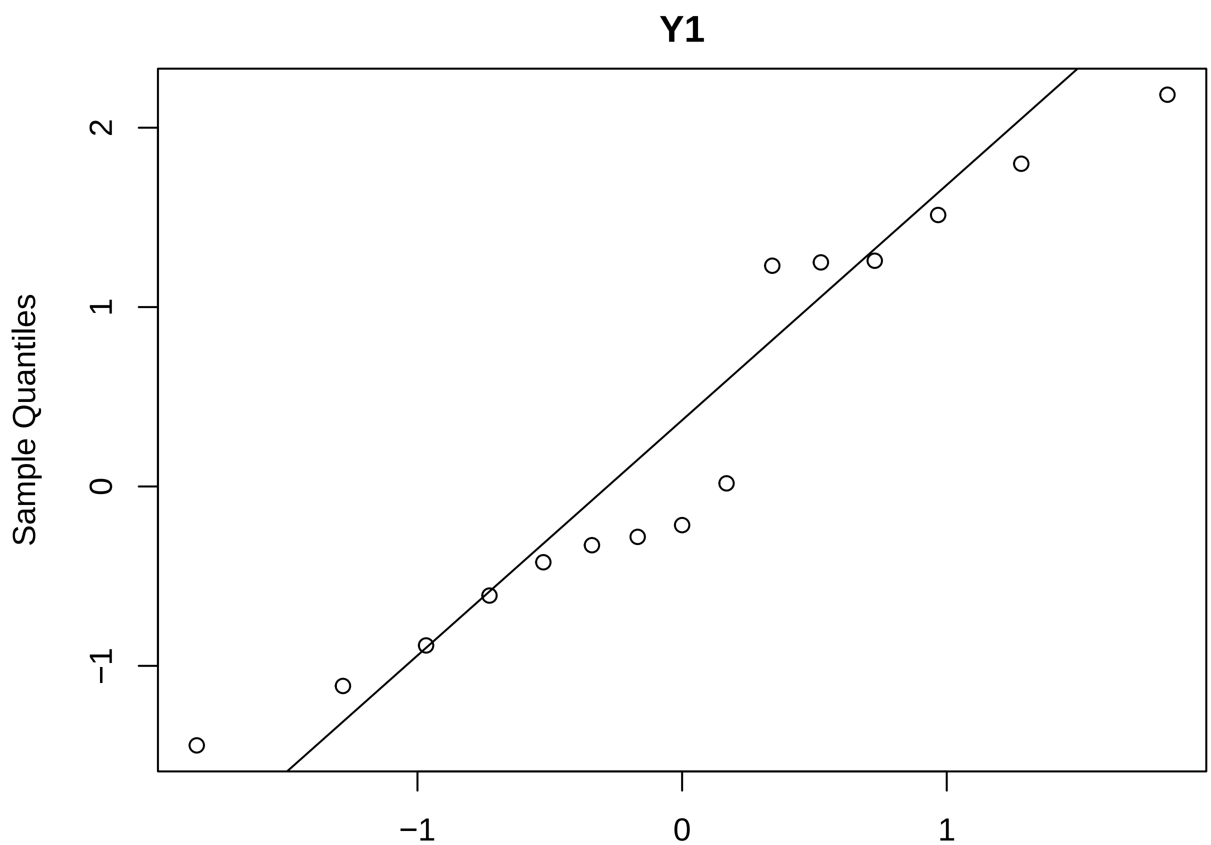



```
## plot type = std.model.resids.ytT
## Hit <Return> to see next plot (q to exit):
```

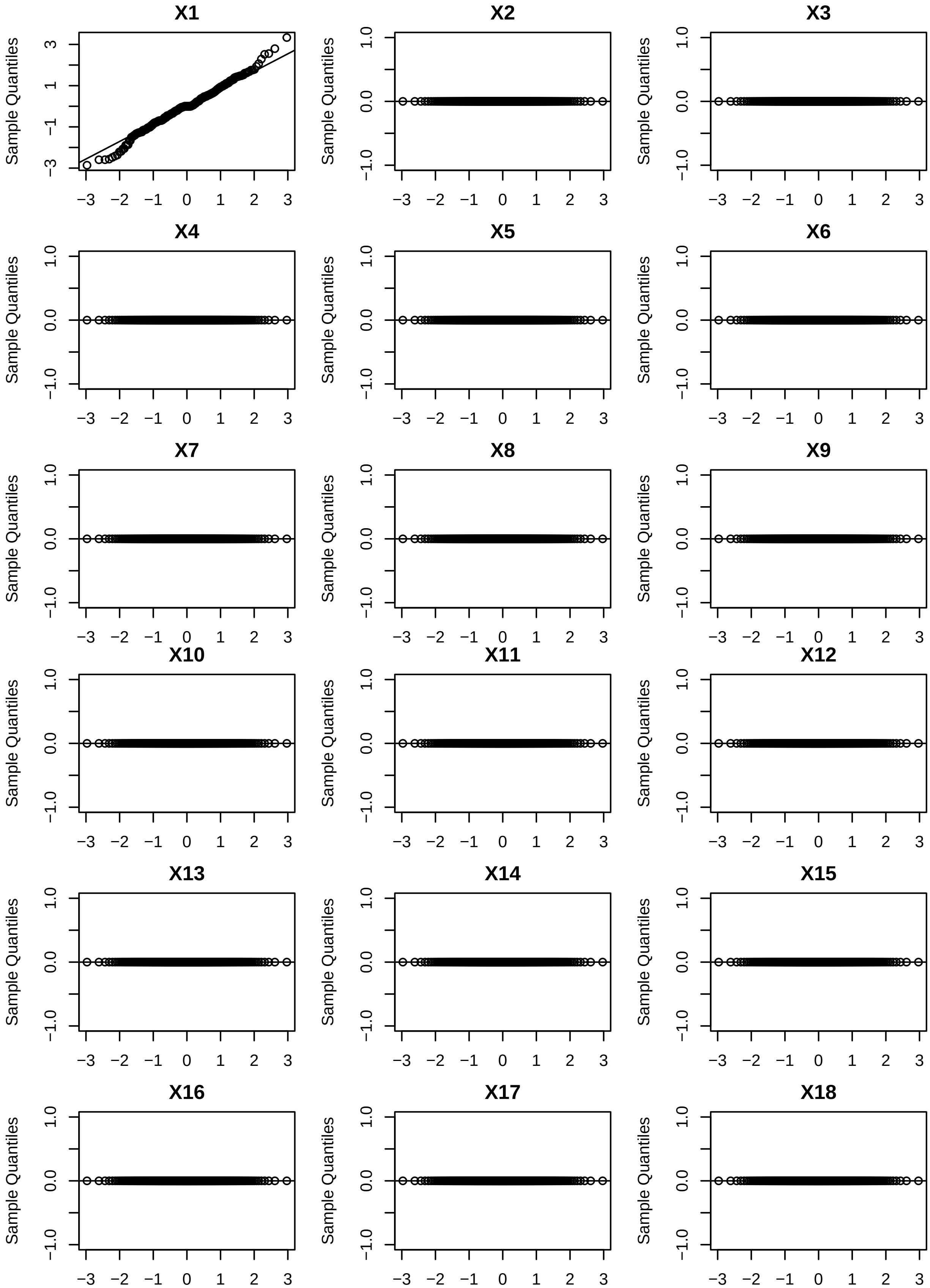

Standardized state residuals, xtT − E[x]
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
0
50
150
250

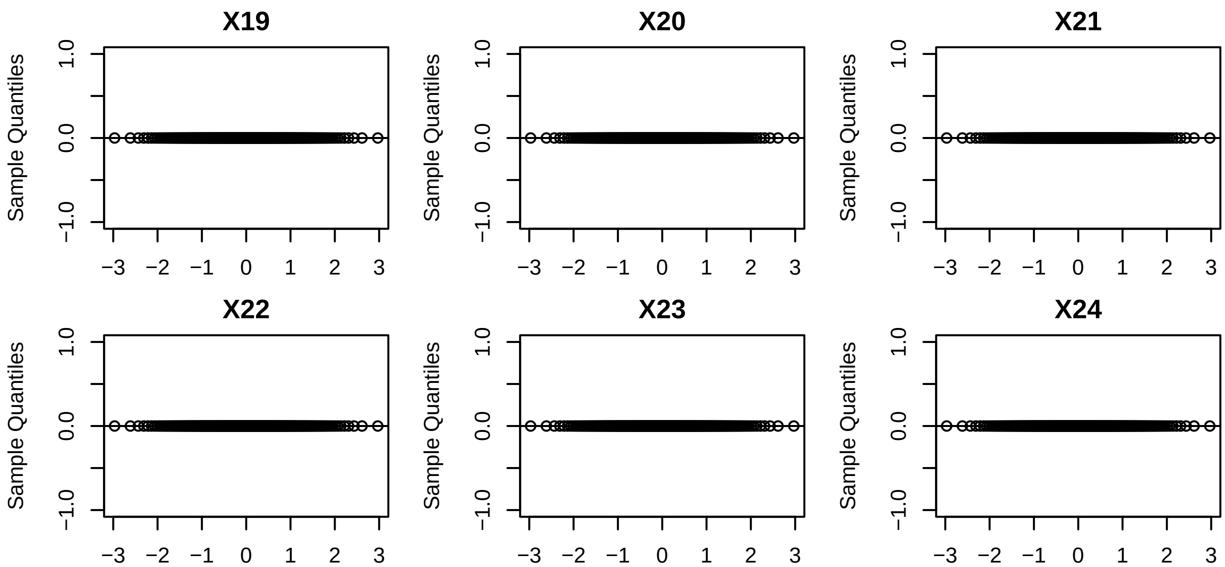



```
## plot type = std.state.resids.xtT
## Hit <Return> to see next plot (q to exit):
```

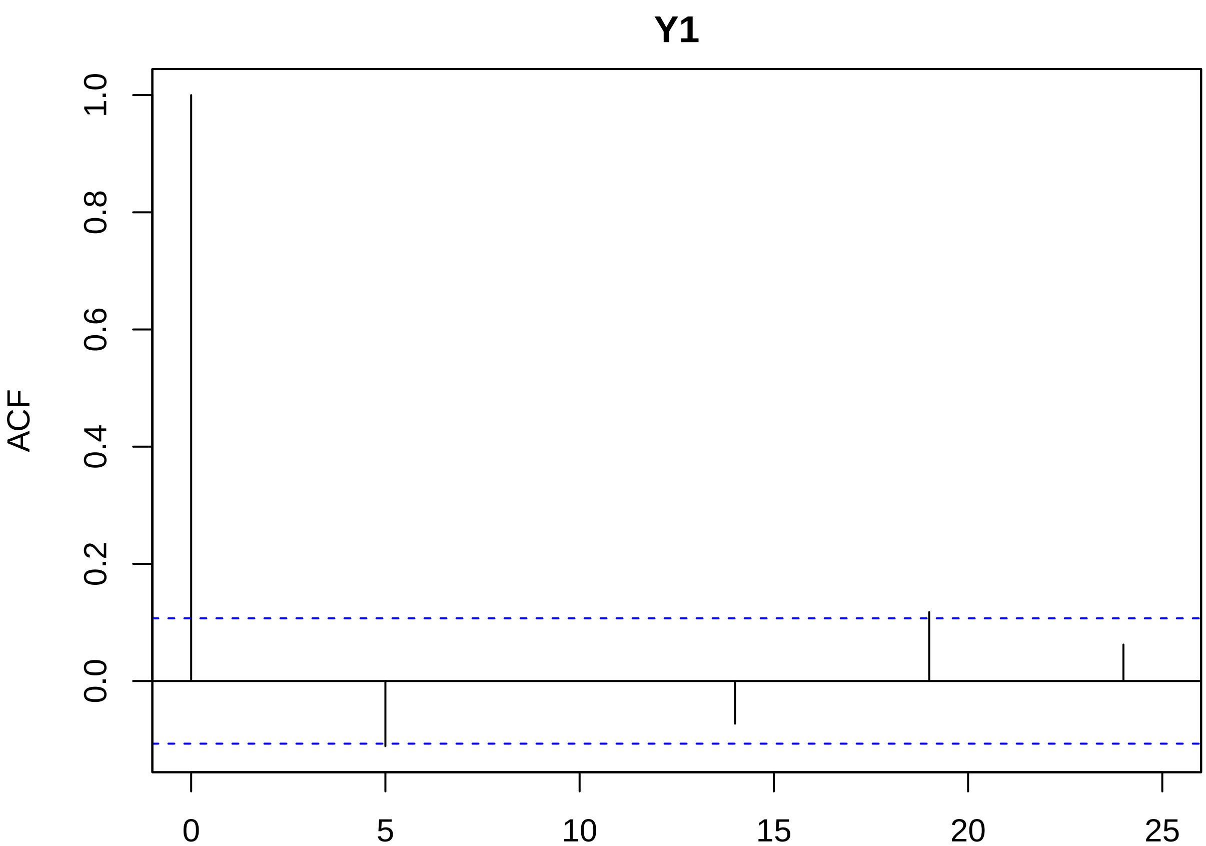



```
## plot type = qqplot.std.model.resids.ytt1
## Hit <Return> to see next plot (q to exit):
```


