ACR2Full

Martin Cavarga

08 February, 2020

Advanced Methods of Time Series Analysis Applied to Quarterly Estimates of Unemployment Rate

Introduction

The chosen source of data is the Labour Force Survey (LFS) quarterly estimates of unemployment rate in the UK since March 1971, up to March 2018.

---

1. Elementary Modeling by an AR Process

We begin by extracting the data from a downloaded file

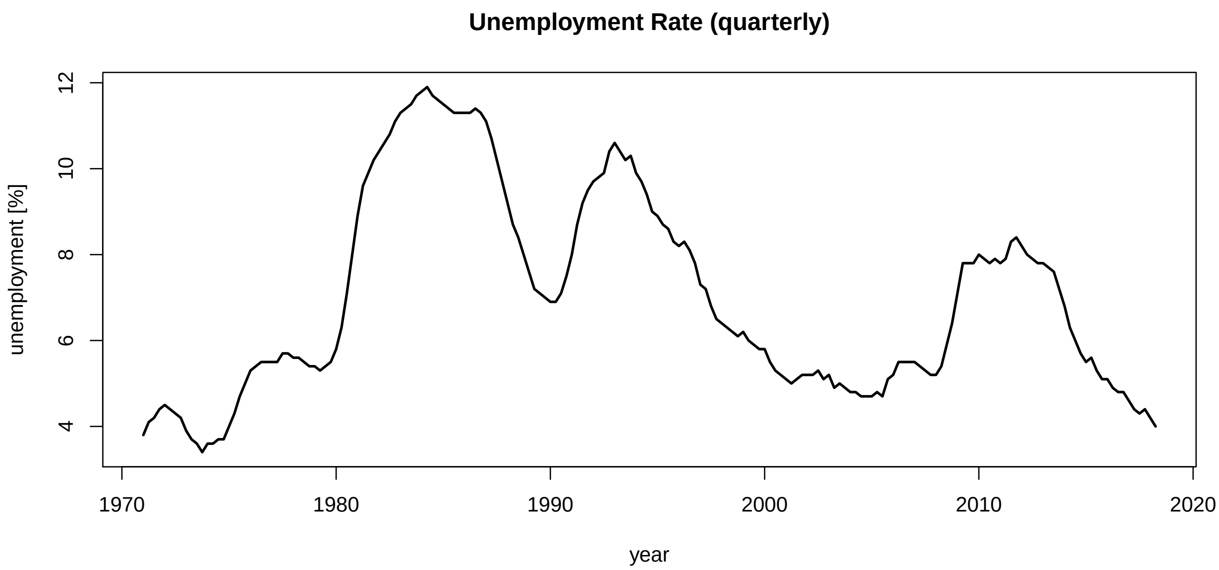
```
##   head.dat.time.
## 1        1971 Q1
## 2        1971 Q2
## 3        1971 Q3
## 4        1971 Q4
## 5        1972 Q1
## 6        1972 Q2
```

```
##   head.months.
## 1            0
## 2            3
## 3            6
## 4            9
## 5            0
## 6            3
```

```
##   head.years.
## 1        1971
## 2        1971
## 3        1971
## 4        1971
## 5        1972
## 6        1972
```

```
##   head.years.
## 1     1971.00
## 2     1971.25
## 3     1971.50
## 4     1971.75
## 5     1972.00
## 6     1972.25
```

1.1: Data Plot

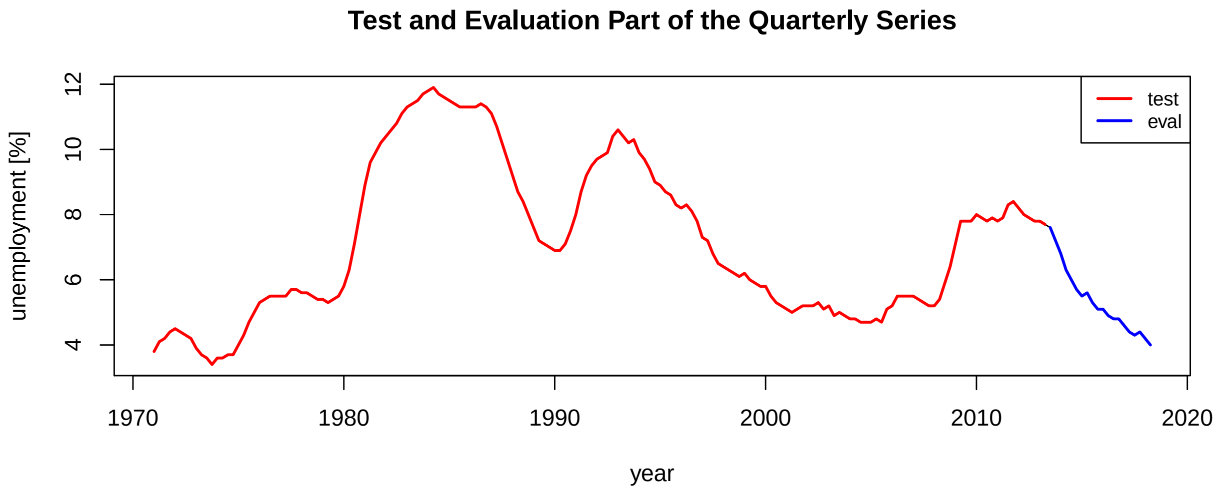


Now even thought we are working with annual data there should not be any seasonal components or trend since the results do not depend on periodic observable phenomena, but rather the complex economic situation over multiple decades. Also the data may include exponentially decaying decrease in unemployment, but only after year 1980, which would suggest a regime-switching stochastic process.

1.2: Test Part and Evaluation Part of the Time Series

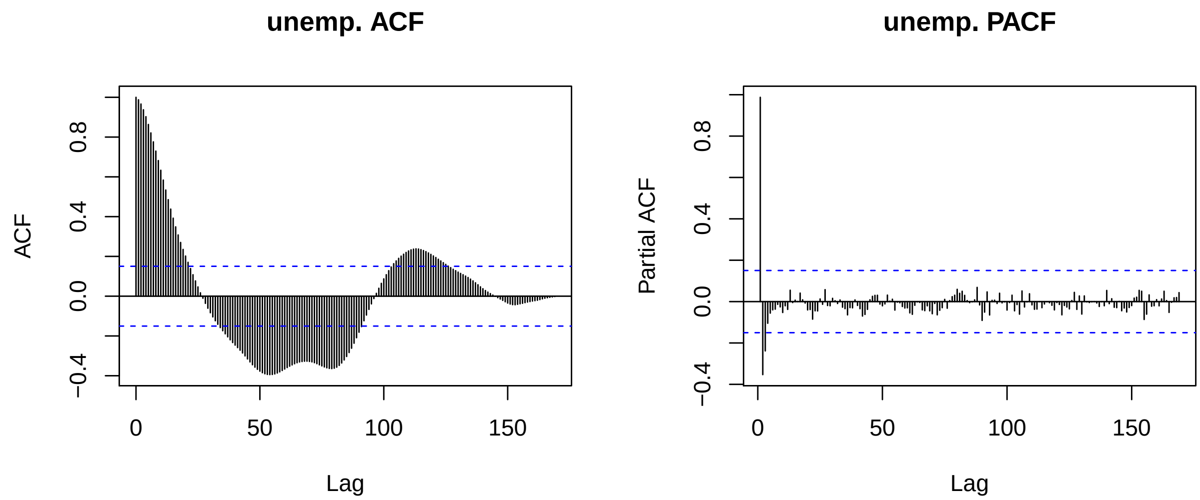
Now we separate the time series into test part, where a suitable model of a stochastic process will be found, and the evaluation part, where predictions given by such model are evaluated. Since our dataset contains quarterly data, we choose the length of the evaluation part of the time series as $L = k * 4$ where k is an arbitrary (and sufficiently small) positive integer.

```
## [1] 20
```



1.3: Mean, Variance, ACF, and PACF of the Test Part

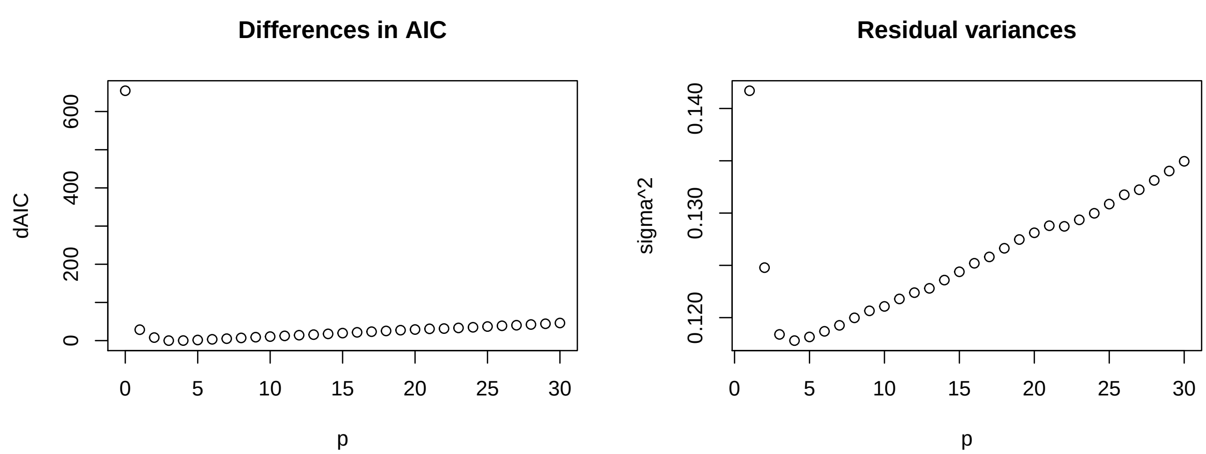
```
##      Min.      Max.      Mean    Median  Variance
##  3.400000 11.900000  7.188824  6.900000  5.661827
```



As we mentioned in section 1.1, the underlying process which gave rise to the observed results is aperiodic, yet it is undoubtedly a process with memory. Unemployment rate strongly depends (aside from other important aspects) on its own history which might extend generations into the past. The results are, however, significantly influenced by external phenomena, such as the global economic crisis in late 2000's.

1.4: Finding a Suitable AR Model

Since the economic situation and the job market remembers its past, we choose a simple $AR(p)$ process with parameter p corresponding to the number of steps after which the process still "remembers" its past. The inbuilt `ar()` function automatically finds the model with the lowest AIC (Akaike's Information Criterion). And by plotting the `$aic` parameter we obtain differences $AIC_{min} - AIC_k$ for all models.



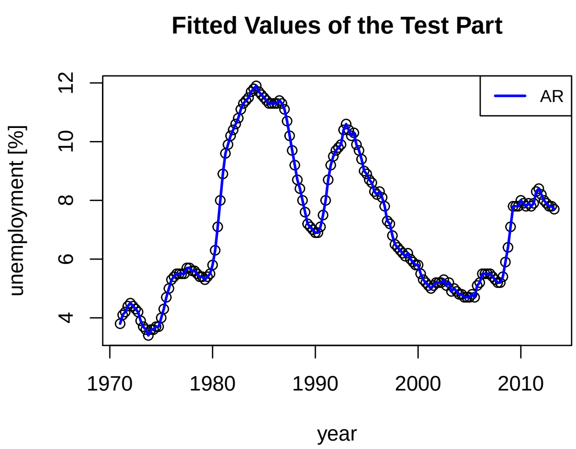
As we can see in the figures, the lowest variance of residues corresponds to an $AR(3)$ process with coefficients:

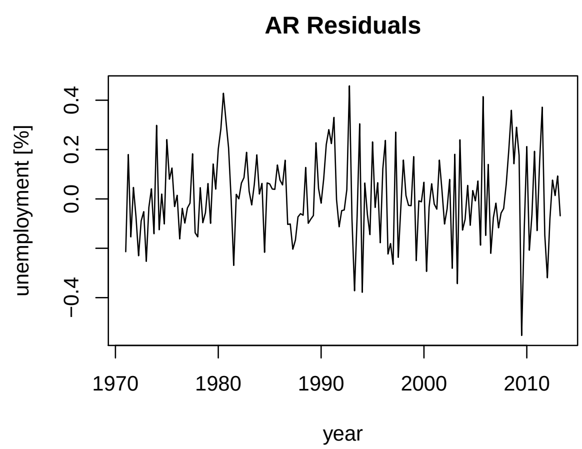
```
##            [,1]        [,2]       [,3]
## coef 1.2519124 -0.03440575 -0.2384831
## se   0.0753756  0.12294642  0.0753756
```

```
## [1] 3
```

Unfortunately, the **ar** function does not return fitted values, thus we need to model the time series via the **arima** function using the AR order **p** from the previous result.

```
##
## Call:
## arima(x = as.numeric(unempseries$test), order = c(p, 0, 0))
##
## Coefficients:
##          ar1      ar2      ar3  intercept
##       1.6000  -0.4176  -0.1951     6.8458
## s.e.  0.0752   0.1412   0.0754     0.9580
##
## sigma^2 estimated as 0.0289:  log likelihood = 56.88,  aic = -103.75
```

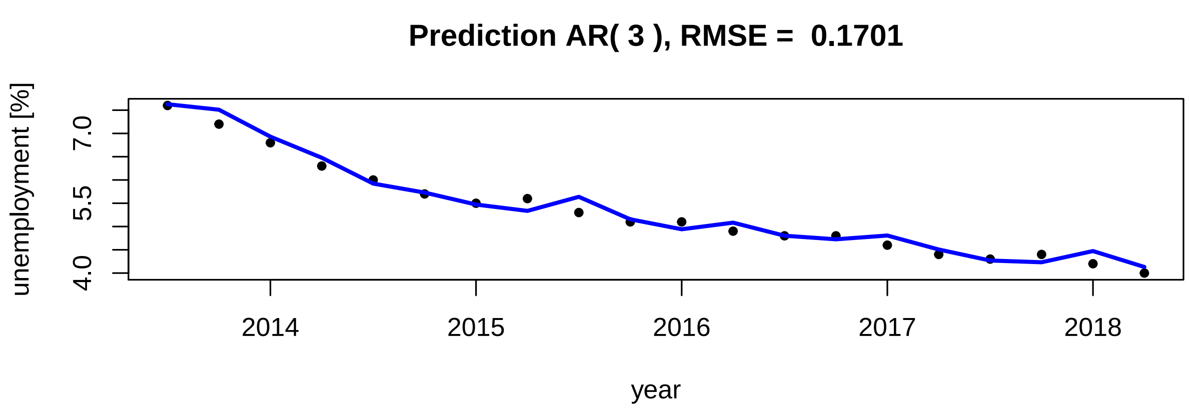




The given AR model seems to fit the time series very well, which may be due to its low oscillation rate.

1.5: 1-Step Predictions Over the Evaluation Part

```
## [1] 0.1701227
```



1.6.: Conclusion

Since it has very low rate of local oscillation, but does not have an easily predictable systematic pattern, the analyzed unemployment rate time series seems to be well-estimated by an $AR(3)$ process with low prediction errors. However, as we mentioned earlier we might be dealing with a 'regime-switching' process. Further analysis will be carried out in the next chapter.

---

2. Finding the Parameters of a SETAR Model

As we mentioned, the unemployment time series might be a result of a regime-switching process. Naturally, the behavior of the unemployment rate in a given country should depend on the current economic situation. The change in the local economy can be described via a set of "thresholds" which determine whether the stochastic process changes its regime. The regime of a stochastic process is defined as a unique ARMA or any other linear stochastic process with unique parameters. We begin by finding the parameters of a Self-Exciting Threshold Autoregressive (SETAR) process, that is: a process whose regime is described by a random variable determined by the very process itself, more specifically its history of up to d steps behind, which in an essence means that the process "influences its regime" up to d time steps into the future.

For the purposes of this analysis we consider only 2 regimes, namely the regime of "job crisis" when the unemployment rate may fluctuate or drop more wildly compared to the regime of "job stability" when the unemployment rate stabilizes or grows.

2.1: Useful Functions

First, we define a, so called, "indicator function" which essentially returns a boolean value from a given input process value `x` and threshold value `c`:

```
Indicator <- function(x, c) ifelse(x > c, 1, 0)
```

Afterwards, we define the basis function for a single regime

```
Yt <- function(x, t, p) c(1, x[(t - 1):(t - p)])
```

which can then be used in the "basis" for two regimes:

```
Xt <- function(x, t, p, d, c, z = x) {
  # z is the threshold variable
  I <- Indicator(z[t - d], c)
  Y <- Yt(x, t, p)
  c((1 - I) * Y, I * Y)
}
```

We can test the function on a given subset of the unemployment time series. Due to the fact that the examined time series is rather 'smooth', for further use, we will examine its differences:

```
xt <- na.omit(diff(as.numeric(dat$unemp))) # take differences in data
x_train <- xt[seq_along(unempseries$test)] # extract test part
nt <- length(x_train)
x_eval <- xt[(nt + 1):length(xt)] # extract eval part
xt <- x_train # set the source data to test part values
```

```
d <- 1; t <- 3;
xt[(t - d): t]
```

```
## [1] 0.1 0.2
```

```
Xt(xt, t, p, d, c = 0)
```

```
## [1] 0.0 0.0 0.0 1.0 0.1 0.3
```

```r
t <- 100;
xt[(t - d): t]
```

```
## [1] -0.3 -0.1
```

```r
Xt(xt, t, p, d, c = 0)
```

```
## [1]  1.0 -0.3 -0.1 -0.2  0.0  0.0  0.0  0.0
```

As we can see, in the first case, with time series values crossing zero from above, the latter half of the coefficient vector gets expressed, corresponding to the series assuming the second regime.

Then we need a function defining a deterministic skeleton of the model:

```r
SkeletonSETAR <- function(x, t, p, d, c, theta, z = x) theta %*% Xt(x, t, p, d, c, z)
```

where `theta` corresponds to the parameter vector, for example:

```
##      [,1]
## [1,]  0.4
```

and the last group of functions we need for the upcoming procedure are functions for the information criteria of a SETAR model:

```r
# Akaike
AIC_SETAR <- function(orders, regimeDataCount, resVariances) {
  sum(regimeDataCount * log(resVariances) + 2 * (orders + 1))
}

# Bayesian
BIC_SETAR <- function(orders, regimeDataCount, resVariances) {
  sum(regimeDataCount * log(resVariances) + log(regimeDataCount) * (orders + 1))
}

#' and test it out:

AIC_SETAR(c(2, 2), c(10, 10), c(0.5, 0.7))
```

```
## [1] 1.501779
```

```r
BIC_SETAR(c(2, 2), c(10, 10), c(0.5, 0.7))
```

```
## [1] 3.317289
```

2.2: The Estimation of Parameters of a SETAR Model

Given a dataset `x` and parameters `p` (AR order), `d` (SETAR delay), and the threshold `c` we find the coefficients of a SETAR model with these parameters by performing a multivariate linear regression. The coefficient vector `PhiParams` is the vector of unknowns of a linear system with matrix $\mathbf{X}$ and a right-hand-side vector $\mathbf{y}$ given by the time series. Although for higher values of `p` the inversion of matrix $\mathbf{X}^\top\mathbf{X}$ (with dimensions $(2p+2)\times(2p+2)$) might be computationally demanding, we will determine the covariance matrix, i.e.: $(\mathbf{X}^\top\mathbf{X})^{-1}$ using a function `inv` from the `matlib` package:

```r
suppressMessages(pkgTest("zeallot"))
suppressMessages(pkgTest("matlib"))

EstimSETAR <- function(x, p, d, c) {

  resultModel <- list()
  resultModel$p = p; resultModel$d = d; resultModel$c = c;
  resultModel$data = x;  n = length(x);  resultModel$n = n;
  k <- max(p, d)
```

```r
  X <- as.matrix(apply(as.matrix((k + 1):n), MARGIN=1, function(t) Xt(x, t, p, d, c) ))
  y <- as.matrix(x[(k + 1):n])

  A = crossprod(t(X), t(X));  b = crossprod(t(X), y)

  if (abs(det(A)) > 0.000001) {
    inv <- inv(A)
    sol_phi <- as.numeric(t(inv %*% b)); sol_se <- sqrt(diag(inv)/n);
    eps <- 0.01;

    # filter out those coeffs that are of the same order of magnitude as their errors
    filter <- sapply(1:(2*(p + 1)), function (i) ifelse(
      abs(sol_phi[i]) <= 2 * abs(sol_se[i]), 0, 1)
    )

    sol_phi <- sol_phi * filter
    sol_se <- sol_se * filter

    solution <- cbind(phi = sol_phi,  se = sol_se)

    resultModel$PhiParams <- solution[,1] # solving (X'X)*phi = X'y
    resultModel$PhiStErrors <- solution[,2]  # standard errors
    skel <- crossprod(X, resultModel$PhiParams); resultModel$skel <- skel;
    resultModel$residuals <- (y - skel)
    resultModel$resSigmaSq <- 1 / (n - k) * sum(resultModel$residuals ^ 2)
    resultModel$name <- paste0("SETAR(",p,",",d,",",round(c,3),")")
    resultModel$nReg <- 2 # 2 regimes

    return(resultModel)
  } else {
    return(NA)
  }
}
```

After performing this procedure for multiple parameters, i.e.: searching the discrete parameter space, we further process the model with minimum residual square sum. For that we'll use:

```r
EstimSETAR_postproc <- function(model) {
  x <- model$data; k <- max(model$p, model$d); c <- model$c; n <- model$n;
  y <- as.matrix(x[(k + 1):n])
  skel <- model$skel; model$skel <- NULL; #skel attribute no longer needed

  model$n1 <- sum(apply(as.matrix(x), MARGIN = 1, function(xt) (1 - Indicator(xt, c))))
  model$n2 <- sum(apply(as.matrix(x), MARGIN = 1, function(xt) Indicator(xt, c)))

  model$resSigmaSq1 <- sum(
    apply(as.matrix(seq_along(y)), MARGIN = 1,
          function(t) ifelse((1 - Indicator(y[t], c)), (y[t] - skel[t])^2, 0))) / (model$n1 - k)
  model$resSigmaSq2 <- sum(
    apply(as.matrix(seq_along(y)), MARGIN = 1,
          function(t) ifelse(Indicator(y[t], c), (y[t] - skel[t])^2, 0))) / (model$n2 - k)

  model$AIC <- AIC_SETAR(c(p, p), c(model$n1, model$n2), c(model$resSigmaSq1, model$resSigmaSq2))
  model$BIC <- BIC_SETAR(c(p, p), c(model$n1, model$n2), c(model$resSigmaSq1, model$resSigmaSq2))

  return(model)
}
```

and now we test the function for suitable parameters:

```r
str( model <- EstimSETAR_postproc(model) )
```

```
## List of 17
##  $ p          : num 2
##  $ d          : num 2
##  $ c          : num 0
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:6] 0 0.4124 0.4661 0.0692 0.8179 ...
##  $ PhiStErrors: num [1:6] 0 0.0534 0.0694 0.0155 0.0448 ...
##  $ residuals  : num [1:168, 1] 0.1002 -0.1157 -0.2168 -0.0703 -0.0121 ...
##  $ resSigmaSq : num 0.028
##  $ name       : chr "SETAR(2,2,0)"
##  $ nReg       : num 2
##  $ n1         : num 102
##  $ n2         : num 68
##  $ resSigmaSq1: num 0.0213
##  $ resSigmaSq2: num 0.039
##  $ AIC        : num -597
##  $ BIC        : num -578
```

It should be noted that for some values of `p` and `d` the indices of arrays in the algorithms might get out of the range of regularity for the linear system. For that reason we implement exceptions for the outputs of `EstimSETAR` in the following algorithm.

