# R Package Development by Means of Literate Programming (noweb)

Bernhard Pfaff

January 4, 2017

# 1 Introduction

# 2 Detecting Peaks/Troughs

## 2.1 Notation

A uniformly sampled time series $\mathbf{y} = \{y_1, \ldots, y_i, \ldots, y_T\}$ with $T$ data points is considered. The detection of peak/trough points is achieved by a function $S(i, y_i, T)$ that returns for data point $y_i$ a score value.[1]. If this score value surpasses a user-provided threshhold value $\theta$, /i.e/, $S(i, y_i, T) \geq \theta$ then the point is considered as a local peak/trough.

Furthermore, in case local peak/trough points appear closely together with respect to time (clustered), then these points can be classified as a burst or bust, respectively.

## 2.2 Algorithms

In Palshikar (2009) five different score functions $S$ have been suggested. All have in commom, that a centred window of size $2*k+1$ around $y_i$ is considered. That is, for a positive integer $k$ the $k$ right neigbours $N^+(i, k, T) = \{y_{i+1}, \ldots, y_{i+k}\}$ and the $k$ left neighbours $N^-(i, k, T) = \{\{y_{i-k}, \ldots, y_{i-1}\}$ are employed for assessing $y\_i$ as a local peak/trough. The union of $N^-(i, k, T)$ and $N^+(i, k, T)$ is defined as $N(i, k, T) = N^-(i, k, T) \cdot N^+(i, k, T)$ and if the centre point is included as $N'(i, k, T) = N^-(i, k, T) \cdot y_i \cdot N^+(i, k, T)$.

The first function, $S_1$, computes the score value as the average of the maximum differences between $y_i$ with its left and right neighbours. The function is defined as:

$$S_1 = \frac{\max(y_i - y_{i-1}, \ldots, y_i - y_{i-k}) + \max(y_i - y_{i+1}, \ldots, y_i - y_{i+k})}{2} \tag{1}$$

The equation (1) can be casted in R as:

1 ⟨*score-maxdiff* 1⟩≡
```
scmaxdiff <- function(x, k){
```

[1] It suffices to provide a score function for peaks only. Trough points can be detected by using the negative values of the series $\mathbf{y}$.

```
    cp <- k + 1L
    lmax <- max(x[cp] - head(x, k))
    rmax <- max(x[cp] - tail(x, k))
    (lmax + rmax) / 2.0
}
```

Defines:
`scmaxdiff`, used in chunks 4–6.

Instead of using the maximum differences of $y_i$ with its $k$ left and right neighbours as in (1), an alternative is to compute the mean differences and evaluate the average thereof:

$$S_2 = \frac{\frac{(y_i - y_{i-1}, \ldots, y_i - y_{i-k})}{k} + \frac{(y_i - y_{i+1}, \ldots, y_i - y_{i+k})}{k}}{2} \tag{2}$$

This equation can be casted in R as:

2a ⟨*score-diffmean* 2a⟩≡

```
scdiffmean <- function(x, k){
    cp <- k + 1L
    ldmean <- x[cp] - mean(head(x, k))
    rdmean <- x[cp] - mean(tail(x, k))
    (ldmean + rdmean) / 2.0
}
```

Defines:
`scdiffmean`, used in chunks 4–6.

Another variation of score computation that has been proposed by Palshikar (2009) is to consider the differences to the mean of the $k$ left and right neighbours, that is:

$$S_3 = \frac{(y_i - \frac{(y_{i-1}, \ldots, y_{i-k})}{k}) + (y_i - \frac{(y_{i+1}, \ldots, y_{i+k})}{k})}{2} \tag{3}$$

The equation (3) can be casted as R function `scavgdiff` for instance as follows:

2b ⟨*score-avgdiff* 2b⟩≡

```
scavgdiff <- function(x, k){
    cp <- k + 1L
    lmean <- mean(x[cp] - head(x, k))
    rmean <- mean(x[cp] - tail(x, k))
    (lmean + rmean) / 2.0
}
```

Defines:
`scavgdiff`, used in chunks 4–6.

The fourth proposed score function differs from the previous three in the sense that it does take explicitly the differences between $y_i$ and its neighbours explicitly into account, but tries to capture its information content by means of relative entropy. The entropy of a vector $A$ with elements $A = \{a_1, \ldots, a_m\}$ is given as:

$$H_w(A) = \sum_{i=1}^{M} (-p_w(a_i) \log(p_w(a_i))) \tag{4}$$

where $p_w(a_i)$ is an estimate of the density value at $a_i$. The score function is now based on computing the entropies of $H(N((k, i, T))$ and $H(N'(k, i, T))$. Hereby, the densities can be determined by means of a kernel density estimator. The score function is then defined as the difference of the entropies:

$$S_4 = H(N((k, i, T)) - H(N'((k, i, T)) \tag{5}$$

This concept is implemented in the function `scentropy()`. The empirical density is computed by calling `density()`. The ellipsis argument of `scentropy()` is passed down to this function and hereby allowing the user to employ other than the default arguments of `density()`.

