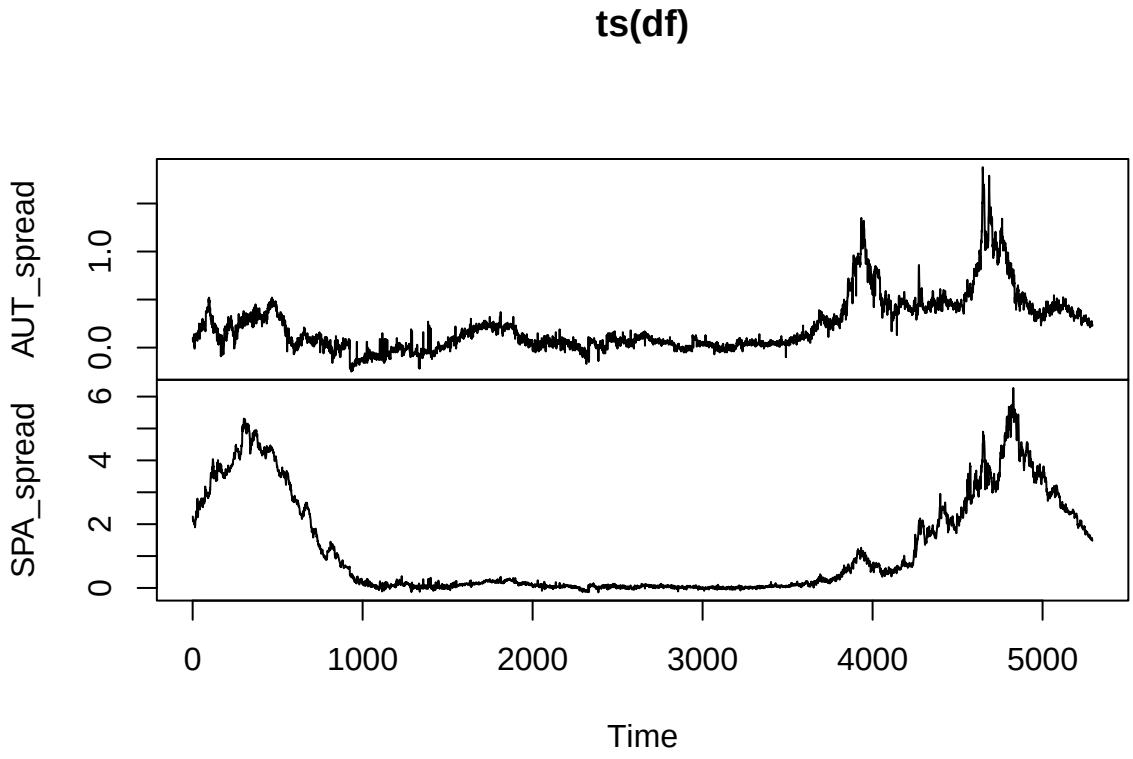
Ex 3 :

