Online Resource 04 - Functions in tempFGN R package

How does the temperature vary over time? Evidence on the Stationary and Fractal nature of Temperature Fluctuations

John Dagsvik, Mariachiara Fortuna, Sigmund H. Moen

Affiliations:

John K. Dagsvik, Statistics Norway, Research Department;

Mariachiara Fortuna, freelance statistician, Turin;

Sigmund Hov Moen, Westerdals Oslo School of Arts, Communication and Technology.

Corresponding author:

John K. Dagsvik, E-mail: john.dagsvik@ssb.no

Mariachiara Fortuna, E-mail: mariachiara.fortuna1@gmail.com (reference for code and analysis)

tempFGN R package

All the functions listed below are part of the R package tempFGN, developed as support for the paper *"How does the temperature vary over time? Evidence on the Stationary and Fractal nature of Temperature Fluctuations"* by John Dagsvik, Mariachiara Fortuna, Sigmund H. Moen.

Functions are reported with a basic roxygen2 documentation.

Full package available at mariachiara.fortuna1@gmail.com. Data and Rmarkdown used to produce results and appendix also available.

Data handling functions

```
#' Reshape data into monthly, long, scaled version
#'
#' @param data the raw data temperature data.frame
#' @param scale logical, default is false. If the data should be scaled
#' @return data.frame
#' @examples
#' monthlyAdj(data)
#' @importFrom dplyr select mutate arrange
#' @importFrom tidyr gather
#' @importFrom lubridate ymd
#' @export
monthlyAdj <- function(data, scale = F) {
  # Scale data if required (produces normalized data)
  if(scale == T) {
    scaledData <- scale(data[,2:13])
    data[,2:13] <- scaledData
  }
  # Change colnames to month numbers
  colnames(data)[2:13] <- 1:12
  # Data Manipulation: Gather data and create a Time variable
  mdata <- data %>%
    select(-V14) %>%
    gather(var, temp, -V1, na.rm =T) %>%
    mutate(Time = ymd(paste(V1, var, "01", sep = "-"))) %>%
    select(Time, temp) %>%
    arrange(Time)
  # Rename the temp column to Zm if scaling was required
  if(scale == T) colnames(mdata)[2] <- "Zm"
  # Return
  return(mdata)
}
```

