

Is VARS more intuitive and efficient than Sobol' indices?

R code

Arnald Puy

```

# 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("arnalduy/sensobol")

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

# 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("2020-07-26",
          R.version = "3.6.3",
          checkpointLocation = getwd())

```

## 1 Figures

```

# 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)
    }
  }
}

# Load the packages
loadPackages(c("tidyverse", "data.table", "cowplot"))

# 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))
}

```

## 2 Functions

```
# 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)
)

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)
}
```

### 3 Plot the 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")

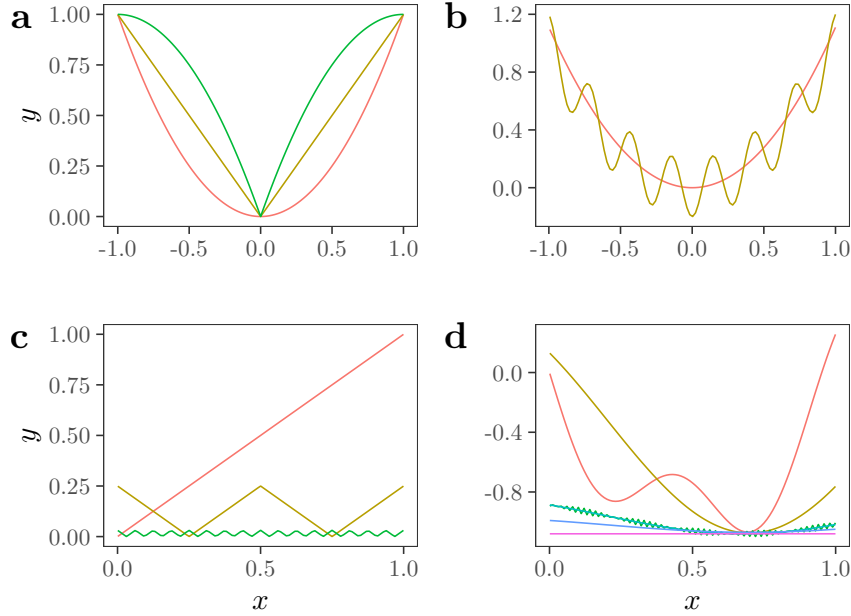
legend <- get_legend(d + theme(legend.position = "top"))

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))
```

Function

<span style="color: red;">—</span> $f_1(x)$	<span style="color: green;">—</span> $f_3(x)$	<span style="color: blue;">—</span> $f_5(x)$
<span style="color: gold;">—</span> $f_2(x)$	<span style="color: cyan;">—</span> $f_4(x)$	<span style="color: magenta;">—</span> $f_6(x)$



# 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) %>%
  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() %>%
```

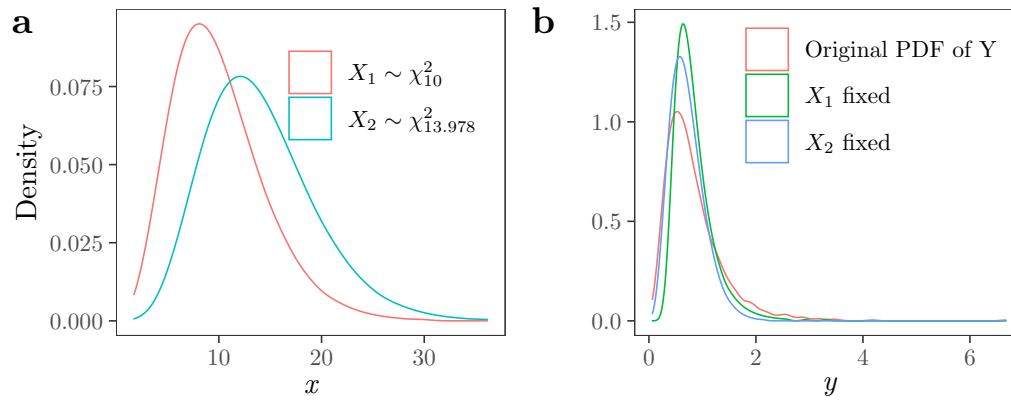
```

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)