3 ⟨*score-entropy* 3⟩≡

```
scentropy <- function(x, k, ...){
    cp <- k + 1L
    dfull <- density(x, ...)$y
    hfull <- sum(-dfull * log(dfull))
    dexct <- density(x[-cp], ...)$y
    hexct <- sum(-dexct * log(dexct))
    hfull - hexct
}
```

Defines:
`scentropy`, used in chunks 4–6.

Finally, a moment-based score function has been put forward in the article by Palshikar. Hereby, the first and second moment of $N((k, i, T))$ are computed and a t-type statistic can be computed as $(y_i - m)/s$. If this statistic surpasses a provided threshhold $h$, then the data point is considered as a local peak/trough.

$$S_5 = \begin{cases} 1 & (y_i - m)/s \geq h \\ 0 & \text{else} \end{cases} \tag{6}$$

This type of scoring algorithm is implemented as function `scttype()` below:

4a ⟨*score-ttype* 4a⟩≡

```
scttype <- function(x, k, tval){
    cp <- k + 1L
    m <- mean(x[-cp])
    s <- sd(x[-cp])
    tstat <- (x[cp] - m) / s
    if ( abs(tstat) < tval ){
        tstat <- 0
    }
    tstat
}
```

Defines:
`scttype`, used in chunks 4–6.

Incidentally, an ensemble forecast of these five algorithms can be utilized for local peak/trough classification can be employed. Hereby, one could either use a hybrid approach, whereby only those data points are considered as peak/trough points, if all five methods coincide. This concept is casted in the function `schybrid()`. Hereby, the signs of all five scoring algorithm are tested for equality.

4b ⟨*score-hybrid* 4b⟩≡

```
schybrid <- function(x, k, tval, ...){
    s <- c(sign(scmaxdiff(x, k)),
           sign(scavgdiff(x, k)),
           sign(scdiffmean(x, k)),
           sign(scentropy(x, k, ...)),
           sign(scttype(x, k, tval)))
    val <- unique(s)
    if ( length(val) < 2 ){
        return(s[1])
    } else {
        return(0)
    }
}
```

Defines:
`schybrid`, used in chunk 6a.
Uses `scavgdiff` 2b, `scdiffmean` 2a, `scentropy` 3, `scmaxdiff` 1, and `scttype` 4a.

It is also conceivable to base the classification on a majority vote. For instance, if three out of the five algorithm classify a data point as a local peak/trough, then this is taken as sufficient evidence. This approach is defined in the function `scvote()` below. The count of same 'votes' is set by the argument `confby`. Its default value is 3, *i.e.* a simple majority. For `confby = 5` the function would return the same classification as `schybrid()` does.

5 ⟨*score-vote* 5⟩≡

```
scvote <- function(x, k, tval, confby = 3, ...){
    s <- c(sign(scmaxdiff(x, k)),
           sign(scavgdiff(x, k)),
           sign(scdiffmean(x, k)),
           sign(scentropy(x, k, ...)),
           sign(scttype(x, k, tval)))
    pos <- rep(1, 5)
    zer <- rep(0, 5)
    neg <- rep(-1, 5)
    spos <- sum(s == pos)
    szer <- sum(s == zer)
    sneg <- sum(s == neg)
    v <- c(spos, szer, sneg)
    idx <- which(v >= confby)
    vals <- c(1, 0, -1)
    if ( length(idx) > 0 ){
        return(vals[idx])
    } else {
        return(0)
    }
}
```

Defines:
`scvote`, used in chunk 6a.
Uses `scavgdiff` 2b, `scdiffmean` 2a, `scentropy` 3, `scmaxdiff` 1, and `scttype` 4a.

## 2.3 Combining score methods

6a ⟨*score-wrapper* 6a⟩≡

```
score <- function(x, k,
                  scoreby = c("vote", "avg", "diff", "max", "ent",
                              "ttype", "hybrid"),
                  tval = 1.0, confby = 3, ...){
    scoreby <- match.arg(scoreby)
    ans <- switch(scoreby,
                  vote = scvote(x, k, tval, confby, ...),
                  avg = scavgdiff(x, k),
                  diff = scdiffmean(x, k),
                  max = scmaxdiff(x, k),
                  ent = scentropy(x, k, ...),
                  ttype = scttype(x, k, tval),
                  hybrid = schybrid(x, k, tval, ...)
                  )
    ans
}
```

Defines:
`score`, used in chunk 7.
Uses `scavgdiff` 2b, `scdiffmean` 2a, `scentropy` 3, `schybrid` 4b, `scmaxdiff` 1, `scttype` 4a, and `scvote` 5.

The content/structure of the file `score.R` is given as:

6b ⟨*score.R* 6b⟩≡

```
⟨man-func-score 22⟩
⟨score-wrapper 6a⟩
#' @rdname score
⟨score-maxdiff 1⟩
#' @rdname score
⟨score-diffmean 2a⟩
#' @rdname score
⟨score-avgdiff 2b⟩
#' @rdname score
⟨score-entropy 3⟩
#' @rdname score
⟨score-ttype 4a⟩
#' @rdname score
⟨score-hybrid 4b⟩
#' @rdname score
⟨score-vote 5⟩
```

This code is written to file `score.R`.

