R documentation

of 'man/breakTrends.Rd' etc.

January 26, 2017

---

breakTrends *select where to break the trends and plot usefull graphs*

Usage

```
breakTrends()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    attach(yieldPrev)
    flatPlot <- ggplot(yieldPrev$flatYield) + geom_point(color = "black",
        aes(x = YEAR, y = OFFICIAL_YIELD)) + geom_line(color = "black",
        aes(x = YEAR, y = OFFICIAL_YIELD)) + geom_smooth(method = "lm",
        color = "brown", se = FALSE, aes(x = YEAR, y = OFFICIAL_YIELD)) +
        geom_smooth(method = "loess", color = "red", se = FALSE,
            aes(x = YEAR, y = OFFICIAL_YIELD)) + geom_smooth(method = "lm",
        formula = y ~ splines::bs(x, 3), color = "orange", se = FALSE,
        aes(x = YEAR, y = OFFICIAL_YIELD))
    if (any(names(yieldPrev) == "breakPoint"))
        flatPlot <- flatPlot + geom_rect(data = yieldPrev$breakPoint,
            aes(xmin = begin, xmax = finish, ymin = -Inf, ymax = +Inf),
            fill = "yellow", alpha = 0.3)
    plot(flatPlot)
    cat(c("Time series starts in", unique(min(flatYield$YEAR)),
        "and ends in ", unique(max(flatYield$YEAR))), fill = TRUE)
    cat(c("Point the trend's edges by year\n"), fill = TRUE)
    trendEdge <- scan(, nmax = 2)
    tempLimit <- data.frame(begin = min(trendEdge), finish = max(trendEdge))
    cutPlot <- flatPlot + geom_rect(data = tempLimit, aes(xmin = begin,
        xmax = finish, ymin = -Inf, ymax = +Inf), fill = "pink",
        alpha = 0.5)
    plot(cutPlot)
```

```
  sewTrends(min(trendEdge), max(trendEdge))
  detach(yieldPrev)
}
```

---

checkTrends *Plot usefull graph to check the existance of trends*

Description

It plots and asks for removing trend or no. In virgilio there is no other way to stop removing trends than answer something ese than "y"

Usage

```
checkTrends()
```

Note

If it seems almost useless, note the role it assumes in the default (virgilio()'s one) path of work.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    attach(yieldPrev)
    if (any(names(yieldPrev) == "flatYield")) {
        cat(c("Official yields have a ", (adf.test(flatYield$OFFICIAL_YIELD)$p.value) *
            100, "% of chances to have a trend."), fill = TRUE)
    }
    else {
       cat(c("Official yields have a ", (adf.test(actualYield$OFFICIAL_YIELD)$p.value) *
            100, "% of chances to have a trend."), fill = TRUE)
    }
   cat(c("Look the chart for the visual assessment. \n Plotted: \n BLACK:actual data \n BROWN:linear model \n
        fill = TRUE)
    if (any(names(yieldPrev) == "flatYield")) {
        offiPlot <- ggplot(flatYield, aes(x = YEAR, y = OFFICIAL_YIELD)) +
            geom_point(color = "black") + geom_line(color = "black") +
            geom_smooth(method = "lm", color = "brown", se = FALSE) +
            geom_smooth(method = "loess", color = "red", se = FALSE) +
            geom_smooth(method = "lm", formula = y ~ splines::bs(x,
                3), color = "orange", se = FALSE)
    }
    else {
        offiPlot <- ggplot(actualYield, aes(x = YEAR, y = OFFICIAL_YIELD)) +
            geom_point(color = "black") + geom_line(color = "black") +
            geom_smooth(method = "lm", color = "brown", se = FALSE) +
            geom_smooth(method = "loess", color = "red", se = FALSE) +
            geom_smooth(method = "lm", formula = y ~ splines::bs(x,
```