```



*# 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(unnamed(center[[i]]), AB[[i]])
  }
  return(do.call(rbind, out))
}
```

*# 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)
}

```

*# 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?
)

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)
}

```

## 4 Sample matrix

```

# 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, 20), 0) # N.stars
mat[, 2] <- round(qunif(mat[, 2], 1, 4), 0) # h
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] 51 99020
```

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

```
head(mat)
```

```
##      N.stars    h  k k_2 k_3 epsilon delta phi tau Nt.vars N.jansen Nt.jansen
## [1,]      12 0.05 26 0.8 0.6      100     2  4  2   5940      220      5940
## [2,]      16 0.05 38 0.6 0.8       51     2  3  1  11568      297     11583
## [3,]       7 0.10 15 0.9 0.5      150     2  6  2    952       60       960
## [4,]       9 0.05 32 0.6 0.9      175     1  5  1   5481      166     5478
## [5,]      18 0.20  9 0.8 0.6       76     2  2  2    666       67       670
## [6,]      14 0.01 21 0.7 0.4      125     3  7  1  29120     1324     29128
```

## 5 The model

```
# 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 = ""))

  set.seed(epsilon)
  large.matrix <- sobol_matrices(matrices = c("A", "AB"), N = N.high,
    params = paste("X", 1:k, sep = ""),
    method = method)

  # Compute metafunction
  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)),
    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)
# Replace 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)
}

# 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)

# 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), ]

##          row N.stars    h  k k_2 k_3 epsilon delta phi tau Nt.vars N.jansen
##  1:      42      20 0.05  5 0.7 0.9    166     2  2  2   1920    320
##  2:     369      13 0.10  6 1.0 0.7     72     3  2  2    715    102
##  3:    1087       3 0.10 10 0.5 0.5     67     2  2  2    273     25
##  4:    1386      20 0.05  5 0.6 0.7     60     3  2  2   1920    320
##  5:    1507       8 0.10  6 0.8 1.0    113     1  2  2    440     63
##  ---
## 157: 44333      18 0.10  3 0.5 0.6     27     2  2  2    504    126
## 158: 44621      17 0.10  7 0.6 0.9    165     1  2  2   1088    136
## 159: 44782      13 0.05  3 0.8 0.6    173     2  2  2    754    188
## 160: 44782      13 0.05  3 0.8 0.6    173     2  2  2    754    188
## 161: 44819       9 0.10 10 0.5 1.0     48     3  2  2    819     74
##          Nt.jansen estimator correlation  Nt
##  1:          1920   VARS-TO          NaN 1920
##  2:           714   VARS-TO           NA  715
##  3:           275   VARS-TO          NaN  273
##  4:          1920   VARS-TO           NA 1920
##  5:           441   VARS-TO           NA  440
##  ---

```

```
## 157:      504  VARS-T0      NaN  504
## 158:     1088  VARS-T0      NA 1088
## 159:      752   Jansen     NaN  752
## 160:      752  VARS-T0     NaN  754
## 161:      814  VARS-T0      NA  819

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

##      estimator      V1
## 1:      Jansen 0.0007324219
## 2:      VARS-T0 0.0028409091

# 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")
```

## 6 Uncertainty analysis

```
# 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-T0 0.9503634 0.8740740 0.9905648
## 2:   Jansen 0.9826840 0.9354839 0.9985708

