ACR2Full

Martin Cavarga

26 January, 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

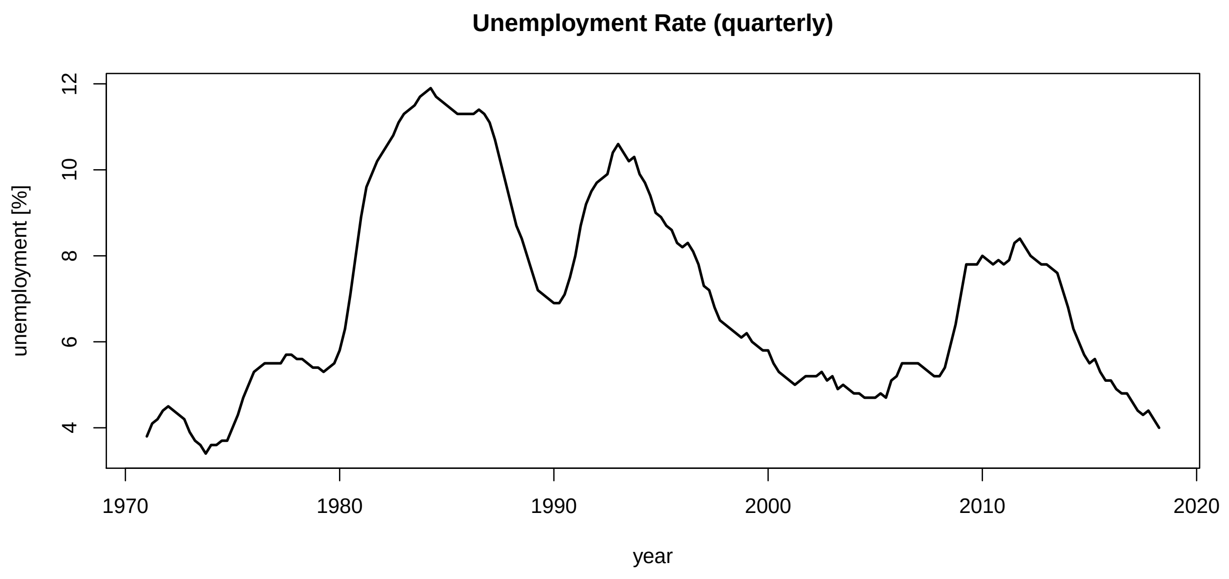
```
##   head.dat.time.
## 1        1971 Q1
## 2        1971 Q2
## 3        1971 Q3
## 4        1971 Q4
## 5        1972 Q1
## 6        1972 Q2
```

```
##   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

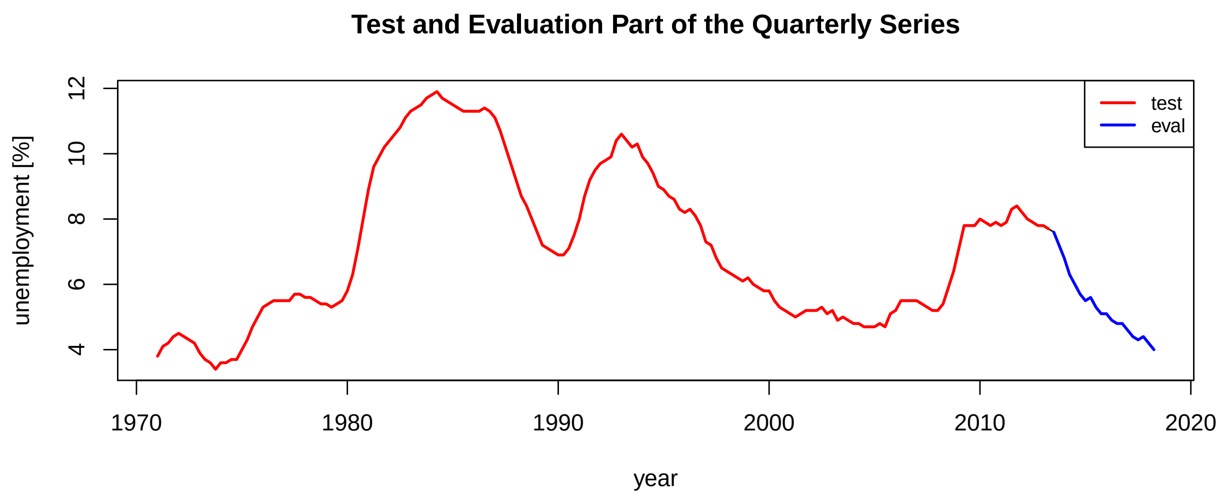


Now even thought we are working with annual data there should not be any seasonal components or trend since the results do not depend on periodic observable phenomena, but rather the complex economic situation over multiple decades. Also the data may include exponentially decaying decrease in unemployment,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

1.2: Test Part and Evaluation Part of the Time Series

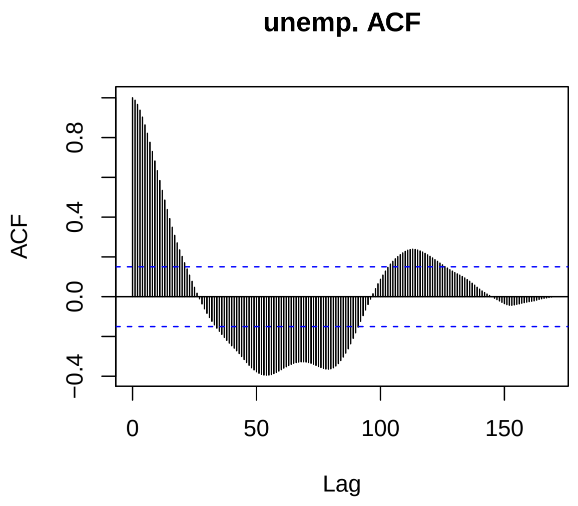
Now we separate the time series into test part, where a suitable model of a stochastic process will be found, and the evaluation part, where predictions given by such model are evaluated. Since our dataset contains quarterly data, we choose the length of the evaluation part of the time series as $L =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

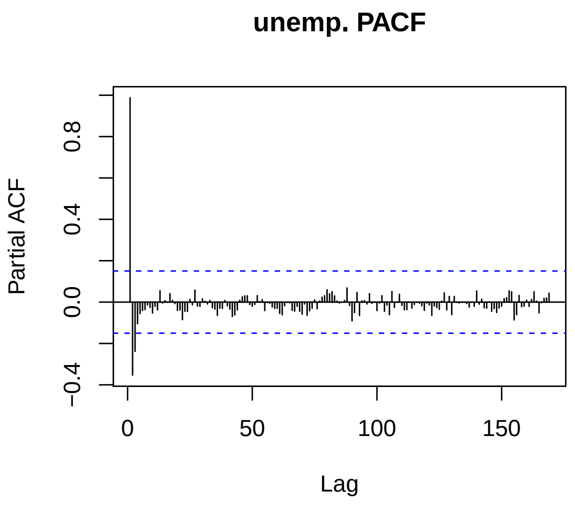
```
## [1] 20
```



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
```

