Descripción de las Funciones de la librería KDSeries. Versión: 0.8

A continuación se muestran los algoritmos desarrollados que se están implementando en la librería para R denominada “KDSeries”.

I.1. Función “kdplotseries”: Dibuja series temporales

Dibuja una matriz de series temporales en varias partes.

Llamada a la función:
kdplotseries(Series, Num=4,Ini=1,End=dim(Series)[1], Colors=1:dim(Series)[2])

Parámetros de Entrada:

- *Series*=Matriz compuesta por series temporales (cada columna es una ST)
- *Num*=Número de divisiones horizontales de la figura.
- *Ini*=Primer Punto.
- *End*=Ultimo Punto.
- *Colors*=Lista de Colores para cada ST.

Parámetros de Salida: Ninguno

Figura 8. Ejemplo de aplicación de la función “kdplotseries”.

Programa: kdfilter.R

```
#########################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#########################################################################
# (kdplotseries.R) Plot Time Series In X Parts with Differents Filters
#########################################################################

# INPUT PARAMETERS:
# Series=Matrix with Time Series (Each Column is one Time Serie)
# Num=Num of Horizontal plots
# Ini=First point
# End=Last point
# Colors=Color for each Time Series
# ------------------------------------------------------------------------
------------------------
# OUTPUT PARAMETERS:
# Returns: 1. None
# ------------------------------------------------------------------------
------------------------
kdplotseries <- function(Series, Num=4,Ini=1,End=dim(Series)[1],
Colors=1:dim(Series)[2])
{
      # mar=margin (Bot,Left,Top,Right), cex.axis=text_axis,
      # lab=div axis y title size
      # mgp=Axis Text Dist (title, labels, line)

      par(bg="white", mar=c(1,1,1.5,0.5), lab=c(10,5,6),
mgp=c(1,0.2,0), mfrow=c(Num,1))
      Long=(End-Ini+1)/Num
      for (h in 1:Num)
            {
            Etiq=paste("Series Part:",h,sep="")

            plot(Series[(1+Long*(h-1)):(Long*h),1],main=Etiq, type="l",
ylab="f(t)",col=Colors[1])
            for (j in 2:dim(Series)[2])
                  {
                  lines(Series[(1+Long*(h-
1)):(Long*h),j],col=Colors[j])
                  }
            }
}
```

I.2. Función “kdfilter”: Filtra una ST

Filtra una serie temporal con filtros kernel.

Llamada a la función:
kdfilter(TSerie, WidthW, Filter="gauss")

Parámetros de Entrada:

- *TSerie*=Serie Temporal a Filtrar.
- *WidthW*=Tamaño de la ventana del filtro deslizante.
- *Filter*=Filtro a aplicar. “gauss”=gaussiano, “mean” o “rect”=media, “median”=mediana, “max”=máximo, “min”=mínimo.

Parámetros de Salida: Serie Filtrada.

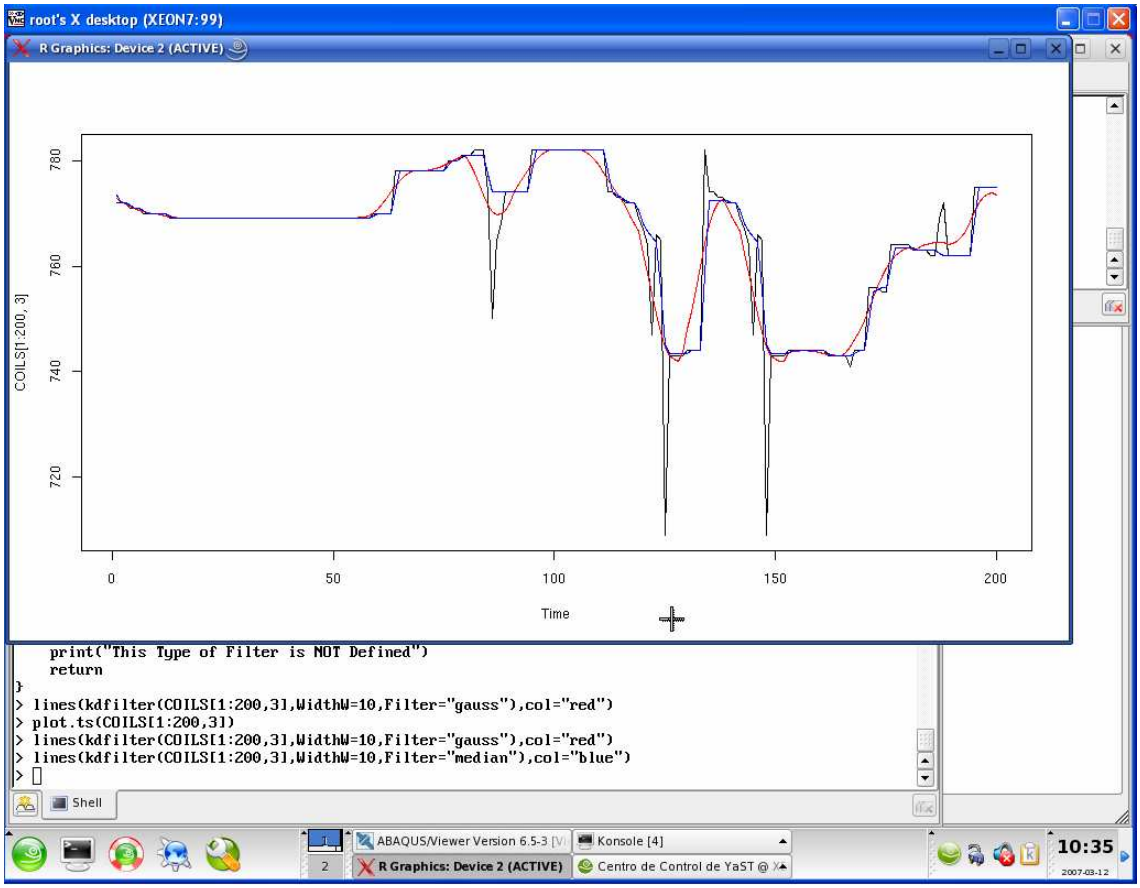


Figura 9. Ejemplo de aplicación de la función “kdfilter”. En este caso, se aplica un filtro Gaussiano (rojo) y otro de mediana (azul) a una ST (negro).

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdfilter.R)Filter a Time Series
#####################################################################

# INPUT PARAMETERS:
# TSerie=Time Series
# WidthW=Filter's Window Width
# Filter=Type of Filter.      "mean"=rectangular window, "gauss"=gauss
window, "median"=median from window
#                                          "max"=select max point from
window, "min"=select min point from window
# ------------------------------------------------------------------
------------------------

# OUTPUT PARAMETERS:
# SerieFilt=Time Series Filtered
# ------------------------------------------------------------------
------------------------

kdfilter <- function (TSerie, WidthW, Filter="gauss")
{
storage.mode(TSerie) <- "double"
if (Filter=="rect" || Filter=="mean")
      {
      SerieFilt <-
.Call("meanfilter",TSerie,WidthW,PACKAGE="KDSeries")
      return(SerieFilt)
      }

if (Filter=="gauss")
      {
      t <- 1: WidthW
      FiltW <- (1/(sqrt(2*pi)*sd(t)))*exp((-(t-
mean(t))^2)/(2*sd(t)^2))
      FiltW <- FiltW/sum(FiltW)
      SerieFilt <- filter(TSerie, FiltW, sides=2, circular=TRUE)
      return(SerieFilt)
      }

if (Filter=="median")
      {
      SerieFilt <- .Call("medianfilter",TSerie,WidthW,
PACKAGE="KDSeries")
      return(SerieFilt)
      }
```

```
if (Filter=="max")
      {
      SerieFilt <- .Call("maxfilter",TSerie,WidthW,
PACKAGE="KDSeries")
      return(SerieFilt)
      }

if (Filter=="min")
      {
      SerieFilt <- .Call("minfilter",Serie,WidthW, PACKAGE="KDSeries")
      return(SerieFilt)
      }
print("This Type of Filter is NOT Defined");
return

}
```

I.3. Función “kdmatfilter”: Crea una matriz con diferentes filtrados de una ST

Crea una matriz resultado de realizar un filtrado a una ST con diferentes anchos de ventana.

Llamada a la función:
kdmatfilter(TSerie, WidthWVect, Filter="gauss")

Parámetros de Entrada:

- *TSerie*=Serie Temporal a Filtrar.
- *WidthWVect*=Vector con diferentes tamaños de ventanas.
- *Filter*=Filtro a aplicar. “gauss”=gaussiano, “mean” o “rect”=media, “median”=mediana, “max”=máximo, “min”=mínimo.

Parámetros de Salida: Una Matriz con el resultado de diferentes tamaños del filtros.

Programa: kdmatfilter.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdmatfilter.R) # Obtain a Matrix with Different Filters
#####################################################################

# INPUT PARAMETERS:
# TSerie=Time Serie
# WidthWVect=Vector with differents filter's windows width
# Filter=Type of Filter.      "mean"=rectangular window, "gauss"=gauss
window, "median"=median from window
#                                          "max"=select max point from
window, "min"=select min point from window
# ------------------------------------------------------------------

# OUTPUT PARAMETERS:
# Returns: 1. Matrix with Different Filters (each file is a time serie
filtered)
# ------------------------------------------------------------------

kdmatfilter <- function (TSerie, WidthWVect, Filter="gauss")
{
      storage.mode(WidthWVect) <- "double"
```

```
      # Results MAtrix
      storage.mode(TSerie) <- "double"

      MFilt <- matrix(0,length(WidthWVect),length(TSerie))
      storage.mode(MFilt) <- "double"

      # ZeroCrossings Matrix

      CrossZ <- matrix(0,length(WidthWVect),length(TSerie))
      storage.mode(CrossZ) <- "integer"

      # Filter Window's Width Vector
      WinW <- WidthWVect

      # Vector of ZeroCrossings  Summatory
      NumCZ <- rep(1,length(WidthWVect))

      j=0
      for (h in WidthWVect)
            {
            j=j+1
            ifelse (h==1, MFilt[j,] <- TSerie, MFilt[j,] <-
kdfilter(TSerie,h,Filter))
            CrossZ[j,] <- .Call("zerocrossings",MFilt[j,],
PACKAGE="KDSeries")
            NumCZ[j] <- sum(CrossZ[j,])
            }
      MAT <- list(TSerie=TSerie,MFilt=MFilt,WinW=WinW,CrossZ=CrossZ,
NumCZ=NumCZ)
      return(MAT)
}
```

I.4. Función “kdplotmat”: Visualiza una matriz con diferentes filtrados de una ST

Visualiza la matriz resultado de realizar un filtrado a una ST con “*kdmatfilter*”.

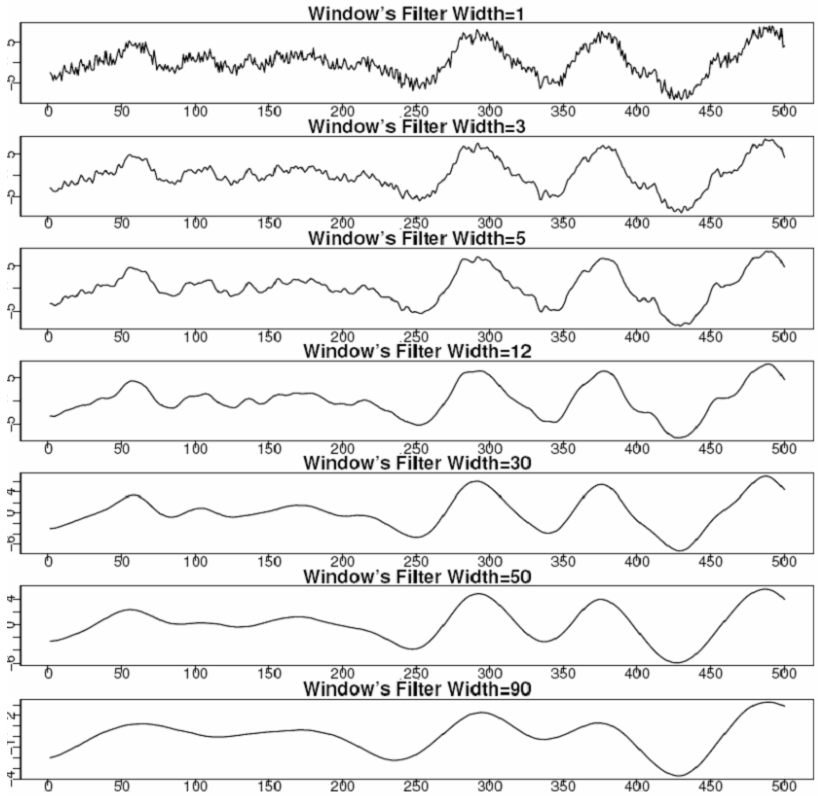


Figura 10. Filtros realizados con ventana "campana de gauss" de diferentes anchuras.

Llamada a la función:
kdplotmat (MAT, Positions=1:dim(MAT$MFilt)[1], Ini=1,End=dim(MAT$MFilt)[2])

Parámetros de Entrada:

- *MAT*=Matriz con diferentes filtros de una ST (viene de kdmatfilter).
- *Positions*=Indica cuales son las series filtradas que queremos visualizar.
- *Filter*=Filtro a aplicar. “gauss”=gaussiano, “mean” o “rect”=media, “median”=mediana, “max”=máximo, “min”=mínimo.
- *Ini*=Primer Punto a dibujar.
- *End*=Ultimo Punto a dibujar.

Parámetros de Salida: Ninguno.

```
#########################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#########################################################################
# (kdplotmat.R) Plot Matrix with Differents Filters
#########################################################################

# INPUT PARAMETERS:
# MAT=Matrix with Differents Filters(from kdmatfilter())
# Positions=Plot filter placed in Positions
# Ini=First point
# End=Last point
# ------------------------------------------------------------------

# OUTPUT PARAMETERS:
# Returns: 1. None
# ------------------------------------------------------------------

kdplotmat <- function(MAT,
Positions=1:dim(MAT$MFilt)[1],Ini=1,End=dim(MAT$MFilt)[2])
{
      LL <- length(Positions)

      # mar=margin (Bot,Left,Top,Right), cex.axis=text_axis,
      # lab=div axis y title size
      # mgp=Axis Text Dist (title, labels, line)

      par(bg="white", mar=c(1,1,1.5,0.5), lab=c(10,5,6),
mgp=c(1,0.2,0), mfrow=c(LL,1))
      for (h in 1:LL)
            {
            Etiq=paste("Window's Filter
Width=",MAT$WinW[Positions[h]],sep="")
            plot(MAT$MFilt[Positions[h], Ini:End],main=Etiq, type="l",
ylab="f(t)")
            }
}
```

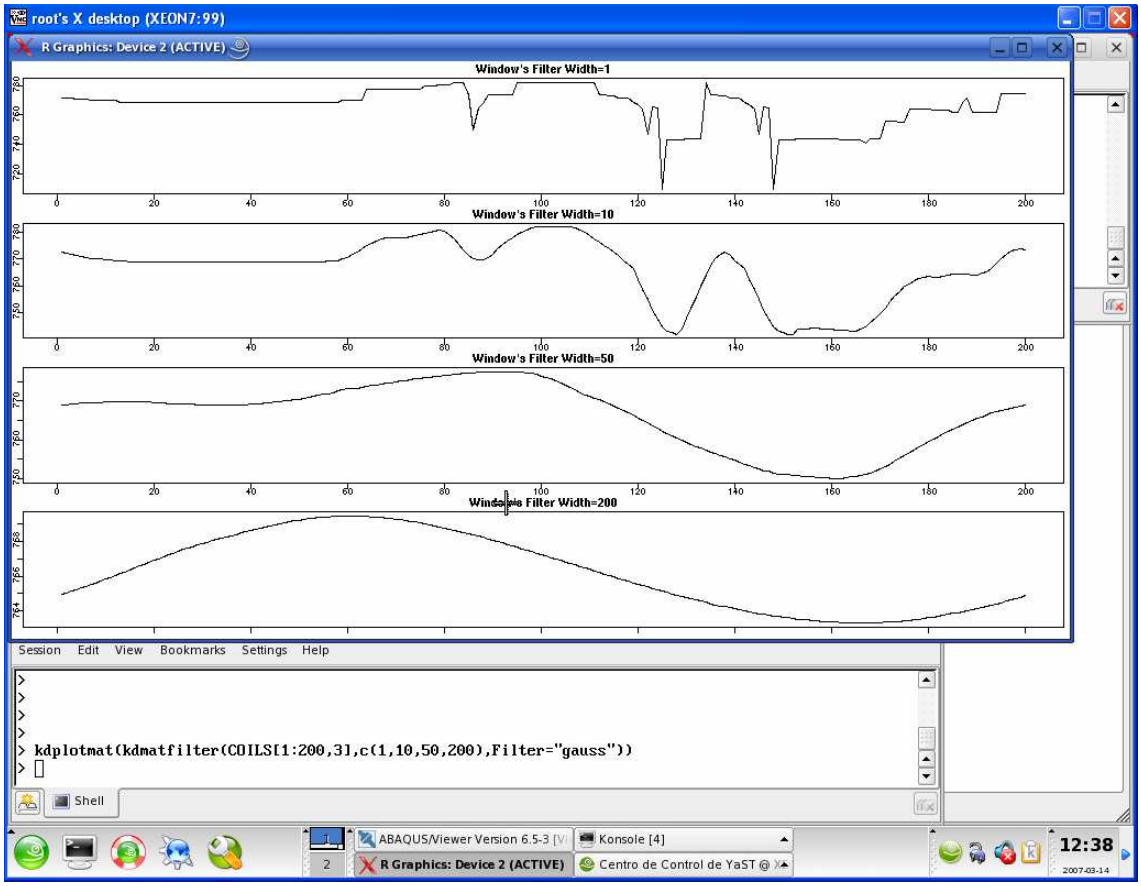


Figura 11. Ejemplo de aplicación de las funciones "kdplotmat" y "kdmatfilter". En este caso se visualiza el resultado de aplicar filtros gaussianos de anchura 1, 10, 50 y 200 a una ST.

I.5. Función “kdplotzcross”: Identifica los cruces por cero de la primera derivada (máximos y mínimos de la función)

Visualiza dónde se producen los cruces por cero para identificar los máximos y mínimos de la función.

Llamada a la función:
kdplotzcross(TSerie)

Parámetros de Entrada:

- *TSerie*=Serie temporal.

Parámetros de Salida: Ninguno.

Programa: kdplotzcross.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdplotzcross.R) Plot Time Series and ZeroCrossings
#####################################################################

# INPUT PARAMETERS:
# TSerie=Time Series
# ------------------------------------------------------------------
-----------------------
# OUTPUT PARAMETERS:
# Returns: 1. None
# ------------------------------------------------------------------

kdplotzcross <- function (TSerie)
{
      plot.ts(TSerie,col="blue")
      storage.mode(TSerie) <- "double"
      ZCross <- rep("-",length(TSerie))
      if (length(TSerie)>1)
ZCross[.Call("zerocrossings",TSerie,PACKAGE="KDSeries")==1] <- "+"
      points(TSerie,pch=ZCross,cex=0.8)
}
```

Figura 12. Ejemplo de aplicación de las funciones "kdplotzcross". En este caso se visualizan los cruces por cero de la primera derivada (máximos y mínimos) de una ST.

I.6. Función “kdplotnumz”: Muestra la evolución del número de cruces por cero para diferentes tipos de anchos de filtros

Muestra en una gráfica la evolución del número de cruces por cero para una matriz con una serie temporal filtrada con diferentes anchos. Esto permite, determinar el mejor ancho de ventana de filtrado.

Llamada a la función:
kdplotnumz (MAT, Positions=1:dim(MAT$MFilt)[1])

Parámetros de Entrada:

- *MAT*=Matriz con diferentes filtros de una ST (viene de *kdmatfilter*()).
- *Positions*=Indica cuales son las series filtradas que queremos visualizar.

Parámetros de Salida: Ninguno.

Programa: kdplotnumz.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdplotnumz.R) Plot Number ZerosCrossing Curve vs Filter's Windows
Width
#####################################################################

# INPUT PARAMETERS:
# MAT=Matrix with Differents Filters(from kdmatfilter())
# Positions=Plot filter placed in Positions
# ------------------------------------------------------------------
------------------------

# OUTPUT PARAMETERS:
# Returns: 1. None
# ------------------------------------------------------------------




kdplotnumz <- function(MAT, Positions=1:dim(MAT$MFilt)[1])
{
      LL <- length(Positions)
      plot.ts(MAT$NumCZ[Positions],xlab="Width",ylab="Num. Zero-
Crossings",type="b",col="blue",axes=FALSE)
```

```
      axis(1,1:LL,MAT$WinW[Positions])
      axis(2,round(seq(min(MAT$NumCZ[Positions]),max(MAT$NumCZ[Positio
ns]),length=LL)))
}
```

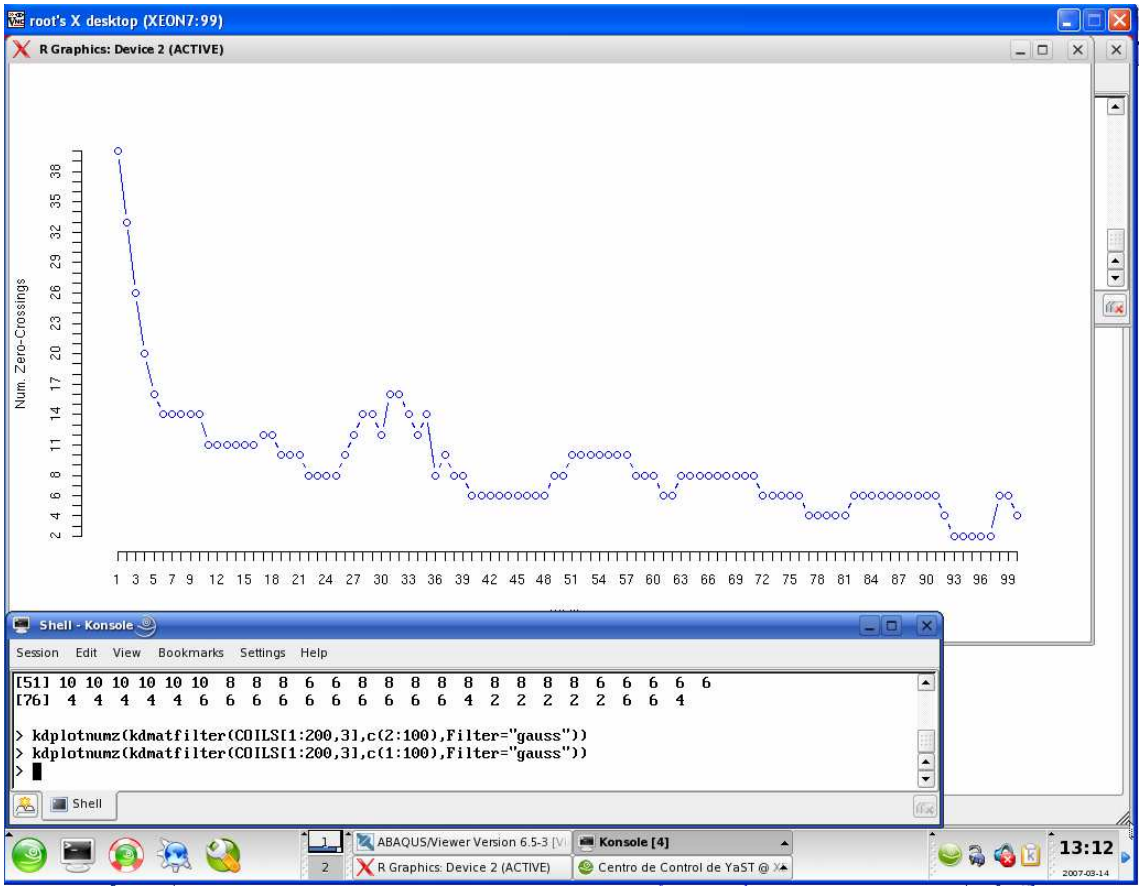


Figura 13. Ejemplo de aplicación de las funciones "kdplotnumz". En este caso, se visualiza el número de cruces por cero de la segunda derivada (número de máximos y mínimos) obtenidos en 100 filtros Gaussianos de ventanas 1 a 100 de ancho.

