Package ‘Markovchart’

January 20, 2025

Type Package

Title Markov Chain-Based Cost-Optimal Control Charts

Version 2.1.5

Author Balazs Dobi & Andras Zempleni

Maintainer Balazs Dobi <dobibalazs@inf.elte.hu>

Description Functions for cost-optimal control charts with a focus on health care applications. Compared to assumptions in traditional control chart theory, here, we allow random shift sizes, random repair and random sampling times. The package focuses on X-bar charts with a sample size of 1 (representing the monitoring of a single patient at a time). The methods are described in Zempleni et al. (2004) <doi:10.1002/asmb.521>, Dobi and Zempleni (2019) <doi:10.1002/qre.2518> and Dobi and Zempleni (2019) <http://ac.inf.elte.hu/Vol_049_2019/129_49.pdf>.

License GPL

Encoding UTF-8

VignetteBuilder knitr

Suggests knitr, kableExtra, gridExtra, reshape2, zoo

Imports stats, parallel, doParallel, optimParallel, foreach, ggplot2, metR

RoxygenNote 7.1.1

NeedsCompilation no

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2022-04-09 23:22:36 UTC

Contents

`diabetes` *Pseudonymised and randomised time series dataset of diabetes patients for control chart applications*

Description

Pseudonymised and randomised time series data of 800 patients. The patients are divided into two main groups by therapy type: insulin analogues (artificial insulins) and glucagon-like peptides (GLP, promotes insulin secretion). The patients' well-being is indicated by the blood sugar (more accurately, the glicated haemoglobin - HbA1c) level.

Usage

```
data("diabetes")
```

Format

A data frame with 87598 observations on the following 11 variables.

`ID` Patient ID

`DATE` Date of the sampling/observation

`AGE` Age of the patient

`THERAPY` Therapy type

`THERAPY_COST_EUR` Therapy cost

`HEALTHCARE_BURDEN_EUR` Event (e.g. heart attack) cost for the health care provider

`HBA1C_AVG` Blood sugar average for the 30-day sampling cycle

`HBA1C_SD` Blood sugar standard deviation for the 30-day sampling cycle

`SAMPLING_IN_MONTH` Number of sampling for the 30-day sampling cycle

`ICD` Diabetes diagnosis type (International Classification of Diseases)

`THERAPY_VECTOR` Therapy vector of the patient, i.e. taking into account the time the therapy lasts after initiation

Details

The example data focuses on two therapy types: insulin analogues (artificial insulins) and glucagon-like peptides (GLP, promotes insulin secretion). Of course there are more treatment types, the database also lists oral antidiabetics (OAD) and human insulins, but we choose to make the data simpler by focusing on GLP and analogue therapies. For the sake of comparison the therapies are grouped in this way: the first group is insulin analogues with possible parallel OAD therapies but human insulin and GLP excluded. The second group is GLP therapies with possible parallel OAD and insulin analogue therapies but human insulin excluded. Essentially we are comparing the effect and cost of insulin analogues with the effect and cost of additional GLP therapies. For cost calculations, the 2021 March 21 EUR-HUF exchange rate was used (1 EUR = 369.05 HUF).

The example below contains a lengthy code which exemplifies the process of gathering useful data for control chart use. Detailed application of this data can be found in the package's vignette.

Source

The original dataset is based on a month-aggregated time series data of diabetic patients from Hungary which was gathered from the period of 2007 September - 2017 September. The data came from two sources: the National Health Insurance Fund of Hungary and the South-Pest Central Hospital. The first source provided information about diagnoses, treatments, health care event and related costs while the latter provided laboratory data regarding blood sugar level. Patients with International Classification of Diseases (ICD) codes (diagnosis) of E10, E11 and E14, and at least one blood sugar measurement were included initially. Only the data of patients with at least one E11 (type II diabetes) diagnosis in the study period was kept. An additional homogenising filter was the requirement of age above 40 at the time of the first diagnosis. Disease progression and therapy effectiveness estimation required at least two blood sugar (HbA1c) measurements with simultaneous therapy data. A total of 4434 patients satisfied all conditions out of which 2151 had at least two HbA1c measurements.

References

https://ecmiindmath.org/2019/08/19/markov-chain-based-cost-optimal-control-charts/