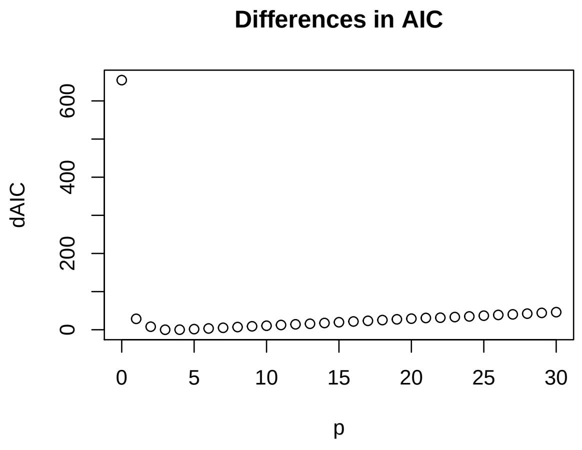


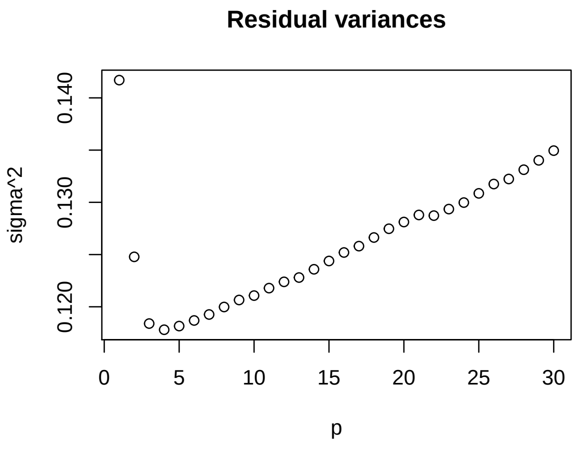


As we mentioned in section 1.1, the underlying process which gave rise to the observed results is aperiodic, yet it is undoubtedly a process with memory. Unemployment rate strongly depends (aside from other important aspects) on its own history which might extend generations into the past. The results 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.



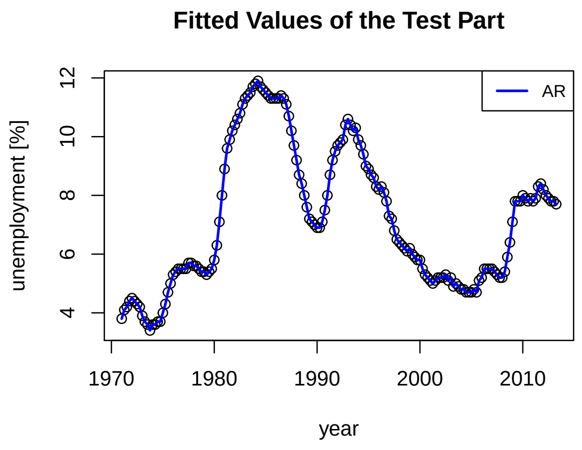


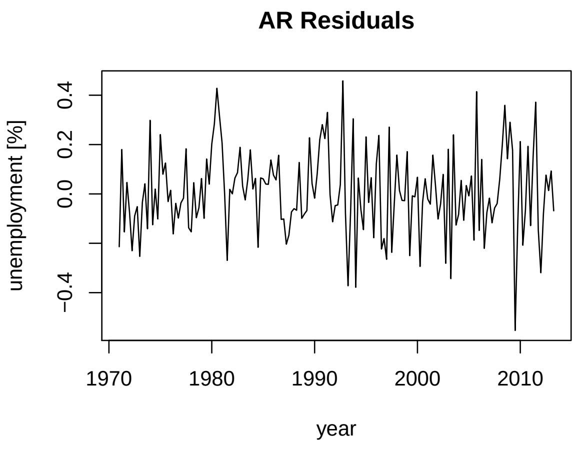
As we can see in the figures, the lowest variance of residues corresponds to an $AR(3)$ process with coefficients:

```
##           [,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
```

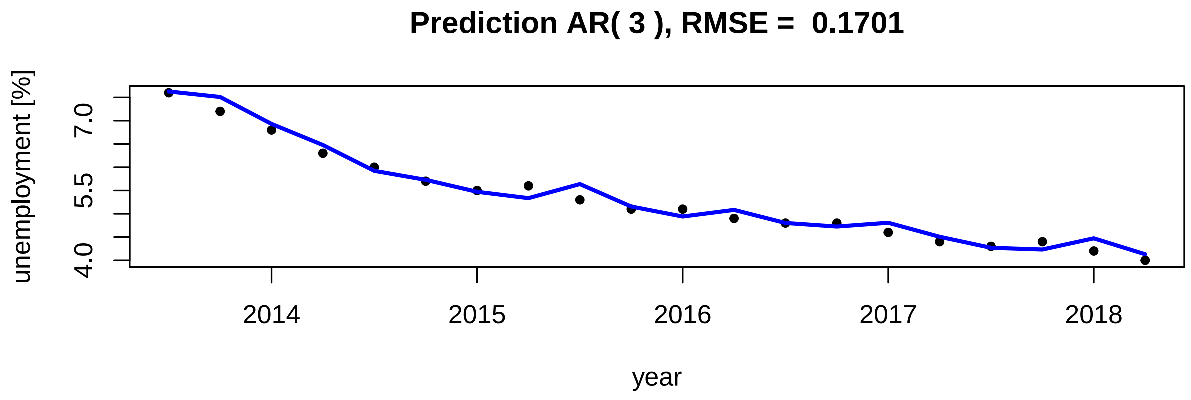




The given AR model seems to fit the time series very well, which may be due to its low oscillation rate.

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: Implementation of 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`:

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

Afterwards, we define the basis function for a single regime

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

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

```r
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:

```r
xt <- diff(as.numeric(dat$unemp))
```

```
## [1] 0.1 0.2

## [1] 0.0 0.0 0.0 1.0 0.1 0.3

## [1] -0.3 -0.1

## [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)

  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)

    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:

```
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:

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

```
## List of 15
##  $ p          : num 2
##  $ d          : num 2
##  $ c          : num 0
##  $ data       : num [1:189] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 189
##  $ PhiParams  : num [1:6] 0 0.418 0.398 0.067 0.825 ...
##  $ PhiStErrors: num [1:6] 0 0.0465 0.0598 0.0145 0.0414 ...
##  $ residuals  : num [1:187, 1] 0.1021 -0.1147 -0.2151 -0.0673 -0.0184 ...
##  $ resSigmaSq : num 0.028
##  $ n1         : num 119
##  $ n2         : num 70
##  $ resSigmaSq1: num 0.0218
##  $ resSigmaSq2: num 0.0394
##  $ AIC        : num -666
##  $ BIC        : num -645
```

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:

```
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  0.0050 119  70 -666.8254 -656.7701 0.02961197
## 2  2 1  0.0050 119  70 -666.4097 -651.3269 0.02894374
## 3  2 2  0.1077 148  41 -681.6555 -667.5231 0.02661637
## 4  3 1  0.2025 161  28 -666.6429 -648.9885 0.02801226
## 5  3 2  0.1077 148  41 -672.4633 -653.6202 0.02692823
## 6  3 3 -0.1925  47 142 -663.0007 -643.7768 0.02724960
## 7  4 1  0.1077 148  41 -661.6821 -638.1282 0.02796801
## 8  4 2  0.0050 119  70 -664.0602 -638.9221 0.02698033
## 9  4 3 -0.1925  47 142 -654.0314 -630.0016 0.02775992
## 10 4 4  0.2025 161  28 -663.9855 -641.9174 0.02735368
## 11 5 1  0.3052 172  17 -671.0349 -647.1506 0.02617107
## 12 5 2  0.1077 148  41 -666.1422 -637.8775 0.02646857
```

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):

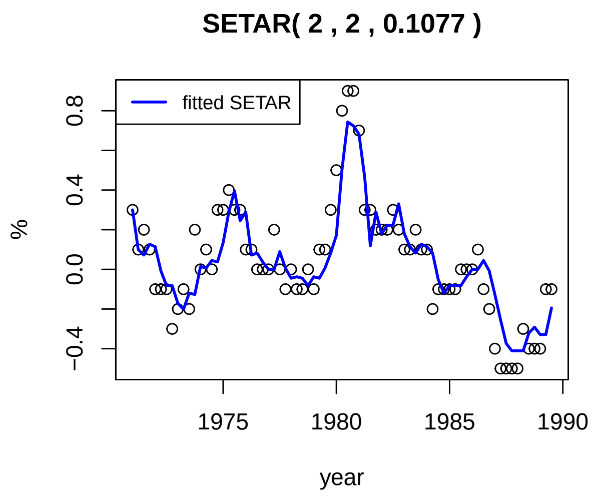
```
##    p d       c  n1  n2       AIC       BIC resSigmaSq
## 3  2 2  0.1077 148  41 -681.6555 -667.5231 0.02661637
## 1  1 1  0.0050 119  70 -666.8254 -656.7701 0.02961197
## 5  3 2  0.1077 148  41 -672.4633 -653.6202 0.02692823
## 2  2 1  0.0050 119  70 -666.4097 -651.3269 0.02894374
## 4  3 1  0.2025 161  28 -666.6429 -648.9885 0.02801226
## 11 5 1  0.3052 172  17 -671.0349 -647.1506 0.02617107
## 6  3 3 -0.1925  47 142 -663.0007 -643.7768 0.02724960
## 10 4 4  0.2025 161  28 -663.9855 -641.9174 0.02735368
## 8  4 2  0.0050 119  70 -664.0602 -638.9221 0.02698033
## 15 5 5  0.1077 148  41 -666.5125 -638.2478 0.02629603
## 7  4 1  0.1077 148  41 -661.6821 -638.1282 0.02796801
## 12 5 2  0.1077 148  41 -666.1422 -637.8775 0.02646857
```

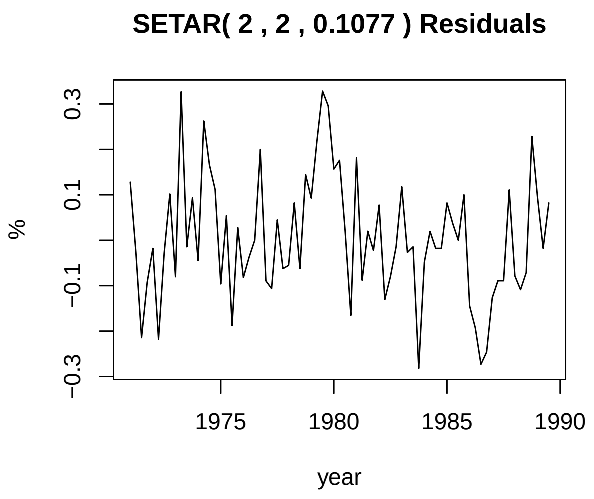
and we can also include errors of the estimated regression coefficients:

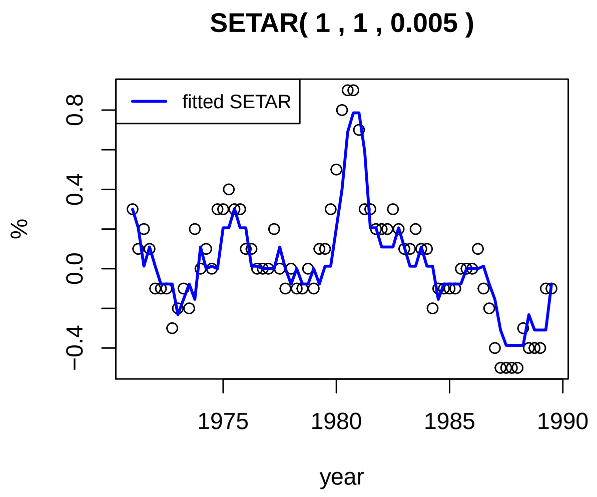
```
## $`2/2/0.1077`
##          [,1]      [,2]       [,3]       [,4]       [,5]        [,6]
## Phi         0 0.4470165 0.37460889 0.09138806 1.08026643 -0.42454326
## stdError    0 0.0365959 0.04350063 0.02448193 0.05596549  0.07400055
##
## $`1/1/0.005`
##          [,1]       [,2]        [,3]       [,4]
## Phi         0 0.77306739 -0.08385375 0.96647682
## stdError    0 0.04785017  0.01408073 0.04124099
##
## $`3/2/0.1077`
##          [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## Phi         0 0.42513468 0.33722534 0.07776119 0.08019403 1.07869738
## stdError    0 0.03806445 0.04703535 0.03721094 0.02484264 0.05656146
##                 [,7] [,8]
## Phi      -0.31056937    0
## stdError  0.09454831    0
##
## $`2/1/0.005`
##          [,1]      [,2]       [,3]        [,4]      [,5]       [,6]
## Phi         0 