Is VARS more intuitive and efficient than Sobol' indices?

R code

Arnald Puy

Contents

1 Presentation

This document presents the code workflow of the paper "Is VARS more intuitive and efficient than Sobol' indices?", by A. Puy, S. Lo Piano, and A. Saltelli, currently under review. The abstract is the following:

The Variogram Analysis of Response Surfaces (VARS) has been proposed by Razavi and Gupta in Water Resources Research as a new comprehensive framework in sensitivity analysis. According to the authors, VARS provides a more intuitive notion of sensitivity and it is much more computationally efficient than Sobol' indices. Here we review these arguments and critically compare the performance of VARS-TO, for total-order index, against the total-order Jansen estimator. We argue that, unlike classic variance-based methods, VARS lacks a clear definition of what is an "important" factor, and prove that the alleged computational superiority of VARS does not hold when its uncertain space is thoroughly explored. We conclude that while VARS enriches the spectrum of existing methods for sensitivity analysis, especially for a diagnostic use of mathematical models, it complements rather than substitutes classic estimators used in variance-based sensitivity analysis.

The results of the paper should be fully reproducible in any personal computer. Questions about the code or the computational design should be addressed to A. Puy (apuy@princeton.edu, arnald.puy@pm.me).

1.1 Preliminary functions

We start by creating a function to load all required libraries for the analysis in one go. We also set a checkpoint to ensure that our code is fully reproducible for anyone anytime. Finally, we design a short theme function for all the plots that will be produced in the analysis.

```r
# PRELIMINARY FUNCTIONS -------------------------------------------------------

# Function to read in all required packages 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)
    }
  }
}

# Install development version of sensobol
remotes::install_github("arnaldpuy/sensobol")

# Load the packages
loadPackages(c("tidyverse", "sensobol", "data.table", "parallel",
               "foreach", "doParallel", "pcaPP", "scales",
               "cowplot", "logitnorm", "benchmarkme", "Rfast",
               "wesanderson", "xlsx", "grid", "gridExtra"))
```

```r
# Create custom theme
theme_AP <- function() {
  theme_bw() +
    theme(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))
}

# Set checkpoint
dir.create(".checkpoint")
library("checkpoint")

checkpoint("2020-11-01",
           R.version ="4.0.3",
           checkpointLocation = getwd())
```

2 The issue of intuitivity

In this section we present the code that produces Figs.1 and 2 of the main manuscript.

2.1 Functions to plot

```r
# DEFINE FUNCTIONS TO PLOT ------------------------------------------------------

fig1_fun <- list(
  "fun1" = function(x) x ^ 2,
  "fun2" = function(x) ifelse(x < 0, -x, x),
  "fun3" = function(x) ifelse(x < 0, - (x + 1) ^ 2 + 1, - (x - 1) ^ 2 + 1)
)

fig2_fun <- list(
  "fun1" = function(x) 1.11 * x ^ 2,
  "fun2" = function(x) x ^ 2 - 0.2 * cos(7 * pi * x)
)

fig3_fun <- list(
  "fun1" = function(x) x,
  "fun2" = function(x) ((-1) ^ as.integer(4 * x) * (0.125 - (x %% 0.25)) + 0.125),
  "fun3" = function(x) ((-1) ^ as.integer(32 * x) *
                           (0.03125 - 2 * (x %% 0.03125)) + 0.03125) / 2
)

fig4_fun <- list(
  "fun1" = function(x) -sin(pi * x) - 0.3 * sin(3.33 * pi * x),
  "fun2" = function(x) -0.76 * sin(pi * (x - 0.2)) - 0.315,
  "fun3" = function(x) -0.12 * sin(1.05 * pi * (x - 0.2)) -
    0.02 * sin(95.24 * pi * x) - 0.96,
  "fun4" = function(x) -0.12 * sin(1.05 * pi * (x - 0.2)) - 0.96,
  "fun5" = function(x) -0.05 * sin(pi * (x - 0.2)) - 1.02,
  "fun6" = function(x) -1.08
)

# FUNCTION TO PLOT THE FUNCTIONS ----------------------------------------------

plot_function <- function(fun, min, max) {
  gg <- ggplot(data.frame(x = runif(1000, min, max)), aes(x)) +
    map(1:length(fun), function(nn) {
      stat_function(fun = fun[[nn]],
                    geom = "line",
                    aes_(color = factor(names(fun[nn]))))
    }) +
    labs(color = "Function",
         x = expression(italic(x)),
         y = expression(italic(y))) +
    theme_AP()
  return(gg)
}
```

2.2 Plot Figures

```
# PLOT FUNCTIONS ---------------------------------------------------------------

a <- plot_function(fun = fig1_fun, -1, 1) +
  scale_color_manual(labels = c("$f_1(x)$","$f_2(x)$", "$f_3(x)$"),
                     values = c("#F8766D", "#B79F00", "#00BA38")) +
  labs(x = "", y = "$y$") +
  theme(legend.position = "none")

b <- plot_function(fun = fig2_fun, -1, 1) +
  scale_color_manual(labels = c("$f_1(x)$","$f_2(x)$"),
                   values = c("#F8766D", "#B79F00")) +
  labs(x = "", y = "") +
  theme(legend.position = "none")

c <- plot_function(fun = fig3_fun, 0, 1) +
  scale_color_manual(labels = c("$f_1(x)$","$f_2(x)$", "$f_3(x)$"),
                   values = c("#F8766D", "#B79F00", "#00BA38")) +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 3)) +
  labs(x = "$x$", y = "$y$") +
  theme(legend.position = "none")

d <- plot_function(fun = fig4_fun, 0, 1) +
  scale_color_discrete(labels = c("$f_1(x)$","$f_2(x)$", "$f_3(x)$",
                                  "$f_4(x)$","$f_5(x)$", "$f_6(x)$")) +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 3)) +
  labs(x = "$x$", y = "") +
  theme(legend.position = "none")

legend <- get_legend(d + theme(legend.position = "top"))

bottom <- plot_grid(a, b, c, d, ncol = 2, align = "hv", labels = "auto")

plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.2, 1))
```

```
# PLOT FIGURE LIU --------------------------------------------------------------

mat <- randtoolbox::sobol(n = 1000, dim = 2)
mat[, 1] <- qchisq(mat[, 1], df = 10)
mat[, 2] <- qchisq(mat[, 2], df = 13.978)

fig.5.tikz <- data.table(mat) %>%
```

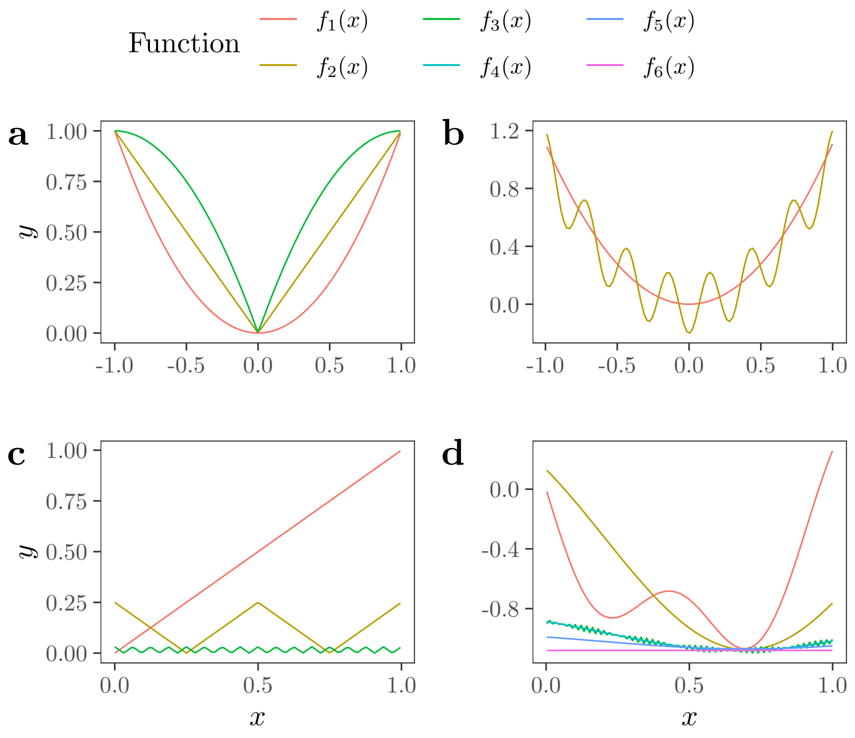


Figure 1: Examples of functions in Razavi and Gupta (2016). a) Unimodal functions with different structures. b) Multimodal versus unimodal function with identical variance. c) Functions covering different ranges in the response. d) A six-dimensional response surface. See Supplementary Materials for a mathematical description of all functions in all sub-plots.

```
  melt(., measure.vars = c("V1", "V2")) %>%
  ggplot(., aes(value, group = variable, colour = variable)) +
  geom_density() +
  labs(x = expression(italic(x)),
       y = "Density") +
  scale_color_discrete(name = "",
                       labels = c("$x_1 \\sim \\chi_{10} ^ {2}$",
                                  "$x_2 \\sim \\chi_{13.978} ^ {2}$")) +
  theme_AP() +
  theme(legend.text.align = 0,
        legend.position = c(0.93, 0.8))

# Fig.6
mat <- randtoolbox::sobol(n = 1000, dim = 2)
Y <- qchisq(mat[, 1], df = 10) / qchisq(mat[, 2], df = 13.978)
X1.fixed <- 10 / qchisq(mat[, 2], df = 13.978)
X2.fixed <- qchisq(mat[, 1], df = 10) / 13.978

fig.6.tikz <- cbind(Y, X1.fixed, X2.fixed) %>%
  data.table() %>%
  melt(., measure.vars = c("Y", "X1.fixed", "X2.fixed")) %>%
  ggplot(., aes(value, color = variable)) +
  geom_density() +
  scale_color_discrete(name = "", labels = c("Original PDF of $y$",
                                             "$x_1$ fixed",
                                             "$x_2$ fixed")) +
  labs(x = expression(italic(y)),
       y = "") +
  theme_AP() +
  theme(legend.text.align = 0,
        legend.position = c(0.6, 0.8))

plot_grid(fig.5.tikz, fig.6.tikz, labels = "auto", align = "hv", ncol = 2)
```

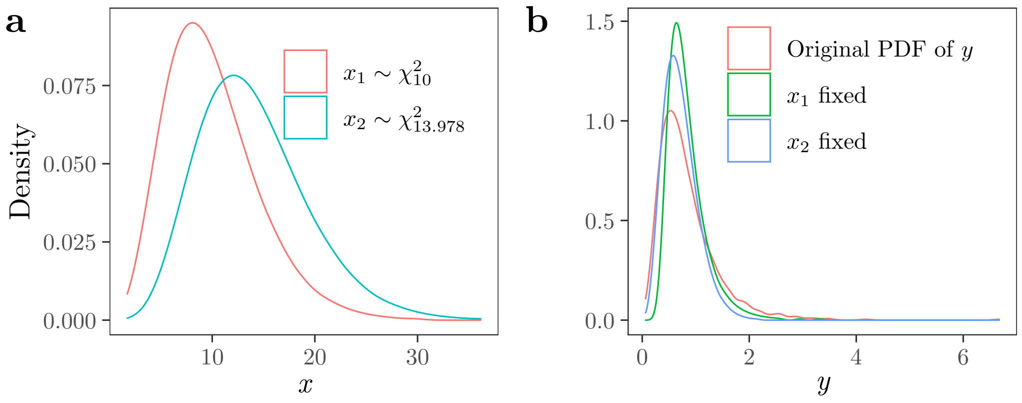


Figure 2: The highly skewed function of Liu, Chen, and Sudjianto (2006). a) Distribution of $x_1$ and $x_2$. b) Comparison of impacts of inputs.