Examples

```
data("diabetes")
str(diabetes)

## Not run:
##### Example of data assessment for control chart use #####
### Packages
require(zoo)
require(reshape2)

RANDOMISED_DATA <- diabetes

### Functions
weighted.var <- function(x, w, na.rm = FALSE) {
    if (na.rm) {
        w <- w[i <- !is.na(x)]
        x <- x[i]
    }
    sum.w <- sum(w)
    sum.w2 <- sum(w^2)
    mean.w <- sum(x * w) / sum(w)
    (sum.w / (sum.w^2 - sum.w2)) * sum(w * (x - mean.w)^2, na.rm =
na.rm)
}

estBetaParams <- function(mu, var) {
  alpha <- ((1 - mu) / var - 1 / mu) * mu ^ 2
  beta <- alpha * (1 / mu - 1)
  return(params = list(alpha = alpha, beta = beta))
}

### Setting up data
```

```
# Way too high HbA1C levels are discarded as outliers
RANDOMISED_DATA$HBA1C_AVG[RANDOMISED_DATA$HBA1C_AVG>20 & !is.na(RANDOMISED_DATA$HBA1C_AVG)] <- NA

# Lowest HbA1c level taken into account
lowest <- 4

### Gathering data for several estimates
RANDOMISED_DATA <- RANDOMISED_DATA[RANDOMISED_DATA$ID %in%
RANDOMISED_DATA$ID[grepl("E11",RANDOMISED_DATA$ICD)],]

# Process standard deviation
sigma_param <- sigma <- sqrt(weighted.mean((RANDOMISED_DATA$HBA1C_SD[
RANDOMISED_DATA$SAMPLING_IN_MONTH>=2 & !is.na(RANDOMISED_DATA$SAMPLING_IN_MONTH)])^2,
RANDOMISED_DATA$SAMPLING_IN_MONTH[RANDOMISED_DATA$SAMPLING_IN_MONTH>=2 &
!is.na(RANDOMISED_DATA$SAMPLING_IN_MONTH)]))

IDLIST <- unique(RANDOMISED_DATA$ID[!is.na(RANDOMISED_DATA$HBA1C_AVG)][
duplicated(RANDOMISED_DATA$ID[!is.na(RANDOMISED_DATA$HBA1C_AVG)])])
IDLIST <- unique(RANDOMISED_DATA$ID[(RANDOMISED_DATA$ID %in% IDLIST) & RANDOMISED_DATA$AGE>39])

shiftdat <- NULL
stimedat <- NULL
repaidat <- NULL
deltats  <- NULL
deltaATC <- NULL
for(i in IDLIST)
{
smalldat   <- RANDOMISED_DATA[RANDOMISED_DATA$ID==i,c("DATE","HBA1C_AVG","THERAPY_VECTOR")]
smalldat    <- smalldat[!is.na(smalldat$DATE) & !is.na(smalldat$HBA1C_AVG),]
patshiftdat <- as.data.frame(cbind(i,smalldat$DATE[2:dim(smalldat)[1]],diff(smalldat$DATE),
diff(smalldat$HBA1C_AVG))[diff(smalldat$HBA1C_AVG)>2*sigma,,drop=FALSE])
if(dim(patshiftdat)[1]>1) stimedat <- rbind(stimedat,cbind(i,diff(as.Date(patshiftdat$V2))))
patrepaidat <- as.data.frame(cbind(i,diff(smalldat$DATE),(smalldat$HBA1C_AVG-lowest)[2:
   length(smalldat$HBA1C_AVG)]/(smalldat$HBA1C_AVG-lowest)[1:(length(smalldat$HBA1C_AVG)-1)],
as.character(smalldat$THERAPY_VECTOR[1:(length(smalldat$THERAPY_VECTOR)-1)]))[
(which(diff(smalldat$HBA1C_AVG)<(-2*sigma) &
smalldat$HBA1C_AVG[1:(length(smalldat$HBA1C_AVG)-1)]>6 &
smalldat$HBA1C_AVG[1:(length(smalldat$HBA1C_AVG)-1)]<=20)),,drop=FALSE])

shiftdat <- rbind(shiftdat,patshiftdat)
repaidat <- rbind(repaidat,patrepaidat)
deltats  <- rbind(deltats,cbind(i,diff(as.Date(RANDOMISED_DATA$DATE[
!is.na(RANDOMISED_DATA$HBA1C_AVG) & RANDOMISED_DATA$ID==i]))))
try(deltaATC <- rbind(deltaATC,cbind(i,diff(as.Date(RANDOMISED_DATA$DATE[
!is.na(RANDOMISED_DATA$THERAPY) & RANDOMISED_DATA$ID==i])))), silent=TRUE)
}
colnames(shiftdat) <- c("ID","TIME","TIMEDIFF","SHIFTSIZE")
colnames(deltats)  <- c("ID","DeltaT")
colnames(deltaATC) <- c("ID","deltaATC")

# delta: expected shift size
delta_param <- mean(shiftdat$SHIFTSIZE[shiftdat$TIMEDIFF>=90 & shiftdat$TIMEDIFF<184])
```

```
# Empirical distribution of elapsed times (between samplings)
summary(deltats[,2])
mean(deltats[,2])
median(deltats[,2])
sd(deltats[,2])

# s: expected number of shifts per unit time
stimedat           <- as.data.frame(stimedat)
colnames(stimedat) <- c("ID","TIMEDIFF")
s_param            <- 1/mean(stimedat$TIMEDIFF[stimedat$TIMEDIFF<367])

# alphas, betas: therapy effectiveness parameters
colnames(repaidat) <- c("ID","TIMEDIFF","REMAIN","THERAP")
repaidat$REMAIN    <- as.numeric(as.character(repaidat$REMAIN))
repaidat$TIMEDIFF  <- as.numeric(as.character(repaidat$TIMEDIFF))
repaidat$THERAP    <- as.character(repaidat$THERAP)
repaidat           <- repaidat[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184,]
repaidat$REMAIN[repaidat$REMAIN<0] <- 0

ther_eff <- as.data.frame(rbind(
cbind("ANALOGUE",repaidat$REMAIN[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184 &
grepl("ANALOGUE",repaidat$THERAP) & !grepl("-H",repaidat$THERAP) & !grepl("GLP",repaidat$THERAP)]),
cbind("GLP",repaidat$REMAIN[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184 &
grepl("GLP",repaidat$THERAP) & !grepl("-H",repaidat$THERAP)])))
ther_eff[,1]       <- factor(ther_eff[,1], levels = c("ANALOGUE", "GLP"))
ther_eff[,2]       <- as.numeric(as.character(ther_eff[,2]))
colnames(ther_eff) <- c("Type","Effectiveness")

ANALOGUE <- estBetaParams(mean(repaidat$REMAIN[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184 &
grepl("ANALOGUE",repaidat$THERAP) & !grepl("-H",repaidat$THERAP) & !grepl("GLP",repaidat$THERAP)]),
 var(repaidat$REMAIN[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184 &
 grepl("ANALOGUE",repaidat$THERAP) & !grepl("-H",repaidat$THERAP) &
 !grepl("GLP",repaidat$THERAP)]))
GLP <- estBetaParams(mean(repaidat$REMAIN[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184 &
                       grepl("GLP",repaidat$THERAP) & !grepl("-H",repaidat$THERAP)]),
                     var(repaidat$REMAIN[repaidat$TIMEDIFF>=90 & repaidat$TIMEDIFF<184 &
                       grepl("GLP",repaidat$THERAP) & !grepl("-H",repaidat$THERAP)]))

### Repair cost
HBA1C_fill <- NULL
for (i in unique(RANDOMISED_DATA$ID[!is.na(RANDOMISED_DATA$HBA1C_AVG)]))
{
  vec <- RANDOMISED_DATA$HBA1C_AVG[RANDOMISED_DATA$ID==i]
  vec[which(is.na(vec))[which(is.na(vec))<which(!is.na(vec))[1]]] <- vec[which(!is.na(vec))[1]]
  vec[which(is.na(vec))[which(is.na(vec))>which(!is.na(vec))[length(which(!is.na(vec)))]]] <-
    vec[which(!is.na(vec))[length(which(!is.na(vec)))]]
  vec <- na.approx(vec)
  HBA1C_fill <- rbind(HBA1C_fill,cbind(i,vec))

  smaldat <- RANDOMISED_DATA[RANDOMISED_DATA$ID==i,]
  smaldat$THERAPY_COST_EUR[smaldat$THERAPY_COST_EUR==0 & smaldat$THERAPY_VECTOR!=""] <- NA
  if(is.na(smaldat$THERAPY_COST_EUR[1])) smaldat$THERAPY_COST_EUR[1]                <- 0
  new_burden <- na.locf(smaldat$THERAPY_COST_EUR)
  seged                      <- cbind(rle(is.na(smaldat$THERAPY_COST_EUR))[[2]],
                                      rle(is.na(smaldat$THERAPY_COST_EUR))[[1]])
  seged[,2][seged[,1]==0]   <- seged[,2][seged[,1]==0]-1
  seged[,2][seged[,1]==1]   <- seged[,2][seged[,1]==1]+1
 if(seged[length(seged[,1]),1]==0) seged[length(seged[,2]),2] <- seged[length(seged[,2]),2]+1
  seged2                    <- cbind(rep(seged[,1], seged[,2]),rep(seged[,2], seged[,2]))
  new_burden[seged2[,1]==1] <- new_burden[seged2[,1]==1]/seged2[,2][seged2[,1]==1]