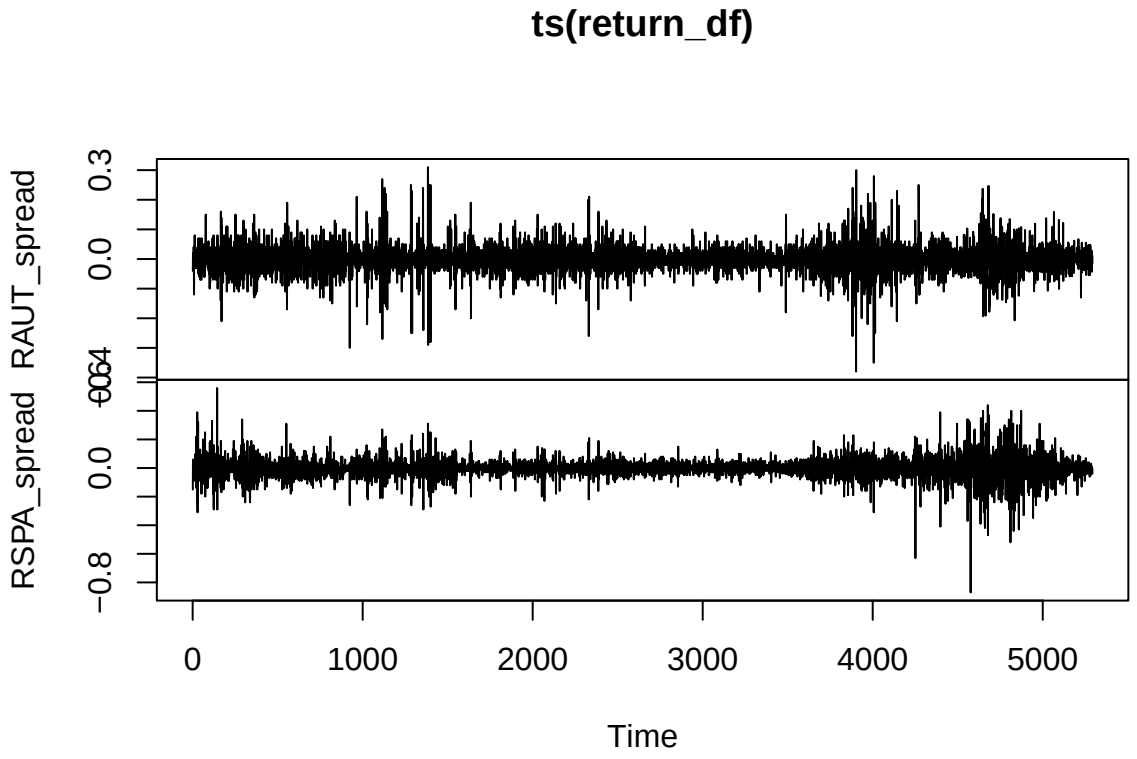
Noha

2025-11-27

```
# Construct yield spreads vs Germany
#data <- data %>%
#  mutate(
#    AUT_spread = data$`Austria, Government Benchmarks, Macrobond, 10 Year, Yield` - data$`Germany, Gov
#    SPA_spread = data$`Spain, Government Benchmarks, Macrobond, 10 Year, Yield` - data$`Germany, Gover
#  ) %>%
#  drop_na()

# Convert to time series
#AUT_spread<-xts(data$AUT_spread, order.by = data$Date...7 )
#SPA_spread<-xts(data$SPA_spread, order.by = data$Date...39 )

#AUT_spread <- ts(AUT_spread)
#SPA_spread <- ts(SPA_spread)

#RAUT_spread = diff(log(AUT_spread), lag = 1)
#RSPA_spread = diff(log(SPA_spread), lag = 1)

#df = cbind(AUT_spread, SPA_spread)
#log_df = cbind(RAUT_spread, RSPA_spread)

# Construct yield spreads vs Germany
data <- data %>%
  drop_na()

# Convert to time series
GER <- xts(data$`Germany, Government Benchmarks, Macrobond, 10 Year, Yield`, order.by = data$Date...15)
AUT <- xts(data$`Austria, Government Benchmarks, Macrobond, 10 Year, Yield`, order.by = data$Date...7)
SPA <- xts(data$`Spain, Government Benchmarks, Macrobond, 10 Year, Yield`, order.by = data$Date...39 )

AUT_spread <- ts(AUT - GER)
SPA_spread <- ts(SPA - GER)

RAUT_spread = diff(AUT_spread, lag = 1)
RSPA_spread = diff(SPA_spread, lag = 1)

df = cbind(AUT_spread, SPA_spread)
return_df = cbind(RAUT_spread, RSPA_spread)

plot(ts(df))
```




```
plot(ts(return_df))
```