2.3: SETAR Parameter Estimation Procedure

To answer the question: 'how does one find the right parameters `p`, `d` and `c` for their desired SETAR model?', we implement the following procedure:

```r
pmax <- 7 # set maximum order p
# limit the c parameter by the 7.5-th and 92.5 percentile
cmin <- as.numeric(quantile(xt, 0.075)); cmax <- as.numeric(quantile(xt, 0.925));
h = (cmax - cmin) / 100 # determine the step by which c should be iterated
models <- list()
modelColumns <- list()
for (p in 1:pmax) {
  for (d in 1:p) {
    pdModels <- list()
    for (c in seq(cmin, cmax, h)) {
      tmp <- EstimSETAR(xt, p, d, c) # try to run the function
      # then test whether it returns`NA` as a result
      if (!as.logical(sum(is.na(tmp))) ) {
        pdModels[[length(pdModels) + 1]] <- tmp
      }
    }
    sigmas <- as.numeric(lapply(pdModels, function(m) m$resSigmaSq))
    orders <- order(sigmas)
    # only the model whose parameter c gives the lowest residual square sum is chosen for postprocessing
    min_sigma_model <- EstimSETAR_postproc(pdModels[[ orders[1] ]])
    models[[length(models) + 1]] <- min_sigma_model
    modelColumns[[length(modelColumns) + 1]] <- c(
      p, d, min_sigma_model$c,
      min_sigma_model$n1, min_sigma_model$n2,
      min_sigma_model$AIC, min_sigma_model$BIC,
      min_sigma_model$resSigmaSq)
  }
}
```

```
##    p d         c  n1  n2       AIC       BIC resSigmaSq
## 1  1 1  0.000325 102  68 -595.3267 -585.6377 0.02955918
## 2  2 1  0.205425 142  28 -596.8388 -583.9747 0.02908718
## 3  2 2  0.102875 129  41 -616.5615 -602.8413 0.02624694
## 4  3 1  0.300650 153  17 -601.5380 -586.0834 0.02777303
## 5  3 2  0.102875 129  41 -606.5288 -588.2352 0.02668915
## 6  3 3 -0.197450  34 136 -592.0333 -574.2772 0.02778604
## 7  4 1  0.300650 153  17 -595.8224 -576.5041 0.02800968
## 8  4 2  0.102875 129  41 -600.5090 -577.6421 0.02681811
## 9  4 3  0.000325 102  68 -587.9837 -563.7613 0.02750077
## 10 4 4  0.205425 142  28 -597.1284 -575.6882 0.02698219
## 11 5 1  0.205425 142  28 -589.0005 -563.2723 0.02758865
## 12 5 2  0.102875 129  41 -598.8347 -571.3944 0.02590032
```

Now we have a set of models in their original order. To find the best suitable model, we choose 12 models with the lowest BIC (Bayesian Information Criterion):

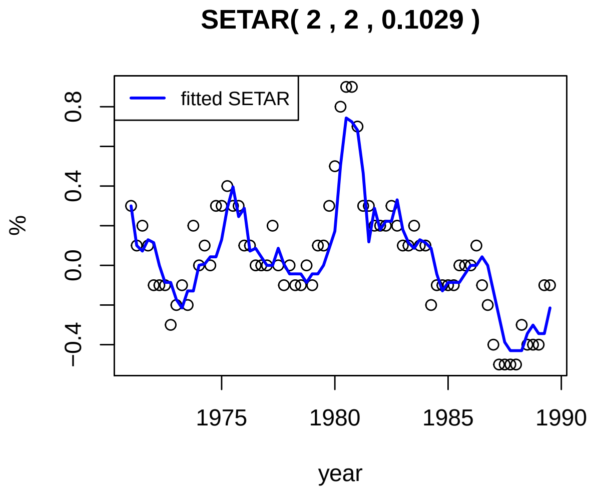
```
##    p d         c  n1  n2       AIC       BIC resSigmaSq
## 3  2 2  0.102875 129  41 -616.5615 -602.8413 0.02624694
## 5  3 2  0.102875 129  41 -606.5288 -588.2352 0.02668915
## 4  3 1  0.300650 153  17 -601.5380 -586.0834 0.02777303
## 1  1 1  0.000325 102  68 -595.3267 -585.6377 0.02955918
## 2  2 1  0.205425 142  28 -596.8388 -583.9747 0.02908718
## 8  4 2  0.102875 129  41 -600.5090 -577.6421 0.02681811
## 7  4 1  0.300650 153  17 -595.8224 -576.5041 0.02800968
## 10 4 4  0.205425 142  28 -597.1284 -575.6882 0.02698219
## 6  3 3 -0.197450  34 136 -592.0333 -574.2772 0.02778604
## 15 5 5  0.102875 129  41 -600.5783 -573.1380 0.02542239
## 14 5 4  0.205425 142  28 -597.2758 -571.5476 0.02618782
## 12 5 2  0.102875 129  41 -598.8347 -571.3944 0.02590032
```

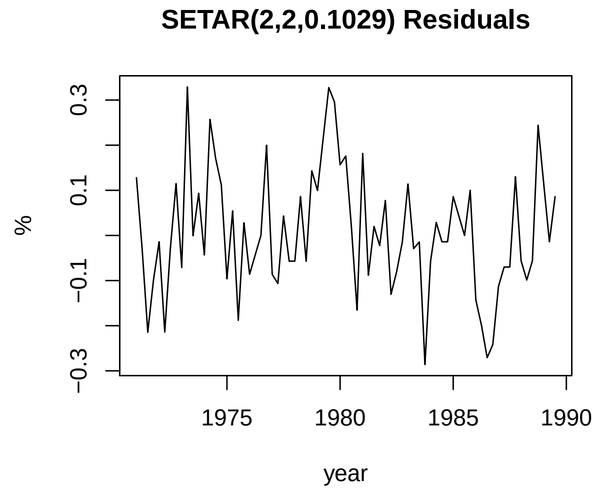
and we can also include errors of the estimated regression coefficients:

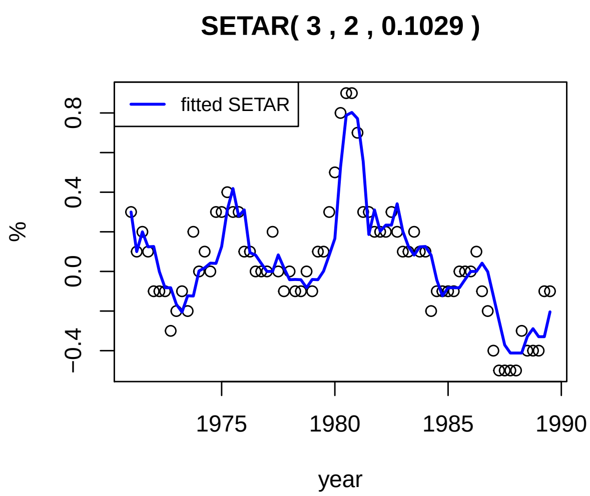
```
## $`220.1029/`
##          [,1]       [,2]       [,3]       [,4]       [,5]        [,6]
## Phi         0 0.43170625 0.42834406 0.09138806 1.08026643 -0.42454326
## stdError    0 0.04164731 0.05038936 0.02581381 0.05901015  0.07802637
##
## $`320.1029/`
##          [,1]       [,2]       [,3] [,4]       [,5]       [,6]        [,7] [,8]
## Phi         0 0.41811454 0.40547459    0 0.08019403 1.07869738 -0.31056937    0
## stdError    0 0.04340129 0.05441896    0 0.02619414 0.05963855  0.09969198    0
##
## $`310.3007/`
##          [,1]       [,2]       [,3] [,4]       [,5]      [,6]      [,7]
## Phi         0 0.53829523 0.20456086    0 -0.1620930 1.1391822 0.6438414
## stdError    0 0.04368799 0.04233384    0  0.0792829 0.1808141 0.1619481
##                [,8]
## Phi      -0.9934059
## stdError  0.1556738
##
## $`113e-04/`
##                [,1]       [,2]        [,3]       [,4]
## Phi      0.02224840 0.78551645 -0.06795137 0.93659667
## stdError 0.01108097 0.05520027  0.01518854 0.04389936
##
## $`210.2054/`
##          [,1]       [,2]       [,3] [,4]      [,5] [,6]
## Phi         0 0.50086623 0.22030155    0 0.8652436    0
## stdError    0 0.04539197 0.03816144    0 0.1019782    0
##
```

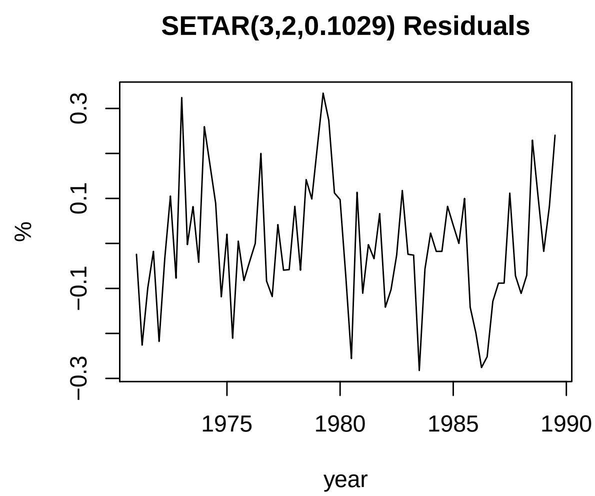
```
## $`420.1029/`
##          [,1]      [,2]      [,3] [,4] [,5]       [,6]       [,7]        [,8]
## Phi         0 0.4229181 0.4253314    0    0 0.08769237 1.06176402 -0.31828032
## stdError    0 0.0434935 0.0555784    0    0 0.02700914 0.06146456  0.09992171
##          [,9] [,10]
## Phi         0     0
## stdError    0     0
##
## $`410.3007/`
##          [,1]       [,2]       [,3] [,4] [,5] [,6]      [,7]      [,8] [,9]
## Phi         0 0.54084131 0.20899552    0    0    0 0.9452945 0.7317165    0
## stdError    0 0.04390061 0.04344423    0    0    0 0.1823845 0.1623094    0
##               [,10]
## Phi      -1.2916701
## stdError  0.1591013
##
## $`440.2054/`
##          [,1]       [,2]       [,3] [,4]        [,5]       [,6]      [,7]
## Phi         0 0.65727359 0.38978649    0 -0.16773802 0.14173560 0.6582130
## stdError    0 0.03774249 0.04650459    0  0.04752073 0.04020795 0.0967757
##                [,8]      [,9]      [,10]
## Phi      -0.3314201 0.4747251 -0.4104174
## stdError  0.1119202 0.1083704  0.1026022
##
## $`33-0.1974/`
##          [,1] [,2]       [,3]      [,4] [,5]       [,6]       [,7]       [,8]
## Phi         0    0 0.46444897 0.3403872    0 0.75043610 0.12079526 -0.1255942
## stdError    0    0 0.08494846 0.1407401    0 0.03879669 0.04629148  0.0410894
##
## $`550.1029/`
##                 [,1]       [,2]       [,3] [,4] [,5]       [,6] [,7]       [,8]
## Phi      0.016299311 0.64365718 0.38921338    0    0 -0.1625067    0 0.47118620
## stdError 0.008034404 0.04103325 0.05004459    0    0  0.0539052    0 0.07585229
##                 [,9]      [,10]      [,11] [,12]
## Phi      -0.19079304 0.53307667 -0.3303657     0
## stdError  0.08097692 0.09450724  0.1058457     0
##
## $`540.2054/`
##          [,1]       [,2]       [,3] [,4] [,5]        [,6]       [,7]       [,8]
## Phi         0 0.63242444 0.40442194    0    0 -0.20691700 0.16324920 0.63510151
## stdError    0 0.03804026 0.04658868    0    0  0.03955612 0.04111555 0.09720526
##                [,9]     [,10]      [,11] [,12]
## Phi      -0.2803961 0.4214599 -0.4884326     0
## stdError  0.1138933 0.1103896  0.1218861     0
##
## $`520.1029/`
##                 [,1]       [,2]       [,3]       [,4] [,5]        [,6]
## Phi      0.018185178 0.42527088 0.43763948 0.11801416    0 -0.14016981
## stdError 0.008024606 0.04349921 0.05570058 0.04784635    0  0.04199251
##                [,7]       [,8]       [,9] [,10] [,11] [,12]
## Phi      0.11594786 1.03987845 -0.3667765     0     0     0
## stdError 0.02955887 0.06321611  0.1031897     0     0     0
```

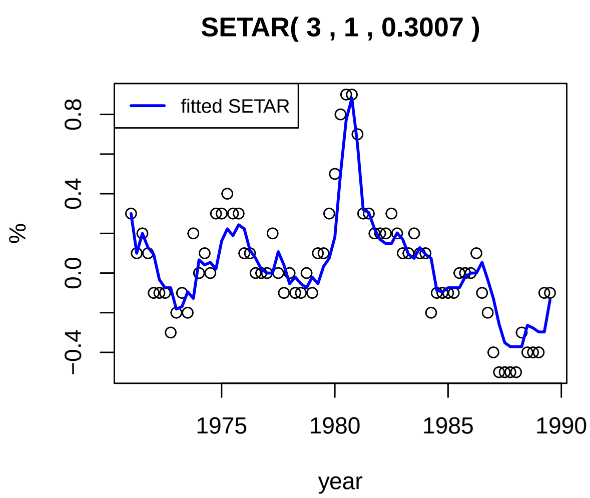
We can now visualize the results of the top 3 models:

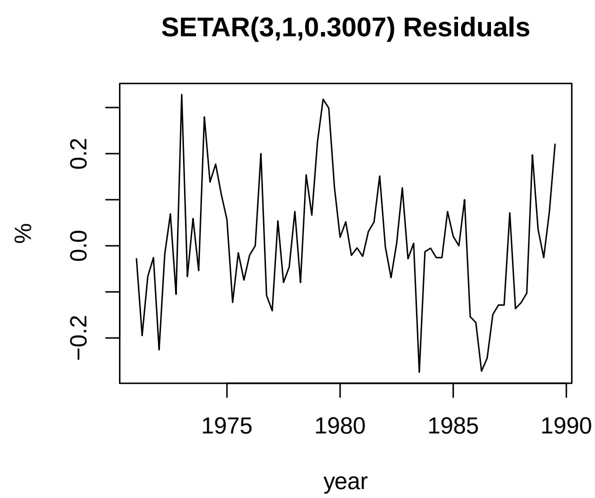










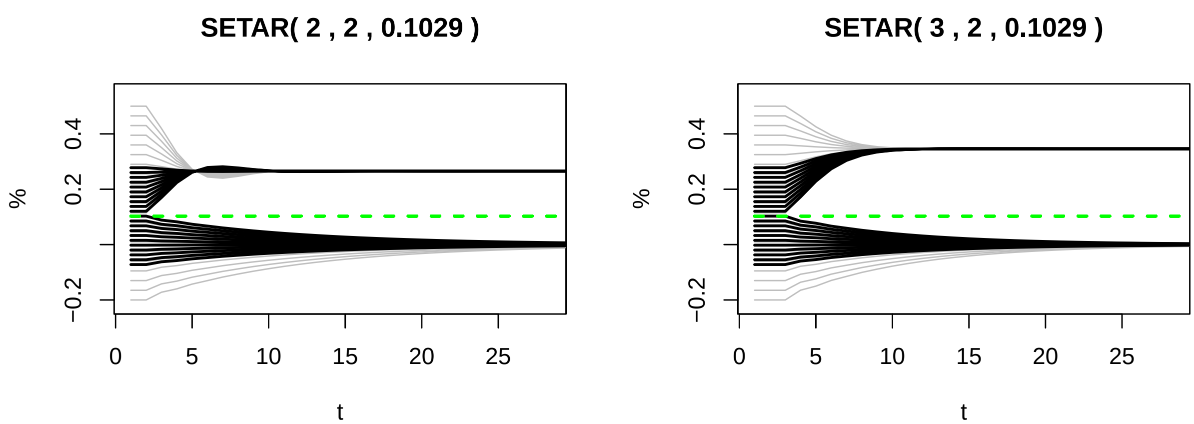


The results suggest that the best SETAR models have a threshold `c` quite close to zero and more-or-less the same

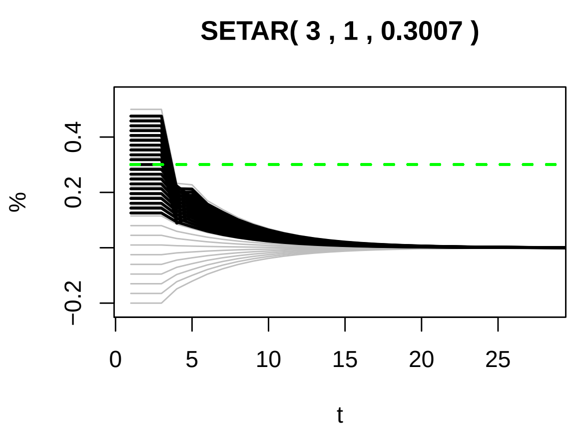
RSS. The very first with a lower `BIC` (Bayesian Information Criterion), has slightly larger RSS than the models that come after it. To verify the correctness of our procedure we will need to compare it with inbuilt functions from a verified library.

2.4: SETAR Equilibria and Equilibrium Simulations

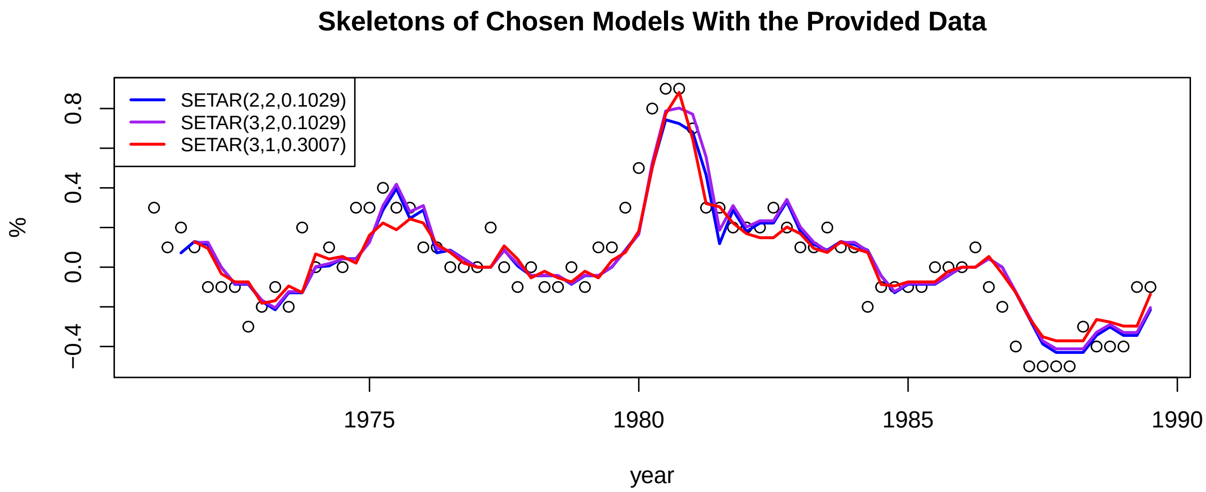
It is also essential to find out whether the skeletons of the selected SETAR models have some equilibria. The estimation of the exact equilibria of the piecewise-linear skeletons with $p = 1$ is straightforward: We find the fixed points of the skeletons by finding the intersections between their graphs and the identity line id$x = x$, given the model parameters (coefficients). However, the results of our search have mostly higher AR degrees, thus we will need to determine the models' equilibria using a more general method, namely letting the model skeletons evolve with multiple input initial conditions.



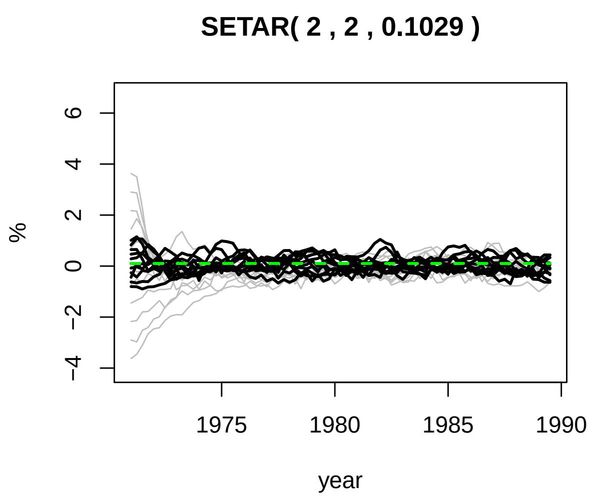

```
## $`SETAR( 3 , 1 , 0.301 ) equilibria`
## [1] 0.0000 0.2654
##
## $`SETAR( 3 , 1 , 0.301 ) equilibria`
## [1] 0.0000 0.3459
##
## $`SETAR( 3 , 1 , 0.301 ) equilibria`
## [1] 0
```

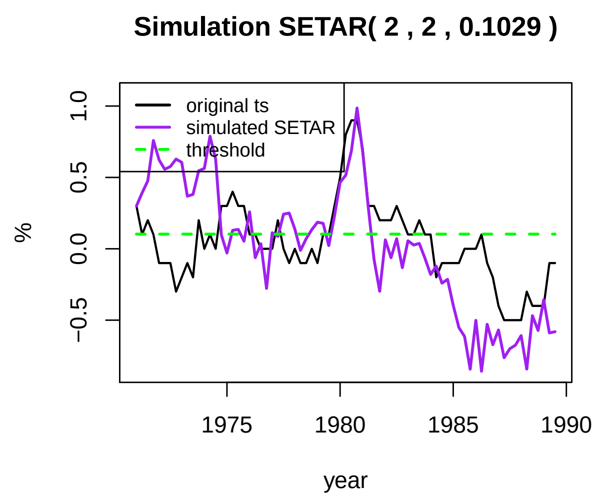


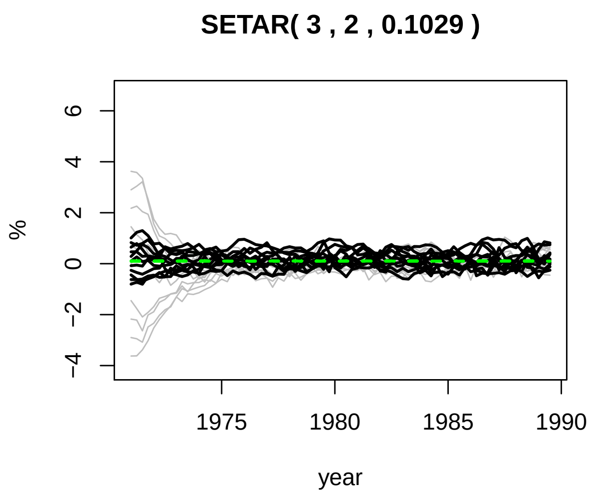
As we see, the trajectories of the top 3 models gravitate towards 0 in all models, but in the first and second model they can end up in one more position, close to zero. It might also be interesting to see how the trajectories evolve when we add an iid noise on top of the model skeleton. First we observe the skeleton behavior in our data:

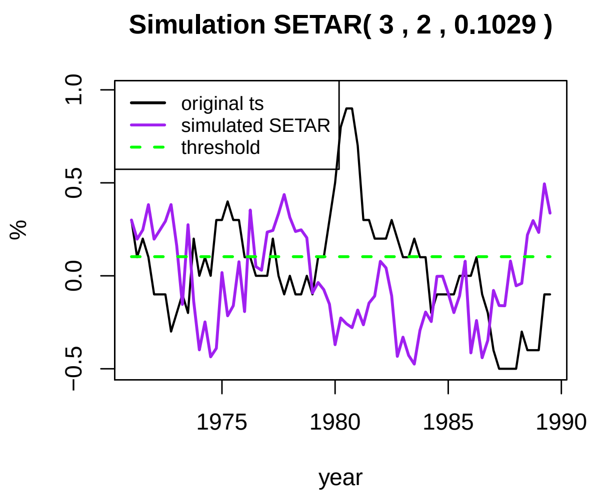


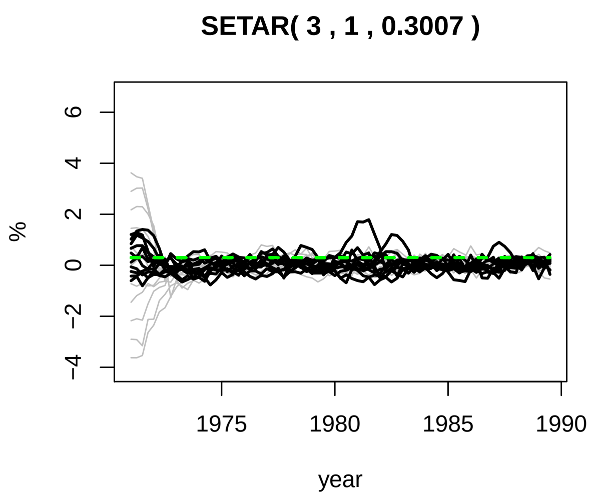
And then we carry out multiple simulations with initial conditions close to the threshold. The added noise will have the same deviance as the residual square sum.

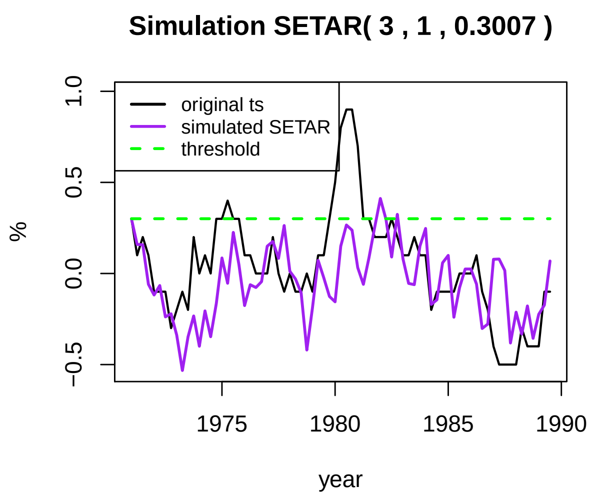












The trajectories of all of the first three models seem to gravitate toward 0 significantly fast (or alternatively: towards their threshold values which are close to zero as well). The relatively low oscillation rate of the original time series suggests that the differences of this time series will, at most, fluctuate around 0. The change between the 'high' and 'low' regimes does not seem very significant, at leat on the larger scale. The validity of the model will be tested in chapter 3.

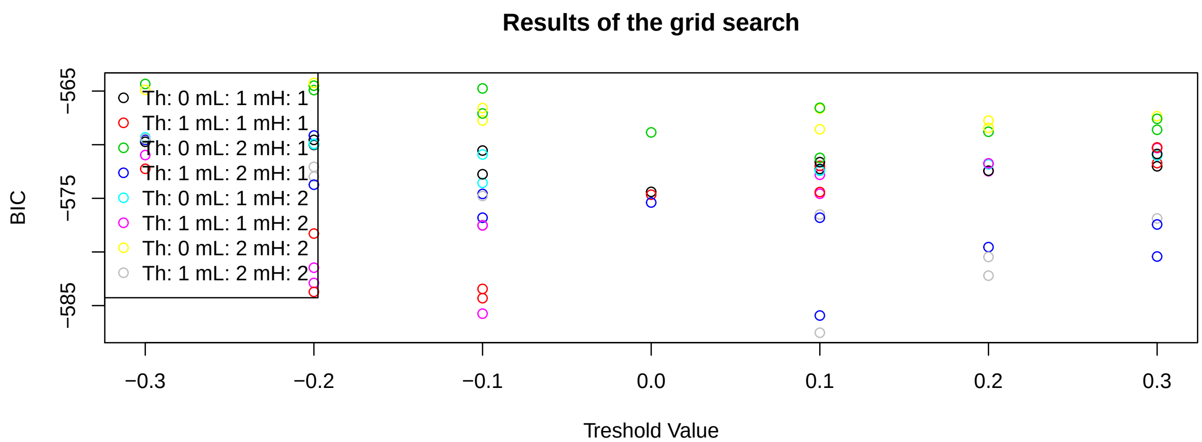
2.5: Comparison Of the Results With Inbuilt Functions

To verify the correctness of our methods we proceed to construct the top 3 SETAR models by plugging their parameters into inbuilt functions:

```
suppressMessages(pkgTest("tsDyn"))

#Testing a function which selects an orders automatically:
mmax <- 2

par(mfrow=c(1,1))
( result1 <- selectSETAR(xt, m=mmax, thDelay=0:(mmax-1), criterion="BIC", same.lags=T, trim=0.1)  )
```

```
## Using maximum autoregressive order for low regime: mL = 2
```

```
## Using maximum autoregressive order for high regime: mH = 2
## Searching on 21 possible threshold values within regimes with sufficient ( 10% ) number of observations
## Searching on  84  combinations of thresholds ( 21 ), thDelay ( 2 ) and m ( 2 )
```




```
## Results of the grid search for 1 threshold
##    thDelay m   th       BIC
## 1        1 2  0.1 -587.5300
## 2        1 2  0.1 -585.9265
## 3        1 1 -0.1 -585.7518
## 4        1 1 -0.1 -584.3112
## 5        1 1 -0.2 -583.7230
## 6        1 1 -0.1 -583.4471
## 7        1 1 -0.2 -582.8902
## 8        1 2  0.2 -582.2097
## 9        1 1 -0.2 -581.4681
## 10       1 2  0.2 -580.4653
```

the estimated thDelay corresponds to `d-1`.

```
## List of 17
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num -0.1
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:6] 0.0711 0.7998 0.1755 0 0.6911 ...
##  $ PhiStErrors: num [1:6] 0.023 0.0946 0.0666 0 0.0444 ...
##  $ residuals  : num [1:168, 1] 0.0838 -0.0539 -0.2005 -0.0466 -0.0736 ...
##  $ resSigmaSq : num 0.0293
##  $ name       : chr "SETAR(2,1,-0.1)"
##  $ nReg       : num 2
##  $ n1         : num 50
##  $ n2         : num 120
##  $ resSigmaSq1: num 0.0357
##  $ resSigmaSq2: num 0.0272
##  $ AIC        : num -583
##  $ BIC        : num -564
```

Note that we set `thDelay=0:(mmax-1)` instead of `1:mmax`. `selectSETAR` uses `thDelay = 0` for step `d=1` delay correspondence: $x_{t-d} < c$ or $x_{t-d} > c$. the resulting BIC's are different, possibly due to the package using a different formula

```
## Results of the grid search for 1 threshold
```

```
##    thDelay m   th       BIC
## 1        1 2  0.1 -590.7913
## 2        1 2  0.1 -589.0388
## 3        1 1 -0.1 -588.2120
## 4        1 1 -0.1 -586.5338
## 5        1 1 -0.2 -585.9216
## 6        1 1 -0.1 -585.7998
## 7        1 2  0.2 -585.1638
## 8        1 1 -0.2 -585.0533
## 9        4 2  0.1 -585.0007
## 10       4 2  0.1 -584.5197
```

Setting higher `mmax`, the function returns a list of models similar to the one given by our procedure in section 2.3. We can also compare the accuracy of the computation of the regression coefficients in our `EstimSETAR` method, with for example: `setar()` function (from `tsDyn` library as well):

```
setars <- list()
coeffComparison <- list()
resSigmaComparison <- list()
n <- length(xt)
for (i in 1:3) {
  p <- models[[ orders[i] ]]$p
  d <- models[[ orders[i] ]]$d
  c <- models[[ orders[i] ]]$c
  setars[[i]] <- setar(xt, m=p)
  k <- max(p, d)
  resSigmaComparison[[i]] <- c(
    (1 / (n - k) * sum(setars[[i]]$residuals ^ 2)),
    models[[ orders[i] ]]$resSigmaSq
    )
  inbuiltParams <- t(setars[[i]]$coefficients)
  key <- paste(p, d, round(c, digits=4), sep=" / ")
  coeffComparison[[key]] <- rbind(
    inbuiltParams,
    t(append(models[[orders[i]]]$PhiParams, models[[orders[i]]]$c))
  )
  row.names(coeffComparison[[key]]) <- t(c("inbuilt", "custom"))
}
```

```
##
##  1 T: Trim not respected:  0.8511905 0.1488095 from th: 0.3
##  1 T: Trim not respected:  0.8502994 0.1497006 from th: 0.3
##  1 T: Trim not respected:  0.8502994 0.1497006 from th: 0.3
##  1 T: Trim not respected:  0.8502994 0.1497006 from th: 0.3
##  1 T: Trim not respected:  0.8502994 0.1497006 from th: 0.3
```

```
coeffComparison
```

```
## $`2 / 2 / 0.1029`
##             const.L    phiL.1    phiL.2     const.H    phiH.1     phiH.2
## inbuilt -0.01014873 0.4562865 0.2714053 -0.03164651 0.9628571 -0.1110418
## custom   0.00000000 0.4317063 0.4283441  0.09138806 1.0802664 -0.4245433
##               th
## inbuilt 0.100000
## custom  0.102875
##
## $`3 / 2 / 0.1029`
##              const.L    phiL.1    phiL.2     phiL.3    const.H    phiH.1
## inbuilt -0.009656784 0.4908912 0.1958345 0.03467407 0.09657510 0.5947604
## custom   0.000000000 0.4181145 0.4054746 0.00000000 0.08019403 1.0786974
##             phiH.2     phiH.3       th
```

```
## inbuilt  0.5578468 -0.6112305 0.300000
## custom  -0.3105694  0.0000000 0.102875
##
## $`3 / 1 / 0.3007`
##              const.L    phiL.1    phiL.2     phiL.3    const.H    phiH.1
## inbuilt -0.009656784 0.4908912 0.1958345 0.03467407  0.0965751 0.5947604
## custom   0.000000000 0.5382952 0.2045609 0.00000000 -0.1620930 1.1391822
##            phiH.2     phiH.3      th
## inbuilt 0.5578468 -0.6112305 0.30000
## custom  0.6438414 -0.9934059 0.30065
```

```
# comparing RSS

resSigmaComparison <- data.frame(matrix(unlist(resSigmaComparison), nrow=3, byrow=T))
colnames(resSigmaComparison) <- c("inbuilt", "custom")
row.names(resSigmaComparison) <- t(paste("rss",1:3))
resSigmaComparison
```

```
##          inbuilt     custom
## rss 1 0.02844427 0.02624694
## rss 2 0.02765805 0.02668915
## rss 3 0.02765805 0.02777303
```

Without specifying the threshold value, the inbuilt `setar` function finds threshold values quite close to those of our custom procedures. The AR order `p` for both regimes, however, has to be specified in advance. The comparison of the model coefficients suggests that our custom method was more-or-less accurate.