  RANDOMISED_DATA$THERAPY_COST_EUR[RANDOMISED_DATA$ID==i] <-new_burden
}

RANDOMISED_DATA$HBA1C_fill <- NA
RANDOMISED_DATA$HBA1C_fill[RANDOMISED_DATA$ID%in%HBA1C_fill[,1]] <- HBA1C_fill[,2]
RANDOMISED_DATA$HBA1C_fill_filter <- RANDOMISED_DATA$HBA1C_fill
RANDOMISED_DATA$HBA1C_fill_filter[RANDOMISED_DATA$HBA1C_fill_filter>=10] <- NA

discparam    <-150
cutHBA1C_AVG <-cut(na.omit(RANDOMISED_DATA$HBA1C_fill_filter),breaks=discparam)
newlvls      <-seq(min(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)),
                   max(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)),
                   (max(na.omit(RANDOMISED_DATA$HBA1C_fill_filter))-
              min(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)))/discparam)[1:discparam] +
                   (max(na.omit(RANDOMISED_DATA$HBA1C_fill_filter))-
                     min(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)))/discparam/2
levels(cutHBA1C_AVG) <- newlvls
costs                <- cbind(as.numeric(as.character(cutHBA1C_AVG)),
                              RANDOMISED_DATA$THERAPY_COST_EUR[!is.na(
                              RANDOMISED_DATA$HBA1C_fill_filter)]/30,
                              as.character(RANDOMISED_DATA$THERAPY[
                              !is.na(RANDOMISED_DATA$HBA1C_fill_filter)]))
costs           <- as.data.frame(costs)
colnames(costs) <- c("HBA1C","HC_BURDEN","THERAP")
costs$HBA1C     <- as.numeric(as.character(costs$HBA1C))
costs$HC_BURDEN <- as.numeric(as.character(costs$HC_BURDEN))
costs$THERAP    <- as.character(costs$THERAP)

costs$THERAP[grepl("ANALOGUE", costs$THERAP) & !grepl("GLP", costs$THERAP)] <- "ANALOGUE"
costs$THERAP[grepl("GLP",costs$THERAP)]                                     <- "GLP"

cost.ANALOGUE <- as.data.frame(cbind(sort(unique(costs$HBA1C[costs$THERAP=="ANALOGUE"])),
                           as.numeric(tapply(costs$HC_BURDEN[costs$THERAP=="ANALOGUE"],
                                 costs$HBA1C[costs$THERAP=="ANALOGUE"],mean))))
colnames(cost.ANALOGUE) <- c("HBA1C","HC_BURDEN")

cost.GLP <-as.data.frame(cbind(sort(unique(costs$HBA1C[costs$THERAP=="GLP"])),
                           as.numeric(tapply(costs$HC_BURDEN[costs$THERAP=="GLP"],
                             costs$HBA1C[costs$THERAP=="GLP"],mean))))
colnames(cost.GLP) <-c("HBA1C","HC_BURDEN")

## ANALOGUE therapy
# Mean
cost.ANALOGUE           <- na.omit(as.data.frame(cbind(as.numeric(
                                     costs$HBA1C[costs$THERAP=="ANALOGUE"]),
                                     costs$HC_BURDEN[costs$THERAP=="ANALOGUE"])))
colnames(cost.ANALOGUE) <- c("HBA1C","HC_BURDEN")
cost.ANALOGUE           <- cost.ANALOGUE[order(cost.ANALOGUE$HBA1C),]
cost.ANALOGUE           <- cost.ANALOGUE[cost.ANALOGUE$HBA1C>lowest,]
cost.ANALOGUE$HBA1C     <- cost.ANALOGUE$HBA1C-min(lowest)

# Fit non-linear model
mod.ANALOGUE <- nls(HC_BURDEN ~  a + b/(HBA1C + c),
                    start = list(a = 5, b = -5, c = 1), cost.ANALOGUE,
                    control = list(maxiter = 50000, minFactor = 0.000000000000001))

cost_ANALOGUE_plotdat <- cbind("ANALOGUE",as.data.frame(cbind(seq(0,6,6/99),
                                predict(mod.ANALOGUE,
                                 newdata=data.frame(HBA1C = seq(0,6,6/99))))))

# Variance
cost_var.ANALOGUE  <- na.omit(as.data.frame(cbind(sort(unique(
                               costs$HBA1C[costs$THERAP=="ANALOGUE"])),
                            as.numeric(tapply(costs$HC_BURDEN[costs$THERAP=="ANALOGUE"],
                               costs$HBA1C[costs$THERAP=="ANALOGUE"],var)))))
colnames(cost_var.ANALOGUE) <- c("HBA1C","HC_BURDEN")
cost_var.ANALOGUE           <- cost_var.ANALOGUE[cost_var.ANALOGUE$HBA1C>lowest,]
cost_var.ANALOGUE$HBA1C     <- cost_var.ANALOGUE$HBA1C-min(lowest)

# Fit non-linear model
mod_var.ANALOGUE <- nls(HC_BURDEN ~  a + b/(HBA1C + c),
                        start = list(a = 5, b = -3, c = 0.1),
                        cost_var.ANALOGUE[cost_var.ANALOGUE$HBA1C<10-lowest,],
                        control = list(maxiter = 50000, minFactor = 0.000000000000001))

cost_ANALOGUE_plotdat <- cbind(cost_ANALOGUE_plotdat,
                               cost_ANALOGUE_plotdat[,3] -
                                 sqrt(predict(mod_var.ANALOGUE,
                                   newdata=data.frame(HBA1C = seq(0,6,6/99)))),
                               cost_ANALOGUE_plotdat[,3] +
                                 sqrt(predict(mod_var.ANALOGUE,
                                   newdata=data.frame(HBA1C = seq(0,6,6/99)))))
colnames(cost_ANALOGUE_plotdat) <- c("Data","HbA1c","Therapy cost","low","high")

## GLP
# Mean
cost.GLP           <- na.omit(as.data.frame(cbind(as.numeric(
                                costs$HBA1C[costs$THERAP=="GLP"]),
                                costs$HC_BURDEN[costs$THERAP=="GLP"])))
colnames(cost.GLP) <- c("HBA1C","HC_BURDEN")
cost.GLP           <- cost.GLP[order(cost.GLP$HBA1C),]
cost.GLP           <- cost.GLP[cost.GLP$HBA1C>lowest,]
cost.GLP$HBA1C     <- cost.GLP$HBA1C-min(lowest)

# Simple linear
mod.GLP <- nls(HC_BURDEN ~ a + b * HBA1C,
               start = list(a = 1, b = 1), cost.GLP,
               control = list(maxiter = 50000, minFactor = 0.000000000000001))

cost_GLP_plotdat <-cbind("GLP",as.data.frame(cbind(seq(0,6,6/99),
                     predict(mod.GLP, newdata=data.frame(HBA1C = seq(0,6,6/99))))))

# Variance
cost_var.GLP           <- na.omit(as.data.frame(cbind(sort(unique(
                                   costs$HBA1C[costs$THERAP=="GLP"])),
                                 as.numeric(tapply(costs$HC_BURDEN[costs$THERAP=="GLP"],
                                   costs$HBA1C[costs$THERAP=="GLP"],var)))))
colnames(cost_var.GLP) <- c("HBA1C","HC_BURDEN")
cost_var.GLP           <- cost_var.GLP[cost_var.GLP$HBA1C>lowest,]
cost_var.GLP$HBA1C     <- cost_var.GLP$HBA1C-min(lowest)

# Simple linear
mod_var.GLP <- nls(HC_BURDEN ~  a + b*(HBA1C),
                   start = list(a = 5, b = -3), cost_var.GLP,
                   control = list(maxiter = 50000, minFactor = 0.000000000000001))

cost_GLP_plotdat <- cbind(cost_GLP_plotdat,
                          cost_GLP_plotdat[,3] -
                   sqrt(predict(mod_var.GLP, newdata=data.frame(HBA1C = seq(0,6,6/99)))),
                          cost_GLP_plotdat[,3] +
                   sqrt(predict(mod_var.GLP, newdata=data.frame(HBA1C = seq(0,6,6/99)))))
colnames(cost_GLP_plotdat) <- c("Data","HbA1c","Therapy cost","low","high")