0.6529871 0.17407482 -0.07606073 0.8439843 0.13690162
## stdError    0 0.0553955 0.04046129  0.01428069 0.0588705 0.04653227
##
## $`3/1/0.2025`
##                 [,1]       [,2]       [,3] [,4]       [,5]      [,6]      [,7]
## Phi      -0.016031633 0.47644330 0.17961495    0 0.09565432 0.6065901 0.4742265
## stdError  0.006371236 0.04073824 0.03876079    0 0.04078723 0.1056911 0.1128485
##                 [,8]
## Phi      -0.51651552
## stdError  0.08511057
##
## $`5/1/0.3052`
##          [,1]       [,2]       [,3] [,4] [,5]        [,6]       [,7]      [,8]
## Phi         0 0.52727732 0.20998940    0    0 -0.10193788 -0.1516049 0.9469673
## stdError    0 0.03864581 0.03882394    0    0  0.03254612  0.0752377 0.1730010
##               [,9] [,10]      [,11] [,12]
## Phi      0.6897296     0 -1.2185014     0
## stdError 0.1718518     0  0.2012334     0
##
## $`3/3/-0.1925`
##          [,1] [,2]       [,3]      [,4] [,5]       [,6]       [,7]        [,8]
## Phi         0    0 0.37617199 0.3370807    0 0.77640045 0.09495748 -0.10633335
## stdError    0    0 0.07367042 0.1244072    0 0.03560396 0.04222421  0.03817885
##
```

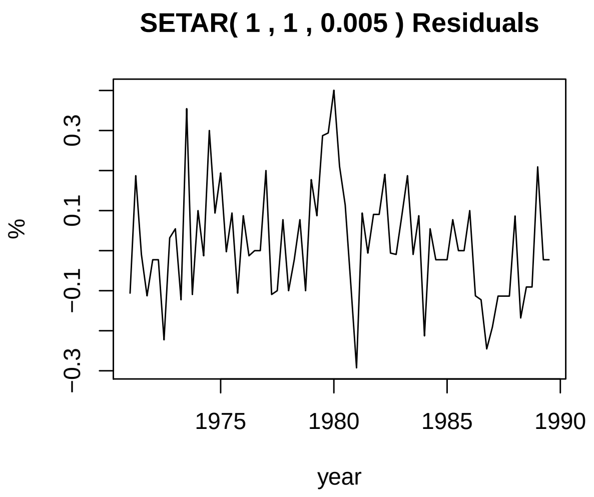
```
## $`4/4/0.2025`
##          [,1]       [,2]       [,3] [,4]        [,5]      [,6]      [,7]
## Phi         0 0.64375473 0.35315892    0 -0.16539696 0.1417356 0.6582130
## stdError    0 0.03355499 0.04093065    0  0.04170061 0.0381334 0.0917825
##                [,8]      [,9]       [,10]
## Phi      -0.3314201 0.4747251 -0.41041737
## stdError  0.1061456 0.1027790  0.09730839
##
## $`4/2/0.005`
##          [,1]      [,2]       [,3]       [,4]        [,5]       [,6]       [,7]
## Phi         0 0.3919871 0.37782961 0.21044159 -0.15760552 0.06034065 0.85546918
## stdError    0 0.0474868 0.06443638 0.04665194  0.03878837 0.01497240 0.04218251
##          [,8]        [,9] [,10]
## Phi         0 -0.21231417     0
## stdError    0  0.06274908     0
##
## $`5/5/0.1077`
##          [,1]       [,2]       [,3] [,4] [,5]        [,6] [,7]       [,8]
## Phi         0 0.63042460 0.36030920    0    0 -0.14851822    0 0.47118620
## stdError    0 0.03592345 0.04353335    0    0  0.04669166    0 0.07193865
##                 [,9]      [,10]      [,11] [,12]
## Phi      -0.19079304 0.53307667 -0.3303657     0
## stdError  0.07679886 0.08963108  0.1003846     0
##
## $`4/1/0.1077`
##                 [,1]       [,2]       [,3] [,4] [,5] [,6]       [,7]
## Phi      -0.019148364 0.42794647 0.22064662    0    0    0 0.73762740
## stdError  0.007248623 0.04499754 0.04109007    0    0    0 0.08730348