```
            3), color = "orange", se = FALSE)
    }
    plot(offiPlot)
    cat(c("Do you see a trend in the data?\n\ty \n remove trend \n \n\tn \n "),
        fill = TRUE)
    yieldPrev$flattyn <- scan(, what = "text", nmax = 1)
    if (any(names(yieldPrev) == "flatYield")) {
    }
    else {
        flatYield <- actualYield
        yieldPrev$flatYield <- actualYield
    }
    if (length(c(setdiff(seq(min(flatYield$YEAR), max(flatYield$YEAR)),
        flatYield$YEAR), flatYield$YEAR[duplicated(flatYield$YEAR)])) ==
        0)
        cat("")
    else cat(c("By the way there are \n MISSING:", setdiff(seq(min(flatYield$YEAR),
        max(flatYield$YEAR)), flatYield$YEAR), " \n REPLICATED: ",
        flatYield$YEAR[duplicated(flatYield$YEAR)]))
    detach(yieldPrev)
  }
```

---

configure *import all the data needed*

Usage

```
configure()
#never tested:
configure(depth="advanced")
```

Arguments

depth

Details

In normal mode, you ha only to follow the screen instruction. If you are trying the advanced version, it is supposed to allow importing dataset different than the ones used since now. Not yet tested!

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (depth = "base")
{
    cat("If you created your csv databases using MS Office, write <1> here, else <0>",
        fill = TRUE)
    msoffice <- scan(, nmax = 1)
    cat("Provide OFFICIAL yield database", fill = TRUE)
    if (msoffice == 1)
```

```
    eurostat <- read.table(file.choose(), header = T, sep = ";")
else eurostat <- read.csv(file.choose())
if (depth == "advanced" && depth == "adv-offi") {
    freakoffi <- eurostat
   cat(c("Here are the columns in your file. Which one represents the crop code? \n ",
        names(freakoffi), " \n Write it here: \n "), fill = TRUE)
    scn <- scan(, what = "text", nmax = 1)
  cat(c("Here are the columns in your file. Which one represents the Nation (is going to be drop)? \n ",
        names(freakoffi), " \n Write it here: \n "), fill = TRUE)
    nut <- scan(, what = "text", nmax = 1)
    cat(c("Here are the columns in your file. Which one represents the Years? \n ",
        names(freakoffi), " \n Write it here: \n "), fill = TRUE)
    epoch <- scan(, what = "text", nmax = 1)
  cat(c("Here are the columns in your file. Which one represents the Official Yield? \n ",
        names(freakoffi), " \n Write it here: \n "), fill = TRUE)
    harv <- scan(, what = "text", nmax = 1)
    eurostat$STAT_CROP_NO <- freakoffi$as.name(scn)
    eurostat$NUTS_CODE <- freakoffi$as.name(nut)
    eurostat$YEAR <- freakoffi$as.name(epoch)
    eurostat$OFFICIAL_YIELD <- freakoffi$as.name(harv)
}
cat("Provide SIMULATE yield database", fill = TRUE)
if (msoffice == 1)
    prev <- read.table(file.choose(), header = T, sep = ";")