### Out-of-control cost
COST_CUMU<-NULL
for (i in unique(RANDOMISED_DATA$ID[!is.na(RANDOMISED_DATA$HEALTHCARE_BURDEN_EUR)]))
{
vec       <- RANDOMISED_DATA$HEALTHCARE_BURDEN_EUR[RANDOMISED_DATA$ID==i]
vec2      <- rollapply(vec,min(6,length(vec)),
              sum,align="left",partial=TRUE)/
               (pmin(length(vec)-(1:length(vec))+1,6)*30)
COST_CUMU <- rbind(COST_CUMU,cbind(i,vec2))
}

RANDOMISED_DATA$COST_CUMU <- NA
RANDOMISED_DATA$COST_CUMU[RANDOMISED_DATA$ID%in%COST_CUMU[,1]] <- COST_CUMU[,2]

discparam    <- 150
cutHBA1C_AVG <- cut(na.omit(RANDOMISED_DATA$HBA1C_fill_filter),breaks=discparam)
newlvls      <- seq(min(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)),
                    max(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)),
                    (max(na.omit(RANDOMISED_DATA$HBA1C_fill_filter))-
               min(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)))/discparam)[1:discparam] +
                      (max(na.omit(RANDOMISED_DATA$HBA1C_fill_filter))-
                        min(na.omit(RANDOMISED_DATA$HBA1C_fill_filter)))/discparam/2
levels(cutHBA1C_AVG) <- newlvls
ooc_costs <- cbind(round(as.numeric(as.character(cutHBA1C_AVG)),1),
              RANDOMISED_DATA$COST_CUMU[!is.na(RANDOMISED_DATA$HBA1C_fill_filter)])
ooc_costs <- as.data.frame(ooc_costs)
```

```
# Mean
disc_ooc_cost           <- as.data.frame(cbind(as.numeric(ooc_costs[,1]),ooc_costs[,2]))
colnames(disc_ooc_cost) <- c("HBA1C","COST")
disc_ooc_cost           <- disc_ooc_cost[order(disc_ooc_cost$HBA1C),]
disc_ooc_cost           <- disc_ooc_cost[disc_ooc_cost$HBA1C >= lowest,]
disc_ooc_cost$HBA1C     <- disc_ooc_cost$HBA1C - lowest

mod.COST <- nls(COST ~ a + c*HBA1C^2, start = list(a = 1, c = 1), disc_ooc_cost)

cost_COST_plotdat <- cbind("Complications",as.data.frame(cbind(seq(0, 6, 6/99),
                       predict(mod.COST, newdata=data.frame(HBA1C = seq(0, 6, 6/99))))))

# Variance
disc_ooc_cost_var           <- as.data.frame(cbind(sort(unique(ooc_costs[,1])),
                                 as.numeric(tapply(ooc_costs[,2],ooc_costs[,1],var))))
colnames(disc_ooc_cost_var) <- c("HBA1C","COST")

disc_ooc_cost_var       <- disc_ooc_cost_var[disc_ooc_cost_var$HBA1C>=lowest,]
disc_ooc_cost_var$HBA1C <- disc_ooc_cost_var$HBA1C-lowest

mod_var.COST <- nls(COST ~ a + c*HBA1C^2,
                    start = list(a = 0.5, c = 0.5), disc_ooc_cost_var,
                    control = list(maxiter = 50000, minFactor = 0.000000000000001))

cost_COST_plotdat <- cbind(cost_COST_plotdat,
                           cost_COST_plotdat[,3] -
                             sqrt(predict(mod_var.COST,
                               newdata=data.frame(HBA1C = seq(0,6,6/99)))),
                           cost_COST_plotdat[,3] +
                             sqrt(predict(mod_var.COST,
                               newdata=data.frame(HBA1C = seq(0,6,6/99)))))
colnames(cost_COST_plotdat) <- c("Data","HbA1c","Therapy cost","low","high")

cost_plots       <- rbind(cost_ANALOGUE_plotdat,cost_GLP_plotdat,cost_COST_plotdat)
cost_plots$HbA1c <- cost_plots$HbA1c + lowest
cost_plots[,"Therapy cost"]               <- cost_plots[,"Therapy cost"]/1
cost_plots[,"low"]                        <- cost_plots[,"low"]/1
cost_plots[,"low"][cost_plots[,"low"]<0]  <- 0
cost_plots[,"high"]                       <- cost_plots[,"high"]/1

cost_plots <- cost_plots

### Sampling cost: official, government-regulated prices related to sampling
### converted from HUF to EUR
sampling_cost=2875/369.05

### Empirical costs for comparison
# GLP
mean(RANDOMISED_DATA[grepl("GLP", RANDOMISED_DATA$THERAPY),]$THERAPY_COST_EUR)/30 +
mean(RANDOMISED_DATA[grepl("GLP", RANDOMISED_DATA$THERAPY),]$COST_CUMU) +
sampling_cost/mean(deltats[,2])

sd(RANDOMISED_DATA[grepl("GLP", RANDOMISED_DATA$THERAPY),]$THERAPY_COST_EUR/30 +
RANDOMISED_DATA[grepl("GLP", RANDOMISED_DATA$THERAPY),]$COST_CUMU +
sampling_cost/mean(deltats[,2]))

# ANALOGUE
mean(RANDOMISED_DATA[grepl("ANALOGUE", RANDOMISED_DATA$THERAPY) &
 !grepl("GLP", RANDOMISED_DATA$THERAPY),]$THERAPY_COST_EUR)/30 +
mean(RANDOMISED_DATA[grepl("ANALOGUE", RANDOMISED_DATA$THERAPY) &
 !grepl("GLP", RANDOMISED_DATA$THERAPY),]$COST_CUMU) +
sampling_cost/mean(deltats[,2])

sd(RANDOMISED_DATA[grepl("ANALOGUE", RANDOMISED_DATA$THERAPY) &
 !grepl("GLP", RANDOMISED_DATA$THERAPY),]$THERAPY_COST_EUR/30 +
RANDOMISED_DATA[grepl("ANALOGUE", RANDOMISED_DATA$THERAPY) &
 !grepl("GLP", RANDOMISED_DATA$THERAPY),]$COST_CUMU +
sampling_cost/mean(deltats[,2]))

### Empirical HbA1c distribution
# GLP
empi.GLP  <- RANDOMISED_DATA$HBA1C_fill[grepl("GLP", RANDOMISED_DATA$THERAPY) &
               RANDOMISED_DATA$HBA1C_fill>=4 & RANDOMISED_DATA$HBA1C_fill<=20]
cutHBA1C  <- cut(na.omit(empi.GLP),breaks=100)
newlvls   <- seq(min(na.omit(empi.GLP)),max(na.omit(empi.GLP)),
                     (max(na.omit(empi.GLP))-min(na.omit(empi.GLP)))/100)[1:100] +
                      (max(na.omit(empi.GLP))-min(na.omit(empi.GLP)))/100/2
levels(cutHBA1C)    <- newlvls
empi.GLP            <- as.data.frame(table(cutHBA1C)/sum(table(cutHBA1C)))
empi.GLP$cutHBA1C   <- as.numeric(as.character(empi.GLP$cutHBA1C))
empi.GLP            <- cbind("GLP", empi.GLP)
colnames(empi.GLP)  <- c("Therapy", "HbA1c", "Probability")

# ANALOGUE
empi.ANALOGUE  <- RANDOMISED_DATA$HBA1C_fill[grepl("ANALOGUE", RANDOMISED_DATA$THERAPY) &
                    !grepl("GLP", RANDOMISED_DATA$THERAPY) &
                      RANDOMISED_DATA$HBA1C_fill>=4 & RANDOMISED_DATA$HBA1C_fill<=20]
cutHBA1C        <- cut(na.omit(empi.ANALOGUE),breaks=100)
newlvls         <- seq(min(na.omit(empi.ANALOGUE)),
                       max(na.omit(empi.ANALOGUE)),
                       (max(na.omit(empi.ANALOGUE))-
                         min(na.omit(empi.ANALOGUE)))/100)[1:100] +
                          (max(na.omit(empi.ANALOGUE))-
                            min(na.omit(empi.ANALOGUE)))/100/2
levels(cutHBA1C)        <- newlvls
empi.ANALOGUE           <- as.data.frame(table(cutHBA1C)/sum(table(cutHBA1C)))
empi.ANALOGUE$cutHBA1C  <- as.numeric(as.character(empi.ANALOGUE$cutHBA1C))
empi.ANALOGUE           <- cbind("ANALOGUE", empi.ANALOGUE)
colnames(empi.ANALOGUE) <- c("Therapy", "HbA1c", "Probability")

# Merging datasets
hba1c_empir <- rbind(empi.ANALOGUE, empi.GLP)

# The list of gathered data and statistics:
# sigma_param, s_param, delta_param, ANALOGUE
# GLP, mod.ANALOGUE, mod_var.ANALOGUE
# mod.GLP, mod_var.GLP, mod.COST, mod_var.COST
# cost_plots, sampling_cost, hba1c_empir


## End(Not run)
```