X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
Sample Quantiles




```
## plot type = qqplot.std.state.resids.xtT
## Hit <Return> to see next plot (q to exit):
```




```
## plot type = acf.std.model.resids.ytt1
```

```r
y_forecasted <- fitted.values(fit_bfgs_with_kem)[(T-h+1):T,".fitted"]
y_actual <- y[(T-h+1):T]
to_remove <- which(is.na(y_actual) == TRUE)

rmse <- sqrt(mean((y_actual[-to_remove] - y_forecasted[-to_remove])^2))
```

```r
rmse
```

```
## [1] 0.3868232
```

```r
sqrt( mean( ((y_actual - y_forecasted)^2), na.rm=TRUE) )
```

```
## [1] 0.3868232
```

```r
#####################################
# Model 2: linear regression with ARMA errors#
#####################################

#####################################
# Initial analysis of the Air data #
#####################################

library(astsa) # For datasets and sarima function
library(forecast) # For Arima and forecast functions
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:astsa':
##
##     gas
```

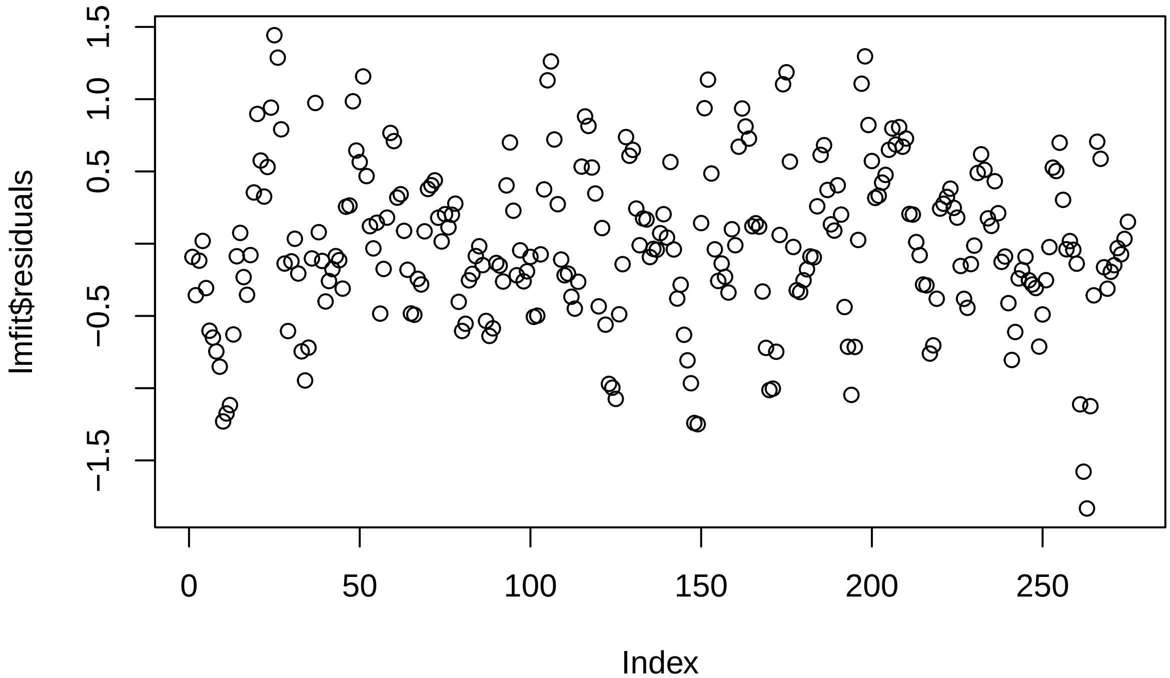
```r
load('~/Downloads/Air.RData')

# Create a 1 x 336 matrix containing hourly measurements of the
# log of True CO over the first 2 weeks (336/24=14):

ind <- 1:336