else prev <- read.csv(file.choose())
if (depth == "advanced" && depth == "adv-simul") {
    freakprev <- prev
   cat(c("Here are the columns in your file. Which one represents the crop code? \n ",
        names(freakprev), " \n Write it here: \n "), fill = TRUE)
    scn <- scan(, what = "text", nmax = 1)
  cat(c("Here are the columns in your file. Which one represents the Nation (is going to be drop)? \n ",
        names(freakprev), " \n Write it here: \n "), fill = TRUE)
    nut <- scan(, what = "text", nmax = 1)
    cat(c("Here are the columns in your file. Which one represents the Years? \n ",
        names(freakprev), " \n Write it here: \n "), fill = TRUE)
    epoch <- scan(, what = "text", nmax = 1)
  cat(c("Here are the columns in your file. Which one represents the decade (phase of the year, if you lik
        names(freakprev), " \n Write it here: \n "), fill = TRUE)
    decad <- scan(, what = "text", nmax = 1)
    prev$CROP_NO <- freakprev$as.name(scn)
    prev$NUTS_CODE <- freakpre$as.name(nut)
    prev$YEAR <- freakoffi$pre(epoch)
    prev$DECADE <- freakoffi$as.name(decad)
}
cat(c("OFFICIAL yield data contains information for the following CROPs:\n",
    unique(eurostat$STAT_CROP_NO), "\n Choose one:", fill = TRUE))
cropO <- scan(, nmax = 1)
cat(c("SIMULATED yield data contains information for the following CROPs:\n",
    unique(prev$CROP_NO), "\n Choose one:", fill = TRUE))
cropS <- scan(, nmax = 1)
cat(c("Following countries are provided by the DataBases:\n",
    "OFFICIAL:", levels(eurostat$NUTS_CODE), "\n SIMULATION:",
    levels(prev$NUTS_CODE), "(case sensitive\n"), fill = TRUE)
cat(" OFFICIAL COUNTRY:")
countryO <- scan(, what = "text", nmax = 1)
cat("\n SIMULATION COUNTRY")
countryS <- scan(, what = "text", nmax = 1)
```

```
  actualYield <- subset(eurostat, eurostat$STAT_CROP_NO ==
      cropO & eurostat$NUTS_CODE == countryO)[, c(which(names(eurostat) ==
      "YEAR"), which(names(eurostat) == "OFFICIAL_YIELD"))]
  yieldPrev$actualYield <- actualYield[order(actualYield$YEAR),
      ]
  relatedModel <- subset(prev, prev$CROP_NO == cropS & prev$NUTS_CODE ==
      countryS)
  currentYear <- max(unique(relatedModel$YEAR))
  yieldPrev$currentYear <- currentYear
  currentDecade <- max(subset(relatedModel, relatedModel$YEAR ==
      currentYear)$DECADE)
  cat(c("It seems forecasting the year", currentYear, "with data till the ",
      currentDecade, "th decade"), fill = TRUE)
  yieldPrev$relatedModel <- subset(relatedModel, relatedModel$DECADE ==
      currentDecade)[, c(-which(names(prev) == "CROP_NO"),
      -which(names(prev) == "DECADE"), -which(names(prev) ==
          "NUTS_CODE"))]
  yieldPrev$saveCrop <- cropS
  yieldPrev$saveCountry <- countryS
}
```