`LDL` *Aggregated low-density-lipoprotein patient data for control chart applications*

Description

A data frame containing aggregated low-density-lipoprotein (LDL) patient data gathered from various sources.

Usage

```
data("LDL")
```

Format

A data frame with 1 observation on the following 12 variables.

`target_value` Target LDL value. Set according to the European guideline for patients at risk. Garmendia et al. (2000)

`standard_deviation` Process standard deviation. Estimated using real life data from Hungary, namely registry data through Healthware Consulting Ltd. (Budapest, Hungary).

`expected_shift_size` Expected shift size, around 0.8 increase in LDL per year on average, due to the expected number of shifts per year. Estimated with the help of a health care professional from the Healthware Consulting Ltd.

`num_exp_daily_shifts` We expect around 3 shifts per year on average. Estimated with the help of a health professional from the Healthware Consulting Ltd.

`rep_size_first` First repair size distribution parameter. Estimated using an international study which included Hungary. Garmendia et al. (2000)

`rep_size_second` Second repair size distribution parameter.

`samp_prob_first` First sampling probability parameter. Patient non-compliance in LDL controlling medicine is quite high, and this is represented through the parametrisation of the logistic function. Lardizabal and Deedwania (2010)

`samp_prob_second` Second sampling probability parameter.

`sampling_cost` Sampling cost in Euro. Estimated using the official LDL testing cost and visit cost in Hungary.

`OOC_cost` Out-of-control operation (health care event) cost in Euro. Estimated using real world data of cardiovascular event costs from Hungary

`base_rep_cost` Base repair (treatment) cost in Euro. Estimated using the simvastatin therapy costs in Hungary

`prop_rep_cost` Shift-proportional (illness-proportional) repair cost in Euro. Estimated using the simvastatin therapy costs in Hungary.

Details

It is very difficult to give a good estimate for the type and parameters of a distribution that properly models the non-compliance (sampling probability), thus the data here can at best be regarded as close approximations to a real-life situation. This is not a limiting factor, as patients themselves can have vast differences in their behaviour, so evaluation of different scenarios are often required. Since high LDL levels rarely produce noticeable symptoms, the sampling probability should only depend on the time between samplings (Institute for Quality and Efficiency in Health Care, 2017). Thus, the sampling probability parameters assume the use of a logistic function and not a beta distribution in the `Markovstat` function. It is important to note that the proportional costs usually assumed to increase according to a Taguchi-type loss function (squared loss), thus huge expenses can be generated if the patient's health is highly out-of-control. For cost calculations, the 2021 March 21 EUR-HUF exchange rate was used (1 EUR = 369.05 HUF).

Source

Dobi, B. and Zempléni, A. Markov chain-based cost-optimal control charts for health care data. Quality and Reliability Engineering International, 2019;35(5):1379–1395. https://doi.org/10.1002/qre.2518

References

Boekholdt SM, Arsenault BJ, Mora S, et al. Association of LDL cholesterol, non–HDL cholesterol, and apolipoprotein B levels with risk of cardiovascular events among patients treated with statins: a meta-analysis. J Am Med Assoc. 2012;307(12):1302-1309. https://doi.org/10. 1001/jama.2012.366

Garmendia F, Brown AS, Reiber I, Adams PC. Attaining United States and European guideline LDL-cholesterol levels with simvastatin in patients with coronary heart disease (the GOALLS study). Curr Med Res Opin. 2000;16(3):208-219. PMID: 11191012.

Lardizabal JA, Deedwania PC. Benefits of statin therapy and compliance in high risk cardiovascular patients. Vasc Health Risk Manag. 2010;6:843-853. https://doi/org/10.2147/VHRM.S9474

High cholesterol: Overview. Institute for Quality and Efficiency in Health Care. https://www.ncbi.nlm.nih.gov/books/NBK27
[10 September 2021] Bookshelf ID: NBK279318.

Examples

```
#Print data
data("LDL")
LDL

###

# Run analysis from Dobi & Zempleni 2019.
# Note that the code below is generated with updated HUF-EUR rate and
# a more accurate R implementation than in the original paper.

stat_LDL_cost <- Markovstat(
  shiftfun = 'exp', h = 50, k = 3.15-LDL$target_value,
  sigma = LDL$standard_deviation,
  s = LDL$num_exp_daily_shifts,
  delta = LDL$expected_shift_size,
  RanRep = TRUE, alpha = LDL$rep_size_first, beta = LDL$rep_size_second,
  RanSam = TRUE, q = LDL$samp_prob_first, z = LDL$samp_prob_second,
  Vd = 100, V = 6-LDL$target_value)

#Defining parallel_opt parallel settings.
#parallel_opt can also be left empty to be defined automatically by the function.
require(parallel)
num_workers <- min(c(detectCores(),2))
parall <- list(cl=makeCluster(num_workers), forward=FALSE, loginfo=TRUE)
res_LDL_cost <- Markovchart(
  statdist = stat_LDL_cost,
  OPTIM = TRUE, p = 1,
  cs = LDL$sampling_cost,
  cf = LDL$base_rep_cost,
  coparams = c(0,LDL$OOC_cost),
  crparams = c(LDL$base_rep_cost,LDL$prop_rep_cost),
  parallel_opt = parall)

num_workers <-  min(c(detectCores(),2))
parall      <-  list(cl=makeCluster(num_workers), forward=FALSE, loginfo=TRUE)
res_LDL_cost_grid <- Markovchart(
  statdist = stat_LDL_cost,
  h=seq(50,75,2.5),
  k=seq(0.05,0.25,0.02),
  p = 1,
  cs = LDL$sampling_cost,
  cf = LDL$base_rep_cost,
  coparams = c(0,LDL$OOC_cost),
  crparams = c(LDL$base_rep_cost,LDL$prop_rep_cost),
  parallel_opt = parall)

require(ggplot2)
plot(res_LDL_cost_grid,
     y = 'Expected \ndaily cost',
     mid = '#ff9494',
     high = '#800000',
     xlab = 'Days between samplings',
     ylab = 'Critical LDL increase') +
     geom_point(aes(x = res_LDL_cost$Parameters[[1]],
                    y = res_LDL_cost$Parameters[[2]]))
```

`Markovchart` *Cost-efficient X-bar control charts with fixed/random shift size, random repair and random sampling time.*

Description

Wrapper for Markov chain-based cost optimal control charts. Includes cost calculation methods for different shift size distributions and optimisation with respect to the average cost and cost standard deviation where the free parameters are the sampling interval (h) and the control limit/critical value (k).

Usage

```
Markovchart(statdist, h = NULL, k = NULL,
            OPTIM = FALSE, p = 1, constantr = FALSE,
            ooc_rep = 0, cs = NULL, cofun = cofun_default,
            coparams = NULL, crfun = crfun_default, crparams = NULL,
            cf = crparams, vcofun = vcofun_default,
            vcoparams = c(0, 0), vcrfun = vcrfun_default,
            vcrparams = c(0, 0), method = c("L-BFGS-B", "Nelder-Mead",
            "BFGS", "CG", "SANN", "Brent"), parallel_opt = NULL,
            silent = TRUE, ...)
```