2.3 Analysis of the literature citing VARS

We also provide the code used to analyze the literature citing VARS.

```
# ANALYSIS OF THE LITERATURE -----------------------------------------------

dt.vars <- read.xlsx("VARS_literature_completed.xlsx",
                     sheetIndex = 1,
                     colIndex = c(1:2, 4:14)) %>%
  data.table()

vars.measures <- c("IVARS10", "IVARS30", "IVARS50", "VARS_TO")
vars.measures.plot <- c("IVARS$_{10}$", "IVARS$_{30}$", "IVARS$_{50}$", "VARS-TO")

dt.vars <- setnames(dt.vars, vars.measures, vars.measures.plot)

# Number of VARS measures used in each study
dt.vars[, n.vars.measures:= rowSums(.SD == TRUE, na.rm = TRUE), .SDcols = vars.measures.plot]

# New column
dt.vars <- dt.vars[, Authors.vars:= str_detect(Authors, "Razavi|Gupta")]
```

```r
# EXTRACTION OF SOME STATISTICS ---------------------------------------------------

# Total number of studies citing VARS
total.N <- dt.vars[, .N]

# Total number of studies using VARS
total.using.vars <- dt.vars[, sum(Used.vars == TRUE)]

# Total number of studies that use or does not use VARS
dt.vars[, .(count = .N, proportion = .N / total.N), Used.vars]
```

```
##    Used.vars count proportion
## 1:     FALSE    53  0.6162791
## 2:      TRUE    33  0.3837209
```

```r
# Only when VARS is used
dt.vars[Used.vars == TRUE, .(count = .N, proportion = .N / total.using.vars), Authors.vars]
```

```
##    Authors.vars count proportion
## 1:         TRUE    13  0.3939394
## 2:        FALSE    20  0.6060606
```

```r
# PLOT FIGURES

a <- dt.vars[, .N, Used.vars] %>%
  .[, variable:= "Used VARS?"] %>%
  .[, Used.vars:= ifelse(Used.vars == TRUE, "Yes", "No")]

b <- dt.vars[Used.vars == "TRUE"] %>%
  melt(., measure.vars = vars.measures.plot) %>%
  .[!is.na(value), ] %>%
  .[, .(N = sum(value == TRUE)), variable] %>%
  setnames(., "variable", "Used.vars") %>%
  .[, variable:= "Which IVARS measure?"]

plotA <- rbind(a, b) %>%
  ggplot(., aes(Used.vars, N)) +
  geom_bar(stat = "identity") +
  facet_grid(~variable,
             space = "free_x",
             scales = "free") +
  theme_AP() +
  labs(x = "", y = "$N$") +
  theme(strip.background = element_rect(fill = "white"))

plotB <- dt.vars[Used.vars == "TRUE"] %>%
  .[, .N, Authors.vars] %>%
  .[, Authors.vars:= ifelse(Authors.vars == TRUE, "Yes",
                              ifelse(Authors.vars == FALSE, "No", Authors.vars))] %>%
  ggplot(., aes(Authors.vars, N)) +
  geom_bar(stat = "identity") +
  theme_AP() +
  labs(x = "Authors VARS", y = "")

top <- plot_grid(plotA, plotB, labels = "auto", align = "hv", rel_widths = c(1, 0.45))
```

```
## Warning: Graphs cannot be vertically aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

```r
c <- data.frame(V1 = as.numeric(unlist(strsplit(dt.vars$k, split = ",")))) %>%
  ggplot(., aes(V1)) +
  geom_histogram() +
  labs(x = "$k$", y = "Count") +
  theme_AP()

d <- data.frame(V1 = as.numeric(unlist(strsplit(dt.vars$N.stars, split = ",")))) %>%
  ggplot(., aes(V1)) +
  geom_histogram() +
  labs(x = "N$_{stars}$", y = "") +
  theme_AP()

e <- data.frame(V1 = as.numeric(unlist(strsplit(dt.vars$h, split = ",")))) %>%
  ggplot(., aes(V1)) +
  geom_histogram() +
  labs(x = "$h$", y = "") +
  theme_AP()

bottom <- plot_grid(c, d, e, ncol = 3, labels = c("c", "d", "e"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 57 rows containing non-finite values (stat_bin).
```

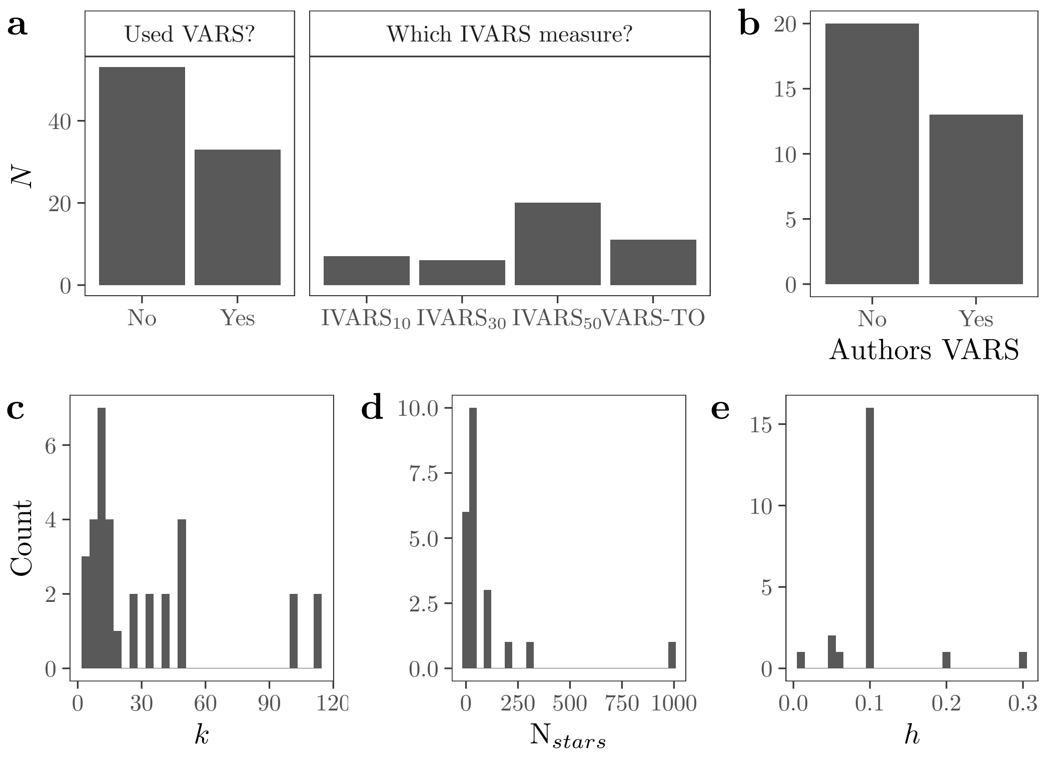
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 67 rows containing non-finite values (stat_bin).
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

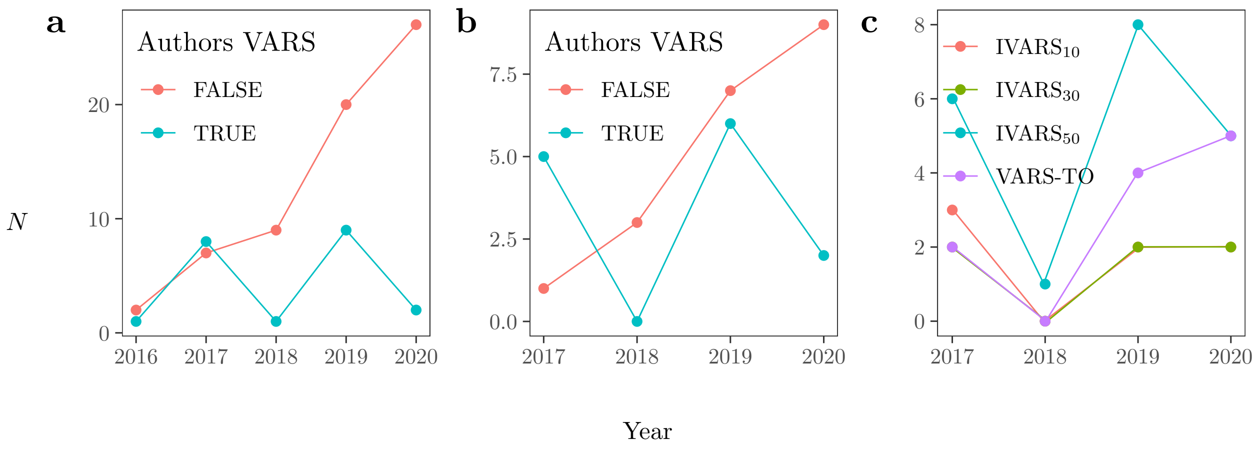
```
## Warning: Removed 65 rows containing non-finite values (stat_bin).
```

```r
plot_grid(top, bottom, ncol = 1)
```



```r
# PLOT FIGURES ------------------------------------------------------------------

# Papers citing VARS through time
setkey(dt.vars, Authors.vars, Year)
a <- dt.vars[CJ(Authors.vars, Year, unique = TRUE), .N, by = .EACHI] %>%
  ggplot(., aes(Year, N, color = Authors.vars, group = Authors.vars)) +
  geom_line() +
  geom_point() +
  scale_color_discrete(name = "Authors VARS") +
  labs(x = "", y = "") +
  theme_AP() +
  theme(legend.position = c(0.33, 0.75))

# Papers that apply VARS
temp <- dt.vars[Used.vars == TRUE]
setkey(temp, Authors.vars, Used.vars, Year)
b <- temp[CJ(Authors.vars, Used.vars,  Year, unique = TRUE), .N, by = .EACHI] %>%
  ggplot(., aes(Year, N, color = Authors.vars, group = Authors.vars)) +
  geom_line() +
  geom_point() +
  scale_color_discrete(name = "Authors VARS") +
  labs(x = "", y = "") +
  theme_AP() +
  theme(legend.position = c(0.33, 0.75))
```

```r
# Proportion of IVARS measures used
temp <- dt.vars[Used.vars == "TRUE"] %>%
  melt(., measure.vars = vars.measures.plot)
setkey(temp, variable, Year)
c <- temp[CJ(variable, Year, unique = TRUE), sum(value, na.rm = TRUE), by = .EACHI] %>%
  ggplot(., aes(Year, V1, color = variable, group = variable)) +
  geom_line(position=position_jitter(w = 0.02, h = 0.02)) +
  geom_point() +
  scale_color_discrete(nam= "") +
  labs(x = "", y = "") +
  theme_AP() +
  theme(legend.position = c(0.33, 0.75))

all <- plot_grid(a, b, c, ncol = 3, align = "hv", labels = "auto")

grid.arrange(arrangeGrob(all, bottom = grid::textGrob(label = "Year"),
                         left = grid::textGrob(label = "$N$")))
```



3 The issue of efficiency

This section presents the code to compare the efficiency of VARS and the Jansen estimator (Jansen 1999). We first start by coding all the functions required to compute VARS:

The function `vars_matrices` creates the STAR-VARS sample matrix that VARS needs in order to compute VARS-TO. In our code, the sample matrix can be created with random numbers or with Sobol' Quasi Random Numbers (the default) (Sobol' 1967, @Sobol1976).

The function `vars_ti` allows to compute VARS-TO on the model output obtained from a STAR-VARS matrix. Our code pairs values separated by h (`method=one.step`) or separated by h, $2h$, $3h$,... (the default, `method=all.step`).

We also code the Jansen estimator (Jansen 1999) (`jansen_ti`), and a function to compute Savage scores (`savage_scores`) (Savage 1956). Finally, we code a function that allows to randomly transform the distribution of model inputs to normal, logitnormal or beta distributions (`random_distributions`). This is done in order to check how VARS-TO and Jansen behave when the underlying model input distributions is uncertain (parameter ϕ) in the main manuscript.