@

Within this file, all score-related methods and the wrapper-function `score()` is included. The function definitions are interspersed with the roxygen tags, which will be parsed to the `Rd`-file `score.Rd`.

So far the function `score()` has been created, by which a single point is assessed for being a local maximum or minimum. For analyzing a whole time series for its local extrema, this routine can be applied to each data point and its left/right neighbours. This task is accomplished with the function `hiker()` as defined next.

7 ⟨*hiker-func* 7⟩≡

```
hiker <- function(y, k,
                  scoreby = c("vote", "avg", "diff", "max", "ent",
                              "ttype", "hybrid"),
                  tval = 1.0, confby = 3, ...){
```

⟨*hiker-check* 8⟩

```
    ## rolling centered window for peak scores
    s <- rollapply(y, width = ms, FUN = score,
                   k = k, scoreby = scoreby, tval = tval, ...)
```

⟨*hiker-output* 9a⟩

```
}
```

Uses `score` 6a.

@

The arguments of the function are `y` for the time series object, `k` for the count of left/right neighbours, and `scoreby` for the selection of the scoring method. The arguments `tval` and `confby` belong the scoring concepts 'ttype' and 'hybrid', respectively, and the ellipsis argument is passed down to the call of `scentropy()` for `scoreby = 'ent'`.

The function body consists of three parts. First, the provided arguments are checked for their validity (as shown in the following code chunk). The computation of the scores is accomplished with the `rollapply()` function of the package **zoo**. Finally, the returned object is created.

8 ⟨*hiker-check* 8⟩≡

```
y <- as.zoo(y)
## checking arguments
k <- as.integer(abs(k))
ms <- 2 * k + 1L
if ( is.null(dim(y)) ){
    yname <- "series"
    n <- length(y)
    if ( n < ms ) {
        stop(paste("Sample size of 'y' is ", n,
                   " and k = ", k, ".\n", sep = ""))
        }
} else {
    n <- nrow(y)
    yname <- colnames(y)[1]
    if ( n < ms ) {
        stop(paste("Sample size of 'y' is ", n,
                   " and k = ", k, ".\n", sep = ""))
    }
    if ( ncol(y) > 1 ) {
        stop("Provide univariate time series of S3-class 'zoo'.\n")
    }
}
if ( (confby < 3) || (confby > 5) ){
    stop("\nArgument 'confby' must be integer and in set {3, 4, 5}.\n")
}
scoreby <- match.arg(scoreby)
```

@

Within the check section of the function body, the object `y` is first coerced to a `zoo` object and the count of neighbours is coerced to a positive integer. Next, the size of the sub-sample for computing the scores is assigned to the object `ms`. The remaining part consists of ckecks whether the series is univariate and its length is at least $2 \times k + 1$. Finally, the scoring method is determined from the argument `scoreby` by means of the `match.arg` function.

9a ⟨*hiker-output* 9a⟩≡

```
## merging time series and scores
ans <- merge(y, s)
colnames(ans) <- c("Series", "Scores")
des <- switch(scoreby,
              vote = "majority vote",
              avg = "average of averaged differences",
              diff = "average of mean differences",
              max = "average of maximum differences",
              ent = "difference of entropies",
              ttype = "t-type statistic",
              hybrid = "hybrid")
new("HikeR", ys = ans, k = k, scoreby = des, yname = yname)
```

@

9b ⟨*hiker.R* 9b⟩≡

⟨*man-func-hiker* 23a⟩
⟨*hiker-func* 7⟩

This code is written to file `hiker.R`.

@

# 3 Package structure

## 3.1 Preliminaries

First, a skeleton of the package

9c ⟨*DESCRIPTION.R* 9c⟩≡

```
Package: hiker
Title: Local Peak and Trough of a Time Series
Version: 0.0.0.9000
Authors@R: person("Bernhard", "Pfaff", email = "bernhard@pfaffikus.de",
                  role = c("aut", "cre"))
Description: Methods for detecting local peaks and troughs of a time series.
Depends: R (>= 3.3.1), zoo, methods
License: GPL-3
Encoding: UTF-8
LazyData: true
```

This code is written to file `DESCRIPTION.R`.

@

## 3.2 Import directives, S4-classes and generics

10a ⟨*Allclasses.R* 10a⟩≡

```
#' @import methods
NULL
#' @import zoo
NULL
#' @importFrom graphics plot
NULL
#' @importFrom stats density sd na.omit start end smooth
NULL
#' @importFrom utils head tail
NULL

# Setting old (aka S3) classes
setOldClass("zoo")


⟨man-class-HikeR 23b⟩
setClass("HikeR", slots = list(ys = "zoo",
                               k = "integer",
                               scoreby = "character",
                               yname = "character"))

⟨man-class-PTBB 24a⟩
setClass("PTBB", slots = list(pt = "zoo",
                              type = "character",
                              h = "numeric"))
```

This code is written to file `Allclasses.R`.