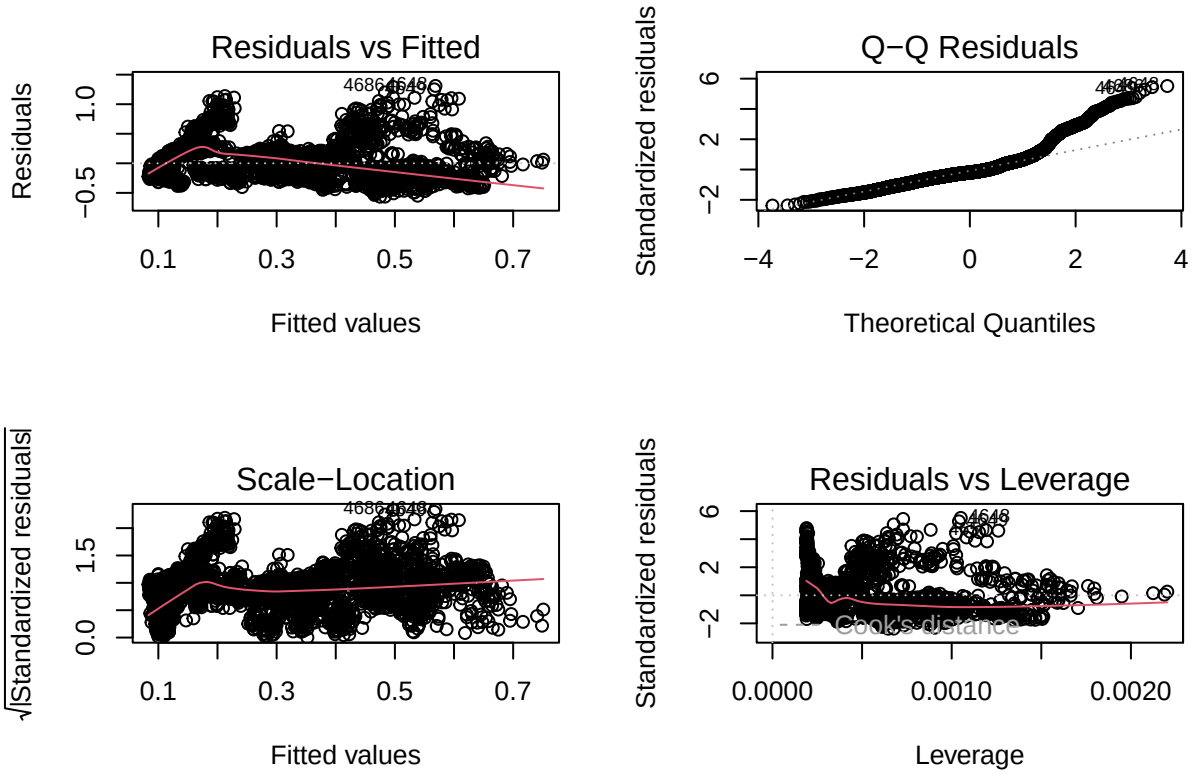
1°/ Markov-Switching Model :

1.1°/ Linear benchmark model :

```
# Start with 1 lag
lin_mod <- lm(AUT_spread ~ stats::lag(SPA_spread, 1))
summary(lin_mod)
```

```
##
## Call:
## lm(formula = AUT_spread ~ stats::lag(SPA_spread, 1))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56491 -0.12994 -0.04681  0.08923  1.30935
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 0.097854   0.004142   23.62   <2e-16 ***
## stats::lag(SPA_spread, 1) 0.104158   0.002107   49.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2376 on 5291 degrees of freedom
## Multiple R-squared:  0.316,  Adjusted R-squared:  0.3158
## F-statistic:  2444 on 1 and 5291 DF,  p-value: < 2.2e-16
```

```r
par(mfrow=c(2,2))
plot(lin_mod)
```




```r
# Autocorrelation
bgtest(lin_mod, order = 4)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 4
##
## data:  lin_mod
## LM test = 5123.9, df = 4, p-value < 2.2e-16
```

```r
# ARCH effects
ArchTest(residuals(lin_mod), lags = 5)
```

```
##
##  ARCH LM-test; Null hypothesis: no ARCH effects
##
## data:  residuals(lin_mod)
## Chi-squared = 4987, df = 5, p-value < 2.2e-16
```

```r
# Normality
jarque.bera.test(residuals(lin_mod))
```

```
##
##  Jarque Bera Test
##
## data:  residuals(lin_mod)
```

```
## X-squared = 5701.1, df = 2, p-value < 2.2e-16
```