```r
# CREATE STAR-VARS MATRICES ----------------------------------------------------

vars_matrices <- function(star.centers, params, h, method = "QRN") {
  out <- center <- sections <- A <- B <- AB <- X <- out <- list()
  if(method == "QRN") {
    mat <- randtoolbox::sobol(n = star.centers, dim = length(params))
  } else if(method == "R") {
    mat <- replicate(length(params), stats::runif(star.centers))
  } else {
    stop("method should be either QRN, R or LHS")
  }
  for(i in 1:nrow(mat)) {
    center[[i]] <- mat[i, ]
    sections[[i]] <- sapply(center[[i]], function(x) {
      all <- seq(x %% h, 1, h)
      non.zeros <- all[all!= 0]
    })
    B[[i]] <- sapply(1:ncol(mat), function(x)
      sections[[i]][, x][!sections[[i]][, x] %in% center[[i]][x]])
    A[[i]] <- matrix(center[[i]], nrow = nrow(B[[i]]),
                     ncol = length(center[[i]]), byrow = TRUE)
    X[[i]] <- rbind(A[[i]], B[[i]])
    for(j in 1:ncol(A[[i]])) {
      AB[[i]] <- A[[i]]
      AB[[i]][, j] <- B[[i]][, j]
      X[[i]] <- rbind(X[[i]], AB[[i]])
    }
    AB[[i]] <- X[[i]][(2 * nrow(B[[i]]) + 1):nrow(X[[i]]), ]
    out[[i]] <- rbind(unname(center[[i]]), AB[[i]])
  }
  output <- do.call(rbind, out)
  output[output == 1] <- 0.999
  return(output)
}

# CREATE VARS FUNCTION -----------------------------------------------------------

# Function to cut by size
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, size)
}

# VARS-TO algorithm
vars_ti <- function(Y, star.centers, params, h, method = "all.step") {
  n.cross.points <- length(params) * ((1 / h) - 1) + 1
  index.centers <- seq(1, length(Y), n.cross.points)
  mat <- matrix(Y[-index.centers], ncol = star.centers)
  indices <- CutBySize(nrow(mat), nb = length(params))
  out <- list()
  for(i in 1:nrow(indices)) {
    out[[i]] <- mat[indices[i, "lower"]:indices[i, "upper"], ]
  }
  if(method == "one.step") {
    d <- lapply(1:length(params), function(x)
      lapply(1:ncol(out[[x]]), function(j) {
        da <- c(out[[x]][, j][1],
                rep(out[[x]][, j][-c(1, length(out[[x]][, j]))], each = 2),
                out[[x]][, j][length(out[[x]][, j])])
      }))
  } else if(method == "all.step") {
    d <- lapply(1:length(params), function(x)
      lapply(1:ncol(out[[x]]), function(j) {
        da <- c(combn(out[[x]][, j], 2))
      }))
  } else {
    stop("method should be either one.step or all.step")
  }
  out <- lapply(d, function(x)
    lapply(x, function(y) matrix(y, nrow = length(y) / 2, byrow = TRUE)))
  variogr <- unlist(lapply(out, function(x) lapply(x, function(y)
    mean(0.5 * (y[, 1] - y[, 2]) ^ 2))) %>%
      lapply(., function(x) do.call(rbind, x)) %>%
      lapply(., mean))
  covariogr <- unlist(lapply(out, function(x)
    lapply(x, function(y) cov(y[, 1], y[, 2]))) %>%
      lapply(., function(x) Rfast::colmeans(do.call(rbind, x))))
  VY <- var(Y[index.centers])
  Ti <- (variogr + covariogr) / VY
  output <- data.table::data.table(Ti)
  output[, `:=`(parameters = params)]
  return(output)
}
```

```r
# DEFINE JANSEN TOTAL-ORDER INDEX ------------------------------------------------

jansen_ti <- function(d, N, params) {
  m <- matrix(d, nrow = N)
  k <- length(params)
  Y_A <- m[, 1]
  Y_AB <- m[, -1]
  f0 <- (1 / length(Y_A)) * sum(Y_A)
  VY <- 1 / length(Y_A) * sum((Y_A - f0) ^ 2)
  Ti <- (1 / (2 * N) * Rfast::colsums((Y_A - Y_AB) ^ 2)) / VY
  output <- data.table(Ti)
  output[, `:=`(parameters = paste("X", 1:k, sep = ""))]
  return(output)
}

# DEFINE SAVAGE SCORES -----------------------------------------------------------
savage_scores <- function(x) {
  true.ranks <- rank(-x)
  p <- sort(1 / true.ranks)
  mat <- matrix(rep(p, length(p)), nrow = length(p), byrow = TRUE)
  mat[upper.tri(mat)] <- 0
  out <- sort(rowSums(mat), decreasing = TRUE)[true.ranks]
  return(out)
}

# DEFINE FUNCTION FOR RANDOM DISTRIBUTIONS ---------------------------------------

sample_distributions <- list(
  "uniform" = function(x) x,
  "normal" = function(x) qnorm(x, 0.5, 0.2),
  "beta" = function(x) qbeta(x, 8, 2),
  "beta2" = function(x) qbeta(x, 2, 8),
  "beta3" = function(x) qbeta(x, 2, 0.5),
  "beta4" = function(x) qbeta(x, 0.5, 2),
  "logitnormal" = function(x) qlogitnorm(x, 0, 3.16)
  # Logit-normal, Bates too?
)
```

```r
random_distributions <- function(X, phi) {
  names_ff <- names(sample_distributions)
  if(!phi == length(names_ff) + 1) {
    out <- apply(X, 2, function(x)
      sample_distributions[[names_ff[phi]]](x))
  } else {
    temp <- sample(names_ff, ncol(X), replace = TRUE)
    out <- sapply(seq_along(temp), function(x) sample_distributions[[temp[x]]](X[, x]))
  }
  return(out)
}
```

```r
# PLOT RANDOM DISTRIBUTIONS -----------------------------------------------------

names_ff <- names(sample_distributions)
prove <- randtoolbox::sobol(n = 1000, dim = length(names_ff))

out <- data.table(sapply(seq_along(names_ff), function(x)
  sample_distributions[[names_ff[x]]](prove[, x])))

b <- data.table::melt(out) %>%
  ggplot(., aes(value, group = variable, colour = variable)) +
  geom_density() +
  scale_color_discrete(labels = c("$\\mathcal{U}\\sim(0, 1)$",
                                  "$\\mathcal{N}\\sim(0.5, 0.2)$",
                                  "$Beta\\sim(8, 2)$",
                                  "$Beta\\sim(2, 8)$",
                                  "$Beta\\sim(2, 0.5)$",
                                  "$Beta\\sim(0.5, 2)$",
                                  "$Logitnormal\\sim(0, 3.16)$"),
                       name = "Distribution") +
  labs(x = expression(italic(x)),
       y = "Density") +
  theme_AP()
```

```
## Warning in melt.data.table(out): id.vars and measure.vars are internally
## guessed when both are 'NULL'. All non-numeric/integer/logical type columns are
## considered id.vars, which in this case are columns []. Consider providing at
## least one of 'id' or 'measure' vars in future.
```

```r
b
```

```r
# DEFINE METAFUNCTION -----------------------------------------------------------

function_list <- list(
  Linear = function(x) x,
  Quadratic = function(x) x ^ 2,
  Cubic = function(x) x ^ 3,
  Exponential = function(x) exp(1) ^ x / (exp(1) - 1),
```

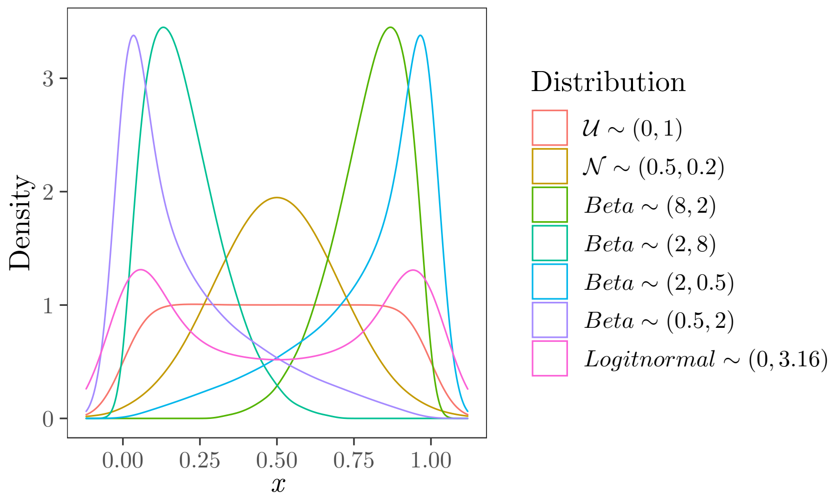


Figure 3: Distributions used in the metafunction.

```
  Periodic = function(x) sin(2 * pi * x) / 2,
  Discontinuous = function(x) ifelse(x > 0.5, 1, 0),
  Non.monotonic = function(x) 4 * (x - 0.5) ^ 2,
  Inverse = function(x) (10 - 1 / 1.1) ^ -1 * (x + 0.1) ^ - 1,
  No.effect = function(x) x * 0,
  Trigonometric = function(x) cos(x)
)
```