##                [,8] [,9]       [,10]
## Phi      0.22543326    0 -0.26154596
## stdError 0.09092595    0  0.06522126
##
## $`5/2/0.1077`
##          [,1]       [,2]       [,3]       [,4] [,5]        [,6]       [,7]
## Phi         0 0.43098256 0.36558788 0.14174809    0 -0.11613653 0.11594786
## stdError    0 0.03820563 0.04794712 0.04242419    0  0.03735369 0.02803376
##                [,8]        [,9] [,10] [,11] [,12]
## Phi      1.03987845 -0.36677650     0     0     0
## stdError 0.05995444  0.09786554     0     0     0
```

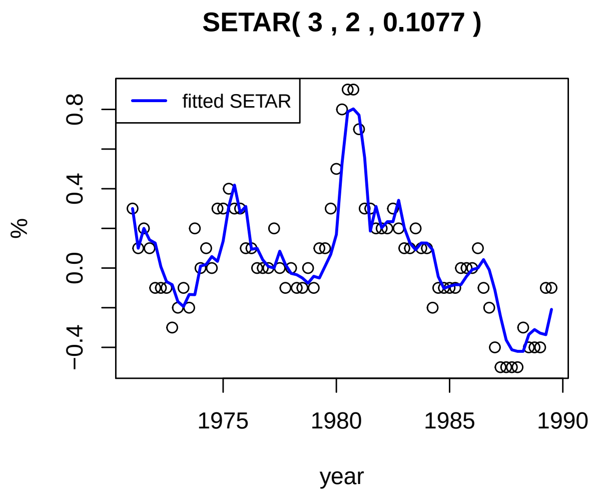
We can now visualize the results of the top 3 models:

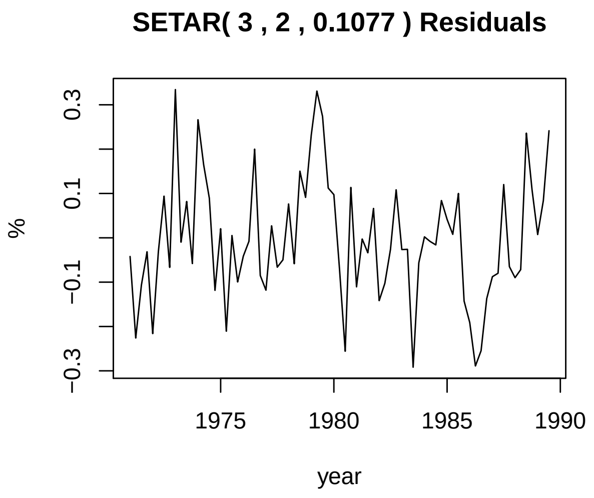










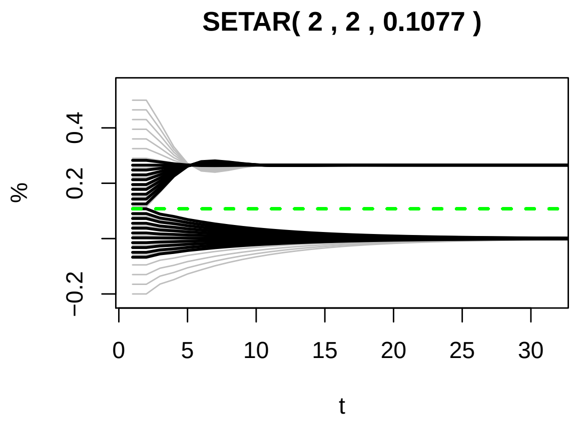


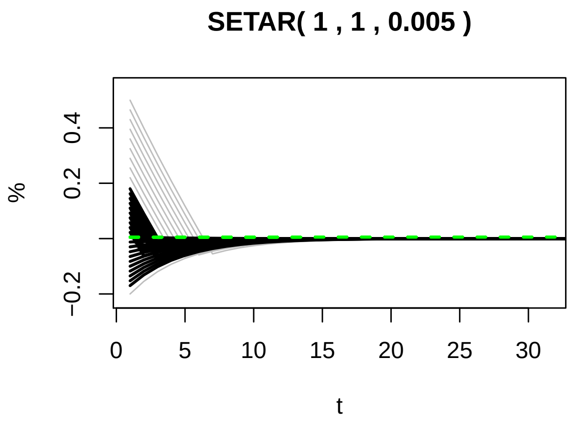
The results suggest that the best SETAR models have a threshold `c` quite close to zero and more-or-less the same

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.

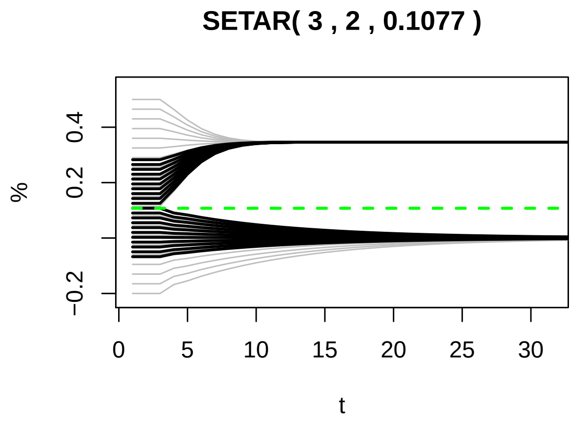
2.4: SETAR Equilibria and Equilibrium Simulations

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 $\mathrm{id}x = x$, 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.

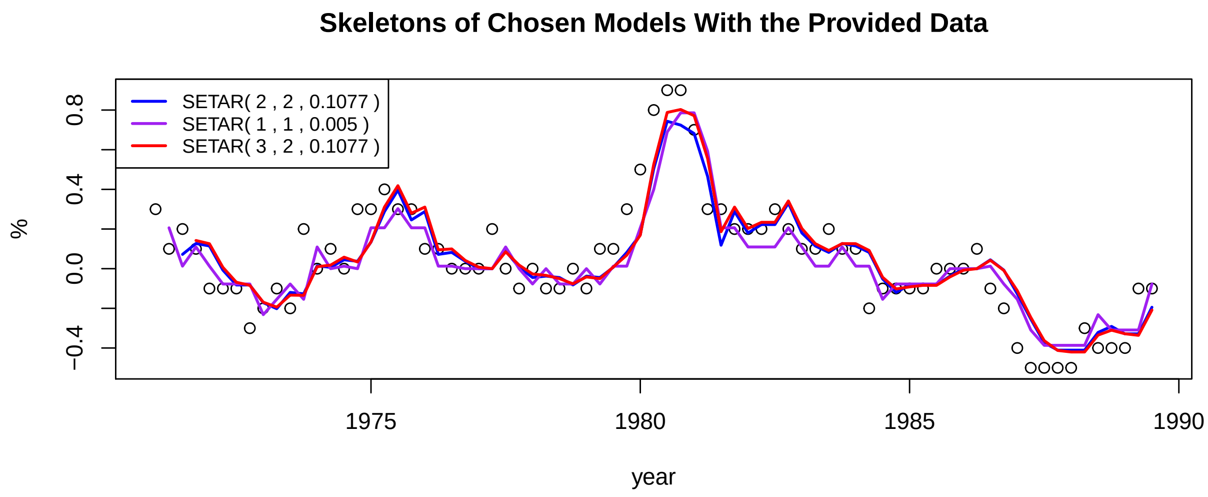




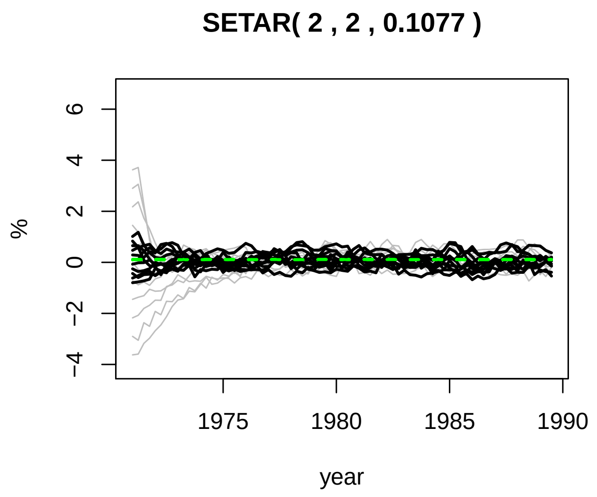
```
## $`SETAR( 3 , 2 , 0.108 ) equilibria`
## [1] 0.0000 0.2654
##
## $`SETAR( 3 , 2 , 0.108 ) equilibria`
## [1] 0
##
## $`SETAR( 3 , 2 , 0.108 ) equilibria`
## [1] 0.0000 0.3459
```

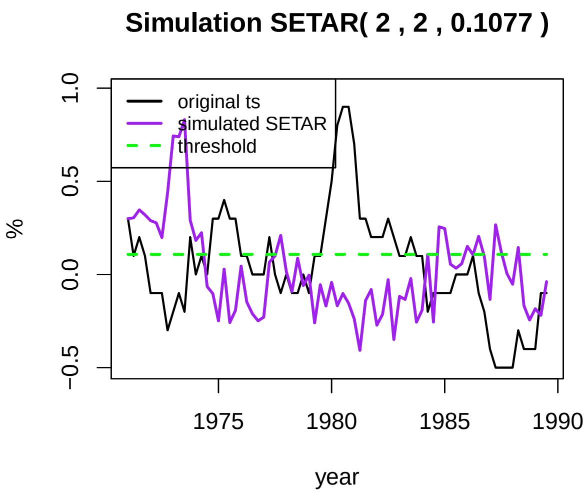


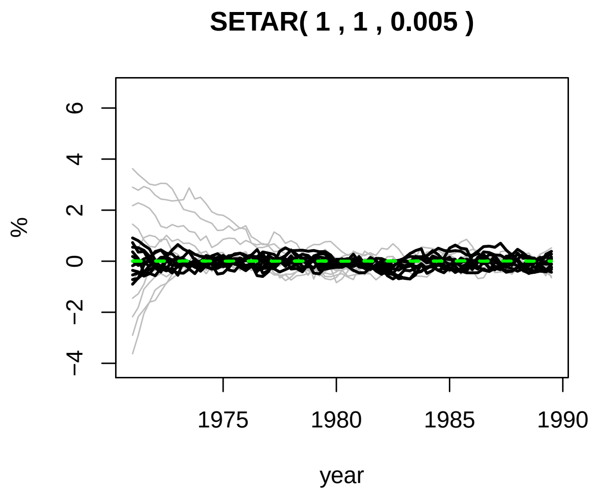
As we see, the trajectories of the top 3 models gravitate towards 0 in all models, but in the first and second model they can end up in one more position, close to zero. It might also be interesting to see how the trajectories evolve when we add an iid noise on top of the model skeleton. First we observe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

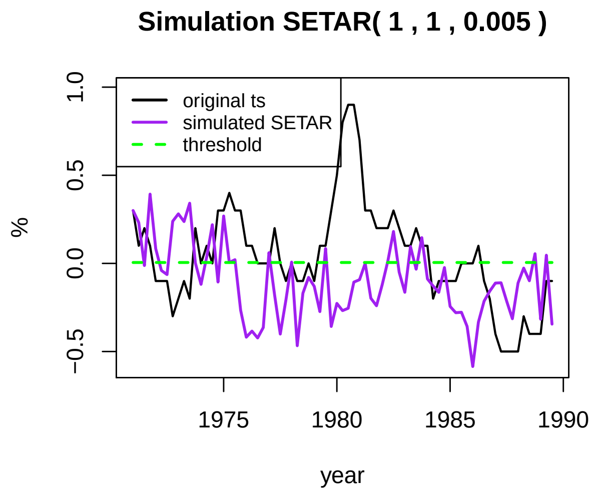


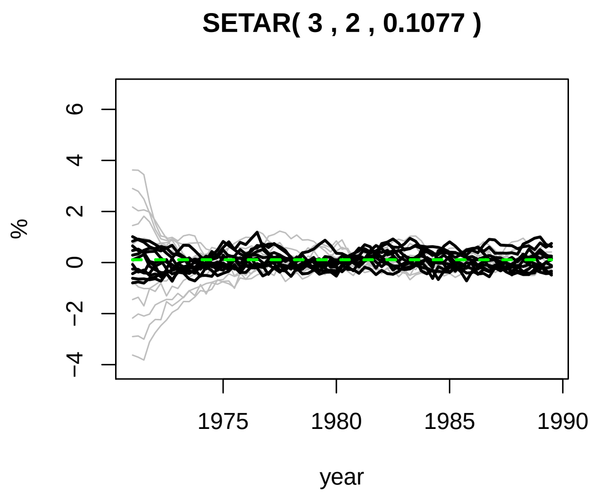
And then we carry out multiple simulations with initial conditions close to the threshold. The added noise will have the same deviance as the residual square sum.

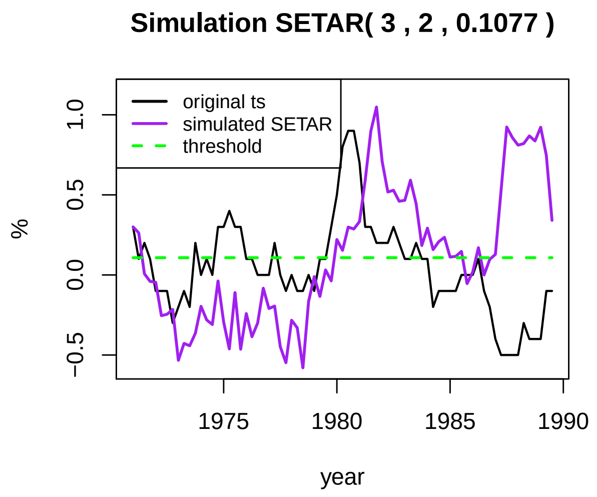












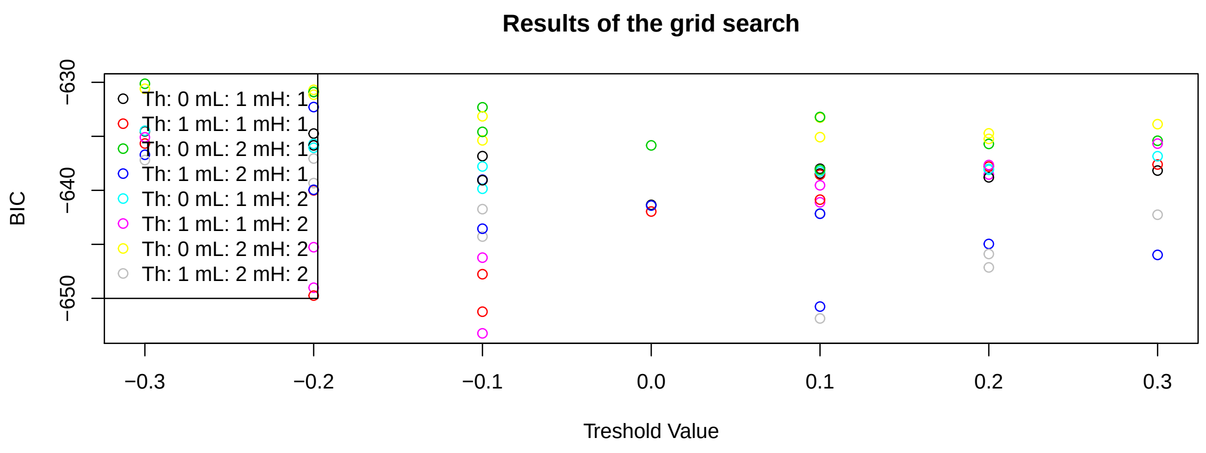
The trajectories of all of the first three models seem to gravitate toward 0 significantly fast (or alternatively: towards their threshold values which are close to zero as well). The relatively low oscillation rate of the original time series suggests that the differences of this time series will, at 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:

```r
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 20 possible threshold values within regimes with sufficient ( 10% ) number of observations
## Searching on  80  combinations of thresholds ( 20 ), thDelay ( 2 ) and m ( 2 )
```



```
## Results of the grid search for 1 threshold
##    thDelay m   th       BIC
## 1        1 1 -0.1 -653.2390
## 2        1 2  0.1 -651.8608
## 3        1 1 -0.1 -651.2349
## 4        1 2  0.1 -650.7615
## 5        1 1 -0.2 -649.7445
## 6        1 1 -0.2 -649.0154
## 7        1 1 -0.1 -647.7634
## 8        1 2  0.2 -647.1349
## 9        1 1 -0.1 -646.2319
## 10       1 2  0.3 -645.9771
```

the estimated thDelay corresponds to `d-1`.

```
## List of 15
##  $ p          : num 2
##  $ d          : num 1
##  $ c          : num -0.1
##  $ data       : num [1:189] 0.3 0.1 0.2 0.1 -0.1 ...
##  $ n          : int 189
##  $ PhiParams  : num [1:6] 0.0676 0.8578 0 -0.0224 0.6647 ...
##  $ PhiStErrors: num [1:6] 0.01958 0.07962 0 0.00756 0.04184 ...
##  $ residuals  : num [1:187, 1] 0.096 -0.0305 -0.184 -0.0311 -0.0818 ...
##  $ resSigmaSq : num 0.0292
##  $ n1         : num 65
##  $ n2         : num 124
##  $ resSigmaSq1: num 0.032
##  $ resSigmaSq2: num 0.0282
##  $ AIC        : num -650
##  $ BIC        : num -630
```

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 1 -0.1 -655.5737
## 2        1 2  0.1 -654.7770
## 3        1 2  0.1 -653.6045
## 4        1 1 -0.1 -653.4817
## 5        1 1 -0.2 -651.7920
## 6        1 1 -0.2 -651.0374
## 7        1 2  0.2 -649.8661
## 8        1 1 -0.1 -649.7745
## 9        4 2  0.1 -648.7036
## 10       1 2  0.2 -648.5209
```