# 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()

```

unc

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

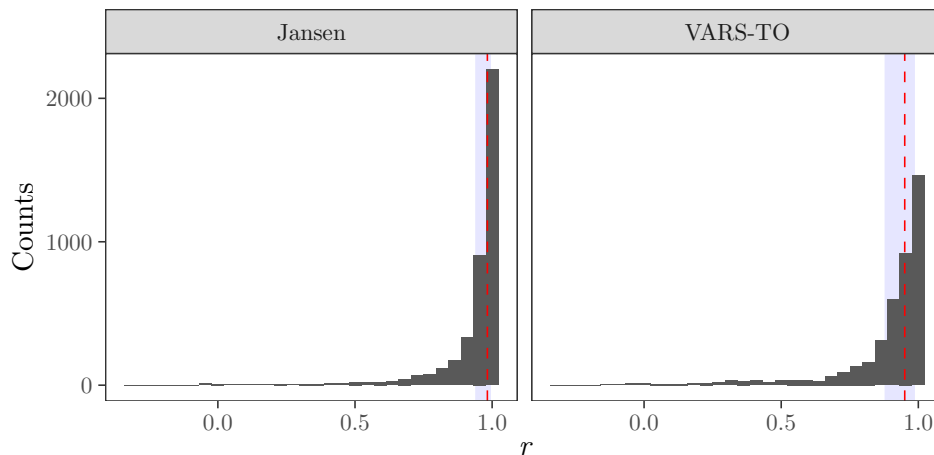


Figure 1: Empirical distribution of  $r$  for Jansen and VARS.

```

# SCATTERPLOT OF MODEL RESULTS -----
scat <- ggplot(A, aes(Nt, k, color = correlation)) +
  geom_point(size = 0.4) +
  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")

scat
```

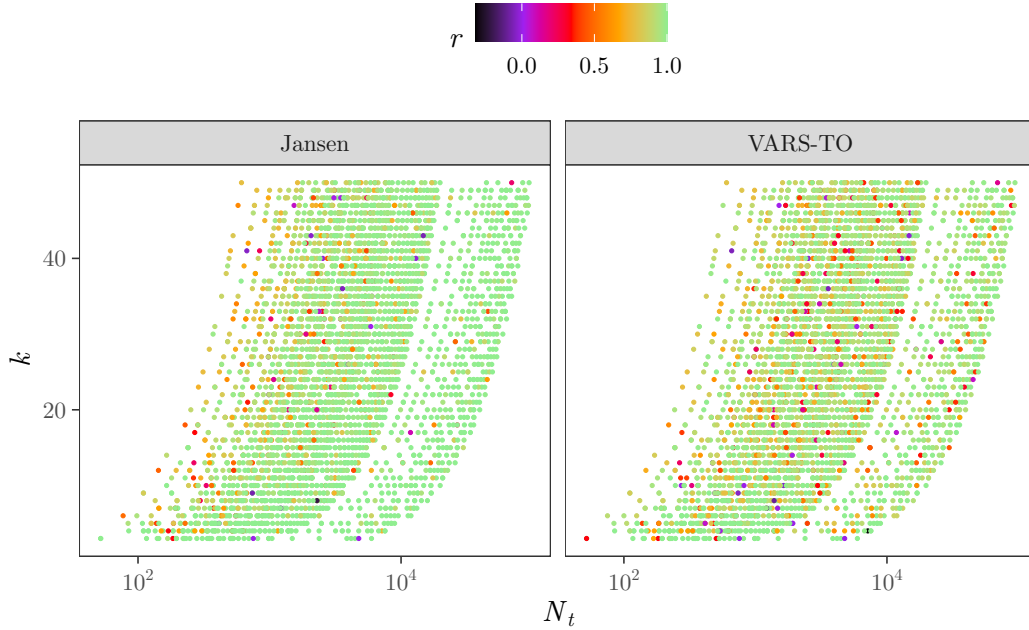


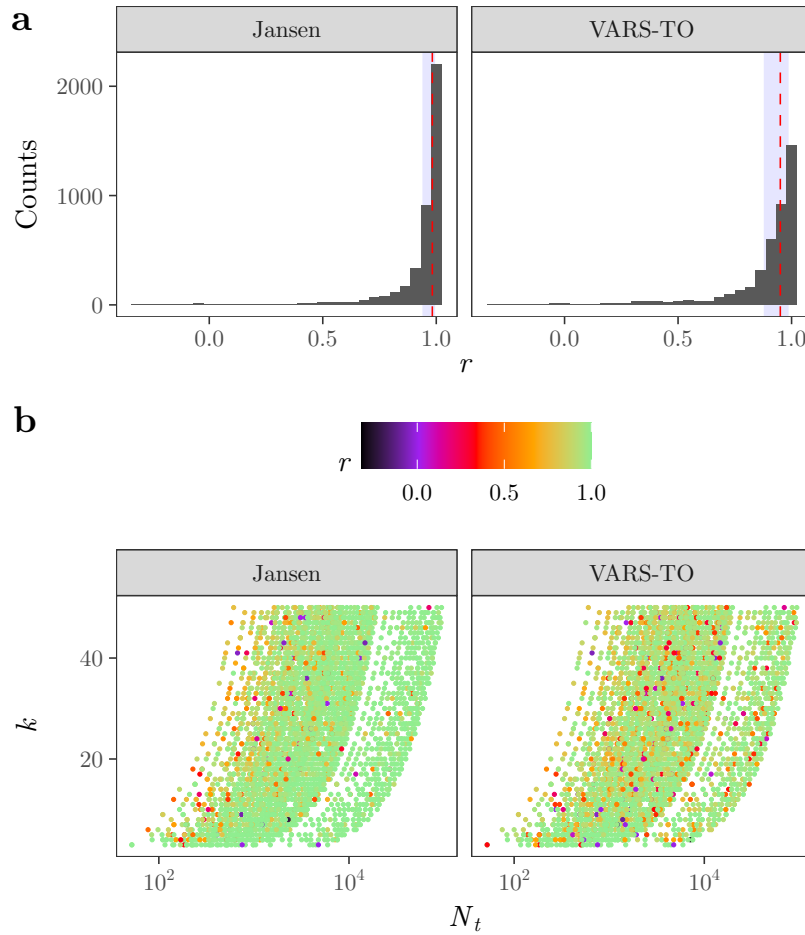
Figure 2: Scatterplots of the total number of model runs  $N_t$  against the function dimensionality  $k$ . The greener (darker) the colour, the better (worse) the performance.