```
# PLOT METAFUNCTION --------------------------------------------------------------

a <- ggplot(data.frame(x = runif(100)), aes(x)) +
  map(1:length(function_list), function(nn) {
    stat_function(fun = function_list[[nn]],
                  geom = "line",
                  aes_(color = factor(names(function_list[nn])),
                       linetype = factor(names(function_list[nn]))))
  }) +
  labs(color= "Function", linetype = "Function",
       x = expression(italic(x)),
       y = expression(italic(y))) +
  scale_color_discrete(labels = c("$f(x) = x ^ 3$",
                                  "$f(x) = 1~\\mbox{if}(x > 0.5), 0~\\mbox{otherwise}$",
                                  "$f(x) = e ^ x / (e - 1)$",
                                  "$f(x) = (10 - 1 / 1.1) ^ {-1} (x + 0.1) ^ {-1}$",
                                  "$f(x) = x$",
                                  "$f(x) = 0$",
                                  "$f(x) = 4(x - 0.5) ^  2$",
                                  "$f(x) = \\sin (2 \\pi x) / 2$",
                                  "$f(x) = x ^ 2$",
                                  "$f(x) = \\cos(x)$")) +
  scale_linetype_discrete(labels = c("$f(x) = x ^ 3$",
                                     "$f(x) = 1~\\mbox{if}(x > 0.5), 0~\\mbox{otherwise}$",
                                     "$f(x) = e ^ x / (e - 1)$",
                                     "$f(x) = (10 - 1 / 1.1) ^ {-1} (x + 0.1) ^ {-1}$",
                                     "$f(x) = x$",
                                     "$f(x) = 0$",
                                     "$f(x) = 4(x - 0.5) ^  2$",
                                     "$f(x) = \\sin (2 \\pi x) / 2$",
                                     "$f(x) = x ^ 2$",
                                     "$f(x) = \\cos(x)$")) +
  theme_AP() +
  theme(legend.position = "right")

a
```

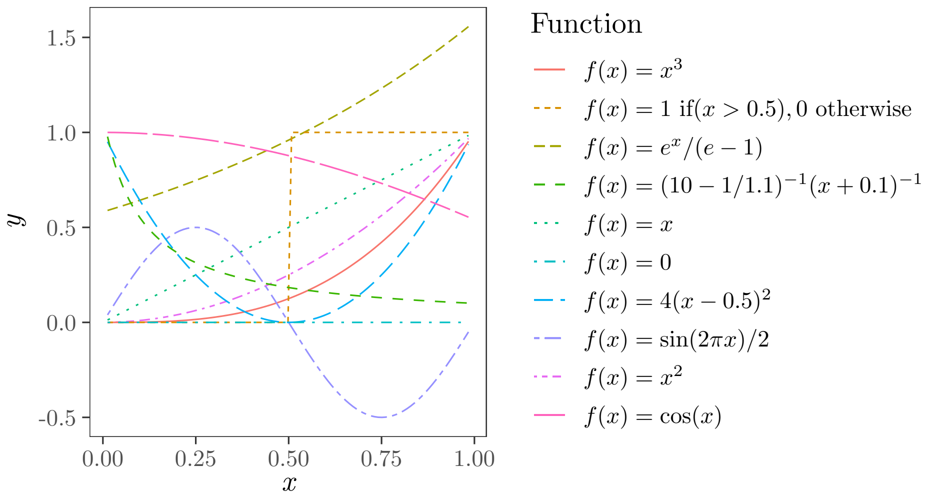


Figure 4: Functions used in the metafunction of Becker (2020).

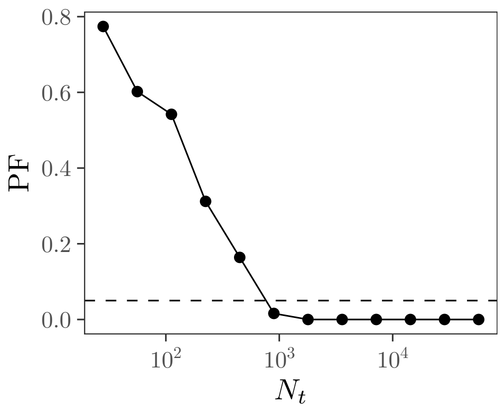
3.1 The case of the six-dimensional response surface model

```
# DEFINE SIX-DIMENSIONAL MODEL ------------------------------------------------

six_dimension <- function(X) {
  X1 <- - sin(pi * X[, 1]) - 0.3 * sin(3.33 * pi * X[, 1])
  X2 <- -0.76 * sin(pi * (X[, 2] - 0.2)) - 0.315
  X3 <- -0.12 * sin(1.05 * pi * (X[, 3] - 0.2)) -
    0.02 * sin(95.24 * pi * X[, 3]) - 0.96
  X4 <- -0.12 * sin(1.05 * pi * (X[, 4] - 0.2)) - 0.96
  X5 <- -0.05 * sin(pi * (X[, 5] - 0.2)) - 1.02
  X6 <- -1.08
  out <- X1 + X2 + X3 + X4 + X5 + X6
  return(out)
}

# DEFINE THE MODEL TO COMPUTE THE PROBABILITY OF FAILURE OF SOBOL' SI -----------
```

```r
# Define matrices: create function
sobol_fixing <- function(mat, fixed.point = 0.2) {
  X <- mat
  for(j in 1:ncol(mat)) {
    AB <- mat
    AB[, -j] <- fixed.point
    X <- rbind(X, AB)
  }
  return(X)
}

# Function to cut by size
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, size)
}

# Probability of failure function
PF_function <- function(replicas, N) {
  params <- paste("X", 1:6, sep = "")
  mat <- randtoolbox::sobol(n = N * replicas, dim = length(params))
  # Get replicas
  indices <- CutBySize(nrow(mat), block.size = N)
  out <- list()
  for(i in 1:nrow(indices)) {
    out[[i]] <- mat[indices[i, "lower"]:indices[i, "upper"], ]
  }
  mat.replicas <- lapply(out, sobol_fixing)
  Y <- lapply(mat.replicas, six_dimension)
  varY <- lapply(Y, function(x) var(x[1:N]))
  ind <- lapply(Y, function(x) matrix(x[-c(1:N)], ncol = ncol(mat)))
  final <- lapply(1:replicas, function(x)
    (colmeans(ind[[x]] ^ 2) - colmeans(ind[[x]]) ^ 2) / varY[[x]])
  true.ranks <- c(2, 1, 3, 4, 5, 6)
  estimated.ranks <- lapply(final, function(x) rank(-x))
  vec.output <- unlist(lapply(estimated.ranks,
                              function(x) identical(x, true.ranks)))
  # Probability of failure
  PF <- 1 - (sum(vec.output) / length(vec.output))
  # Number of model runs
  Nt <- nrow(mat.replicas[[1]])
  output <- data.table(PF, Nt)
  return(output)
}
```

```
# RUN THE SIMULATIONS -------------------------------------------------------------

x <- seq(2, 13)
PF.output <- parallel::mclapply(x, function(y)
  PF_function(replicas = 500, N = 2 ^ y),
  mc.cores = parallel::detectCores() * 0.75)
```

```
# PLOT THE RESULTS --------------------------------------------------------------

a <- rbindlist(PF.output) %>%
  ggplot(., aes(Nt, PF)) +
  geom_line() +
  geom_point() +
  geom_hline(yintercept = 0.05, lty = 2) +
  scale_x_log10(breaks = scales::trans_breaks("log10", function(x) 10^x),
                labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  labs(x = "$N_t$",
       y = "PF") +
  theme_AP()

a
```



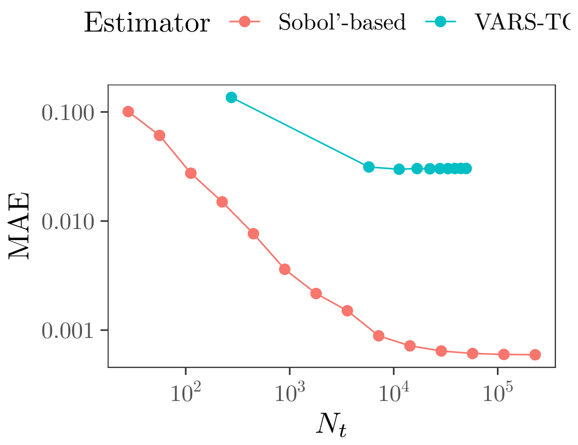
3.1.1 Computation of the MAE

```
# MAE ON THE SIX-DIMENSIONAL MODEL -----------------------------------------------------------

# MAE FOR THE SI ESTIMATOR -------------------------------

# Compute the analytical variance on a large sample size
N <- 2 ^ 24
params <- paste("X", 1:6, sep = "")
A <- randtoolbox::sobol(n = N, dim = length(params))
Y <- six_dimension(A)
VY <- var(Y)
v_analytical <- c(0.0972, 0.136, 0.00358, 0.00301, 0.000587, 0)
```

```r
analytical <- v_analytical / VY

# Set the number of replicas
replicas <- 50

# Create function to calculate Si according to Equation 18
# Probability of failure function
compute_si <- function(replicas, N) {
  params <- paste("X", 1:6, sep = "")
  mat <- randtoolbox::sobol(n = N * replicas, dim = length(params))
  # Get replicas
  indices <- CutBySize(nrow(mat), block.size = N)
  out <- list()
  for(i in 1:nrow(indices)) {
    out[[i]] <- mat[indices[i, "lower"]:indices[i, "upper"], ]
  }
  mat.replicas <- lapply(out, sobol_fixing)
  Y <- lapply(mat.replicas, six_dimension)
  varY <- lapply(Y, function(x) var(x[1:N]))
  ind <- lapply(Y, function(x) matrix(x[-c(1:N)], ncol = ncol(mat)))
  final <- lapply(1:replicas, function(x)
    (colmeans(ind[[x]] ^ 2) - colmeans(ind[[x]]) ^ 2) / varY[[x]])
  return(final)
}

# Define the sample sizes and compute
x <- seq(2, 15)
si.output <- parallel::mclapply(x, function(y)
  compute_si(replicas = replicas, N = 2 ^ y),
  mc.cores = parallel::detectCores() * 0.75)

# Rearrange
out <- lapply(si.output, function(x) lapply(x, function(y) data.table(y))) %>%
  lapply(., function(x) lapply(x, function(y) cbind(y, analytical))) %>%
  lapply(., function(x) rbindlist(x, idcol = "Replica"))

names(out) <- 2 ^ x

si_analytical <- rbindlist(out, idcol = "N") %>%
  .[, .(MAE = mean(abs(analytical - y))), N] %>%
  .[, N:= as.numeric(N)] %>%
  .[, C:= N * (length(params) + 1)] %>%
  .[, estimator:= "Sobol'-based"]

# MAE FOR VARS-TO -------------------------------------

# Define h
```

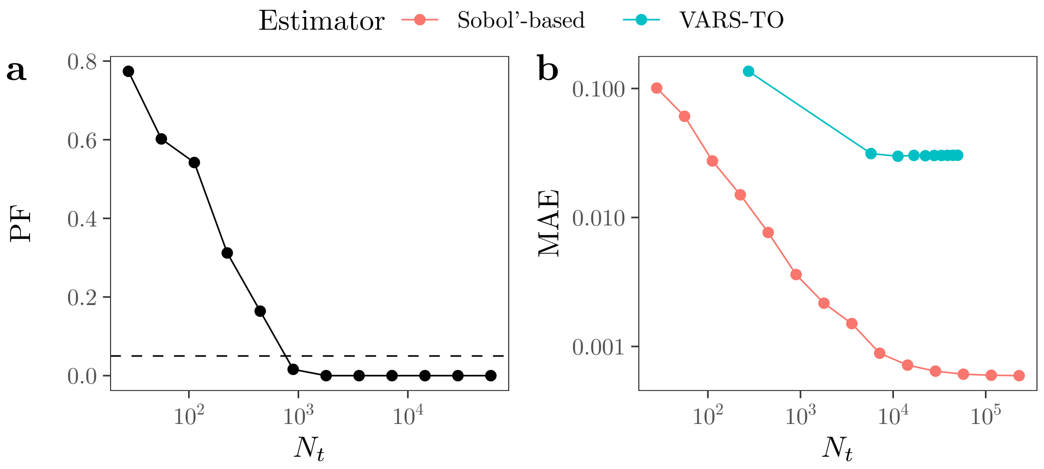
```r
h <- 0.1

# Function to replicate star vars matrices
vars_matrices2 <- function(mat, params, h, method = "QRN") {
  out <- center <- sections <- A <- B <- AB <- X <- out <- list()
  for(i in 1:nrow(mat)) {
    center[[i]] <- mat[i, ]
    sections[[i]] <- sapply(center[[i]], function(x) {
      all <- seq(x %% h, 1, h)
      non.zeros <- all[all!= 0]
    })
    B[[i]] <- sapply(1:ncol(mat), function(x)
      sections[[i]][, x][!sections[[i]][, x] %in% center[[i]][x]])
    A[[i]] <- matrix(center[[i]], nrow = nrow(B[[i]]),
                     ncol = length(center[[i]]), byrow = TRUE)
    X[[i]] <- rbind(A[[i]], B[[i]])
    for(j in 1:ncol(A[[i]])) {
      AB[[i]] <- A[[i]]
      AB[[i]][, j] <- B[[i]][, j]
      X[[i]] <- rbind(X[[i]], AB[[i]])
    }
    AB[[i]] <- X[[i]][(2 * nrow(B[[i]]) + 1):nrow(X[[i]]), ]
    out[[i]] <- rbind(unname(center[[i]]), AB[[i]])
  }
  output <- do.call(rbind, out)
  output <- replace(output, which(output == 1), 0.999)
  return(output)
}

