# A new sample-based algorithm to compute the total sensitivity index: R code

*Arnald Puy*

## Contents

# 1 Preliminary steps

```
# LOAD ALL THE REQUIRED PACKAGES ------------------------------------------

# Define function to read in all required libraries in one go:
loadPackages <- function(x) {
  for(i in x) {
    if(!require(i, character.only = TRUE)) {
      install.packages(i, dependencies = TRUE)
      library(i, character.only = TRUE)
    }
  }
}

# Load packages
loadPackages(c("sensobol", "data.table", "ggplot2", "parallel", "scales"))
```

# 2 Define functions

## 2.1 Test functions

```
# TEST FUNCTIONS ----------------------------------------------------------

# Function A1:
A1 <- function(X) {
  # Preallocate
  mat <- tmp <- vector(mode = "list", length = nrow(X))
  Y <- vector(mode = "numeric", length = nrow(X))
  for(i in 1:nrow(X)) {
    mat[[i]] <- matrix(rep(X[i, ], times = ncol(X)),
                       nrow = ncol(X),
                       ncol = ncol(X),
                       byrow = TRUE)
    mat[[i]][upper.tri(mat[[i]])] <- 1
    tmp[[i]] <- matrixStats::rowProds(mat[[i]])
    Y[[i]] <- sum(tmp[[i]] * (-1) ^ (1:ncol(X)))
  }
  return(Y)
}

# Function A2:
A2 <- function(X) {
  a <- c(0, 0.5, 3, 9, 99, 99)
  y <- 1
  for (j in 1:6) {
```

```r
    y <- y * (abs(4 * X[, j] - 2) + a[j])/(1 + a[j])
  }
  return(y)
}

# Function B1:
B1 <- function(X) {
  y <- 1
  for(j in 1:ncol(X)) {
    y <- y * (ncol(X) - X[, j]) / (ncol(X) - 0.5)
  }
  return(y)
}

# Function B2:
B2 <- function(X) {
  y <- 1
  for(j in 1:ncol(X)) {
    y <- y * ((1 + 1 / ncol(X)) ^ ncol(X)) * X[, j] ^ (1 / ncol(X))
  }
  return(y)
}

# Function B3:
B3 <- function(X) {
  a <- rep(6.52, 6)
  y <- 1
  for (j in 1:6) {
    y <- y * (abs(4 * X[, j] - 2) + a[j])/(1 + a[j])
  }
  return(y)
}

# Function C1:
C1 <- function(X) {
  y <- 1
  for (j in 1:ncol(X)) {
    y <- y * (abs(4 * X[, j] - 2))}
  return(y)
}

# Function C2:
C2 <- function(X) {
  2 ^ ncol(X) * matrixStats::rowProds(X)
}
```