2.6: Conclusion

The results of the SETAR Parameter Estimation Procedure in section 2.3 show that the 3 best 2-regime SETAR models are:

```
##     unlist.results.
## 1 SETAR(2,2,0.1029)
## 2 SETAR(3,2,0.1029)
## 3 SETAR(3,1,0.3007)
```

The first model with the lowest `BIC` (Bayesian Information Criterion) has the most accurate estimation of its 4 regression parameters, with the highest residual square sum. The first model seems to have a stable equilibrium at their threshold values.

3: Tests of Linearity/Nonlinearity of SETAR models

We need to make sure a non-linear model (SETAR, for example) is really suitable for describing the process. In order to find out, we test the null hypothesis that a linear model is more suitable than a non-linear one. In the case of a 2-regime model we are looking for, so called, nuisance parameters, i.e.: $H_0 : \Phi_1 = \Phi_2$ where Φ_1 and Φ_2 are the parameters of the low and the high regime respectively.

3.1: Hansen's Conditions

Hansen proposed three conditions to test whether a SETAR model can be tested for linearity using the so called Likelihood-Ratio (LR) test:

```
Hansen <- function(d, c, Phi) {
  p <- (length(Phi)/2) - 1
  #separate regimes into rows
  Phi <- do.call(rbind, split(Phi,rep(1:2,each=(p + 1))))
```

```r
  # (p10-p20)+(p1d-p2d)*c <= 0
  c1 <- !isTRUE(all.equal( 0, apply(Phi[,c(1,1 + d),drop=F],2, diff) %*% c(1,c) ))
  # p1j neq p2j, j notin {0,d}
  c2 <- all(apply(Phi[,-c(1,1 + d),drop=F], 2, function(x) !identical(0, diff(x))))
  # sum_j|pij| < 1 forall i=1,2
  c3 <- all(apply(Phi[,-1,drop=F], 1, function(x) sum(abs(x))) < 1)
  c(cond1=c3, cond2=c2, cond3=c3)
}
```

If all three are satisfied the model can be tested using the LR test:

```
##              cond1 cond2 cond3
## 220.103 /    FALSE  TRUE FALSE
## 320.103 /    FALSE  TRUE FALSE
## 310.301 /    FALSE  TRUE FALSE
## 110 /         TRUE  TRUE  TRUE
## 210.205 /     TRUE  TRUE  TRUE
## 420.103 /    FALSE  TRUE FALSE
## 410.301 /    FALSE  TRUE FALSE
## 440.205 /    FALSE  TRUE FALSE
## 33-0.197 /    TRUE  TRUE  TRUE
## 550.103 /    FALSE  TRUE FALSE
## 540.205 /    FALSE  TRUE FALSE
## 520.103 /    FALSE  TRUE FALSE
```

It appears that only the first and the fifth model can be tested using the LR test. The rest will have to be assessed using the Lagrange Multiplier (LM) test.

3.2: LR and LM Tests

In this section we formulate the basic procedures for the LR (Likelihood Ratio), and LM (Lagrange Multiplier) tests:

```r
LRtest <- function(x, p, var, alpha=0.05) {
  tmp <- ar(x, aic=F, order.max=pmax, method = "ols")
  tmp <- tmp$var.pred  # linear model residual variance
  testat <- length(x)*(tmp-var)/tmp  # test statistic
  CDF <- Vectorize( function(t) {  # test statistic CDF
    fun <- function(t) 1 + sqrt(t/(2*pi))*exp(-t/8) + 1.5*exp(t)*pnorm(-1.5*sqrt(t)) -
      (t+5)*pnorm(-sqrt(t)/2)/2
    if(abs(t)>300 || is.infinite(t)) return(sign(t))
    if(t >=0 ) fun(t) else 1-fun(-t)
  })
  # for alpha=2.5%: CV=11.03329250
  if(alpha==0.05) critval <- 7.68727553
  else critval <- uniroot(function(x) CDF(x) - (1-alpha), c(-1000,1000))$root
  # (test statistics, critical value, p-value)
  c(TS=testat, CV=critval, p_value=1-CDF(testat))
}

LRtest(xt, models[[ orders[1] ]]$p, models[[ orders[1] ]]$resSigmaSq)
```

```
##          TS          CV     p_value
## 17.152950411  7.687275530  0.007928351
```

```r
suppressMessages(pkgTest("dynlm"))

LMtest <- function(x, p, d, alpha = 0.05) {
  # prevent from passing (accidental and needless) name to result
  names(p) <- NULL
  # if x is not a ts object, by chance
```

```r
  x <- as.ts(x)
  # requires dynlm package (it can be implemented withou dynlm, see model2)
  model1 <- dynlm(x ~ L(x,1:p))
  y <- model1$residuals
  # a list of shifted time series
  tmp <- c(
    list(y),
    lapply(1:p, function(i) stats::lag(x, -i)),
    lapply(1:p, function(i) stats::lag(x, -i)*stats::lag(x,-d)),
    list(stats::lag(x,-d)^3)
  )
  tmp <- do.call(function(...) ts.intersect(..., dframe=T), tmp)
  names(tmp) <- c("y", paste0("x",1:p), paste0("xd",1:p), "xd^3")
  # cannot be done with the dynlm package
  model2 <- lm(y ~ ., data = tmp)
  z <- model2$residuals
  testat <- (length(x)-p) * (sum(y^2)/sum(z^2) - 1)
  c(TS=testat, CV=qchisq(1-alpha, df=p+1), p_value=1-pchisq(testat, df=p+1))
}

LMtest(xt, models[[ orders[1] ]]$p, models[[ orders[1] ]]$d)
```

```
##           TS           CV      p_value
## 1.696816e+01 7.814728e+00 7.174773e-04
```

We can easily automate the testing procedure with the following results:

```
##    p d         c Hansen Cond.        TS        CV      p-value
## 3  2 2  0.102875        FALSE 16.968165  7.814728 0.0007174773
## 5  3 2  0.102875        FALSE 16.343839  9.487729 0.0025908411
## 4  3 1  0.300650        FALSE  8.351652  9.487729 0.0795136152
## 1  1 1  0.000325         TRUE -2.135613  7.687276 0.7970372758
## 2  2 1  0.205425         TRUE  0.613035  7.687276 0.3539835179
## 8  4 2  0.102875        FALSE 16.214712 11.070498 0.0062570609
## 7  4 1  0.300650        FALSE 12.687710 11.070498 0.0264878050
## 10 4 4  0.205425        FALSE 10.173680 11.070498 0.0704610459
## 6  3 3 -0.197450         TRUE  8.190160  7.687276 0.0448575191
## 15 5 5  0.102875        FALSE 15.615335 12.591587 0.0159745090
## 14 5 4  0.205425        FALSE 10.919148 12.591587 0.0909076284
## 12 5 2  0.102875        FALSE 17.915958 12.591587 0.0064456881
```

For significance level `alpha = 0.05` the linearity hypothesis is not rejected only for the following models:

```
##                    unlist.res.
## 1 SETAR(2,2,0.102875000000001)
## 2 SETAR(3,2,0.102875000000001)
## 3 SETAR(4,2,0.102875000000001)
```

The remaining models can be considered non-linear.

3.3 Modified LR Test Via Boostrapping

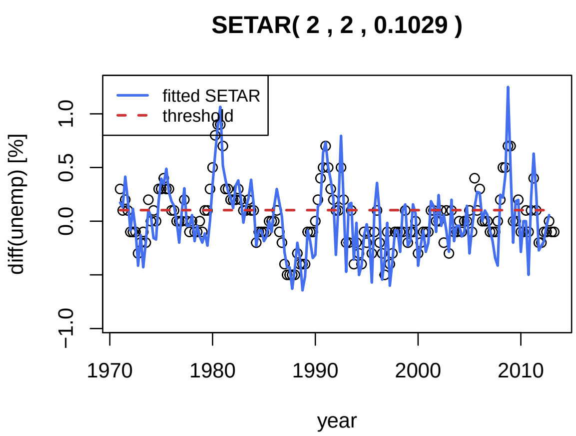
The proposed LR test has a significant drawback in the fact that it can only be done when Hansen's conditions are satisfied. This is due to the fact that we do not know the distribution of the resulting F-statistic. According to Hansen (1996), however, the distribution of a bootstrapped statistic $F*$ converges weakly in probability to the distribution of F, so that repeated bootstrap draws from $F*$ can be used to approximate the asymptotic distribution of F. A parallelized implementation can be seen in the following snippet:

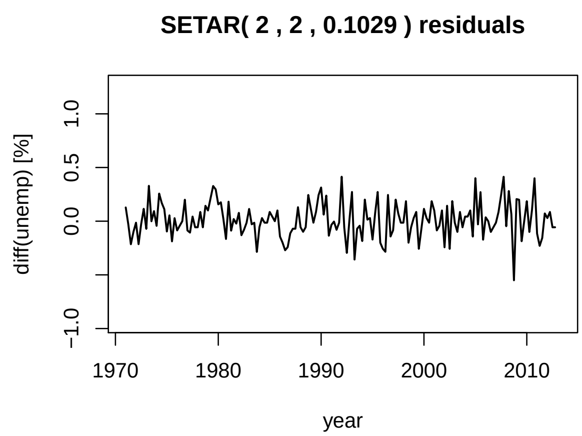
```
  ...
```

```
if (FALSE %in% Hansen(p, d, model$PhiParams)) {
  suppressMessages(pkgTest("tsDyn"))
  suppressMessages(pkgTest("parallel"))
  suppressMessages(pkgTest("doSNOW")) # using doSNOW package for parllel computing
  n_cores <- detectCores() - 1
  cl <- makeCluster(n_cores, type="SOCK")
  registerDoSNOW(cl)
  log <- capture.output({
    testResults <- suppressWarnings(
      setarTest(x, m=p, thDelay=0:(d - 1), nboot=nboot ,trim=0.1, test="1vs", hpc="foreach")
    )
  })
  stopCluster(cl)
  ...
}
...
```

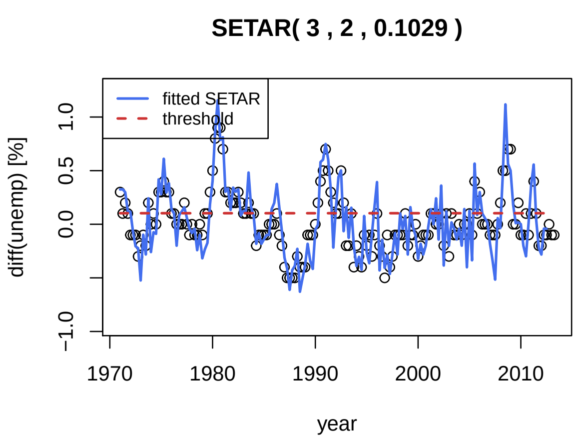
3.4 Visualisation of Non-Linear Models

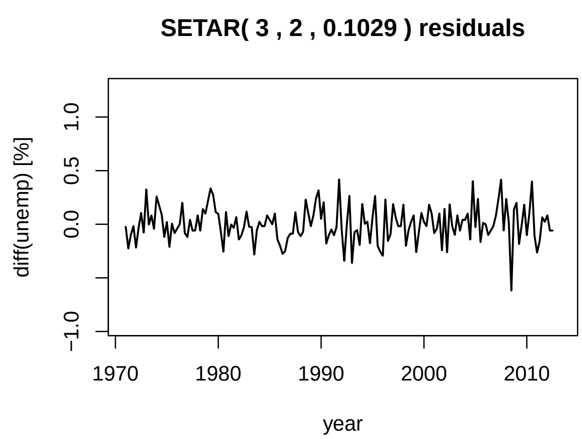
From the results of the previous procedure, we will visualize the models for which the linearity null-hypothesis was rejected based on the LR and LM tests:



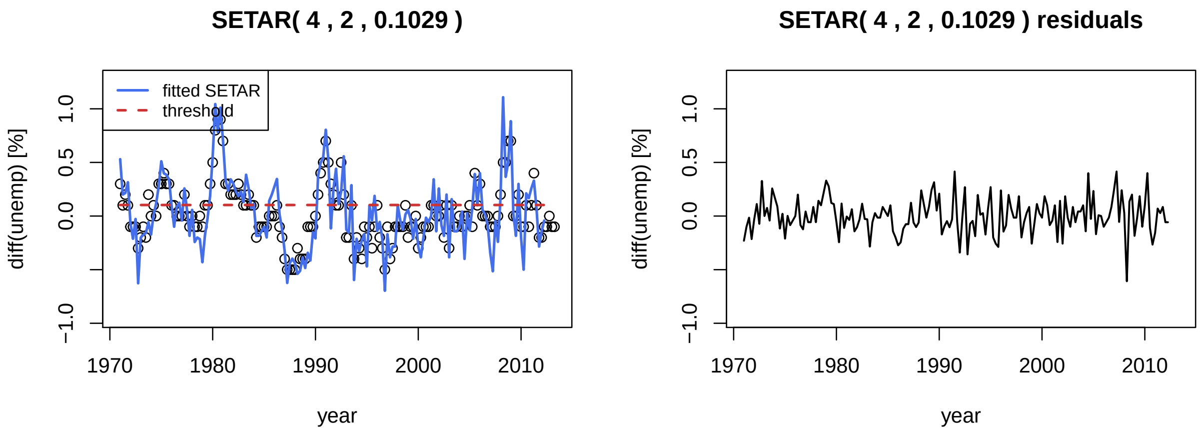



```
## resSigmaSq
## 0.02624694
```

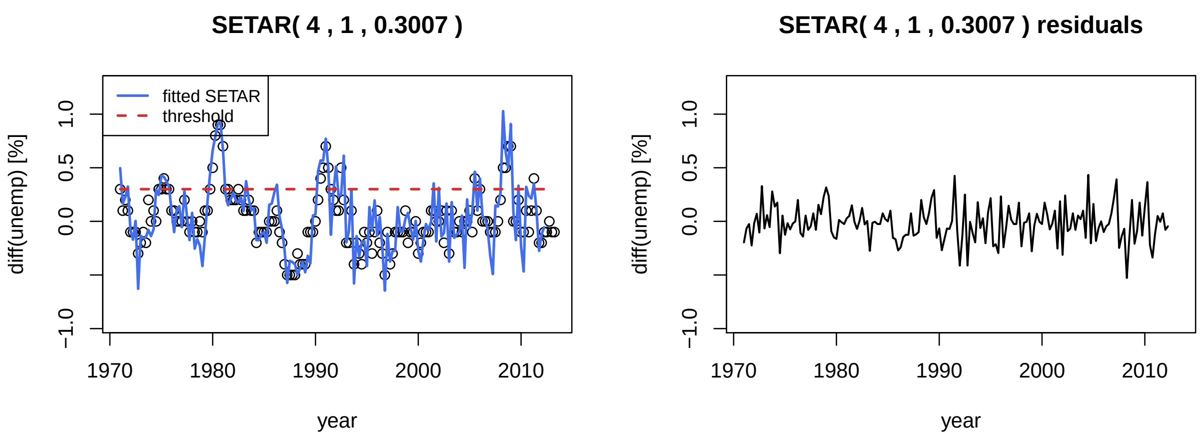




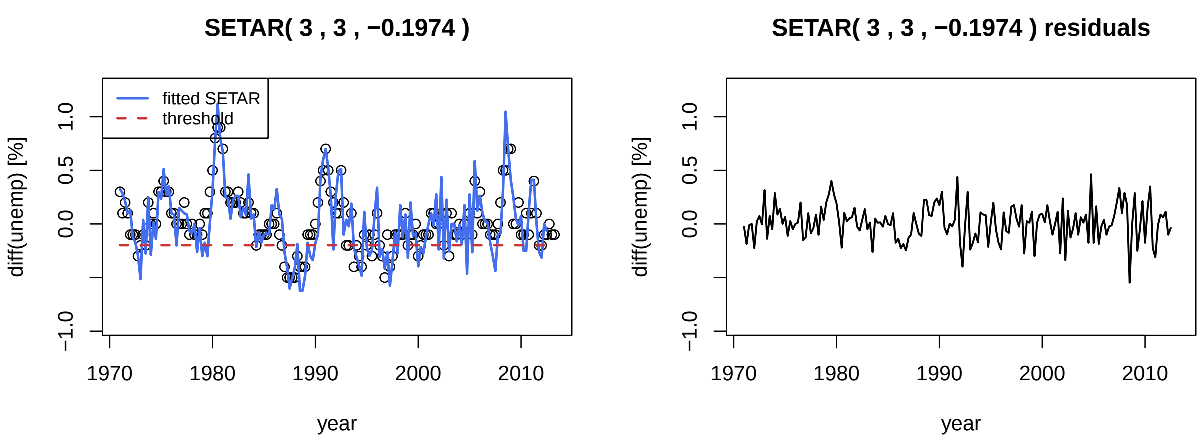
```
## resSigmaSq
## 0.02668915
```



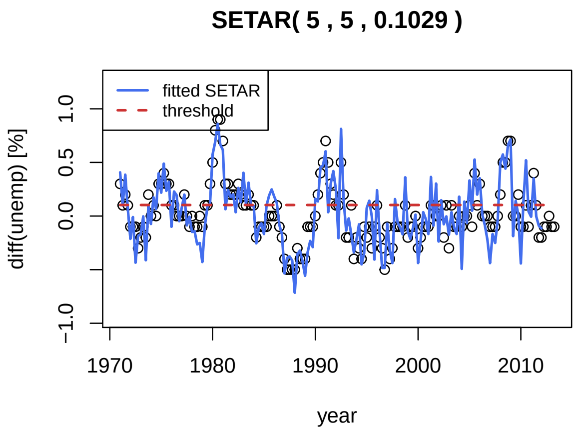

```
## resSigmaSq
## 0.02681811
```

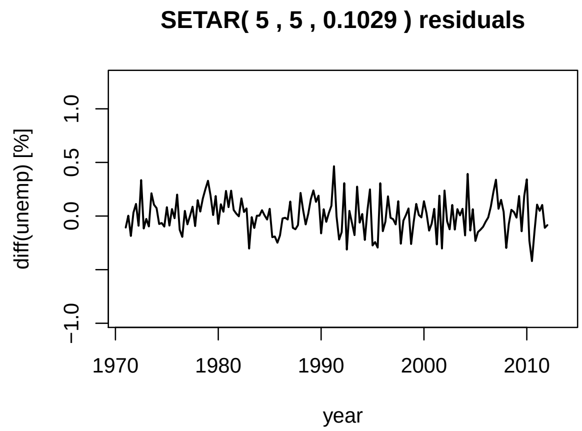



```
## resSigmaSq
## 0.02800968
```

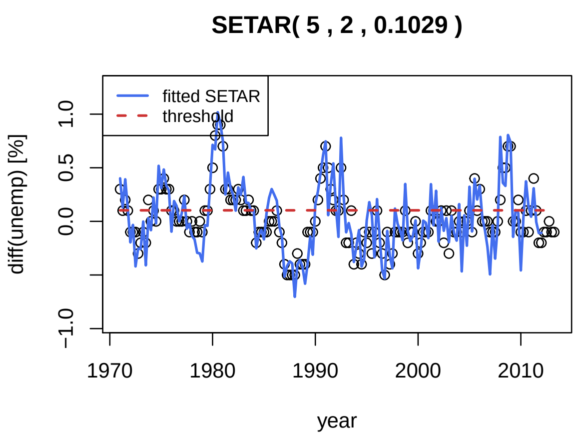


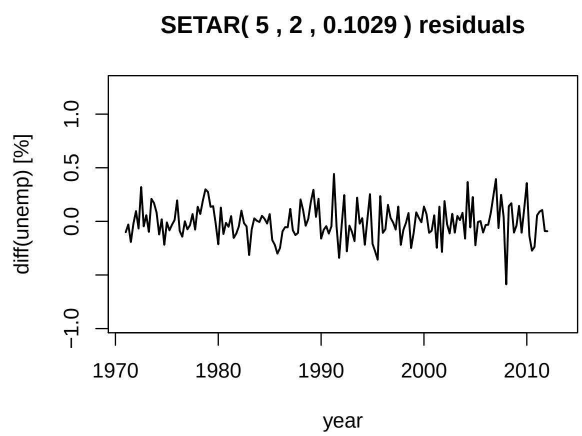
```
## resSigmaSq
## 0.02778604
```






```
## resSigmaSq
## 0.02542239
```






```
## resSigmaSq
## 0.02590032
```

3.5 Conclusion

Since the differences in the unemployment rate have been used, we show the threshold value as well as the fitted values of the models in the same plot. The switching between the high and the low regimes can is clearly visible for all the selected models (perhaps, except the second one, with its threshold value quite close to zero). It is not yet clear whether another regime should be present in the stochastic process. This will be assessed in the following chapter.