I.7. Función “kdplotscales”: Muestra una gráfico donde se identifican los máximos y mínimos resultantes de cada filtro con anchuras de ventanas diferentes

Muestra en una gráfica donde están situados los máximos y mínimos de una serie temporal filtrada con diferentes anchos. Esto permite, determinar los rangos mejores de anchos de ventana de filtrado que distorsionen lo menos posible la ST pero que reduzcan el número de máximos y mínimos.

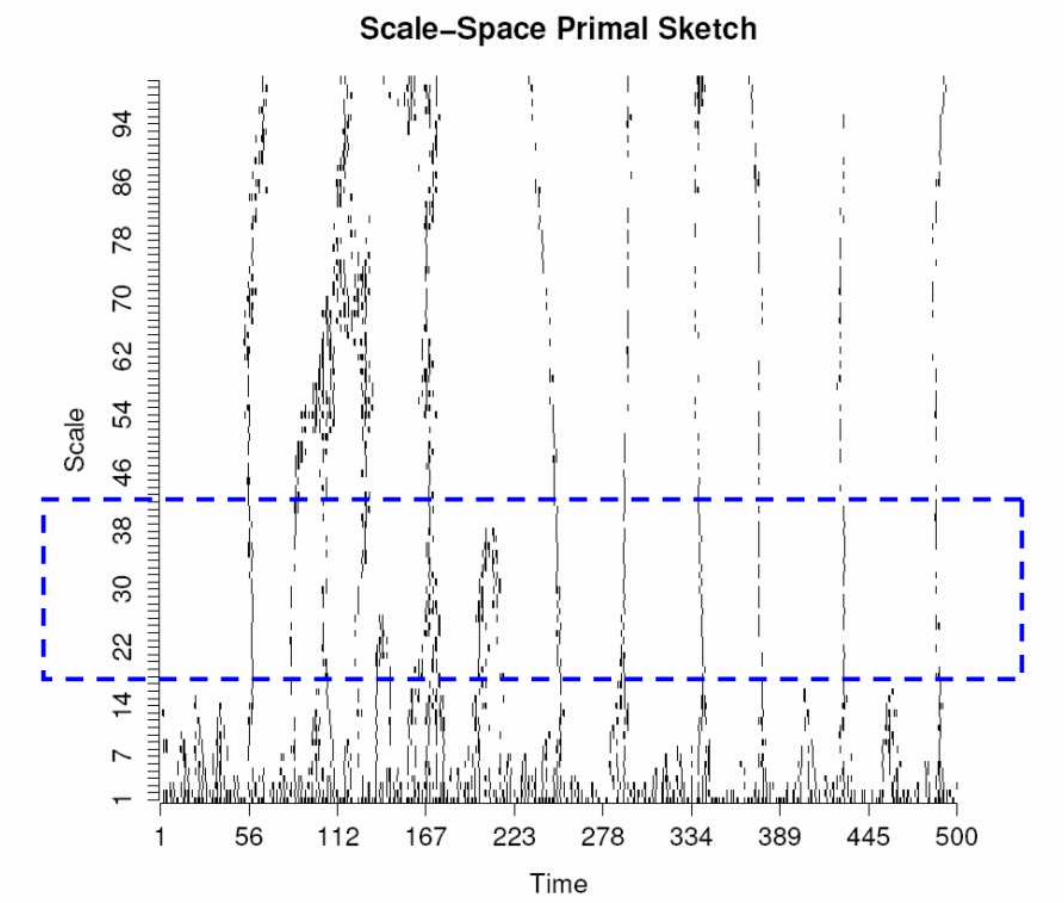


Figura 14. Diagrama de escalas donde se puede observar la posición de los cruces por cero para cada ventana de filtrado.

Llamada a la función:
kdplotscales(MAT, Positions=1:dim(MAT$MFilt)[1],Ini=1,End=dim(MAT$MFilt)[2])

Parámetros de Entrada:

- *MAT*=Matriz con diferentes filtros de una ST (viene de *kdmatfilter*()).
- *Positions*=Indica cuales son las series filtradas que queremos visualizar.
- *Ini*=Punto inicial.
- *End*=Punto final.

Parámetros de Salida: Ninguno.

Programa: kdplotscales.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
```

```
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#########################################################################
# (kdplotscales.R) Plot Tree_Scales
#########################################################################

# INPUT PARAMETERS:
# MAT=Matrix with Differents Filters(from kdmatfilter())
# Positions=Plot filter placed in Positions
# Ini=First point
# End=Last point
# ------------------------------------------------------------------
-----------------------

# OUTPUT PARAMETERS:
# Returns: 1. None
# ------------------------------------------------------------------
kdplotscales <- function(MAT,
Positions=1:dim(MAT$MFilt)[1],Ini=1,End=dim(MAT$MFilt)[2])
{
      image(t(MAT$CrossZ[Positions,Ini:End]),ylab="Scale",xlab="Time",
axes=FALSE,main="Scale-Space Primal Sketch",col=c("White","Black"))
      LL <- length(Positions)

      axis(2,(0:(LL-1))/(LL-1),MAT$WinW[Positions])
      axis(1,seq(0,1,length=10),round(seq(Ini,End,length=10)))
}
}
```

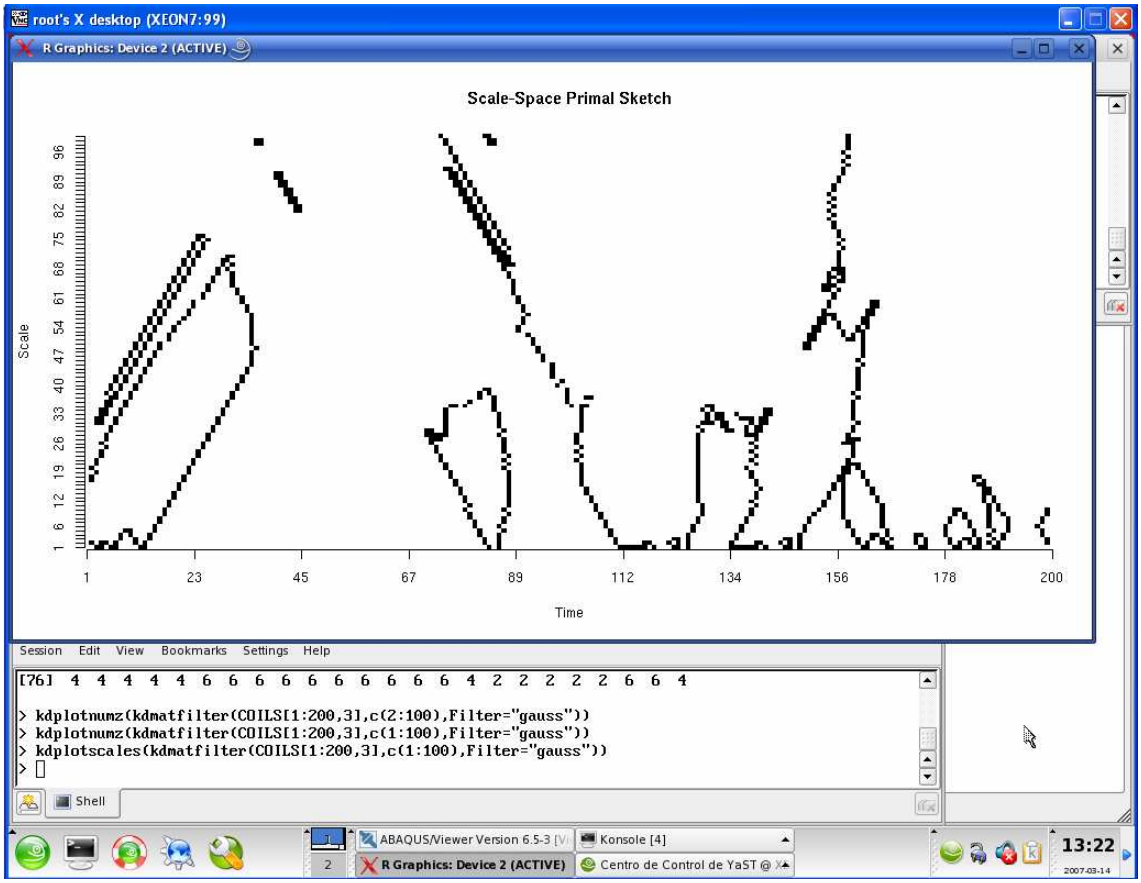


Figura 15. Ejemplo de aplicación de las funciones "kdplotscales". En este caso, se desarrolla un gráfico donde en el eje Y se identifica el ancho de cada filtro y en el eje X la posición del máximo o mínimo (punto negro).

I.8. Función “kdextract”: Extrae segmentos de una serie temporal a partir de una serie de umbrales

Permite la extracción de segmentos de series temporales según diversos criterios establecidos y una serie de umbrales configurables por el usuario.

Llamada a la función:
kdextract(TSerie, Param=c(rep(TRUE,13)),
Threshold=c(0.05,0.05,0.05,0.40,0.10,0.40,0.10,0.8,0.2,0.05), LongMin=rep(0,13))

Donde:

Parámetros de Entrada:

- *TSerie*=Serie temporal (se recomienda filtrarla previamente).
- *Param*=
  - Param[1]=TRUE (Obtiene segmentos incrementales) INCREMENTAL
  - Param[2]=TRUE (Obtiene segmentos decrementales) DECREMENTAL
  - Param[3]=TRUE (Obtiene segmentos horizontales) HORIZONTAL
  - Param[4]=TRUE (Obtiene segmentos incrementales por encima de un umbral positivo) INCHIGH
  - Param[5]=TRUE (Obtiene segmentos incrementales por debajo de un umbral positivo) INCLOW
  - Param[6]=TRUE (Obtiene segmentos decrementales por encima de un umbral positivo) DECHIGH
  - Param[7]=TRUE (Obtiene segmentos decrementales por debajo de un umbral positivo) DECLOW
  - Param[8]=TRUE (Obtiene segmentos por encima de un umbral positivo) VALHIGH
  - Param[9]=TRUE (Obtiene segmentos por debajo de un umbral positivo) VALLOW
  - Param[10]=TRUE (Obtiene segmentos por cerca de cero) VALZERO
  - Param[11]=TRUE (Obtiene segmentos entre VALHIGH y VALLOW) VALMED
  - Param[12]=TRUE (Obtiene segmentos entre INCHIGH y INCLOW) INCMED
  - Param[13]=TRUE (Obtiene segmentos entre DECHIGH y DECLOW DECMED
- *Threshold*=Umbrales de corte.
  - Threshold[1]=% del rango(TSerie)=si es INC y ESTÁ POR ENCIMA DE ESE VALOR este segmento es INCREMENTAL
  - Threshold[2]=% del rango(TSerie)=si DEC y ESTÁ POR ENCIMA DE ESE VALOR este segmento es DECREMENTAL
  - Threshold[3]=% del rango(TSerie)=si INC o DEC sino ESTÁ POR DEBAJO DE ESE VALOR este segmento es HORIZONAL
  - Threshold[4]=% del rango(TSerie)=si INC y ESTÁ POR ENCIMA DE ESE VALOR este segmento es INCHIGH
  - Threshold[5]=% del rango(TSerie)=si INC y ESTÁ POR DEBAJO DE ESE VALOR este segmento es INCLOW
  - Threshold[6]=% del rango(TSerie)=si DEC y ESTÁ POR ENCIMA DE ESE VALOR este segmento es DECHIGH
  - Threshold[7]=% del rango(TSerie)=si DEC y ESTÁ POR DEBAJO DE ESE VALOR este segmento es DECLOW

- Threshold[8]=% del rango(TSerie)=si TSerie ESTÁ POR ENCIMA DE ESE VALOR este segmento es VALHIGH
- Threshold[9]=% del rango(TSerie)=si TSerie ESTÁ POR DEBAJO DE ESE VALOR este segmento es VALLOW
- Threshold[10]=% of max(abs(TSerie))=si abs(TSerie) ESTÁ ENTRE 0 y ESE VALOR este segmento es VALZERO

- *LongMin* =Número mínimo de puntos para que un segmento pueda ser considerado como tal.

Parámetros de Salida:

- MATPatt=Matriz con los diferentes segmentos encontrados
  1. $TSerie=Serie Temporal
  2. $DEC=DEC Segments [col1=Posición, col2=Longitud, col3=Altura]
  3. $INC=INC Segments [col1= Posición, col2= Longitud, col3= Altura]
  4. $HOR=HOR Segments [col1= Posición, col2= Longitud, col3= Altura]
  5. Etc.....

Programa: kdextract.R

```
######################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# --------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# --------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

######################################################################
# (kdextract.R) Extract Segments from Time Series Vector Filtered
######################################################################

# INPUT PARAMETERS:
# TSerie=Time Series (Usually filtered)

# Param[1]=TRUE (Measures INCREMENTAL)
# Param[2]=TRUE (Measures DECREMENTAL)
# Param[3]=TRUE (Measures HORIZONTAL)
# Param[4]=TRUE (Measures INCHIGH)
# Param[5]=TRUE (Measures INCLOW)
# Param[6]=TRUE (Measures DECHIGH)
# Param[7]=TRUE (Measures DECLOW)
# Param[8]=TRUE (Measures VALHIGH)
# Param[9]=TRUE (Measures VALLOW)
# Param[10]=TRUE (Measures VALZERO)
# Param[11]=TRUE (Measures VALMED (Values Between VALHIGH and VALLOW)
# Param[12]=TRUE (Measures INCMED (Values Between INCHIGH and INCLOW)
# Param[13]=TRUE (Measures DECMED (Values Between DECHIGH and DECLOW)

# Threshold[1]=% of range(TSerie)=if INC and OVER this value is
INCREMENTAL
```

```
# Threshold[2]=% of range(TSerie)=if DEC and OVER this value is
DECREMENTAL
# Threshold[3]=% of range(TSerie)=if INC or DEC but BELOW this value
is HORIZONAL
# Threshold[4]=% of range(TSerie)=elseif INC and OVER this value is
INCHIGH
# Threshold[5]=% of range(TSerie)=elseif INC and BELOW this value is
INCLOW
# Threshold[6]=% of range(TSerie)=elseif DEC and OVER this value is
DECHIGH
# Threshold[7]=% of range(TSerie)=elseif DEC and BELOW this value is
DECLOW
# Threshold[8]=% of range(TSerie)=if TSerie is OVER this value is
VALHIGH
# Threshold[9]=% of range(TSerie)=if TSerie is BELOW this value is
VALLOW
# Threshold[10]=% of max(abs(TSerie))=if abs(TSerie) BETWEEN 0 and
this value is VALZERO

# LongMin[1-13]= Number of Min. Elements to be considered pattern
# ------------------------------------------------------------------

# OUTPUT PARAMETERS:
# MATPatt=Different type of patterns found:
#                 1. $TSerie=Time Series
#                 2. $DEC=DEC Segments
#                             [col1=Position, col2=length, col3=height]
#                 3. $INC=INC Segments
#                             [col1=Position, col2=length, col3=height]
#                 4. $HOR=HOR Segments
#                             [col1=Position, col2=length, col3=height]
#                 5. Etc.....
# ------------------------------------------------------------------

kdextract <- function(TSerie, Param=c(rep(TRUE,13)),
Threshold=c(0.05,0.05,0.05,0.40,0.10,0.40,0.10,0.8,0.2,0.05),
LongMin=rep(0,13))
{
      storage.mode(TSerie) <- "double"
      SerieP <- list(TSerie=TSerie)

      # Scale bettwen 0 and 1
      MinTSerie = min(TSerie)
      RangeTSerie=max(TSerie)-min(TSerie)
      TSerie <- (TSerie-MinTSerie)/RangeTSerie

      # Obtains Zeros of First Derivate
      Zeros <- .Call("zerocrossings",TSerie, PACKAGE="KDSeries")
      Zeros[length(Zeros)] <- 1

      # Where is Zeros?
      WhereZeros <- 1:length(Zeros)
      WhereZeros <- WhereZeros[Zeros %in% 1]

      # Get Long, Height and Position
      PosicionPat <- c(1,WhereZeros[-length(WhereZeros)])
      LongPat <- WhereZeros-PosicionPat
      AltPat <- TSerie[WhereZeros]- TSerie[PosicionPat]



      # Extract Segments

      # DECREMENTAL
```

```
if (Param[2])
{
      Cuales <- AltPat<(-Threshold[2])

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[2]]
      AltP <- AltP[LongP>=LongMin[2]]
      LongP <- LongP[LongP>=LongMin[2]]

      DEC <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(DEC=DEC))
      }

# HORIZONTAL
if (Param[3])
      {
      Cuales <- abs(AltPat)<Threshold[3]

      # Remove patterns with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[3]]
      AltP <- AltP[LongP>=LongMin[3]]
      LongP <- LongP[LongP>=LongMin[3]]

      HOR <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(HOR=HOR))
      }

# INCREMENTAL
if (Param[1])
{
      Cuales <- AltPat>Threshold[1]

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[1]]
      AltP <- AltP[LongP>=LongMin[1]]
      LongP <- LongP[LongP>=LongMin[1]]

      INC <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(INC=INC))
}

# INCLOW
if (Param[5])
      {
      Cuales <- AltPat>=Threshold[1] & AltPat<Threshold[5]

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[5]]
```

```
      AltP <- AltP[LongP>=LongMin[5]]
      LongP <- LongP[LongP>=LongMin[5]]

      INCLOW <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(INCLOW=INCLOW))
      }


# INCMED
if (Param[12])
      {
      Cuales <- AltPat>=Threshold[5] & AltPat<=Threshold[4]

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[12]]
      AltP <- AltP[LongP>=LongMin[12]]
      LongP <- LongP[LongP>=LongMin[12]]

      INCMED <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(INCMED=INCMED))
      }

# INCHIGH
if (Param[4])
      {
      Cuales <- AltPat>Threshold[4]

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[4]]
      AltP <- AltP[LongP>=LongMin[4]]
      LongP <- LongP[LongP>=LongMin[4]]

      INCHIGH <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(INCHIGH=INCHIGH))
      }



# DECLOW
if (Param[7])
      {
      Cuales <- AltPat<=(-Threshold[2]) & AltPat>(-Threshold[7])

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[7]]
      AltP <- AltP[LongP>=LongMin[7]]
      LongP <- LongP[LongP>=LongMin[7]]

      DECLOW <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(DECLOW=DECLOW))
      }
```

```
# DECMED
if (Param[13])
      {
      Cuales <- AltPat>=(-Threshold[6]) & AltPat<=(-Threshold[7])

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[13]]
      AltP <- AltP[LongP>=LongMin[13]]
      LongP <- LongP[LongP>=LongMin[13]]

      DECMED <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(DECMED=DECMED))
      }

# DECHIGH
if (Param[6])
      {
      Cuales <- AltPat<(-Threshold[6])

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPat[Cuales]
      AltP <- AltPat[Cuales]
      LongP <- LongPat[Cuales]

      PosP <- PosP[LongP>=LongMin[6]]
      AltP <- AltP[LongP>=LongMin[6]]
      LongP <- LongP[LongP>=LongMin[6]]

      DECHIGH <- cbind(PosP, LongP, AltP)
      SerieP <- c(SerieP, list(DECHIGH=DECHIGH))
      }


# VALLOW
if (Param[9])
      {
      # Extract Segments with Low Values
      SerieH <- as.numeric(TSerie<Threshold[9])

      # Obtains Zeroscrossing
      CerosH <- c(0,SerieH[-length(SerieH)])-SerieH

      # Where is zeros?
      DondeCerosH <- 1:length(CerosH)
      DondeCerosH <- DondeCerosH[abs(CerosH) %in% 1]

      # Get Long, Height and Position
      PosicionPathH <- c(1,DondeCerosH[-length(DondeCerosH)])
      LongPathH <- DondeCerosH-PosicionPathH

      # Only with -1 start points
      LongPathH <- LongPathH[CerosH[PosicionPathH]==-1]
      PosicionPathH <- PosicionPathH[CerosH[PosicionPathH]==-1]

      # Remove Segments with long lower than LongMin
      PosP <- PosicionPathH
      LongP <- LongPathH

      PosP <- PosP[LongP>=LongMin[9]]
      LongP <- LongP[LongP>=LongMin[9]]
```

```
            VALLOW <- cbind(PosP, LongP)

            SerieP <- c(SerieP, list(VALLOW=VALLOW))
            }
      # VALMED
      if (Param[11])
            {
            # Extract Segments Between with Low Values
            SerieH <- as.numeric(TSerie>=Threshold[9] &
TSerie<=Threshold[8])

            # Obtains Zeroscrossing
            CerosH <- c(0,SerieH[-length(SerieH)])-SerieH

            # Where is zeros?
            DondeCerosH <- 1:length(CerosH)
            DondeCerosH <- DondeCerosH[abs(CerosH) %in% 1]

            # Get Long, Height and Position
            PosicionPathH <- c(1,DondeCerosH[-length(DondeCerosH)])
            LongPathH <- DondeCerosH-PosicionPathH

            # Only with -1 start points
            LongPathH <- LongPathH[CerosH[PosicionPathH]==-1]
            PosicionPathH <- PosicionPathH[CerosH[PosicionPathH]==-1]

            # Remove Segments with long lower than LongMin
            PosP <- PosicionPathH
            LongP <- LongPathH

            PosP <- PosP[LongP>=LongMin[11]]
            LongP <- LongP[LongP>=LongMin[11]]

            VALMED <- cbind(PosP, LongP)

            SerieP <- c(SerieP, list(VALMED=VALMED))
            }
      # VALHIGH
      if (Param[8])
            {
            # Extract Segments with High Values
            SerieH <- as.numeric(TSerie>Threshold[8])

            # Obtains Zeroscrossing
            CerosH <- c(0,SerieH[-length(SerieH)])-SerieH

            # Where is zeros?
            DondeCerosH <- 1:length(CerosH)
            DondeCerosH <- DondeCerosH[abs(CerosH) %in% 1]

            # Get Long, Height and Position
            PosicionPathH <- c(1,DondeCerosH[-length(DondeCerosH)])
            LongPathH <- DondeCerosH-PosicionPathH

            # Only with -1 start points
            LongPathH <- LongPathH[CerosH[PosicionPathH]==-1]
            PosicionPathH <- PosicionPathH[CerosH[PosicionPathH]==-1]

            # Remove Segments with long lower than LongMin
            PosP <- PosicionPathH
            LongP <- LongPathH

            PosP <- PosP[LongP>=LongMin[8]]
```

```
            LongP <- LongP[LongP>=LongMin[8]]

            VALHIGH <- cbind(PosP, LongP)

            SerieP <- c(SerieP, list(VALHIGH=VALHIGH))
            }
      # VALZERO
      if (Param[10])
            {
            # Extract Segments with Zero Values
            TSerie <- SerieP[[1]]/RangeTSerie
            ThresholAdapt=max(abs(TSerie))*Threshold[10]
            SerieH <- as.numeric(abs(TSerie)<ThresholAdapt)