Estimator functions

```r
#' Compute cos(Yj)
#'
#' \code{cosYd} is an helping function for the characteristic function estimators
#' @param Yj vector - FBM process
#' @param d numeric
#' @param lambda numeric parameter, default=0.5
#' @param na.remove logical
#' @return vector
cosYd <- function(Yj, d=d, lambda=0.5, na.remove=F){
  if(na.remove == T) {Yj <- Yj[which(Yj!="NA")]}
  Tj <- length(Yj)
  Ydiff <- Yj[(d+1):Tj]-Yj[1:(Tj-d)]
  kYdiff <- rep(Ydiff,each=(Tj-d))
  rYdiff <- rep(Ydiff,(Tj-d))
  f <- sqrt(sum(cos(lambda*(kYdiff-rYdiff)/sqrt(d))))/(Tj-d)
  return(f)}


#' Mu estimation by characteristic function
#'
#' @param Zj vector, Fractional Gaussian Noise process
#' @param lambda parameter, default = 0.5
#' @return numeric, mu estimator
#' @export
estim.cf.mu <- function(Zj, lambda=0.5){
  S <- (1/T)*sum(sin(lambda*Zj))
  C <- (1/T)*sum(cos(lambda*Zj))
  mu <- (1/lambda)*atan(S/C)
  names(mu) <- "Mean"
  return(mu)}


#' Sigma and H estim. by characteristic function
#'
#' @param Yj vector, Fractional Brownian Motion process
#' @param maxd numeric, default = 10
#' @param lambda parameter, default = 0.5
#' @param FBM logical, whether the Yj vector is a FBM. Dafault is T
#' @param na.remove logical
#' @return vector, sigma and H estimators
#' @export
estim.cf.reg <- function(Yj, maxd=10, FBM=T, lambda=0.5, na.remove=F){
  if(FBM==F){Yj <- cumsum(Yj)}
  dgraph <- numeric(maxd)
  for (d in 1:maxd){
    dgraph[d] <- cosYd(Yj, d=d, lambda=lambda, na.remove=na.remove)
  }
  yy <- log(-log(abs(dgraph)))
  xx <- log(1:maxd)
  coef <- lm(yy~xx)$coefficients
  sigma <- 2*exp(coef[1]-2*log(lambda))
```

```r
  Hreg <- (coef[2]+1)/2
  res <- c(sigma,Hreg)
  names(res) <- c("Sigma","H")
  return(res)}



#' Sigma and H estim. by characteristic function
#'
#' @param Yj vector, Fractional Brownian Motion process
#' @param maxd numeric, default = 10
#' @param lambda parameter, default = 0.5
#' @param FBM logical, whether the Yj vector is a FBM. Dafault is T
#' @param na.remove logical
#' @return numeric, sigma estimator
#' @export
estim.cf.sigma <- function(Yj, maxd=10, FBM=T, lambda=0.5, na.remove=F){
  sigma <- estim.cf.reg(Yj, maxd=maxd, FBM=FBM, lambda=lambda, na.remove=na.remove)[1]
  return(sigma)}



#' H estim. by characteristic function
#'
#' @param Yj vector, Fractional Brownian Motion process
#' @param maxd numeric, default = 10
#' @param lambda parameter, default = 0.5
#' @param FBM logical, whether the Yj vector is a FBM. Dafault is T
#' @param na.remove logical
#' @return numeric, H estimator
#' @export
estim.cf.H <- function(Yj, maxd=10, FBM=T, lambda=0.5, na.remove=F){
  H <- estim.cf.reg(Yj, maxd=maxd, FBM=FBM, lambda=lambda, na.remove=na.remove)[2]
  return(H)}



#' H estim. by Whittle method
#'
#' @param Zj vector, FGN process
#' @return numeric, H estimator
#' @importFrom longmemo WhittleEst
#' @export
estim.w.H <- function(Zj) {
  Hw <- WhittleEst(Zj)$coefficients[1]
  names(Hw) <- "Hw"
  return(Hw)}



#' Alfa estim. by characteristic function
#'
#' @param Yj vector, Fractional Brownian Motion process
```

```
#' @param lambda parameter, default = 0.5
#' @param FBM logical, whether the Yj vector is a FBM. Dafault is T
#' @param na.remove logical
#' @return numeric, alpha estimator
#' @export
estim.cf.alpha <- function(Yj, FBM=T, lambda=lambda, na.remove=F){
  if(FBM==F){Yj <- cumsum(Yj)}
  l <- seq(0.1,1,by=0.1)
  lgraph <- numeric(10)
  for (i in 1:10){
    lgraph[i] <- cosYd(Yj=Yj,d=2,lambda=l[i])
  }
  yy <- log(-log(abs(lgraph)))
  alpha <- lm(yy~log(l))$coefficients[2]
  names(alpha) = "Alpha"
  return(alpha)}



#' Coefficients estim. by characteristic function
#'
#' @param Zj vector, Fractional Gaussian Noise process
#' @param maxd numeric, default = 10
#' @param lambda parameter, default = 0.5
#' @param FBM logical, whether the Yj vector is a FBM. Dafault is T
#' @param na.remove logical
#' @return vector, Mu, Sigma, H and Alpha estimators
#' @export
estim.cf.coef <- function(Zj, maxd=10, FBM=T, lambda=0.5, na.remove=F){
  Mean <- estim.cf.mu(Zj, lambda=lambda)
  reg <- estim.cf.reg(Zj, maxd=maxd, FBM=F, lambda=lambda, na.remove=na.remove)
  Alpha <- estim.cf.alpha(Zj, FBM=F, lambda=lambda, na.remove=na.remove)
  out <- c(Mean, reg, Alpha)
  return(out)
}



# Omega() function in the test-functions.R file

#' Maximum likelihood estimator for the mean
#'
#' @param Xj vector - FGN process
#' @param H numeric - H parameter
#' @param sigma numeric - Sigma value, default=1
#' @export
FgnMean <- function(Xj,H=H,sigma=1){
  Xjna <- Xj[!is.na(Xj)]
  n <- length(Xjna)
  OmegaInv <- solve(Omega(H=H,n=n))
  mu <- sum(OmegaInv%*%Xjna)/sum(OmegaInv)}
```

```r
# Omega() function in the test-functions.R file

#' Maximum likelihood estimator for the standard deviation
#'
#' @param Xj vector - FGN process
#' @param mean numeric - Estimated mean of the FGN process
#' @export
FgnVar <- function(Xj,mean=mean){
  Xjna <- Xj[!is.na(Xj)]
  n <- length(Xjna)
  var <- sum((Xjna-rep(mean,n))^2)/n}
```