4. 3-Regime SETARs and Diagnostic Tests of SETAR Models

The next step in the analysis using SETAR models is verifying whether 2 regimes suffice. If they do not, we will have to consider the possibility that a third regime needs to be added. In that case, we need to write methods for such model

4.1 Useful Functions

```
# the indicator function for 3 regimes:
Indicator3 <- function(x, c) {
  tmp <- rep(F,3)
  tmp[findInterval(x, c, left.open = T) + 1] <- T
  tmp
}

Indicator3(-4, c(-1,1))
```

```
## [1]  TRUE FALSE FALSE
```

```
Indicator3(0, c(-1,1))
```

```
## [1] FALSE  TRUE FALSE
```

```
Indicator3(4, c(-1,1))
```

```
## [1] FALSE FALSE  TRUE
```

```
# SETAR3 basis vector
Yt3 <- function(x, t, p) c(1, x[(t - 1):(t - p)])

# SETAR3 skeleton
Xt3 <- function(x, t, p, d, c, z = x) {
  # z is the threshold variable
  I <- Indicator3(z[t - d], c)
  Y <- Yt(x, t, p)
  c(I[1] * Y, I[2] * Y, I[3] * Y)
}

# covariance matrix of the 3 regime SETAR
CovMat3 <- function(x, p, d, c) {
  n <- length(x)
  # this will become the covariance matrix
  Yc <- matrix(0., ncol = (3 * p + 3), nrow = (3 * p + 3))
  k <- max(p, d)
  for (t in (k + 1):n) {
    XT <- Xt3(x, t, p, d, c)
    Yc <- Yc + (XT %o% XT)
  }
  det <- det(Yc)
  if (det > -0.00001 && det < 0.00001) {
    return(NA)
  } else {
    return(inv(Yc))
  }
}

CovMat3(xt, p=2, d=1, c=c(-0.1, 0.2))
```

```
##             [,1]       [,2]        [,3]        [,4]        [,5]       [,6]
## [1,]  0.09009639  0.2885639 -0.00608143  0.00000000  0.00000000  0.0000000
## [2,]  0.28856387  1.5217885 -0.52175095  0.00000000  0.00000000  0.0000000
## [3,] -0.00608143 -0.5217510  0.75481283  0.00000000  0.00000000  0.0000000
## [4,]  0.00000000  0.0000000  0.00000000  0.01209957 -0.01656548  0.0002522
## [5,]  0.00000000  0.0000000  0.00000000 -0.01656548  1.66822853 -0.3459105
## [6,]  0.00000000  0.0000000  0.00000000  0.00025220 -0.34591053  0.4059073
## [7,]  0.00000000  0.0000000  0.00000000  0.00000000  0.00000000  0.0000000
## [8,]  0.00000000  0.0000000  0.00000000  0.00000000  0.00000000  0.0000000
```

```
## [9,] 0.00000000 0.0000000 0.00000000 0.00000000 0.00000000 0.0000000
##             [,7]       [,8]        [,9]
## [1,]  0.00000000  0.0000000  0.00000000
## [2,]  0.00000000  0.0000000  0.00000000
## [3,]  0.00000000  0.0000000  0.00000000
## [4,]  0.00000000  0.0000000  0.00000000
## [5,]  0.00000000  0.0000000  0.00000000
## [6,]  0.00000000  0.0000000  0.00000000
## [7,]  0.15071243 -0.3415514  0.06411848
## [8,] -0.34155138  1.3811044 -0.73962398
## [9,]  0.06411848 -0.7396240  0.77066336
```

To find out, whether the third regime should be added, we need to test for the independence of residuals:

4.2 The Brock-Dechert-Scheinkman (BDS) Test

Regarded as the most successful tests for nonlinearity due to its universality, the BDS test relies on evaluating a correlation integral $C(q, r)$ as a measure of repeated occurrence of patterns in the time series. It is the estimate of the probability of two arbitrary q-dimensional points in $\mathbb{R}^q$ being no further than $r\hat{\sigma}_\varepsilon$ apart ($0.5 \leq r \leq 1.5$). If the data is generated by an iid process, the correlation integral should approach $C(q, r) \to C(1, r)^q$.

```
## remaining nonlinearity rejected  SETAR(2,2,0.103)  with mean p-val =  0.875
## remaining nonlinearity rejected  SETAR(3,2,0.103)  with mean p-val =  1
## possible remaining nonlinearity for SETAR(3,1,0.301)  with mean p-val =  0
## possible remaining nonlinearity for SETAR(1,1,0)  with mean p-val =  0.125
## possible remaining nonlinearity for SETAR(2,1,0.205)  with mean p-val =  0.125
## remaining nonlinearity rejected  SETAR(4,2,0.103)  with mean p-val =  0.875
## possible remaining nonlinearity for SETAR(4,1,0.301)  with mean p-val =  0.25
## remaining nonlinearity rejected  SETAR(4,4,0.205)  with mean p-val =  0.75
## possible remaining nonlinearity for SETAR(3,3,-0.197)  with mean p-val =  0.25
## remaining nonlinearity rejected  SETAR(5,5,0.103)  with mean p-val =  0.625
## remaining nonlinearity rejected  SETAR(5,4,0.205)  with mean p-val =  0.625
## remaining nonlinearity rejected  SETAR(5,2,0.103)  with mean p-val =  0.75

##
## ==== Remaining nonlinearities detected for: ======

## [1] "SETAR(3,1,0.301)"  "SETAR(1,1,0)"      "SETAR(2,1,0.205)"
## [4] "SETAR(4,1,0.301)"  "SETAR(3,3,-0.197)"
```

4.3 SETAR3 Parameter Estimation

Similarly to section 2.3, we construct an estimation procedure with two distinct threshold parameters c_1 and c_2. Our helper functions are ready from section 4.1. Using them we implement:

```
suppressMessages(pkgTest("zeallot"))
suppressMessages(pkgTest("matlib"))

EstimSETAR3 <- function(x, p, d, c) {
    resultModel <- list()
    resultModel$p = p; resultModel$d = d; resultModel$c = c;
    resultModel$data = x;  n = length(x);  resultModel$n = n;

    k <- max(p, d)

    X <- as.matrix(apply(as.matrix((k + 1):n), MARGIN=1, function(t) Xt3(x, t, p, d, c) ))
    y <- as.matrix(x[(k + 1):n])
    K <- CovMat3(x, p, d, c); b <- crossprod(t(X), y);
```

```r
    if (as.logical(sum(is.na(K)))) {
      return(NA)
    } else {
      sol_phi <- as.numeric(t(K %*% b)); sol_se <- sqrt(diag(K)/n);
      eps <- 0.01;

      # filter out those coeffs that are of the same order of magnitude as their errors
      filter <- sapply(1:(3*(p + 1)), function (i) ifelse(
          abs(sol_phi[i]) <= 2 * abs(sol_se[i]), 0, 1
        )
      )

      sol_phi <- sol_phi * filter
      sol_se <- sol_se * filter

      solution <- cbind(phi = sol_phi,  se = sol_se)
      resultModel$PhiParams <- solution[,1] # solving (X'X)*phi = X'y
      resultModel$PhiStErrors <- solution[,2]  # standard errors
      skel <- crossprod(X, resultModel$PhiParams); resultModel$skel <- skel;
      resultModel$residuals <- (y - skel)
      resultModel$resSigmaSq <- 1 / (n - k) * sum(resultModel$residuals ^ 2)

      return(resultModel)
    }
}

str( test_model <- EstimSETAR3(xt, p=2, d=1, c=c(-0.1, 0.2)) )
```

```
## List of 10
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num [1:2] -0.1 0.2
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:9] 0.0711 0.7998 0.1755 -0.0171 0.3404 ...
##  $ PhiStErrors: num [1:9] 0.02302 0.09461 0.06663 0.00844 0.09906 ...
##  $ skel       : num [1:168, 1] 0.0908 0.1688 0.0662 -0.0265 -0.0264 ...
##  $ residuals  : num [1:168, 1] 0.1092 -0.0688 -0.1662 -0.0735 -0.0736 ...
##  $ resSigmaSq : num 0.0285
```

After we find a suitable SETAR3 model, we implement a postprocessing method:

```r
EstimSETAR3_postproc <- function(model) {
    x <- model$data; k <- max(model$p, model$d); p <- model$p; c <- model$c; n <- model$n;
    y <- as.matrix(x[(k + 1):n])
    skel <- model$skel; # model$skel <- NULL; #skel attribute no longer needed

    # regime counts
    nRegCounts <- rowSums( matrix(as.numeric((sapply(x, function(xi) Indicator3(xi, c)))), nrow=3) )
    model$nRegCounts <- nRegCounts
    # names(model$nRegCounts) <- c("n1", "n2", "n3")

    # regime sigma sq:
    regSigmaSq <- rowSums(
      matrix(
        as.numeric((sapply(y, function(xi) Indicator3(xi, c)))), nrow=3)
       %*% (y - skel)^2 ) / (nRegCounts - k)
    model$regSigmaSq <- regSigmaSq
    # names(model$regSigmaSq) <- c("rss1", "rss2", "rss3")
```

```r
    # count valid p orders
    pOrders <- rowSums(
      matrix(as.numeric((
        sapply(1:(3 * (p + 1)),
               function(i) ifelse( (i %% (p + 1) != 1 && model$PhiParams[i] != 0), 1, 0) )
        )
      ), nrow=3, byrow=T)
    )
    model$pOrders <- pOrders
    # names(model$pOrders) <- c("p1", "p2", "p3")

    model$AIC <- AIC_SETAR(pOrders, model$nRegCounts, model$resSigmaSq)
    model$BIC <- BIC_SETAR(pOrders, model$nRegCounts, model$resSigmaSq)

    return(model)
}

str( test_model <- EstimSETAR3_postproc(test_model) )
```

```
## List of 15
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num [1:2] -0.1 0.2
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:9] 0.0711 0.7998 0.1755 -0.0171 0.3404 ...
##  $ PhiStErrors: num [1:9] 0.02302 0.09461 0.06663 0.00844 0.09906 ...
##  $ skel       : num [1:168, 1] 0.0908 0.1688 0.0662 -0.0265 -0.0264 ...
##  $ residuals  : num [1:168, 1] 0.1092 -0.0688 -0.1662 -0.0735 -0.0736 ...
##  $ resSigmaSq : num 0.0285
##  $ nRegCounts : num [1:3] 50 84 36
##  $ regSigmaSq : num [1:3] 0.0302 0.0178 0.0553
##  $ pOrders    : num [1:3] 2 2 1
##  $ AIC        : num -589
##  $ BIC        : num -573
```

Since the number of regimes m appears as a parameter in the implementation above, we can compose a generalized method for estimation and postprocessing for any number of regimes:

```r
Indicator_m <- function(x, c, m) {
  tmp <- rep(F, m)
  tmp[findInterval(x, c, left.open = T) + 1] <- T
  tmp
}

Xt_m <- function(x, t, p, d, c, m, z = x) {
  I <- Indicator_m(z[t - d], c, m)
  Y <- Yt(x, t, p)
  sapply(1:m, function(j) I[j] * Y)
}

# test for packages
suppressMessages(pkgTest("zeallot"))
suppressMessages(pkgTest("matlib"))


EstimSETAR_m <- function(x, p, d, c, m) {
    m = as.integer(m)
    if (m <= 0) {
```

```r
      message("Error: regime count m has to be a positive integer")
      return(NA)
    }
    if (length(c) != m - 1) {
      message("Error: Incompatible dimensions of threshold vector and regime count.");
      return(NA)
    }

    resultModel <- list()
    resultModel$nReg <- m
    resultModel$p = p; resultModel$d = d; resultModel$c = c;
    resultModel$data = x;  n = length(x);  resultModel$n = n;

    k <- max(p, d)

    X <- as.matrix(apply(as.matrix((k + 1):n), MARGIN=1, function(t) Xt_m(x, t, p, d, c, m) ))
    y <- as.matrix(x[(k + 1):n])
    K <- crossprod(t(X), t(X)); b <- crossprod(t(X), y);

    detK <- abs(det(K))

    if (detK < 0.000001) {
      return(NA)
    } else {
      K <- inv(K)
      sol_phi <- as.numeric(t(K %*% b)); sol_se <- sqrt(diag(K)/n);
      eps <- 0.01;

      # filter out those coeffs that are of the same order of magnitude as their errors
      filter <- sapply(1:(m*(p + 1)), function (i) ifelse(
          abs(sol_phi[i]) <= 2 * abs(sol_se[i]), 0, 1
        )
      )

      sol_phi <- sol_phi * filter
      sol_se <- sol_se * filter

      solution <- cbind(phi = sol_phi,  se = sol_se)
      resultModel$PhiParams <- solution[,1] # solving (X'X)*phi = X'y
      resultModel$PhiStErrors <- solution[,2]  # standard errors
      skel <- crossprod(X, resultModel$PhiParams); resultModel$skel <- skel;
      resultModel$residuals <- (y - skel)
      resultModel$resSigmaSq <- 1 / (n - k) * sum(resultModel$residuals ^ 2)

      return(resultModel)
    }
}

EstimSETAR_m_postproc <- function(model) {
    m <- model$nReg; x <- model$data; n <- model$n;
    k <- max(model$p, model$d); p <- model$p; c <- model$c;
    y <- as.matrix(x[(k + 1):n])
    skel <- model$skel;

    # regime counts
    nRegCounts <- rowSums( matrix(as.numeric( sapply(x, function(xi) Indicator_m(xi, c, m)) ), nrow=m) )
    model$nRegCounts <- nRegCounts
```

```r
    # count valid p orders
    pOrders <- rowSums(
      matrix(as.numeric((
        sapply(1:(m * (p + 1)),
               function(i) ifelse( (i %% (p + 1) != 1 && model$PhiParams[i] != 0), 1, 0) )
        )
      ), nrow=m, byrow=T)
    )
    model$pOrders <- pOrders

    k_m <- pmax(pOrders, model$d) # k-offset for different regimes

    # regime sigma sq:
    regSigmaSq <- rowSums(
      matrix(as.numeric(
        sapply(y, function(xi) Indicator_m(xi, c, m)) ), nrow=m) %*% (y - skel)^2 ) / (nRegCounts - k_m)
    model$regSigmaSq <- regSigmaSq
    # names(model$regSigmaSq) <- sapply(1:m, function(j) paste0("rss", j))

    model$AIC <- AIC_SETAR(pOrders, model$nRegCounts, model$resSigmaSq)
    model$BIC <- BIC_SETAR(pOrders, model$nRegCounts, model$resSigmaSq)

    c <- round(c, digits=3) # 3 dec. places seems enough
    model$name <- paste0("SETAR(", p, ",", d, ",", paste(na.omit(c), collapse=','), ")")

    return(model)
}

getModelName <- function(model) {
  p <- model$p; d <- model$d; c <- round(model$c, digits=3) # 3 dec. places seems enough
  paste0("SETAR(", p, ",", d, ",", paste(na.omit(c), collapse=','), ")")
}


str( EstimSETAR_m_postproc( EstimSETAR_m(xt, p=2, d=1, c=-0.1, m=2) ) ) # 2 regimes
```

```
## List of 17
##  $ nReg       : int 2
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num -0.1
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:6] 0.0711 0.7998 0.1755 0 0.6911 ...
##  $ PhiStErrors: num [1:6] 0.023 0.0946 0.0666 0 0.0444 ...
##  $ skel       : num [1:168, 1] 0.1162 0.1539 0.1005 -0.0534 -0.0264 ...
##  $ residuals  : num [1:168, 1] 0.0838 -0.0539 -0.2005 -0.0466 -0.0736 ...
##  $ resSigmaSq : num 0.0293
##  $ nRegCounts : num [1:2] 50 120
##  $ pOrders    : num [1:2] 2 2
##  $ regSigmaSq : num [1:2] 0.0357 0.0272
##  $ AIC        : num -588
##  $ BIC        : num -574
##  $ name       : chr "SETAR(2,2,-0.1)"
```

```r
str( EstimSETAR_m_postproc( EstimSETAR_m(xt, p=2, d=1, c=c(-0.1, 0.2), m=3) ) ) # 3 regimes
```

```
## List of 17
##  $ nReg       : int 3
```

```
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num [1:2] -0.1 0.2
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:9] 0.0711 0.7998 0.1755 -0.0171 0.3404 ...
##  $ PhiStErrors: num [1:9] 0.02302 0.09461 0.06663 0.00844 0.09906 ...
##  $ skel       : num [1:168, 1] 0.0908 0.1688 0.0662 -0.0265 -0.0264 ...
##  $ residuals  : num [1:168, 1] 0.1092 -0.0688 -0.1662 -0.0735 -0.0736 ...
##  $ resSigmaSq : num 0.0285
##  $ nRegCounts : num [1:3] 50 84 36
##  $ pOrders    : num [1:3] 2 2 1
##  $ regSigmaSq : num [1:3] 0.0302 0.0178 0.0538
##  $ AIC        : num -589
##  $ BIC        : num -573
##  $ name       : chr "SETAR(2,2,-0.1,0.2)"
```

```
str( EstimSETAR_m_postproc( EstimSETAR_m(xt, p=2, d=1, c=c(-0.1, 0.1, 0.2), m=4) ) ) # 4 regimes
```

```
## List of 17
##  $ nReg       : int 4
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num [1:3] -0.1 0.1 0.2
##  $ data       : num [1:170] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 170
##  $ PhiParams  : num [1:12] 0.0711 0.7998 0.1755 0 0.4321 ...
##  $ PhiStErrors: num [1:12] 0.023 0.0946 0.0666 0 0.1364 ...
##  $ skel       : num [1:168, 1] 0.1028 0.1688 0.083 -0.0233 -0.0264 ...
##  $ residuals  : num [1:168, 1] 0.0972 -0.0688 -0.183 -0.0767 -0.0736 ...
##  $ resSigmaSq : num 0.0285
##  $ nRegCounts : num [1:4] 50 65 19 36
##  $ pOrders    : num [1:4] 2 2 1 1
##  $ regSigmaSq : num [1:4] 0.0305 0.0172 0.0206 0.0536
##  $ AIC        : num -585
##  $ BIC        : num -567
##  $ name       : chr "SETAR(2,2,-0.1,0.1,0.2)"
```

4.4 SETAR Estimation procedure

Now that we prepared all necessary functions we may proceed to search for 3-regime SETAR's in a suitable search space. This time we will construct our outer loop through delays d which will be reduced to only the delays that are contained within the models with detected remaining nonlinearity:

```
##  unique delays:
```

```
## [1] 1 3
```

Still, even after this alleviation, the search might be computationally demanding. To obtain our results within reasonable time we use `foreach` and `doParallel` packages to compute search through c_1 and c_2 thresholds, and then process the results:

```
m <- 3 # 3-regime setars
pmax <- 7 # set maximum order p
# limit the c parameter by the 7.5-th and 92.5 percentile
cmin <- as.numeric(quantile(xt, 0.075)); cmax <- as.numeric(quantile(xt, 0.925));
h = (cmax - cmin) / 50 # determine the step by which c should be iterated

models3 <- list()
model3Columns <- list()
```

```r
suppressMessages(pkgTest("foreach"))
suppressMessages(pkgTest("doParallel"))

pkgs <- c("zeallot", "matlib")

n_cores <- (detectCores() - 1)

for (d in delays) {
  for (p in d:pmax) {
    # PARALLEL LOOP

    cl <- makeCluster(n_cores)
    registerDoParallel(cl)
    pdModels <- foreach(c1 = seq(from = cmin, to = cmax - h, by = h), .packages = pkgs) %:%
      foreach(c2 = seq(c1 + h, cmax, by = h), .packages = pkgs) %dopar% {
        # skip models with slim regime
        if(sum(xt > c1 & xt < c2) < length(xt) * 0.15) {
          NA
        } else {
          tmp <- EstimSETAR_m(xt, p, d, c(c1, c2), m) # try to run the function
          # then test whether it returns`NA` as a result
          if (!as.logical(sum(is.na(tmp))) ) {
            list(tmp)
          }
        }
      }

    stopCluster(cl)

    # OLD LOOP:

    #pdModels <- list()
    #for(c1 in seq(from = cmin, to = cmax - h, by = h)) {
    #  for(c2 in seq(c1 + h, cmax, by = h)) {
    #    if(sum(xt > c1 & xt < c2) < length(xt) * 0.15) next # skip models with slim regime
    #    tmp <- EstimSETAR_m(xt, p, d, c(c1, c2), m) # try to run the function
    #    # then test whether it returns`NA` as a result
    #    if (!as.logical(sum(is.na(tmp))) ) {
    #      pdModels[[length(pdModels) + 1]] <- tmp
    #    }
    #  }
    #}

    # frankly, this is a mess, but I can only get this nested list from the parallel loop
    pdOmitted <- lapply(unlist(pdModels, recursive=F),
                        function(m) if(!is.logical(m) && !is.null(m)) m else list(list(resSigmaSq = Inf)))
    sigmas <- as.numeric(lapply(pdOmitted, function(m) m[[1]]$resSigmaSq))
    s_orders <- order(sigmas)
    # only the model whose parameter c gives the lowest residual square sum is chosen for postprocessing
    min_sigma_model <- EstimSETAR_m_postproc(pdOmitted[[ s_orders[1] ]][[1]])

    models3[[length(models3) + 1]] <- min_sigma_model
    model3Columns[[length(model3Columns) + 1]] <- c(
      min_sigma_model$p, min_sigma_model$pOrders, d, round(min_sigma_model$c, digits=4),
      min_sigma_model$nRegCounts,
      min_sigma_model$AIC, min_sigma_model$BIC,
      min_sigma_model$resSigmaSq)
  }
```

```
}
```

```
##     p p1 p2 p3 d      c1     c2  n1 n2 n3       AIC       BIC resSigmaSq
## 6   6  2  4  6 1  0.0077 0.4032 102 55 13 -606.0682 -584.2020 0.02371665
## 1   1  1  1  1 1  0.0077 0.4032 102 55 13 -588.3777 -577.9832 0.02925708
## 10 5  4  3  5 3  0.0077 0.4032 102 55 13 -600.4441 -575.9002 0.02451439
## 9   4  4  3  1 3  0.0077 0.2128 102 40 28 -597.8376 -575.2928 0.02609259
## 3   3  2  2  3 1  0.0077 0.3007 102 51 17 -592.0732 -575.0699 0.02731196
## 8   3  2  3  2 3 -0.1974 0.1102  34 95 41 -594.6764 -574.7411 0.02689692
## 4   4  2  2  3 1  0.0077 0.3007 102 51 17 -590.4034 -573.4001 0.02758155
## 12 7  4  5  1 3  0.0077 0.2128 102 40 28 -599.1070 -573.1844 0.02529623
## 11 6  4  4  2 3  0.0077 0.3007 102 51 17 -597.9831 -572.6995 0.02546401
## 2   2  2  2  1 1  0.0077 0.2128 102 40 28 -587.4819 -571.8759 0.02872771
## 7   7  2  4  6 1  0.0077 0.3007 102 51 17 -594.8218 -571.4552 0.02533871
## 5   5  3  4  5 1  0.0077 0.2128 102 40 28 -585.8058 -558.8683 0.02671882
```

These are the SETAR3 models that can replace the SETAR2's with remaining nonlinearity, ordered by BIC with estimated coefficients:

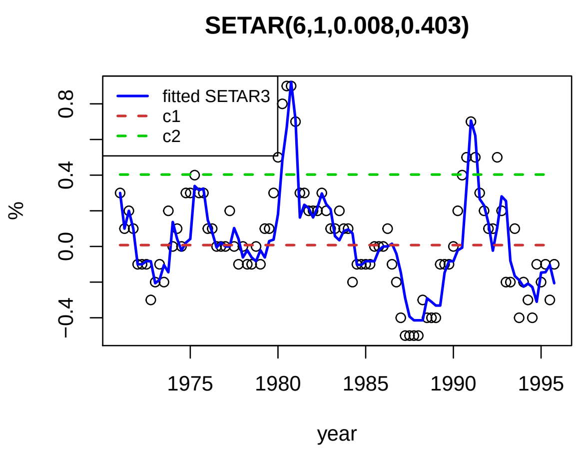
```
## $`6/1/0.0077/0.4032`
##          [,1]       [,2]       [,3] [,4] [,5] [,6] [,7]        [,8]      [,9]
## Phi         0 0.61653921 0.21203788    0    0    0    0 -0.07628129 0.9039121
## stdError    0 0.06602492 0.05388165    0    0    0    0  0.02374417 0.1204506
##               [,10] [,11] [,12]       [,13]      [,14] [,15]     [,16]
## Phi      0.23071211     0     0 -0.38829598 0.37594717     0 0.5952972
## stdError 0.07483189     0     0  0.09668024 0.06779655     0 0.2217849
##              [,17]     [,18]      [,19]      [,20]     [,21]
## Phi      0.5757660 0.8781735 -1.7036164 -0.9607070 0.9337207
## stdError 0.2741307 0.3392136  0.3646186  0.4046036 0.2702892
##
## $`1/1/0.0077/0.4032`
##                [,1]       [,2]        [,3]      [,4]      [,5]      [,6]
## Phi      0.02224840 0.78551645 -0.07770132 0.9679972 0.2052632 0.5394736
## stdError 0.01108097 0.05520027  0.02213275 0.1055138 0.0940983 0.1418585
##
## $`5/3/0.0077/0.4032`
##                [,1]       [,2]       [,3]       [,4] [,5]        [,6] [,7]
## Phi      0.03810950 0.40296041 0.47557956 0.30848816    0 -0.23684544    0
## stdError 0.01142569 0.05153827 0.05854604 0.07903847    0  0.04429775    0
##                [,8] [,9] [,10]      [,11]       [,12] [,13]     [,14]
## Phi      0.95805218    0     0 0.29143233 -0.14816307     0 1.1627497
## stdError 0.06331587    0     0 0.07466087  0.06152756     0 0.1221084
##               [,15]     [,16]      [,17]     [,18]
## Phi      -0.5575423 0.6823293 -1.2225993 0.8829041
## stdError  0.1366731 0.2324756  0.3037091 0.3023727
##
## $`4/3/0.0077/0.2128`
##                [,1]       [,2]       [,3]       [,4]       [,5]        [,6]
## Phi      0.03286233 0.45308572 0.45465353 0.21599102 -0.2175703 -0.19174182
## stdError 0.01138346 0.05067842 0.05841507 0.07712192  0.0479597  0.03715713
##                [,7] [,8]      [,9]      [,10] [,11]      [,12] [,13] [,14]
## Phi      0.97849538    0 1.2970699 0.15215669     0 0.85746204     0     0
## stdError 0.07334453    0 0.2718713 0.07087534     0 0.07761762     0     0
##          [,15]
## Phi          0
## stdError     0
##
## $`3/1/0.0077/0.3007`
##          [,1]       [,2]       [,3] [,4]       [,5]      [,6]       [,7] [,8]
```

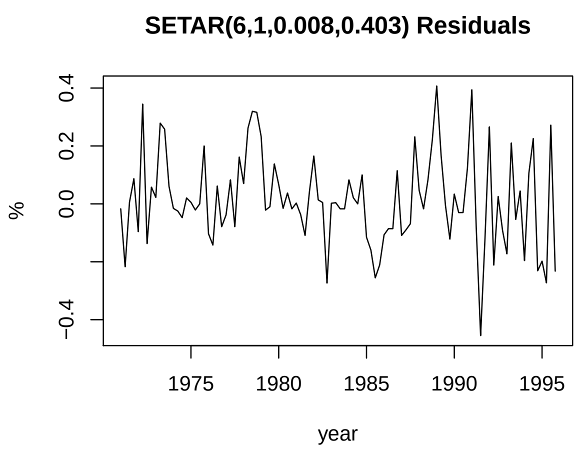
```
## Phi         0 0.63670999 0.20001159    0 -0.11084064 1.0999019 0.19848071    0
## stdError    0 0.06561067 0.05227234    0  0.02629794 0.1631747 0.07297422    0
##                [,9]     [,10]     [,11]      [,12]
## Phi      -0.1620930 1.1391822 0.6438414 -0.9934059
## stdError  0.0792829 0.1808141 0.1619481  0.1556738
##
## $`3/3/-0.1974/0.1102`
##          [,1] [,2]       [,3]      [,4] [,5]       [,6]       [,7]        [,8]
## Phi         0    0 0.46444897 0.3403872    0 0.74034263 0.18395620 -0.33687693
## stdError    0    0 0.08494846 0.1407401    0 0.04993483 0.05472303  0.09788391
##                [,9]      [,10] [,11]       [,12]
## Phi      0.10205804 0.83474301     0 -0.19346821
## stdError 0.02665198 0.07256303     0  0.08388762
##
## $`4/1/0.0077/0.3007`
##          [,1]       [,2]       [,3] [,4] [,5]        [,6]      [,7]       [,8]
## Phi         0 0.63408758 0.19294366    0    0 -0.10658790 1.0854541 0.22081955
## stdError    0 0.06571486 0.05321454    0    0  0.02653244 0.1637477 0.07593937
##          [,9] [,10] [,11]     [,12]     [,13] [,14]      [,15]
## Phi         0     0     0 0.9452945 0.7317165     0 -1.2916701
## stdError    0     0     0 0.1823845 0.1623094     0  0.1591013
##
## $`7/3/0.0077/0.2128`
##                [,1]       [,2]       [,3]       [,4] [,5]        [,6] [,7] [,8]
## Phi      0.03931831 0.38890695 0.48231252 0.31968627    0 -0.20077354    0    0
## stdError 0.01145946 0.05259231 0.05904765 0.07945911    0  0.05230426    0    0
##                [,9]      [,10] [,11]     [,12]     [,13] [,14]     [,15]
## Phi      -0.1848496 1.01184551     0 1.4179174 0.2655614     0 0.3747398
## stdError  0.0393715 0.07692815     0 0.2986228 0.0863464     0 0.1125958
##               [,16] [,17]      [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## Phi      -0.41363322     0 0.89276494     0     0     0     0     0     0
## stdError  0.09747221     0 0.08013842     0     0     0     0     0     0
##
## $`6/3/0.0077/0.3007`
##                [,1]       [,2]       [,3]       [,4] [,5]        [,6] [,7]
## Phi      0.03930124 0.38947090 0.48316931 0.31943638    0 -0.20275456    0
## stdError 0.01145897 0.05247646 0.05880927 0.07944407    0  0.05084832    0
##                 [,8]       [,9] [,10]     [,11]      [,12]       [,13] [,14]
## Phi      -0.07897789 1.02609047     0 0.4209110 0.25941276 -0.26762997     0
## stdError  0.02739269 0.07000051     0 0.1758623 0.08014829  0.07789168     0
##               [,15]      [,16] [,17] [,18] [,19] [,20]     [,21]
## Phi      -0.19522243 0.84735463     0     0     0     0 0.5621783
## stdError  0.08339824 0.08712986     0     0     0     0 0.2253509
##
## $`2/1/0.0077/0.2128`
##          [,1]       [,2]       [,3]        [,4]      [,5]       [,6] [,7]
## Phi         0 0.63927945 0.20483690 -0.10216780 1.0032116 0.23323966    0
## stdError    0 0.06430803 0.04621211  0.03643341 0.2649536 0.06785877    0
##               [,8] [,9]
## Phi      0.8652436    0
## stdError 0.1019782    0
##
## $`7/1/0.0077/0.3007`
##          [,1]       [,2]       [,3] [,4] [,5] [,6] [,7] [,8]        [,9]
## Phi         0 0.61851858 0.21682708    0    0    0    0    0 -0.10673878
## stdError    0 0.06605312 0.05447731    0    0    0    0    0  0.02727445
##              [,10] [,11] [,12] [,13]      [,14]      [,15]       [,16] [,17]
## Phi      1.1952899     0     0     0 -0.4240925 0.54630845 -0.18147747     0
## stdError 0.1674092     0     0     0  0.0984843 0.09049834  0.06325224     0
```

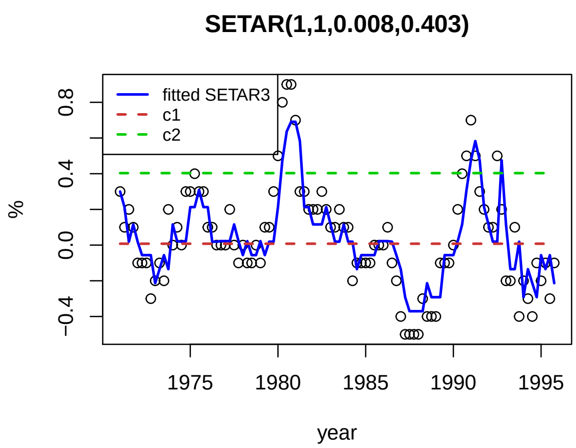
```
##              [,18]     [,19]     [,20]      [,21]     [,22]     [,23] [,24]
## Phi      0.8917613 0.5522111 0.5021494 -1.2558669 -1.195117 1.1338624     0
## stdError 0.1883102 0.1918823 0.2450830  0.2130059  0.380413 0.2572561     0
##
## $`5/1/0.0077/0.2128`
##          [,1]       [,2]       [,3] [,4] [,5]        [,6]        [,7]      [,8]
## Phi         0 0.62415573 0.20979553    0    0 -0.08764835 -0.11761709 1.1852898
## stdError    0 0.06589458 0.05385056    0    0  0.04291969  0.03860717 0.2886749
##                [,9]     [,10] [,11]       [,12]      [,13]     [,14]     [,15]
## Phi      0.16935271 0.2711071     0 -0.23957558 0.08979767 0.4831994 0.7720685
## stdError 0.08296071 0.1008866     0  0.09615684 0.04481110 0.1181061 0.1520152
##               [,16]      [,17]     [,18]
## Phi      -0.3921818 -0.6566286 0.3819791
## stdError  0.1437785  0.1595922 0.1730214
```

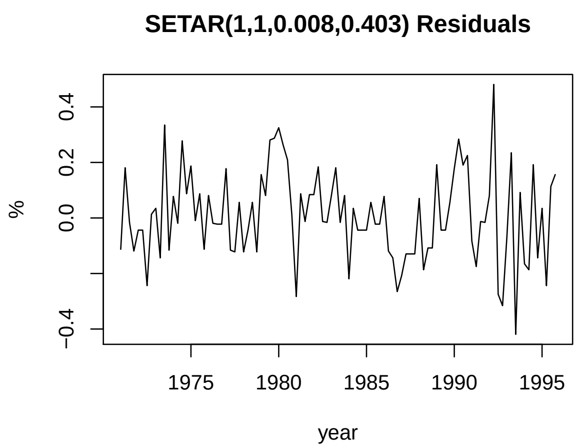
As we might notice, some models differ only in their information criteria and coefficients, since they were estimated from a different maximum order p.