            # Obtains Zeroscrossing
            CerosH <- c(0,SerieH[-length(SerieH)])-SerieH

            # Where is zeros?
            DondeCerosH <- 1:length(CerosH)
            DondeCerosH <- DondeCerosH[abs(CerosH) %in% 1]

            # Get Long, Height and Position
            PosicionPatH <- c(1,DondeCerosH[-length(DondeCerosH)])
            LongPathH <- DondeCerosH-PosicionPatH

            # Only with -1 start points
            LongPathH <- LongPathH[CerosH[PosicionPatH]==-1]
            PosicionPatH <- PosicionPatH[CerosH[PosicionPatH]==-1]

            # Remove Segments with long lower than LongMin
            PosP <- PosicionPatH
            LongP <- LongPathH

            PosP <- PosP[LongP>=LongMin[10]]
            LongP <- LongP[LongP>=LongMin[10]]

            VALZERO <- cbind(PosP, LongP)
            SerieP <- c(SerieP, list(VALZERO=VALZERO))
            }

return(SerieP)
}
```

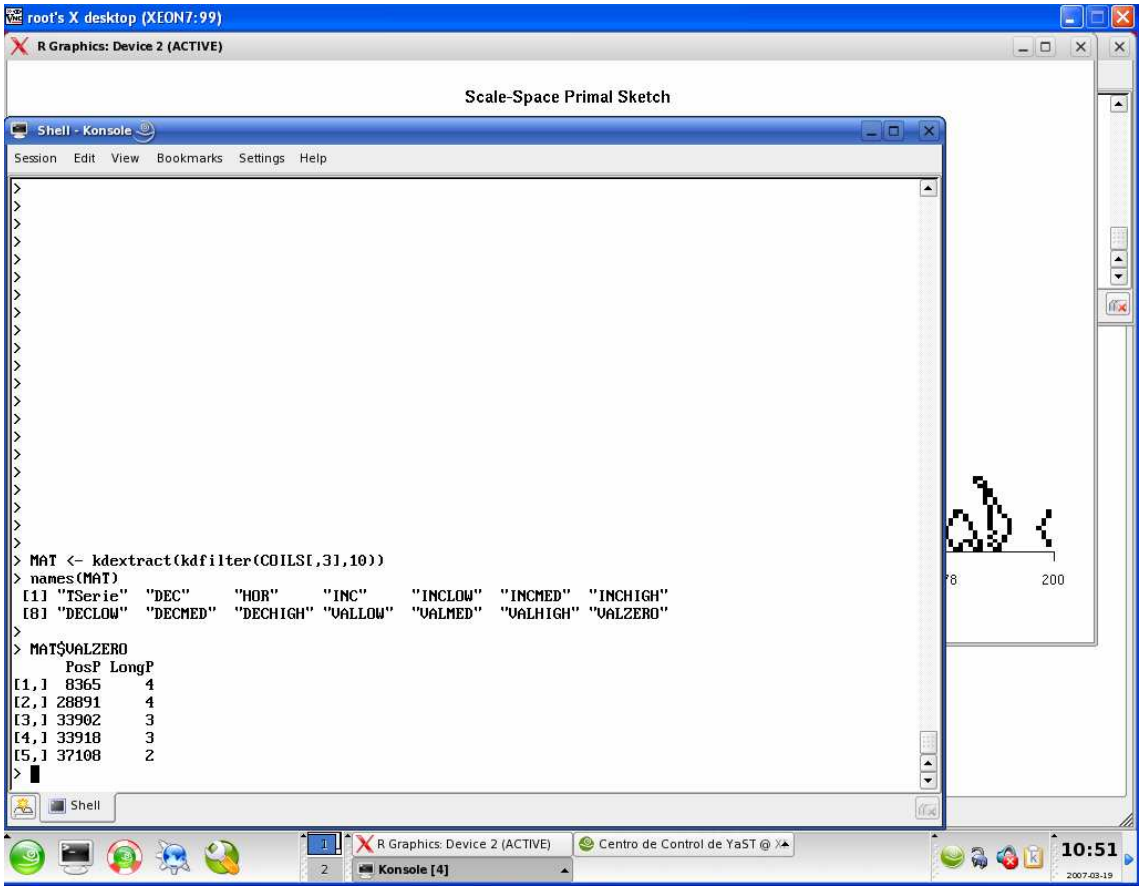
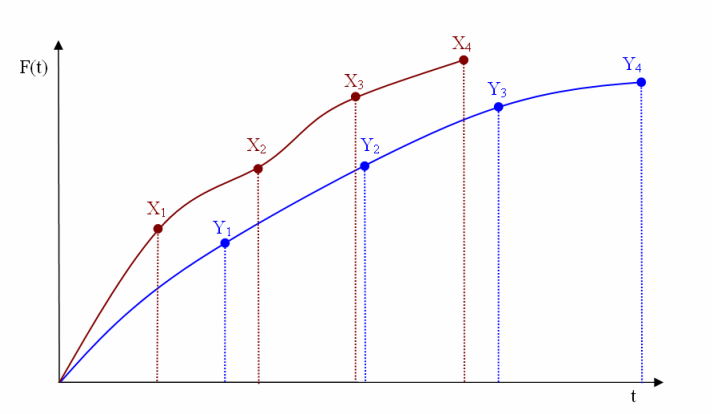


Figura 16. Ejemplo de aplicación de la función "kdextract". En este caso, se desarrolla un gráfico donde en el eje Y se identifica el ancho de cada filtro y en el eje X la posición del máximo o mínimo (punto negro).

I.9. Función “kdclusterpatt”: Agrupa Segmentos Similares.

Agrupa segmentos similares a partir de la distancia euclídea. Solamente es válido para segmentos del tipo INC, DEC y HOR. El proceso de segmentación, se realiza mediante el uso de dendrogramas que obtienen grupos a partir de dos datos obtenidos de los segmentos: la longitud y la forma. El segundo parámetro corresponde con la distancia euclídea de valores equidistantes entre segmentos.



$$DIS_{Shape} = \frac{1}{N} \cdot \sqrt{\sum_{i=1}^{N} \left(X(i \cdot \frac{L_X}{N}) - Y(i \cdot \frac{L_Y}{N}) \right)^2}$$

Figura 17. Calculo de la disimilitud por forma.

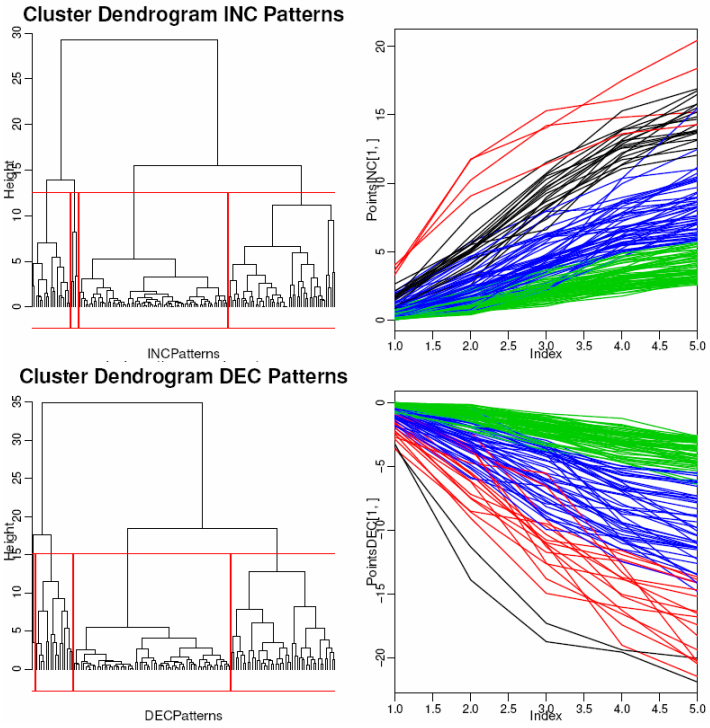


Figura 18. Clasificación de patrones crecientes y decrecientes mediante el uso de algoritmos jerárquicos.

Llamada a la función:
kdclusterpatt(MATPatt, INCParam=c(10,"g",4,"g",4), DECParam=c(10,"g","4","g",4), HORParam=c("g","4"))
Donde:

Parámetros de Entrada:

- *MATPatt* =Matriz con diferentes segmentos encontrados (proviene de *kdextract*() o *kdmergepatt*()).
- *INCParam*=
    1. Número de subsegmentos a dividir. Lo divide en subsegmentos para obtener la distancia y forma.
    2. Uso de dendrograma por grupos (g) o altura (h) para clasificar la forma.
    3. Valor de corte del dendrograma para la forma.
    4. Uso de dendrograma por grupos (g) o altura (h) para la longitud.
    5. Valor de corte del dendrograma para la longitud.
- *DECParam*=Igual que INCParam.
- *HORParam*=
    1. Uso de dendrograma por grupos (g) o altura (h) para la longitud.
    2. Valor de corte del dendrograma para la longitud.

Parámetros de Salida:

1. $MATSort: Patrones ordenados. [col1=Posición, col2=Código del patrón.]
2. $FamilyINC: Patrones de la familia INC si hay.
3. $FamilyDEC: Patrones de la familia DEC si hay.
4. $FamilyHOR: Patrones de la familia HOT si hay.
5. $PointsINC: Puntos usados para agrupar los INC.
6. $GroupsINCShape[[X]]: Familia de patrones INC por forma.
7. $GroupsINCLength[[X]]: Familia de patrones INC por longitud.
8. $LengthPattINC[[X]]: Longitud de los patrones INC.
9. $PointsDEC: Puntos usados para agrupar los DEC.
10. $GroupsDECShape[[X]]: Familia de patrones DEC por forma.
11. $GroupsDECLength[[X]]: Familia de patrones DEC por longitud.
12. $LengthPattDEC[[X]]: Longitud de los patrones DEC.
13. $GroupsHORLength[[X]]: Familia de patrones HOR por longitud.
14. $LengthPattHOR[[X]]: Longitud de los patrones HOR.

Programa: kdclusterpatt.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
```

```
# (kdclusterpatt.R) Cluster Similar Segment from Segment Matrix
(MATPatt=OBTAINED FROM kdextract.R)
#########################################################################

# INPUT PARAMETERS:
# MATPatt=Different type of segment found (FROM kdextract() or
kdmergepatt())
# INCParam= 1.- Number of Segments. Divide one Segment into frames in
order to obtain shape distance
#                 2.- Cluster dendrogram from shape distance by groups
(g) or by height (h)
#                 3.- Value of threshold dendrogram from shape distance
#                 4.- Cluster dendrogram from length distance by groups
(g) or by height (h)
#                 5.- Value of threshold dendrogram from length
distance
# DECParam= 1.- Number of Segments. Divide one Segment into frames in
order to obtain shape distance
#                 2.- Cluster dendrogram from shape distance by groups
(g) or by height (h)
#                 3.- Value of threshold dendrogram from shape distance
#                 4.- Cluster dendrogram from length distance by groups
(g) or by height (h)
#                 5.- Value of threshold dendrogram from length
distance
# HORParam= 1.- Cluster dendrogram from length distance by groups (g)
or by height (h)
#                 2.- Value of threshold dendrogram from length
distance
# ---------------------------------------------------------------------
------------------------

# OUTPUT PARAMETERS:
#                  1. $MATSort:Patterns Sorted
#                              [col1=Position, col2=Code Pattern
(FamilyCode (see below))
#                  2. $FamilyINC: Patterns of family INC as they appear
#                  3. $FamilyDEC: Patterns of family DEC as they appear
#                  4. $FamilyHOR: Patterns of family HOR as they appear

#                  5. $PointsINC: Points used to cluster INC patterns
#                  6. $GroupsINCShape[[X]]: INC Patterns Family X by
Shape
#                  7. $GroupsINCLength[[X]]: INC Patterns Family X by
Length
#                  8. $LengthPattINC[[X]]: INC Pattern's Length of
Family X
#                  9. $PointsDEC: Points used to cluster DEC patterns
#                 10. $GroupsDECShape[[X]]: DEC Patterns Family X by
Shape
#                 11. $GroupsDECLength[[X]]: DEC Patterns Family X by
Length
#                 12. $LengthPattDEC[[X]]: DEC Pattern's Length of
Family X
#                 13. $GroupsHORLength[[X]]: HOR Patterns Family X by
Length
#                 14. $LengthPattHOR[[X]]: HOR Pattern's Length of
Family X
# ---------------------------------------------------------------------


kdclusterpatt <- function(MATPatt, INCParam=c(10,"g",4,"g",4),
DECParam=c(10,"g","4","g",4), HORParam=c("g","4"))
{
```

```
      SimiPatt <- NULL

      ###################### Obtains groups using hierarchical
clustering (INC)  ##########################

      # mar=margins (Bot,Left,Top,Right), cex.axis=axis's text
      # lab=axis y size title
      # mgp=Text Axis's Distance (title, labels, line)
      par(bg="white", mar=c(2,2,2,2), cex.axis=0.7, cex.lab=0.8,
lab=c(10,5,6), mgp=c(1,0.5,0), xaxs="i")
      NF <- layout(matrix(c(1,2,3,4),2,2,byrow=TRUE))
      BorderPlot=2

      # FamilyINC=0001ssssssllllll  (0001=INC, s=shape type, l=length
type)
      FamilyINC <- rep(0,dim(MATPatt$INC)[1])

      # Cluster INCREMENTAL Segments
      MATPos <- MATPatt$INC[,1]
      MATLen <- MATPatt$INC[,2]
      PointsINC <- NULL
      for (i in 1:as.numeric(INCParam[1]))
            {
            PointsINC <- cbind(PointsINC,
MATPatt$TSerie[MATPos+MATLen*(i/as.numeric(INCParam[1]))]-
MATPatt$TSerie[MATPos])
            }

      # Cluster by Shape
      Hc <- hclust(dist(PointsINC))

      plot(Hc,hang=-1,main="Cluster Dendrogram INC Patterns by
SHAPE",xlab="INCPatterns",labels=FALSE)

      if (INCParam[2]=="h")
            {
            GroupsINCShape <- rect.hclust(Hc,
h=as.numeric(INCParam[3]), border=1:30)
            }
      else
            {
            GroupsINCShape <- rect.hclust(Hc,
k=as.numeric(INCParam[3]), border=1:as.numeric(INCParam[3]))
            }

      # Cluster by Length
      Hc <- hclust(dist(MATLen))

      plot(Hc,hang=-1,main="Cluster Dendrogram INC Patterns by
LENGHT",xlab="INCPatterns",labels=FALSE)

      if (INCParam[4]=="h")
            {
            GroupsINCLength <- rect.hclust(Hc,
h=as.numeric(INCParam[5]), border=1:30)
            }
      else
            {
            GroupsINCLength <- rect.hclust(Hc,
k=as.numeric(INCParam[5]), border=1:as.numeric(INCParam[5]))
            }
```

```
      # Plots Cluster INCSegments by SHAPE
      ColorsPat <- rep(0,dim(PointsINC)[1])
      for (i in 1:length(GroupsINCShape))
            {
            ColorsPat[GroupsINCShape[[i]]] <- i
            # FamilyINC=0001ssssssllllll  (0001=INC, s=shape type,
l=length type)
            FamilyINC[GroupsINCShape[[i]]] <- 4096+i*64
            }
      plot(PointsINC[1,],type="l",ylim=range(PointsINC),col=ColorsPat[
1],main="INC Clusters by Shape")
      for (i in 2:(dim(PointsINC)[1]))
            {
            lines(PointsINC[i,],col=ColorsPat[i])
            }

      # Plots Cluster INCSegmentsby LENGTH
      TablePat <- rep(0, length(GroupsINCLength))
      LengthMax <- TablePat
      LengthPattINC <- as.list(NULL)

      for (i in 1:length(GroupsINCLength))
            {
            # FamilyINC=0001ssssssllllll  (0001=INC, s=shape type,
l=length type)
            FamilyINC[GroupsINCLength[[i]]] <-
FamilyINC[GroupsINCLength[[i]]]+i
            TablePat[i] <- length(GroupsINCLength[[i]])
            LengthMax[i] <- max(MATLen[GroupsINCLength[[i]]])
            LengthPattINC <- c(LengthPattINC,
list(MATLen[GroupsINCLength[[i]]]))
            }

      boxplot(LengthPattINC,col=1:length(TablePat),ylim=c(0,max(Length
Max)+5),main="Boxplot INC Length
Cluster",xlab="Cluster",ylab="Length")
      text(1:length(TablePat),LengthMax+2,TablePat)

########################################## END INC SEGMENTS
##########################################



      ######################## Obtains groups using hierarchical
clustering (DEC)  ############################

      # mar=margins (Bot,Left,Top,Right), cex.axis=axis's text
      # lab=axis y size title
      # mgp=Text Axis's Distance (title, labels, line)
      X11()
      par(bg="white", mar=c(2,2,2,2), cex.axis=0.7, cex.lab=0.8,
lab=c(10,5,6), mgp=c(1,0.5,0), xaxs="i")
      NF <- layout(matrix(c(1,2,3,4),2,2,byrow=TRUE))
      BorderPlot=2

      # FamilyDEC=0002ssssssllllll  (0002=DEC, s=shape type, l=length
type)
      FamilyDEC <- rep(0,dim(MATPatt$DEC)[1])

      # Cluster DECREMENTAL Patterns
      MATPos <- MATPatt$DEC[,1]
      MATLen <- MATPatt$DEC[,2]
      PointsDEC <- NULL
      for (i in 1:as.numeric(DECParam[1]))
```

```
            {
            PointsDEC <- cbind(PointsDEC,
MATPatt$TSerie[MATPos+MATLen*(i/as.numeric(DECParam[1]))]-
MATPatt$TSerie[MATPos])
            }

      # Cluster by Shape
      Hc <- hclust(dist(PointsDEC))

      plot(Hc,hang=-1,main="Cluster Dendrogram DEC Patterns by
SHAPE",xlab="DECPatterns",labels=FALSE)

      if (DECParam[2]=="h")
            {
            GroupsDECShape <- rect.hclust(Hc,
h=as.numeric(DECParam[3]), border=1:30)
            }
      else
            {
            GroupsDECShape <- rect.hclust(Hc,
k=as.numeric(DECParam[3]), border=1:as.numeric(DECParam[3]))
            }

      # Cluster by Length
      Hc <- hclust(dist(MATLen))

      plot(Hc,hang=-1,main="Cluster Dendrogram DEC Patterns by
LENGHT",xlab="DECPatterns",labels=FALSE)

      if (DECParam[4]=="h")
            {
            GroupsDECLength <- rect.hclust(Hc,
h=as.numeric(DECParam[5]), border=1:30)
            }
      else
            {
            GroupsDECLength <- rect.hclust(Hc,
k=as.numeric(DECParam[5]), border=1:as.numeric(DECParam[5]))
            }


      # Plots Cluster DECPatterns by SHAPE
      ColorsPat <- rep(0,dim(PointsDEC)[1])
      for (i in 1:length(GroupsDECShape))
            {
            # FamilyDEC=0002ssssssllllll  (0002=DEC, s=shape type,
l=length type)
            FamilyDEC[GroupsDECShape[[i]]] <- 8192+i*64
            ColorsPat[GroupsDECShape[[i]]] <- i
            }
      plot(PointsDEC[1,],type="l",ylim=range(PointsDEC),col=ColorsPat[
1],main="DEC Clusters by Shape")
      for (i in 2:(dim(PointsDEC)[1]))
            {
            lines(PointsDEC[i,],col=ColorsPat[i])
            }

      # Plots Cluster DECPatterns by LENGTH
      TablePat <- rep(0, length(GroupsDECLength))
      LengthMax <- TablePat
      LengthPattDEC <- as.list(NULL)

      for (i in 1:length(GroupsDECLength))
            {
```

```
            # FamilyDEC=0002ssssssllllll  (0002=DEC, s=shape type,
l=length type)
            FamilyDEC[GroupsDECLength[[i]]] <-
FamilyDEC[GroupsDECLength[[i]]]+i
            TablePat[i] <- length(GroupsDECLength[[i]])
            LengthMax[i] <- max(MATLen[GroupsDECLength[[i]]])
            LengthPattDEC <- c(LengthPattDEC,
list(MATLen[GroupsDECLength[[i]]]))
            }

      boxplot(LengthPattDEC,col=1:length(TablePat),ylim=c(0,max(Length
Max)+5),main="Boxplot DEC Length
Cluster",xlab="Cluster",ylab="Length")
      text(1:length(TablePat),LengthMax+2,TablePat)

########################################## END DEC PATTERNS
##########################################


###################### Obtains groups using hierarchical clustering
(HOR)  ############################

      # mar=margins (Bot,Left,Top,Right), cex.axis=axis's text
      # lab=axis y size title
      # mgp=Text Axis's Distance (title, labels, line)
      X11()
      par(bg="white", mar=c(2,2,2,2), cex.axis=0.7, cex.lab=0.8,
lab=c(10,5,6), mgp=c(1,0.5,0), xaxs="i")
      NF <- layout(matrix(c(1,2),2,1,byrow=TRUE))
      BorderPlot=2

      # FamilyHOR=0003000000llllll  (0003=HOR, s=shape type, l=length
type)
      FamilyHOR <- rep(0,dim(MATPatt$HOR)[1])

      # Cluster HORIZONTAL Patterns
      MATLen <- MATPatt$HOR[,2]

      # Cluster by Length
      Hc <- hclust(dist(MATLen))

      plot(Hc,hang=-1,main="Cluster Dendrogram HOR Patterns by
LENGHT",xlab="HORPatterns",labels=FALSE)

      if (HORParam[1]=="h")
            {
            GroupsHORLength <- rect.hclust(Hc,
h=as.numeric(HORParam[2]), border=1:30)
            }
      else
            {
            GroupsHORLength <- rect.hclust(Hc,
k=as.numeric(HORParam[2]), border=1:as.numeric(HORParam[2]))
            }


      # Plots Cluster HORPatterns by LENGTH
      TablePat <- rep(0, length(GroupsHORLength))
      LengthMax <- TablePat
      LengthPattHOR <- as.list(NULL)

      for (i in 1:length(GroupsHORLength))
            {
```

```
            # FamilyHOR=0003000000llllll  (0003=HOR, s=shape type,
l=length type)
            FamilyHOR[GroupsHORLength[[i]]] <- 12288+i
            TablePat[i] <- length(GroupsHORLength[[i]])
            LengthMax[i] <- max(MATLen[GroupsHORLength[[i]]])
            LengthPattHOR <- c(LengthPattHOR,
list(MATLen[GroupsHORLength[[i]]]))
            }

      boxplot(LengthPattHOR,col=1:length(TablePat),ylim=c(0,max(Length
Max)+5),main="Boxplot HOR Length
Cluster",xlab="Cluster",ylab="Length")
      text(1:length(TablePat),LengthMax+2,TablePat)

########################################## END DEC PATTERNS
##########################################

      # Creates serie using nominal patterns
      MATSort <-
rbind(cbind(Pos=MATPatt$INC[,1],Code=FamilyINC),cbind(Pos=MATPatt$DEC[
,1],Code=FamilyDEC),cbind(Pos=MATPatt$HOR[,1],Code=FamilyHOR))
      MATSort <- MATSort[order(MATSort[,1]),]


      SimiPatt <- list(MATSort=MATSort, FamilyINC=FamilyINC,
FamilyHOR=FamilyHOR, FamilyDEC=FamilyDEC, PointsINC=PointsINC,
GroupsINCShape=GroupsINCShape, GroupsINCLength=GroupsINCLength,
LengthPattINC=LengthPattINC,  PointsDEC=PointsDEC,
GroupsDECShape=GroupsDECShape, GroupsDECLength=GroupsDECLength,
LengthPattDEC=LengthPattDEC, GroupsHORLength=GroupsHORLength,
LengthPattHOR=LengthPattHOR)
      return(SimiPatt)
}
```