```
# 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_discrete(name = "Estimator") +
  labs(x = expression(italic(k)),
       y = expression(median(italic(r)))) +
  geom_line() +
  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::rowmeans(indices)

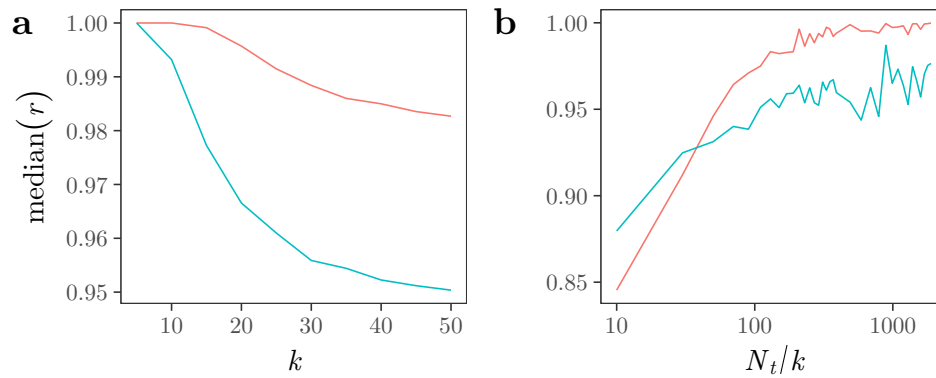
# Plot r as a function of Nt/k
a2 <- lapply(out, function(x) x[, median(correlation, na.rm = TRUE), estimator]) %>%
  rbindlist(., idcol = "N") %>%
  .[, N:= as.numeric(N)] %>%
  ggplot(., aes(N, V1, group = estimator, color = estimator)) +
  geom_line() +
  labs(x = expression(italic(N[t]/k)),
    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

# Merge both plots
legend <- get_legend(a1 + theme(legend.position = "top"))
sides <- plot_grid(a1, a2, ncol = 2, rel_widths = c(1, 1), align = "hv",
                  labels = "auto")

plot_grid(legend, sides, rel_heights = c(0.2, 1), ncol = 1)

```

Estimator — Jansen — VARS-TO



## 7 Sensitivity analysis