T <- length(ind)
y <- t(as.matrix(log(Air[ind,1])))

# Center/scale the exogenous variables (except for time):

temp <- scale(Air[ind, 3])
rhum <- scale(Air[ind, 4])
hum <- scale(Air[ind, 5])
time <- time(Air)

# Split the data set into a training set, which we will use to
# fit the models, and a test set, which we will use to evaluate
# forecast accuracy.

h <- 48 # Size of test set

ytrain <- y[1:(T-h)]
temptrain <- temp[1:(T-h)]
rhumtrain <- rhum[1:(T-h)]
humtrain <- hum[1:(T-h)]
timetrain <- time[1:(T-h)]
```

```r
ytest <- y[(T-h+1):T]
temptest <- temp[(T-h+1):T]
rhumtest <- rhum[(T-h+1):T]
humtest <- hum[(T-h+1):T]
timetest <- time[(T-h+1):T]
```

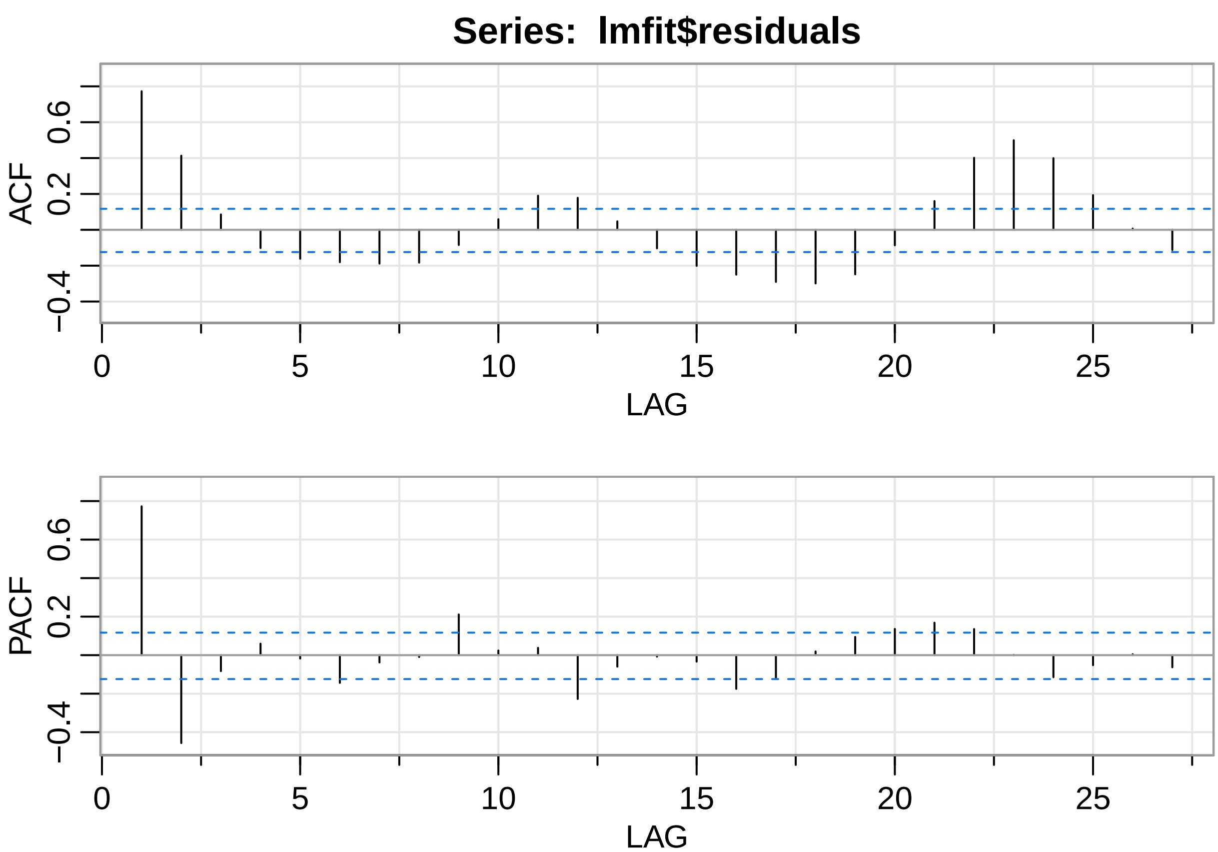
```r
# Fit linear regression model to obtain residuals
# and then look at the ACF/PACF plot to determine ARMA order:

lmfit <- lm(ytrain ~ timetrain + temptrain + rhumtrain + humtrain, na.action=na.omit)
summary(lmfit)
```

```
##
## Call:
## lm(formula = ytrain ~ timetrain + temptrain + rhumtrain + humtrain,
##     na.action = na.omit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.83230 -0.30916 -0.03928  0.35073  1.44270
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.02009    0.08397  12.149  < 2e-16 ***