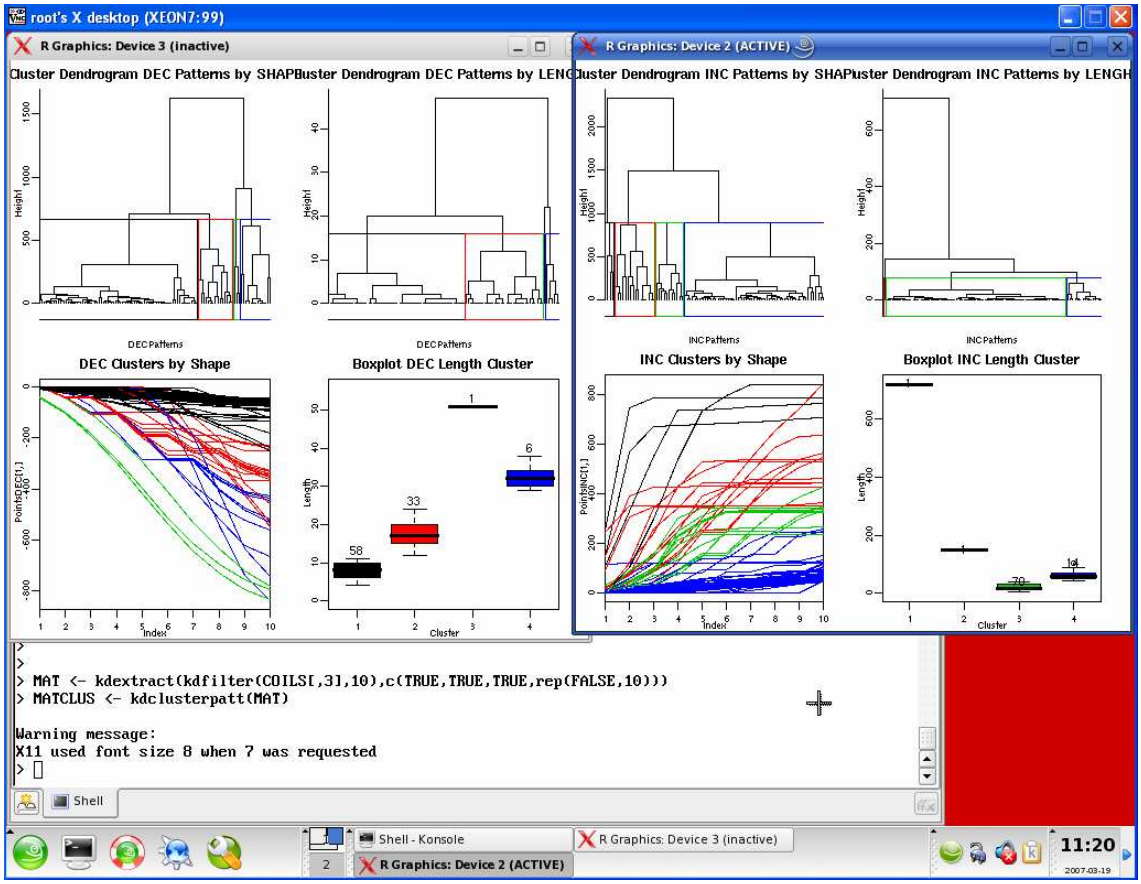


Figura 19. Ejemplo de aplicación de la función "kdcluster". En este caso, se agrupan segmentos incrementales, decrementales y horizontales buscando cuatro familias de cada tipo y usando un dendrograma para segmentarlos según la longitud y forma del segmento.

I.10. Función “kdmergepatt”: Funde segmentos parecidos.

Funde segmentos parecidos y elimina los segmentos que no son importantes.

Llamada a la función:
kdmergepatt(MAT, Delete=c(0,0,1), MinHeight=c(0.05,-0.05))

Parámetros de Entrada:
- *MAT*=Matriz proveniente de *kdextract*().
- *Delete*= Borra segmentos con longitud menos o igual a: x={IND, DEC, HOR}.
- *MinHeight*= Mínima altura para ser considerado. {IND,DEC}.

Parámetros de Salida: Matriz con los segmentos fundidos o eliminados.

Programa: kdmerge.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdmergepatt.R) Merge segments from MAT=(from kdextract())
#####################################################################

# INPUT PARAMETERS:
# MAT=(from kdextract())
# Delete=Delete segments with a length minor or equal than x=(INC,
DEC, HOR)
# MinHeigth=(INC,DEC) thresholds to be considerated HOR, INC, DEC
(INC= (if > INC) else HOR, DEC= (if < DEC) else HOR
# ------------------------------------------------------------------
------------------------

# OUTPUT PARAMETERS:
# Returns: 1. MAT=Matrix with merged segments
# ------------------------------------------------------------------
------------------------

kdmergepatt <- function(MAT, Delete=c(0,0,1), MinHeight=c(0.05,-0.05))
{
      # INC=1, DEC=2, HOR=3
```

```
      MATPatt <-
rbind(cbind(MAT$INC[MAT$INC[,2]>Delete[1],],1),cbind(MAT$DEC[MAT$DEC[,
2]>Delete[2],],2),cbind(MAT$HOR[MAT$HOR[,2]>Delete[3],],3))

      SortP <- 0
      while (SortP==0)
      {
            SortP <- 1

            # Sort Patterns
            MATPatt <- MATPatt[order(MATPatt[,1]),]
            LenPatt <- dim(MATPatt)[1]
            MATM <- NULL
            i=1
            while (i<LenPatt)
                  {
                  Pos=MATPatt[i,1]
                  Len=MATPatt[i,2]
                  Alt=MATPatt[i,3]
                  Tipo=MATPatt[i,4]
                  j=i+1
                  while (MATPatt[j,4]==Tipo && j<=LenPatt)
                              {
                              SortP <- 0
                              Len=Len+MATPatt[j,2]
                              Alt=Alt+MATPatt[j,3]
                              j=j+1
                              }
                  MATM <- rbind(MATM, c(Pos, Len, Alt, Tipo))
                  i=j
                  }
            MATPatt=MATM
      }
      MATFIN <- list(TSerie=MAT$TSerie, DEC=MATM[MATM[,4]==2,1:3],
HOR=MATM[MATM[,4]==3,1:3], INC=MATM[MATM[,4]==1,1:3])

      return(MATFIN)
}
```

I.11. Función “kdplotpatt”: Visualiza los patrones encontrados.

Dibuja una serie temporal y visualiza los patrones encontrados. Válido solamente para patrones INC, DEC, y HOR.

Llamada a la función:

kdplotpatt(Pat, Ini=1, End=length(Pat$Serie))

Parámetros de Entrada:

- *Pat*=Matriz proveniente de *kdextract*()).
- *Ini*= Punto Inicial.
- *Fin*=Punto Final.

Parámetros de Salida: Ninguno.

Programa: kdplotpatt.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdplotpatt.R) Plot Segments Extracted from kdextract()
#####################################################################

# INPUT PARAMETERS:
# Pat=Different type of segments found (FROM kdclusterpatt())
# Ini=First point
# End=Last point
# ------------------------------------------------------------------
------------------------

# OUTPUT PARAMETERS:
# Returns: 1. None
# ------------------------------------------------------------------

kdplotpatt <- function (Pat, Ini=1, End=length(Pat$Serie))
{
# mar=margins (Bot,Left,Top,Right), cex.axis=axis's text
# lab=axis y size title
# mgp=Text Axis's Distance (title, labels, line)

      par(bg="white", mar=c(2,3,0.5,0.5), cex.axis=0.7, cex.lab=0.8,
lab=c(10,5,6), mgp=c(1,0.5,0), xaxs="i")
```

```
      NF <- layout(matrix(c(1,2),2,1,byrow=TRUE),
widths=c(1,1),heights=c(3,2), respect=FALSE)

      # Plot Temporal Series
      plot (Pat$TSerie[Ini:End],ylab="f(t)",xlab="t",type="l")

      # Plot Bars
      NumPat <- length(Pat)-1
      HeighB <- rep(0,NumPat+2)

      par(las=1, mgp=c(1,0.3,0))
      mp <- barplot(HeighB, names.arg=names(c("",Pat[2:
(NumPat+1)],"")),xlim=c(Ini,End),horiz=TRUE,axes=FALSE)

      Incr <- (max(mp)/NumPat)/5
      for (h in 1:NumPat)
      {
            if (length(Pat[[h+1]])!=0)
                  {
                  rect(Pat[[h+1]][,1],mp[h+1]-Incr,
Pat[[h+1]][,1]+Pat[[h+1]][,2], mp[h+1]+Incr, col=h)
                  }
      }
}
```

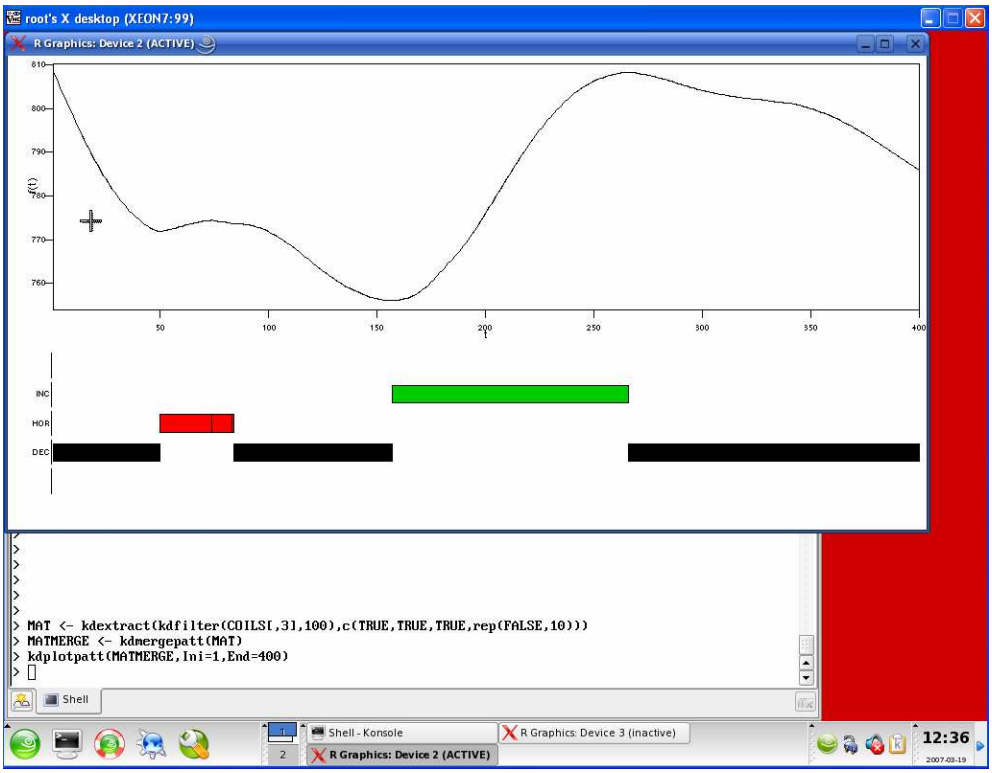


Figura 20. Ejemplo de uso de la función "kdplotpatt".

I.12. Función “kddiscretize”: Discretiza una serie temporal.

Esta función discretiza una serie temporal dependiendo de una serie de niveles preestablecidos por Thershold. Es decir, convierte una función temporal continua en el tiempo en una función escalonada, cuyos únicos posibles valores son los determinados en Thershold. Cada uno de esos valores fijos se pueden representar por etiquetas. Vamos a ver un ejemplo típico con las notas de una clase. Éstas se dan en su valor numérico pero queremos clasificarlas con las etiquetas típicas de "Aprobado","Notable"...Siempre que los valores numéricos estén entre unos márgenes establecidos por el profesor (en este caso los típicos que coconemos). Así, para una clase con 14 alumnos, la llamada a la función sería:

```
> Threshold<-c(0,2,5,6,7,9,10)
> Labels<-
c("MuyDeficiente","Insuficiente","Aprobado","Bien","Notable","Sobresal
iente")
> notasclase<-c(2,4,5,6,7.5,2.3,1.6,8.9,9.6,4.1,0,1.3,5.6,4.2)
> kddiscretize(notasclase,Threshold,Labels)

Y su salida:
 [1] "Insuficiente"  "Insuficiente"  "Aprobado"      "Bien"
 [5] "Notable"       "Insuficiente"  "MuyDeficiente" "Notable"
 [9] "Sobresaliente" "Insuficiente"  "MuyDeficiente" "MuyDeficiente"
[13] "Aprobado"      "Insuficiente"
```

Llamada a la función:

kddiscretize <- function (TSerie, Threshold,Labels)

Parámetros de Entrada:

- *TSerie*=Serie temporal.
- *Threshold* = Vector con valores de corte.
- *Labels* =Vector de etiquetas.

Parámetros de Salida: Serie temporal discretizada.

Programa: kddiscretize.R

```
#######################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# --------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.
# (kddiscretize.R)discretize a Time Series.
#######################################################################
```

```
# INPUT PARAMETERS:
# TSerie=Time Series
# Threshold=margins for chaging the label
# Labels=characters for describing the different levels of Threshold
#
# ------------------------------------------------------------------------

# OUTPUT PARAMETERS:
# SerieFilt=Time Series discretized
# ------------------------------------------------------------------------
------------------------

kddiscretize <- function (TSerie, Threshold,Labels)
{
TSerieFilt<-TSerie
#Check errors
if ((length(Threshold)) < (length(Labels)))
{
print("The length of Threshold can't be bigger than the length of
Labels")
return()
}

for (i in 1:length(TSerie))
{
j<-1
      repeat
            {
            if (j==length(Threshold))
            {
            TSerieFilt[i]<-Labels[length(Labels)]
            break
            }
            if ((TSerie[i]>Threshold[j]) & (TSerie[i]<Threshold[j+1]))
            {
            TSerieFilt[i]<-Labels[j]
            break
            }
            if (TSerie[i]==Threshold[j])
            {
            TSerieFilt[i]<-Labels[j]
            break
            }
            if (j>length(Threshold))
            {
            print("An error has ocurred.")
            return
            }
            j<-j+1
            }

}
return(TSerieFilt)

}
```

I.13. Función “kdfilterFFT”: Realiza un filtrado mediante la transformada FFT.

Realiza un filtrado basado en la transformada rápida de fourier (FFT) de una serie temporal determinada. Como parámetros podemos pasarle la anchura de la ventana que queremos usar (WidthW), el desplazamiento con se desplazará dicha ventana, el tipo de ventana que queremos usar para ponderar los elementos de la serie temporal (cuadrada o campana de Gauss) y rango de armónicos que queremos eliminar de la transformada de la función temporal (por defecto de un 20% de la frecuencia máxima a un 70%) siendo la frecuencia fundamental igual a 1/T (T= anchura de ventana en unidades de tiempo).

Una llamada típica a esta función sería por ejemplo:

```
kdfilterFFT(COILS$VEL[1000:1800],10,7,"gauss",c(30,80))
```

O si queremos simplemete (usando los valores por defecto):

```
kdfilterFFT(COILS$VEL[1000:1800])
```

Llamada a la función:

kdfilterFFT <- function (TSerie, WidthW=0.02*length(TSerie), Slide=WidthW, Filter="mean" , Range=c(20,70))

Parámetros de Entrada:

- *TSerie*=Serie temporal.
- *WidthW* = Anchura de la ventana.
- *Slide* =Desplazamiento de la ventana.
- *Filter*=Tipo de filtro Gaussiano=”gauss” o Rectangular=”rect”
- *Range*=Rango de frecuencias a mantener (en porcentaje).

Parámetros de Salida: Serie temporal discretizada.

Programa: kdfilterFFT.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.
# (kddiscretize.R)discretize a Time Series.

#####################################################################
# (kdfilterFFT.R)Filter a Time Series by FFT function
```

```
#########################################################################

# INPUT PARAMETERS:
# TSerie=Time Series
# WidthW=Filter's Window Width
# Slide=Displacement of the window filter
# Filter=Type of Filter. "mean"=rectangular window, "gauss"=gauss
window, "median"=median from window
# Range=Range of frecuencies that we want to eliminate in % (min % of
total length,max % of total length)
#
# ---------------------------------------------------------------------
-----------------------

# OUTPUT PARAMETERS:
# SerieFilt=Time Series Filtered
# ---------------------------------------------------------------------
-----------------------

kdfilterFFT <- function (TSerie, WidthW=0.02*length(TSerie),
Slide=WidthW , Filter="mean" , Range=c(20,70))
{
AuxTSerie<-TSerie
if (WidthW>length(TSerie))
{
print("WidthW must be smaller than length of the TSerie")
return()
}
storage.mode(TSerie) <- "double"
i1<-1
i2<-WidthW
#####################################
while (i2!=length(TSerie))
{
      if (i2>=length(TSerie)) {i1<-i1+Slide;i2<-length(TSerie)}

      if (Filter=="rect" || Filter=="mean")
            SerieFilt <-
.Call("meanfilter",TSerie[i1:i2],WidthW,PACKAGE="KDSeries")

      if (Filter=="gauss")
      {
            t <- 1: WidthW
            FiltW <- (1/(sqrt(2*pi)*sd(t)))*exp((-(t-
mean(t))^2)/(2*sd(t)^2))
            FiltW <- FiltW/sum(FiltW)
            SerieFilt <- filter(TSerie[i1:i2], FiltW, sides=2,
circular=TRUE)
      }
      SerieFilt.frec<-fft(SerieFilt)
      aux1<-SerieFilt.frec[-1]
      margin1<-as.integer((Range[1]/100)*(length(SerieFilt.frec)%/%2))
      margin2<-as.integer((Range[2]/100)*(length(SerieFilt.frec)%/%2)
      )
      aux1[(margin1):(margin2)]<-0
      aux1[((length(SerieFilt.frec))-
margin1):((length(SerieFilt.frec))-margin2)]<-0
      aux1<-c(SerieFilt.frec[1],aux1)
      aux2<-Re(fft(aux1,inverse=T))/(length(aux1))
      AuxTSerie[i1:i2]<-aux2
      if (i2<length(TSerie)) {i1<-i1+Slide;i2<-i2+Slide}

}
```

```
#########################################################################End
while
plot(TSerie,type="l")
lines(AuxTSerie,col="red",type="l")
return(AuxTSerie)

}
```

I.14. Función “kdplotdiscrete”: Dibuja la serie original y la discretizada.

Esta función dibuja la serie temporal original y la discretizada proveniente de la función kddiscretize. Una llamada típica para el ejemplo:

```
      > Threshold<-c(0,2,5,6,7,9,10)
      > Labels<-
      c("MuyDeficiente","Insuficiente","Aprobado","Bien","Notable","So
bresa liente")
      > notasclase<-c(2,4,5,6,7.5,2.3,1.6,8.9,9.6,4.1,0,1.3,5.6,4.2)
```

La llamada a la función:

```
kdplotdiscretize(kddiscretize(notasclase,Threshold,Labels),notasclase,
Threshold,Labels)
```

Llamada a la función:

kdplotdiscretize <- function (TSerieD,TSerie,Threshold,Labels)

Parámetros de Entrada:

- *TSerieD*=Serie temporal discretizada.
- *TSerie*=Serie temporal normal.
- *Threshold* = Vector con valores de corte.
- *Labels* =Vector de etiquetas.

Parámetros de Salida: Dibuja la Serie temporal discretizada y Serie.

Programa: kdplotdiscretize.R

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#####################################################################
# (kdplotdiscretize.R) Plot graphics from kddiscretize
#####################################################################
# INPUT PARAMETERS:
# TSerieD=discretized series
# TSerie=original series
# Threshold=margins for chaging the label
# ------------------------------------------------------------------
# OUTPUT PARAMETERS:
```

```
# Returns: 1. Plot Series.
# ------------------------------------------------------------------------
kdplotdiscretize <- function (TSerieD,TSerie,Threshold,Labels)
{
AuxSerie<-TSerie

for (i in 1:length(TSerieD))
{
j<-1
      repeat
      {
            if (Labels[j]==TSerieD[i])
            {
            AuxSerie[i]<-Threshold[j]
            break
            }
      if (j>length(Labels))
      {
      print("An error has ocurred.")
      return
      }
      j<-j+1
      }
j<-0
}
plot.ts(TSerie,type="p")
lines(AuxSerie,type="s",col="red")
return(AuxSerie)
}
##############################################################################
```

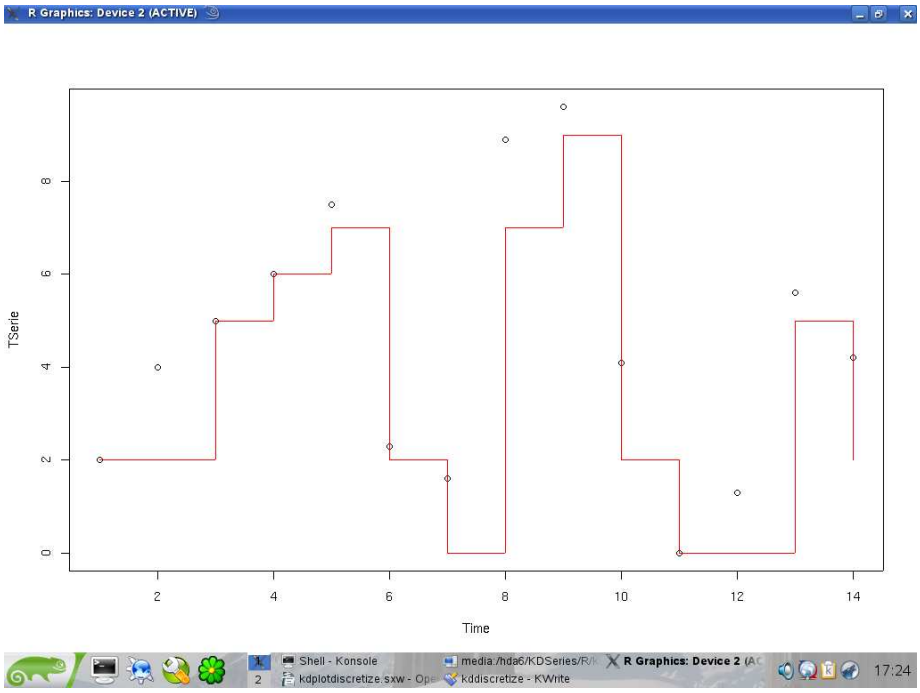


Figura 21. Podemos observar la serie temporal original (puntos negros), con sus valores numéricos exactos y la serie discretizada en rojo.

I.15. Función “kdtypefilters”: Ddevuelve una matriz en la que cada columna es la serie filtrada con un tipo de filtro.