@

10b ⟨*Allgenerics.R* 10b⟩≡

```
# generic for extracting peaks
setGeneric("peaks", function(object, ...) standardGeneric("peaks"))
# generic for extracting troughs
setGeneric("troughs", function(object, ...) standardGeneric("troughs"))
# generic for extracting bursts
setGeneric("bursts", function(object, ...) standardGeneric("bursts"))
# generic for extracting busts
setGeneric("busts", function(object, ...) standardGeneric("busts"))
# generic for computing ridges
setGeneric("ridges", function(object, ...) standardGeneric("ridges"))
# generic for computing phases
setGeneric("phases", function(object, ...) standardGeneric("phases"))
# generic for extracting topeaks
setGeneric("topeaks", function(object, ...) standardGeneric("topeaks"))
# generic for extracting totroughs
setGeneric("totroughs", function(object, ...) standardGeneric("totroughs"))
# generic for computing runs
setGeneric("runs", function(x) standardGeneric("runs"))
```

This code is written to file `Allgenerics.R`.

@

## 3.3 Methods for S4-class 'HikeR'

In this section the S4-methods for objects of class `HikeR` are discussed. The provided methods are for showing `show()`, summarizing `summary()`, retrieval of peaks `peaks()` and troughs `troughs()` for this type of objects. Furthermore, the concept of bursts phases (close occurrence of peaks with respect to time) and busts (close occurrence of troughs with respect to time) are defined as methods `bursts()` and `busts()`, respectively. Additional methods for characterising the progression of a time series, such as 'ridges', 'phases', 'to-peaks' and 'to-troughs' are provided, too. Finally, a `plot()`-method is available whereby the user can hightlight/shade the local optima and the phases in between them. All of these methods are contained in the file `hiker-methods.R`. The skeleton of this file is provided next.

11a ⟨*HikerMethods.R* 11a⟩≡
⟨*HikeR-show* 11b⟩
⟨*HikeR-summary* 12a⟩
⟨*HikeR-peaks* 12b⟩
⟨*HikeR-troughs* 12c⟩
⟨*HikeR-bursts* 13a⟩
⟨*HikeR-busts* 13b⟩
⟨*HikeR-ridges* 14a⟩
⟨*HikeR-phases* 14b⟩
⟨*HikeR-topeaks* 15⟩
⟨*HikeR-totroughs* 16⟩
⟨*HikeR-plot* 17⟩

This code is written to file `HikerMethods.R`.

@

### 3.3.1 `show`-method

11b ⟨*HikeR-show* 11b⟩≡
⟨*man-HikeR-show* 24b⟩

```
setMethod("show",
          signature(object = "HikeR"), function(object){
              cat(paste("Peak/trough score computed as: ",
                        object@scoreby, ".\n", sep = ""))
              cat(paste("Count of left/right neighbours: ", object@k,
                        ".\n", sep = ""))
              cat("\nSummary statistics of scores:\n")
              print(summary(object))
          }
)
```

@

### 3.3.2 `summary`-method

12a ⟨*HikeR-summary* 12a⟩≡

```
⟨man-HikeR-summary 24c⟩
setMethod("summary",
          signature(object = "HikeR"),
          function (object, ...){
              summary(na.omit(coredata(object@ys[, 2])))
          }
)
```

@

### 3.3.3 `peaks`-method

12b ⟨*HikeR-peaks* 12b⟩≡

```
⟨man-HikeR-peaks 24d⟩
setMethod("peaks",
    signature(object = "HikeR"),
    function (object, h = 0) {
        ans <- object@ys[, 2] > h
        new("PTBB", pt = ans, type = "peak", h = h)
    }
)
```

@

### 3.3.4 `troughs`-method

12c ⟨*HikeR-troughs* 12c⟩≡

```
⟨man-HikeR-troughs 24e⟩
setMethod("troughs",
    signature(object = "HikeR"),
    function (object, h = 0) {
        ans <- object@ys[, 2] < -h
        new("PTBB", pt = ans, type = "trough", h = h)
    }
)
```

@

### 3.3.5 bursts-method

13a ⟨*HikeR-bursts* 13a⟩≡
```
⟨man-HikeR-bursts 24f⟩
setMethod("bursts",
    signature(object = "HikeR"),
    function (object, h = 0, b = object@k) {
        lpts <- object@ys[, 2] > h
        lidx <- which(lpts == TRUE)
        nidx <- length(lidx)
        if ( nidx > 1 ){
            for ( i in 2:nidx ){
                didx <- lidx[i] - lidx[i - 1]
                if ( didx <= b ){
                    lpts[(lidx[i]):(lidx[i - 1])] <- TRUE
                }
            }
        }
        ans <- zoo(lpts, order.by = index(object@ys))
        new("PTBB", pt = ans, type = "burst", h = h)
})
```
@