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:  0.855615 0.144385 from th: 0.2
##  1 T: Trim not respected:  0.8510638 0.1489362 from th: 0.2
##  1 T: Trim not respected:  0.8548387 0.1451613 from th: 0.2
```

```
coeffComparison
```

```
## $`2 / 2 / 0.1077`
##             const.L    phiL.1    phiL.2      const.H    phiH.1     phiH.2     th
## inbuilt -0.01776063 0.4484198 0.2619552 -0.03164651 0.9628571 -0.1110418 0.1000
## custom   0.00000000 0.4470165 0.3746089  0.09138806 1.0802664 -0.4245433 0.1077
##
## $`1 / 1 / 0.005`
##            const.L    phiL.1     const.H    phiH.1    th
## inbuilt 0.01546135 0.7730673 -0.08385378 0.9664769 0.000
## custom  0.00000000 0.7730674 -0.08385375 0.9664768 0.005
##
## $`3 / 2 / 0.1077`
##             const.L    phiL.1    phiL.2     phiL.3    const.H    phiH.1
## inbuilt -0.01993301 0.4540914 0.1736989 0.07281194 0.05031824 0.6395189
## custom   0.00000000 0.4251347 0.3372253 0.07776119 0.08019403 1.0786974
##             phiH.2    phiH.3     th
## inbuilt  0.5246069 -0.517298 0.2000
## custom  -0.3105694  0.000000 0.1077
```

```
# 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 0.02844930 0.02661637
## rss 2 0.02946193 0.02961197
## rss 3 0.02774780 0.02692823
```

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 , 0.1077 )
## 2 SETAR( 1 , 1 , 0.00500000000000039 )
## 3               SETAR( 3 , 2 , 0.1077 )
```

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))))
  # (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
## 2 / 2 / 0.1077  FALSE  TRUE FALSE
## 1 / 1 / 0.005    TRUE  TRUE  TRUE
## 3 / 2 / 0.1077  FALSE  TRUE FALSE
## 2 / 1 / 0.005    TRUE  TRUE  TRUE
## 3 / 1 / 0.2025  FALSE  TRUE FALSE
## 5 / 1 / 0.3052  FALSE  TRUE FALSE
## 3 / 3 / -0.1925  TRUE  TRUE  TRUE
## 4 / 4 / 0.2025  FALSE  TRUE FALSE
## 4 / 2 / 0.005   FALSE  TRUE FALSE
## 5 / 5 / 0.1077  FALSE  TRUE FALSE
## 4 / 1 / 0.1077  FALSE  TRUE FALSE
## 5 / 2 / 0.1077  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
## 16.902125068  7.687275530  0.008293265
```

```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
  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
## 1.670958e+01 7.814728e+00 8.108925e-04
```

We can easily automate the testing procedure in the following loop:

```
alpha = 0.05
results <- list()
nonlinear <- list()
nonLinCount <- 0
for (i in 1:12) {
  p <- models[[ orders[i] ]]$p; d <- models[[ orders[i] ]]$d; c <- models[[ orders[i] ]]$c;
  hansenResult <- Hansen(d, c, models[[ orders[i] ]]$PhiParams)
  if (FALSE %in% hansenResult) {
    hansenResult <- FALSE
    testResult <- LMtest(xt, p, d)
  } else {
    hansenResult <- TRUE
    testResult <- LRtest(xt, p, models[[ orders[i] ]]$resSigmaSq)
  }
  if (testResult[3] < alpha) {
    nonLinCount <- nonLinCount + 1
    nonlinear[[nonLinCount]] <- models[[ orders[i] ]]
  }
  results[[i]] <- append(cbind(p, d, c, hansenResult), testResult)
}
results <- data.frame(matrix(unlist(results), nrow=12, byrow=T))
colnames(results) <- c("p", "d", "c", "Hansen Cond.", "TS", "CV", "p-value")
row.names(results) <- orders[1:12]
results[,4] <- as.logical(results[,4])

results
```

```
##    p d       c Hansen Cond.        TS        CV      p-value
## 3  2 2  0.1077        FALSE 16.709581  7.814728 0.0008108925
## 1  1 1  0.0050         TRUE -2.467044  7.687276 0.8167197359
## 5  3 2  0.1077        FALSE 16.907313  9.487729 0.0020147602
## 2  2 1  0.0050         TRUE  1.853695  7.687276 0.2221925179
## 4  3 1  0.2025        FALSE  6.619573  9.487729 0.1574105391
## 11 5 1  0.3052        FALSE  8.858398 12.591587 0.1816996934
## 6  3 3 -0.1925         TRUE 12.807731  7.687276 0.0176960795
## 10 4 4  0.2025        FALSE  7.373623 11.070498 0.1943030316
## 8  4 2  0.0050        FALSE 16.821155 11.070498 0.0048517734
## 15 5 5  0.1077        FALSE 10.169591 12.591587 0.1176894456
## 7  4 1  0.1077        FALSE 10.303332 11.070498 0.0670827796
## 12 5 2  0.1077        FALSE 18.580085 12.591587 0.0049348331
```

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

```
##                 unlist.res.
## 1  SETAR( 2 , 2 , 0.1077 )
## 2  SETAR( 3 , 2 , 0.1077 )
## 3 SETAR( 3 , 3 , -0.1925 )
```

The remaining models can be considered non-linear.

3.3 Modified LR Test Via Boostrapping

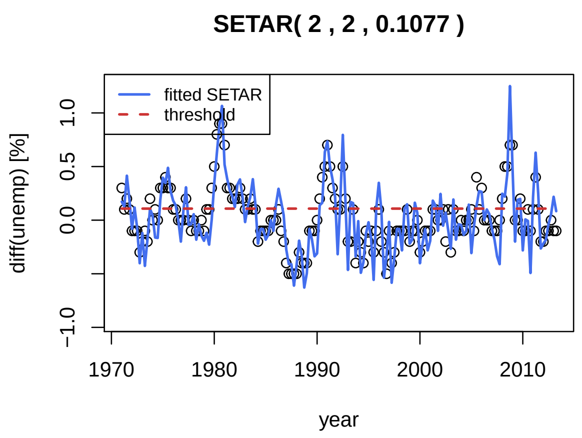
The proposed LR test has a significant drawback in the fact that it can only be done when Hansen's conditions are satisfied. This is due to the fact that we do not know the distribution of the resulting F-statistic. According to Hansen (1996), however, the distribution of a bootstrapped statistic $F*$ 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:

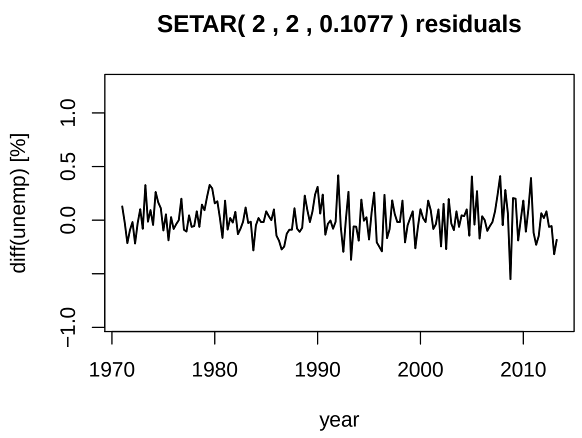
```
...