Esta función nos devuelve una matriz en la que cada columna almacena una serie temporal que corresponden, cada una de ellas, a una serie filtrada utilizando 5 filtros distintos: media, gauss, mediana, máximo y mínimo. Un ejemplo sería éste:

```
> kdtypefilters(COILS$VEL[1:1000],20)
          mean     gauss median max min
   [1,] 132.50 134.34715  132.5 145 120
   [2,] 133.75 136.14227  145.0 145 120
   [3,] 135.00 137.83769  145.0 145 120
   [4,] 136.25 139.39385  145.0 145 120
   [5,] 137.50 140.78194  145.0 145 120
   [6,] 138.75 141.98524  145.0 145 120
   [7,] 140.00 142.99898  145.0 145 120
   [8,] 141.25 143.82896  145.0 145 120
   [9,] 142.50 144.48935  145.0 145 120
  [10,] 143.75 145.00000  145.0 145 120
  [11,] 145.00 145.00000  145.0 145 145
  [12,] 145.00 145.00000  145.0 145 145
  [13,] 145.00 145.00000  145.0 145 145
  [14,] 145.00 145.00000  145.0 145 145
  [15,] 145.00 145.00000  145.0 145 145
  [16,] 145.00 145.00000  145.0 145 145
  [17,] 145.00 145.00000  145.0 145 145
  [18,] 145.00 145.00000  145.0 145 145
  [19,] 145.00 145.00000  145.0 145 145
  [20,] 145.00 145.00000  145.0 145 145
.............................................................
[998,] 128.75 128.85773  120.0 145 120
 [999,] 130.00 130.65285  120.0 145 120
[1000,] 131.25 132.50000  120.0 145 120
```

Llamada a la función:

kdtypefilters <- function (TSerie, WidthW=10)

Parámetros de Entrada:

- *TSerie*=Serie temporal normal.
- *WidthW* = Ancho de la ventana de filtrado.

Parámetros de Salida: Matriz donde cada columna es la serie temporal filtrada con un tipo de filtro distinto.

Programa: kdtypefilters.**R**

```
#####################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
```

```
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.
```

```
##############################################################################
# (kdtypefilters.R)Filter a Time Series with different type of filters
##############################################################################

# INPUT PARAMETERS:
# TSerie=Time Series
# WidthW=Filter's Window Width
# ------------------------------------------------------------------------
# OUTPUT PARAMETERS:
# SerieFilt= matrix that each column is a series filtered by a
different filter.
# ------------------------------------------------------------------------

kdfilters <- function (TSerie, WidthW=10)
{
#to change the storage mode of TSerie (real->double)
storage.mode(TSerie) <- "double"
matresult<-matrix(0,length(TSerie),5)

#meanfilter
matresult[,1] <- .Call("meanfilter",TSerie,WidthW,PACKAGE="KDSeries")
#gaussfilter
t <- 1: WidthW
FiltW <- (1/(sqrt(2*pi)*sd(t)))*exp((-(t-mean(t))^2)/(2*sd(t)^2))
FiltW <- FiltW/sum(FiltW)
matresult[,2] <- filter(TSerie, FiltW, sides=2, circular=TRUE)
#medianfilter
matresult[,3]<- .Call("medianfilter",TSerie,WidthW,
PACKAGE="KDSeries")
#max
matresult[,4]<- .Call("maxfilter",TSerie,WidthW, PACKAGE="KDSeries")
#min
matresult[,5] <- .Call("minfilter",TSerie,WidthW, PACKAGE="KDSeries")
colnames(matresult)<-c("mean","gauss","median","max","min")
return(matresult)
}
```

I.16. Función “kdapproxPCA”: Comprime una serie temporal usando el indice PCA

Esta función surge a partir del índice PCA. Este índice se basa en la observación de que podemos obtener una aproximación muy buena si segmentamos la serie en tramos iguales y guardamos el valor de la media de valores de ese tramo en un vector auxiliar. En la llamada a la función podemos especificar la longitud de los tramos que queremos emplear en la segmentación. Si no se da un valor, la función toma 15 tramos por defecto. Este valor habrá que tenerlo muy en cuenta si la serie temporal es muy amplia, o si aparecen muchos cruces por cero en las series.

Un ejemplo de llamada a la función y su respuesta podría ser:

```
> kdapproxPCA(velocidades,15)
>
$compressed_series
 [1] 120.00000 119.98246 120.00000 120.00000 120.00000 120.00000
119.98246
 [8] 119.94737  83.70175 106.87719 114.24561 120.00000 106.84211
95.91228
[15] 107.49123

$compressed_length
[1] 15

$original_length
[1] 851
```

Vemos cómo se ha comprimido la función de 851 muestras a 15. Esto supondría también una gran pérdida de información que el usuario tendrá que valorar en su medida. La función también nos devuelve una gráfica de la señal comprimida. En este caso se ha dibujado la señal comprimida en color rojo y la señal original en negro. La pérdida en la definición de los detalles de la gráfica comprimida, al menos en este caso, es bastante evidente aunque en muchas ocasiones la pérdida de información puede ser asumible.

Llamada a la función:

kdapproxPCA <- function (TSerie,len=length(TSerie)/15)

Parámetros de Entrada:

- *TSerie*=Serie temporal normal.
- *len* = Longitud de los segmentos.

Parámetros de Salida: Lista con la serie temporal comprimida.

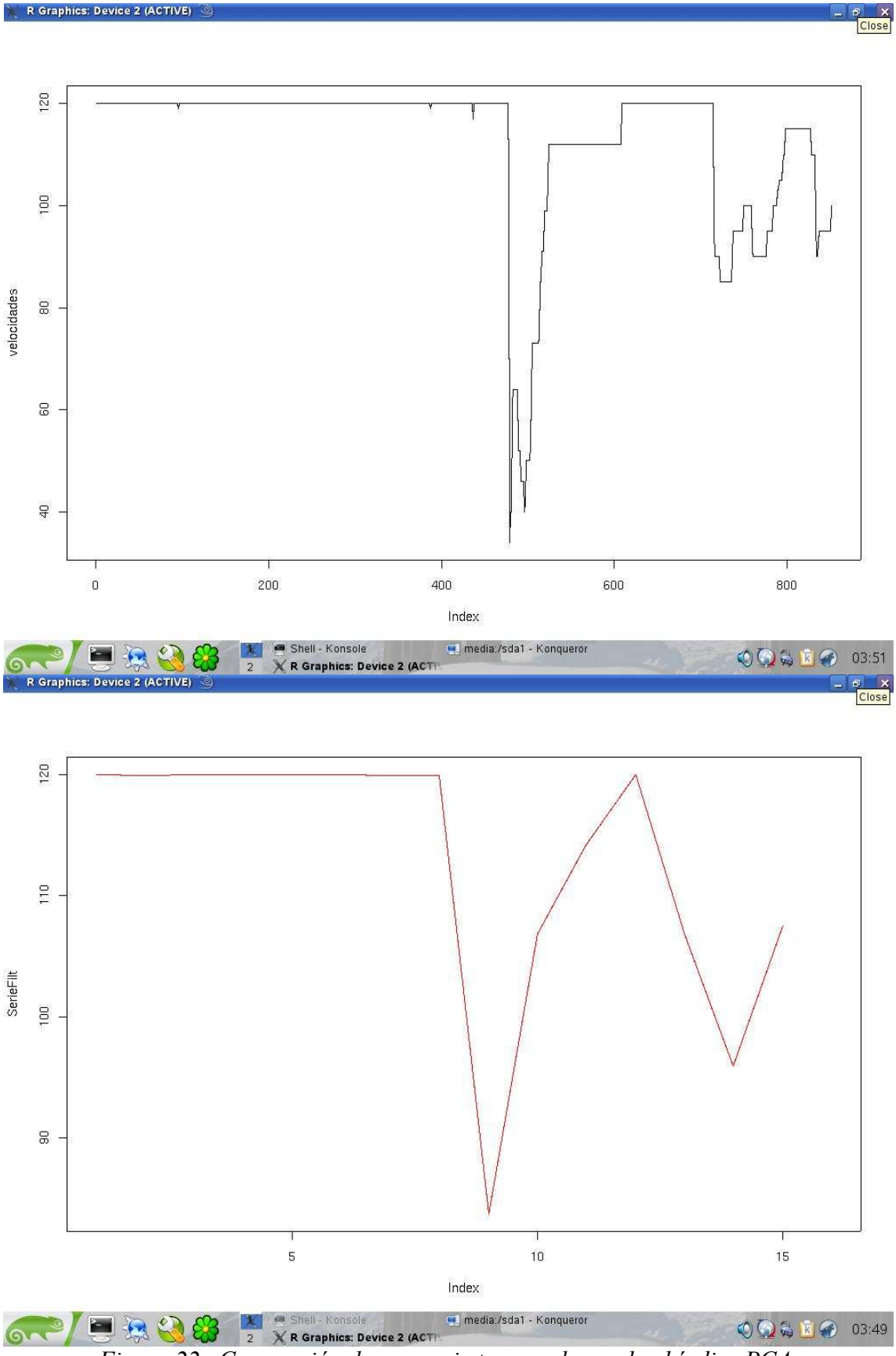


Figura 22. Compresión de una serie temporal usando el índice PCA.

I.17. Función “kdapproxPAA”: Sustituye una serie temporal usando el indice PCA

Esta función surge a partir del índice PAA, que en principio es muy parecido al PCA. La diferencia entre las dos funciones es que kdapproxPAA no realiza una compresión de los datos, sino que realiza simplemente un filtrado de los mismos tomando, sustituyendo cada valor de la serie por la media de sí mismo y los del tramo segmentado. Esta función es equivalente a un filtrado de media con ventana rectangular.

Un ejemplo de llamada a la función y su respuesta podría ser:

```
> kdapproxPAA(velocidades)

  [1] 120.00000 120.00000 120.00000 120.00000 120.00000 120.00000
120.00000
  [8] 120.00000 120.00000 120.00000 120.00000 120.00000 120.00000
120.00000
 [15] 120.00000 120.00000 120.00000 120.00000 120.00000 120.00000
120.00000
 [22] 120.00000 120.00000 120.00000 120.00000 120.00000
120.00000.......
....120.00000 119.98246 119.98246 119.98246 119.98246 119.98246
119.98246....
```

La utilización de esta función permite la suavización de la serie temporal, eliminando ruido que pudiera presentar. La salida gráfica de esta función podría ser:

Si comparamos el resultado de aplicar esta función con otro parámetro de longitud de tramo segmentado (o ventana) en la misma serie temporal, podemos observar que en este caso no hemos perdido tanta información de la serie temporal original, pero sí hemos eliminado ciertas perturbaciones que pudieran ser debidas a ruido en los detectores (marcadas en rojo).

Esta función surge a partir del índice PCA. Este índice se basa en la observación de que podemos obtener una aproximación muy buena si segmentamos la serie en tramos iguales y guardamos el valor de la media de valores de ese tramo en un vector auxiliar. En la llamada a la función podemos especificar la longitud de los tramos que queremos emplear en la segmentación. Si no se da un valor, la función toma 15 tramos por defecto. Este valor habrá que tenerlo muy en cuenta si la serie temporal es muy amplia, o si aparecen muchos cruces por cero en las series.

Llamada a la función:

kdapproxPAA <- function (TSerie,len=length(TSerie)/15)

Parámetros de Entrada:

- *TSerie*=Serie temporal normal.
- *len* = Longitud de los segmentos.

Parámetros de Salida: Lista con la serie temporal filtrada con el indice PAA.

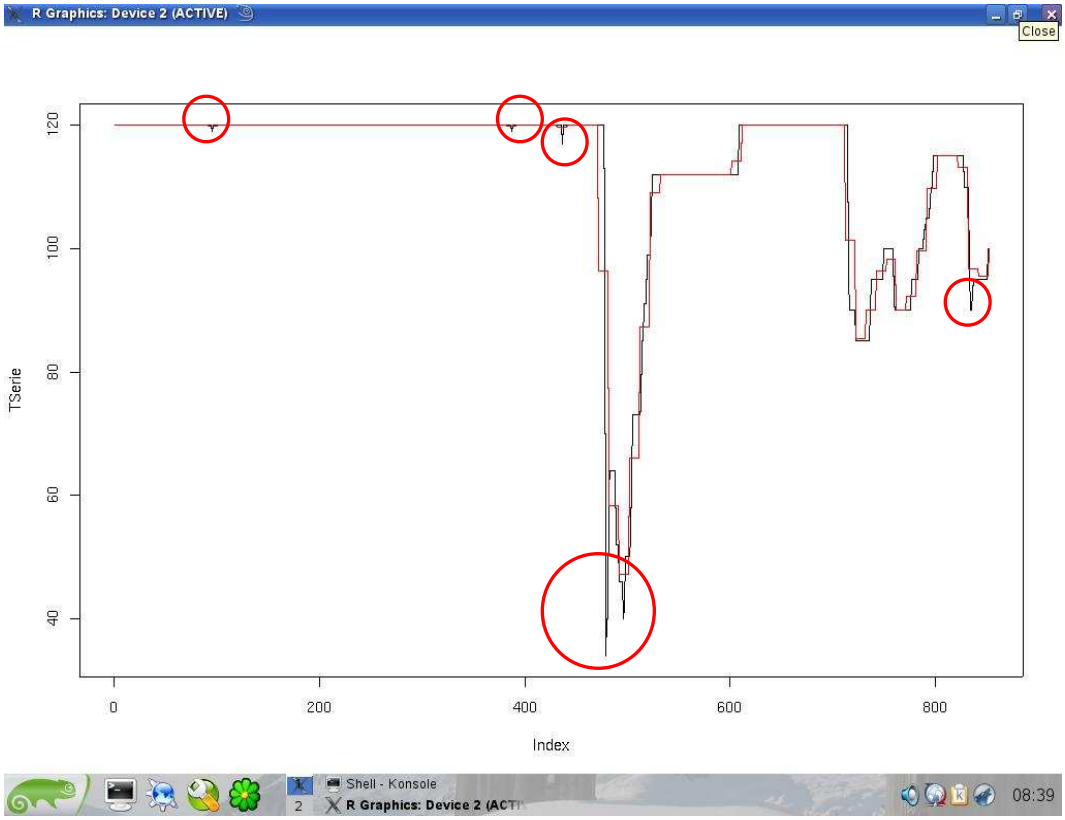


Figura 23. Filtrado de una serie temporal usando el índice PAA.

```
#########################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.


#########################################################################
# (kdapproxPAA.R) Constant approximation by frames (mean
approximation).
#########################################################################

# INPUT PARAMETERS:
# TSerie=Time Series
# len=length of sections
# ------------------------------------------------------------------------

# OUTPUT PARAMETERS:
# SerieFilt=Time Series filtered
# ------------------------------------------------------------------------

kdapproxPAA <- function (TSerie,len=length(TSerie)/15)
{
```

```
SerieFilt<-c(1,2)
i<-1
position<-1
aux<-1
max<-position+len
while (position<length(TSerie))
{
      min<-position
      aux<-TSerie[min:max]
      aux1<-mean(aux)
      while(i<=max)
      {
            SerieFilt[i]<-aux1
            i<-i+1
      }
      position<-max
      max<-position+len

}
      min<-position
      max<-length(TSerie)
      aux<-TSerie[min:max]
      aux1<-mean(aux)
      SerieFilt[i]<-aux1
      plot(TSerie,type="l")
      lines(SerieFilt,type="l",col="red")

      return(SerieFilt)

}
```

I.18. Función “kdapproxAPCA”: Comprime una serie temporal usando una mejora del indice PCA

Esta función comprime la información de la serie temporal almacenando la media de la serie por tramos. La longitud de estos tramos depende de la cantidad de máximos y mínimos relativos que aparecen en la serie. La función devuelve dos vectores que almacenan, uno el valor medio de los tramos y el otro el número de posiciones que ocupa (longitud de los tramos). Una llamada y una salida típica de la función kdapproxAPCA:

```
> kdapproxAPCA(COILS$VEL[1:2000])
$Values
 [1] 145.00000 144.50000 144.97959 118.75000 140.19565 137.50000
130.00000
 [8] 119.00000 118.60000 107.33333 124.32584 133.00000 120.00000
115.00000
[15] 110.00000 105.66667 100.00000  90.50000  84.09524  74.33333...
$Number_positions
 [1]  35   1  48   3  45   1   2   1   4   2  88   2   4   1   6   2
18   3 104
[20]   2   2   3 447   1  33   2 124   1 108   1 290   1  49...
[[3]]
   [1] 145.00000 145.00000 145.00000 145.00000
```

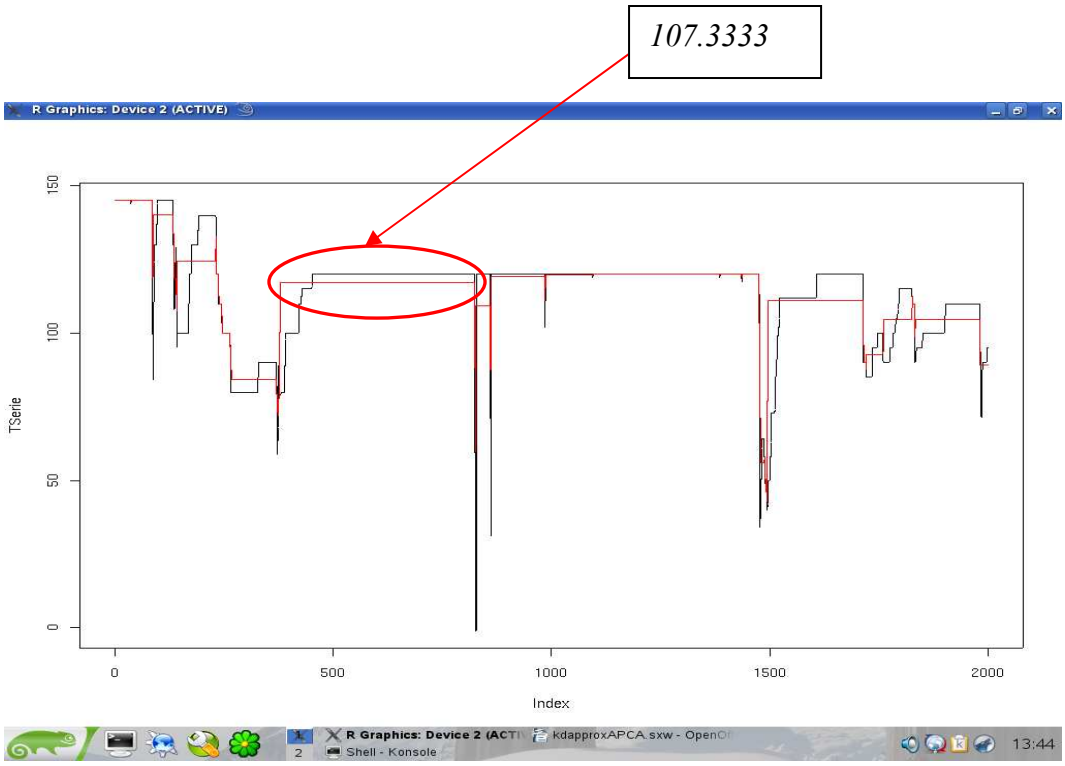


Figura 24. Filtrado de una serie temporal usando el índice PAA. Observamos el negro la serie original y en rojo la serie comprimida. Se observa una pérdida de información pero se consigue una compresión muy buena.

Llamada a la función:

kdapproxAPCA <- function (TSerie,len=length(TSerie)/15)

Parámetros de Entrada:

- *TSerie*=Serie temporal normal.
- *len* = Longitud de los segmentos.

Parámetros de Salida: Lista con la serie temporal comprimida.

Esta función es simplemente una mejora de la función PCA. Esto permite que en las zonas donde la serie temporal presenta mayores irregularidades, podamos tener una mayor precisión en los datos. Por ejemplo, si durante un día entero no ha variado prácticamente la temperatura de un horno, podremos ahorrarnos gran cantidad de datos que son de gran ayuda.

Para ilustrar esto vamos a fijarnos en el ejemplo que se ha presentado aparecen dos vectores llamados values y positions, que almacenan respectivamente los valores medios y el número de posiciones que ocupan. Por ejemplo, el valor 107.3333 se repite durante 88 posiciones, con lo que en vez de almacenar 88 veces ese valor, utilizamos dos posiciones de memoria para el propósito.

```
#########################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#########################################################################
# (kdapproxAPCA.R) returns a reduced representation of the data.The data have been
divided into
# variable length frames.
#########################################################################

# INPUT PARAMETERS:
# TSerie=Time Series
# len=length of sections
# ------------------------------------------------------------------------------
# OUTPUT PARAMETERS:
# returned_list=reduced series and lengths.
# ------------------------------------------------------------------
-----------------------

kdapproxAPCA <- function (TSerie,len=length(TSerie)/15)
{
storage.mode(TSerie) <- "double"
```

```
zeros<-.Call("zerocrossings",TSerie,PACKAGE="KDSeries")
zerospos<-grep(1,zeros)

values<-c(1,2)
positions<-c(1,2)
i2<-1

SerieFilt<-c(1,2)
i<-1
i1<-1
position<-1
aux<-1
aux1<-c(1,2)
max<-zerospos[i1]
i1<-i1+1
not<-0
while ((position+len)<length(TSerie))
{
      min<-position
      aux<-TSerie[min:max]
      aux1<-mean(aux)
      values[i2]<-aux1
      positions[i2]<-max-min
      i2<-i2+1
      while(i<=max)
      {
            SerieFilt[i]<-aux1
            i<-i+1
      }
      position<-max
      if (not==0) max<-zerospos[i1]
      if ((not==1)&((position+len)<length(TSerie))) max<-position+len
      i1<-i1+1
      if (i1>length(zerospos)) not<-1
}
      min<-position
      max<-length(TSerie)
      aux<-TSerie[min:max]
      aux1<-mean(aux)
      values[i2]<-aux1
      positions[i2]<-max-min
      while(i<=max)
      {
            SerieFilt[i]<-aux1
            i<-i+1
      }
      plot(TSerie,type="l")
      lines(SerieFilt,type="l",col="red")
      list_return<-
list(Values=values,Number_positions=positions,SerieFilt)
      return(list_return)

}
```

I.19. Función “kdextractSubPatt”: Extrae subpatrones de una serie temporal

Permite extraer subpatrones de series temporales configurables por el usuario.

Llamada a la función:

kdextractsubpatt <- function(TSerie, pattType, pattRangeX=NULL, pattRangeY=NULL, rangeType=c("N","N"), threshold=NULL, level=NULL, namePatt)

Parámetros de Entrada:

- *TSerie*=Serie temporal filtrada.
- *pattType*= Tipo de subpatrón a buscar: Incremental ("I"), decremental ("D"), horizontal ("H") or threshold ("T")
- *pattRangeX*= Rango formado por un vector de dos valores c(mínimo, máximo) de X en los que tiene que estar comprendido el subpatrón.
- *pattRangeY*= Rango formado por un vector de dos valores c(mínimo, máximo) de Y en los que tiene que estar comprendido el subpatrón.
- *rangeType*= Si se consideran los rangos de X e Y por valores fijos (N) o porcentajes (P). Por ejemplo, un vector c(“N”,”P”) considera los rangos de X por valores fijos y los de Y por porcentajes.
- *threshold*= Valores de corte en donde buscar los patrones. Por defecto es NULL.
- *level*= Los patrones se buscarán por encima ("+"), debajo ("-") or entre dos valores ("+-") de threshold. Por defecto es NULL.
- *namePatt*= Nombre del patron encontrado.

Parámetros de Salida:.