### 3.3.6 busts-method

13b ⟨*HikeR-busts* 13b⟩≡
```
⟨man-HikeR-busts 25a⟩
setMethod("busts",
    signature(object = "HikeR"),
    function (object, h = 0, b = object@k) {
        lpts <- object@ys[, 2] < h
        lidx <- which(lpts == TRUE)
        nidx <- length(lidx)
        if ( nidx > 1 ){
            for ( i in 2:nidx ){
                didx <- lidx[i] - lidx[i - 1]
                if ( didx <= b ){
                    lpts[(lidx[i]):(lidx[i - 1])] <- TRUE
                }
            }
        }
        ans <- zoo(lpts, order.by = index(object@ys))
        new("PTBB", pt = ans, type = "bust", h = h)
})
```

@

### 3.3.7 ridges-method

14a ⟨*HikeR-ridges* 14a⟩≡

```
⟨man-HikeR-ridges 25b⟩
setMethod("ridges",
    signature(object = "HikeR"),
    function (object, h = 0, b = object@k) {
        N <- nrow(object@ys)
        k <- object@k
        bustp <- busts(object, h = h, b = b)@pt
        burstp <- bursts(object, h = h, b = b)@pt
        bbp <- cbind(bustp, burstp)
        ans <- zoo(FALSE, order.by = index(bustp))
        ridx <- which ( (rowSums(bbp) > 1) | (rowSums(bbp) < 1) )
        ans[ridx] <- TRUE
        ans[1:object@k] <- NA
        ans[(N - k + 1):N] <- NA
        new("PTBB", pt = ans, type = "ridge", h = h)
})
```

@

### 3.3.8 phases-method

14b ⟨*HikeR-phases* 14b⟩≡

```
⟨man-HikeR-phases 25c⟩
setMethod("phases",
    signature(object = "HikeR"),
    function (object, h = 0, b = object@k) {
        N <- nrow(object@ys)
        ans <- rep(NA, N)
        bustp <- busts(object, h = h, b = b)@pt
        ans[which(bustp == TRUE)] <- "bust"
        burstp <- bursts(object, h = h, b = b)@pt
        burstp
        ans[which(burstp == TRUE)] <- "burst"
        ridgep <- ridges(object, h = h, b = b)@pt
        ans[which(ridgep == TRUE)] <- "ridge"
        ans <- factor(ans)
        ans <- zoo(ans, order.by = index(object@ys))
        new("PTBB", pt = ans, type = "phase", h = h)
})
```

@

### 3.3.9 topeaks-method

15 ⟨*HikeR-topeaks* 15⟩≡

```
⟨man-HikeR-topeaks 25d⟩
setMethod("topeaks",
    signature(object = "HikeR"),
    function (object, h = 0) {
        N <- nrow(object@ys)
        k <- object@k
        ans <- zoo(rep(TRUE, N), order.by = index(object@ys))
        ans[1:k] <- NA
        ans[(N - k + 1):N] <- NA
        peakp <- peaks(object, h)@pt
        pidx <- which(peakp == TRUE)
        npidx <- length(pidx)
        troup <- troughs(object, h)@pt
        tidx <- which(troup == TRUE)
        ntidx <- length(tidx)
        if ( npidx == 0 ){
            warning("\nNo local peak points.\n")
            return(NULL)
        }
        if ( ntidx == 0 ){
            warning("\nNo local trough points.\n")
            return(NULL)
        }
        ## if trough comes first, set prior points to FALSE
        if ( tidx[1] < pidx[1] ){
            ans[(k + 1):tidx[1]] <- FALSE
        }
        for ( i in 1:ntidx ) {
            previouspeaks <- which(pidx < tidx[i])
            countpreviouspeaks <- length(previouspeaks)
            if ( countpreviouspeaks > 0 ){
                maxpos <- which.max(object@ys[pidx[previouspeaks], 1])
                ans[(pidx[maxpos] + 1):tidx[i]] <- FALSE
                pidx <- pidx[-c(1:countpreviouspeaks)]
            }
        }
        new("PTBB", pt = ans, type = "topeak", h = h)
})
```

@

### 3.3.10 `totroughs`-method

16 ⟨*HikeR-totroughs* 16⟩≡
⟨*man-HikeR-totroughs* 25e⟩

```
setMethod("totroughs",
    signature(object = "HikeR"),
    function (object, h = 0) {
        ans <- topeaks(object, h)
        ans@pt <- !ans@pt
        ans@type <- "tottrough"
        ans
})
```