## 2.2 Creation of the sample matrices

```r
# SAMPLE MATRICES ------------------------------------------------------------

# Function to split a matrix into N parts
CutBySize <- function(m, block.size, nb = ceiling(m / block.size)) {
  int <- m / nb
  upper <- round(1:nb * int)
  lower <- c(1, upper[-nb] + 1)
  size <- c(upper[1], diff(upper))
  cbind(lower, upper)
}

# Function to create an A and AB matrix
scrambled_sobol <- function(A, B) {
  X <- rbind(A, B)
  for(i in 1:ncol(A)) {
    AB <- A
    AB[, i] <- B[, i]
    X <- rbind(X, AB)
  }
  AB <- rbind(A, X[((2*nrow(A)) + 1):nrow(X), ])
  return(AB)
}

# Function to create replicas of the A, B and AB matrices
scrambled_replicas <- function(N, k, replicas) {
  df <- randtoolbox::sobol(n = N * replicas, dim = k * 2)
  indices <- CutBySize(nrow(df), nb = replicas)
  X <- A <- B <- out <- list()
  for(i in 1:nrow(indices)) {
    lower <- indices[i, "lower"]
    upper <- indices[i, "upper"]
    X[[i]] <- df[lower:upper, ]
  }
  for(i in seq_along(X)) {
    A[[i]] <- X[[i]][, 1:k]
    B[[i]] <- X[[i]][, (k + 1) : (k * 2)]
  }
  for(i in seq_along(A)) {
    out[[i]] <- scrambled_sobol(A[[i]], B[[i]])
  }
  return(out)
}

# Separate matrices
separate_matrices <- function(data) {
```

```
  indices <- CutBySize(nrow(data), nb = k + 1)
  X <- list()
  for(i in 1:nrow(indices)) {
    lower <- indices[i, "lower"]
    upper <- indices[i, "upper"]
    X[[i]] <- data[lower:upper, ]
  }
  return(X)
}

# Define the sample sizes of the sample matrices
# for the new algorithm
computations <- function(x, k) {
  Nb <- x
  # Total number of runs
  Nc <- x * (k + 1)
  # Warm up sample size (one fourth)
  Ntot <- Nc / 4
  # Base sample when using 1/4 of initial sample
  Nts <- Ntot / (k + 1)
  # Number of saved runs
  Nsa <- Nc - Ntot
  # Initial sample size of the saved runs
  Nin <- Nsa / (k + 1)
  # Runs to estimate the remaining 3/4 factors
  Nrem <- Nin * (4 + 1)
  # Runs saved
  Nsaved <- Nsa - Nrem
  # Number of extra runs per factor
  Nextra <- Nsaved / 4
  df <- data.frame(Nb, Nc, Ntot, Nts, Nsa, Nin, Nrem, Nsaved, Nextra)
  return(df)
}
```

```
# SOBOL' STi INDICES -----------------------------------------------------

sobol_compute_Ti <- function(Y_A, Y_AB) {
  f0 <- (1 / length(Y_A)) * sum(Y_A)
  VY <- 1 / length(Y_A) * sum((Y_A - f0) ^ 2)
  STi <- ((1 / (2 * length(Y_A))) * sum((Y_A - Y_AB) ^ 2)) / VY
  return(STi)
}

sobol_Mapply_Ti <- function(d) {
  return(mapply(sobol_compute_Ti,
                d[, "Y_A"],
                d[, "Y_AB"]))
```

```
}

sobol_Ti <- function(Y, params) {
  # Calculate the number of parameters
  k <- length(params)
  # Calculate the length of the A matrix
  p <- length(1:(length(Y) / (k + 1)))
  # Extract the model output of the A matrix
  Y_A <- Y[1:p]
  # Extract the model output of the AB matrix
  Y_AB <- Y[(p+1):length(Y)]
  # Create vector with parameters
  parameters <- rep(params, each = length(Y_A))
  # merge vector with data table
  vec <- cbind(Y_A, Y_AB)
  out <- data.table::data.table(vec, parameters)
  out.1 <- out %>%
    # remove rows with NA
    stats::na.omit()
  # Compute Sobol'indices
  output <- out.1[, sobol_Mapply_Ti(.SD), by = parameters]
  return(output)
}
```