- MATPatt= Matriz con los patrones encontrados [col1=Posición, col2=longitud, col3=altura, col4=Nombre del Patron]

Ejemplo:

```
> library(KDSeries)
> data(COILS)
>
> # Preprocessing Temp 2
> DATA <- COILS$TMP2M
> plot.ts(DATA)
> DATA2 <- kdfilterremove(DATA,200)
> plot.ts(DATA2[1:1000])
> DATA3 <- kdfilter(DATA2,20)
> plot.ts(DATA3[1:1000])
> #I <-
kdextractsubpatt(DATA3,"I",pattRangeY=c(50,100),namePatt="INC1")
> I <-
kdextractsubpatt(DATA3,"I",pattRangeY=c(50,100),namePatt="INC1",thresh
old=c(780,800),level="+-")
> I$PATT[1:10,]
    PosP LongP     PosY     AltP namePatt
1    195    52 750.7536 79.88099     INC1
```

```
2   1497    43 759.9018 70.23312     INC1
3   1798    52 725.4275 90.68528     INC1
4   4236    33 799.3201 55.68223     INC1
5   4324    50 793.9364 60.25899     INC1
6   4514    21 747.6423 52.64799     INC1
7   8099    35 734.4793 88.29717     INC1
8  13202    38 773.7443 54.69858     INC1
9  13352    22 773.8036 73.10013     INC1
10 16472    32 751.0924 80.25870     INC1
>
segments(x0=I$PATT[,1],y0=I$PATT[,3],x1=I$PATT[,1]+I$PATT[,2],y1=I$PAT
T[,3]+I$PATT[,4],col="red")
> D <-
kdextractsubpatt(DATA3,"D",pattRangeY=c(50,100),namePatt="DEC1")
> D$PATT[1:10,]
    PosP LongP     PosY      AltP namePatt
1    954    30 841.1657 -54.25534     DEC1
2   1574    24 826.4428 -58.58808     DEC1
3   1729    26 774.1969 -57.12766     DEC1
4   2038    35 824.5869 -50.62410     DEC1
5   3941    29 827.9818 -68.61295     DEC1
6   4269    34 855.0023 -58.39322     DEC1
7   4422    36 826.8562 -76.86638     DEC1
8   8578    47 834.3368 -63.98063     DEC1
9  13240    37 828.4429 -65.27953     DEC1
10 14101    22 850.4298 -62.70058     DEC1
>
segments(x0=D$PATT[,1],y0=D$PATT[,3],x1=D$PATT[,1]+D$PATT[,2],y1=D$PAT
T[,3]+D$PATT[,4],col="blue")
>
> H <-
kdextractsubpatt(DATA3,"H",pattRangeX=c(20,1000),pattRangeY=c(0,10),na
mePatt="HOR1")
> H$PATT[1:10,]
   PosP LongP     PosY     AltP namePatt
1   334    34 820.7827 5.547854     HOR1
2   429    27 776.9736 9.991085     HOR1
3   664    23 802.0000 0.000000     HOR1
4   687    23 802.0000 1.000000     HOR1
5   710    26 803.0000 0.000000     HOR1
6  1041    21 789.4338 3.473113     HOR1
7  1182    23 788.0421 2.435395     HOR1
8  1438    24 784.2028 3.690147     HOR1
9  2178    20 773.2475 5.233391     HOR1
10 2240    41 772.3590 2.641050     HOR1
>
segments(x0=H$PATT[,1],y0=H$PATT[,3],x1=H$PATT[,1]+H$PATT[,2],y1=H$PAT
T[,3]+H$PATT[,4],col="magenta")
>
>
>
>
>
> MATTOTAL <- rbind(D$PATT,I$PATT, H$PATT)
>
```

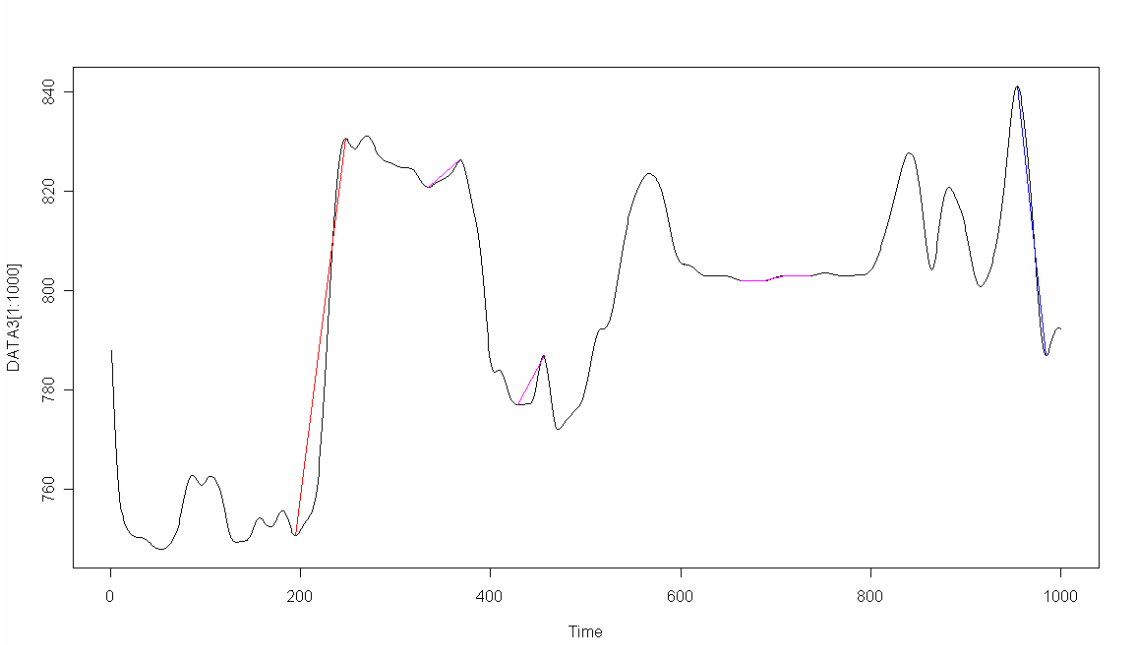


Figura 25. Subpatrones obtenidos Incrementales, Horizontales y Decrementales de una serie temporal filtrada.

```
#########################################################################
# Written by: EDMANS (Engineering Data Mining And Numerical
Simulations) Research GROUP.
# License: GPL version 2 or newer
# ------------------------------------------------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ------------------------------------------------------------------
# Acknowledgments: The authors thank the Dirección General de
Investigación of the Spanish Ministry of Science and Technology for
the financial support of the projects DPI2004-07264-C02-01 and
DPI2006-03060; and the European Union for the projects  RFS-CR-03012 y
RFS-CR-04023.
Finally, the authors also thank the Autonomous Government of La Rioja
for its support through the 2º Plan Riojano de I+D+i.

#########################################################################
# (kdextractSubPatt.R) Extract Segments from Time Series Vector
Filtered
#########################################################################
# INPUT PARAMETERS:
# TSerie=Time Series (Usually filtered)

# pattType= Type of segment to extract: incremental ("I"), decremental
("D"), horizontal ("H") or threshold ("T")
# pattRangeX= Minimum and maximum Delta(x) values to consider a
pattern
# pattRangeY= Minimum and maximum Delta(y) values to consider a
pattern
# rangeType= Vector with the type range of x and y: fixed number ("N")
or percentage ("P"). The default is c("N","N")
# threshold= Threshold value to search patterns. The default is NULL
```

```
# level= Patterns will be search above ("+"), below ("-") or between
("+-") the threshold values. The default is NULL.
# namePatt= Name of the searched segments
# ------------------------------------------------------------------
-----------------------

# OUTPUT PARAMETERS:
# MATPatt=Patterns found matrix:
#          [col1=Position, col2=length, col3=height, col4=NamePatt]
# ------------------------------------------------------------------
-----------------------

kdextractsubpatt <- function(TSerie, pattType, pattRangeX=NULL,
pattRangeY=NULL, rangeType=c("N","N"), threshold=NULL, level=NULL,
namePatt)
{
      storage.mode(TSerie) <- "double"
      SerieP <- list(TSerie=TSerie)

      # Scale between 0 and 1
      MinTSerie = min(TSerie)
      RangeTSerie=max(TSerie)-min(TSerie)
      TSerieNorm <- (TSerie-MinTSerie)/RangeTSerie

      # Obtains Zeros of First Derivate
      Zeros <- .Call("zerocrossings",TSerieNorm, PACKAGE="KDSeries")
      Zeros[length(Zeros)] <- 1

      # Where is Zeros?
      WhereZeros <- 1:length(Zeros)
      WhereZeros <- WhereZeros[Zeros %in% 1]

      # Get Long, Height and Position
      PosicionPat <- c(1,WhereZeros[-length(WhereZeros)])
      LongPat <- WhereZeros-PosicionPat
      PosicionY <- TSerie[PosicionPat]
      AltPat <- TSerie[WhereZeros]- TSerie[PosicionPat]


      if (pattType=="I") {
            Cuales <- AltPat>0

            # Remove Segments outside the threshold
            if (!is.null(threshold) & !is.null(level)) {
                  if (level=="+") {
                        Cuales <- Cuales &
(PosicionY+AltPat)>=threshold
                  } else if (level=="-") {
                        Cuales <- Cuales & (PosicionY)<=threshold
                  } else if (level=="+-") {
                        Cuales <- Cuales &
(PosicionY+AltPat)>=threshold[1] & (PosicionY)<=threshold[2]
                  }
            }

            PosP <- PosicionPat[Cuales]
            LongP <- LongPat[Cuales]
            PosY <- PosicionY[Cuales]
            AltP <- AltPat[Cuales]

            # Remove Segments outside the x-y range
            if (!is.null(pattRangeX)) {
                  if (rangeType[1] == "N") {
                        XRange <- pattRangeX
```

```
                  } else {
                        XRange <- (pattRangeX/100) * length(TSerie)
                  }

                  PosP <- PosP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  PosY <- PosY[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  AltP <- AltP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  LongP <- LongP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
            }
            if (!is.null(pattRangeY)) {
                  if (rangeType[2] == "N") {
                        # Scale range
                        YRange <- pattRangeY
                  } else {
                        YRange <- pattRangeY*RangeTSerie/100
                  }

                  PosP <- PosP[which((AltP>=YRange[1] &
AltP<=YRange[2]))]
                  LongP <- LongP[which((AltP>=YRange[1] &
AltP<=YRange[2]))]
                  PosY <- PosY[which((AltP>=YRange[1] &
AltP<=YRange[2]))]
                  AltP <- AltP[which((AltP>=YRange[1] &
AltP<=YRange[2]))]
            }


            namePatt <- rep(namePatt,length(LongP))
            PATT <- data.frame(PosP, LongP, PosY, AltP, namePatt)
            SerieP <- list(SerieP=TSerie, PATT=PATT)

      } else if (pattType=="D") {
            Cuales <- AltPat<0

            if (!is.null(threshold) & !is.null(level)) {
                  if (level=="+") {
                        Cuales <- Cuales & (PosicionY)>=threshold
                  } else if (level=="-") {
                        Cuales <- Cuales &
(PosicionY+AltPat)<=threshold
                  } else if (level=="+-") {
                        Cuales <- Cuales & (PosicionY)>=threshold[1] &
(PosicionY+AltPat)<=threshold[2]
                  }
            }

            PosP <- PosicionPat[Cuales]
            LongP <- LongPat[Cuales]
            PosY <- PosicionY[Cuales]
            AltP <- AltPat[Cuales]

            # Remove Segments outside the x-y range
            if (!is.null(pattRangeX)) {
                  if (rangeType[1] == "N") {
                        XRange <- pattRangeX
                  } else {
                        XRange <- (pattRangeX/100) * length(TSerie)
                  }
```

```
                  PosP <- PosP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  PosY <- PosY[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  AltP <- AltP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  LongP <- LongP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
            }
            if (!is.null(pattRangeY)) {
                  if (rangeType[2] == "N") {
                        # Scale range
                        YRange <- abs(pattRangeY)
                  } else {
                        YRange <- abs(pattRangeY)*RangeTSerie/100
                  }

                  PosP <- PosP[which((abs(AltP)>=YRange[1] &
abs(AltP)<=YRange[2]))]
                  LongP <- LongP[which((abs(AltP)>=YRange[1] &
abs(AltP)<=YRange[2]))]
                  PosY <- PosY[which((abs(AltP)>=YRange[1] &
abs(AltP)<=YRange[2]))]
                  AltP <- AltP[which((abs(AltP)>=YRange[1] &
abs(AltP)<=YRange[2]))]
            }

            namePatt <- rep(namePatt,length(LongP))
            PATT <- data.frame(PosP, LongP, PosY, AltP, namePatt)
            SerieP <- list(SerieP=TSerie, PATT=PATT)

      } else if (pattType=="H") {
            if (!is.null(pattRangeY)) {
                  if (rangeType[2] == "N") {
                        # Scale range
                        YRange <- pattRangeY
                  } else {
                        YRange <- pattRangeY*RangeTSerie/100
                  }
                  Cuales <- AltPat>=YRange[1] & AltPat<=YRange[2]
            } else {
                  Cuales <- AltPat == 0
            }

            # Remove Segments outside the threshold
            if (!is.null(threshold) & !is.null(level)) {
                  if (level=="+") {
                        Cuales <- Cuales &
(PosicionY+AltPat)>=threshold
                  } else if (level=="-") {
                        Cuales <- Cuales & (PosicionY)<=threshold
                  } else if (level=="+-") {
                        Cuales <- Cuales &
(PosicionY+AltPat)>=threshold[1] & (PosicionY)<=threshold[2]
                  }
            }

            PosP <- PosicionPat[Cuales]
            LongP <- LongPat[Cuales]
            PosY <- PosicionY[Cuales]
            AltP <- AltPat[Cuales]

            # Remove Segments outside the x-y range
            if (!is.null(pattRangeX)) {
```

```
                  if (rangeType[1] == "N") {
                        XRange <- pattRangeX
                  } else {
                        XRange <- (pattRangeX/100) * length(TSerie)
                  }

                  PosP <- PosP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  PosY <- PosY[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  AltP <- AltP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  LongP <- LongP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
            }

            namePatt <- rep(namePatt,length(LongP))
            PATT <- data.frame(PosP, LongP, PosY, AltP, namePatt)
            SerieP <- list(SerieP=TSerie, PATT=PATT)

      } else if (pattType=="T") {
            if (!is.null(threshold) & !is.null(level)) {
                  if (level=="+") {
                        TSerieThresh <- TSerie - threshold
                        TSerieThresh[TSerieThresh<0] <- 0
                        TSerieThresh[TSerieThresh>0] <- 1
                  } else if (level=="-") {
                        TSerieThresh <- TSerie - threshold
                        TSerieThresh[TSerieThresh>0] <- 0
                        TSerieThresh[TSerieThresh<0] <- 1
                  } else if (level=="+-") {
                        TSerieThresh <- TSerie - threshold[1]
                        TSerieThresh2 <- TSerie - threshold[2]
                        TSerieThresh[TSerieThresh<0 & TSerieThresh2>0]
<- 0
                        TSerieThresh[TSerieThresh2<0 & TSerieThresh>0]
<- 1
                  }
                  # Where is Ones?
                  WhereOnes <- 1:length(TSerieThresh)
                  WhereOnes <- WhereOnes[TSerieThresh %in% 1]

                  longs <- WhereOnes[-1]-WhereOnes[-length(WhereOnes)]
                  WhereLongs <- which(longs>1)
                  PosP <- c(WhereOnes[c(1,WhereLongs+1)])
                  LongP <- c(WhereLongs[1],WhereLongs[-1] -
WhereLongs[-length(WhereLongs)],length(WhereOnes)-
WhereLongs[length(WhereLongs)])


                  # Remove Segments outside the x-y range
                  if (!is.null(pattRangeX)) {
                        if (rangeType[1] == "N") {
                              XRange <- pattRangeX
                        } else {
                              XRange <- (pattRangeX/100) *
length(TSerie)
                        }

                        PosP <- PosP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                        LongP <- LongP[which((LongP>=XRange[1] &
LongP<=XRange[2]))]
                  }
```

```
                PosY <- rep(threshold[1],length(LongP))
                AltP <- rep(0,length(LongP))
                namePatt <- rep(namePatt,length(LongP))
                PATT <- data.frame(PosP, LongP, PosY, AltP, namePatt)
                SerieP <- list(SerieP=TSerie, PATT=PATT)
            }
        }
        return(SerieP)
}
```

I.20. Función “kdsearchpatt”: Busca combinación de subpatrones

Permite extraer patrones de una matriz de subpatrones buscando secuencias de ellos dentro de una ventana de ancho configurable por el usuario.

Llamada a la función:

kdsearchpatt <- function(MAT, SubPatterns, WinW, namePatt, Plot=FALSE, SerieP=0,Xlim=c(1,length(SerieP)))

Parámetros de Entrada:

- *MAT*= Matriz obtenida de la unión de varias matrices de patrones de la función kdextractsubpatt.
- *SubPatterns* =Secuencia de patrones. Permite el uso de sintaxis del comando “grep" en formato Perl.
- *WinW*= Ancho de la ventana de búsqueda.
- *Plot*= Si es TRUE dibuja la serie temporal y los patrones encontrados.
- *SerieP*= Serie para dibujar si Plot=TRUE.
- *Xlim*= Límites mínimo y máximo para dibujar.
- *namePatt*= Nombre del patron encontrado.
-

Parámetros de Salida:.

- PATTERN=Matriz de patrones encontrados.
- MAT2=Matriz de combinación de subpatrones encontrados.

Ejemplo:

```
library(KDSeries)
data(COILS)

# Preprocessing Temp 2
DATA <- COILS$TMP2M
plot.ts(DATA)
DATA2 <- kdfilterremove(DATA,200)
plot.ts(DATA2[1:1000])
DATA3 <- kdfilter(DATA2,20)
plot.ts(DATA3[1:1000])
#I <- kdextractsubpatt(DATA3,"I",pattRangeY=c(50,100),namePatt="INC1")
I <- kdextractsubpatt(DATA3,"I",pattRangeY=c(50,100),namePatt="INC1",
threshold=c(780,800),level="+-")
I$PATT[1:10,]
segments(x0=I$PATT[,1],y0=I$PATT[,3],x1=I$PATT[,1]+I$PATT[,2],
y1=I$PATT[,3]+I$PATT[,4],col="red")
D <- kdextractsubpatt(DATA3,"D",pattRangeY=c(50,100), namePatt="DEC1")
D$PATT[1:10,]
segments(x0=D$PATT[,1],y0=D$PATT[,3],x1=D$PATT[,1]+D$PATT[,2],y1=D$PAT
T[,3]+D$PATT[,4],col="blue")

H <-
kdextractsubpatt(DATA3,"H",pattRangeX=c(20,1000),pattRangeY=c(0,10),na
mePatt="HOR1")
H$PATT[1:10,]
```

```
segments(x0=H$PATT[,1],y0=H$PATT[,3],x1=H$PATT[,1]+H$PATT[,2],y1=H$PAT
T[,3]+H$PATT[,4],col="magenta")


MATTOTAL <- rbind(D$PATT,I$PATT, H$PATT)
PATRONES <-
kdsearchpatt(MATTOTAL,SubPatterns=c("INC1","DEC1"),WinW=150,
namePatt="INC_DEC", Plot=TRUE, SerieP=H$SerieP,Xlim=c(4000,5000))

> PATRONES
$PATTERN
   PosP LongP namePatt
1  1497   101  INC_DEC
2  4236    67  INC_DEC
3  4324   134  INC_DEC
4 13202    75  INC_DEC
5 35407   127  INC_DEC
6 36786   136  INC_DEC
7 51662    76  INC_DEC

$MAT2
    PosP LongP     PosY     AltP namePatt PosP.1 LongP.1   PosY.1
AltP.1
32  1497    43 759.9018 70.23312     INC1   1574      24 826.4428 -
58.58808
34  4236    33 799.3201 55.68223     INC1   4269      34 855.0023 -
58.39322
35  4324    50 793.9364 60.25899     INC1   4422      36 826.8562 -
76.86638
38 13202    38 773.7443 54.69858     INC1  13240      37 828.4429 -
65.27953
50 35407    79 762.2538 50.93327     INC1  35500      34 814.2895 -
61.41658
53 36786    54 757.8621 71.04530     INC1  36872      50 824.0203 -
62.84630
60 51662    24 784.2694 51.49803     INC1  51717      21 819.5083 -
63.26126
   namePatt.1 TOTALW
32       DEC1    101
34       DEC1     67
35       DEC1    134
38       DEC1     75
50       DEC1    127
53       DEC1    136
60       DEC1     76
```

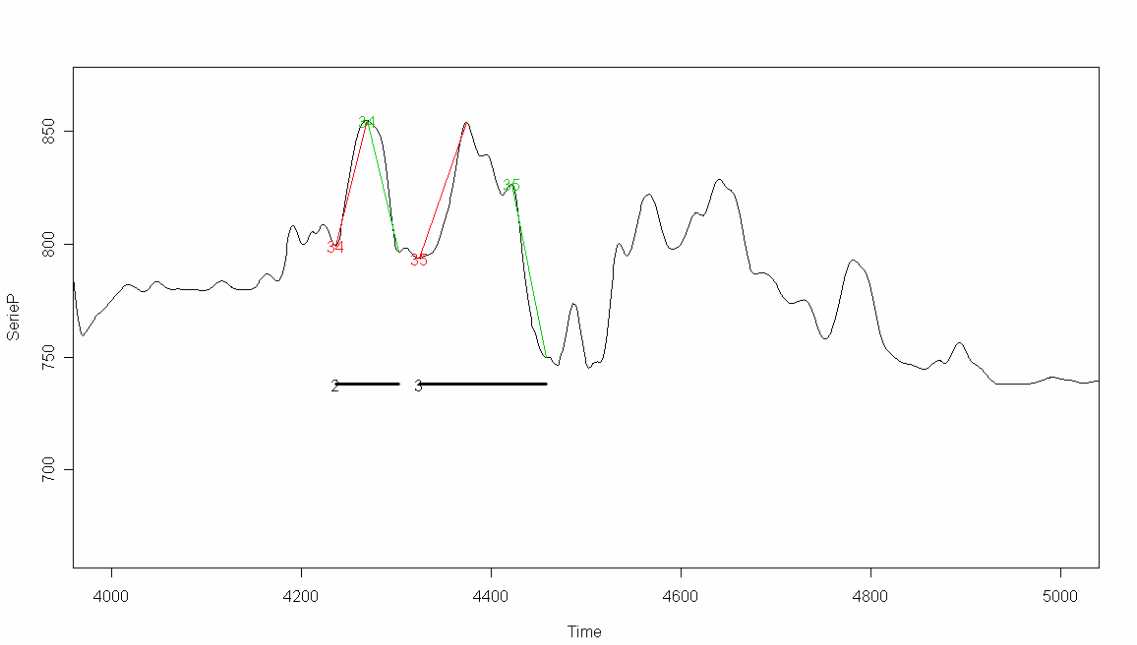


Figura 26. Dos patrones INC+DEC obtenidos en una zona de una serie temporal.

```
##############################################################################
#########################
# Written by: Dr. Francisco Javier Martínez de Pisón Ascacíbar
(fjmartin@dim.unirioja.es) (2005)
# EDMANS (Engineering Data Mining And Numerical Simulations) Research
GROUP.
# License: GPL version 2 or newer
# ----------------------------------------------------------------------
------------------------
# Project Engineering Group
# Department of Mechanical Engineering. C/ Luis de Ulloa, 20.
# 26004. Universidad de La Rioja. Spain
# ----------------------------------------------------------------------
------------------------
# EDMANS Members= Francisco Javier Martínez de Pisón Ascacibar
(fjmartin@dim.unirioja.es), Ana González Marcos
(ana.gonzalez@unirioja.es),
# ,Manuel Castejón Limas(manuel.castejon@unileon.es),
# ,Alpha V. Pernía Espinoza (alpha.veronica@dim.unirioja.es), Joaquín
B. Ordieres Meré (joaquin.ordieres@dim.unirioja.es),
# ,Eliseo P. Vergara González (eliseo.vergara@dim.unirioja.es),
Fernando Alba Elías (fernando.alba@dim.unirioja.es)
# ,
##############################################################################
#########################

##############################################################################
#########################
# (kdsearchpatt.R) Search Patterns from Subppatern's Matrix
##############################################################################
#########################

# INPUT PARAMETERS:
# MAT=rbind of severals MATRIX from kdextractsubpatt.R

# SubPatterns= Grep subpatterns match (see ?regexp and ?grep with
perl=TRUE)
# WinW= Maximum windows windth to search a pattern
# namePatt= Name of the searched segments
```

```
# Plot= if TRUE plots SerieP with Patterns found
# SerieP= This is the time Series to plot
# Xlim=c(MinX,MaxX) to plot
# ------------------------------------------------------------------------
-----------------------

# OUTPUT PARAMETERS (list):
# PATTERN=Patterns found matrix
# MAT2=SubPatterns combinations matrix
# ------------------------------------------------------------------------
-----------------------



kdsearchpatt <- function(MAT, SubPatterns, WinW, namePatt, Plot=FALSE,
SerieP=0,Xlim=c(1,length(SerieP)))
{
      # Order subpatterns
      MAT <- MAT[order(MAT[,1]),]
      NUMC <- length(SubPatterns)

      if (nrow(MAT)<=NUMC) return;
      if (nrow(MAT)<1) return;
      if (NUMC<1) return;
      if (WinW<1) return;

      # Fuse N sub patterns
      FILAS <- nrow(MAT)
      MAT2 <- MAT[1:(1+FILAS-NUMC),]
      for (h in 2:NUMC)
            {
            MAT2 <- data.frame(MAT2,MAT[h:(h+FILAS-NUMC),])
            }

      TOTALW <- MAT2[,ncol(MAT2)-4]+     MAT2[,ncol(MAT2)-3]-MAT2[,1]
      MAT2 <- data.frame(MAT2,TOTALW=TOTALW)

      # Remove window's size > WinW
      MAT2 <- MAT2[MAT2$TOTALW<=WinW,]

      # Search patterns
      CUALES <- rep(TRUE,nrow(MAT2))
      for (h in 1:NUMC)
            {
            MASCARA <- rep(FALSE,nrow(MAT2))
            MASCARA[grep(SubPatterns[h],MAT2[,h*5],perl=TRUE)] <- TRUE
            CUALES <- CUALES & MASCARA
            }
      MAT2 <- MAT2[CUALES,]
      if (nrow(MAT2)<1) return (list(PATTERN=NULL,MAT2=NULL))
      PATTERN <-
data.frame(PosP=MAT2[,1],LongP=MAT2$TOTALW,namePatt=namePatt)
      if (Plot==TRUE && SerieP!=0)
            {
            plot.ts(SerieP,xlim=Xlim)

            for (h in 0:(NUMC-1))
                  {

      segments(x0=MAT2[,1+5*h],y0=MAT2[,3+5*h],x1=MAT2[,1+5*h]+MAT2[,2
+5*h],y1=MAT2[,3+5*h]+MAT2[,4+5*h],col=h+2)

      segments(x0=PATTERN[,1],y0=min(SerieP[Xlim[1]:Xlim[2]]),x1=PATTE
RN[,1]+PATTERN[,2],y1=min(SerieP[Xlim[1]:Xlim[2]]),col=1,lwd=3)
```

```
      text(MAT2[,1+5*h],MAT2[,3+5*h],rownames(MAT2),col=h+2)

      text(PATTERN[,1],min(SerieP[Xlim[1]:Xlim[2]]),row.names(PATTERN)
)

                  }
            }

      return(list(PATTERN=PATTERN,MAT2=MAT2))
}
```