# Function to compute mae
mae_vars <- function(star.centers, h, params, replicas) {
  mat <- randtoolbox::sobol(n = star.centers * replicas, dim = length(params))
  # Get replicas
  indices <- CutBySize(nrow(mat), block.size = star.centers)
  out <- list()
  for(i in 1:nrow(indices)) {
    out[[i]] <- mat[indices[i, "lower"]:indices[i, "upper"], ]
  }
  out <- lapply(out, function(x) vars_matrices2(mat = x,  params = params, h = h))
  Y <- lapply(out, six_dimension)
  output <- lapply(Y, function(y) vars_ti(Y = y, star.centers = star.centers, params = params,
  return(output)
}

# Define the sample size and compute
x <- seq(5, 1000, 100)
vars.dt <- parallel::mclapply(x, function(y)
  mae_vars(star.centers = y, h = h, params = params, replicas = replicas),
  mc.cores = parallel::detectCores() * 0.75)

# Rearrange
out <- lapply(vars.dt, function(x) lapply(x, function(y) data.table(y))) %>%
  lapply(., function(x) lapply(x, function(y) cbind(y, analytical))) %>%
  lapply(., function(x) rbindlist(x, idcol = "Replica"))

names(out) <- x

vars_analytical <- rbindlist(out, idcol = "Star.centers") %>%
  .[, .(MAE = mean(abs(analytical - Ti))), Star.centers] %>%
  .[, Star.centers:= as.numeric(Star.centers)] %>%
  .[, C:= Star.centers * (length(params) * ((1 / 0.1) -1) + 1)] %>%
  .[, estimator:= "VARS-TO"]
```

```
# PLOT MAE OF SOBOL-BASED AND VARS-TO -----------------------------------------
# Plot
b <- rbind(si_analytical[, .(C, estimator, MAE)],
      vars_analytical[, .(C, estimator, MAE)]) %>%
  ggplot(., aes(C, MAE, group = estimator, color = estimator)) +
  geom_point() +
  geom_line() +
  scale_x_log10(breaks = scales::trans_breaks("log10", function(x) 10^x),
                labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  scale_y_log10() +
  scale_color_discrete(name = "Estimator") +
  labs(x = "$N_t$", y = "MAE") +
  theme_AP() +
  theme(legend.position = "top")
b
```



```
# MERGE FIGURES ----------------------------------------------------------------

legend <- get_legend(b + theme(legend.position = "top"))
```

```r
bottom <- plot_grid(a, b + theme(legend.position = "none"),
                    ncol = 2, labels = "auto", align = "hv")

plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.1, 1))
```



3.2 The case of the HYMOD and MESH models

In this section we present the workflow of the analysis. The following code snippets are based on Puy et al. (2020).

3.2.1 The sample matrix

In this section we create the sample matrix using Sobol' quasi random numbers (Sobol' 1967, @Sobol1976). We then transform them to their appropriate distributions (see Table 1 in Puy, Lo Piano, and Saltelli (2020)), and define three columns with the total number of model runs for VARS-TO and the Jansen estimator (which is constrained by the total number of model runs for VARS-TO).

```r
# DEFINE SETTINGS -------------------------------------------------------------------

N <- 2 ^ 12 # Sample size of sample matrix
R <- 500 # Number of bootstrap replicas
n_cores <- ceiling(detectCores() * 0.5)
order <- "first"
params <- c("N.stars", "h", "k", "k_2", "k_3", "epsilon", "delta", "phi", "tau")
N.high <- 2 ^ 12 # Maximum sample size of the large sample matrix

# CREATE SAMPLE MATRIX ----------------------------------------------------------

mat <- sobol_matrices(N = N, params = params, order = order)

# TRANSFORM MATRIX --------------------------------------------------------------

mat[, 1] <- round(qunif(mat[, 1], 3, 50), 0) # N.stars
mat[, 2] <- round(qunif(mat[, 2], 1, 4), 0) # h
```

```r
mat[, 3] <- round(qunif(mat[, 3], 3, 50)) # k
mat[, 4] <- round(qunif(mat[, 4], 0.5, 1), 1) # k_2
mat[, 5] <- round(qunif(mat[, 5], 0.3, 1), 1) # k_3
mat[, 6] <- round(qunif(mat[, 6], 1, 200), 0) # epsilon
mat[, 7] <- round(qunif(mat[, 7], 1, 3), 0) # delta
mat[, 8] <- round(qunif(mat[, 8], 1, 8), 0) # phi
mat[, 9] <- floor(mat[, 9] * (2 - 1 + 1)) + 1 # tau

# DEFINE TOTAL NUMBER OF RUNS ---------------------------------------------------

# Correct h
mat[, "h"] <- ifelse(mat[, "h"] == 1, 0.01,
                     ifelse(mat[, "h"] == 2, 0.05,
                            ifelse(mat[, "h"] == 3, 0.1, 0.2)))

# For vars
Nt.vars <- round(apply(mat, 1, function(x)
  x["N.stars"] * (x["k"] * ((1 / x["h"]) - 1) + 1)), 0)

# For jansen
N.jansen <- round(apply(cbind(mat, Nt.vars), 1, function(x)
  x["Nt.vars"] / (x["k"] + 1))) %>%
  ifelse(. == 1, 2, .) # Transform N = 1 to N = 2 for jansen

Nt.jansen <- apply(cbind(mat, Nt.vars, N.jansen), 1, function(x)
  x["N.jansen"] * (x["k"] + 1))

# FINAL MATRIX -------------------------------------------------------------------

mat <- cbind(mat, Nt.vars, N.jansen, Nt.jansen)

# Check min and max total number of runs
sapply(c(min, max), function(x) x(mat[, "Nt.vars"]))
```

```
## [1]     63 242599
```

```r
# SHOW SAMPLE MATRIX -------------------------------------------------------------

head(mat)
```

```
##      N.stars    h  k k_2 k_3 epsilon delta phi tau Nt.vars N.jansen Nt.jansen
## [1,]      26 0.05 26 0.8 0.6     100     2   4   2   12870      477     12879
## [2,]      38 0.05 38 0.6 0.8      51     2   3   1   27474      704     27456
## [3,]      15 0.10 15 0.9 0.5     150     2   6   2    2040      128      2048
## [4,]      21 0.05 32 0.6 0.9     175     1   5   1   12789      388     12804
## [5,]      44 0.20  9 0.8 0.6      76     2   2   2    1628      163      1630
## [6,]      32 0.01 21 0.7 0.4     125     3   7   1   66560     3025     66550
```

3.2.2 The model

The model runs rowwise: for $v = 1, 2, ..., 2^{12}$ rows, it constructs three sample matrices using either random numbers ($\tau_v = 1$) or Sobol' quasi-random numbers ($\tau_v = 2$):

1. A sample matrix for Jansen (column dimension of k_v, row dimension of $N_{t_v} = N_v(k_v + 1)$.
2. A sample matrix for VARS-TO (column dimension of k_v, row dimension of $N_{t_v} = N_{star_v}\left[k_v((\frac{1}{\Delta h_v}) - 1) + 1\right]$).
3. A large sample matrix to compute the "true" indices (column dimension of k_v, row dimension of $N_{t_v} = 2^{12}(k_v + 1)$.

Then it transforms the model inputs into the probability distributions defined by ϕ_v. The test function for benchmarking is applied to all three sample matrices simultaneously, with its functional form, degree of active second and third-order effects defined by ϵ_v, k_2 and k_3 respectively.

Finally, it computes the VARS-TO indices, the Jansen indices and the “true” indices. The performance of VARS-TO and Jansen is then asssessed by checking how good the computed indices correlate with the true indices ($\delta = 1$), with the true ranks ($\delta = 2$), and with the most important ranks only ($\delta = 3$).

```
# DEFINE MODEL ---------------------------------------------------------------------

model_ti <- function(N.stars, h, k, k_2, k_3, epsilon, delta, tau, phi, N.jansen, N.high) {
  if(tau == 1) {
    method <- "R"
  } else if(tau == 2) {
    method <- "QRN"
  }
  # Create sample matrices
  set.seed(epsilon)
  vars.matrix <- vars_matrices(star.centers = N.stars, params = paste("X", 1:k, sep = ""), h =
                               method = method)
  set.seed(epsilon)
  jansen.matrix <- sobol_matrices(matrices = c("A", "AB"), N = N.jansen,
                                  params = paste("X", 1:k, sep = ""),
                                  method = method)
  set.seed(epsilon)
  large.matrix <- sobol_matrices(matrices = c("A", "AB"), N = N.high,
                                 params = paste("X", 1:k, sep = ""),
                                 method = method)
  # Compute metafunciton
  set.seed(epsilon)
  output <- sensobol::metafunction(data = random_distributions(X = rbind(jansen.matrix,
                                                                         vars.matrix,
                                                                         large.matrix),
                                                              phi = phi),
                                   k_2 = k_2, k_3 = k_3, epsilon = epsilon)
  # Compute sobol' indices on a large sample size
  full.ind <- jansen_ti(d = tail(output, nrow(large.matrix)),
```

```r
                        N = N.high,
                        params = paste("X", 1:k, sep = ""))
  full.ind[, sample.size:= "N"]
  # Define indices of Y for VARS
  lg.jansen <- 1:(N.jansen * (k + 1))
  Nt.vars <- N.stars * (k * ((1 / h) - 1) + 1)
  lg.vars <- (max(lg.jansen) + 1): (max(lg.jansen) + Nt.vars)
  # Compute VARS
  ind.vars <- vars_ti(output[lg.vars], star.centers = N.stars,
                      params = paste("X", 1:k, sep = ""), h = h) %>%
    .[, sample.size:= "n"]
  full.vars <- rbind(ind.vars, full.ind)[, estimator:= "VARS-TO"]
  # Compute Jansen
  ind.jansen <- jansen_ti(d = output[lg.jansen], N = N.jansen,
                          params = paste("X", 1:k, sep = ""))
  ind.jansen[, sample.size:= "n"]
  full.jansen <- rbind(ind.jansen, full.ind)[, estimator:= "Jansen"]
  out <- rbind(full.vars, full.jansen)
  out.wide <- dcast(out, estimator + parameters ~ sample.size, value.var = "Ti")
  # Replace NaN
  for (i in seq_along(out.wide))
    set(out.wide, i=which(is.nan(out.wide[[i]])), j = i, value = 0)
  # Replace Inf
  for (i in seq_along(out.wide))
    set(out.wide, i=which(is.infinite(out.wide[[i]])), j = i, value = 0)
  # Replcace Na
  for (i in seq_along(out.wide))
    set(out.wide, i=which(is.na(out.wide[[i]])), j = i, value = 0)
  # CHECK DELTA
  if(delta == 1) { # Regular Pear
    final <- out.wide[, .(correlation = cor(N, n)), estimator]
  } else if(delta == 2) { # kendall tau
    final <- out.wide[, .(correlation = pcaPP::cor.fk(N, n)), estimator]
  } else { # Savage ranks
    final <- out.wide[, lapply(.SD, savage_scores), .SDcols = c("N", "n"), estimator][
      , .(correlation = cor(N, n)), estimator]
  }
  return(final)
}
```

3.2.3 Execution of the model

The model is executed in parallel. In my computer it took approximately 8 hours (see the computer specifications below).