### 2.3 New algorithm

```
# DEFINE THE NEW ALGORITHM ------------------------------------------------

new_estimator <- function(sample.size, type, k, params, replicas, model) {
  # Create vector for the A and AB matrices
  col_names <- c("A", paste("X", 1:k, sep = ""))
  # Define the settings
  setting <- computations(x = sample.size, k)
  # Create the matrices
  A <- scrambled_replicas(setting$Nb, k, replicas)
  names(A) <- 1:replicas
  # Separate the matrices
  X <- lapply(A, function(x) separate_matrices(x))
  # Name the matrices
  for(i in names(X)) {
    names(X[[i]]) <- col_names
  }
  # Compute model output
  Y <- lapply(X, function(x) lapply(x, function(y) model(y)))
  if(type == "old") { # RUN THE TRADITIONAL APPROACH --------------
    ###############################################################
```

```
  # Compute STi following the traditional approach
  out <- lapply(Y, function(x) sobol_Ti(do.call(c, x), params))
  # Finalize
  final <- rbindlist(out, idcol = "replica") %>%
    .[, sample.size := setting$Nb] %>%
    .[, algorithm:= "old"]
}
if(type == "new") { # RUN THE NEW ALGORITHM ---------------------
  ################################################################
  # Retrieve model output for the warm-up samples
  Y.1 <- lapply(Y, function(x) lapply(x, function(y) y[1:setting$Nts]))
  # Compute Sobol' Ti for the warm-up samples
  out <- lapply(Y.1, function(x) sobol_Ti(do.call(c, x), params))
  # Retrieve the fixed parameters
  STi.fixed <- lapply(out, function(x)
    x[V1 < quantile(V1, probs = 1 - 75 / 100)])
  # Retrieve the non-fixed parameters
  STi.non.fixed <- lapply(out, function(x)
    x[V1 > quantile(V1, probs = 1 - 75 / 100)]) %>%
    lapply(., function(x) c("A", x[, parameters]))
  # Retrieve model runs of the non.fixed parameters
  new.runs <- list()
  for(i in names(Y)) {
    new.runs[[i]] <-  Y[[i]][STi.non.fixed[[i]]]
  }
  extra.runs <- lapply(new.runs, function(x)
    lapply(x, function(y) y[(setting$Nts + 1):setting$Nb]))
  # unite the vectors of the same simulation
  temp <- lapply(extra.runs, function(x) c(do.call(cbind, x)))
  # Retrieve vector with the name of the parameters
  # with the extra runs
  new <- lapply(STi.non.fixed, function(x) x[-1])
  # Compute Sobol' Ti for the extra runs
  out <- list()
  for(i in names(temp)) {
    out[[i]] <- sobol_Ti(temp[[i]], params = new[[i]])
  }
  # Cbind the fixed parameters
  all.together <- list()
  for(i in names(out)) {
    all.together[[i]] <- rbind(out[[i]], STi.fixed[[i]]) %>%
      .[order(parameters)]
  }
  # Finalize
  final <- rbindlist(all.together, idcol = "replica") %>%
    .[, sample.size := setting$Nb] %>%
    .[, algorithm:= "new"]
```

```
  }
  return(final)
}
```

# 3 Load analytical values

```
# READ IN THE ANALYTICAL VALUES, COMPUTED BY SAMUELE LO PIANO -----------------

# Read the .csv file
AE <- fread("AE_df.csv")

# Re-arrange
analytical <- melt(AE, id.vars = "Function") %>%
  split(., .$Function) %>%
  lapply(., function(x) x[, .(value)])
```

# 4 Run the model

## 4.1 Define settings

```
# DEFINE GENERAL SETTINGS -----------------------------------------------------

# Create vector with the name of the test functions
test_functions <- c("A1", "A2", "B1", "B2", "B3", "C1", "C2")

# Vector power of two
x <- seq(4, 13, 1)

# Get the initial sample sizes
Nb <- sapply(x, function(x) 2 ^ x)

# Set number of factors
k <- 6
params <- paste("X", 1:k, sep = "")

# Set number of sample matrix replicas
replicas <- 50
```

## 4.2 Run the model

```
# RUN THE MODEL ---------------------------------------------------------------
```

```
out <- list()
run_model <- as.list(c(test_functions))
names(run_model) <- test_functions
estimators <- c("new", "old")
for(i in names(run_model)) {
  if(i == "A1") {
    test_F <- A1
  } else if(i == "A2") {
    test_F <- A2
  } else if(i == "B1") {
    test_F <- B1
  } else if(i == "B2") {
    test_F <- B1
  } else if(i == "B3") {
    test_F <- B3
  } else if(i == "C1") {
    test_F <- C1
  } else if(i == "C2") {
    test_F <- C2
  }
  out[[i]] <- lapply(estimators, function(x)
    lapply(Nb, function(Nb) new_estimator(sample.size = Nb,
                                          type = x,
                                          k = k,
                                          params = params,
                                          replicas = replicas,
                                          model = test_F)))
}
```