1.2°/ Markov-Switching regression (2 regimes) :

```
AICs <- c()

for (p in 1:4) {
  mod <- lm(AUT_spread ~ stats::lag(SPA_spread, -p))
  AICs[p] <- AIC(mod)
}

which.min(AICs)
```

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

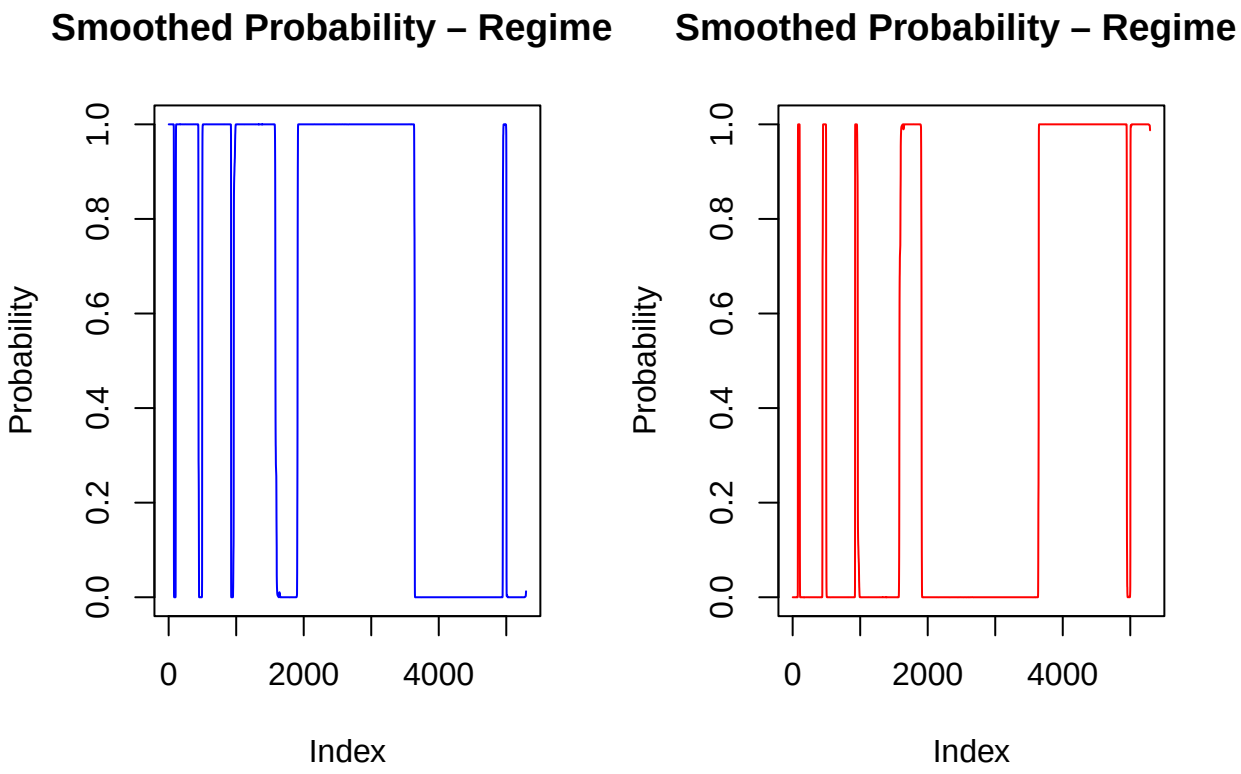
```
# Manually align the data
SPA_spread_lag <- SPA_spread[-length(SPA_spread)]  # Dropping the last element
AUT_spread_aligned <- AUT_spread[-1]  # Dropping the first element

# Fit the Markov switching model with the aligned variables
ms_mod <- msmFit(
  lm(AUT_spread_aligned ~ SPA_spread_lag),
  k = 2,
  control = list(parallel = FALSE),
  sw = c(TRUE, TRUE, TRUE)  # intercept, slope, variance switch
)

summary(ms_mod)
```

```
## Markov Switching Model
##
## Call: msmFit(object = lm(AUT_spread_aligned ~ SPA_spread_lag), k = 2,
##     sw = c(TRUE, TRUE, TRUE), control = list(parallel = FALSE))
##
##        AIC       BIC   logLik
##   -7350.03 -7289.439 3679.015
##
## Coefficients:
##
## Regime 1
## ---------
##                    Estimate Std. Error t value  Pr(>|t|)
## (Intercept)(S)       0.0195     0.0017  11.471 < 2.2e-16 ***
## SPA_spread_lag(S)    0.0519     0.0010  51.900 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07334947
## Multiple R-squared: 0.5175
##
## Standardized Residuals:
##           Min            Q1           Med            Q3           Max
## -3.047828e-01 -9.990689e-03  4.429905e-05  2.663979e-02  2.520019e-01
##
## Regime 2
```

```
## ---------
##                   Estimate Std. Error t value  Pr(>|t|)
## (Intercept)(S)      0.2987     0.0100  29.870 < 2.2e-16 ***
## SPA_spread_lag(S)   0.1008     0.0041  24.585 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2588158
## Multiple R-squared: 0.2575
##
## Standardized Residuals:
##           Min            Q1           Med            Q3           Max
## -0.5900182974 -0.0279259167 -0.0003493027 -0.0002888717  1.1580784146
##
## Transition probabilities:
##             Regime 1    Regime 2
## Regime 1 0.998446945 0.002948282
## Regime 2 0.001553055 0.997051718
```