```r
# RUN MODEL -----------------------------------------------------------------------

# Define parallel computing
```

```
cl <- makeCluster(n_cores)
registerDoParallel(cl)

# Compute
Y.ti <- foreach(i=1:nrow(mat),
                .packages = c("sensobol", "data.table", "dplyr",
                              "pcaPP", "logitnorm")) %dopar%
  {
    model_ti(N.stars = mat[[i, "N.stars"]],
             h = mat[[i, "h"]],
             k = mat[[i, "k"]],
             k_2 = mat[[i, "k_2"]],
             k_3 = mat[[i, "k_3"]],
             epsilon = mat[[i, "epsilon"]],
             delta = mat[[i, "delta"]],
             phi = mat[[i, "phi"]],
             tau = mat[[i, "tau"]],
             N.jansen = mat[[i, "N.jansen"]],
             N.high = N.high)
  }

# Stop parallel cluster
stopCluster(cl)
```

3.2.4 Arrange the results

We arrange the results as to allow further inspection. We also show the simulations that yielded `Na` or `NaN` (below 0.2% for both Jansen and VARS-TO), and substitute them for zeroes. We finally export the results to `.csv`.

```
# ARRANGE OUTPUT ------------------------------------------------------------------

out_cor <- rbindlist(Y.ti, idcol = "row")

mt.dt <- data.table(mat) %>%
  .[, row:= 1:.N]

full_output <- merge(mt.dt, out_cor) %>%
  .[, Nt:= ifelse(estimator == "VARS-TO", Nt.vars, Nt.jansen)]

# Show rows with NA or NaN
full_output[is.na(correlation), ]
```

```
## Empty data.table (0 rows and 16 cols): row,N.stars,h,k,k_2,k_3...
```

```
# Compute proportion of rows with Na or NaN
full_output[, sum(is.na(correlation)) / .N, estimator]
```

```
##    estimator V1
## 1:    Jansen  0
## 2:   VARS-TO  0
```

```
# Substitute NA by 0
full_output <- full_output[, correlation:= ifelse(is.na(correlation) == TRUE, 0, correlation)]

A <- full_output[,.SD[1:N], estimator][, ratio:= Nt / k]
```

```
# EXPORT RESULTS ---------------------------------------------------------------

fwrite(A, "A.csv")
fwrite(full_output, "full_output.csv")
```

3.2.5 Uncertainty analysis

In this section we compute some statistics on the model output and plot the results.

```
# UNCERTAINTY ANALYSIS ---------------------------------------------------------

# Compute median and quantiles
dt_median <- A[, .(median = median(correlation),
                   low.ci = quantile(correlation, 0.25),
                   high.ci = quantile(correlation, 0.75)), estimator]

A[, .(median = median(correlation),
      low.ci = quantile(correlation, 0.25),
      high.ci = quantile(correlation, 0.75)), estimator][order(median)]
```

```
##    estimator    median    low.ci   high.ci
## 1:   VARS-TO 0.9725532 0.9285714 0.9957326
## 2:    Jansen 0.9884170 0.9539868 0.9992740
```

```
# PLOT UNCERTAINTY -------------------------------------------------------------

# Histograms
unc <- ggplot(A, aes(correlation)) +
  geom_rect(data = dt_median,
            aes(xmin = low.ci,
                xmax = high.ci,
                ymin = -Inf,
                ymax = Inf),
            fill = "blue",
            color = "white",
            alpha = 0.1,
            inherit.aes = FALSE) +
  geom_histogram() +
  geom_vline(data = dt_median, aes(xintercept = median),
             lty = 2,
             color = "red") +
  facet_wrap(~estimator,
              ncol = 4) +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = expression(italic(r)),
       y = "Counts") +
  theme_AP() +
  theme(strip.background = element_rect(fill = "white"))

unc
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

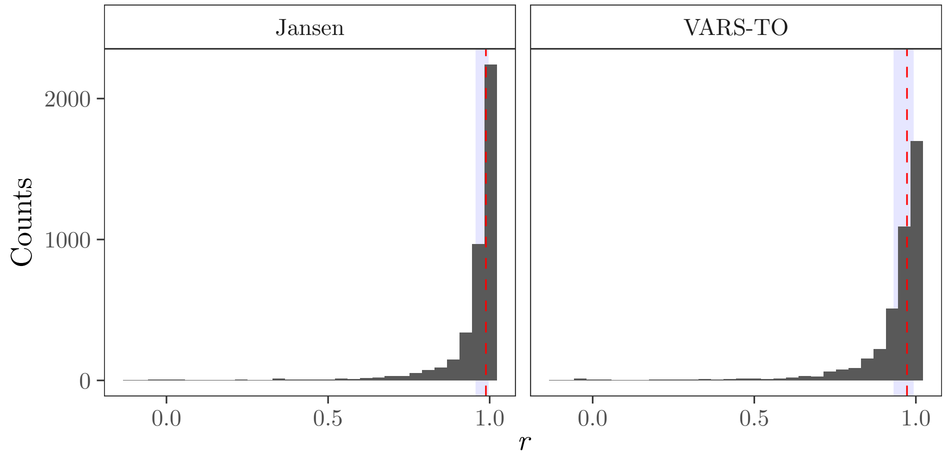


Figure 5: Empirical distribution of r for Jansen and VARS.

```
# SCATTERPLOT OF MODEL RESULTS ----------------------------------------------------

scat <- ggplot(A, aes(Nt, k, color = correlation)) +
  geom_point(size = 0.2) +
  scale_colour_gradientn(colours = c("black", "purple", "red", "orange", "lightgreen"),
                         name = expression(italic(r)),
                         breaks = c(0, 0.5, 1)) +
  scale_x_log10(breaks = scales::trans_breaks("log10", function(x) 10 ^ (2 * x)),
                labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = expression(italic(N[t])),
       y = expression(italic(k))) +
  facet_wrap(~estimator,
              ncol = 4) +
  theme_AP() +
  theme(legend.position = "top") +
  theme(strip.background = element_rect(fill = "white"))

scat
```

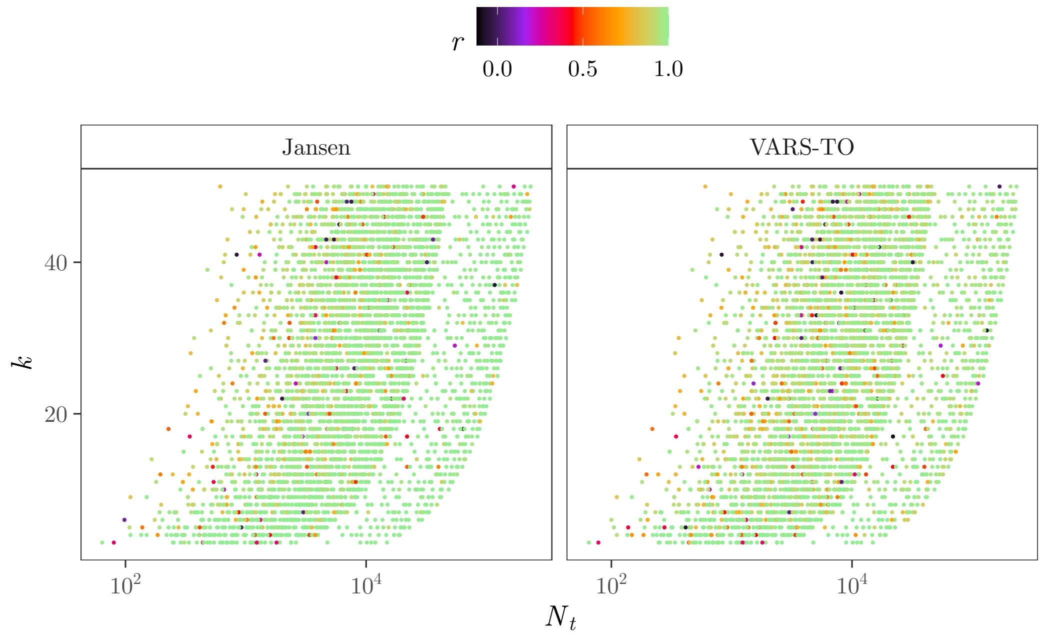


Figure 6: Scatterplots of the total number of model runs N_t against the function dimensionality k. The greener (darker) the colour, the better (worse) the performance.

```
# MERGE PLOTS --------------------------------------------------------------------

plot_grid(unc, scat, ncol = 1, labels = "auto", align = "hv",
          rel_heights = c(0.72, 1))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.
```

```
# PLOT R MEDIANS AS FUNCTION OF K AND NT/K ----------------------------------------

vv <- seq(5, 50, 5)

dt <- lapply(vv, function(x) A[k <= x, median(correlation), estimator])

names(dt) <- vv

# Plot r as a function of k
a1 <- rbindlist(dt, idcol = "k") %>%
  .[, k:= as.numeric(k)] %>%
  ggplot(., aes(k, V1, color = estimator)) +
  scale_color_manual(name = "Estimator",
                     values = wes_palette(n = 2, name = "Cavalcanti1")) +
  labs(x = expression(italic(k)),
       y = expression(median(italic(r)))) +
  geom_line() +
```