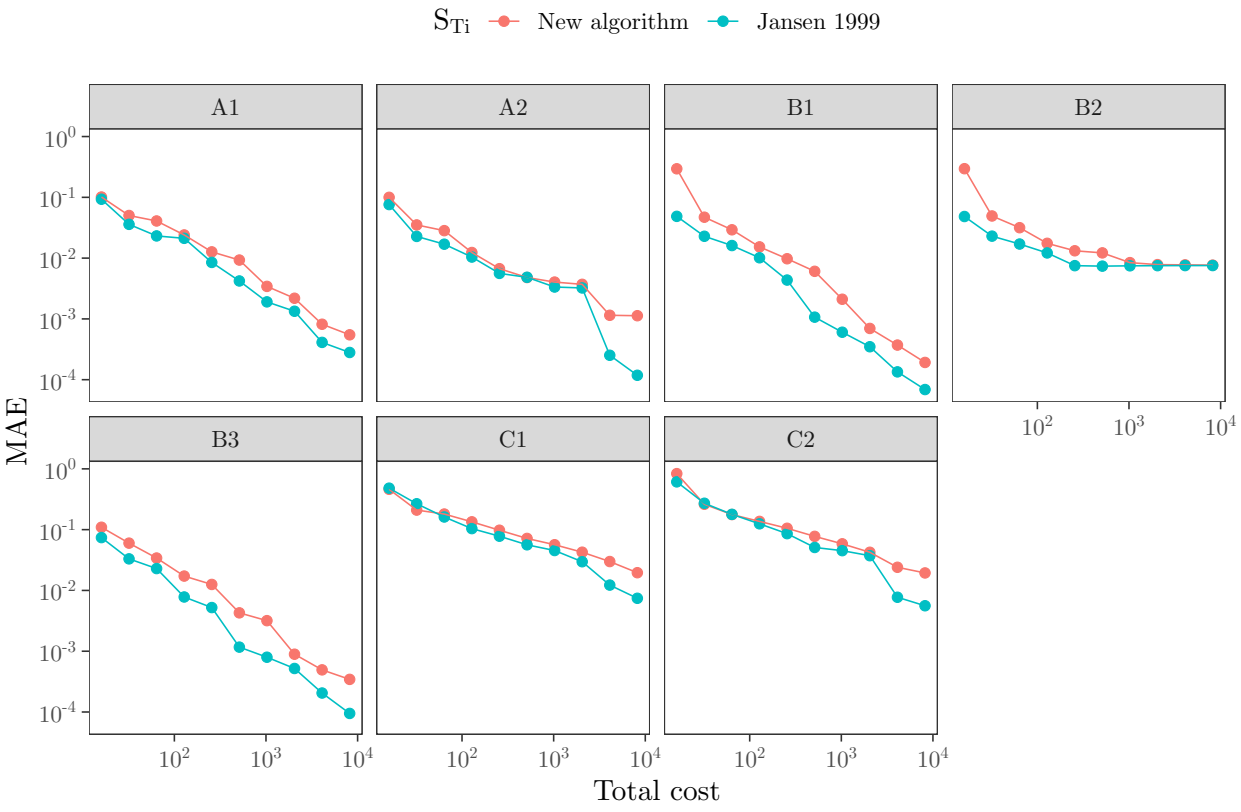
### 4.3 Compute the Mean Absolute Error (MAE)

```
# COMPUTE MAE ------------------------------------------------------------------

# Arrange data
temp <- lapply(out, function(x)
  lapply(x, function(y) rbindlist(y))) %>%
  lapply(., function(x) rbindlist(x))

# Merge indices with analytical values
for(i in names(temp)) {
  temp[[i]] <- cbind(temp[[i]], analytical[[i]])
}

# Compute the MAE
MAE <- rbindlist(temp, idcol = "Function") %>%
  setnames(., c("V1", "value"), c("estimated", "analytical")) %>%
```

```
  .[, .(MAE = mean(abs(estimated - analytical))),
    .(Function, sample.size, algorithm)] %>%
  .[, sample.size:= as.numeric(sample.size)]
```

### 4.4 Plot results

```
# PLOT MAE ---------------------------------------------------------------------

ggplot(MAE, aes(sample.size, MAE, color = algorithm)) +
  geom_point() +
  geom_line() +
  scale_color_discrete(name = expression(S[Ti]),
                       labels = c("New algorithm", "Jansen 1999")) +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x))) +
  scale_y_log10(labels = trans_format("log10", math_format(10^.x))) +
  labs(x = "Total cost",
       y = "MAE") +
  facet_wrap(~Function,
             ncol = 4) +
  theme_bw() +
  theme(aspect.ratio = 1,
        legend.position = "top",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        legend.background = element_rect(fill = "transparent",
                                         color = NA),
        legend.key = element_rect(fill = "transparent",
                                  color = NA))
```


$S_{Ti}$
New algorithm
Jansen 1999
A1
A2
B1
B2
B3
C1
C2
MAE
Total cost
$10^0$
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
$10^2$
$10^3$
$10^4$