Arguments

`statdist`	The stationary distribution of the Markov chain. Must be an object of class `Markov_stationary`, preferably created by `Markovstat`.
`h`	The time between samplings. Must be a positive value, can be a numeric vector. For optimisation, this is the initial value. Inherited from `statdist` if not given.
`k`	The control limit (critical value). Must be a positive value, can be a numeric vector. For optimisation, this is the initial value. Only one sided shifts are allowed, thus there is only one control limit. Inherited from `statdist` if not given.
`OPTIM`	Logical. Should the resulting `G-value` (weighted average of the expected cost and cost standard deviation) be optimised by finding the adequate value of `h` and `k`.
`p`	The weight of the cost expectation in the calculation of the `G-value`; should be between 0 and 1.
`constantr`	Logical. Should the repair cost be assumed to constantly occur over time (`TRUE`) or assumed to only occur when there is a repair due to an alarm (`FALSE`, default)? If `TRUE`, then the repair cost should be given per unit time.
`ooc_rep`	Numeric value between 0 and 1. The percentage of repair cost ocurring during out-of-control operation. Default is 0. If a value greater than 0 is set, then `constantr` should be `TRUE`, but it is not forced.
`cs`	Sampling cost per sampling.
`cofun`	A function describing the relationship between the distance from the target value and the resulting out-of-control costs. Default is calculated using a base and a distance-scaling out-of-control parameter. See "Details".
`coparams`	Numeric vector. Parameters of `cofun`.
`crfun`	A function describing the relationship between the distance from the target value and the resulting repair costs. The default function assumes a linear relationship between the repair cost and the distance, and uses a base and a distance-scaling repair cost parameter. See "Details".
`crparams`	Numeric vector. Parameters of `crfun`.
`cf`	Numeric. The false alarm cost. Only relevant when `statdist` was created using `shiftfun="deg"`.

vcofun	A function describing the relationship between the distance from the target value and the resulting out-of-control cost variance. For the default function see "Details".
vcoparams	Numeric vector. Parameters of vcofun.
vcrfun	A function describing the relationship between the distance from the target value and the resulting repair cost variance. For the default function see "Details".
vcrparams	Numeric vector. Parameters of vcrfun.
method	Method used for optimisation. Same as the argument of optim, but the default here is "L-BFGS-B", because it turned out to be more robust in testing.
parallel_opt	A list of parallel options. See e.g. the argument parallel in the documentation of optimParallel. Can be left empty, in this case the number of cores (threads) is automatically detected and all but one is used. (Single-core computers simply use one core.)
silent	Should the call be returned? Default is FALSE.
...	Further arguments to be passed down to optimParallel.

Details

The constantr parameter is used for different repair assumptions. In traditional control chart theory, repair cost only occurs in case of an alarm signal. This is represented by constantr=FALSE, which is the default. In this case the repair is just a momentary cost, occurring at the time of the sampling. However this model is inappropriate in several cases in healthcare. For example there are chronic diseases that require constant medication (repair in the sense of the model). In this approach (constantr=TRUE) the repair cost still depends on the state of the process during sampling, but occurs even if there is no alarm and is divided by h to represent the constant repair through the whole sampling interval. Thus the repair cost should be given in a way which corresponds to the model used.

The default cofun calculates the out-of-control (OOC) cost using a base and a distance-scaling OOC parameter:

$$c_o = c_{ob} + c_{os}A^2(v),$$

where c_o is the total OOC cost, c_{ob} is the base OOC cost (even without shift), c_{os} is the shift-scaling cost and $A^2(v)$ is the squared distance from the target value. This latter part is defined like this because a Taguchi-type loss function is used. This $A^2(v)$ incorporates the distances (the base of the losses) incurred not just at the time of the sampling, but also between samplings (hence it dependens on h). Even if the user defines a custom cost function for the OOC cost, this $A^2(v)$ term must be included, as a closed form solution has been developed for the calculation of the squared distances in case of exponential shifts, considerably decreasing run times. Thus the arguments of the OOC cost function should look like this: function($A^2(v)$, other parameters contained in a vector). $A^2(v)$ is fed to the cost function as a vector, thus the function should vectorised with respect to this argument. The default function looks like this:

```
cofun_default <-function(sqmudist,coparams)
{
  sqmudist=sqmudist
  cob=coparams[1]
  cos=coparams[2]
  co <-cob + cos*sqmudist
  return(co)
}
```

The default vcofun also uses a Taguchi-type loss function and has identical parts and requirements as cofun. The final standard deviation itself is calculated using the law of total variance. The default vcofun is:

```
vcofun_default <-function(sqmudist,vcoparams)
{
  sqmudist=sqmudist
  vcob=vcoparams[1]
  vcos=vcoparams[2]
  vco <-vcob + vcos*sqmudist
  return(vco)
}
```

The defaults for the repair cost and cost variance are simple linear functions. For crfun it is

$$c_r = c_{rb} + c_{rs}v,$$

where the notation are the same as before and "r" denotes repair. A custom function can be defined more freely here, but the first argument should be v and the second a vector of further parameters.

The default function are:

```
crfun_default <-function(mudist,crparams)
{
  mudist=mudist
  crb=crparams[1]
  crs=crparams[2]
  cr <-crb + crs*mudist
  return(cr)
}
```

```
vcrfun_default <-function(mudist,vcrparams)
{
  mudist=mudist
  vcrb=vcrparams[1]
  vcrs=vcrparams[2]
  vcr <-vcrb + vcrs*mudist;
  return(vcr)
}
```

Value

The value depends on the parameters:

If either `h` or `k` have length greater than 1, then the `G-value` (weighted average of average cost and cost standard deviation) is calculated for all given values without optimisation. The value of the function in this case is a data frame of class codeMarkov_grid with `length(h)*length(k)` number of rows and three columns for `h`, `k` and the `G-value`.

If `h` and `k` are both of length 1 (they may be inherited from `statdist`), then the value of the function is a `Markov_chart` object, which is a list of length 4, detailing the properties of the control chart setup.

`Results`	Vector of `G-value`, expected cost, cost standard deviation and further process moments. Note that these further moments only take into account the process variation (i.e. the standard deviation of the process itself), while the "Total cost std. dev." takes into account all sources of variance (e.g. the different costs that can occur due to being out-of-control). The "Total cost std. dev." is only relevant and calculated for non-degenerate distributions.
`Subcosts`	Vector of sub-costs that are parts of the total expected cost.
`Parameters`	A vector that contains the time between samplings (h) and critical value (k) which was used in the control chart setup.
`Stationary_distribution`	The stationary distribution of the Markov chain. Further information about the stationary distribution can be calculated using the `Markovstat` function.

Author(s)

Balazs Dobi and Andras Zempleni

References

Zempleni A, Veber M, Duarte B and Saraiva P. (2004) Control charts: a cost-optimization approach for processes with random shifts. *Applied Stochastic Models in Business and Industry*, 20(3), 185-200.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts for health care data. *Quality and Reliability Engineering International*, 35(5), 1379-1395.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts with different shift size distributions. *Annales Univ. Sci. Budapest., Sect. Comp.*, 49, 129-146.