## timetrain   -0.04642    0.01335  -3.476 0.000593 ***
## temptrain   -0.28789    0.16090  -1.789 0.074703 .
## rhumtrain   -0.60382    0.17396  -3.471 0.000603 ***
## humtrain     0.23625    0.07690   3.072 0.002341 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5591 on 270 degrees of freedom
##   (13 observations deleted due to missingness)
## Multiple R-squared:  0.1883, Adjusted R-squared:  0.1762
## F-statistic: 15.65 on 4 and 270 DF,  p-value: 1.561e-11
```

```r
plot(lmfit$residuals)
```




```r
acf2(lmfit$residuals)
```

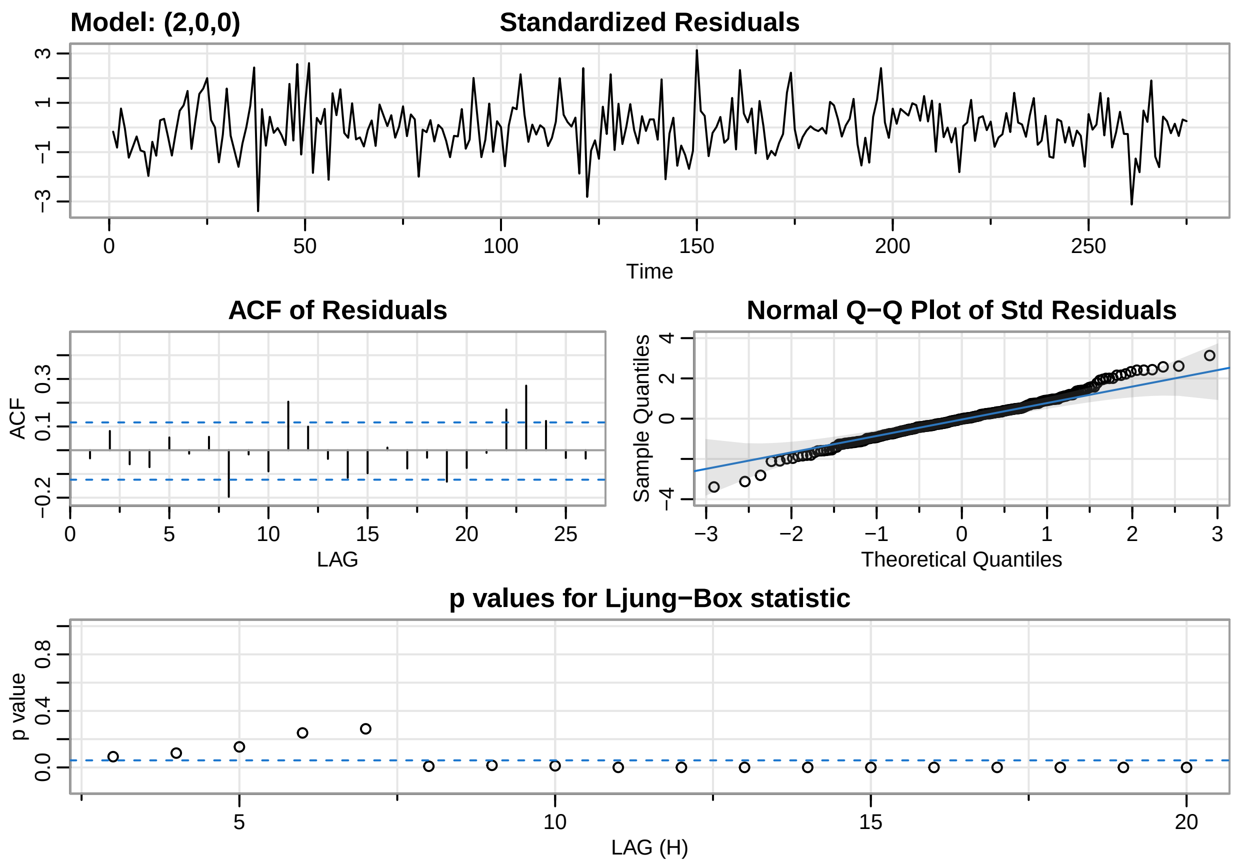



```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12]
## ACF  0.77  0.41  0.09 -0.10 -0.16 -0.18 -0.19 -0.18 -0.08  0.06  0.19  0.18
## PACF 0.77 -0.46 -0.08  0.06 -0.02 -0.14 -0.04 -0.01  0.21  0.02  0.04 -0.23
##      [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
```

```
## ACF   0.05 -0.10 -0.20 -0.25 -0.29 -0.30 -0.25 -0.09  0.16  0.40   0.5  0.40
## PACF -0.06 -0.01 -0.03 -0.17 -0.12  0.02  0.09  0.14  0.17  0.14   0.0 -0.11
##      [,25] [,26] [,27]
## ACF   0.19  0.01 -0.11
## PACF -0.05  0.00 -0.06
```