Test functions

```r
#' Self-similarity graphical test
#'
#' @param Yj vector - FBM process
#' @param maxd numeric - Parameter, default=10
#' @param lambda numeric - Parameter, default=0.5
#' @param main string, title
#' @param subtitle logical. Defines if a subtitle should be printed
#' @param cex.axis numeric
#' @param cex.lab numeric
#' @param cex.main numeric
#' @param cex.dots numeric
#' @param lwd numeric
#' @export
fgtSelfSim <- function(Yj=Yj, maxd=10, lambda=0.5, main=NULL, subtitle = T,
                       cex.axis=0.8, cex.lab=0.8, cex.main=1, cex.dots=1, lwd=1){
  dgraph <- NULL
  for (d in 1:maxd){
    fun <- cosYd(Yj=Yj,d=d,lambda=lambda)
    dgraph <- c(dgraph,fun)}
  yy <- log(-log(abs(dgraph)))
  xx <- log(1:maxd)
  aa <- lm(yy~xx)$coefficients[1] # Intercept of the regression line
  bb <- lm(yy~xx)$coefficients[2] # Slope of the regression line
  Hreg <- (bb+1)/2    # Estimated (by regression) H
  par(bty="l",cex.axis=cex.axis,cex.lab=cex.lab,cex.main=cex.main,
      font.lab=2,mar=.1+c(5,4,4,2),las=1)
  p <- plot(xx,yy,pch=16,xlab="",ylab="",
            main=paste("Self-similarity test",main),cex=cex.dots,
            panel.first=abline(v=axTicks(1),h=axTicks(2),col="grey90"))
  abline(a=aa,b=bb,lty=2,col=2,lwd=lwd)
  if(subtitle ==T) {mtext(paste("Estimated H by regression =",round(Hreg,2)),
                          side=3,line=0.2,cex=0.7)}
  return(list(x=xx, y=yy, intercept=aa, slope=bb, Hregression=Hreg, plot=p))}



#' Normality graphical test
#'
#' @param Yj vector - FBM process
#' @param xmax numeric - Parameter, default=1
#' @param lambda numeric - Parameter, default=0.5
#' @param main string, title
#' @param cex.axis numeric
#' @param cex.lab numeric
#' @param cex.main numeric
#' @param cex.dots numeric
#' @param lwd numeric
#' @export
fgtNormality <- function(Yj=Yj, xmax=1, main=NULL,cex.axis=0.8,
                         cex.lab=0.8,cex.main=1,cex.dots=1,lwd=1){
  xx <- seq(0.1, xmax, by=0.1)
```

```r
  dd <- 2
  lgraph <- NULL
  for (l in xx){
    fun <- cosYd(Yj=Yj,d=dd,lambda=l)
    lgraph <- c(lgraph,fun)}
  yy <- log(-log(abs(lgraph)))
  aa <- lm(yy~log(xx))$coefficients[1]
  bb <- lm(yy~log(xx))$coefficients[2]
  par(bty="l",cex.axis=cex.axis,cex.lab=cex.lab,cex.main=cex.main,
      font.lab=2,mar=.1+c(5,4,4,2),las=1)
  plot(log(xx),yy,pch=16,xlab="",ylab="",cex=cex.dots,
       main=paste("Normality test",main),
       panel.first=abline(v=axTicks(1),h=axTicks(2),col="grey90"))
  abline(a=aa,b=bb,lty=2,col=2,lwd=lwd)
  mtext(paste("Estimated alpha =",round(bb,2)),
        side=3,line=0.2,cex=0.7)
  return(list(x=xx,y=yy,intercept=aa,slope=bb))}



#' Omega matrix - Helping function for ML estimators and Chi-Square test
#'
#' @param H numeric - H parameter
#' @param n numeric
#' @param sigma numeric - Sigma value, default=1
Omega <- function(H=H,n=n,sigma=1){
  d <- abs(matrix(rep(1:n,n),nrow=n)-matrix(rep(1:n,n),nrow=n,byrow=T))
  autocorr <- (sigma^2/2)*(abs(d+1)^(2*H)-2*(abs(d)^(2*H))+abs(d-1)^(2*H))
  return(autocorr)}



#' Chi-Square test
#'
#' @param Zj vector - FGN process
#' @param  H numeric - H parameter
#' @param TT numeric - Length of the time series
#' @export
Qstat <- function(Zj, H=H, TT=TT){
  OmegaInv <- solve(Omega(H=H,n=TT))
  Q <- ((t(Zj)%*%OmegaInv%*%Zj)-TT)/sqrt(2*TT)
  return(Q)
}


#' Significance - Stars
#'
#' @param x vector
#' @export
signZ <- function(x) {
  ifelse(x<qnorm(0.005)|x>qnorm(0.995),"***",
         ifelse(x<qnorm(0.025)|x>qnorm(0.975),"*"," "))}
```