Examples

```
#Defining parallel_opt parallel settings.
#parallel_opt can also be left empty to be defined automatically by the function.
require(parallel)
num_workers <-  min(c(detectCores(),2))
parall     <-  list(cl=makeCluster(num_workers), forward=FALSE, loginfo=TRUE)


#Fixed shift size (essentially Duncan's cycle model) - no optimisation.
stat_deg <- Markovstat(shiftfun="deg", h=1, k=1, sigma=1, s=0.2, delta=2.5)
```

```
res1     <- Markovchart(statdist=stat_deg, cs=1, crparams=20, coparams=50)
res1

#Fixed shift size (essentially Duncan's cycle model) - with optimisation.
res2 <-  Markovchart(statdist=stat_deg, OPTIM=TRUE, cs=1, crparams=20, coparams=50,
                    lower = c(0.01,0.01), upper = c(5,5),
                    parallel_opt=parall)
res2

#Exponential shift - no optimisation - default cost functions.
stat_exp <- Markovstat(shiftfun="exp", h=0.5, k=2, sigma=1, s=0.2, delta=2,
                       RanRep=FALSE, Vd=30, V=18)
res3     <- Markovchart(stat_exp, p=0.9, cs=1, coparams=c(10,3), crparams=c(1,2))
res3

#Exponential shift - with optimisation - default cost functions.
stat_exp2 <- Markovstat(shiftfun="exp", h=1, k=1, sigma=1, s=0.2, delta=2,
                        RanRep=TRUE, alpha=1, beta=3, Vd=30, V=18)
parall    <- list(cl=makeCluster(num_workers), forward=FALSE, loginfo=TRUE)
res4      <- Markovchart(statdist=stat_exp2, OPTIM=TRUE, p=0.9, cs=1,
                         coparams=c(10,3), crparams=c(1,2), vcoparams=c(8,1.5),
                         vcrparams=c(5,2), parallel_opt=parall)
res4

#Exponential-geometric mixture shift - no optimisation -
#random sampling - custom repair variance function.
stat_expgeo <- Markovstat(shiftfun="exp-geo",h=1.5, k=2, sigma=1,
                          s=0.2, delta=1.2, probmix=0.7, probnbin=0.8,
                          disj=2, RanRep=TRUE, alpha=1, beta=3, RanSam=TRUE,
                          StateDep=TRUE, a=1, b=15, Vd=100, V=8)

vcrfun_new <-function(mudist,vcrparams)
{
  mudist=mudist
  vcrb=vcrparams[1]
  vcrs=vcrparams[2]
  vcrs2=vcrparams[3]

  vcr <-vcrb + vcrs/(mudist + vcrs2)
  return(vcr)
}

res5 <- Markovchart(statdist=stat_expgeo, p=0.9, cs=1,
                   coparams=c(10,6), crparams=c(20,3),
                   vcoparams=c(10000,100), vcrfun=vcrfun_new,
                   vcrparams=c(50000,-600000,1.5))
res5

#Exponential shift - no optimisation  - vectorised.
parall <- list(cl=makeCluster(num_workers), forward=FALSE, loginfo=TRUE)
Gmtx   <-Markovchart(statdist=stat_exp2, h=seq(1,10,by=(10-1)/5),
                      k=seq(0.1,5,by=(5-0.1)/5), p=0.9, cs=1,
                      coparams=c(10,3), crparams=c(1,2),
                      vcoparams=c(8,1.5), vcrparams=c(5,2),
                      V=18, parallel_opt=parall)
Gmtx
```

`Markovsim` *Progression and monitoring simulation of a process with random shift size, random repair and random sampling time.*

Description

Wrapper for simulation of processes with a Markov chain-based control chart setup. Includes methods for different shift size distributions.

Usage

```
Markovsim(shiftfun = c("exp", "exp-geo"), num = 100, h, k, sigma,
          s, delta, probmix = 0, probnbin = 0.5, disj=1,
          RanRep = FALSE, alpha = NULL, beta = NULL, RanSam = FALSE,
          StateDep = FALSE, a = NULL, b = NULL, q = NULL,
          z = NULL, detail = 100, Vd = 50, V, burnin = 1)
```

Arguments

`shiftfun`	A string defining the shift size distribution to be used. Must be either "exp", "exp-geo".
`num`	Integer. The number of sampling intervals simulated. This means that the time elapsed in the simulation is `num*h`.
`h`	The time between samplings. Must be a positive value.
`k`	The control limit (critical value). Must be a positive value. Only one sided shifts are allowed, thus there is only one control limit.
`sigma`	Process standard deviation (the distribution is normal).
`s`	Expected number of shifts in an unit time interval.
`delta`	Expected shift size.
`probmix`	The weight of the geometric distribution in case of exponential-geometric mixture shift distribution and should be between 0 and 1.
`probnbin`	The probability parameter of the geometric distribution in case of exponential-geometric mixture shift distribution and should be between 0 and 1.
`disj`	The size of a discrete jump in case of exponential-geometric mixture shift distribution, must be a positive number.
`RanRep`	Logical. Should the repair be random? Default is FALSE (no).
`alpha`	First shape parameter for the random repair beta distribution.
`beta`	Second shape parameter for the random repair beta distribution.

`RanSam`	Logical. Should the sampling be random? Default is FALSE (no).
`StateDep`	Logical. Should the sampling probability also depend on the distance from the target value (state dependency)? (If TRUE, a beta distribution is used for the sampling probability, if FALSE then a logistic function.)
`a`	First parameter*`h` for the random sampling time beta distribution. The first shape parameter is `a/h` to create dependency on the time between samplings.
`b`	Second shape parameter for the random sampling time beta distribution.
`q`	The steepness of the curve of the random sampling time logistic function.
`z`	The logistic sigmoid's midpoint of the random sampling time logistic function.
`detail`	The detail of the simulation, i.e. how many data points (including the moment of the sampling itself) should be simulated within a unit time. Should be a positive integer greater than 1, and the user should consider the length of the sampling interval `h`, as a shorter interval leads to less datapoints.
`Vd`	Integer discretisation parameter: the number of states after the equidistant discretisation of the state space. Should be an integer value greater than 2. This parameter is needed to calculate a stationary distibution that can be compared to results of the `Markovchart` function.
`V`	Numeric discretisation parameter: the maximum (positive) distance from the target value taken into account. This parameter is needed to calculate a stationary distribution that can be compared to results of the `Markovchart` function and for the calculation of sampling probabilities in the case of random sampling.
`burnin`	Numeric burn-in parameter: the number of samplings deemed as a burn-in period. Should be an integer greater than one.

Details

The simulation only includes the more complicated process and control chart cases and is meant for model checking and for situations when the exact calculation is problematic (such as low probabilities in the stationary distribution leading to rounding errors).

Value

A `Markov_sim` object which is a list of length 4.

`Value_at_samplings`	The process value at sampling.
`Sampling_event`	The event at sampling, each can either be success (there was a sampling but no alarm), alarm (sampling with alarm) or failure (no sampling occurred).
`Simulation_data`	The simulated data (distances from the target value).
`Stationary_distribution`	The stationary distribution of the Markov chain, created by discretising the simulated data. See the documentaion of the `Markovchart` function.

Author(s)

Balazs Dobi and Andras Zempleni

References

Zempleni A, Veber M, Duarte B and Saraiva P. (2004) Control charts: a cost-optimization approach for processes with random shifts. *Applied Stochastic Models in Business and Industry*, 20(3), 185-200.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts for health care data. *Quality and Reliability Engineering International*, 35(5), 1379-1395.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts with different shift size distributions. *Annales Univ. Sci. Budapest., Sect. Comp.*, 49, 129-146.

See Also

`Markovstat`

Examples

```
#Simulation using exponential shifts, random repair and random samling.
simres1 <-Markovsim(shiftfun="exp", num=500, h=1, k=1, sigma=1, s=0.2, delta=2,
                          RanRep=TRUE, alpha=1, beta=3, RanSam=TRUE, StateDep=TRUE,
                          a=0.1, b=1, V=10)
simres1
hist(simres1[[1]], 20, freq=FALSE)

#Simulation using exponential-geometric mixture shifts, random repair and random samling.
simres2 <-Markovsim(shiftfun="exp-geo", num=500, h=1, k=1, sigma=1, s=0.2, delta=2,
                    probmix=0.9, probnbin=0.6, RanRep=TRUE, alpha=1, beta=3, RanSam=TRUE,
                          StateDep=TRUE, a=0.1, b=1, V=10)
simres2
hist(simres2[[1]], 20, freq=FALSE)
```

`Markovstat` *Stationary distribution calculation for processes with fixed/random shift size, random repair and random sampling time.*

Description

Calculates the stationary distribution of a process described by a discrete state, discrete time Markov chain. The process is described by a degradation-repair cycle type model. The user must give parameters describing both the degradation and the repair. The process is not repaired until the problem is discovered by sampling, hence the control chart setup. The same, single element is monitored (i.e. the sample size is always 1).

