```
#' Prewhitens a Time Series
#'
#' Variance flattening across frequencies
#'
#' @param dat A \code{data.frame} with each column containing a series
to be prewhitened. See details.
#' @param order An \code{integer} giving the order of AR to use.
#' @param nw A \code{numeric} specifying the time bandwidth parameter
#' @param k An \code{integer} specifying number of tapers.
#' @param nFFT.add An \code{integer} indicating how much zero-padding
should be done.
#' Should be at least 1.
#' @param commonFilter A \code{logical} indicating whether the same
prewhitening filter should
#' be used on each series (uses the average of the log-spectra if TRUE).
#' @param FtestCutoff Significant level to detect a line component to be
removed before prewhitening
#' (value between 0 and 1).
#' @param nw.pr NW to use with periodic reconstruction / line component
removal.
#' @param k.pr Number of tapers to use with periodic reconstruction /
line component removal.
#' @param returnDataOnly Whether to return the data columns only (TRUE,
default) or all of the columns
#' of the original data.frame (FALSE).
#' @param excludeCols Which columns should not be prewhitened.
#'
#' @export
prewhiten <- function(dat, order = 2, nw = 4, k = 7, nFFT.add = 2,
commonFilter = TRUE
                      , FtestCutoff = NULL, nw.pr = NULL, k.pr = NULL
                      , returnDataOnly = TRUE
                      , excludeCols = c("date", "doy", "Date", "DOY",
"day", "Day"
                                        , "month", "Month", "year",
"Year", "time", "Time")){

  dateCols <- dat[, which(colnames(dat) %in% excludeCols), drop = FALSE]
  dat <- dat[, which(!(colnames(dat) %in% excludeCols)), drop = FALSE]
  # remove the overall mean
  # dat.one <- dat - mean(dat)

  # spec.one <- welchEst(dat.two, deltaT = 1, nw = nw, k = k, blockSize
= blockSize, nodes=nodes
  #                      , overlap = overlap)

  if (ncol(dat) > 1 & commonFilter == FALSE){
    stop("Use commonFilter = TRUE or only prewhiten one series at a
time.")
  }

  if (!is.null(FtestCutoff) & is.null(nw.pr)){
    nw.pr <- nw
```

```
    k.pr <- k
  }

  # round up to the nearest power of two, then add nFFT.add (which
should be at least 1).
  nFFT <- 2^(ceiling(log2(nrow(dat))) + nFFT.add)

  # remove periodic components first
  if (!is.null(FtestCutoff)){
    recon <- list()
    recon <- lapply(dat, function(dd){
      driegert::periodicRecon(dd, FtestCutoff = FtestCutoff, nFFT.add =
nFFT.add + 1
                              , nw = nw.pr, k = k.pr)
    })
    vn <- names(dat)
    dat4sp <- list()
    for (i in 1:length(vn)){
      dat4sp[[vn[i]]] <- dat[[vn[i]]] - mean(dat[[vn[i]]]) -
recon[[vn[i]]]
    }
  } else {
    dat4sp <- dat
  }

  spec <- as.data.frame(lapply(dat4sp, function(dd){
    multitaper::spec.mtm(dd, nw = nw, k = k, deltat = 1, nFFT = nFFT,
plot = FALSE)$spec
  }))

  if (commonFilter){
    spec.ave <- exp(apply(log(spec), 1, mean))
    tmp <- determineArCoef(spec.ave, order)
    phi <- tmp$phi
    acf <- tmp$acf

    dat.pw <- lapply(dat, function(dd){
      # dd - fitAr(dd, phi)
      fitAr(dd, phi) - dd
    })
  } else {
    tmp <- lapply(spec, function(ss){
      determineArCoef(ss, order)
    })
    phi <- lapply(tmp, "[[", "phi")
    acf <- lapply(tmp, "[[", "acf")

    # dat.pw <- dat - fitAr(dat, phi) # prewhitened series
    dat.pw <- fitAr(dat[, 1, drop=TRUE], phi[[1]]) - dat[, 1, drop=TRUE]
  }

  if (!returnDataOnly){
    cbind(dateCols, dat.pw)[-(1:order), ]
  } else {
```

```
    list(dat.pw = dat.pw, phi = phi, acf = acf)
  }
}

correctForPrewhiten <- function(x, phi, eigenCoefficient = TRUE){
  if (eigenCoefficient){
    M <- dim(x)[1]
    ncols <- dim(x)[2]
    ndims <- length(dim(x))
  } else {
    M <- length(x)
  }

  pwCor <- fft(c(-1, phi, rep(0, M-length(phi)-1)))

  if (eigenCoefficient){
    for (i in 1:ncols){
      # In MT data I have 4D data (freq x taper x block x direction)
      if (ndims > 2){
        for (j in 1:dim(x)[3]){
          x[, i, j] <- x[, i, j] / pwCor
        }
      } else {
        x[, i] <- x[, i] / pwCor
      }
    }
  } else {
    x <- x / abs(pwCor)^2
  }

  x
}

# I'm not using the welchSpec version anymore
determineArCoef <- function(welchSpec, order){
  # calculate the autocovariances
  ### Need to divide by the length of the series - I don't think this
matters for the solution..
  fullSpec <- c(welchSpec, rev(welchSpec[-c(1, length(welchSpec))]))
  acvs <- Re(fft(fullSpec, inverse=TRUE)) / length(fullSpec)

  ## using multiple series
  #acf.ave <- tanh((atanh(acf.x[2:(order+1)]) + atanh(acf.y[2:
(order+1)]) + atanh(acf.z[2:(order+1)])) / 3)

  # max order to use
  ac <- acvs[1:(order+1)] / acvs[1]

  phi <- solvePhi(ac, order)

  list(phi = phi, acf = ac)
}

solvePhi <- function(ac, order){
```

```
  # Create the autocovariance matrix
  R <-  matrix(NA, nrow = order, ncol= order)
  R[1, ] <- ac[1:order]
  if (order > 2){
    for (i in 2:(order-1)){
      R[i, ] <- c(rev(ac[1:i]), ac[2:(order - i+1)])
    }
  }
  R[order, ] <- rev(R[1, ])

  # solve for the AR coefficients
  phi <- solve(R, ac[2:(order+1)])

  phi
}

fitAr <- function(dat, phi){
  order <- length(phi)
  fitted <- rep(NA, length(dat))
  for (i in (order+1):length(dat)){
    fitted[i] <- sum(dat[(i-1):(i-order)] * phi)
  }

  fitted
}

correctHforPw <- function(Hi, phi){
  # pw.H <- fft(c(-1, phi, rep(0, tt.sp$mtm$nFFT - 4)))
[1:tt.sp$mtm$nfreqs]

  ## Use nFFTx and nfreqy I believe as the AR coefficients deltat
correspond to the
  # x-series sampling rate, but then we only need the first nfreqy
frequencies anyway.
  pw.H <- fft(c(-1, phi, rep(0, Hi$param[["nFFTx"]] - (length(phi) +
1))))[ 1:Hi$param[["nfreqy"]] ]

  cIdx <- param[["nFFTx"]] / 2 #?
  # pw.H <- c(rev(pw.H.tmp[-1])[-1], pw.H.tmp)

  Hcor <- Hi$H
  for (i in 1:ncol(Hi$H)){
    Hcor[, i] <- Hi$H / pw.H
  }

  Hcor
```