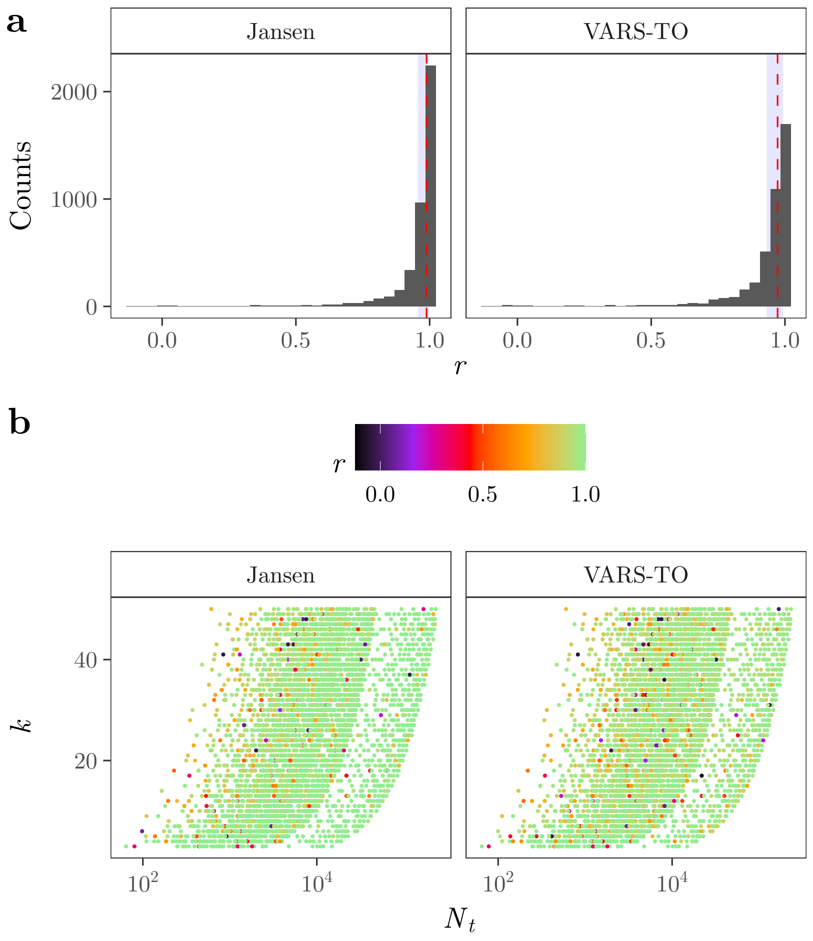


Figure 7: Uncertainty analysis conducted on 2^{12} simulations. a) Histograms of r. The vertical red, dashed line shows the median value. The transparent, blue rectangle frames the 0.25, 0.75 quantiles. b) Scatterplots showing the performance of the estimators as a function of the total number of model runs N_t and the model dimensionality k. Each simulation is a dot. The greener (darker) the colour, the better (worse) the performance. The white space between 10^3 and 10^4 in the x axis is caused by the uneven distribution selected for h (see Table~1 of the manuscript and the Supplementary Materials).

```
  theme_AP() +
  theme(legend.position = "none")

# Median Nt/k
dt.tmp <- A[, .(min = min(ratio), max = max(ratio))]

v <-  seq(0, ceiling(dt.tmp$max), 20)
a <- c(v[1], rep(v[-c(1, length(v))], each = 2), v[length(v)])
indices <- matrix(a, ncol = 2, byrow = TRUE)

out <- list()
for(i in 1:nrow(indices)) {
  out[[i]] <- A[ratio > indices[i, 1] & ratio < indices[i, 2]]
}

names(out) <- Rfast::rowMedians(indices)

# Plot r as a function of Nt/k
a2 <- lapply(out, function(x) x[, median(correlation, na.rm = TRUE), estimator]) %>%
  rbindlist(., idcol = "N") %>%
  .[, N:= as.numeric(N)] %>%
  ggplot(., aes(N, V1, group = estimator, color = estimator)) +
  geom_line() +
  labs(x = expression(italic(N[t]/k)),
       y = "") +
  scale_color_manual(name = "Estimator",
                     values = wes_palette(n = 2, name = "Cavalcanti1")) +
  scale_x_log10() +
  theme_AP() +
  theme(legend.position = "none")

# Plot median Nt/k for the number of simulations
a3 <- rbindlist(out, idcol = "samples")[, .N, .(estimator, samples)] %>%
  .[, samples:= as.numeric(samples)] %>%
  ggplot(., aes(samples, N, color = estimator, group = estimator)) +
  geom_line() +
  scale_y_log10() +
  scale_color_manual(name = "Estimator",
                    values = wes_palette(n = 2, name = "Cavalcanti1")) +
  labs(x = "median$(N_t / k)$",
       y = "Nº simulations") +
  theme_AP() +
  theme(legend.position = "none")


# Merge all plots
legend <- get_legend(a1 + theme(legend.position = "top"))
```

```r
sides <- plot_grid(a1, a2, ncol = 2, rel_widths = c(1, 1), align = "hv",
                   labels = "auto")
top.down <- plot_grid(sides, a3, ncol = 1, labels = c("", "c"), rel_heights = c(1, 0.8))

plot_grid(legend, top.down, rel_heights = c(0.1, 1), ncol = 1, align = "hv")
```

```
## Warning: Graphs cannot be vertically aligned unless the axis parameter is set.
## Placing graphs unaligned.

## Warning: Graphs cannot be horizontally aligned unless the axis parameter is set.
## Placing graphs unaligned.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.
```

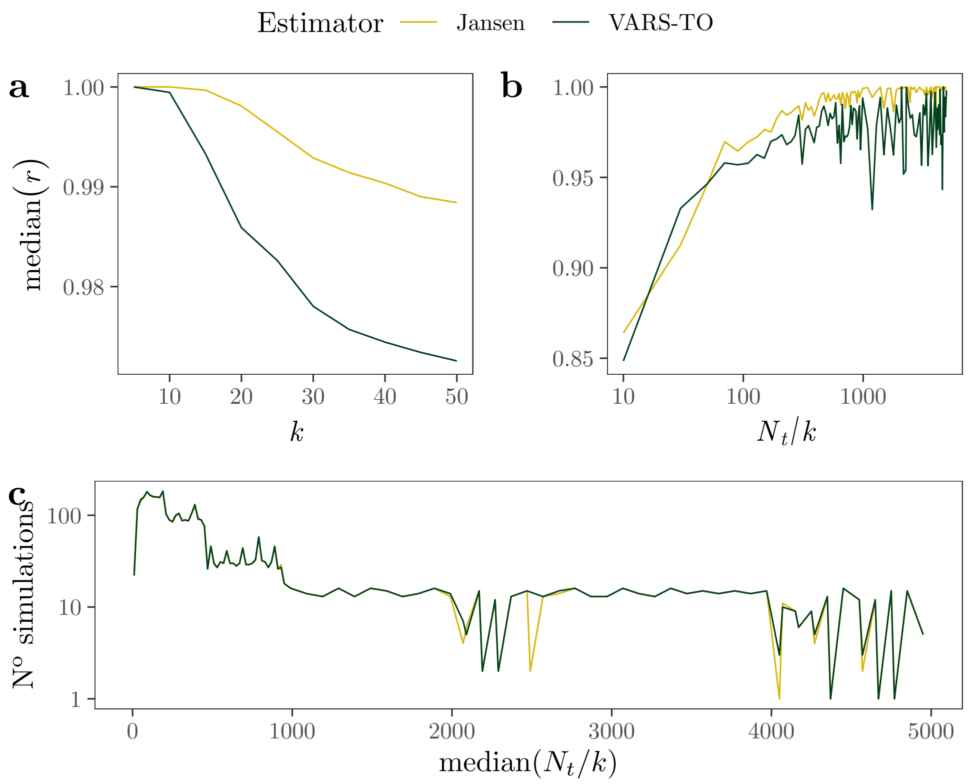
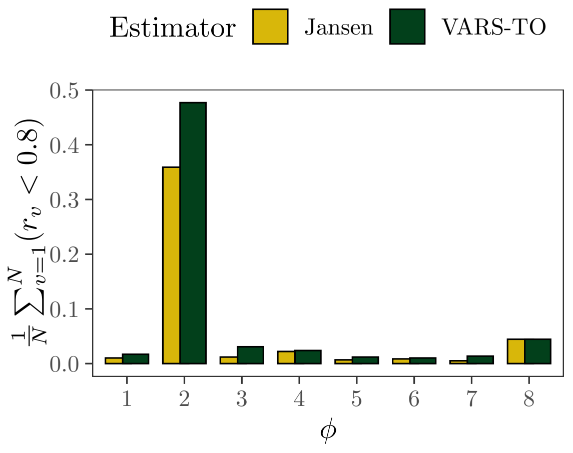


Figure 8: Comparison between the accuracy and efficiency of VARS-TO and Jansen (1999). a) Evolution of the median r value across different dimensions k. b) Evolution of the median r value across the different number of runs allocated to each model input (N_t/k).

```r
# PLOT NUMBER OF RUN  BELOW 0.8 FOR EACH LEVEL OF PHI ---------------------------

A[, sum(correlation < 0.8) / .N, .(estimator, phi)] %>%
  .[order(phi)] %>%
  ggplot(., aes(factor(phi), V1, fill = estimator)) +
  geom_bar(stat = "identity",
           position = position_dodge(0.6),
           color = "black") +
  scale_fill_manual(values = wes_palette(n = 2, name = "Cavalcanti1"),
                    name = "Estimator") +
  labs(x = "$\\phi$",
       y = "$\\frac{1}{N}\\sum_{v=1}^N(r_v<0.8)$") +
  theme_AP() +
  theme(legend.position = "top")
```



```r
# CHECK HOW MUCH NT DIFFERS BETWEEN VARS AND JANSEN ------------------------------

A <- A[, diff:= abs(Nt.jansen - Nt.vars)]

# Plot
ggplot(A, aes(row, diff)) +
  geom_line(size = 0.005) +
  labs(x = "Simulation",
       y = "$ | N_{t_{vars}} - N_{t_{jansen}} | $") +
  theme_AP()
```

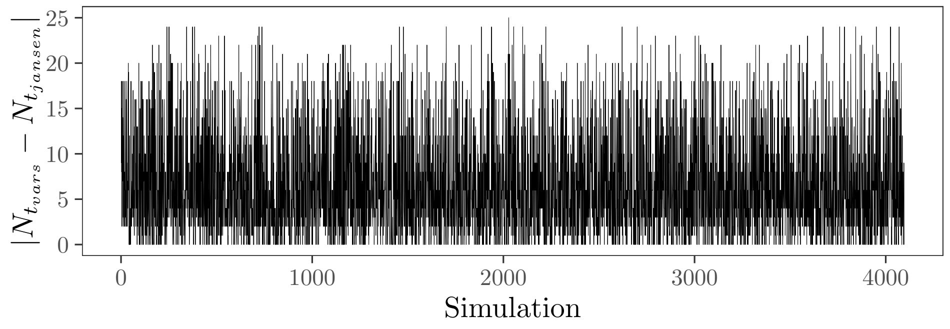


Figure 9: Line plot displaying the absolute difference between the total number of model runs allocated to Jansen and the total number of runs allocated to VARS-TO in each simulation.

```r
# CHECK HOW MUCH NT DIFFERS BETWEEN VARS AND JANSEN 2 ------------------------------

A %>%
  ggplot(., aes(diff)) +
  geom_histogram() +
  labs(y = "Counts",
       x = "$ | N_{t_{vars}} - N_{t_{jansen}} | $") +
  theme_AP()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

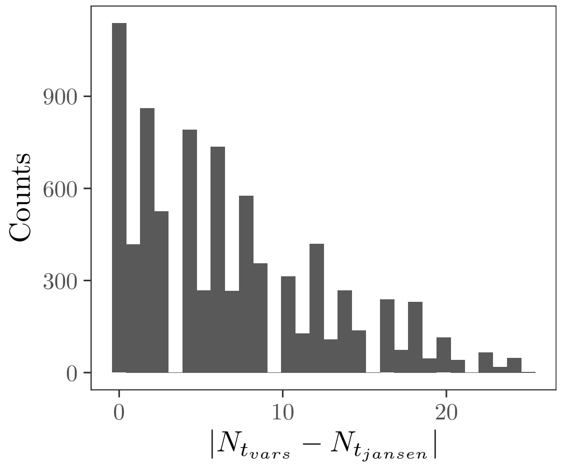


Figure 10: Histogram displaying the absolute difference between the total number of model runs allocated to Jansen and the total number of runs allocated to VARS-TO.

3.2.6 Sensitivity analysis

In this last section we check how sensitive is VARS-TO to uncertainty in the benchmark settings and to its own design parameters. We first plot the model inputs against the model output in scatterplots, and compute the Sobol' indices using the Jansen (1999) estimators.