@

### 3.3.11 `plot`-method

17 ⟨*HikeR-plot* 17⟩≡

```
⟨man-HikeR-plot 25f⟩
setMethod("plot",
    signature(x = "HikeR", y = "missing"),
    function (x, type = c("series", "score", "both", "zoo"),
              h = 0, b = x@k, phase = c("none", "pt", "bb"),
              main = NULL, sub = NULL,
              pt.peak = list(col = "darkgreen", pch = 19, cex = 0.8),
              pt.trough = list(col = "darkred", pch = 19, cex = 0.8),
              area.se = list(col = "lightgray"),
              area.pb = list(col = "seagreen", density = 20),
              area.tb = list(col = "red2", density = 20),
              ...){
        type <- match.arg(type)
        phase <- match.arg(phase)
        N <- nrow(x@ys)
        xidx <- 1:N
        if ( is.null(sub) ){
            sub <- paste("Score method: ", x@scoreby,
                         ", k = ", x@k, sep = "")
        }
        if ( type == "score" ){
            srange <- range(na.omit(x@ys[, 2]))
            if ( is.null(main) ){
                main <- paste("Scores of time series:", x@yname)
            }
            plot(c(1, N), srange, axes = FALSE,
                 main = main,
                 sub = sub,
                 xlab = "Time", ylab = "Score", type = "n")
            do.call(graphics::rect, c(list(xleft = 1,
                                           ybottom = srange[1],
                                           xright = x@k,
                                           ytop = srange[2]),
                                      area.se))
            do.call(graphics::rect, c(list(xleft = N - x@k + 1,
                                           ybottom = srange[1],
                                           xright = N,
                                           ytop = srange[2]),
                                      area.se))
            graphics::lines(coredata(x@ys[, 2]), type = "h", ...)
            graphics::abline(h = 0)
            graphics::abline(h = c(h, -h), col = "red")
            graphics::box()
            graphics::axis(1, at = xidx, labels = index(x@ys), tick = FALSE)
            idx <- pretty(srange)
```

```
    graphics::axis(2, at = idx, labels = idx)
}
if ( type == "series" ){
    yrange <- range(na.omit(x@ys[, 1]))
    if ( is.null(main) ){
        if ( phase == "none" ){
            main <- paste("Peaks and Troughs of:", x@yname)
        } else if ( phase == "pt" ){
            main <- paste("To-Peak and To-Trough Phases of:", x@yname)
        }
        else if ( phase == "bb") {
            main <- paste("Burst and Bust Phases of:", x@yname)
        }
    }
    plot(c(1, N), yrange, axes = FALSE,
         main = main, sub = sub,
         xlab = "Time", ylab = "", type = "n")
    do.call(graphics::rect, c(list(xleft = 1,
                                   ybottom = yrange[1],
                                   xright = x@k,
                                   ytop = yrange[2]),
                              area.se))
    do.call(graphics::rect, c(list(xleft = N - x@k + 1,
                                   ybottom = yrange[1],
                                   xright = N,
                                   ytop = yrange[2]),
                              area.se))
    graphics::lines(coredata(x@ys[, 1]), ...)
    peakidx <- which(peaks(x, h = h)@pt == TRUE)
    ynum <- coredata(x@ys[index(x@ys)[peakidx], 1])
    do.call(graphics::points, c(list(y = ynum, x = peakidx),
                                pt.peak))
    troughidx <- which(troughs(x, h = h)@pt == TRUE)
    ynum <- coredata(x@ys[index(x@ys)[troughidx], 1])
    do.call(graphics::points, c(list(y = ynum, x = troughidx),
                                pt.trough))
    if ( phase  == "bb" ){
        p <- phases(x, h = h, b = b)
        pchar <- as.character(coredata(p@pt))
        burstz <- new("PTBB",
                      pt = zoo(pchar == "burst",
                               order.by = index(p@pt)),
                      type = "burst",
                      h = p@h)
        burstr <- runs(burstz)
        if ( !is.null(burstr) ) {
            xleft <- which(as.character(index(burstz@pt)) %in%
                           as.character(burstr[, "From"]))
            xright <- which(as.character(index(burstz@pt)) %in%
                            as.character(burstr[, "To"]))
```

```
            for (i in 1:length(xleft)) {
                do.call(graphics::rect, c(list(xleft = xleft[i],
                                               ybottom = yrange[1],
                                               xright = xright[i],
                                               ytop = yrange[2]),
                                          area.pb))
            }
        }
        bustz <- new("PTBB",
                     pt = zoo(pchar == "bust",
                              order.by = index(p@pt)),
                      type = "bust",
                     h = p@h)
        bustr <- runs(bustz)
        if ( !is.null(bustr) ) {
            xleft <- which(as.character(index(bustz@pt)) %in%
                           as.character(bustr[, "From"]))
            xright <- which(as.character(index(bustz@pt)) %in%
                            as.character(bustr[, "To"]))
            for (i in 1:length(xleft)) {
                do.call(graphics::rect, c(list(xleft = xleft[i],
                                               ybottom = yrange[1],
                                               xright = xright[i],
                                               ytop = yrange[2]),
                                          area.tb))
            }
        }
    }
    if ( phase == "pt" ){
        top <- topeaks(x, h)
        topr <- runs(top)
        tot <- totroughs(x, h)
        totr <- runs(tot)
        if ( !is.null(topr) ) {
            xleft <- which(as.character(index(top@pt)) %in%
                           as.character(topr[, "From"]))
            xright <- which(as.character(index(top@pt)) %in%
                            as.character(topr[, "To"]))
            for (i in 1:length(xleft)) {
                do.call(graphics::rect, c(list(xleft = xleft[i],
                                               ybottom = yrange[1],
                                               xright = xright[i],
                                               ytop = yrange[2]),
                                          area.pb))
            }
        }
        if ( !is.null(totr) ) {
            xleft <- which(as.character(index(tot@pt)) %in%
                           as.character(totr[, "From"]))
            xright <- which(as.character(index(tot@pt)) %in%
```

```
                                    as.character(totr[, "To"]))
                    for (i in 1:length(xleft)) {
                        do.call(graphics::rect, c(list(xleft = xleft[i],
                                                       ybottom = yrange[1],
                                                       xright = xright[i],
                                                       ytop = yrange[2]),
                                                  area.tb))
                    }
                }
            }
            graphics::box()
            graphics::axis(1, at = xidx, labels = index(x@ys),
                           tick = FALSE)
            idx <- pretty(yrange)
            graphics::axis(2, at = idx, labels = idx)
        }
        if ( type == "both" ){
            op <- graphics::par(no.readonly = TRUE)
            graphics::par(mfrow = c(2, 1))
            plot(x, type = "series", h = h, ...)
            plot(x, type = "score", h = h, ...)
            graphics::par(op)
        }
        if ( type == "zoo" ){
            plot(x@ys, ...)
        }
    }
)
```