I.21. Programa “kdfilter.c” para filtrado de ST.

```
#include <string.h>
#include <R.h>
#include <S.h>
#include <Rinternals.h>
#include <Rdefines.h>
#include <math.h>
/*index of importantpoints functions*/
int q,cont,last;
SEXP positions,auxpos;
/***************
Functions*********************************************/

SEXP out(SEXP x, SEXP y)
      {
      int i, j, nx, ny;
      double tmp;
      SEXP ans;
      nx = length(x); ny = length(y);
      PROTECT(ans = allocMatrix(REALSXP, nx, ny));
      for(i = 0; i < nx; i++)
            {
            tmp = REAL(x)[i];
            for(j = 0; j < ny; j++) REAL(ans)[i + nx*j] = tmp *
REAL(y)[j];
            }
      UNPROTECT(1);
      return(ans);
}

/* Obtain where are zerocrossings in a Time Series */
SEXP zerocrossings(SEXP Vect)
{
      int i, LV;
      double Slope1, Slope2;
      SEXP VectResult;
      LV = length(Vect);
      PROTECT(VectResult=allocVector(INTSXP, LV));
      INTEGER(VectResult)[0]=0;
      INTEGER(VectResult)[LV-1]=0;
      for (i=1; i<(LV-1); i++)
            {
                  Slope1=REAL(Vect)[i]-REAL(Vect)[i-1];
                  Slope2=REAL(Vect)[i+1]-REAL(Vect)[i];
                  /* INTEGER(VectResult)[i]=((Slope1<0.0) &
(Slope2>=0.0)) | ((Slope1>=0.0) & (Slope2<0.0)); */
                  INTEGER(VectResult)[i]=((Slope1<0.0) & (Slope2>=0.0))
| ((Slope1>=0.0) & (Slope2<0.0)) |
                  ((Slope1==0.0) & (Slope2>0.0)) | ((Slope1>0.0) &
(Slope2==0.0));
            }
      UNPROTECT(1);
      return(VectResult);
}

/* Make a median filter in a Time Series */
SEXP medianfilter(SEXP Vect, SEXP NumW)
{
      int h,i,j,LV,min,posi,LW;
      double temporal;
```

```
	double VectWin[(int)REAL(NumW)[0]];
	LV = length(Vect);
	SEXP VectResult;
	LW=(int)REAL(NumW)[0];
	PROTECT(VectResult=allocVector(REALSXP, LV));

	for (h=0;h<LV;h++)
		{
			/* Put elements into the slide window */
			for(i=0;i<LW;i++)
			{
				posi=i+h-LW/2;
				if (posi>=LV) posi=posi-LV;
				if (posi<0) posi=LV+posi;
				VectWin[i]=REAL(Vect)[posi];
			}

			/* Sorts slide window's elements using "Selection
Sort" Algorithm */
			for(i=0; i<(LW-1); i++)
				{
				min=i;
				for(j=i+1; j<LW; j++)
					{
					if(VectWin[j] < VectWin[min]) min = j;
					}
				temporal=VectWin[i];
				VectWin[i]=VectWin[min];
				VectWin[min]=temporal;
				}

		/* Save median element */
		if (LW%2==1)	REAL(VectResult)[h]=VectWin[LW/2];
		else REAL(VectResult)[h]=(VectWin[(LW/2)-
1]+VectWin[LW/2])/2.0;
		}
	UNPROTECT(1);
	return(VectResult);
}

/* Make a min filter in a Time Series */
SEXP minfilter(SEXP Vect, SEXP NumW)
{
	int h,i,LV,posi,LW;
	double min;
	LV = length(Vect);
	SEXP VectResult;
	LW=(int)REAL(NumW)[0];
	PROTECT(VectResult=allocVector(REALSXP, LV));

	for (h=0;h<LV;h++)
		{
			/* Search min element in window */
			posi=h-LW/2;
			if (posi>=LV) posi=posi-LV;
			if (posi<0) posi=LV+posi;
			min=REAL(Vect)[posi];

			for(i=1;i<LW;i++)
			{
				posi=i+h-LW/2;
				if (posi>=LV) posi=posi-LV;
				if (posi<0) posi=LV+posi;
				if (REAL(Vect)[posi]<min) min=REAL(Vect)[posi];
```

```
                }
                /* Save min element */
                REAL(VectResult)[h]=min;
          }
      UNPROTECT(1);
      return(VectResult);
}

/* Make a max filter in a Time Series */
SEXP maxfilter(SEXP Vect, SEXP NumW)
{
      int h,i,LV,posi,LW;
      double max;
      LV = length(Vect);
      SEXP VectResult;
      LW=(int)REAL(NumW)[0];
      PROTECT(VectResult=allocVector(REALSXP, LV));

      for (h=0;h<LV;h++)
            {
                /* Search min element in window */
                posi=h-LW/2;
                if (posi>=LV) posi=posi-LV;
                if (posi<0) posi=LV+posi;
                max=REAL(Vect)[posi];

                for(i=1;i<LW;i++)
                {
                      posi=i+h-LW/2;
                      if (posi>=LV) posi=posi-LV;
                      if (posi<0) posi=LV+posi;
                      if (REAL(Vect)[posi]>max) max=REAL(Vect)[posi];
                }
                /* Save min element */
                REAL(VectResult)[h]=max;
          }
      UNPROTECT(1);
      return(VectResult);
}

/* Makes a mean filter in a Time Series */
SEXP meanfilter(SEXP Vect, SEXP NumW)
{
      int h,i,LV,posi,LW;
      double mean;
      LV = length(Vect);
      SEXP VectResult;
      LW=(int)REAL(NumW)[0];
      PROTECT(VectResult=allocVector(REALSXP, LV));

      for (h=0;h<LV;h++)
            {
                /* Search min element in window */
                mean=0.0;

                for(i=0;i<LW;i++)
                {
                      posi=i+h-LW/2;
                      if (posi>=LV) posi=posi-LV;
                      if (posi<0) posi=LV+posi;
                      mean+=REAL(Vect)[posi];
                }
                /* Save min element */
                REAL(VectResult)[h]=mean/LW;
```

```
            }
      UNPROTECT(1);
      return(VectResult);
}


/* Makes a mean filter in a Time Series */
SEXP removefiltermin(SEXP Vect, SEXP NumT)
{
      int h,LV;
      double yval,mint;
      LV = length(Vect);
      SEXP VectResult;
      mint=(int)REAL(NumT)[0];
      PROTECT(VectResult=allocVector(REALSXP, LV));
      yval=REAL(Vect)[0];
      if (yval<mint) yval=mint;
      REAL(VectResult)[0]=yval;

      for (h=1;h<LV;h++)
            {
                  if (REAL(Vect)[h]<mint)
                        REAL(VectResult)[h]=yval;
                  else
                        {
                        REAL(VectResult)[h]=REAL(Vect)[h];
                        yval=REAL(Vect)[h];
                        }
            }
      UNPROTECT(1);
      return(VectResult);
}

/* Makes a mean filter in a Time Series */
SEXP removefiltermax(SEXP Vect, SEXP NumT)
{
      int h,LV;
      double yval,maxt;
      LV = length(Vect);
      SEXP VectResult;
      maxt=(int)REAL(NumT)[0];
      PROTECT(VectResult=allocVector(REALSXP, LV));
      yval=REAL(Vect)[0];
      if (yval>maxt) yval=maxt;
      REAL(VectResult)[0]=yval;

      for (h=1;h<LV;h++)
            {
                  if (REAL(Vect)[h]>maxt)
                        REAL(VectResult)[h]=yval;
                  else
                        {
                        REAL(VectResult)[h]=REAL(Vect)[h];
                        yval=REAL(Vect)[h];
                        }
            }
      UNPROTECT(1);
      return(VectResult);
}

/* Makes a mean filter in a Time Series */
SEXP removefilterrange(SEXP Vect, SEXP NumT)
{
      int h,LV;
```

```
      double yval,mint,maxt;
      LV = length(Vect);
      SEXP VectResult;
      mint=(int)REAL(NumT)[0];
      maxt=(int)REAL(NumT)[1];
      PROTECT(VectResult=allocVector(REALSXP, LV));
      yval=REAL(Vect)[0];
      if (yval<mint) yval=mint;
      if (yval>maxt) yval=maxt;

      REAL(VectResult)[0]=yval;

      for (h=1;h<LV;h++)
            {
                  if (REAL(Vect)[h]<mint || REAL(Vect)[h]>maxt)
REAL(VectResult)[h]=yval;
                  else
                        {
                        REAL(VectResult)[h]=REAL(Vect)[h];
                        yval=REAL(Vect)[h];
                        }
            }
      UNPROTECT(1);
      return(VectResult);
}

/* Makes a mean filter in a Time Series */
SEXP removefilterrangeinv(SEXP Vect, SEXP NumT)
{
      int h,LV;
      double yval,mint,maxt;
      LV = length(Vect);
      SEXP VectResult;
      mint=(int)REAL(NumT)[0];
      maxt=(int)REAL(NumT)[1];
      PROTECT(VectResult=allocVector(REALSXP, LV));
      yval=REAL(Vect)[0];
      if (yval>mint && yval<maxt) yval=mint;

      REAL(VectResult)[0]=yval;

      for (h=1;h<LV;h++)
            {
                  if (REAL(Vect)[h]>mint && REAL(Vect)[h]<maxt)
REAL(VectResult)[h]=yval;
                  else
                        {
                        REAL(VectResult)[h]=REAL(Vect)[h];
                        yval=REAL(Vect)[h];
                        }
            }
      UNPROTECT(1);
      return(VectResult);
}



/*****************************************************************
*******************************/
/*****************************************************************
*******************************/
/*****************************************************************
*******************************/
```

```
/*C functions for the kdfilterImportantPoints R function*/
/*Find the first important point */
double findfirst(SEXP vect, float R)
      {
      int imin=1;
      int imax=1;
      int LV;
      LV = length(vect);
      while((q<LV) & (((REAL(vect)[q]/REAL(vect)[imin])<R) ||
((REAL(vect)[imax]/REAL(vect)[q])<R)))
      {
            if(REAL(vect)[q]<REAL(vect)[imin])
            imin=q;

            if(REAL(vect)[q]>REAL(vect)[imax])
            imax=q;
      q++;
      }

      if(imin<imax)
            return(REAL(vect)[imax]);

      else
            return(REAL(vect)[imin]);


}

/*-----------------------------------------------------------------
*/

/*Find the first important minimum after the qth point */
double findminimum(SEXP vect, float R)
      {
      int imin=q;
      int LV;
      LV = length(vect);

      while((q<LV)&((REAL(vect)[q]/REAL(vect)[imin])<R))
      {
            if(REAL(vect)[q]<REAL(vect)[imin])
            imin=q;
            q++;
      }
      if((q<LV)&(REAL(vect)[imin]<REAL(vect)[q]))
            {cont=imin;return(REAL(vect)[imin]);}
      return(REAL(vect)[imin]);
}

/*-----------------------------------------------------------------
*/

/*Find the first important maximum after the qth point */
double findmaximum(SEXP vect, float R)
{
      int imax=q;
      int LV;
      LV = length(vect);

      while((q<LV)&((REAL(vect)[imax]/REAL(vect)[q])<R))
      {
            if(REAL(vect)[q]>REAL(vect)[imax])
```

```
             imax=q;
            q++;
      }
      if((q<LV)&(REAL(vect)[imax]>REAL(vect)[q]))
            {cont=imax;return(REAL(vect)[imax]);}
      return(REAL(vect)[imax]);

}
/*-------------------------------------------------------------------
*/

/*protect memory*/
double memo(SEXP vect)
{
int LV;
LV=length(vect);
PROTECT(positions=allocVector(INTSXP,LV));
return(LV);
}
/*-------------------------------------------------------------------
*/
/*Top-level function for finding important points.*/
SEXP importantpoints(SEXP vect,SEXP vectorR)
      {
      float R;
      int i,LV;
      int rep=0;
      q=0;
      i=0;
      R=REAL(vectorR)[0];
      SEXP VectResult,auxvect1;
      LV = length(vect);
      PROTECT(VectResult=allocVector(REALSXP, LV));

      REAL(VectResult)[i]=REAL(vect)[q];
      INTEGER(positions)[i]=(q+1);
      /*Index of VectResult*/
      i++;
      q++;
      /*Looking for the index of first important point*/
      REAL(VectResult)[i]=findfirst(vect,R);
      INTEGER(positions)[i]=q;
      cont=q;
      i++;
      if((q<LV) & (REAL(vect)[q]>REAL(vect)[0]))
      {REAL(VectResult)[i]=findmaximum(vect,R);INTEGER(positions)[i]=c
ont+1;}
      i++;
      /*Looking for the alternative minimum and a maximun*/
      while ((i<LV)&(rep==0))
      {
            if (rep==0)
            REAL(VectResult)[i]=findminimum(vect,R);
            if(rep==0)
            INTEGER(positions)[i]=cont+1;
            if(INTEGER(positions)[i]==INTEGER(positions)[i-1])
            {rep=1;last=i;}
            i++;
            if (rep==0)
            REAL(VectResult)[i]=findmaximum(vect,R);
            if(rep==0)
            INTEGER(positions)[i]=cont+1;
            if(INTEGER(positions)[i]==INTEGER(positions)[i-1])
            {rep=1;last=i;}
```

```
            i++;
      }
      PROTECT(auxvect1=allocVector(REALSXP, last));
      PROTECT(auxpos=allocVector(INTSXP, last));

      for (i=0;i<last;i++)
      {
      REAL(auxvect1)[i]=REAL(VectResult)[i];
      }
      for (i=0;i<last;i++)
      {
      INTEGER(auxpos)[i]=INTEGER(positions)[i];
      }
      UNPROTECT(1);
      return(auxvect1);
}

/*-------------------------------------------------------------------
*/

/*Returns the positions of the important points*/
SEXP posfunction(SEXP vect)
      {
      return(auxpos);
}
/*-------------------------------------------------------------------
*/
```