Plots functions

```r
#' Temperature plot
#'
#' @param Xj vector - Temperature process
#' @param Year vector - Years of recorded data
#' @param main string - Title, default=NULL
#' @param cex.axis numeric, default=0.8
#' @param cex.lab numeric, default=0.8
#' @param cex.main numeric, default=1
#' @param break.val logical - Defines if there is a breaking value, after which
#' the data should be represented with a different linetype. Defaul=F
#' @export
temperaturePlot <- function(Xj=Xj, Year=Year, main=NULL, cex.axis=0.8,
                            cex.lab=0.8, cex.main=1, break.val=F){
  par(bty="l",cex.axis=cex.axis,cex.lab=cex.lab,cex.main=cex.main,
      font.lab=2,las=1)
  plot(Year,Xj,type="n",main=main,ylab="",xlab="",
       panel.first=abline(v=axTicks(1),h=axTicks(2),col="grey90"))
  if(break.val==F){
    lines(Year,Xj,col=4,lwd=2)}
  else {
    nb <- which(Year==break.val)
    ny <- length(Year)
    lines(Year[1:nb],Xj[1:nb],col=4,lwd=2)
    lines(Year[nb:ny],Xj[nb:ny],lwd=1,col=4)}
  abline(h=mean(Xj,na.rm=T),col=2,lty=2)}



#' Compute theoretical autocorrelation
#'
#' @param d numeric
#' @param H numeric, H parameter value
th.autocorr <- function(d,H) {
  0.5*((d+1)^(2*H)-2*(d)^(2*H)+abs(d-1)^(2*H))}



#' Autocorrelation plot
#'
#' @param Xj vector - FGN process
#' @param maxd numeric - Numbers of lags to be plotted
#' @param lambda numeric, default=0.5
#' @param H logical - Define if a theoretical autocorrelation is plotted and how .
#' The possible options for H are:
#' H=F          No theoretical autocorrelation is plotted
#' H=x          with x=number. The autocorrelation is computed with H=x
#' H="cfH"      H is estimated via ch. function estimator, and then used to plot the
#' th. autocorr.
#' @param main string - Title, default=NULL
#' @param cex.axis numeric, default=0.8
#' @param cex.lab numeric, default=0.8
```

```r
#' @param cex.main numeric, default=1
#' @param cex.dots numeric, default=1
#' @param ylim vector
#' @param labels logical. Whether the numerical values of the empirical
#' autocorrelation has to be plotted
#' @param correct logical. If the computation of the autocorrelation has
#' to be corrected
#' @export
autocorrPlot <- function(Xj, maxd=10, H=F, cex.axis=0.8, lambda=0.5,
                         main=NULL, cex.lab=0.8, cex.main=1, cex.dots=1,
                         ylim=c(min(min(rho),-0.5),1), labels=T,
                         correct=F){
  Xj <- Xj[which(Xj!="NA")]  # Remove missing data
  Tj <- length(Xj)
  if (H!=F) {
    if (H=="cfH") H <- estim.cf.H(Yj=Xj, maxd=maxd, lambda=lambda, FBM=F)}
  rho <- NULL    # Future vector of autocorrelation
  for (i in 1:maxd){
    rho <- c(rho,cor(Xj[(i+1):Tj],Xj[1:(Tj-i)]))}
  if (correct==T) {
    rho <- (rho+Tj^(2*H-2))/(1+Tj^(2*H-2))}
  par(bty="l",cex.axis=cex.axis,cex.lab=cex.lab,cex.main=cex.main,
      font.lab=2,las=1)
  p <- plot(1:maxd, rho, pch=16, type="o", lwd=2, cex=cex.dots,
            col=4, xlab="Lag", ylab=expression(rho),
            ylim=ylim,
            main=paste(main,"Autocorrelation"),
            panel.first=abline(v=axTicks(1), h=axTicks(2), col="grey90"))
  abline(h=0,lty=2,col=4)
  if(labels==T) {text(1:maxd,(rho+0.15),round(rho,digits=2),col=4,cex=cex.lab)}
  if(H!=F){
    thCorr <- th.autocorr(1:maxd,H)
    lines(thCorr,col=2,lwd=2)
    legend("bottomright",c("Empirical autocorrelation",
                           paste("FGN theorical autocorrelation, H=",
                                 round(H,2))),
           lwd=c(2,2),col=c(4,2),cex=cex.lab,bty="n")}
  return(list(plot=p, rho=rho))
}



#' Blue-red plot
#'
#' @param Zj vector - Temperature process, usually normalized
#' @param Time vector - Time of recorded data
#' @param mean numeric - Mean value, default=0
#' @param ic vector - Confidence inteval, default=c(-3,3)
#' @param main string - Title, default=NULL
#' @param cex.axis numeric, default=0.8
#' @param cex.lab numeric, default=0.8
#' @param cex.main numeric, default=1
#' @param break.val logical - Defines if there is a breaking value, after which
```

```r
#' the data should be represented with a different color. Defaul=F
#' @export
blueRedPlot <- function(Zj=Zj, Time=Time, mean=0, ic=c(-3,3),
                        main=NULL, cex.axis=0.8,
                        cex.lab=0.8, cex.main=1, break.val=F){
  blueRed <- ifelse(Zj>mean,"red","blue")
  if (break.val!=F) {
    blueRed <- replace(blueRed,which(Time>break.val&Zj>mean),
                       "coral")
    blueRed <- replace(blueRed,which(Time>break.val&Zj<mean),
                       "steelblue2")}
  par(bty="l",cex.axis=cex.axis,cex.lab=cex.lab,cex.main=cex.main,
      font.lab=2,las=1)
  plot(Time,Zj,type="h",ylab="",xlab="",col=blueRed,lwd=3,
       main=paste(main,"Deviation from the mean"),
       ylim=c(min(min(Zj,na.rm=T),ic[1]),max(max(Zj,na.rm=T),ic[2])),
       panel.first=abline(v=axTicks(1),h=axTicks(2),col="grey90"))
  abline(h=c(ic[1],ic[2]),lty=2,col=c(4,2))
  abline(h=0,lwd=3)}
```