```r
# SCATTERPLOTS -------------------------------------------------------------------

params_notikz <- c("N[stars]", "h", "k", "k[2]", "k[3]",
                   "epsilon", "delta", "phi", "tau")

# Scatterplot of model inputs against output
A[estimator == "VARS-TO"] %>%
  setnames(., params, params_notikz) %>%
  melt(., measure.vars = params_notikz) %>%
  ggplot(., aes(value, correlation)) +
  geom_point(size = 0.2, alpha = 0.05) +
  facet_wrap(~variable,
             scales = "free_x",
             labeller = label_parsed) +
  labs(y = expression(italic(r)),
       x = "") +
  theme_AP() +
  theme(strip.background = element_rect(fill = "white"))
```

```r
# SOBOL' INDICES -----------------------------------------------------------------

params_tikz <- c("$N_{stars}$", "$h$", "$k$", "$k_2$", "$k_3$", "$\\epsilon$",
                 "$\\delta$", "$\\phi$", "$\\tau$")

 # Compute Sobol' indices except for the cluster Nt,k
indices <- full_output[, sobol_indices(Y = correlation,
```

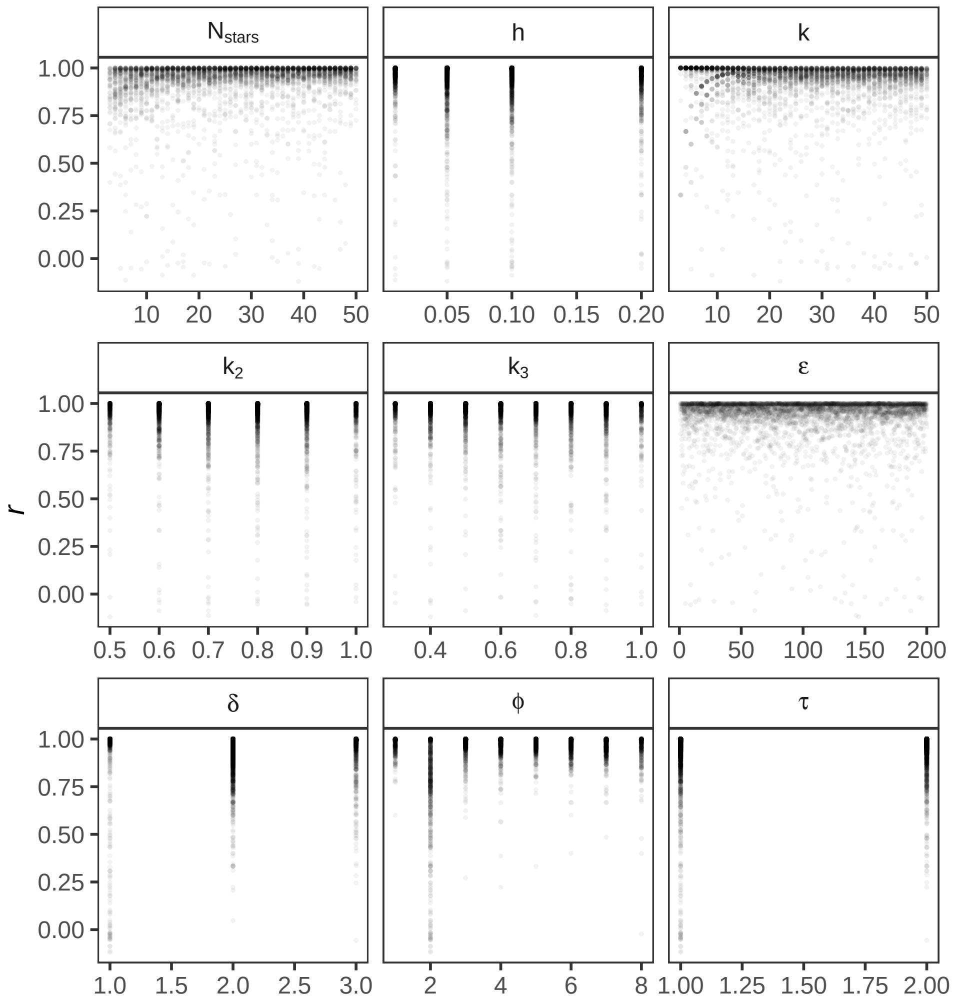


Figure 11: Scatterplots of model output r against the model inputs.

```
                                       N = N,
                                       params = params_tikz,
                                       first = "jansen",
                                       boot = TRUE,
                                       R = R,
                                       order = order),
                           estimator]
```

```
# PLOT SOBOL' INDICES -----------------------------------------------------------

indices[sensitivity == "Si" | sensitivity == "Ti"] %>%
  .[estimator == "VARS-TO"] %>%
  ggplot(., aes(parameters, original, fill = sensitivity)) +
  geom_bar(stat = "identity",
           position = position_dodge(0.6),
           color = "black") +
  geom_errorbar(aes(ymin = low.ci,
                    ymax = high.ci),
                position = position_dodge(0.6)) +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = "",
       y = "Sobol' index") +
  scale_fill_discrete(name = "Sobol' indices",
                      labels = c(expression(S[italic(i)]),
                                 expression(T[italic(i)]))) +
  theme_AP() +
  theme(legend.position = "top")
```

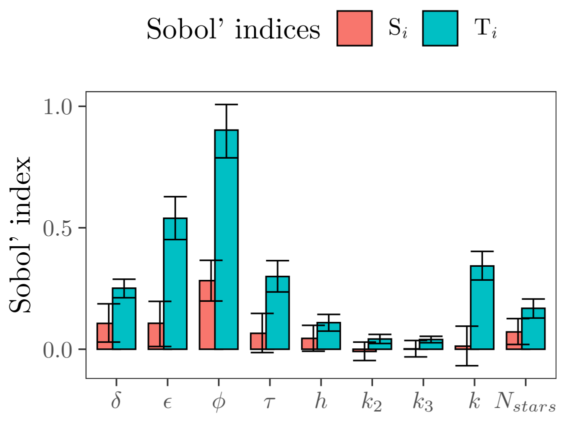


Figure 12: Sobol' indices.

```
# SUM OF FIRST-ORDER INDICES ------------------------------------------------------

indices[sensitivity == "Si", sum(original)]
```

```
## [1] 1.177513
```

```r
# EXPORT SOBOL' INDICES ---------------------------------------------------------

fwrite(indices, "indices.csv")
```

4 Session information

```
# SESSION INFORMATION ---------------------------------------------------------------

sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] grid      parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] checkpoint_0.4.9   gridExtra_2.3      xlsx_0.6.4.2       wesanderson_0.3.6
##  [5] Rfast_2.0.1        RcppZiggurat_0.1.6 Rcpp_1.0.5         benchmarkme_1.0.4
##  [9] logitnorm_0.8.37   cowplot_1.1.0      scales_1.1.1       pcaPP_1.9-73
## [13] doParallel_1.0.16  iterators_1.0.13   foreach_1.5.1      data.table_1.13.2
## [17] sensobol_0.3       forcats_0.5.0      stringr_1.4.0      dplyr_1.0.2
## [21] purrr_0.3.4        readr_1.3.1        tidyr_1.1.1        tibble_3.0.4
## [25] ggplot2_3.3.2      tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.2           jsonlite_1.7.1       modelr_0.1.8
##  [4] Rdpack_2.0           assertthat_0.2.1     xlsxjars_0.6.1
##  [7] blob_1.2.1           cellranger_1.1.0     yaml_2.2.1
## [10] remotes_2.2.0        pillar_1.4.6         backports_1.2.0
## [13] lattice_0.20-41      glue_1.4.2           digest_0.6.27
## [16] rbibutils_1.3        rvest_0.3.6          colorspace_1.4-1
## [19] htmltools_0.5.0      Matrix_1.2-18        pkgconfig_2.0.3
## [22] broom_0.7.0          haven_2.3.1          mvtnorm_1.1-1
## [25] generics_0.0.2       ellipsis_0.3.1       withr_2.3.0
## [28] cli_2.1.0            magrittr_1.5         crayon_1.3.4
## [31] readxl_1.3.1         evaluate_0.14        fs_1.5.0
## [34] fansi_0.4.1          xml2_1.3.2           tools_4.0.3
## [37] hms_0.5.3            gbRd_0.4-11          lifecycle_0.2.0
## [40] munsell_0.5.0        reprex_0.3.0         compiler_4.0.3
## [43] rlang_0.4.8          rstudioapi_0.11      rmarkdown_2.5.2
## [46] gtable_0.3.0         codetools_0.2-16     DBI_1.1.0
## [49] curl_4.3             benchmarkmeData_1.0.4 R6_2.5.0
## [52] lubridate_1.7.9       knitr_1.30            rJava_0.9-13
## [55] stringi_1.5.3         vctrs_0.3.4           dbplyr_1.4.4
## [58] tidyselect_1.1.0      xfun_0.18
```

```r
## Return the machine CPU
cat("Machine:     "); print(get_cpu()$model_name)
```

```
## Machine:

## [1] "Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz"
```

```r
## Return number of true cores
cat("Num cores:   "); print(detectCores(logical = FALSE))
```

```
## Num cores:

## [1] 8
```

```r
## Return number of threads
cat("Num threads: "); print(detectCores(logical = FALSE))
```

```
## Num threads:

## [1] 8
```

References

Becker, William. 2020. "Metafunctions for benchmarking in sensitivity analysis." *Reliability Engineering & System Safety*, August, 107189. https://doi.org/10.1016/j.ress.2020.107189.

Jansen, M. 1999. "Analysis of variance designs for model output." *Computer Physics Communications* 117 (1-2): 35–43. https://doi.org/10.1016/S0010-4655(98)00154-4.

Liu, Huibin, Wei Chen, and Agus Sudjianto. 2006. "Relative entropy based method for probabilistic sensitivity analysis in engineering design." *Journal of Mechanical Design, Transactions of the ASME* 128 (2): 326–36. https://doi.org/10.1115/1.2159025.

Puy, Arnald, William Becker, Samuele Lo Piano, and Andrea Saltelli. 2020. "The battle of total-order sensitivity estimators," September. http://arxiv.org/abs/2009.01147.

Puy, Arnald, Samuele Lo Piano, and Andrea Saltelli. 2020. "Is VARS more intuitive and efficient than Sobol' indices?"

Razavi, Saman, and Hoshin V. Gupta. 2016. "A new framework for comprehensive, robust, and efficient global sensitivity analysis: 2. Application." *Water Resources Research* 52 (1): 440–55. https://doi.org/10.1002/2015WR017558.

Savage, I. Richard. 1956. "Contributions to the theory of rank order statistics - the two sample case." *Annals of Mathematical StatisticsStatistics* 27: 590–615.

Sobol', Ilya. M. 1967. "On the distribution of points in a cube and the approximate evaluation of integrals." *USSR Computational Mathematics and Mathematical Physics* 7 (4): 86–112. https://doi.org/10.1016/0041-5553(67)90144-9.

———. 1976. "Uniformly distributed sequences with an additional uniform property." *USSR Computational Mathematics and Mathematical Physics* 16 (5): 236–42. https://doi.org/10.1016/0041-5553(76)90154-3.