```
# 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 = 0.5, alpha = 0.1) +
  facet_wrap(~variable,
             scales = "free_x",
             labeller = label_parsed) +
  labs(y = expression(italic(r)),
       x = "") +
  theme_AP()

# 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,
                                       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 = "",
```

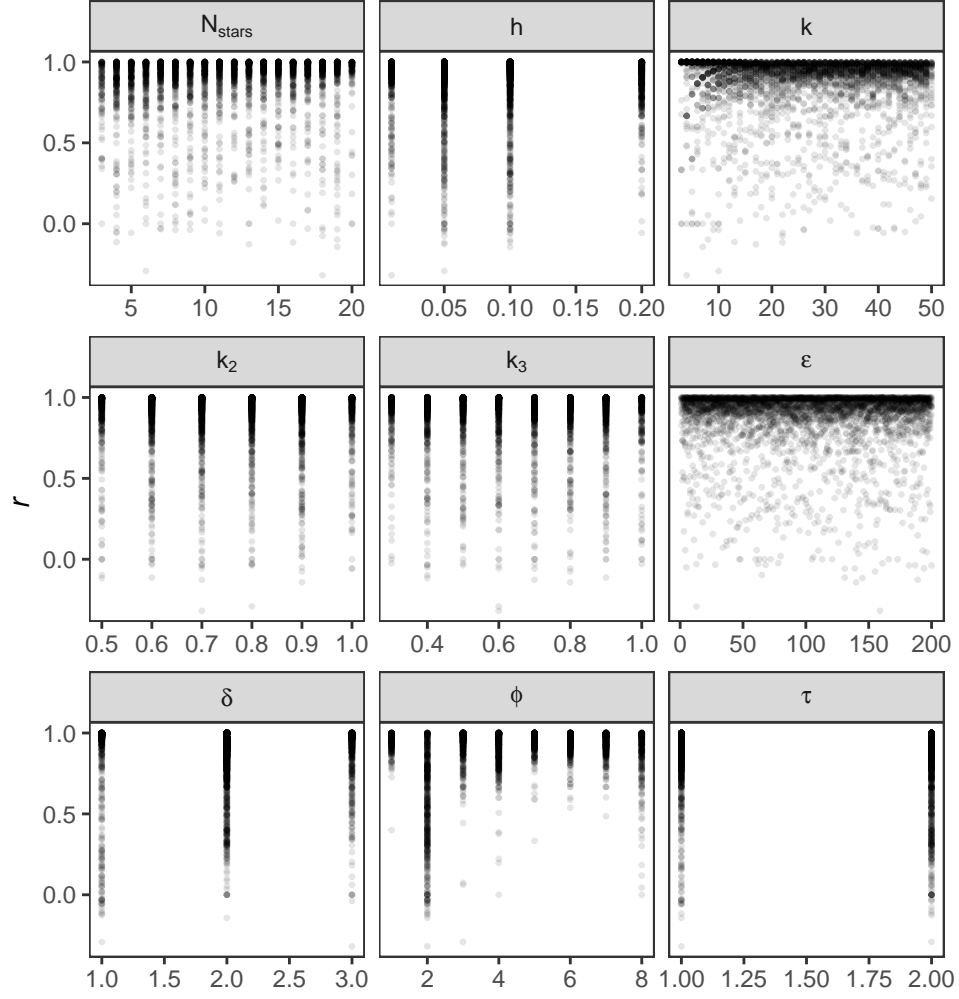


Figure 3: Scatterplots of model output  $r$  against the model inputs.