1.3°/ Posterior regim probabilities :

```r
par(mfrow = c(1, 2))

plot(ms_mod@Fit@smoProb[,1],
     type = "l", col = "blue",
     main = "Smoothed Probability - Regime 1",
     ylab = "Probability")

plot(ms_mod@Fit@smoProb[,2],
     type = "l", col = "red",
     main = "Smoothed Probability - Regime 2",
     ylab = "Probability")
```



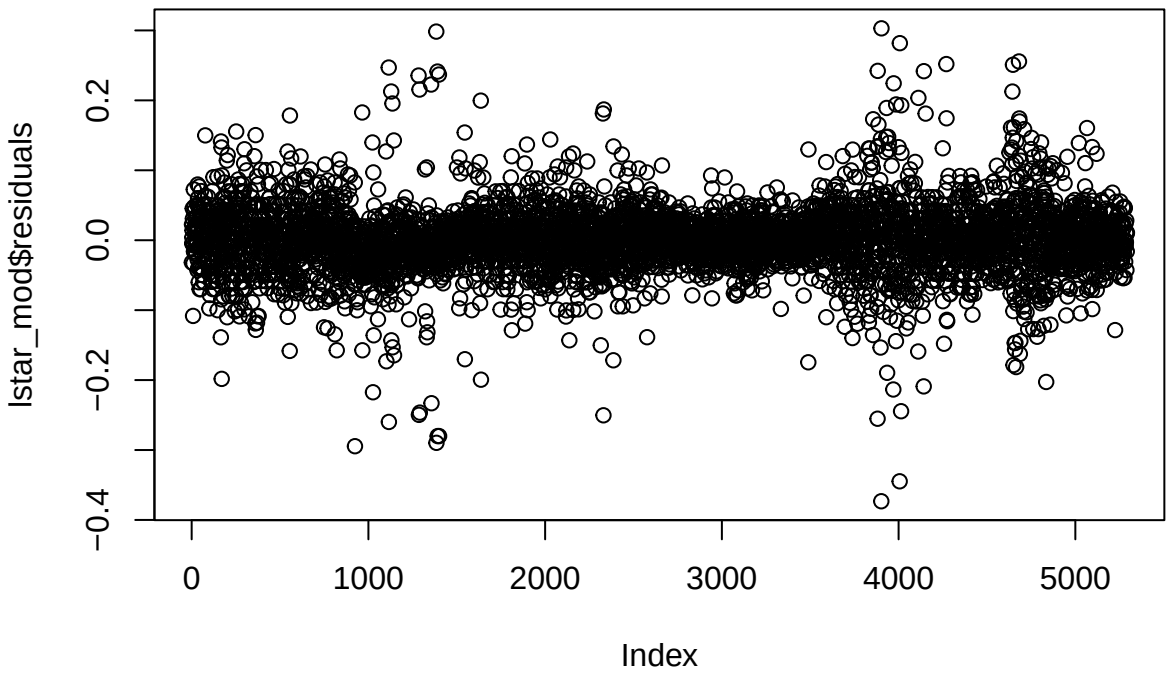
2°/ LSTAR Model :