I.22. Programa “kdseries.c” para implementación de algoritmos RAST (En desarrollo).

```
Programa: kdseries.c

#include <string.h>
#include <R.h>
#include <S.h>
#include <Rinternals.h>
#include <Rdefines.h>
#include <math.h>

#define MAXLEVEL 7      /* MAX PATTERN LEVEL */
#define MAXLONG 22 /* MAX PATTERN LONG */
#define MAXCOMBISIZE 20000 /* MAX COMBINATORY ELEMENTS SIZE */

/* Find similar patterns in a Time Series level1*/
/* WinS={Support, printme}*/
SEXP findsimilarpat_level1(SEXP VectPattern, SEXP WinS)
{
      long int h, i, j, k;
      long int limitpatt, blockpatt, countpatt, Nump;
      long int countpattfound, Numpfound;

      int printme, incpatt, Support;

      int *pattern;

      SEXP VectResult;

      Support=(int)INTEGER(WinS)[0];
      printme=(int)INTEGER(WinS)[1];

      Nump=length(VectPattern);
      if (Nump==0)
            {
            return (VectResult);
            }

      /* Alloc buffer memory */
      blockpatt=1000*2;
      limitpatt=blockpatt;
      pattern=(int *)S_alloc(limitpatt, sizeof(int));

      /*Initializate pointer*/
      countpatt=0;

      for (h=0; h<Nump; h++)
            {
            incpatt=0;
            for (j=0;j<countpatt;j++)
                  {
                  if (*(pattern+(j*2))==INTEGER(VectPattern)[h])
                        {
                              *(pattern+(j*2)+1)+=1;
                              incpatt=1;
                              break;
                        }
                  }
            if (incpatt==0)
```

```
                {

      *(pattern+(countpatt*2))=INTEGER(VectPattern)[h];
                        *(pattern+1+(countpatt*2))=1;
                        countpatt++;
                        if (countpatt*2>=limitpatt)
                              {
                              limitpatt+=blockpatt;
                              pattern=(int *)S_realloc((char *)pattern,
limitpatt, limitpatt-blockpatt, sizeof(int));
                              }
                }
          }



      /* Print patterns level one*/
      if (printme==1)
            {
            Rprintf("######################\n");
            Rprintf("NumPat=%d,Support=%d\n",countpatt,Support);
            Rprintf("######################\n");
            for (j=0; j<countpatt; j++)
                  {
                  Rprintf("Num.=%d\tCode=%d,Len=%d\n", j,
*(pattern+(j*2)), *(pattern+1+(j*2)));
                  }
            }


      /*Initializates pointer*/
      Numpfound=0;

      /*Count Patterns with Num>=Support*/
      for (j=0; j<countpatt; j++)
            {
            if (*(pattern+1+(j*2))>=Support) Numpfound++;
            }

      /* Save patterns with Num>=Support */
      PROTECT(VectResult=allocMatrix(INTSXP, Numpfound+1, 2));

      INTEGER(VectResult)[0]=Numpfound;
      INTEGER(VectResult)[Numpfound+1]=countpatt;
      countpattfound=1;
      for (j=0; j<countpatt; j++)
            {
            if (*(pattern+1+(j*2))>=Support)
                  {
                  /* Save patterns with number>=Support */
                  INTEGER(VectResult)[countpattfound]=*(pattern+(j*2));

      INTEGER(VectResult)[countpattfound+(Numpfound+1)]=*(pattern+1+(j
*2));
                  countpattfound++;
                  }
            }

      UNPROTECT(1);
      return(VectResult);
}

/* Find similar consecutive patterns in a Time Series levelN*/
/* WinS={Support, printme, level}*/
```

```
SEXP findsimilarpat_levelN(SEXP VectPattern, SEXP WinS, SEXP VectPat1,
SEXP VectPatN)
{
      long int h, i, j, k, l, m;
      long int limitpatt, blockpatt, countpatt, Nump;
      long int countpattfound, Numpfound;

      int printme, incpatt, Support, level;
      int searchp, searchp2, NumpN, Nump1;
      int *pattern;
      int patternbuff[50];

      SEXP VectResult;

      Support=(int)INTEGER(WinS)[0];
      printme=(int)INTEGER(WinS)[1];
      level=(int)INTEGER(WinS)[2];

      Nump=length(VectPattern);
      if (Nump==0)
            {
            return (VectResult);
            }
      /* Alloc buffer memory */
      blockpatt=1000*(level+1);
      limitpatt=blockpatt;
      pattern=(int *)S_alloc(limitpatt, sizeof(int));

      /*Initializates pointer*/
      countpatt=0;
      /* h=each pattern */
      Nump1=INTEGER(VectPat1)[0];
      NumpN=INTEGER(VectPatN)[0];
      for (h=0; h<(Nump-level+1); h++)
                  {
                  /* Searches if pattern VectPatN[n] is in "h" position
*/
                  for (i=1;i<=NumpN;i++)
                        {
                        searchp=1;
                        for (j=0;j<(level-1);j++)
                              {
                              patternbuff[j]=INTEGER(VectPattern)[h+j];
                              if
(INTEGER(VectPatN)[i+(j*(NumpN+1))]!=patternbuff[j])
                                    {
                                          searchp=0;
                                          break;
                                    }
                              }
                        /* if this pattern is in this position then*/
                        if (searchp==1)
                              {
                              /* Is next_position in VectPat1? */
                              for (j=1;j<=Nump1;j++)
                                    {
                                    /* If next_position is in VectPat1
save pattern o increment if it exists*/
                                    patternbuff[level-
1]=INTEGER(VectPattern)[h+level-1];
                                    if
(INTEGER(VectPat1)[j]==patternbuff[level-1])
                                          {
                                          /*
```

```
                                        Rprintf("\nh=%d:",h);
                                        for (l=0;l<level;l++)
                                              {

      Rprintf("%d,",patternbuff[l]);

                                              }
                                        */

                                        searchp2=0;
                                        for (k=0;k<countpatt;k++)
                                              {
                                              searchp2=1;
                                              for (l=0;l<level;l++)
                                                    {
                                                    if
(*(pattern+l+(k*(level+1)))!=patternbuff[l])
                                                          {

      searchp2=0;

      break;
                                                          }
                                                    }
                                              if (searchp2==1) break;
                                              }
                                        if (searchp2==1)
                                              {

      *(pattern+level+(k*(level+1)))+=1;
                                              }
                                        else
                                              {
                                              for (l=0;l<level;l++)
                                                    {

      *(pattern+l+(countpatt*(level+1)))=patternbuff[l];
                                                    }

      *(pattern+level+(countpatt*(level+1)))=1;
                                              countpatt++;
                                              if
(countpatt*(level+1)>=limitpatt)
                                                    {

      limitpatt+=blockpatt;
                                                    pattern=(int
*)S_realloc((char *)pattern, limitpatt, limitpatt-blockpatt,
sizeof(int));
                                                    }
                                              }
                                        break;
                                        }
                                  }
                            break;
                            }
                      }
            }


      /* Print patterns level N*/
      if (printme==1)
            {
            Rprintf("##########################################\n");
```

```
      Rprintf("NumPat=%d,Support=%d,Level=%d\n",countpatt,Support,leve
l);
            Rprintf("##########################################\n");
            for (j=0; j<countpatt; j++)
                  {
                  Rprintf("Num.=%d\tLen=%d\t={", j,
*(pattern+level+(j*(level+1))));
                  for (k=0;k<=level;k++)
                        {
                        Rprintf("Code%d=%d,", k,
*(pattern+k+(j*(level+1))));
                        }
                  Rprintf("}\n");
                  }
            }

      /*Initializates pointer*/
      Numpfound=0;

      /*Counts Patterns with Num>=Support*/
      for (j=0; j<countpatt; j++)
            {
            if (*(pattern+level+(j*(level+1)))>=Support) Numpfound++;
            }

      /* Save patterns with Num>=Support */
      PROTECT(VectResult=allocMatrix(INTSXP, Numpfound+1, level+1));

      for (k=0;k<=level;k++)
            {
            INTEGER(VectResult)[k*(Numpfound+1)]=0;
            }
      INTEGER(VectResult)[0]=Numpfound;
      INTEGER(VectResult)[Numpfound+1]=countpatt;
      countpattfound=1;

      for (j=0; j<countpatt; j++)
            {
            if (*(pattern+level+(j*(level+1)))>=Support)
                  {
                  /* Saves patterns with number>=Support */
                  for (k=0;k<=level;k++)
                        {

      INTEGER(VectResult)[countpattfound+(k*(Numpfound+1))]=*(pattern+
k+(j*(level+1)));
                        }
                  countpattfound++;
                  }
            }

      UNPROTECT(1);
      return(VectResult);
}

/* Find position of patterns*/
/* WinS={printme, level}*/
SEXP findpositionpat_levelN(SEXP VectPattern, SEXP WinS, SEXP
VectPatN, SEXP Position)
{
      long int h, i, j, k, l, m;
      long int limitpatt, blockpatt, countpatt, Nump;
```

```
      int printme, incpatt, Support, level;
      int searchp, searchp2, NumpN, Nump1;
      int *pattern;
      int patternbuff[50];

      SEXP VectResult;

      printme=(int)INTEGER(WinS)[0];
      level=(int)INTEGER(WinS)[1];

      Nump=length(VectPattern);
      if (Nump==0)
            {
            return (VectResult);
            }

      /* Alloc buffer memory */
      blockpatt=1000*2;
      limitpatt=blockpatt;
      pattern=(int *)S_alloc(limitpatt, sizeof(int));

      /*Initializate pointer*/
      countpatt=0;
      /* h=each pattern size=1 */
      NumpN=INTEGER(VectPatN)[0];
      /* Rprintf("Level=%d,Nump=%d,NumpN=%d\n",level,Nump,NumpN);*/
      for (h=0; h<(Nump-level+1); h++)
            {
            /* Search if pattern VectPatN[n] is in "h" position */
            for (i=1;i<=NumpN;i++)
                  {
                  searchp=1;
                  for (j=0;j<level;j++)
                        {
                        /*if (h==1)
Rprintf("h=%d,i=%d,j=%d\n,Num1=%d,Num2=%d\n",h,i,j,INTEGER(VectPatN)[i
+(j*(NumpN+1))],INTEGER(VectPattern)[h+j]);*/
                        if
(INTEGER(VectPatN)[i+(j*(NumpN+1))]!=INTEGER(VectPattern)[h+j])
                              {
                              searchp=0;
                              break;
                              }
                        }
                        /* if this pattern is in this position then
save it and position*/
                  if (searchp==1)
                        {
                        *(pattern+(countpatt*2))=i;

      *(pattern+1+(countpatt*2))=INTEGER(Position)[h];
                        countpatt++;
                        if ((countpatt*2)>=limitpatt)
                              {
                              limitpatt+=blockpatt;
                              pattern=(int *)S_realloc((char *)pattern,
limitpatt, limitpatt-blockpatt, sizeof(int));
                              }
                        break;
                        }
                  }
            }

      /* Save patterns with Num>=Support */
```

```
      PROTECT(VectResult=allocMatrix(INTSXP, countpatt, 2));

      for (h=0;h<countpatt;h++)
            {
            INTEGER(VectResult)[h]=*(pattern+1+(h*2));
            INTEGER(VectResult)[h+countpatt]=*(pattern+(h*2));
            }

      UNPROTECT(1);
      return(VectResult);
}


/* Obtain combinations without repetition
 N=Number of elements
 Level=Num of elements of each combination
 */
SEXP combinations(SEXP N, SEXP Levels)
{
      long int h, i, j, k, m;
      long int NumCombi, Cont;
      int *combi;
      int Num, Lev, L;

      Num=(int)INTEGER(N)[0];
      Lev=(int)INTEGER(Levels)[0];


/* NumCombi=N!/((N-Levels)!*Level!) */
      NumCombi=1;
      for (h=(Num-Lev+1);h<=Num;h++)
            {
            NumCombi*=h;
            }
      for (h=2;h<=Lev;h++)
            {
                  NumCombi/=h;
            }

      /*Rprintf("N=%d\tLevels=%d\tNumCombi=%d\n",Num,Lev,NumCombi);*/
      SEXP MatResult;

      /* Save patterns with Num>=Support */
      PROTECT(MatResult=allocMatrix(INTSXP, NumCombi, Lev));

      /* Fills first combination */
      combi=(int *)S_alloc(Lev, sizeof(int));
      h=1;
      while (h<=Lev)
            {
            *(combi+h-1)=h++;
            }

      Cont=0;
      L=Lev;
      /* Lev=MAXLEVEL, L=Current Level*/
      while (L>0)
            {
            if (L==Lev)
                  {
                  /*Rprintf("\n");*/
                  for (h=0;h<Lev;h++)
                        {
```

```
      INTEGER(MatResult)[Cont+(h*NumCombi)]=*(combi+h);
                  /*Rprintf("%d:",*(combi+h));*/
                  }
            Cont++;
            }
      *(combi+L-1)+=1;
      if (L!=Lev)
            {
            for (h=(L+1);h<=Lev;h++)
                  {
                  *(combi+h-1)=*(combi+h-2)+1;
                  }
            }
      if (*(combi+L-1)>(Num-Lev+L))
            {
                  L--;
            }
      else
            {
            L=Lev;
            }
      }

      UNPROTECT(1);
      return(MatResult);
}



/* Obtaining combinations
 Num=Number of elements
 Lev=Num of elements of each combination
 NumCombi=Num Combinations
 */

void c_combinations(int *Buffer, int Num, int Lev, int NumCombi)
{
      long int h, i, j, k, m;
      long Cont;
      int combi[100];
      int L;

      h=1;
      while (h<=Lev)
            {
            *(combi+h-1)=h++;
            }

      Cont=0;
      L=Lev;
      /* Lev=MAXLEVEL, L=Current Level*/
      while (L>0)
            {
            if (L==Lev)
                  {
                  /*Rprintf("\n");*/
                  for (h=0;h<Lev;h++)
                        {
                        *(Buffer+Cont+(h*NumCombi))=*(combi+h);
                        /*Rprintf("Cont:%d,%d:",Cont,*(combi+h));*/
                        }
                  Cont++;
                  }
```

```
            *(combi+L-1)+=1;
            if (L!=Lev)
                  {
                  for (h=(L+1);h<=Lev;h++)
                        {
                        *(combi+h-1)=*(combi+h-2)+1;
                        }
                  }
            if (*(combi+L-1)>(Num-Lev+L))
                  {
                        L--;
                  }
            else
                  {
                  L=Lev;
                  }
            }
      return;
}

/* Search pattern in a Buffer with elements by columns*/
/* Buffer=Patterns+NumPat [1:(Level+1)] */
/* totalpatt=Num of total patterns (row) */
/* Pattern=Pattern to Search */
/* Level=Num Level Pattern */
long int col_findpat(int *Buffer, long int totalpatt, int *Pattern,
int Level)
{
      long int h, i;
      int searchp;
      for (h=0; h<totalpatt; h++)
            {
            /* Searches if pattern VectPatN[n] is in "h" position */
            searchp=1;
            for (i=0;i<Level;i++)
                  {
                  if (*(Pattern+i)!=*(Buffer+h+(i*totalpatt)))
                        {
                        searchp=0;
                        break;
                        }
                  }
            /* if this pattern is in this position then return
position*/
            if (searchp==1)
                  {
                  return(h);
                  }
            }

      /* NOT FOUND */
      return(-1);
}


/* Search pattern in a Buffer with elements by rows*/
/* Buffer=Patterns+NumPat [1:(Level+1)] */
/* totalpatt=Num of total patterns (row) */
/* Pattern=Pattern to Search */
/* Level=Num Level Pattern */
long int row_findpat(int *Buffer, long int totalpatt, int *Pattern,
int Level, int Width)
{
      long int h, i;
```

```
      int searchp;
      for (h=0; h<totalpatt; h++)
            {
            /* Searches if pattern VectPatN[n] is in "h" position */
            searchp=1;
            for (i=0;i<Level;i++)
                  {
                  if (*(Pattern+i)!=*(Buffer+i+(h*Width)))
                        {
                        searchp=0;
                        break;
                        }
                  }
            /* if this pattern is in this position then return
position*/
            if (searchp==1)
                  {
                  return(h);
                  }
            }

      /* NOT FOUND */
      return(-1);
}



/* Find similar combinatory patterns in a Time Series levelN using one
slide window*/
/* WinS={Support, printme, level}*/
SEXP findcombinatorypat_levelN(SEXP VectPattern, SEXP WinS, SEXP
VectPatN)
{
      long int h, i;
      int j, k, l, m;
      long int limitpatt, blockpatt, countpatt, Nump, NumpN,
sizecombi;
      long int countpattfound, Numpfound;

      int printme, incpatt, Support, level;
      int searchp, searchp2, ActualWidWin, maxwinwidth;
      int *pattern;
      int *pattVectPatN;
      int patternbuff[1000];
      int combinat[MAXCOMBISIZE*MAXLEVEL];
      int localpatt[MAXLEVEL];
      int NumCombi, NumCmax;
      int norep;

      SEXP VectResult;

      Support=(int)INTEGER(WinS)[0];
      printme=(int)INTEGER(WinS)[1];
      level=(int)INTEGER(WinS)[2];
      maxwinwidth=(int)INTEGER(WinS)[3];
      norep=(int)INTEGER(WinS)[4];

      Nump=length(VectPattern)/2;
       /* Rprintf("Nump=%d",Nump); */

      if (Nump==0)
            {
            return (VectResult);
            }
```

```
      if (level<2 || maxwinwidth<1 || level>MAXLEVEL)
            {
            return (VectResult);
            }


      /* Alloc buffer memory */
      blockpatt=1000*(level+1+norep);
      limitpatt=blockpatt;
      pattern=(int *)S_alloc(limitpatt, sizeof(int));

      /* Alloc buffer previous pattern memory */
      NumpN=length(VectPatN)/level;
      pattVectPatN=(int *)S_alloc(NumpN*level, sizeof(int));

      /*Rprintf("NumpN=%d, Level=%d\n",NumpN,level);*/
      /* Filling Previous pattern buffer */
      for (k=1;k<NumpN;k++)
            {
            for (j=0;j<(level-1);j++)
                  {
                  *(pattVectPatN+k-
1+(j*NumpN))=INTEGER(VectPatN)[k+(NumpN*j)];
                  /*Rprintf("Pos=%d,k=%d, l=%d, Val=%d\n",k-
1+(j*NumpN),k,j,INTEGER(VectPatN)[k+(NumpN*j)]);*/
                  }
            }

      /* FINDS PATTERNS LEVEL "level" */
      /*Initializates pointer*/
      countpatt=0;
      /* h=each position */

      for (h=0; h<(Nump-level+1); h++)
            {
            /* Obtain elements which are into slide windows */
            /* ======================================== */
            i=1;
            /* Get distance with second element*/
            *(patternbuff)=INTEGER(VectPattern)[h];
            ActualWidWin=INTEGER(VectPattern)[(h+1)+Nump];
            if (printme==5) Rprintf("\nPos
%d=%d(%d):",h+1,*(patternbuff+i-1),ActualWidWin);
            while (ActualWidWin<=maxwinwidth && (h+1+i)<=Nump &&
i<1000)
                  {
                  *(patternbuff+i)=INTEGER(VectPattern)[h+i];
                  if ((h+1+i)<Nump)
ActualWidWin+=INTEGER(VectPattern)[(h+1+i)+Nump];
                  if (printme==5)
Rprintf("%d(%d):",*(patternbuff+i),ActualWidWin);
                  i++;
                  }
            /*if ((h+i)!=Nump) i--;*/
            if (printme==5) Rprintf("\t(N=%d,MaxWidth=%d,
ActualWidth=%d)",i,maxwinwidth,ActualWidWin-
INTEGER(VectPattern)[(h+i)+Nump]);

            /* Generating serial combinations into window's width i-1
selecting level-1 elements*/
            /* Obtaining combinatory positions */
            if (i>=level)
                  {
                  /* Computing num combinatory elements */
```

```
                  /* NumCombi=N!/((N-Levels)!*Level!) */
                  if ((level-1)==1)
                        {
                              NumCombi=i-1;
                              for (j=1;j<=NumCombi;j++)
                                    {
                                    *(combinat+j-1)=j;
                                    }
                        }
                  else
                        {
                        NumCombi=1;
                        for (j=((i-1)-(level-1)+1);j<=(i-1);j++)
                              {
                              NumCombi*=j;
                              }
                        for (j=2;j<=(level-1);j++)
                              {
                              NumCombi/=j;
                              }
                        /* Obtaining combinatory positions */
                        if (NumCombi>=MAXCOMBISIZE)
                              {
                              Rprintf("ERROR. Max number of
combinations excedeed. Change: MAXCOMBISIZE!!!");
                              return(VectResult);
                              }
                        c_combinations(combinat,i-1,level-1,NumCombi);
                        }
                  if (printme==2) Rprintf("\n(N-1)=%d,(level-
1)=%d,NumCombi=%d",i-1,level-1,NumCombi);

                  /* fill with 0's patterns norep buffer */
                  if (norep)
                        {
                        for (j=0;j<countpatt;j++)
                              {
                              *(pattern+level+1+((level+2)*j))=0;
                              }
                        }


                  /* Getting local patterns */
                  localpatt[0]=INTEGER(VectPattern)[h];
                  for (k=0;k<NumCombi;k++)
                        {
                              if (printme==7) Rprintf("\n
h=%d,N=%d,P=%d",h+1,NumCombi,localpatt[0]);
                              for (l=0;l<(level-1);l++)
                                    {

      localpatt[l+1]=INTEGER(VectPattern)[h+*(combinat+k+(l*NumCombi))
];
                                    if (printme==7)
Rprintf(":%d",localpatt[l+1]);
                                    }
                              /* (level-1) elements from pattern into
VectPatN buffer? */
                              if (col_findpat(pattVectPatN, NumpN,
localpatt, level-1)!=-1)
                                    {
                                    /* There are (level) elements into
pattern buffer? */
```

```
                                    countpattfound=row_findpat(pattern,
countpatt, localpatt, level, level+1+norep);
                                    if (countpattfound==-1)
                                          {
                                          countpatt++;
                                          if (printme==3)
Rprintf("\nCounpatt=%d",countpatt);
                                          if
((countpatt*(level+1+norep))>=limitpatt)
                                                {
                                                limitpatt+=blockpatt;
                                                pattern=(int
*)S_realloc((char *)pattern, limitpatt, limitpatt-blockpatt,
sizeof(int));
                                                      if
(pattern==NULL)
                                                      {

      Rprintf("\n##############\nREALLOCATING PATTERN MEMORY ERROR
(Size=%ld Bytes)!!!\n##############\n",limitpatt*sizeof(int));

      return(VectResult);
                                                      }
                                                }
                                          /* Saves new pattern */
                                          if (norep)
                                                {
                                                for (l=0;l<level;l++)
                                                      {

      *(pattern+l+((level+2)*(countpatt-1)))=*(localpatt+l);
                                                      }

      *(pattern+level+((level+2)*(countpatt-1)))=1;


      *(pattern+level+1+((level+2)*(countpatt-1)))=1;
                                                }
                                          else
                                                {
                                                for (l=0;l<level;l++)
                                                      {

      *(pattern+l+((level+1)*(countpatt-1)))=*(localpatt+l);
                                                      }

      *(pattern+level+((level+1)*(countpatt-1)))=1;

                                                }
                                          }
                                    else
                                          {
                                          /* if pattern exits and there
is not a similar pattern into the window increments number patt*/
                                          if (norep)
                                                {
                                                if
(*(pattern+level+1+((level+2)*countpattfound))==0)
                                                      {

      *(pattern+level+((level+2)*countpattfound))+=1;

      *(pattern+level+1+((level+2)*countpattfound))=1;
                                                      }
```

```
                                          }
                                    else
                                          {

      *(pattern+level+((level+1)*countpattfound))+=1;
                                          }
                                    }
                               }
                         }
                   }
             }


      if (printme==3)
             {
             for (h=0;h<countpatt;h++)
                   {
                   Rprintf("\nP%d=",h);
                   for (l=0;l<=level;l++)
                         {

      Rprintf("::%d",*(pattern+l+((level+1+norep)*h)));
                         }
                   }
             }

      /*Counts Patterns with Num>=Support*/
      Numpfound=0;
      NumCmax=0;
      for (j=0; j<countpatt; j++)
             {
             if (*(pattern+level+((level+1+norep)*j))>=Support)
Numpfound++;
             if (*(pattern+level+((level+1+norep)*j))>NumCmax)
NumCmax=*(pattern+level+((level+1+norep)*j));
             }

      /* ALLOC MATRIX */
      PROTECT(VectResult=allocMatrix(INTSXP, Numpfound+1, level+1));

      /* First row with number of patterns wthat meet supports and
total patterns*/
      for (k=0;k<=level;k++)
             {
             INTEGER(VectResult)[k*(Numpfound+1)]=0;
             }

      INTEGER(VectResult)[0]=Numpfound; /* Num patterns that meet
supports in this level */
      INTEGER(VectResult)[Numpfound+1]=countpatt; /* Num total
patterns in this level */
      INTEGER(VectResult)[2*(Numpfound+1)]=NumCmax; /* Num max similar
elements found */

      /* Save patterns with Num>=Support */
      countpattfound=1;
      for (j=0; j<countpatt; j++)
             {
             if (*(pattern+level+((level+1+norep)*j))>=Support)
                   {
                   /* Saves patterns with number>=Support */
                   for (k=0;k<=level;k++)
                         {
```

```
      INTEGER(VectResult)[countpattfound+(k*(Numpfound+1))]=*(pattern+
k+((level+1+norep)*j));
                        }
                  countpattfound++;
                  }
            }


      UNPROTECT(1);
      return(VectResult);
}



/* Find rules from pat1 to pat 2 that meet confidence*/
/* AntePat=antecedent patterns, TotalPat=(antecedent+consecuent),
patterns */
/* Param={Confidence, LevelAntepatt, LevelTotalpatt, printme}*/
SEXP findrules(SEXP AntePat, SEXP TotalPat, SEXP Param)
{
      long int h, i;
      int j, k, l, m;

      int printme;
      float SuppAnte, SuppCons;
      float Confidence, Found;
      int LevelAntepatt, LevelTotalpatt;

      int *pattern;
      int pattbuffTotal[1000];
      int NumAnte, NumTotal;
      int path, patj;
      int countrules;
      long int limitpatt, blockpatt, countpatt;

      SEXP VectResult;

      Confidence=((float)INTEGER(Param)[0])/1000.0;
      LevelAntepatt=(int)INTEGER(Param)[1];
      LevelTotalpatt=(int)INTEGER(Param)[2];
      printme=(int)INTEGER(Param)[3];

      NumAnte=length(AntePat)/(LevelAntepatt+1);
      NumTotal=length(TotalPat)/(LevelTotalpatt+1);



      if (NumAnte<1 || NumTotal<1 || LevelAntepatt>=LevelTotalpatt ||
LevelAntepatt<1)
            {
            return (VectResult);
            }


      if (printme==1) Rprintf("\nNumAnte=%d,
NumTotal=%d",NumAnte,NumTotal);
      /* Alloc buffer memory */
      blockpatt=1000*4;
      limitpatt=blockpatt;
      pattern=(int *)S_alloc(limitpatt, sizeof(int));

      /*Initializates pointer*/
      countrules=0;
```

```
      /* Searching rules */
      for (h=0;h<NumTotal;h++)
            {
            /* Filling TotalPatt Buffer */
            for (j=0;j<LevelAntepatt;j++)
                  {
                  *(pattbuffTotal+j)=INTEGER(TotalPat)[h+(NumTotal*j)];
                  }

            /* Searching where is antecent */
            for (k=0;k<NumAnte;k++)
                  {
                  /* Filling Total Buffer */
                  Found=1;
                  for (j=0;j<LevelAntepatt;j++)
                        {
                        if
(*(pattbuffTotal+j)!=INTEGER(AntePat)[k+(NumAnte*j)])
                              {
                              Found=0;
                              break;
                              }
                        }
                  if (Found==1)
                        {

      SuppAnte=(float)INTEGER(AntePat)[k+(NumAnte*LevelAntepatt)];

      SuppCons=(float)INTEGER(TotalPat)[h+(NumTotal*LevelTotalpatt)];
                        if (Confidence<=(SuppCons/SuppAnte))
                              {
                              /* Save position Ante, position
Consecuent, Support, Confidence */
                              *(pattern+(countrules*4))=k+2;
                              *(pattern+1+(countrules*4))=h+2;

      *(pattern+2+(countrules*4))=(int)SuppAnte;

      *(pattern+3+(countrules*4))=(int)1000.0*(SuppCons/SuppAnte);

                              countrules++;
                              if (printme==3)
Rprintf("\nCounrules=%d",countrules);
                              if ((countrules*4)>=limitpatt)
                                    {
                                    limitpatt+=blockpatt;
                                    pattern=(int *)S_realloc((char
*)pattern, limitpatt, limitpatt-blockpatt, sizeof(int));
                                    if (pattern==NULL)
                                          {

      Rprintf("\n##############\nREALLOCATING PATTERN MEMORY ERROR
(Size=%ld Bytes)!!!\n##############\n",limitpatt*sizeof(int));
                                          return(VectResult);
                                          }
                                    }
                              }
                        break;
                        }
                  }
            }

      /* ALLOC MATRIX */
      PROTECT(VectResult=allocMatrix(INTSXP, countrules, 4));
```

```
      /* Save rules with meet confidence */

      for (j=0; j<countrules; j++)
            {
            INTEGER(VectResult)[j]=*(pattern+(j*4));
            INTEGER(VectResult)[j+countrules]=*(pattern+1+(j*4));
            INTEGER(VectResult)[j+(2*countrules)]=*(pattern+2+(j*4));
            INTEGER(VectResult)[j+(3*countrules)]=*(pattern+3+(j*4));
            }

      UNPROTECT(1);
      return(VectResult);
}
```