Autocorrelation functions

```r
sim.multiFGN <- function(N, Tj, H){
  1:N %>%
    map(~ as.vector(longmemo::simFGN0(n = Tj, H = H))) %>%
    map_dfc(~ .)
}
```

```r
autocorrelation_function <- function(d, X){
  TJ <- length(t(X))
  X_mean <- mean(X, na.rm = T)
  Xdiff_den <- sum((X[1:TJ] - X_mean)^2)
  Pk <- c()
  for (k in 1:d) {
    Xdiff_num <- (X[(1 + k) : TJ] - X_mean)*(X[1:(TJ - k)] - X_mean)
    Rk <- (sum(Xdiff_num))/((Xdiff_den))
    Pk[k] <- (Rk + TJ^(-0.1))/(1 + TJ^(-0.1))
  }
  return(Pk)
}

Theoretical_autocorrelation_function <- function(d){
  gammak <- c()
  for (k in 1:d) {
    gammak[k] <- 0.5*((k+1)^(1.9)-2*(k^1.9)+(k-1)^(1.9))}
  return(gammak)
}

unbiased_autocorrelation_function <- function(d, X) {
  Rk <- true_autocorrelation_function(d, X)
  Pk_new <- c()
```

```
  for (k in 1:d) {
    Pk_new[k] <- (Rk[k] + 0.9)/1.9}
  return(Pk_new)
}
```