2.1 LSTAR model :

```
lstar_mod <- tsDyn::lstar(
  AUT_spread,
  m = 1,
  d = 1,
  control = list(trace = TRUE)
)
```

```
## Using maximum autoregressive order for low regime: mL = 1
## Using maximum autoregressive order for high regime: mH = 1
## Using default threshold variable: thDelay=0
## Performing grid search for starting values...
## Starting values fixed: gamma =  100 , th =  0.1561482 ; SSE =  11.86673
## Grid search selected lower/upper bound gamma (was:  1 100 ]).
##                     Might try to widen bound with arg: 'starting.control=list(gammaInt=c(1,200))'
## initial  value 11.866731
## final  value 11.866697
## converged
## Optimization algorithm converged
## Optimized values fixed for regime 2  : gamma =  100 , th =  0.1574295 ; SSE =  11.8667
```

```
plot(lstar_mod$residuals)
```




```
summary(lstar_mod)
```

```
##
## Non linear autoregressive model
##
## LSTAR model
## Coefficients:
## Low regime:
##    const.L     phiL.1
## 0.00595829 0.85747303
##
## High regime:
##      const.H       phiH.1
## -0.003569786  0.132596304
##
## Smoothing parameter: gamma = 100
##
## Threshold
## Variable: Z(t) = + (1) X(t)
##
## Value: 0.1574
##
## Residuals:
##         Min          1Q      Median          3Q         Max
## -0.37325229 -0.02301121  0.00017726  0.02138515  0.30297406
##
```

```
## Fit:
## residuals variance = 0.002242,  AIC = -32277, MAPE = 41.58%
##
## Coefficient(s):
##            Estimate  Std. Error  t value  Pr(>|z|)
## const.L  5.9583e-03  9.5584e-04   6.2336 4.559e-10 ***
## phiL.1   8.5747e-01  1.3636e-02  62.8825 < 2.2e-16 ***
## const.H -3.5698e-03  2.2166e-03  -1.6105    0.1073
## phiH.1   1.3260e-01  1.4075e-02   9.4207 < 2.2e-16 ***
## gamma    1.0000e+02  7.3257e+01   1.3650    0.1722
## th       1.5743e-01  1.0528e-02  14.9532 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Non-linearity test of full-order LSTAR model against full-order AR model
##  F = 9.0878 ; p-value = 0.0025854
##
## Threshold
## Variable: Z(t) = + (1) X(t)
```

2.2°/ Non-Linearity test (Tsay) :

```
resid <- lstar_mod$residuals
nonlinearityTest(resid, verbose = TRUE)
```

```
##       ** Teraesvirta's neural network test  **
##       Null hypothesis: Linearity in "mean"
##       X-squared =  11.13024  df =  2  p-value =  0.003829126
##
##       ** White neural network test  **
##       Null hypothesis: Linearity in "mean"
##       X-squared =  5.527845  df =  2  p-value =  0.06304399
##
##       ** Keenan's one-degree test for nonlinearity  **
##       Null hypothesis: The time series follows some AR process
##       F-stat =  0.06242075  p-value =  0.8027198
##
##       ** McLeod-Li test  **
##       Null hypothesis: The time series follows some ARIMA process
##       Maximum p-value =  0
##
##       ** Tsay's Test for nonlinearity **
##       Null hypothesis: The time series follows some AR process
##       F-stat =  2.38564  p-value =  1.211517e-63
##
##       ** Likelihood ratio test for threshold nonlinearity **
##       Null hypothesis: The time series follows some AR process
##       Alternativce hypothesis: The time series follows some TAR process
##       X-squared =  169.6182  p-value =  2.220446e-16
```