`cutTrend` *Actually do the cutting of trends, not simply setting it*

Description

Two input parameters: It receives the edges to cut at.

Usage

```
#called by sewTrends()
cutTrend(inizio, fine)
```

Arguments

`inizio`	is the lower year (included) of the trended lapse
`fine`	is the upper year (included) of the trended lapse

Note

The lower year included in the trend comes out as it get in. Considering that the trand is, at that time, experiencing the year of its start no changes actually occur.

Mathematically, the trend is accounted to be linear, and so described by the following:

$yield_n~unTrended= yield_n~Offical - (year_n - year_startTrend) * Trend_coefficient $

So that is clear that for n = startTrend (aka "inizio") it is that the trend and the unTrended versions of yield are the same.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (inizio, fine)
{
    notSoFlat <- yieldPrev$flatYield
    preflat <- subset(notSoFlat, notSoFlat$YEAR >= inizio & notSoFlat$YEAR <=
        fine)
    flatLin <- lm(OFFICIAL_YIELD ~ YEAR, data = preflat)
    cutEnv <- new.env()
    cutEnv$modello <- flatLin
    cutEnv$flatting <- preflat
    if (any(names(yieldPrev) == "safeTrend)")) {
        cutEnv$trendCorr <- (flatLin$coefficients[2] - yieldPrev$safeTrend)
    }
    else {
        cutEnv$trendCorr <- flatLin$coefficients[2]
    }
    smootherer <- function(num) {
        model <- cutEnv$modello
        flat <- (model$model[num, 1]) - (model$model[num, 2] -
            model$model[1, 2]) * cutEnv$trendCorr
        cutEnv$flatting[num, 2] <- flat
    }
    lapply(X = seq(1, length(preflat$OFFICIAL_YIELD)), FUN = smootherer)
    preflat <- cutEnv$flatting
    if (any(names(yieldPrev) == "breakPoint"))
        yieldPrev$breakPoint <- rbind(yieldPrev$breakPoint, c(inizio,
            fine, as.numeric(flatLin$coefficients[2])))
    else yieldPrev$breakPoint <- data.frame(begin = inizio, finish = fine,
        trend = as.numeric(flatLin$coefficients[2]))
    postFlat <- subset(notSoFlat, notSoFlat$YEAR < inizio | notSoFlat$YEAR >
        fine)
    flatFlat <- rbind(preflat, postFlat)
    yieldPrev$flatYield <- flatFlat[order(flatFlat$YEAR), ]
  }
```

`foreYield-package` *foreYield forecasting yield result*

Description

Using past yield and model predictors, forecast the yield resulting for the current (the last) year (simulated year). Nothing new but the interface "virgilio()" providing easy use without any (almost) knowledge of R

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.
The packages is designed to easily apply at forecasting yield. It only requires a couple of data as input: previous yields (Officials) and predictors coming from SIMULATIONs. By default (using configure(), which is the same virgilio() does) those are assumed to come from .csv files but you can provide them by yourself having care them to be homologous to the supposed data.frame structure. If not screening the sources, yuo can check the required structure in configure().

Author(s)

Fosco Vesely

Maintainer: Fosco Vesely <fosco@tana.it>

References

~~ Literature or other references for background information ~~

Examples

```
#This should allow you to just follow the instructions on screen
virgilio()
```

`loadYieldSession` *simply restore a past session (saved)*

Usage

```
loadYieldSession()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    cat(c("Select the desired previous session saved \n "), fill = TRUE)
    oldYieldSession <- file.choose()
    load(file = oldYieldSession, envir = yieldPrev)
    cat(c("Session in oldYieldSession loaded \n "), fill = TRUE)
  }
```

modSel *Interactive choice of the model you like more*

Description

the main statistical stuff is here

Usage

```
modSel()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    tableXregression <- merge(yieldPrev$flatYield, yieldPrev$relatedModel,
        by = "YEAR")
    allSign <- regsubsets(OFFICIAL_YIELD ~ .^2 + ., data = tableXregression[,
        c(-1)], nbest = 2, method = "exhaustive", nvmax = 4,
        really.big = TRUE)
    summaSign <- summaryHH(allSign, names = seq(1, length(allSign$xnames)),
        statistics = "adjr2")
    plot(summaSign, col = "green", cex = 0.8)
    print(summaSign)
    cat("Note: one of the accounted parameter is (Intercept) \n \n Select a model")
    modId <- scan(, nmax = 1)
    yieldPrev$model_formula <- c("OFFICIAL_YIELD ~ ")
    compleFormula <- function(parametro) {
        yieldPrev$model_formula <- paste(yieldPrev$model_formula,
            names(coef(allSign, id = modId))[parametro], if (parametro ==
                length(names(coef(allSign, id = modId))))
                sep = " "
            else sep = " +")
    }
    lapply(X = seq(2, length(names(coef(allSign, id = modId)))),
        FUN = compleFormula)
    regrSW <- lm(as.formula(yieldPrev$model_formula), data = tableXregression)
    print(regrSW)
    relatedModel <- yieldPrev$relatedModel
    expYield <- predict(regrSW, newdata = subset(relatedModel,
        relatedModel$YEAR == yieldPrev$currentYear), se.fit = TRUE,
        type = "response", level = 0.95, interval = "prediction")
    yieldPrev$expYield <- expYield
    yieldPrev$modelLM <- regrSW
    yieldPrev$tableXregression <- tableXregression
    attach(yieldPrev)
    validC <- CV(yieldPrev$modelLM)
    detach(yieldPrev)
    yieldPrev$CVmsRes <- c(validC[1], validC[5])
  }
```

responseYield *Some adjustment and mainly prints the results*

Usage