@

### 3.4 runs-method for S4-class 'PTBB'

21 ⟨*PtbbMethods.R* 21⟩≡

```
⟨man-PTBB-runs 26a⟩
setMethod("runs",
          signature(x = "PTBB"), function(x){
              if ( any(na.omit(x@pt)) ){
                  p <- as.numeric(coredata(x@pt))
                  run <- rle(p)
                  peakruns <- which(run$values == 1)
                  cumidx <- cumsum(run$lengths)
                  n <- length(cumidx)
                  xidxright <- cumidx
                  xidxleft <- c(1, cumidx + 1)[-c(n + 1)]
                  xidx <- cbind(xidxleft, xidxright)
                  peakidx <- xidx[peakruns, ]
                  runscount <- nrow(xidx)
                  runsidx <- 1:runscount
                  ans <- sapply(runsidx, function(i)
                      x@pt[xidx[i, 1]:xidx[i, 2]])
                  pidx <- which(unlist(lapply(ans, function(r)
                      is.element(TRUE, r[1]))))
                  anstrue <- ans[pidx]
                  per <- lapply(anstrue, function(i)
                      data.frame(start(i), end(i)))
                  ans <- data.frame(do.call("rbind", per))
                  ans[, "Type"] <- x@type
                  colnames(ans) <- c("From", "To", "Type")
                  return(ans)
              } else {
                  return(NULL)
              }
          }
)
```

This code is written to file `PtbbMethods.R`.

@

# 4 Appendix

## 4.1 Documentation of functions

22 ⟨*man-func-score* 22⟩≡

```
#' Basic scoring methods for local minima and maxima
#'
#' These are basic functions for evaluating the centre
#' point of a time series as local minimum or maximum.
#' Hereby, a score value is computed according to various methods.
#' If the score is positive, the centre point is tentatively
#' classified as a local peak.
#' Incidentally, negative scores indicate a local minima.
#'
#' @param x \code{numeric}, vector of length \code{2 * k + 1}.
#' @param k \code{integer}, the count of left/right neighbours.
#' @param scoreby \code{character}, the scoring method to be used.
#' @param tval \code{numeric}, factor for standard deviation band
#' if \code{scoreby = 'ttype'}.
#' @param confby \code{integer}, count of minimum vote,
#' values in the set \code{3:5}.
#' @param ... ellipsis argument.
#'
#' @name score
#' @family scores
#' @return \code{numeric}, the score value.
NULL

#' @rdname score
#' @export
```

@

23a ⟨*man-func-hiker* 23a⟩≡

```
#' Peak/trough scores of time series points
#'
#' This function computes the score value for each
#' data point of a time series. The first and last
#' \code{k} observations are set to \code{NA}.
#'
#' @inheritParams score
#' @param y \code{zoo}, univariate time series.
#' @return An object of S4-class \code{HikeR}.
#' @family scores
#'
#' @references Girish K. Palshikar. Simple Algorithms for
#' Peak Detection in Time-Series. In \emph{Proc. 1st Int. Conf.
#' Advanced Data Analysis,
#' Business Analytics and Intelligence}, 2009.
#'
#' @examples
#' TEX <- SP500[, "TEX"]
#' ans <- hiker(TEX, k = 8, scoreby = "hybrid", tval = 0.1)
#' ans
#' plot(ans)
#'
#' @export
```

@

## 4.2 Documentation of S4-classes

23b ⟨*man-class-HikeR* 23b⟩≡

```
#' S4 class \code{HikeR}
#'
#' Formal class for classifying local minima and maxima
#' of a time series.
#'
#' @slot ys \code{zoo}, time series with associated scores.
#' @slot k \code{integer}, count of left/right neigbours around centre point.
#' @slot scoreby \code{character}, scoring method.
#' @slot yname \code{character}, name of the series.
#' @exportClass HikeR
```