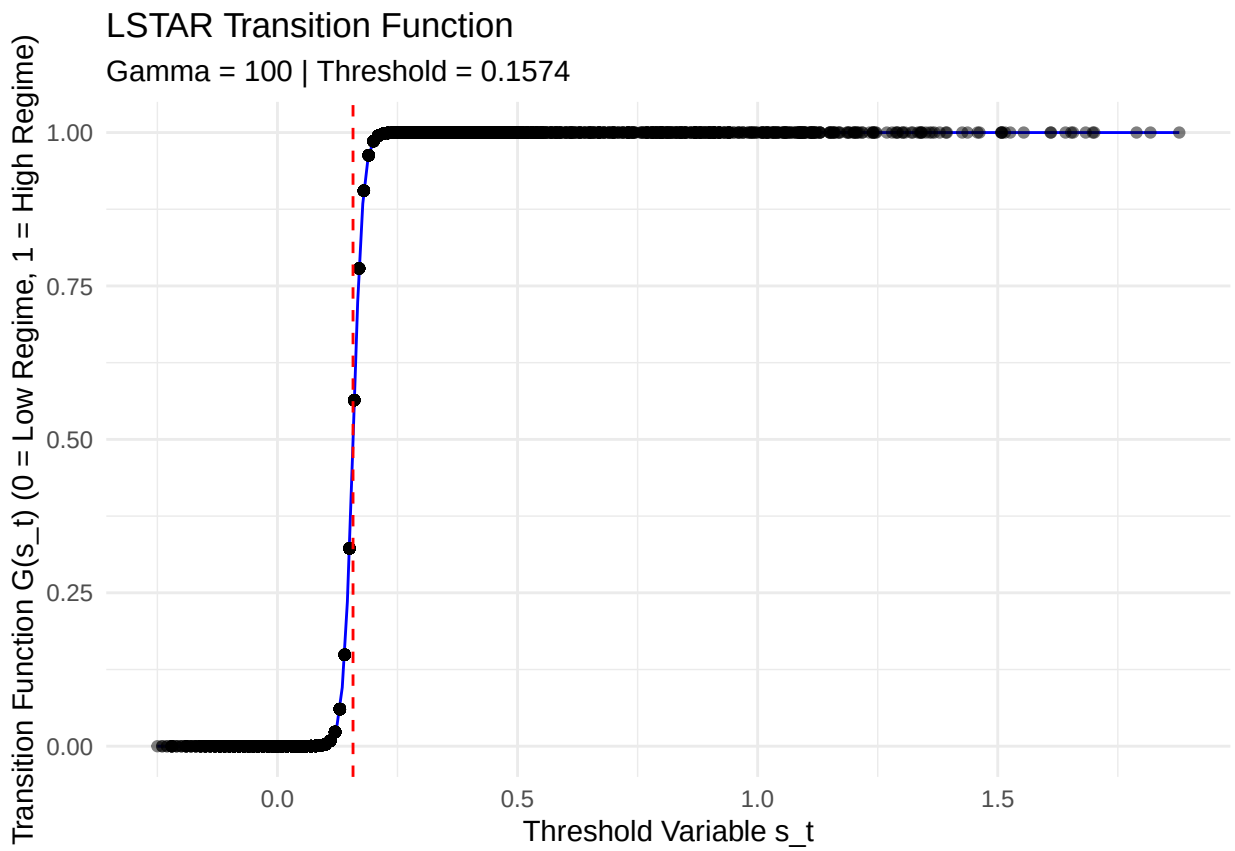
```
## $Terasvirta
##
##  Teraesvirta Neural Network Test
##
```

```
## data:  ts(time.series)
## X-squared = 11.13, df = 2, p-value = 0.003829
##
##
## $White
##
##  White Neural Network Test
##
## data:  ts(time.series)
## X-squared = 5.5278, df = 2, p-value = 0.06304
##
##
## $Keenan
## $Keenan$test.stat
## [1] 0.06242075
##
## $Keenan$df1
## [1] 1
##
## $Keenan$df2
## [1] 5216
##
## $Keenan$p.value
## [1] 0.8027198
##
## $Keenan$order
## [1] 37
##
##
## $McLeodLi
## $McLeodLi$p.values
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##
##
## $Tsay
## $Tsay$test.stat
## [1] 2.38564
##
## $Tsay$p.value
## [1] 1.211517e-63
##
## $Tsay$order
## [1] 37
##
##
## $TarTest
## $TarTest$percentiles
## [1] 24.98573 75.01427
##
## $TarTest$test.statistic
## [1] 169.6182
##
## $TarTest$p.value
## [1] 2.220446e-16
```