Usage

```
Markovstat(shiftfun = c("exp", "exp-geo", "deg"), h, k, sigma,
           s, delta, probmix = 0, probnbin = 0.5, disj = 1,
           RanRep = FALSE, alpha = NULL, beta = NULL,
           RanSam = FALSE, StateDep = FALSE, a = NULL,
           b = NULL, q = NULL, z = NULL, Vd = 100, V,
           Qparam = 30)
```

Arguments

`shiftfun`	A string defining the shift size distribution to be used. Must be either `"exp"` (exponential), `"exp-geo"` (exponential-geometric mixture) or `"deg"` (degenerate). Use `"deg"` for fixed shift size with perfect repair and guaranteed sampling, i.e. Duncan"s traditional cycle model.
`h`	The time between samplings. Must be a positive value.
`k`	The control limit (critical value). Must be a positive value. Only one sided shifts are allowed, thus there is only one control limit.
`sigma`	Process standard deviation (the distribution is assumed to be normal).
`s`	Expected number of shifts in an unit time interval.
`delta`	Expected shift size. Used as the parameter of the exponential distribution (`shiftfun="exp"` or `"exp-geo"`), or simply as the size of the shift (`shiftfun="deg"`).
`probmix`	The weight of the geometric distribution in case of exponential-geometric mixture shift distribution; should be between 0 and 1.
`probnbin`	The probability parameter of the geometric distribution in case of exponential-geometric mixture shift distribution; should be between 0 and 1.
`disj`	The size of a discrete jump in case of exponential-geometric mixture shift distribution, must be a positive number.
`RanRep`	Logical. Should the repair be random? Default is `FALSE` (the repair is perfect, the process is always repaired to the target value). The repair is always perfect (non-random) for `shiftfun="deg"`.
`alpha`	First shape parameter for the random repair beta distribution.
`beta`	Second shape parameter for the random repair beta distribution.
`RanSam`	Logical. Should the sampling be random? Default is `FALSE` (no). The sampling is never random for `shiftfun="deg"`.
`StateDep`	Logical. Should the sampling probability also depend on the distance from the target value (state dependency)? (If TRUE, a beta distribution is used for the sampling probability, if `FALSE` then a logistic function.)
`a`	First parameter*`h` for the random sampling time beta distribution. The first shape parameter is `a/h` to create dependency on the time between samplings as described at the `StateDep` parameter.
`b`	Second shape parameter for the random sampling time beta distribution.
`q`	The steepness of the curve of the random sampling time logistic function.
`z`	The logistic sigmoid"s midpoint of the random sampling time logistic function.
`Vd`	Integer discretisation parameter: the number of states in the equidistant discretisation of the state space. Should be an integer value greater than 2.
`V`	Numeric discretisation parameter: the maximum (positive) distance from the target value taken into account.
`Qparam`	Integer discretisation parameter: the number of maximum events taken into account within a sampling interval.

Value

The function return a list object of class `Markov_stationary`. The list is of length 3:

`Stationary_distribution`	Stationary distribution of the Markov chain. The probabilities in the stationary distribution are labeled. If `shiftfun` is "deg" then the stationary distribution is always of length 4. If `shiftfun` is not "deg" then there are multiple out-of-control (OOC) and true alarm states. These are labeled with an index and the value the state represents. If `shiftfun` is "deg" then the out-of-control and true alarm states are at a distance `delta` from the target value, and the in-control and the false alarm state are always at the target value.
`Transition_matrix`	The transition matrix of the Markov chain. Not printed.
`Param_list`	Parameters given to the function and various technical results used by the `Markovchart` function. Not printed.

Author(s)

Balazs Dobi and Andras Zempleni

References

Zempleni A, Veber M, Duarte B and Saraiva P. (2004) Control charts: a cost-optimization approach for processes with random shifts. *Applied Stochastic Models in Business and Industry*, 20(3), 185-200.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts for health care data. *Quality and Reliability Engineering International*, 35(5), 1379-1395.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts with different shift size distributions. *Annales Univ. Sci. Budapest., Sect. Comp.*, 49, 129-146.

See Also

`Markovchart`

Examples

```
#Fixed shift size (essentially Duncan's cycle model).
res1 <- Markovstat(shiftfun="deg", h=1, k=1, sigma=1, s=0.2, delta=2.5)
res1

#Exponential shift - perfect repair - deterministic sampling
res2 <- Markovstat(shiftfun="exp", h=1, k=1, sigma=1, s=0.2, delta=2, Vd=30, V=18)
res2
#Notice how the In-control and the False-alarm states have non-zero probabilities.
#If the repair would be random (RanRep=TRUE), then these states would have zero probability.

#Exponential-geometric mixture shift - random repair - random sampling.
res3 <- Markovstat(shiftfun='exp-geo', h=1.5, k=2, sigma=1, s=0.2,
                   delta=1.2, probmix=0.7, probnbin=0.8, disj=2,
                   RanRep=TRUE, alpha=1, beta=3, RanSam=TRUE,
                   StateDep=TRUE, a=1, b=15, Vd=40, V=8)
res3
```

`plot.Markov_grid` *Contour plot for* `Markov_grid` *control chart results.*

Description

Convenience function for plotting G-values in a contour plot as the function of the time between samplings and the critical value.

Usage

```
## S3 method for class 'Markov_grid'
plot(
     x, y = expression(atop(italic("G")*-value~per, unit~time)),
     xlab = "Time between samplings", ylab = "Critical value",
     low = "white", mid = "#999999", high = "black",
     colour = "white", nbreaks = 16, ...)
```

Arguments

`x`	A `Markov_grid` data.frame with three columns (preferably created by the Markovchart function): time between samplings, critical value and the weighted mean of the expected cost and the cost standard deviation (G-values).
`y`	The name of the scale.
`xlab`	A title for the x axis.
`ylab`	A title for the x axis.
`low`	Colour for the low end of the gradient.
`mid`	Colour for the midpoint.
`high`	Colour for the high end of the gradient.
`colour`	Colour of the contour lines.
`nbreaks`	Number of contour breaks. Uses `pretty()`, thus actual, plotted number of breaks may differ.
`...`	Further arguments to be passed down to `plot`. Mostly kept due to S3 method compatibility.

Value

A plot object of class gg and `ggplot` produced using the `ggplot2` package.

Note

The plot itself is made using the package `ggplot` by Hadley Wickham et al. The text on the contour lines is added with the `geom_text_contour` function from the package `metR` by Elio Campitelli.

Author(s)

Balazs Dobi and Andras Zempleni

References

Zempleni A, Veber M, Duarte B and Saraiva P. (2004) Control charts: a cost-optimization approach for processes with random shifts. *Applied Stochastic Models in Business and Industry*, 20(3), 185-200.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts for health care data. *Quality and Reliability Engineering International*, 35(5), 1379-1395.

Dobi B and Zempleni A. (2019) Markov chain-based cost-optimal control charts with different shift size distributions. *Annales Univ. Sci. Budapest., Sect. Comp.*, 49, 129-146.

See Also

Markovchart Markovstat

Examples

```
#Defining parallel_opt parallel settings.
#parallel_opt can also be left empty to be defined automatically by the function.
require(parallel)
num_workers <- min(c(detectCores(),2))

#Exponential shift - default cost functions.
stat_exp <- Markovstat(shiftfun="exp", h=1, k=1, sigma=1, s=0.2, delta=2,
                        RanRep=TRUE, alpha=1, beta=3, Vd=30, V=18)

parall <- list(cl=makeCluster(num_workers), forward=FALSE, loginfo=TRUE)
Gmtx   <-Markovchart(statdist=stat_exp, h=seq(1,10,by=(10-1)/5),
                      k=seq(0.1,5,by=(5-0.1)/5), p=0.9, cs=1,
                      coparams=c(10,3), crparams=c(1,2),
                      vcoparams=c(8,1.5), vcrparams=c(5,2),
                      V=18, parallel_opt=parall)
plot(Gmtx)
```

Index