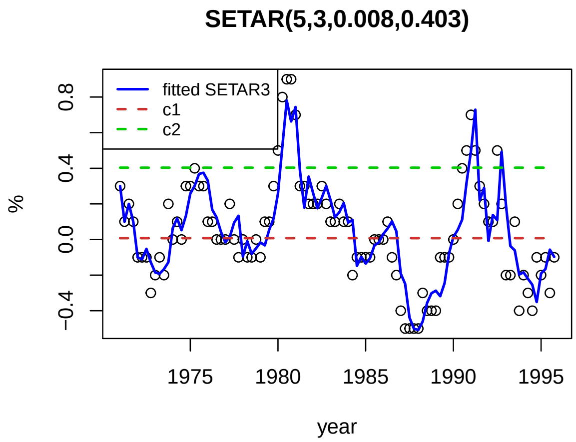
4.5 SETAR3 Visualisation

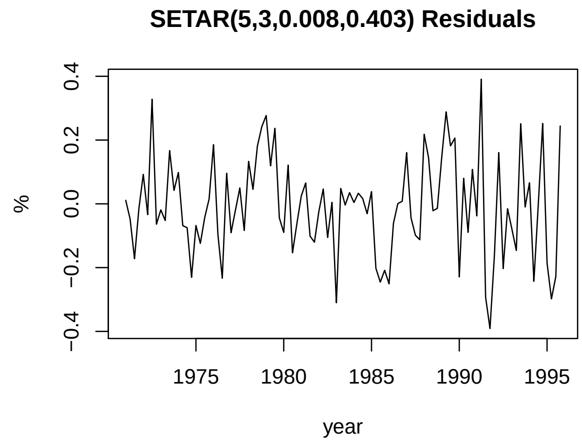


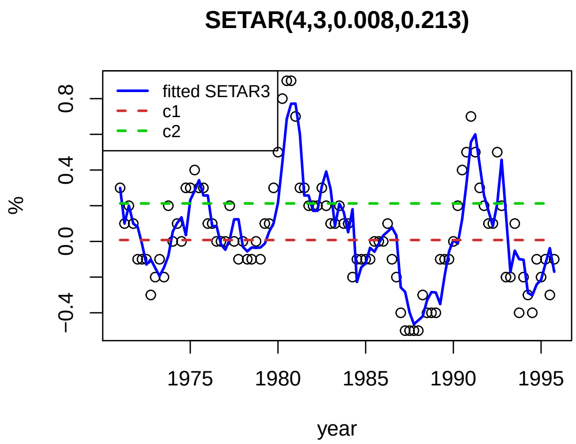


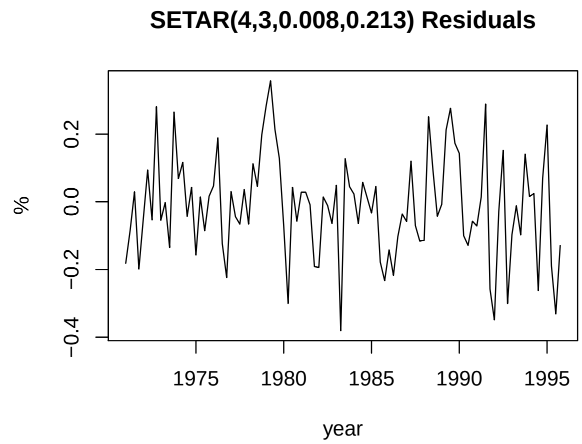


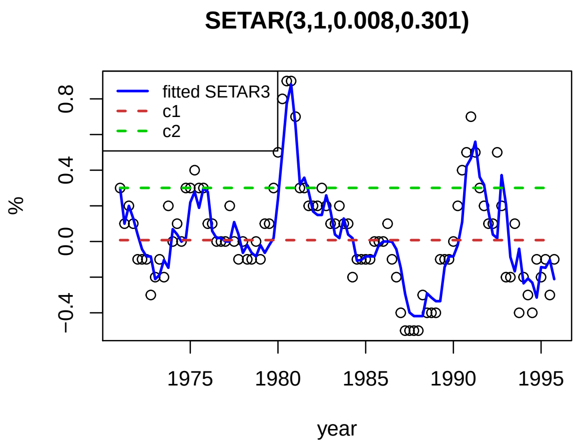


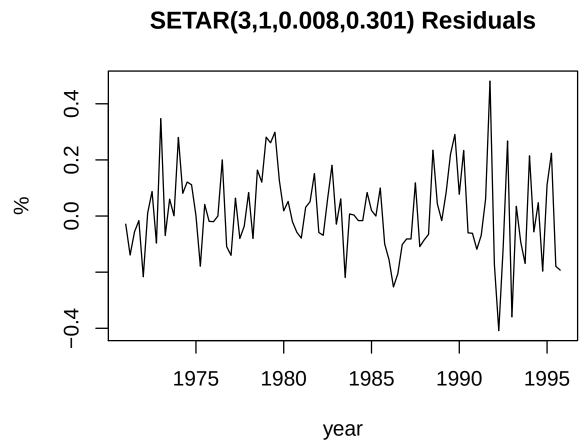

SETAR(5,3,0.008,0.403)
fitted SETAR3
c1
c2
%
year

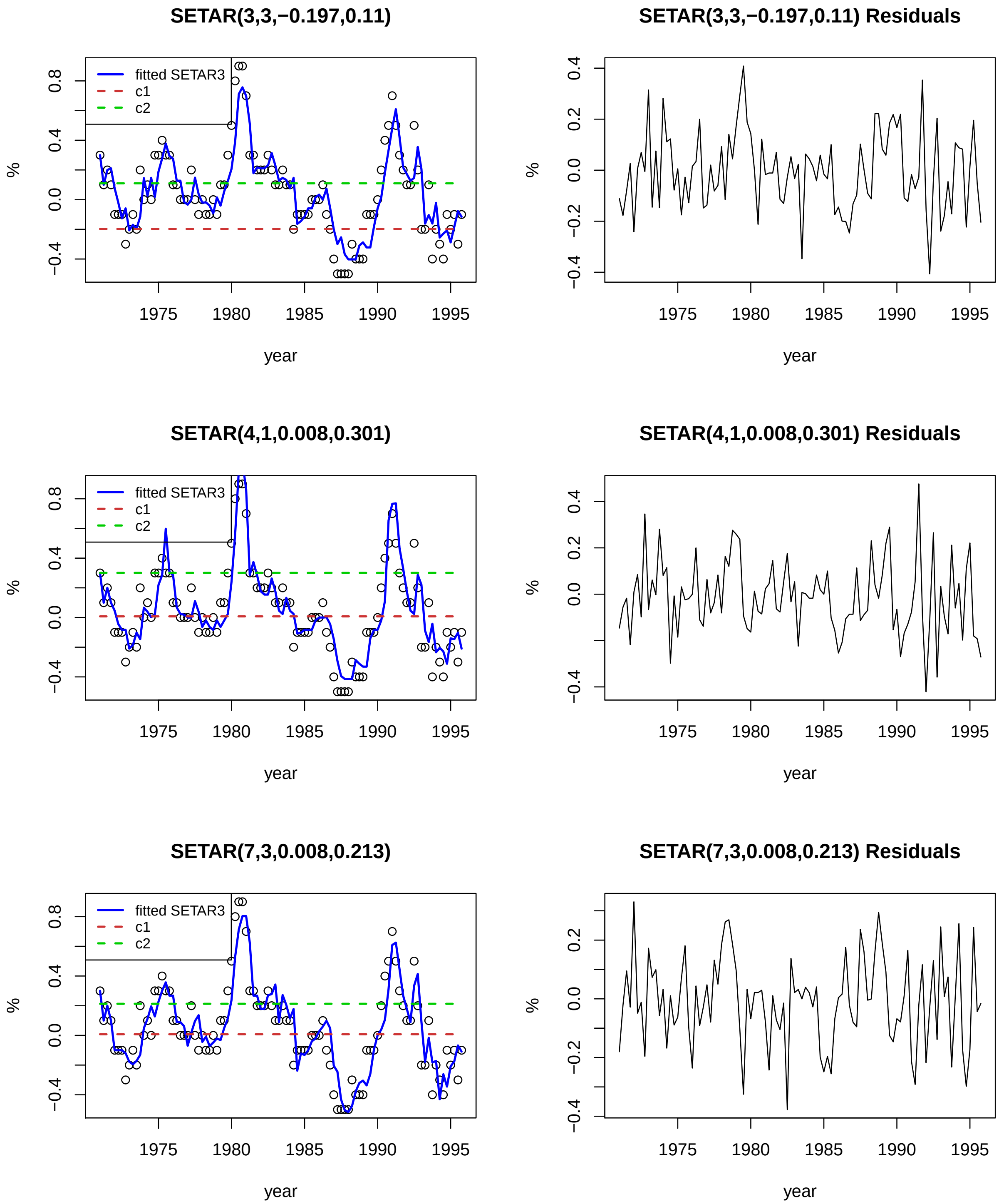

SETAR(5,3,0.008,0.403) Residuals
year


SETAR(4,3,0.008,0.213)
fitted SETAR3
c1
c2
%
year


SETAR(4,3,0.008,0.213) Residuals
year


SETAR(3,1,0.008,0.301)
fitted SETAR3
c1
c2
%
year


SETAR(3,1,0.008,0.301) Residuals
year



4.6 Conclusion

Due to our limited sampling space of `delays` we obtained 12 possible $SETAR3(p, d, c_1, c_2)$ models, and since remaining SETAR3 nonlinearity was detected in 7 of the original SETAR2 models, we replace them by the top 7 newly found SETAR3's:

```
##                    model      BIC
## 1         SETAR(2,2,0.103) -602.841
## 2         SETAR(3,2,0.103) -588.235
## 3   SETAR(6,1,0.008,0.403) -584.202
## 4   SETAR(1,1,0.008,0.403) -577.983
## 5         SETAR(4,2,0.103) -577.642
## 6   SETAR(5,3,0.008,0.403)   -575.9
## 7         SETAR(4,4,0.205) -575.688
## 8   SETAR(4,3,0.008,0.213) -575.293
## 9   SETAR(3,1,0.008,0.301)  -575.07
## 10        SETAR(5,5,0.103) -573.138
## 11        SETAR(5,4,0.205) -571.548
## 12        SETAR(5,2,0.103) -571.394
```

5. Predictions via SETAR Models and Their Evaluation

5.1 Helper functions

Since we have not yet defined a skeleton function, i.e. one that would continue plugging in the time series values even after the end of testing data. For that purpose we implement a step-forward function for an m-regime SETAR:

```r
SETAR_m_singleStep <- function(model, x, t) {
  m <- model$nReg; n <- model$n;
  p <- model$p; d <- model$d; c <- model$c;
  # parameter matrix with regime coefficients by row
  Phi <- matrix(model$PhiParams, nrow=m)

  # extract regime vector
  X <- Xt_m(x, t, p, d, c, m)
  reg_id <- which(colSums(X!=0)!=0)
  x_reg <- X[,reg_id]
  Phi[reg_id,] %*% x_reg
}
```

We can test it on a particular SETAR model:

```r
n_ahead <- 1
model <- models[[ orders[3] ]]
x_out <- c(x_train, rep(0, n_ahead)); nt <- length(x_train)

for (i in 1:n_ahead) {
  x_out[nt + i] <- SETAR_m_singleStep(model, x_out, nt + i)
}
xt <- c(xt, x_eval) # fuse test and eval data
```

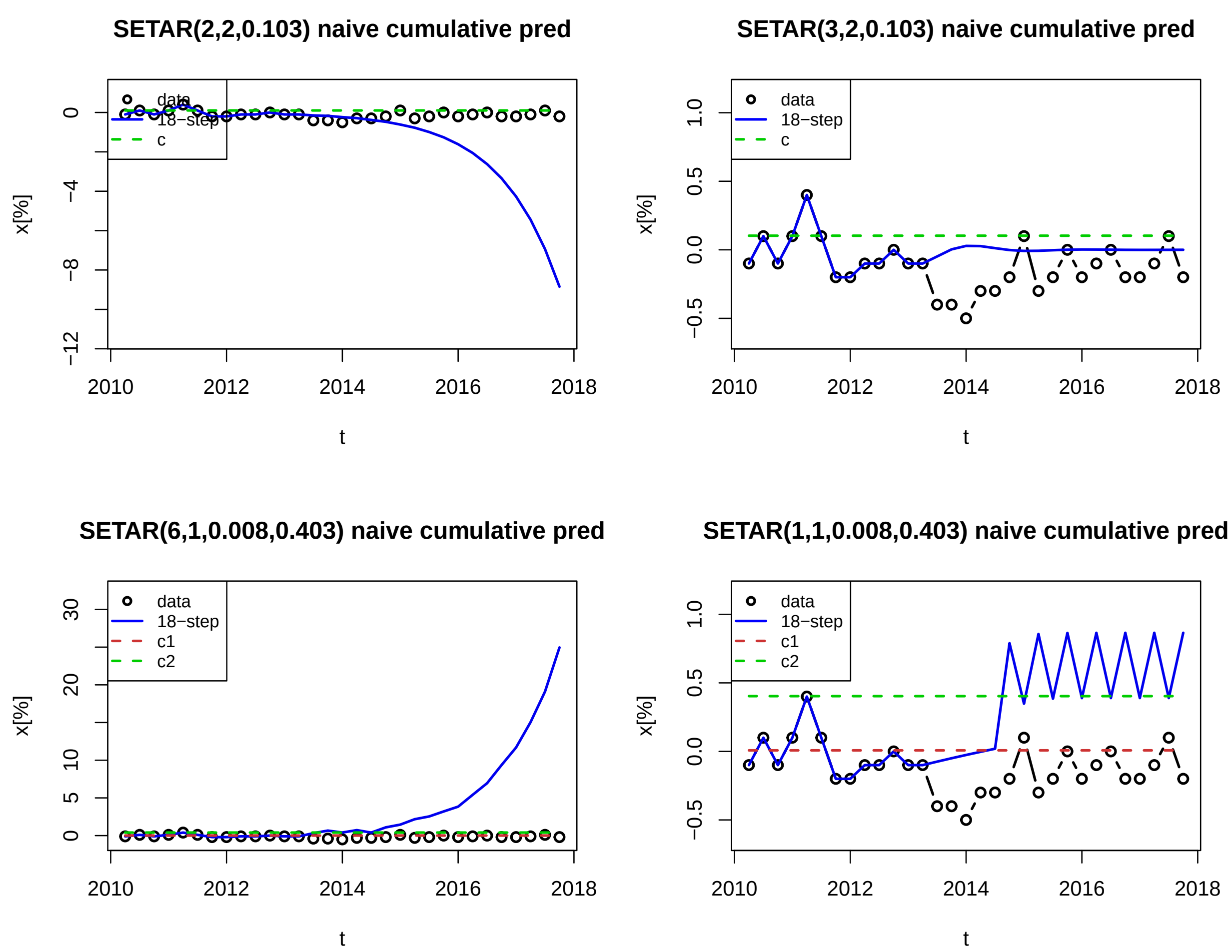
```
## x_out:

## [1] -0.1000000  0.3200232

## data:

## [1] -0.1 -0.4
```

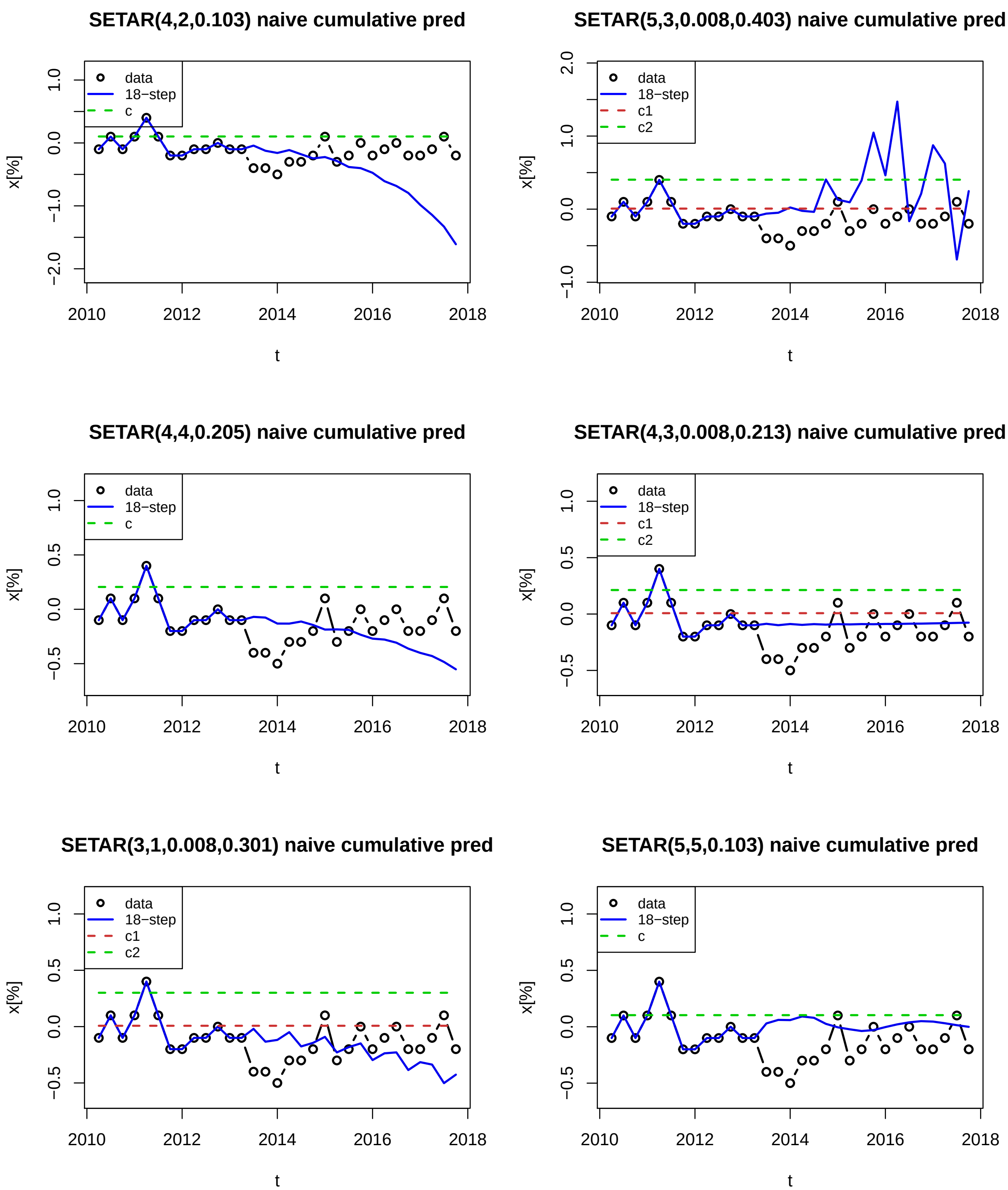
This is a single-step prediction of the data using the first model. When we set `n_ahead > 1`, the step function builds up upon previous predicted values and the skeleton converges to a model's particular equilibrium:

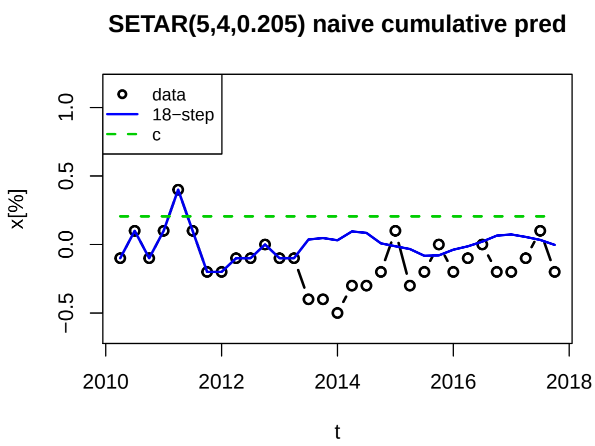
```
##         x_out data
## 1  -0.1000000 -0.1
## 2   0.3200232 -0.4
## 3   0.6626614 -0.4
## 4   0.4079386 -0.5
```

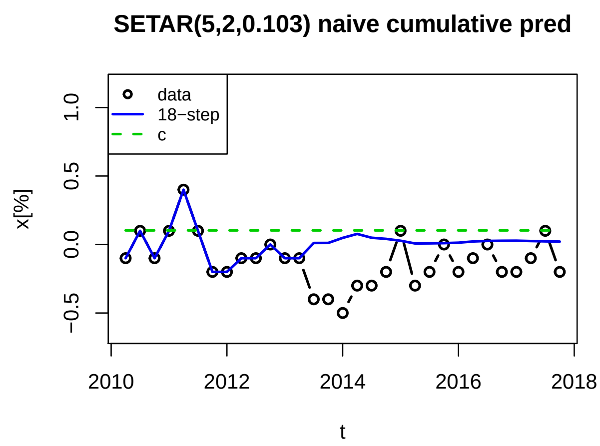
```
## 5   0.7232082 -0.3
## 6   0.3995891 -0.3
## 7   1.0908268 -0.2
## 8   1.4539753  0.1
## 9   2.1750310 -0.3
## 10  2.5423068 -0.2
## 11  3.2042578  0.0
## 12  3.8410996 -0.2
## 13  5.4037755 -0.1
## 14  6.9517158  0.0
## 15  9.3600340 -0.2
## 16 11.6834768 -0.2
## 17 15.0377197 -0.1
## 18 19.1048408  0.1
## 19 24.9552446 -0.2

##
##  equilibria:

## NULL
```

If the model is explosive, the predictions will diverge:




SETAR(4,2,0.103) naive cumulative pred
data
18−step
c
x[%]
t
SETAR(5,3,0.008,0.403) naive cumulative pred
data
18−step
c1
c2
x[%]
t
SETAR(4,4,0.205) naive cumulative pred
data
18−step
c
x[%]
t
SETAR(4,3,0.008,0.213) naive cumulative pred
data
18−step
c1
c2
x[%]
t
SETAR(3,1,0.008,0.301) naive cumulative pred
data
18−step
c1
c2
x[%]
t
SETAR(5,5,0.103) naive cumulative pred
data
18−step
c
x[%]
t





The examples above tested 18 steps of a naiive approach to prediction, by assuming the process evolves via its skeleton. More convenient approaches are `Monte Carlo` (`"MC"`) and `Bootstrap`. Both rely on adding noise to series predictions. Monte Carlo approach simulates normally distributed noise $\epsilon \sim N(0, \hat{\sigma}_\varepsilon)$ from residual standard error $\hat{\sigma}_\varepsilon$, and Bootstrap, on the other hand, does not assume the normality of model residuals and instead randomly samples the residuals themselves.