2.3°/ Transition function :

```
library(ggplot2)

# 1. Extract the coefficients Gamma (smoothness) and Threshold (c)
gamma_val <- coef(lstar_mod)["gamma"]
th_val    <- coef(lstar_mod)["th"]

# 2. Extract the threshold variable used in the model
threshold_var <- lstar_mod$model.specific$thVar
threshold_var_clean <- threshold_var[is.finite(threshold_var)]

# 3. Create a function to calculate the weights based on the LSTAR formula
calculate_weight <- function(s, gamma, c) {
  1 / (1 + exp(-gamma * (s - c)))
}

# 4. Generate a smooth curve for plotting
s_grid <- seq(
  min(threshold_var_clean),
  max(threshold_var_clean),
  length.out = 200
)
#s_grid <- seq(min(threshold_var), max(threshold_var), length.out = 200)
weights_grid <- calculate_weight(s_grid, gamma_val, th_val)

# 5. Calculate actual weights for your data points
actual_weights <- calculate_weight(threshold_var, gamma_val, th_val)

# 6. Plot
plot_data <- data.frame(Threshold_Variable = s_grid, Transition_Weight = weights_grid)
point_data <- data.frame(Threshold_Variable = threshold_var, Transition_Weight = actual_weights)

ggplot() +
  # The Sigmoid Curve (Theoretical transition)
  geom_line(data = plot_data, aes(x = Threshold_Variable, y = Transition_Weight),
            color = "blue") +
  # The Actual Observations (Where your data lies on the curve)
  geom_point(data = point_data, aes(x = Threshold_Variable, y = Transition_Weight),
             alpha = 0.5) +
  # Vertical line for the threshold value
  geom_vline(xintercept = th_val, linetype = "dashed", color = "red") +
  theme_minimal() +
  labs(title = "LSTAR Transition Function",
       subtitle = paste("Gamma =", round(gamma_val, 2), "| Threshold =", round(th_val, 4)),
       y = "Transition Function G(s_t) (0 = Low Regime, 1 = High Regime)",
       x = "Threshold Variable s_t")
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



3°/ Bivariate GARCH :

3.1°/ Specify univariate GARCH :

```
spec_uni <- ugarchspec(
  variance.model = list(model = "sGARCH"),
  mean.model = list(armaOrder = c(1,0)),
  distribution.model = "norm"
)
```

3.2°/ GARCH specification :

```
spec_dcc <- dccspec(
  uspec = multispec(replicate(2, spec_uni)),
  dccOrder = c(1,1),
  distribution = "mvnorm"
)

#data_mat <- cbind(AUT_spread, SPA_spread)

dcc_fit <- dccfit(spec_dcc, data = df)
summary(dcc_fit)
```

```
## Length  Class   Mode
##      1 DCCfit     S4
```

3.3 Dynamic correlation plot :

```
dcc_cor <- rcor(dcc_fit)
plot(dcc_cor[1,2,],
     type = "l",
     main = "Dynamic Correlation: Austria-Spain",
     ylab = "Correlation")
```