```
# Fit an AR(2) model to the residuals.
# Diagnostic plots don't look amazing, but don't look horrible either:

sarima(lmfit$residuals, p=2, d=0, q=0)
```

```
## initial  value -0.587757
## iter   2 value -0.747315
## iter   3 value -1.010212
## iter   4 value -1.125561
## iter   5 value -1.140931
## iter   6 value -1.160254
## iter   7 value -1.160284
## iter   8 value -1.160286
## iter   9 value -1.160287
## iter  10 value -1.160288
## iter  11 value -1.160289
## iter  11 value -1.160289
## iter  11 value -1.160289
## final  value -1.160289
## converged
## initial  value -1.160167
## iter   2 value -1.160174
## iter   3 value -1.160175
## iter   4 value -1.160176
## iter   5 value -1.160177
## iter   6 value -1.160177
## iter   6 value -1.160177
## iter   6 value -1.160177
## final  value -1.160177
## converged
```

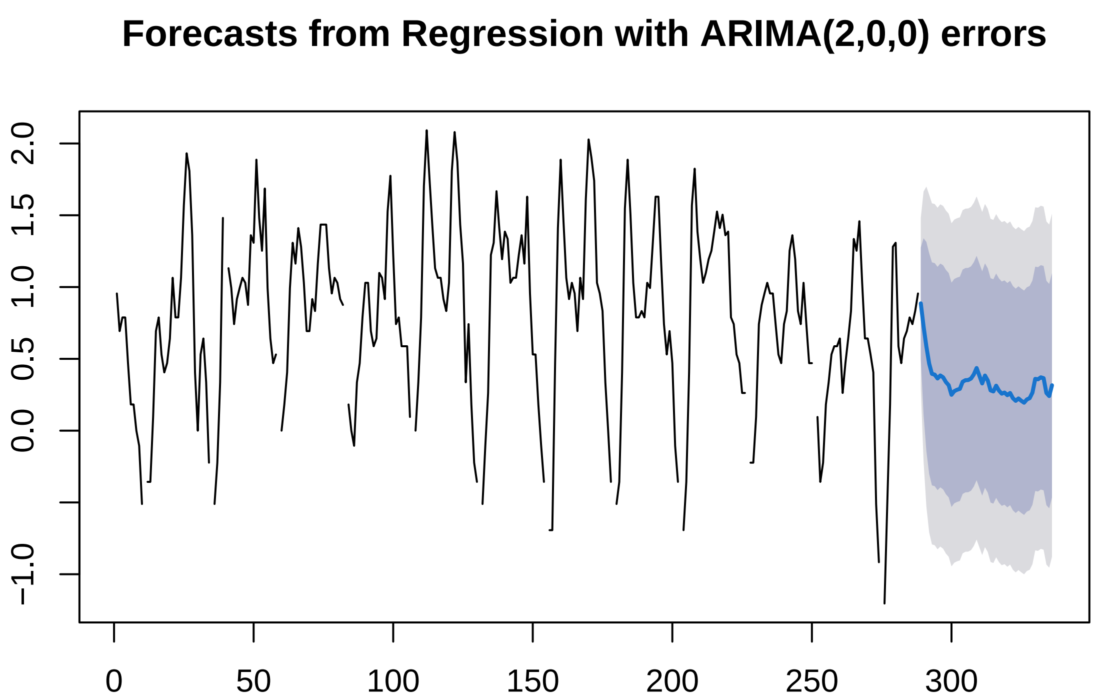



```
## $fit
##
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = xmean, include.mean = FALSE, transform.pars = trans, fixed = fixed,
##     optim.control = list(trace = trc, REPORT = 1, reltol = tol))
##
## Coefficients:
##          ar1      ar2   xmean
##       1.1227  -0.4546  0.0020
## s.e.  0.0535   0.0534  0.0567
##
## sigma^2 estimated as 0.09775:  log likelihood = -71.16,  aic = 150.32
##
## $degrees_of_freedom
## [1] 272
##
## $ttable
##       Estimate     SE t.value p.value
## ar1     1.1227 0.0535 20.9930  0.0000
## ar2    -0.4546 0.0534 -8.5059  0.0000
## xmean   0.0020 0.0567  0.0358  0.9715
##
## $AIC
## [1] 0.5466136
##
```

```
## $AICc
## [1] 0.5469356
##
## $BIC
## [1] 0.5992212
```

```r
# Go ahead and fit the linear regression with AR(2) errors.
# The Arima/forecast functions are most convenient for subsequent forecasting:

fitAR2reg <- Arima(ytrain, order = c(2,0,0), xreg=cbind(time=timetrain, temp=temptrain, rhum=rhumtrain,
```

```r
# Create data frames / data matrices to forecast the last h values of the series
h <- 48 # The number of steps ahead we want to forecast
T <- nrow(Air)

newdf <- data.frame(time=timetest, temp = temptest, rhum = rhumtest, hum = humtest)

newxreg <- model.matrix(~ -1 + time + temp + rhum + hum, data=newdf)


# Forecast plots
fc <- forecast(fitAR2reg ,  h=h, xreg = newxreg)
plot(fc)
```




```r
# Calculate RMSE of forecasts:
sqrt(mean((fc$mean - ytest)^2, na.rm=TRUE))
```

```
## [1] 0.6719156
```