We will use these two approaches in the following implementation:

```
PredictSETAR <- function(
  model, x_train, x_eval, horizon=(length(x_eval) + 1), n_ahead=1,
  type=c("naive", "MC", "bootstrap"), Nboot=100, alpha=0.2, refit=F,
  single.step=F, return.paths=F) {

  type <- match.arg(type)

  p <- model$p;  d <- model$d; # model dims

  if(missing(x_train)) {
    x_train = model$data # training sample
  }

  # result series
  x_res <- x_train
  # training sample size
  nt <- length(x_res)
  sd_res <- sqrt(model$resSigmaSq)

  # extract model residuals
  resid <- as.numeric(model$residuals)
  resid <- resid[!is.na(resid)]

  # fill the prediction part of the array with zeros
  x_res <- c(x_res, rep(0, horizon))
  xrange <- p - ((p - 1):0)

  if(type=="naive") Nboot <- 1

  predictions <- function(x_res, eval.model=model, tmin=nt) {
    noise <- switch(
      type,
        "naive"= rep(0, n_ahead),
        "MC"= rnorm(n_ahead, mean = 0, sd=sd_res),
```

```
      "bootstrap" = sample(resid, size=n_ahead, replace=T)
  )

  for(t in (tmin + (1:n_ahead))) {
    x_res[t] <- SETAR_m_singleStep(eval.model, x_res, t)
    if (!single.step) x_res[t] <- x_res[t] + noise[t - tmin]
  }

  if (!single.step) {
    return(x_res)
  }
  return(x_res[tmin + n_ahead])
}

if(single.step) {
  n_ahead <- 1
  x_data <- c(x_train, x_eval)
  if (nt + horizon > length(x_data)) horizon <- (length(x_eval) + 1)

  c <- model$c; m <- model$nReg
  if (refit) {
    fit_model <- EstimSETAR_m_postproc( EstimSETAR_m(x_data, p, d, c, m) )
  } else {
    fit_model <- model
  }

  x_simulations <- matrix(rep(x_data[nt], Nboot), ncol=Nboot)
  for (t in (nt + 1:horizon)) {
    x_source <- x_data[1:(t - 1)]
    x_simulations <- rbind(x_simulations, replicate(Nboot,
          predictions(x_source, eval.model=fit_model, tmin=(t - 1))
    ))
    x_res[t] <- mean(x_simulations[t - nt,])
  }
  x_pred <- x_res[nt + 1:horizon]
} else {
  # === MULTISTEP ===
  if (n_ahead == 1) n_ahead <- horizon

  x_simulations <- replicate(Nboot, predictions(x_res))
  x_sim_means <- rowMeans(x_simulations, na.rm=T)
  x_pred <- x_sim_means[(nt - 1) + 1:n_ahead]
}

# if not naive compute conf. intervals:
x_errors <- x_pred
if(type != "naive") {
  x_errors <- t(apply(
    x_simulations[(nt - 1) * (!single.step) + 1:horizon, ,drop=F], MARGIN=1, quantile,
    prob=sort(c(alpha, 1 - alpha)), na.rm=T))
}

# compute prediction errors
MSE <- sum((x_eval[1:horizon] - x_pred)^2, na.rm=T) / horizon
# RMSE <- sqrt(MSE)

if(type == "naive"){
  result <- list(pred=x_pred, MSE=MSE)
```

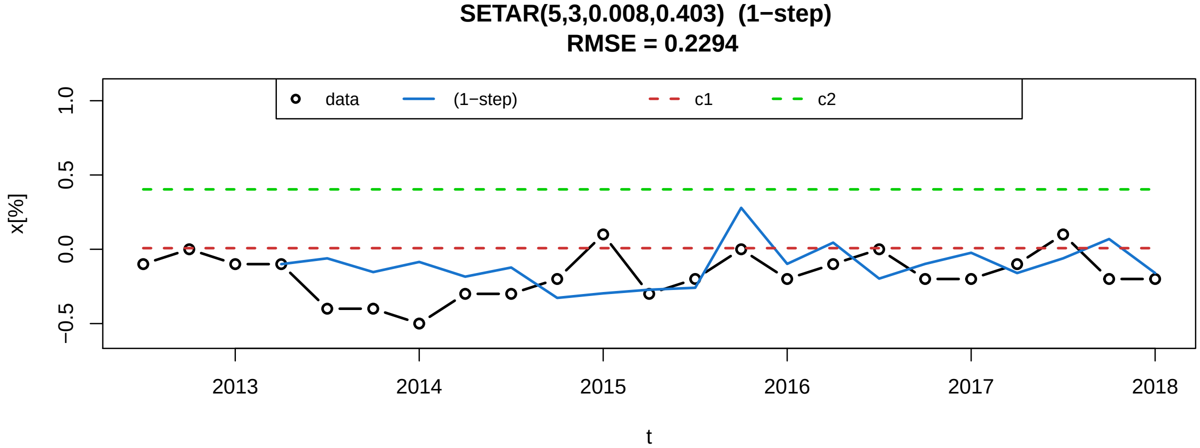
```r
  } else {
    if (return.paths) result <- list(pred=x_pred, se=x_errors, MSE=MSE, alpha=alpha,
                                     paths=x_simulations[(nt - 1) * (!single.step) + 1:horizon,])
    else result <- list(pred=x_pred, se=x_errors, MSE=MSE, alpha=alpha)
  }

  return(result)
}
```

Now we test it on our data:

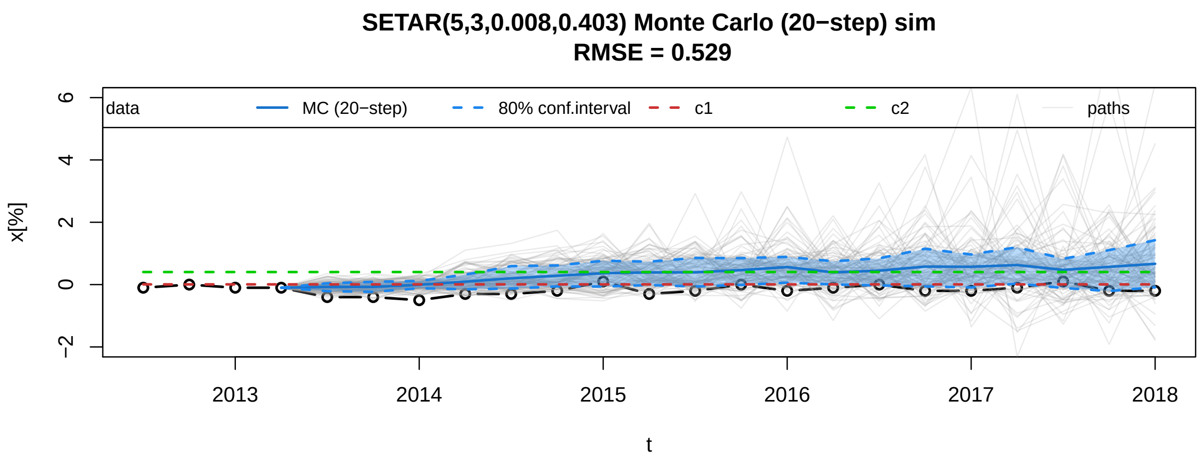
```
## naive (10-step):

##  [1] -0.100 -0.100  0.011  0.011  0.048  0.077  0.049  0.041  0.027  0.008
## [11]  0.008

## Monte Carlo (10-step):

##  [1] -0.100 -0.100 -0.005  0.003  0.093  0.178  0.294  0.409  0.499  0.672
## [11]  0.741

## Bootstrap (10-step):

##  [1] -0.100 -0.100  0.009  0.027  0.119  0.222  0.291  0.424  0.567  0.663
## [11]  0.793

## data:

##  [1] -0.1 -0.4 -0.4 -0.5 -0.3 -0.3 -0.2  0.1 -0.3 -0.2  0.0

## naive (1-step):

##  [1] -0.100 -0.100  0.011 -0.168 -0.132 -0.210 -0.013 -0.024  0.079  0.137
## [11] -0.026

## Monte Carlo (1-step):

##  [1] -0.100 -0.100  0.011 -0.168 -0.132 -0.210 -0.013 -0.024  0.079  0.137
## [11] -0.026

## Bootstrap (1-step):

##  [1] -0.100 -0.100  0.011 -0.168 -0.132 -0.210 -0.013 -0.024  0.079  0.137
## [11] -0.026

## data:

##  [1] -0.1 -0.4 -0.4 -0.5 -0.3 -0.3 -0.2  0.1 -0.3 -0.2  0.0
```

To test other prediction results, such as confidence intervals and simulation paths, we implement a plot procedure:

```r
par(mfrow=c(1, 1))
predictSETAR_andPlot(
  models[[ orders[6] ]], time=dat$time, x_train=x_train, x_eval=x_eval,
  pred_type="MC", single.step=T, plot.paths=F, plot.leg=T)
```

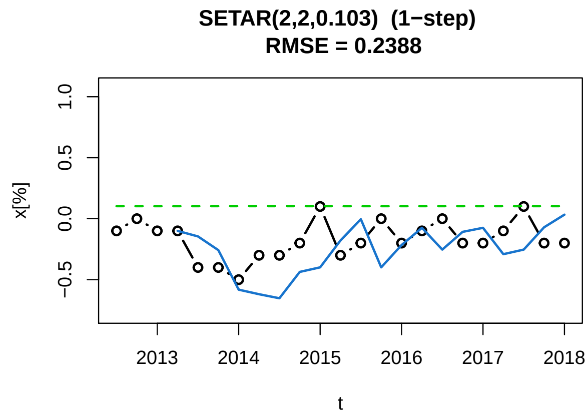


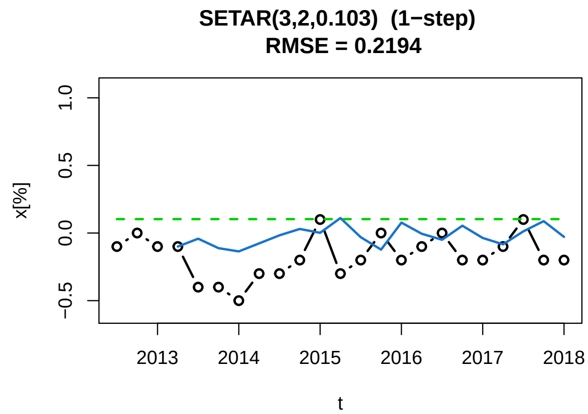

```r
predictSETAR_andPlot(
  models[[ orders[6] ]], time=dat$time, x_train=x_train, x_eval=x_eval,
  pred_type="MC", single.step=F, plot.paths=T, plot.leg=T, plt_range=c(-2,6))
```

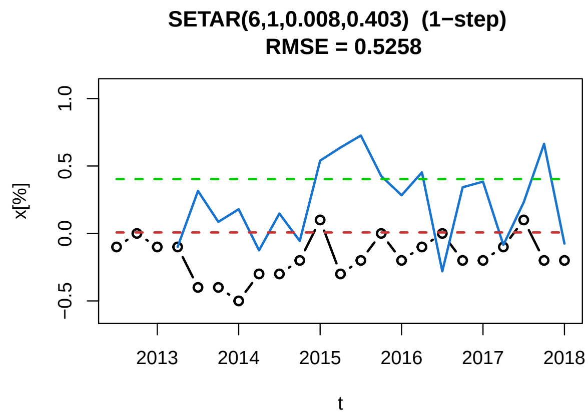


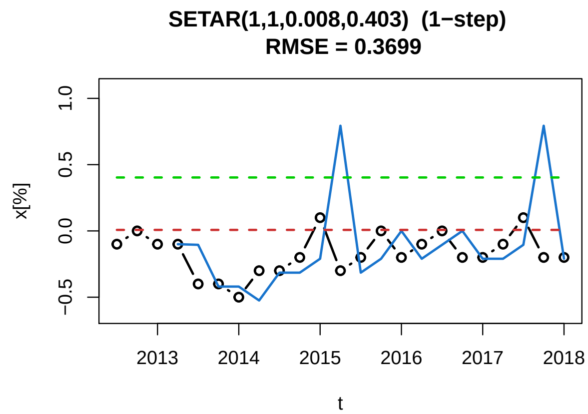
Now we possess all necessary tools to proceed evaluating our SETAR models according to their predictive abilities.

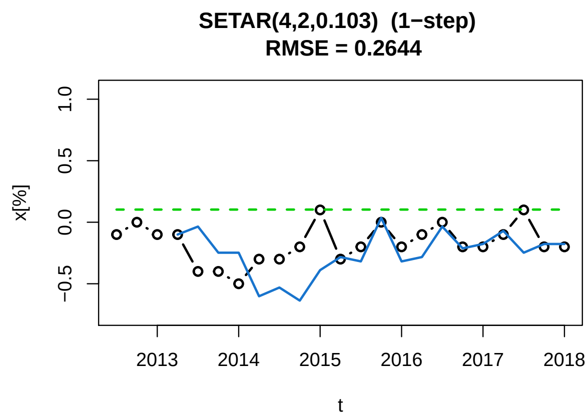
5.2 Single-Step Predictions of SETAR Models

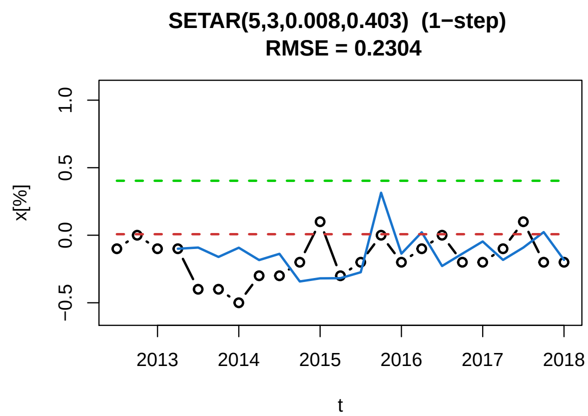


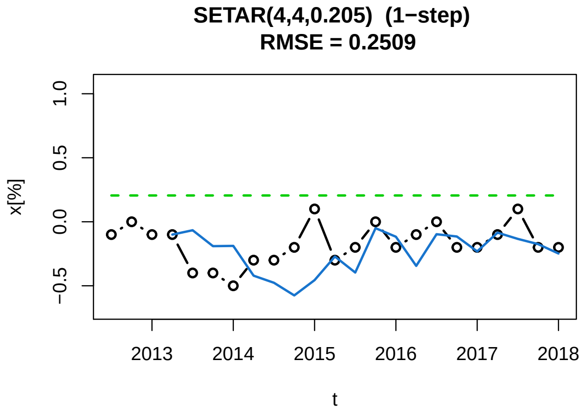


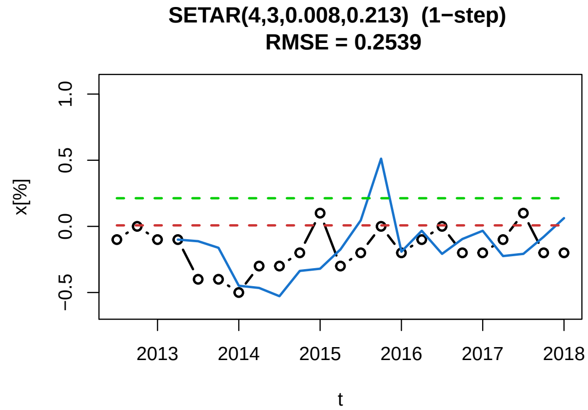


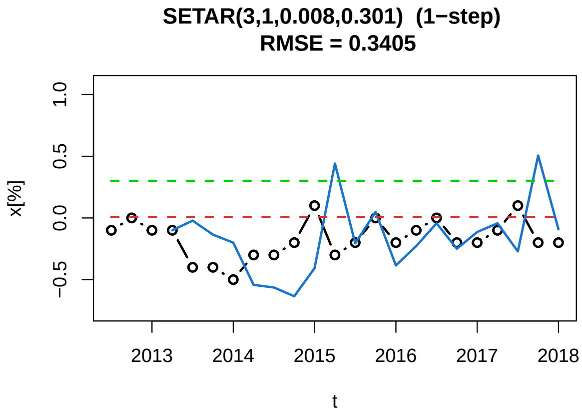


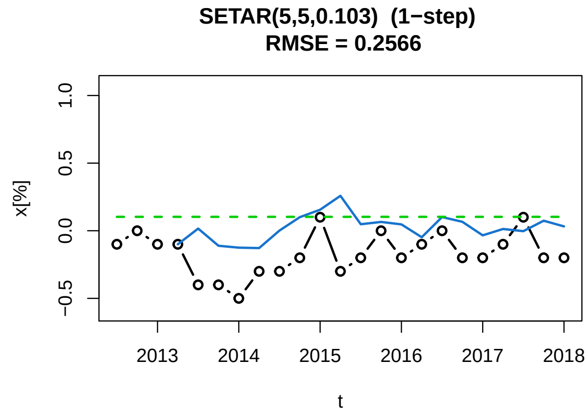


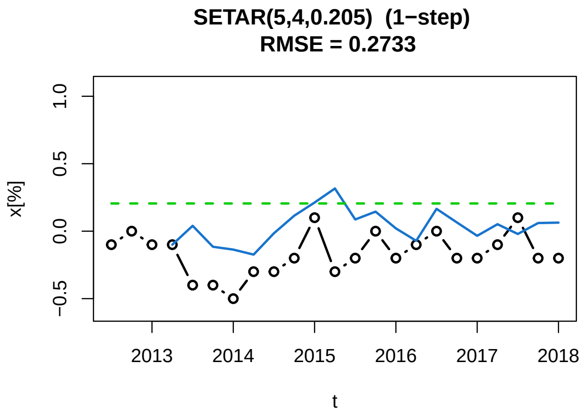


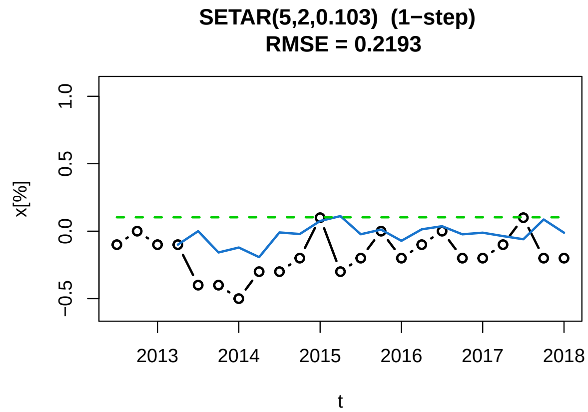






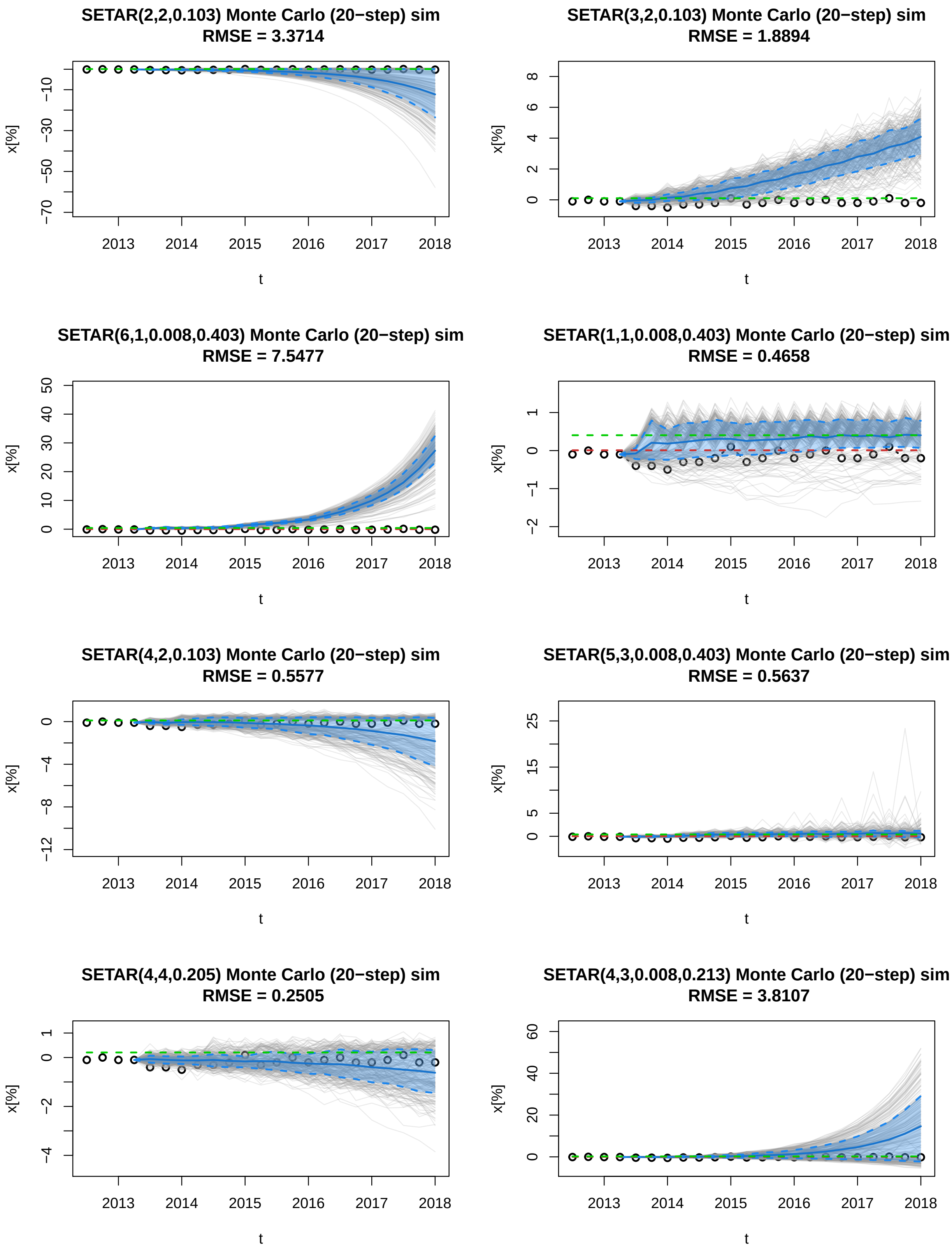




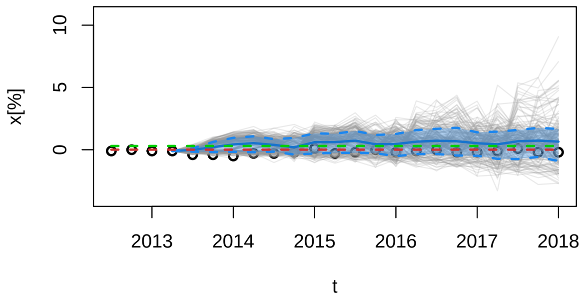


5.3 Multi-Step Predictions of SETAR Models

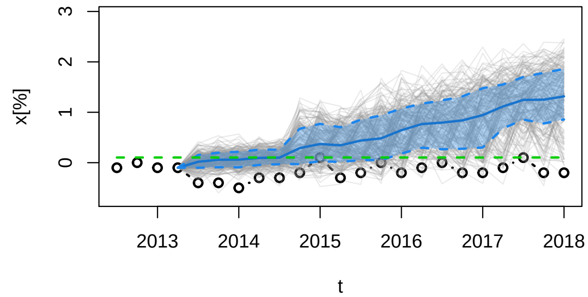
Monte Carlo:


SETAR(2,2,0.103) Monte Carlo (20−step) sim
RMSE = 3.3714
SETAR(3,2,0.103) Monte Carlo (20−step) sim
RMSE = 1.8894
SETAR(6,1,0.008,0.403) Monte Carlo (20−step) sim
RMSE = 7.5477
SETAR(1,1,0.008,0.403) Monte Carlo (20−step) sim
RMSE = 0.4658
SETAR(4,2,0.103) Monte Carlo (20−step) sim
RMSE = 0.5577
SETAR(5,3,0.008,0.403) Monte Carlo (20−step) sim
RMSE = 0.5637
SETAR(4,4,0.205) Monte Carlo (20−step) sim
RMSE = 0.2505
SETAR(4,3,0.008,0.213) Monte Carlo (20−step) sim
RMSE = 3.8107
x[%]
t

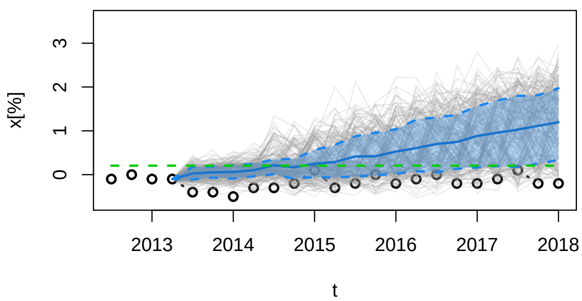
SETAR(3,1,0.008,0.301) Monte Carlo (20−step) sim
RMSE = 0.6797



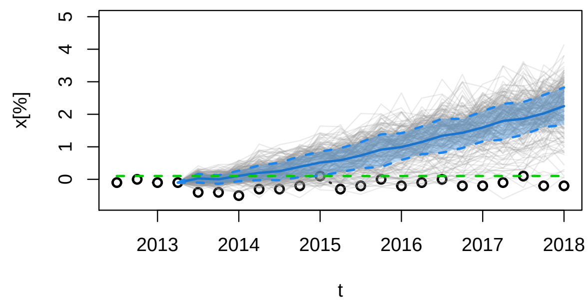
SETAR(5,5,0.103) Monte Carlo (20−step) sim
RMSE = 0.7718



SETAR(5,4,0.205) Monte Carlo (20−step) sim
RMSE = 0.6908

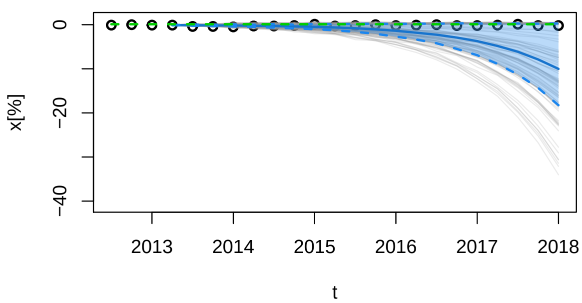


SETAR(5,2,0.103) Monte Carlo (20−step) sim
RMSE = 1.1573

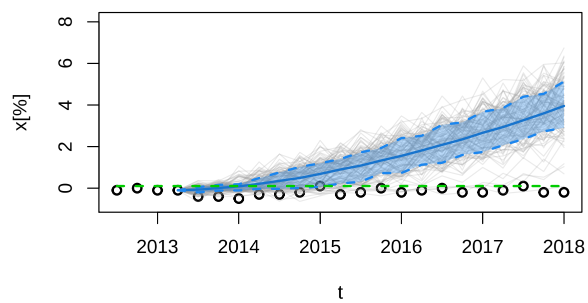


Bootstrap:

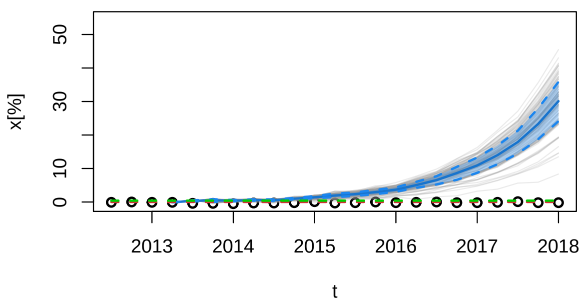
SETAR(2,2,0.103) Bootstrap (20−step) sim
RMSE = 2.7435



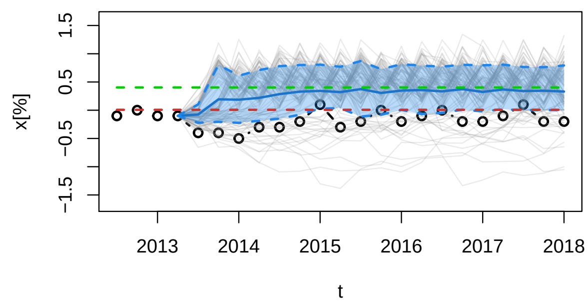
SETAR(3,2,0.103) Bootstrap (20−step) sim
RMSE = 1.8291



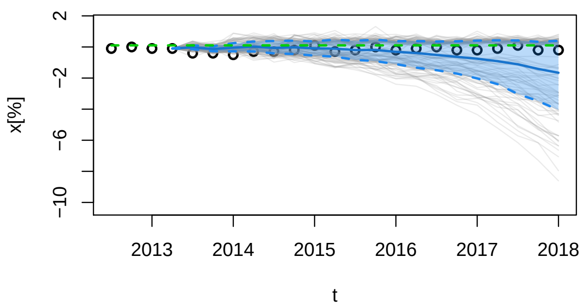
SETAR(6,1,0.008,0.403) Bootstrap (20−step) sim
RMSE = 8.3217



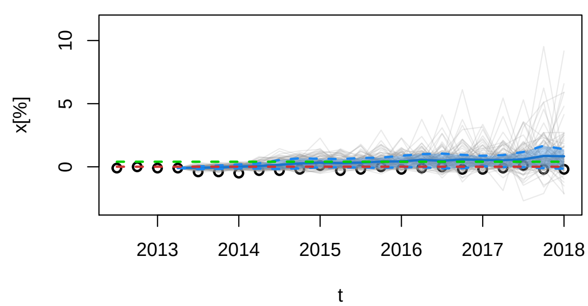
SETAR(1,1,0.008,0.403) Bootstrap (20−step) sim
RMSE = 0.4655