```
responseYield()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    expYield <- yieldPrev$expYield
    knoTime <- yieldPrev$breakPoint
    trendMissing <- (knoTime$finish - knoTime$begin) * knoTime$trend
    cat(c(" \n \n \n RESPONSE \n \n ", "As it is, the forecasted yield for year",
        yieldPrev$currentYear, "is", round(expYield$fit[1], 2),
        "+/-", round(expYield$fit[1] - expYield$fit[2], 2), "."),
        fill = TRUE)
    cat(c("Confidence = 95% \n\n\t\n \n CROSS-VALIDATION returned ",
      round(yieldPrev$CVmsRes[1], 2), "as mean square error\n\n \n\nDue to the marked trends, the forecasted
        round(trendMissing, 2), " resulting in ", round(expYield$fit[1] +
          trendMissing, 2), ". \n\n\n\t\n \n\n\tTimeSeries statistical analysis over OFFICIAL_YIELD would bet
        round(forecast(ets(yieldPrev$actualYield[, 2]), h = 1)$mean[1],
            2), " +/- ", round((forecast(ets(yieldPrev$actualYield[,
            2]), h = 1)$upper[2] - forecast(ets(yieldPrev$actualYield[,
            2]), h = 1)$mean[1]), 2)), fill = TRUE)
  }
```

saveYieldSession *Simply saves yieldPrev environment where you like*

Usage

```
saveYieldSession()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    attach(yieldPrev)
    save(list = ls(yieldPrev), file = paste(Sys.Date(), saveCountry,
        saveCrop, ".R", sep = ""))
```

```
    cat(c("Session saved in ", paste(Sys.Date(), saveCountry,
      saveCrop, ".R", sep = ""), ". \n Use loadYieldSession() to restore in future sessions."),
        fill = TRUE)
  }
```

---

sewTrends *Tuning systemic trends*

Description

If yield experience a trend the models don't grasp you have to remove it to get the right lm. But some part of it may, anyway get explained by the models... Here is the sewing it provides graphs to understand if at least part of the trend you find with checkTrends() is external to simulated parameters. Prompts interactively for the choice of the best parameters describing the trend in the time lapse you selected (from inizio to fine). It also plot a graph with the parameters and yields "normalized" so that matching the trends is easier. Normalization happens around the mean each parameter given in percentage.

Usage