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)
  ...
}
...
```

```
##         TV      CV  p-value      time
## 1  22.8902 15.4298        0 ***  6.4 s
## 2   -2.467  7.6873  0.81672     0.02 s
## 3  24.5741 26.2838      0.5     7.71 s
## 4   1.8537  7.6873 0.222193        0 s
## 5  13.9342 22.7636      0.5     7.35 s
## 6  19.6944 30.2305      0.5     8.52 s
## 7  12.8077  7.6873 0.017696   *    0 s
## 8  24.0823 32.3572      0.5     7.18 s
## 9  24.0823 25.1431      0.5     7.08 s
## 10 27.7311 25.9696        0 *** 7.77 s
## 11 19.3243 29.6145      0.5     7.14 s
## 12 25.8791 30.1832      0.5     7.73 s
```

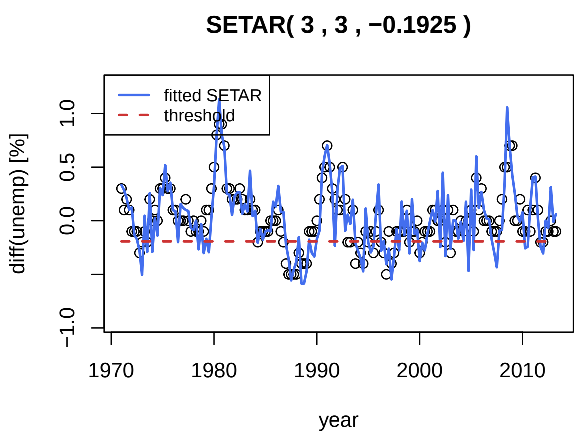
3.4 Visualisation of Non-Linear Models

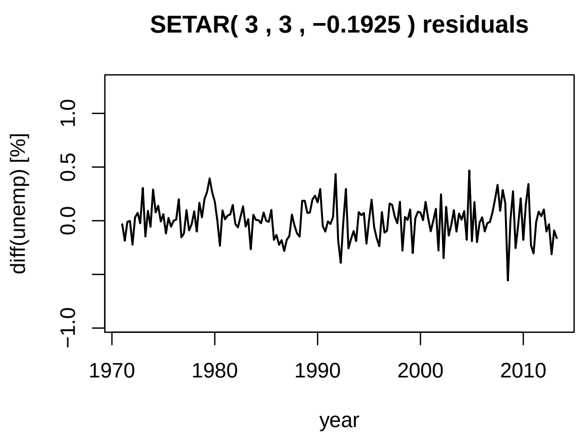
From the results of the previous procedure, we will visualize the models for which the linearity null-hypothesis was rejected based on the LR and LM tests:



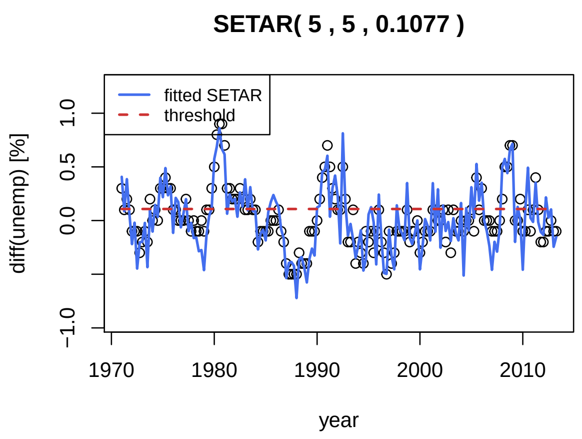


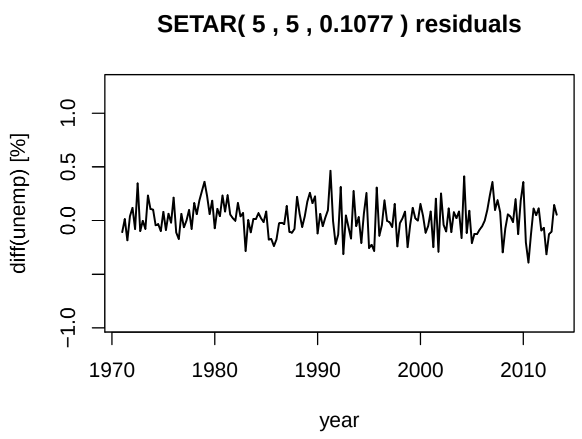
```
## resSigmaSq
## 0.02661637
```





```
## resSigmaSq
##  0.0272496
```





```
## resSigmaSq
## 0.02629603
## [1]  3  6 15
```

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

# SETAR3 skeleton
Xt <- 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 <- Xt(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,] 0.07248048  0.2302972  0.00215751  0.00000000  0.0000000  0.00000000
##  [2,] 0.23029715  1.1980070 -0.38034510  0.00000000  0.0000000  0.00000000
##  [3,] 0.00215751 -0.3803451  0.57098118  0.00000000  0.0000000  0.00000000
##  [4,] 0.00000000  0.0000000  0.00000000  0.01164245 -0.0209984  0.00331308
##  [5,] 0.00000000  0.0000000  0.00000000 -0.02099840  1.5909791 -0.30657670
##  [6,] 0.00000000  0.0000000  0.00000000  0.00331308 -0.3065767  0.38170432
##  [7,] 0.00000000  0.0000000  0.00000000  0.00000000  0.0000000  0.00000000
##  [8,] 0.00000000  0.0000000  0.00000000  0.00000000  0.0000000  0.00000000
##  [9,] 0.00000000  0.0000000  0.00000000  0.00000000  0.0000000  0.00000000
##              [,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
```

```
# skeleton of a model with regression coeffs: theta
SkeletonSETAR3 <- function(x, t, p, d, c, theta, z = x) c(theta %*% CovMat3(x, p, d, c)[t,])
SkeletonSETAR3(xt, t=3, p=2, d=1, c=c(-0.1, 0.2), rep(1, 3*3))
```

```
## [1] 0.1927936
```

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 $(0.5 \le r \le 1.5)$. If the data is generated by an iid process, the correlation integral should approach $C(q, r) \to C(1, r)^q$.

```
## possible remaining nonlinearity for SETAR( 2 , 2 , 0.1077 ) with mean p-val =  0.875
## remaining nonlinearity rejected SETAR( 1 , 1 , 0.005 ) with mean p-val =  0.25
## possible remaining nonlinearity for SETAR( 3 , 2 , 0.1077 ) with mean p-val =  0.75
```

5. Predictions via SETAR Models and Their Evaluation

6. Tests for Non-Linearity of STAR Models