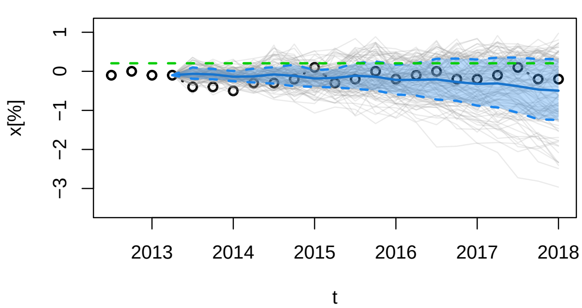
SETAR(4,2,0.103) Bootstrap (20−step) sim
RMSE = 0.4956



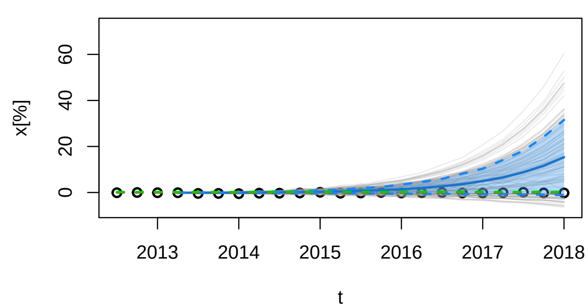
SETAR(5,3,0.008,0.403) Bootstrap (20−step) sim
RMSE = 0.5433



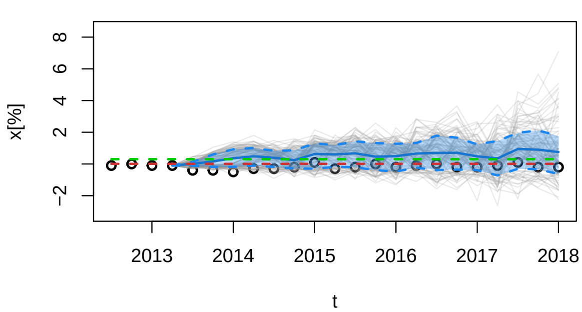
SETAR(4,4,0.205) Bootstrap (20−step) sim
RMSE = 0.214



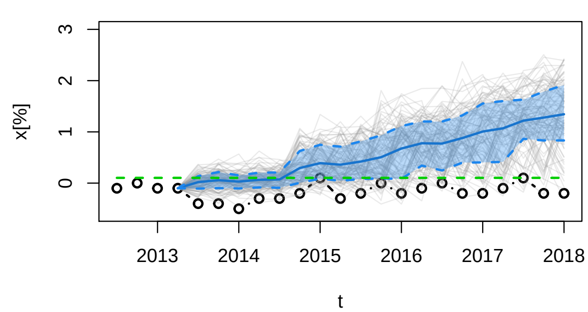
SETAR(4,3,0.008,0.213) Bootstrap (20−step) sim
RMSE = 4.0077

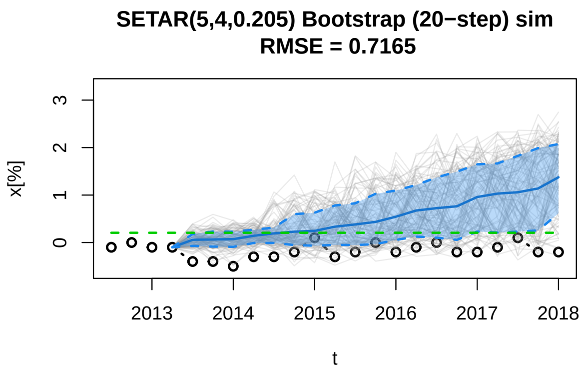


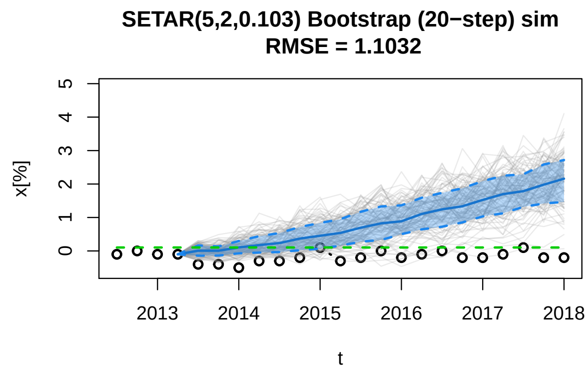
SETAR(3,1,0.008,0.301) Bootstrap (20−step) sim
RMSE = 0.7015



SETAR(5,5,0.103) Bootstrap (20−step) sim
RMSE = 0.7771







5.4 Evaluating Models

From what we observe, some models exhibit explosive properties, and have equilibria significantly distant from the evaluated data set, which, of course, impacts the resulting prediction error. Now we evaluate models according to their predictive properties, and filter out those that produce distant and/or explosive predictions.

```
##                     model      BIC sigmaSq MSE(1-step)   MSE(MC)   MSE(boot)
## 1          SETAR(2,2,0.103) -602.841  0.0262      0.0605   13.1177 ~    6.3739 ~
## 2          SETAR(3,2,0.103) -588.235  0.0267      0.0455    3.4177 ~    3.4439 ~
## 3    SETAR(6,1,0.008,0.403) -584.202  0.0237      0.2804 ~ 58.2155 ~   55.9353 ~
## 4    SETAR(1,1,0.008,0.403) -577.983  0.0293      0.1351 ~  0.2398      0.2153
## 5          SETAR(4,2,0.103) -577.642  0.0268      0.0735    0.2687      0.3725
## 6    SETAR(5,3,0.008,0.403)   -575.9  0.0245      0.0526    0.2758      0.2325
## 7          SETAR(4,4,0.205) -575.688  0.0270      0.0643    0.0662      0.0466
## 8    SETAR(4,3,0.008,0.213) -575.293  0.0261      0.0702   14.1158 ~   13.0770 ~
## 9    SETAR(3,1,0.008,0.301)  -575.07  0.0273      0.1140 ~  0.5061      0.5278
## 10         SETAR(5,5,0.103) -573.138  0.0254      0.0704    0.6513 ~    0.5558
## 11         SETAR(5,4,0.205) -571.548  0.0262      0.0729    0.3923      0.5057
## 12         SETAR(5,2,0.103) -571.394  0.0259      0.0504    1.4456 ~    1.3685 ~
```

We notice that models that quickly diverge have a significantly higher MSE than their $\hat{\sigma}_\varepsilon^2$ (`RSS / (n - k)`). One could then formulate a condition that model is "too divergent" if its training data MSE (`RSS / (n - k)`) is significantly lower than prediction MSE. The significance factor could be taken, for example, as 4, i.e.: if the prediction MSE exceeds 4-multiple of training data `RSS / (n - k)`, the prediction diverges. In the above table, we mark divergent predictions with `"~"`. When examining multi-step predictions we take a factor of 8 of 1-step MSE.

5.5 Conclusion and SETAR Evaluation

After considering all possible configurations of SETAR models, testing them for remaining nonlinearity, and evaluating their predictive properties, we conclude the following:

Some models placed within the top 3 best in their fit onto the training data (according to their BIC), contained SETAR3-type remaining nonlinearity, and thus had their thresholds estimated, turned out to have radically different equilibria than their 2-regime predecessors. Some were even explosive when we examined their predictive properties. Hence their training data fit quality cannot justify their mismatch between data and their properties as stochastic dynamical systems.

With regards to their predictive properties, we pick the best models according to their given MSE depending on the prediction method (`"naive"`, `"mc"`, `"boot"`):

```
## Models sorted by 1-step naive MSE:
```

```
##                     model      BIC sigmaSq MSE(1-step)
## 2          SETAR(3,2,0.103) -588.235  0.0267      0.0455
## 12         SETAR(5,2,0.103) -571.394  0.0259      0.0504
```

```
## 6  SETAR(5,3,0.008,0.403)   -575.9  0.0245      0.0526
## 1        SETAR(2,2,0.103) -602.841  0.0262      0.0605
## 7        SETAR(4,4,0.205) -575.688  0.0270      0.0643
## 8  SETAR(4,3,0.008,0.213) -575.293  0.0261      0.0702
## 10       SETAR(5,5,0.103) -573.138  0.0254      0.0704
## 11       SETAR(5,4,0.205) -571.548  0.0262      0.0729
## 5        SETAR(4,2,0.103) -577.642  0.0268      0.0735
```

```
## Models sorted by Monte Carlo MSE:
```

```
##                     model      BIC sigmaSq MSE_mc
## 7        SETAR(4,4,0.205) -575.688  0.0270 0.0662
## 4  SETAR(1,1,0.008,0.403) -577.983  0.0293 0.2398
## 5        SETAR(4,2,0.103) -577.642  0.0268 0.2687
## 6  SETAR(5,3,0.008,0.403)   -575.9  0.0245 0.2758
## 11       SETAR(5,4,0.205) -571.548  0.0262 0.3923
## 9  SETAR(3,1,0.008,0.301)  -575.07  0.0273 0.5061
```

```
## Models sorted by Bootstrap MSE:
```

```
##                     model      BIC sigmaSq MSE_boot
## 7        SETAR(4,4,0.205) -575.688  0.0270   0.0466
## 4  SETAR(1,1,0.008,0.403) -577.983  0.0293   0.2153
## 5        SETAR(4,2,0.103) -577.642  0.0268   0.3725
## 6  SETAR(5,3,0.008,0.403)   -575.9  0.0245   0.2325
## 10       SETAR(5,5,0.103) -573.138  0.0254   0.5558
## 9  SETAR(3,1,0.008,0.301)  -575.07  0.0273   0.5278
```

and as we observe, the order of the first 4 models remains the same regardless of the prediction method.

6. STAR Model Estimation

While SETAR-type models are governed by a discontinuous transition function (`Indicator`), we have not yet tried a continuous alternative of transitioning between regimes. STAR (Smooth Threshold AutoRegressive) models utilize the possibility of a smooth regime transition. As will the context of the following sections suggest, we can choose a particular transition function between exponential (ESTAR) and logistic (LSTAR) and a scalar parameter γ which describes the "smoothness" of a regime transition. For $\gamma \to \infty$ we obtain a discontinuous step-like transition function, as in SETAR's indicator.

6.1 Tests For STAR-type nonlinearity

To test for STAR-type nonlinearity we use a specialized form of an LM test.

```r
LMtest_LINvsSTAR(xt, p=3, d=1, alternative = "LSTAR")
```

```
##          LM    crit_val     p_value
## 15.63293562 16.91897760  0.07495392
```

```r
LMtest_LINvsSTAR(xt, p=3, d=1, alternative = "ESTAR")
```

```
##          LM    crit_val     p_value
## 14.01087154 12.59158724  0.02951496
```

To decide whether an exponential or a logistic transition function provides a better fit, we test for the last parameter vector of a regression test from `LMtest_LINvsSTAR` is zero. By accepting the hypothesis, we conclude that exponential transition fits our data better:

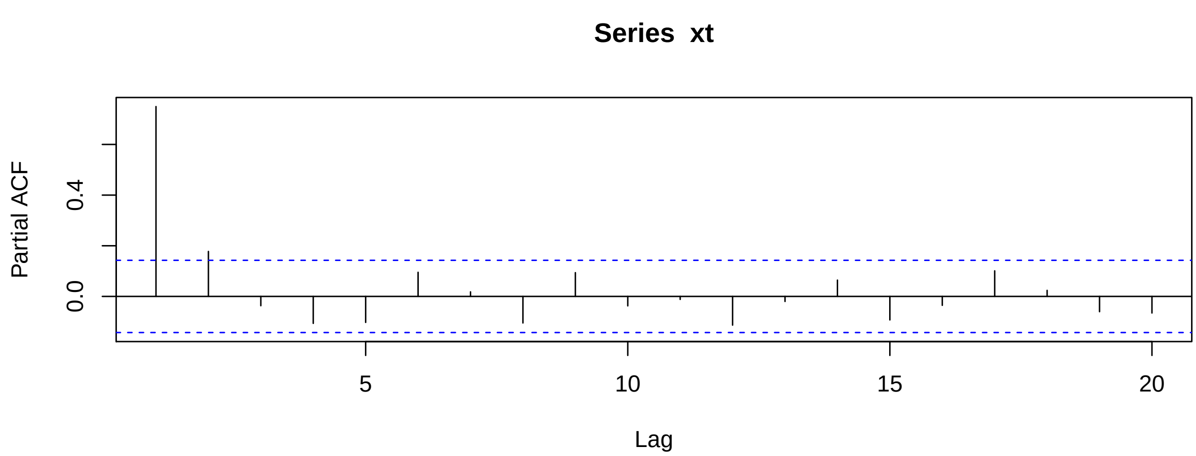
```r
LMtest_EvsL <- function(x, p, d, alpha=0.05) {
  x <- as.ts(x)  # if x is not a ts object
  tmp <- c(  # list of delayed ts
    list(x),
```

```r
    lapply(1:p, function(i) stats::lag(x, -i))
  )
  tmp <- do.call(function(...) ts.intersect(..., dframe = T), tmp)
  names(tmp) <- c("x", paste0("x", 1:p))
  y <- lm(x ~ ., data = tmp)$residuals
  y <- c(rep(NA, p), y)
  attributes(y) <- attributes(x)  # make y the same ts object as x
  tmp <- c(
    list(y),
    lapply(1:p, function(i) stats::lag(x, -i)),  # lag shifts forward, therefore the minus operator
    lapply(1:p, function(i) stats::lag(x, -i) * stats::lag(x, -d)),
    lapply(1:p, function(i) stats::lag(x, -i) * stats::lag(x, -d)^2)
  )

  # === ESTAR regression ========
  tmp <- do.call(function(...) ts.intersect(..., dframe = T), tmp)
  names(tmp)[1] <- "y"
  res_E <- lm(y ~ ., data = tmp)$residuals

  # === LSTAR regression ========
  tmp <- c(tmp, lapply(1:p, function(i) stats::lag(x, -i) * stats::lag(x, -d)^3) )
  tmp <- do.call(function(...) ts.intersect(..., dframe = T), tmp)
  names(tmp)[1] <- "y"
  res_L <- lm(y ~ ., data = tmp)$residuals

  LM <- length(x) * (1 - sum(res_L^2) / sum(res_E^2))
  c(LM = LM, crit_val = qchisq(1 - alpha, df = p), p_value = 1 - pchisq(LM, df = p), alpha_corrected = alpha / p)
}

LMtest_EvsL(xt, p=3, d=1)
```

```
##              LM        crit_val         p_value alpha_corrected
##      1.75193804      7.81472790      0.62544917      0.01666667
```

The statistic of an LM test is assumed to be of $\chi^2(p)$ distribution. This means that the p-value of the test has to be compared with a significance factor normalized by the parameter vector's dimension, i.e.: p-val $< \alpha/p$ (Bonferoni correction). Hence the `alpha_corrected` value.

Naturally, we need to test for STAR-type non-linearity for an array of discrete parameters p and d. The delay parameter d is, of course, bounded by the lag parameter p, the upper bound of which can be assumed from the partial autocorrelation of the series:



Since partial autocorrelation dies out after 2 lag steps, we should limit the sample space by $p_{max} = 2$, however to show a reasonably large sample, we follow an assumption that $p_{max} = 12$:

```
## LSTAR non-linearity test results (linearity rejected):
```

```
##    p d      p_value        crit
## 3  2 2 0.0063931099 0.008333333
## 8  4 2 0.0015530134 0.004166667
## 12 5 2 0.0025031538 0.003333333
## 22 7 1 0.0018492820 0.002380952
## 23 7 2 0.0004380275 0.002380952
## 30 8 2 0.0010607821 0.002083333
## 38 9 2 0.0010282912 0.001851852
```

```
## ESTAR non-linearity test results (linearity rejected):
```

```
##   p d     p_value        crit
## 3 2 2 0.003337111 0.008333333
```

Tests show a clear LSTAR preference, with an overlap with ESTAR nonlinearity for $p = 2$.

```
## $LSTARs
##    p d
## 3  2 2
## 8  4 2
## 12 5 2
## 22 7 1
## 23 7 2
## 30 8 2
## 38 9 2
##
## $ESTARs
##   p d
## 3 2 2
```

The parameter intersection between the above samples should also be examined for preference, using our ESTAR vs LSTAR test:

```
##      p d     LM crit_val p_value alpha_corrected.p
## [1,] 2 2 2.3587   5.9915  0.3075             0.025
```

```
##
##  ESTAR preference:
```

```
## [1] NA
```

Which suggests that we ought to consider only LSTAR models.

6.2 Estimating STAR Model Parameters

First, we implement and test all necessary functions (for m regimes).

Regime Transition

```
SmoothTransition <- function(x, c, gamma, type=c("logistic","exponential")) {
  switch(type,
         logistic = {1 / (1 + exp(-gamma * (x - c)))},
         exponential = {1 - exp(-gamma * (x - c)^2)}
  )
}

SmoothTransition(0.3, c=0.2, gamma=10, type="logistic")
```

```
## [1] 0.7310586
```

```r
SmoothTransition(0.3, c=c(-0.1, 0.2), gamma=c(1, 5), type="logistic")
```

```
## [1] 0.5986877 0.6224593
```

Single Regime Basis

```r
Yt <- function(x, t, p) c(1, x[(t-1):(t-p)])

Yt(xt, t=3, p=2)
```

```
## [1] 1.0 0.1 0.3
```

m-Regime Basis

```r
Xt_m <- function(x, t, p, d, c, gamma, m=2, z = x, type=c("logistic", "exponential")) {
  type <- match.arg(type)
  I <- SmoothTransition(z[t - d], c, gamma, type)
  Y <- Yt(x, t, p)
  as.numeric(sapply(1:m, function(j) (ifelse(j == 1, 1, I[j - 1]) - ifelse(j == m, 0, I[j])) * Y))
}

Xt_m(xt, t=3, p=2, d=1, c=0.2, gamma=10)
```

```
## [1] 0.73105858 0.07310586 0.21931757 0.26894142 0.02689414 0.08068243
```

```r
Xt_m(xt, t=3, p=2, d=1, c=c(-0.1, 0.2), gamma=c(1, 5), m=3)
```

```
## [1] 0.45016600 0.04501660 0.13504980 0.17229333 0.01722933 0.05168800 0.37754067
## [8] 0.03775407 0.11326220
```

Skeleton

```r
skelSTAR_m <- function(x, t, p, d, c, gamma, PhiParams, m=2, z = x, type = c("logistic", "exponential")) {
  c(PhiParams %*% Xt_m(x, t, p, d, c, gamma, m, z, type))
}

skelSTAR_m(xt, t=10, p=2, d=1, c=0.2, gamma=1, PhiParams=rep(1, 6))
```

```
## [1] 0.5
```

```r
skelSTAR_m(xt, t=10, p=2, d=1, c=c(-0.1, 0.2), gamma=c(1, 5), PhiParams=rep(1, 9), m=3)
```

```
## [1] 0.5
```

Information Criteria

```r
IC <- function(n, p, sigmaSq, m=2) {
  lik <- -n * log(2 * pi / sigmaSq) / 2 - (n - p) / 2
  npar <- m * (p + 1) + 2 * (m - 1)
  c(AIC= -2 * lik + 2 * npar, BIC= - 2 * lik + log(n) * npar)
}

IC(n=length(xt), p=2, sigmaSq=0.027)
```

```
##      AIC      BIC
## 1233.011 1258.945
```

```r
IC(n=length(xt), p=2, sigmaSq=0.027, m=3)
```

```
##      AIC      BIC
## 1243.011 1285.154
```

Parameter Estimation

The initial estimation of a STAR model candidate follows the same pattern as estimation methods in sections 2.2 and 4.3:

```r
EstimSTAR_m <- function(x, p, d, c, gamma, m=2, type=c("logistic", "exponential")) {
  type <- match.arg(type)
  m = as.integer(m)
  if (m <= 0) {
    message("Error: regime count m has to be a positive integer")
    return(NA)
  }
  if (length(c) != m - 1) {
    message("Error: Incompatible dimensions of threshold vector and regime count.");
    return(NA)
  }

  resultModel <- list()
  cPrint <- round(c, digits=3); gammaPrint <- round(gamma, digits=3) # 3 dec. places seems enough
  resultModel$name <- paste0(ifelse(type == "logistic", "L", "E"),
              "STAR(", p, ",", d, ",",
              paste(na.omit(cPrint), collapse=','),",",
              paste(na.omit(gammaPrint), collapse=','), ")")
  resultModel$nReg <- m; resultModel$type <- type
  resultModel$p = p; resultModel$d = d;
  resultModel$c = c; resultModel$gamma = gamma;
  resultModel$data = x; n = length(x); resultModel$n = n;

  X <- as.matrix(apply(as.matrix((p + 1):n), MARGIN=1, function(t) Xt_m(x, t, p, d, c, gamma, m) ))
  y <- as.matrix(x[(p + 1):n])
  K <- crossprod(t(X), t(X)); b <- crossprod(t(X), y);
  detK <- abs(det(K))

  if (detK < 0.000001) {
    return(NA)
  } else {
    K <- inv(K)
    sol_phi <- as.numeric(t(K %*% b)) # solving (X'X)*phi = X'y
    # resultModel$PhiStErrors <- solution[,2] # standard errors
    skel <- crossprod(X, sol_phi); resultModel$skel <- skel;
    resultModel$residuals <- (y - skel)
    resultModel$resSigmaSq <- 1 / (n - p) * sum(resultModel$residuals^2)
    resultModel$PhiParams <- sol_phi

    return(resultModel)
  }
}

str( EstimSTAR_m(xt, p=2, d=2, c=0, gamma=10) ) # 2 regimes
```

```
## List of 13
##  $ name      : chr "LSTAR(2,2,0,10)"
##  $ nReg      : int 2
##  $ type      : chr "logistic"
##  $ p         : num 2
##  $ d         : num 2
```

```
## $ c : num 0
## $ gamma : num 10
## $ data : num [1:189] 0.3 0.1 0.2 0.1 -0.1 ...
## $ n : int 189
## $ skel : num [1:187, 1] 0.0947 0.1933 0.118 -0.0343 -0.0808 ...
## $ residuals : num [1:187, 1] 0.1053 -0.0933 -0.218 -0.0657 -0.0192 ...
## $ resSigmaSq: num 0.027
## $ PhiParams : num [1:6] -0.0684 0.2047 0.3672 0.1028 0.9621 ...
```

```
str( EstimSTAR_m(xt, p=2, d=2, c=c(0, 0.2), gamma=c(10, 12), m=3) ) # 3 regimes
```

```
## List of 13
## $ name : chr "LSTAR(2,2,0,0.2,10,12)"
## $ nReg : int 3
## $ type : chr "logistic"
## $ p : num 2
## $ d : num 2
## $ c : num [1:2] 0 0.2
## $ gamma : num [1:2] 10 12
## $ data : num [1:189] 0.3 0.1 0.2 0.1 -0.1 ...
## $ n : int 189
## $ skel : num [1:187, 1] 0.1003 0.1632 0.1187 -0.0264 -0.0782 ...
## $ residuals : num [1:187, 1] 0.0997 -0.0632 -0.2187 -0.0736 -0.0218 ...
## $ resSigmaSq: num 0.0267
## $ PhiParams : num [1:9] -0.0347 0.3063 0.3804 0.0273 0.5822 ...
```

Estimating Standard Errors of STAR Parameters

Notice, that we can no longer find a consitent estimate of parameter variances from the AR covariance matrix alone. Individual regimes are interdependent thanks to smooth transition functions. We will need to find an estimate of a covariance matrix $\mathbf{\Sigma}_{\hat{\theta}}$ of all parameters $\theta = (\phi_1, \phi_2, ..., \phi_m, \gamma_1, ..., \gamma_{m-1}, c_1, ..., c_{m-1})^\top$ (where m is the number of regimes). Under our given conditions, the least square estimate $\hat{\theta}$ of θ is asymptotically normally distributed, i.e.: $\sqrt{n}(\hat{\theta}-\theta) \sim N(\mathbf{0}, \mathbf{\Sigma}_{\hat{\theta}})$ for $n \to \infty$. Where a consitent estimate of the covariance matrix is given by $\hat{\mathbf{\Sigma}}_{\hat{\theta}} = \frac{1}{n}\hat{\mathbf{H}}_n^{-1}\hat{\mathbf{M}}_n\hat{\mathbf{H}}_n^{-1}$, where $\hat{\mathbf{H}}_n = \frac{1}{n-p}\sum_{t=p+1}^{n} \nabla^2 r_t(\hat{\theta})$ is the mean Hessian, $\hat{\mathbf{M}}_n = \frac{1}{n-p}\sum_{t=p+1}^{n} \nabla r_t(\hat{\theta})\nabla r_t(\hat{\theta})^\top$ the mean information matrix, and $r_t(\theta) = \big(x_t - F(x_{t-d}, \theta)\big)^2$ (F is the model skeleton).

Given the dimensions of our model (m - number of regimes, p - AR parameter lag), we obtain symmetrical square matrices of dimension $\dim\theta \times \dim\theta$, where $\dim\theta = m(p+1) + 2(m-1)$. Both $\big(r_t(\hat{\theta})\nabla r_t(\hat{\theta})^\top\big)_{\dim\theta\times\dim\theta}$ and $\big(\nabla^2 r_t(\hat{\theta})\big)_{\dim\theta\times\dim\theta}$ have to be computed for each $t = p+1, ..., n$. Differentiating r_t with respect to its parameters we get $\frac{\partial r_t}{\partial\theta_i} = 2(F_t - x_t)\frac{\partial F_t}{\partial\theta_i} = (\nabla r_t)_i$ and $\frac{\partial^2 r_t}{\partial\theta_i\partial\theta_j} = 2\left(\frac{\partial F_t}{\partial\theta_i}\frac{\partial F_t}{\partial\theta_j} + (F_t - x_t)\frac{\partial^2 F_t}{\partial\theta_i\partial\theta_j}\right) = (\nabla^2 r_t)_{i,j}$. The respective derivatives of skeleton F_t are elements of a gradient vector $(\nabla F_t)_{\dim\theta}$ and Hessian $(\nabla^2 F_t)_{\dim\theta\times\dim\theta}$. It suffices to compute the elements of ∇F_t and $\nabla^2 F_t$ to estimate the mean residual square (**rs**) Hessian and information matrix, and use the above formulas.