```

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")

```

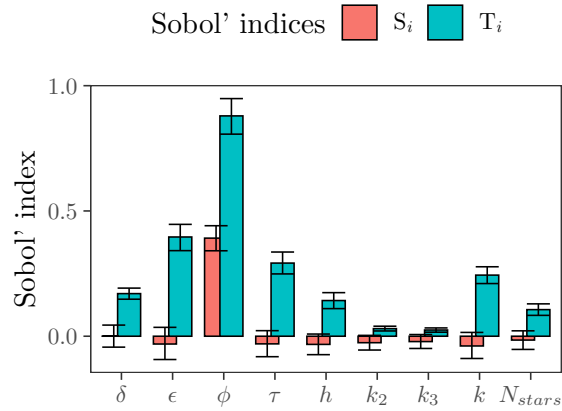


Figure 4: Sobol' indices.

*# SUM OF FIRST-ORDER INDICES* -----

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

```
## [1] 0.5780782
```

## 8 Session information

```
# SESSION INFORMATION -----  
  
sessionInfo()  
  
## R version 3.6.3 (2020-02-29)  
## Platform: x86_64-apple-darwin15.6.0 (64-bit)  
## Running under: macOS Catalina 10.15.5  
##  
## Matrix products: default  
## BLAS: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib  
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/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] parallel stats graphics grDevices utils datasets methods  
## [8] base  
##  
## other attached packages:  
## [1] truncnorm_1.0-8 data.table_1.12.8 checkpoint_0.4.9 Rfast_1.9.9  
## [5] RcppZigurat_0.1.5 Rcpp_1.0.5 benchmarkme_1.0.4 logitnorm_0.8.37  
## [9] cowplot_1.0.0 scales_1.1.1 pcaPP_1.9-73 doParallel_1.0.15  
## [13] iterators_1.0.12 foreach_1.5.0 sensobol_0.3 forcats_0.5.0  
## [17] stringr_1.4.0 dplyr_1.0.0 purrr_0.3.4 readr_1.3.1  
## [21] tidyr_1.1.0 tibble_3.0.3 ggplot2_3.3.2 tidyverse_1.3.0  
##  
## loaded via a namespace (and not attached):  
## [1] httr_1.4.2 jsonlite_1.7.0 modelr_0.1.8  
## [4] Rdpack_1.0.0 assertthat_0.2.1 highr_0.8  
## [7] tikzDevice_0.12.3.1 blob_1.2.1 cellranger_1.1.0  
## [10] yaml_2.2.1 remotes_2.2.0 pillar_1.4.6  
## [13] backports_1.1.8 lattice_0.20-41 glue_1.4.1  
## [16] digest_0.6.25 rvest_0.3.6 colorspace_1.4-1  
## [19] htmltools_0.5.0 Matrix_1.2-18 pkgconfig_2.0.3  
## [22] bibtex_0.4.2.2 broom_0.7.0 haven_2.3.1  
## [25] mvtnorm_1.1-1 farver_2.0.3 generics_0.0.2  
## [28] ellipsis_0.3.1 withr_2.2.0 cli_2.0.2  
## [31] magrittr_1.5 crayon_1.3.4 readxl_1.3.1  
## [34] evaluate_0.14 fs_1.4.2 fansi_0.4.1  
## [37] xml2_1.3.2 tools_3.6.3 hms_0.5.3  
## [40] gbRd_0.4-11 lifecycle_0.2.0 munsell_0.5.0  
## [43] reprex_0.3.0 compiler_3.6.3 tinytex_0.25  
## [46] rlang_0.4.7 grid_3.6.3 rstudioapi_0.11  
## [49] filehash_2.4-2 rmarkdown_2.3 gtable_0.3.0  
## [52] codetools_0.2-16 DBI_1.1.0 curl_4.3
```

```
## [55] benchmarkmeData_1.0.4 R6_2.4.1          lubridate_1.7.9
## [58] knitr_1.29             stringi_1.4.6      vctrs_0.3.2
## [61] dbplyr_1.4.4           tidysselect_1.1.0  xfun_0.16
```

```
## Return the machine CPU
```

```
cat("Machine:    "); print(get_cpu()$model_name)
```

```
## Machine:
```

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

```
## Return number of true cores
```

```
cat("Num cores:  "); print(detectCores(logical = FALSE))
```

```
## Num cores:
```

```
## [1] 8
```

```
## Return number of threads
```

```
cat("Num threads: "); print(detectCores(logical = FALSE))
```

```
## Num threads:
```

```
## [1] 8
```