@

24a ⟨*man-class-PTBB* 24a⟩≡

```
#' S4 class \code{PTBB}
#'
#' Formal class for peaks, troughs, burst, busts and
#' intermittent phase of a time series.
#'
#' @slot pt \code{zoo}, logical: indicating peak/trough points.
#' @slot type \code{character}, type of point/phase.
#' @slot h \code{numeric}, the threshhold for score evaluation.
#' @exportClass PTBB
```

@

## 4.3 Documentation of S4-methods

24b ⟨*man-HikeR-show* 24b⟩≡

```
#' @rdname HikeR-class
#' @param object An object of S4 class \code{HikeR}.
#' @export
```

@

24c ⟨*man-HikeR-summary* 24c⟩≡

```
#' @rdname HikeR-class
#' @aliases summary
#' @param ... Ellipsis argument.
#' @export
```

@

24d ⟨*man-HikeR-peaks* 24d⟩≡

```
#' @rdname HikeR-class
#' @aliases peaks
#' @param h \code{numeric}, the threshold value for scores
#' to be considered as peaks/troughs.
#' @return Object of S4-class \code{PTBB}.
#' @export
```

@

24e ⟨*man-HikeR-troughs* 24e⟩≡

```
#' @rdname HikeR-class
#' @aliases troughs
#' @export
```

@

24f ⟨*man-HikeR-bursts* 24f⟩≡

```
#' @rdname HikeR-class
#' @aliases bursts
#' @param b \code{integer}, intermittent count of points between peaks.
#' @export
```

@

25a ⟨*man-HikeR-busts* 25a⟩≡

```
#' @rdname HikeR-class
#' @aliases busts
#' @export
```

@

25b ⟨*man-HikeR-ridges* 25b⟩≡

```
#' @rdname HikeR-class
#' @aliases ridges
#' @export
```

@

25c ⟨*man-HikeR-phases* 25c⟩≡

```
#' @rdname HikeR-class
#' @aliases phases
#' @export
```

@

25d ⟨*man-HikeR-topeaks* 25d⟩≡

```
#' @rdname HikeR-class
#' @aliases topeaks
#' @export
```

@

25e ⟨*man-HikeR-totroughs* 25e⟩≡

```
#' @rdname HikeR-class
#' @aliases totroughs
#' @export
```

@

25f ⟨*man-HikeR-plot* 25f⟩≡

```
#' @rdname HikeR-class
#' @aliases plot
#' @param x An object of S4 class \code{HikeR}.
#' @param type \code{character}, whether series, scores or both should be plotted.
#' @param phase \code{character}, whether burst/bust or topeak/totrough phases
#' should be drawn in series plot.
#' @param pt.peak \code{list}, named elements are passed to \code{graphics::points()}.
#' @param main \code{character}, main title of plot.
#' @param sub \code{character}, sub title of plot
#' @param pt.trough \code{list}, named elements are passed to \code{graphics::points()}.
#' @param area.se \code{list}, named elements are passed to \code{graphics::rect()}
#' for areas of pre- and post sample points.
#' @param area.pb \code{list}, named elements are passed to \code{graphics::rect()}
#' for 'to-peak' or 'burst' phases.
#' @param area.tb \code{list}, named elements are passed to \code{graphics::rect()}
#' for 'to-trough' or 'bust' phases.
#' @export
```

@

26a ⟨*man-PTBB-runs* 26a⟩≡

```
#' @rdname PTBB-class
#' @aliases runs
#' @param x An object of S4 class \code{PTBB}.
#' @export
```

@

## 4.4 Documentation of data set

26b ⟨*data.R* 26b⟩≡

```
#' Weekly price data of 476 S&P 500 constituents.
#'
#' The data set was used in the reference below. The authors adjusted
#' the price data for dividends and have removed stocks if two or
#' more consecutive missing values were found. In the remaining cases
#' the NA entries have been replaced by interpolated values.
#'
#'
#' @format A S3-class \code{zoo} object with 265 weekly observations
#' of 476 members of the S&P 500 index. The sample starts at 2003-03-03
#' and ends in 2008-03-24.
#'
#' @references Cesarone, F. and Scozzari, A. and Tardella, F.: Portfolio
#'     selection problems in practice: a comparison between linear and
#'     quadratic optimization models, Working Paper, Universita degli
#'     Studi Roma Tre, Universita Telematica delle Scienze Umane and
#'     Universita di Roma, July 2010.
#'     \url{http://arxiv.org/ftp/arxiv/papers/1105/1105.3594.pdf}
#'
#' @source \url{http://w3.uniroma1.it/Tardella/datasets.html},\cr
#' \url{ http://finance.yahoo.com/}
"SP500"
```

This code is written to file `data.R`.

@

## 4.5 Makefile

# 5 Chunk Index

# 6 Identifier Index

# References

Palshikar, G. (2009). Simple algorithms for peak detection in time-series. In *First Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence*, Ahmedabad, India.