The skeleton gradient and Hessian can be divided into sub-matrices according to the differentiated variable:

$$\nabla F_t = \begin{pmatrix} \nabla_{\boldsymbol{\phi}} F_t \\ \nabla_{\boldsymbol{\gamma}} F_t \\ \nabla_{\boldsymbol{c}} F_t \end{pmatrix} \quad , \quad \boldsymbol{\phi} = (\phi_1, ..., \phi_m)^\top \ , \ \boldsymbol{\gamma} = (\gamma_1, ..., \gamma_{m-1})^\top \ , \ \boldsymbol{c} = (c_1, ..., c_{m-1})^\top$$

$$\nabla^2 F_t = \begin{pmatrix} \nabla^2_{\boldsymbol{\phi},\boldsymbol{\phi}} F_t & \nabla^2_{\boldsymbol{\phi},\boldsymbol{\gamma}} F_t & \nabla^2_{\boldsymbol{\phi},\boldsymbol{c}} F_t \\ (\nabla^2_{\boldsymbol{\phi},\boldsymbol{\gamma}} F_t)^\top & \nabla^2_{\boldsymbol{\gamma},\boldsymbol{\gamma}} F_t & \nabla^2_{\boldsymbol{\gamma},\boldsymbol{c}} F_t \\ (\nabla^2_{\boldsymbol{\phi},\boldsymbol{c}} F_t)^\top & (\nabla^2_{\boldsymbol{\gamma},\boldsymbol{c}} F_t)^\top & \nabla^2_{\boldsymbol{c},\boldsymbol{c}} F_t \end{pmatrix}$$

Now take $\boldsymbol{\phi}^{(+)} = \boldsymbol{\phi}$ and $\boldsymbol{\phi}^{(-)} = (\mathbf{0}_{(p+1)\times 1}, \phi_1, ..., \phi_{m-1})^\top$, and also create a transition vector $\boldsymbol{G}_t = (1, G(x_{t-d}, \gamma_1, c_1), ..., G(x_{t-d}, \gamma_{m-1}, c_{m-1}))^\top$. Then the skeleton can be expressed as:

$$F_t = \boldsymbol{G}_t^\top (\boldsymbol{\phi}^{(+)} - \boldsymbol{\phi}^{(-)}) \boldsymbol{Y}_t$$

We notice that since parameters $\phi_{j,i}$ are linear, we get $\nabla^2_{\boldsymbol{\phi},\boldsymbol{\phi}} F_t = \mathbf{0}_{m(p+1)\times m(p+1)}$. Similarly, we can then fill in the remaining derivatives:

$$\nabla_{\boldsymbol{\phi}} F_t = \begin{pmatrix} (1 - G(x_{t-d}, \gamma_1, c_1)) \boldsymbol{Y}_t \\ (G(x_{t-d}, \gamma_1, c_1) - G(x_{t-d}, \gamma_2, c_2)) \boldsymbol{Y}_t \\ \vdots \\ (G(x_{t-d}, \gamma_{m-1}, c_{m-1}) - 0) \boldsymbol{Y}_t \end{pmatrix}, \quad \nabla_{\boldsymbol{\gamma}} F_t = \begin{pmatrix} \frac{\partial G}{\partial \gamma_1}(x_{t-d}, \gamma_1, c_1)(\phi_2 - \phi_1)^\top \boldsymbol{Y}_t \\ \frac{\partial G}{\partial \gamma_2}(x_{t-d}, \gamma_2, c_2)(\phi_3 - \phi_2)^\top \boldsymbol{Y}_t \\ \vdots \\ \frac{\partial G}{\partial \gamma_{m-1}}(x_{t-d}, \gamma_{m-1}, c_{m-1})(\phi_m - \phi_{m-1})^\top \boldsymbol{Y}_t \end{pmatrix}$$

$$\nabla_{\boldsymbol{c}} F_t = \begin{pmatrix} \vdots \\ \text{analogously to } \nabla_{\boldsymbol{\gamma}} \\ \vdots \end{pmatrix}$$

$$\nabla^2_{\boldsymbol{\phi},\boldsymbol{\gamma}} F_t = \begin{pmatrix} -\frac{\partial (\boldsymbol{G}_t)_2}{\partial \gamma_1} \boldsymbol{Y}_t & \mathbf{0}_{(p+1)\times 1} & \cdots & \mathbf{0}_{(p+1)\times 1} \\ \frac{\partial (\boldsymbol{G}_t)_2}{\partial \gamma_1} \boldsymbol{Y}_t & -\frac{\partial (\boldsymbol{G}_t)_3}{\partial \gamma_2} \boldsymbol{Y}_t & \cdots & \mathbf{0}_{(p+1)\times 1} \\ \mathbf{0}_{(p+1)\times 1} & \frac{\partial (\boldsymbol{G}_t)_3}{\partial \gamma_2} \boldsymbol{Y}_t & \cdots & \mathbf{0}_{(p+1)\times 1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{(p+1)\times 1} & \mathbf{0}_{(p+1)\times 1} & \cdots & -\frac{\partial (\boldsymbol{G}_t)_m}{\partial \gamma_{m-1}} \boldsymbol{Y}_t \\ \mathbf{0}_{(p+1)\times 1} & \mathbf{0}_{(p+1)\times 1} & \cdots & \frac{\partial (\boldsymbol{G}_t)_m}{\partial \gamma_{m-1}} \boldsymbol{Y}_t \end{pmatrix}, \quad \nabla^2_{\boldsymbol{\phi},\boldsymbol{c}} F_t = \begin{pmatrix} \vdots \\ \ldots \text{ analogously to } \nabla^2_{\boldsymbol{\phi},\boldsymbol{\gamma}} \ldots \\ \vdots \end{pmatrix}$$

$$\nabla^2_{\boldsymbol{\gamma},\boldsymbol{\gamma}} F_t = \operatorname{diag}\left(\frac{\partial G}{\partial \gamma_1}(x_{t-d}, \gamma_1, c_1)(\phi_2 - \phi_1)^\top \boldsymbol{Y}_t, \ldots, \frac{\partial G}{\partial \gamma_{m-1}}(x_{t-d}, \gamma_{m-1}, c_{m-1})(\phi_m - \phi_{m-1})^\top \boldsymbol{Y}_t \right)$$

$$\nabla^2_{\boldsymbol{c},\boldsymbol{c}} F_t \quad \text{and} \quad \nabla^2_{\boldsymbol{\gamma},\boldsymbol{c}} F_t = \operatorname{diag}\big(\ldots \text{ analogously to } \nabla^2_{\boldsymbol{\gamma},\boldsymbol{\gamma}} \ldots \big)$$

Note that all matrices marked with "analogously to..." contain their respective derivatives of the transition function G. These will be filled in according to its respectice type (logistic, exponential):

$$G_E(x_{t-d}, \gamma, c) = 1 - \mathrm{e}^{-\gamma(x_{t-d}-c)^2}$$

$$\frac{\partial G_E}{\partial \gamma} = \mathrm{e}^{-\gamma(c - x_{t-d})^2}(c - x_{t-d})^2$$

$$\frac{\partial G_E}{\partial c} = 2\mathrm{e}^{-\gamma(c - x_{t-d})^2}\gamma(c - x_{t-d})$$

$$\frac{\partial^2 G_E}{\partial \gamma \partial c} = -2\mathrm{e}^{-\gamma(c - x_{t-d})^2}(c - x_{t-d})(c^2\gamma - 2cx_{t-d} + \gamma x_{t-d}^2\gamma - 1)$$

$$\frac{\partial^2 G_E}{\partial c^2} = -2\mathrm{e}^{-\gamma(c - x_{t-d})^2}\gamma(2c^2\gamma - 4cx_{t-d}\gamma + 2x_{t-d}^2\gamma - 1)$$

$$\frac{\partial^2 G_E}{\partial \gamma^2} = -\mathrm{e}^{-\gamma(c - x_{t-d})^2}(c - x_{t-d})^4$$

$$G_L(x_{t-d}, \gamma, c) = \frac{1}{1+\mathrm{e}^{-\gamma(x_{t-d}-c)}}$$

$$\frac{\partial G_L}{\partial \gamma} = \frac{\mathrm{e}^{-\gamma(x_{t-d}-c)}(x_{t-d}-c)}{\left(1+\mathrm{e}^{-\gamma(x_{t-d}-c)}\right)^2}$$

$$\frac{\partial G_L}{\partial c} = -\frac{\gamma \mathrm{e}^{-\gamma(x_{t-d}-c)}}{\left(\mathrm{e}^{c\gamma}+\mathrm{e}^{\gamma x_{t-d}}\right)^2}$$

$$\frac{\partial^2 G_L}{\partial \gamma \partial c} = -\frac{\mathrm{e}^{\gamma(x_{t-d}-c)}(1-c\gamma+x_{t-d}\gamma+\mathrm{e}^{\gamma(x_{t-d}-c)}(1+c\gamma-x_{t-d}\gamma))}{(1+\mathrm{e}^{\gamma(x_{t-d}-c)})^3}$$

$$\frac{\partial^2 G_L}{\partial c^2} = \frac{\mathrm{e}^{\gamma(x_{t-d}+c)}\gamma^2(\mathrm{e}^{c\gamma}-\mathrm{e}^{x_{t-d}\gamma})}{(\mathrm{e}^{c\gamma}+\mathrm{e}^{x_{t-d}\gamma})^3}$$

$$\frac{\mathrm{e}^{\gamma(c+x_{t-d})}(c-x_{t-d})^2(\mathrm{e}^{c\gamma}-\mathrm{e}^{x_{t-d}\gamma}))}{(\mathrm{e}^{c\gamma}+\mathrm{e}^{x_{t-d}\gamma}))^3}$$

Then we use the above results in the following implementation:

```r
# ==== estimate standard errors of STAR parms: (Phi, gamma, c) ====
starParamCovMatrix <- function(x, p, d, res, Phi, gamma, c, z=x, m=2, type=c("logistic", "exponential")) {
  # implementing this was tedious, and I can only test the symmetry of the result matrix
  type <- match.arg(type)
  n <- length(x); k <- p; times <- (k + 1):n;
```

```r
Phi <- matrix(Phi, nrow=m, byrow=T)
# transition derivatives
# === the following arrays are (m - 1)x(n - k) matrices ====
G <- sapply(times, function(t) SmoothTransition(z[t - d], c, gamma, type))
if (type == "exponential") {
  dG_dgamma <- sapply(times, function(t) exp(-gamma * (c - z[t - d])^2) * (c - z[t - d])^2)
} else {
  dG_dgamma <- sapply(times, function(t) exp(gamma * (z[t - d] - c)) * (z[t - d] - c) /
                      (1 + exp(gamma * (z[t - d] - c)))^2)
}
if (type == "exponential") {
  dG_dc <- sapply(times, function(t) 2 * gamma * (c - z[t - d]) * exp(-gamma * (c - z[t - d])^2))
} else {
  dG_dc <- sapply(times, function(t) -gamma * exp(gamma * (z[t - d] + c)) /
                  (exp(c * gamma) + exp(gamma * z[t - d]))^2)
}
if (type == "exponential") {
  d2G_dgamma2 <- sapply(times, function(t) -exp(-gamma * (c - z[t - d])^2) * (c - z[t - d])^4)
} else {
  d2G_dgamma2 <- sapply(times, function(t) exp(gamma * (c + z[t - d])) * (c - z[t - d])^2 *
            (exp(c * gamma) - exp(z[t - d] * gamma)) / (exp(c * gamma) + exp(z[t - d] * gamma))^3)
}
if (type == "exponential") {
  d2G_dc2 <- sapply(times, function(t) -2 * gamma * exp(-gamma * (c - z[t - d])^2) *
            (-1 + 2 * c^2 * gamma - 4 * c * z[t - d] * gamma + 2 * z[t - d]^2 * gamma))
} else {
  d2G_dc2 <- sapply(times, function(t) gamma^2 * exp(gamma * (c + z[t - d])) *
                    (exp(c * gamma) - exp(z[t - d] * gamma)) /
            (exp(c * gamma) + exp(z[t - d] * gamma))^3)
}
if (type == "exponential") {
  d2G_dgammadc <- sapply(times, function(t) -2 * exp(-gamma * (c - z[t - d])^2) * (c - z[t - d]) *
            (-1 + c^2 * gamma - 2 * c * z[t - d] * gamma + z[t - d]^2 * gamma))
} else {
  d2G_dgammadc <- sapply(times, function(t) -exp(gamma * (z[t - d] - c)) *
            (1 - c * gamma + z[t - d] * gamma + exp(gamma * (z[t - d] - c)) * (1 + c * gamma - z[t - d] * gamma)) /
            (1 + exp(gamma * (z[t - d] - c)))^3)
}
# convert results to (m-1)x(n-k) matrices
dG_dgamma <- matrix(dG_dgamma, nrow=(m-1), ncol=(n-k));
dG_dc <- matrix(dG_dc, nrow=(m-1), ncol=(n-k));
d2G_dgamma2 <- matrix(d2G_dgamma2, nrow=(m-1), ncol=(n-k));
d2G_dc2 <- matrix(d2G_dc2, nrow=(m-1), ncol=(n-k));
d2G_dgammadc <- matrix(d2G_dgammadc, nrow=(m-1), ncol=(n-k));
# dimension of the parameter vector
dim_par = m*(p + 1) + 2*(m - 1)

# (p + 1)x(n - k)
Y <- sapply(times, function(t) Yt(x, t, p))
Ym <- do.call(rbind, replicate(m, Y,simplify=F))
Gt <- rbind(rep(1, (n - k)), G)
Gm <- rbind(Gt, rep(0, (n - k)))
Gm <- matrix(sapply(1:m, function(j) replicate(p+1, Gm[j,] - Gm[j + 1,])), nrow=m*(p+1), byrow=T)
Phim <- (Phi - rbind(rep(0, (p + 1)), Phi[1:(m-1),]))
Phi_m <- Phim[2:m,]

# skeleton gradients and Hessians
skelGrad <- rbind(Gm * Ym, Phi_m %*% Y * dG_dgamma, Phi_m %*% Y * dG_dc)
```

```r
  # d2Skel_dPhi2
  zeros <- matrix(0, nrow=m*(p + 1), ncol=m*(p + 1))
  # (m*(p + 1))x(2*(m - 1))
  PhiSigns <- sapply(1:(m - 1), function(k)
    sapply(1:m, function(j) {
      if(k == j) return(rep(-1, p + 1)) else if(j - 1 == k) return(rep(1, p + 1)) else return(rep(0, p + 1))
  }))
  PhiSigns <- cbind(PhiSigns, PhiSigns)
  PhiCornerMatrices <- lapply(1:(n - k), function(t) PhiSigns * cbind(
    matrix(rep(dG_dgamma[,t], m*(p + 1)), ncol=(m-1), byrow=T),
    matrix(rep(dG_dc[,t], m*(p + 1)), ncol=(m-1), byrow=T)
  ))
  PhiDiagMatrices <- lapply(1:(n - k), function(t) rbind(
    cbind( diag(d2G_dgamma2[,t], nrow=(m-1), ncol=(m-1)), diag(d2G_dgammadc[,t], nrow=(m-1), ncol=(m-1)) ),
    cbind( diag(d2G_dgammadc[,t], nrow=(m-1), ncol=(m-1)), diag(d2G_dc2[,t], nrow=(m-1), ncol=(m-1)) )
  ))
  skelHessians <- lapply(1:(n - k), function(t) rbind(
    cbind(zeros, PhiCornerMatrices[[t]]* Ym[,t]),
    cbind(t(PhiCornerMatrices[[t]]* Ym[,t]) ,
          PhiDiagMatrices[[t]] *
            do.call(cbind,
              replicate(2, do.call(rbind, replicate(2,
                diag(as.numeric(Phi_m %*% Y[,t]), nrow=m-1,ncol=m-1),
            simplify=F)),simplify=F))
    )
  ))
  rsGradients <- lapply(1:(n - k), function(t)
    2 * res[t,] * skelGrad[,t]
  )
  rsHessians <- lapply(1:(n - k), function(t)
    2 * (sapply(1:dim_par, function(i) sapply(1:dim_par, function(j)
      skelGrad[i, t] * skelGrad[j, t])) + res[t,] * skelHessians[[t]])
  )
  MeanHessian <- Reduce('+', rsHessians) / (n - k)
  rsGrads <- lapply(1:(n - k), function(t) rsGradients[[t]] %*% t(rsGradients[[t]]))
  MeanInfMatrix <- Reduce('+', rsGrads) / (n - k)
  invMeanHessian <- inv(MeanHessian)
  # parameter covariance matrix estimate
  (invMeanHessian %*% MeanInfMatrix %*% invMeanHessian) / n
}
```

```
##
## LSTAR(2,1,-0.1,10) PhiParams & standard errors:
##  Phi:

## [1]  0.14844846  0.95684360  0.19873972 -0.07726319  0.83045225  0.13664019

## Phi_se:

## [1] 0.45350968 1.03250814 0.10470951 0.09760295 0.22380264 0.10832803

##
## LSTAR(2,1,-0.1,0.1,10,11) PhiParams & standard errors:
##  Phi:

## [1] -0.21249835  0.32557665  0.04959413  0.03538595 -2.06519468  0.70066290
## [7]  0.09524044  0.75428752 -0.07661200

## Phi_se:

## [1] 0.5193461 0.8390685 0.5202534 0.2855643 6.3644170 3.1718359 1.4882409
```

```
## [8] 1.9094895 0.2314645
```

This procedure can then be added to the postprocessing method:

```r
EstimSTAR_m_postproc <- function(model) {
  m <- model$nReg; x <- model$data; n <- model$n;
  p <- model$p; c <- model$c; gamma <- model$gamma;
  type <- model$type

  y <- as.matrix(x[(p + 1):n])
  Phi <- model$PhiParams;
  model$PhiStErrors <- sqrt( diag(starParamCovMatrix(xt, p=p, d=d, res, Phi, c=c, gamma=gamma, m=m))[1:(m*(p+1))] )

  # Information criteria
  starIC <- as.list(IC(n=n, p=p, sigmaSq=model$resSigmaSq, m=m))
  model$AIC <- starIC$AIC
  model$BIC <- starIC$BIC
  return(model)
}

str( EstimSTAR_m_postproc(EstimSTAR_m(xt, p=2, d=2, c=0, gamma=10)) ) # 2 regimes
```

```
## List of 16
##  $ name       : chr "LSTAR(2,2,0,10)"
##  $ nReg       : int 2
##  $ type       : chr "logistic"
##  $ p          : num 2
##  $ d          : num 2
##  $ c          : num 0
##  $ gamma      : num 10
##  $ data       : num [1:189] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 189
##  $ skel       : num [1:187, 1] 0.0947 0.1933 0.118 -0.0343 -0.0808 ...
##  $ residuals  : num [1:187, 1] 0.1053 -0.0933 -0.218 -0.0657 -0.0192 ...
##  $ resSigmaSq : num 0.027
##  $ PhiParams  : num [1:6] -0.0684 0.2047 0.3672 0.1028 0.9621 ...
##  $ PhiStErrors: num [1:6] 0.0683 0.221 0.1082 0.0766 0.1943 ...
##  $ AIC        : num 1233
##  $ BIC        : num 1259
```

```r
str( EstimSTAR_m_postproc(EstimSTAR_m(xt, p=2, d=2, c=c(0, 0.2), gamma=c(10, 12), m=3)) ) # 3 regimes
```

```
## List of 16
##  $ name       : chr "LSTAR(2,2,0,0.2,10,12)"
##  $ nReg       : int 3
##  $ type       : chr "logistic"
##  $ p          : num 2
##  $ d          : num 2
##  $ c          : num [1:2] 0 0.2
##  $ gamma      : num [1:2] 10 12
##  $ data       : num [1:189] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 189
##  $ skel       : num [1:187, 1] 0.1003 0.1632 0.1187 -0.0264 -0.0782 ...
##  $ residuals  : num [1:187, 1] 0.0997 -0.0632 -0.2187 -0.0736 -0.0218 ...
##  $ resSigmaSq : num 0.0267
##  $ PhiParams  : num [1:9] -0.0347 0.3063 0.3804 0.0273 0.5822 ...
##  $ PhiStErrors: num [1:9] 0.148 0.37 0.104 0.18 1.712 ...
##  $ AIC        : num 1245
##  $ BIC        : num 1287
```

6.3 2-Regime STAR Estimation Procedure

Now that we've prepared all necessary methods, we proceed to implementing an estimation procedure, similar to the search for SETAR models in sections 2.3 and 4.4:

```
# limit the c parameter by the 7.5-th and 92.5 percentile
m <- 2
cmin <- as.numeric(quantile(xt, 0.075)); cmax <- as.numeric(quantile(xt, 0.925));
h = (cmax - cmin) / 50 # determine the step by which c should be iterated
hypars <- starTest$LSTARs # hyperparameter array

models_s <- list() # s as in smooth
models_s_columns <- list()

suppressMessages(pkgTest("foreach"))
suppressMessages(pkgTest("doParallel"))
pkgs <- c("zeallot", "matlib")

n_cores <- (detectCores() - 1)

for(i in 1:nrow(hypars)) {
  p <- hypars[i, 1]; d <- hypars[i, 2];

  cl <- makeCluster(n_cores)
  registerDoParallel(cl)
  pdModels <- foreach(gamma = seq(from = 0.5, to = 10, by = 0.5), .packages = pkgs) %:%
      foreach(c = seq(cmin, cmax, by = h), .packages = pkgs) %dopar% {
          tmp <- EstimSTAR_m(xt, p, d, c, gamma) # try to run the function
          # then test whether it returns`NA` as a result
          if (!as.logical(sum(is.na(tmp))) ) {
              list(tmp)
          }
      }

  stopCluster(cl)

  pdOmitted <- lapply(unlist(pdModels, recursive=F),
      function(m) if(!is.logical(m) && !is.null(m)) m else list(list(resSigmaSq = Inf)))
  sigmas <- as.numeric(lapply(pdOmitted, function(m) m[[1]]$resSigmaSq))
  s_orders <- order(sigmas)
  # only the model whose parameter c gives the lowest residual square sum is chosen for postprocessing
  min_sigma_model <- EstimSTAR_m_postproc(pdOmitted[[ s_orders[1] ]][[1]])

  models_s[[length(models_s) + 1]] <- min_sigma_model
  models_s_columns[[length(models_s_columns) + 1]] <- c(
      min_sigma_model$type,
      p, d, round(min_sigma_model$c, digits=4),
      round(min_sigma_model$gamma, digits=4),
      round(min_sigma_model$AIC, digits=4),
      round(min_sigma_model$BIC, digits=4),
      round(min_sigma_model$resSigmaSq, digits=4))
}
```

```
##   transition p d      c gamma       AIC       BIC resSigmaSq
## 1   logistic 2 2 0.1314    10 1235.1634 1261.0974     0.0267
## 2   logistic 4 2  0.084    10 1244.4277 1283.3287     0.0262
## 3   logistic 5 2 0.0682    10 1251.7345  1297.119     0.0256
## 4   logistic 7 1    0.4    10 1262.9404 1321.2918     0.0249
## 5   logistic 7 2 0.0524    10   1262.99 1321.3414     0.0249
## 6   logistic 8 2    0.4    10 1270.2884 1335.1233     0.0244
```

```
## 7   logistic 9 2    0.4    10 1275.5283 1346.8468     0.0241
```

Again, we sorted the models by their BIC. We seem to have a rather small sample, but that is most likely due to only having 2 models with detected $STAR$-type nonlinearity. Estimated parameters were obtained during the postprocessing phase on a model with lowest $\hat{\sigma}^2_{\varepsilon}$.

On top of that the postprocessing phase also contains an estimation of standard errors of parameters:

```
## $`2/2/0.1314/10`
##                 [,1]      [,2]      [,3]      [,4]      [,5]       [,6]
## Phi      -0.02606206 0.3141173 0.3861456 0.1340813 1.1096505 -0.5167062
## stdError  0.02762157 0.3339043 0.3079119 0.1969468 0.2442104  0.4965146
##
## $`4/2/0.084/10`
##                 [,1]      [,2]      [,3]      [,4]       [,5]       [,6]
## Phi      -0.02584385 0.2448474 0.3857310 0.2221966 -0.1509349 0.10084616
## stdError  0.01605553 0.1350514 0.1472423 0.1311677  0.1169612 0.04153269
##               [,7]       [,8]       [,9]      [,10]
## Phi      1.0820148 -0.2919550 -0.2243718 0.05223799
## stdError 0.1720662  0.2215853  0.2359855 0.21124570
##
## $`5/2/0.0682/10`
##                 [,1]      [,2]      [,3]      [,4]        [,5]       [,6]
## Phi      -0.04248948 0.2321565 0.3591857 0.2681208 -0.07770588 -0.1564351
## stdError  0.03167622 0.1347691 0.1871064 0.1529964  0.14459071  0.1342935
##                [,7]      [,8]       [,9]      [,10]    [,11]       [,12]
## Phi      0.12949202 1.0520199 -0.3358017 -0.2194513 0.134247 -0.07968642
## stdError 0.05422055 0.1990146  0.2002276  0.1739180 0.195584  0.19038761
##
## $`7/1/0.4/10`
##                 [,1]      [,2]      [,3]       [,4]       [,5]       [,6]
## Phi      -0.01305395 0.4644766 0.2186228 0.05211537 0.09023704 -0.1353680
## stdError  0.01540583 0.1012723 0.1032089 0.11314929 0.09496949  0.1400303
##               [,7]       [,8]      [,9]    [,10]     [,11]     [,12]      [,13]
## Phi      0.0173049 0.05492962 0.1538228 0.732765 0.4069634 0.5102138 -2.0261736
## stdError 0.1217447 0.09928803 0.1352218 0.348055 0.4324081 0.5158450  0.6899586
##              [,14]    [,15]      [,16]
## Phi      -0.1071656 1.620023 -0.9436493
## stdError  0.4086061 0.549270  0.3968344
##
## $`7/2/0.0524/10`
##                 [,1]      [,2]      [,3]      [,4]        [,5]       [,6]
## Phi      -0.04521563 0.2358571 0.3611718 0.2631941 -0.06558551 -0.1370850
## stdError  0.03415944 0.1452782 0.2097411 0.1683088  0.13299271  0.1565098
##                [,7]      [,8]       [,9]     [,10]      [,11]      [,12]
## Phi      -0.1683810 0.1522735 0.12016752 1.0797643 -0.3580989 -0.1250175
## stdError  0.1421911 0.1437999 0.03920119 0.1570741  0.1748054  0.1795611
##               [,13]      [,14]     [,15]      [,16]
## Phi      -0.02873088 -0.2084309 0.5731586 -0.4068476
## stdError  0.18394645  0.1511674 0.1748472  0.1471281
##
## $`8/2/0.4/10`
##                   [,1]       [,2]      [,3]       [,4]         [,5]        [,6]
## Phi      -0.0002893663 0.50191280 0.3343700 0.08888659 -0.002650929 -0.16781039
## stdError  0.0156881499 0.07769926 0.1125815 0.10389840  0.113928880  0.09312338
##                 [,7]       [,8]         [,9]     [,10]     [,11]      [,12]
## Phi      -0.02956963 0.07408718 -0.001718691 0.2426348 1.1358110 -0.2603175
## stdError  0.14287006 0.12285477  0.099283647 0.1662739 0.2882373  0.4843620
##             [,13]      [,14]      [,15]     [,16]      [,17]      [,18]
## Phi      0.1252936 -1.3462551 -0.5134945 2.8191238 -0.1872913 -1.5756374
```

```
## stdError 0.4586055  0.5553102  0.6147243 0.7057794  0.6331101  0.4592401
##
## $`9/2/0.4/10`
##                  [,1]       [,2]      [,3]       [,4]      [,5]       [,6]
## Phi      -0.003993142 0.51614137 0.3059474 0.07564694 0.0148206 -0.1558765
## stdError  0.017210015 0.07565293 0.1104765 0.10857337 0.1006055  0.1130306
##                 [,7]       [,8]       [,9]      [,10]     [,11]     [,12]
## Phi      -0.03824676 0.04479152 -0.0763310 0.12585169 0.2750665 1.1560917
## stdError  0.09554390 0.14523712  0.1260665 0.09215054 0.1691217 0.2138622
##               [,13]      [,14]     [,15]      [,16]     [,17]      [,18]
## Phi      -0.2697316 0.04092728 -1.336113 -0.5185651 2.9629550 -0.2457963
## stdError  0.4169601 0.34694193  0.395826  0.7263939 0.5077898  0.5918702
##               [,19]      [,20]
## Phi      -1.6992237 0.09482243
## stdError  0.4100731 0.39841396
```

6.4 Visualisation