```
#breakTrends() call this
sewTrends(inizio, fine)
```

Arguments

inizio	is the lower year (included) of the trended lapse
fine	is the upper year (included) of the trended lapse

Details

checkTrends() call include both "inizio" and "fine",

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (inizio, fine)
{
    tempLimit <- data.frame(begin = inizio, finish = fine)
    attach(yieldPrev)
    flatOff <- merge(flatYield, relatedModel, by = "YEAR")
    yieldPrev$flatOff <- flatOff
    normalizingTrend <- function(campo) {
        yieldPrev$flatOff[, campo] <- (flatOff[, campo]/mean(flatOff[,
            campo])) * 100
    }
    lapply(X = seq(2, length(flatOff)), FUN = normalizingTrend)
    flatOff <- yieldPrev$flatOff
   yieldPrev$flatOff <- flatOff[sapply(flatOff, function(flatOff) !any(is.na(flatOff)))]
```

```
flatOff <- yieldPrev$flatOff
trendAN <- new.env()
trendAN$normPlot <- ggplot(yieldPrev$flatOff) + geom_line(aes(x = YEAR,
    y = OFFICIAL_YIELD), color = "red") + geom_smooth(method = "loess",
    color = "red", aes(x = YEAR, y = OFFICIAL_YIELD), se = FALSE)
trendPlot <- function(yvar) {
    trendAN$normPlot <- trendAN$normPlot + geom_smooth(data = yieldPrev$flatOff,
        method = "loess", color = "cyan", aes_(x = ~YEAR,
            y = as.name(yvar), label = yvar), se = FALSE)
}
lapply(names(yieldPrev$flatOff[, c(-1)]), FUN = trendPlot)
trendAN$PlotNormA <- trendAN$normPlot + geom_line(aes(x = YEAR,
    y = OFFICIAL_YIELD), color = "red", size = 1.5) + geom_smooth(method = "loess",
    color = "orange", aes(x = YEAR, y = OFFICIAL_YIELD),
    se = FALSE, size = 1.5) + labs(x = "YEARS", y = "TREND") +
    geom_rect(data = tempLimit, aes(xmin = begin, xmax = finish,
        ymin = -Inf, ymax = +Inf), fill = "pink", alpha = 0.5)
detach(yieldPrev)
plot(trendAN$PlotNormA)
flatOff1 <- yieldPrev$flatOff
flatOff2 <- subset(flatOff1, flatOff1$YEAR >= inizio & flatOff1$YEAR <=
    fine)
yieldPrev$flatOff <- flatOff2
flatOff <- flatOff2
yieldPrev$friendShip <- data.frame(param = as.character(names(flatOff)[(names(flatOff) ==
    "OFFICIAL_YIELD")]), trendCoef = as.numeric(lm(formula = OFFICIAL_YIELD ~
    YEAR, data = flatOff)$coefficients[2]))
plot(trendAN$PlotNormA + stat_smooth(data = flatOff, method = "lm",
    color = "black", aes(x = YEAR, y = OFFICIAL_YIELD), fullrange = FALSE,
    se = FALSE, size = 1))
mayTrend <- names(flatOff)[(names(flatOff) != "YEAR" & names(flatOff) !=
    "OFFICIAL_YIELD")]
friendTest <- function(mate) {
    allIn <- yieldPrev$flatOff
    formulFriend <- as.formula(paste(as.name(mate), " ~ YEAR",
        sep = ""))
    linMod <- lm(formula = formulFriend, data = allIn)
  yieldPrev$friendShip <- rbind(yieldPrev$friendShip, data.frame(param = paste(c(as.character(mate)),
        sep = ""), trendCoef = as.numeric(linMod$coefficients[2])),
        deparse.level = 1)
}
sapply(X = mayTrend, FUN = friendTest, USE.NAMES = TRUE)
trendShip <- yieldPrev$friendShip
YieldTrend <- trendShip[1, 2]
yieldPrev$yieldTrend <- YieldTrend
cat(c("The found trend for Official_Yield is ", round((trendShip[1,
  2]), 2), "%. \n The following are the similar trends available between the predictors: \n ->Absolute va
    fill = TRUE)
trendShip$trendDiff <- abs(trendShip[, 2] - trendShip[1,
    2])
trendMates <- trendShip[c(-1), ]
trendMates <- (trendMates[order(trendMates$trendDiff), ])
trendMates$diff_perC <- round(trendMates$trendDiff/YieldTrend,
    2)
trendMates$ID <- seq(1, length(trendMates[, 1]))
print(trendMates)
cat(c("Do you want to continue removing the trend in official yields"),
```

```
        fill = TRUE)
    continueToCut <- scan(, what = "text", nmax = 1)
    if (continueToCut == "y") {
      cat(c("Does any of the predictors explain some of the Official's Trend? \n If yes, point which one(s) by
          fill = TRUE)
      mateList <- scan(, nmax = length(trendMates[, 1]))
      yieldPrev$safeTrend <- mean(trendMates$trendCoef[mateList])
      cutTrend(inizio, fine)
    }
  }
```

`virgilio` *guide you all along*

Description

It just provides a supervised path across the story. If you want to use other function step by step and not this one, you have to grant yieldPrev environment

Usage

```
virgilio()
```

Details

Actually it doesn't do anything, but sort the other function (it doesn't save nor load) from earlier to later stages.

Note

No argument are supposed when calling the function

See Also

configure()

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    if (any(ls() == "yieldPrev")) {
    }
    else {
        yieldPrev <- new.env()
        yieldPrev$.conflicts.OK <- c()
    }
    configure()
    checkTrends()
    while (yieldPrev$flattyn == "y") {
```

```
            breakTrends()
            checkTrends()
        }
        modSel()
        responseYield()
    }
```

Index