The battle of total-order sensitivity estimators

Arnald Puy, Samuele Lo Piano, Andrea Saltelli and William Becker

Contents

1 Specific functions **3**
1.1 Sample matrices . . . 3
1.2 Savage scores . . . 4
1.3 Sobol' indices . . . 5

2 The metafunction **7**

3 The model **9**
3.1 Settings . . . 9
3.2 Sample matrix . . . 10
3.3 Define the model . . . 10
3.4 Run the model . . . 11
3.5 Arrange output . . . 12

4 Session information **15**

References **17**

```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)
    }
  }
}

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

# 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("2020-01-23",
           R.version ="3.6.1",
           checkpointLocation = getwd())
```

1 Specific functions

This section presents all functions that will be used in the analysis.

1.1 Sample matrices

We first start by defining two functions to create $\mathbf{A}$, $\mathbf{B}$, $\mathbf{A}_B^{(i)}$ and $\mathbf{B}_A^{(i)}$ matrices using Sobol' quasi-random numbers. Note that the code allows to create matrices to compute up to third-order effects, although we will not use them here.

```
# FUNCTIONS TO CREATE SAMPLE MATRICES ----------------------------------------

scrambled_sobol <- function(matrices, A, B, order, cluster) {
  first <- 1:ncol(A)
  N <- nrow(A)
  if(order == "first") {
    loop <- first
  } else if(order == "second" | order == "third") {
    second <- c(first, utils::combn(1:ncol(A), 2, simplify = FALSE))
    loop <- second
  } else if(order == "third") {
    third <- c(second, utils::combn(1:ncol(A), 3, simplify = FALSE))
    loop <- third
  } else {
    stop("order should be either first, second or third")
  }
  if(is.null(cluster) == FALSE) {
    loop <- cluster
  }
  AB.mat <- "AB" %in% matrices
  BA.mat <- "BA" %in% matrices
  if(AB.mat == TRUE) {
    X <- rbind(A, B)
    for(i in loop) {
      AB <- A
      AB[, i] <- B[, i]
      X <- rbind(X, AB)
    }
    AB <- X[(2 * N + 1):nrow(X), ]
  } else if(AB.mat == FALSE) {
    AB <- NULL
  }
  if(BA.mat == TRUE) {
    W <- rbind(A, B)
    for(i in loop) {
      BA <- B
      BA[, i] <- A[, i]
      W <- rbind(W, BA)
    }
```

```
    BA <- W[(2 * N + 1) : nrow(W), ]
  } else if(BA.mat == FALSE) {
    BA <- NULL
  }
  return(rbind(AB, BA))
}

sobol_matrices <- function(matrices = c("A", "B", "AB"),
                           N, params, order = "first",
                           cluster = NULL) {
  if(length(matrices) >= 4 & !order == "first" ) {
    stop("higher orders should be computed with an A, B and AB or BA matrices")
  }
  k <- length(params)
  df <- randtoolbox::sobol(n = N, dim = k * 2)
  A <- df[, 1:k]
  B <- df[, (k + 1) : (k * 2)]
  out <- scrambled_sobol(matrices = matrices,
                         A = A, B = B, order = order,
                         cluster = cluster)
  A.mat <- "A" %in% matrices
  B.mat <- "B" %in% matrices
  if(A.mat == FALSE) {
    A <- NULL
  }
  if(B.mat == FALSE) {
    B <- NULL
  }
  final <- rbind(A, B, out)
  colnames(final) <- params
  return(final)
}
```

1.2 Savage scores

The following code snippet allows to compute Savage scores (Iman and Conover 1987), which read as

$$SS_i = \sum_{j=1}^{N} \frac{1}{j} \tag{1}$$

Savage scores will be used as a measure of performance to check how well each estimator identifies the true ranks of the most important model inputs.

```
# 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)
}
```

1.3 Sobol' indices

We define here all the T_i estimators we will analyze in our study: Jansen (1999), Janon et al. (2014), Homma and Saltelli (1996), Azzini and Rosati (2019) and Sobol' and Myshetskaya (2008). We then prove that the code works by computing and plotting the T_i indices of three well-known test functions: the Ishigami and Homma (1990), the Sobol' (1993) 's G and the Morris (1991) functions.

```
# COMPUTATION OF SOBOL' Ti INDICES -----------------------------------------

sobol_Ti <- function(d, N, params, total) {
  m <- matrix(d, nrow = N)
  k <- length(params)
  if(!total == "azzini") {
    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)
     # VY <- 1 / length(Y_A) * (sum(Y_A ^ 2) -
    # (1 / N * sum(Y_A ^ 2))) ((Variance used by Becker))
  }
  if(total == "jansen") {
    Ti <- (1 / (2 * N) * Rfast::colsums((Y_A - Y_AB) ^ 2)) / VY
  } else if(total == "homma") {
    Ti <- (VY - (1 / N) * Rfast::colsums(Y_A * Y_AB) + f0 ^ 2) / VY
  } else if(total == "sobol") {
    Ti <- ((1 / N) * Rfast::colsums(Y_A * (Y_A - Y_AB))) / VY
  } else if(total == "monod") {
    Ti <- 1 - (1 / N * Rfast::colsums(Y_A * Y_AB) -
                  (1/ N * Rfast::colsums((Y_A + Y_AB) / 2)) ^ 2) /
      (1 / N * Rfast::colsums((Y_A ^ 2 + Y_AB ^ 2) / 2) -
         (1/ N * Rfast::colsums((Y_A + Y_AB) / 2)) ^ 2)
  }
  if(total == "azzini") {
    Y_A <- m[, 1]
    Y_B <- m[, 2]
    Y_AB <- m[, 3:(3 + k - 1)]
    Y_BA <- m[, (ncol(m) - k + 1):ncol(m)]
    Ti <- 1 - abs(Rfast::colsums((Y_A - Y_BA) * (Y_B - Y_AB)) /
                    (1 / 2 * Rfast::colsums((Y_A - Y_B) ^ 2 + (Y_AB - Y_BA) ^ 2)))
  }
```

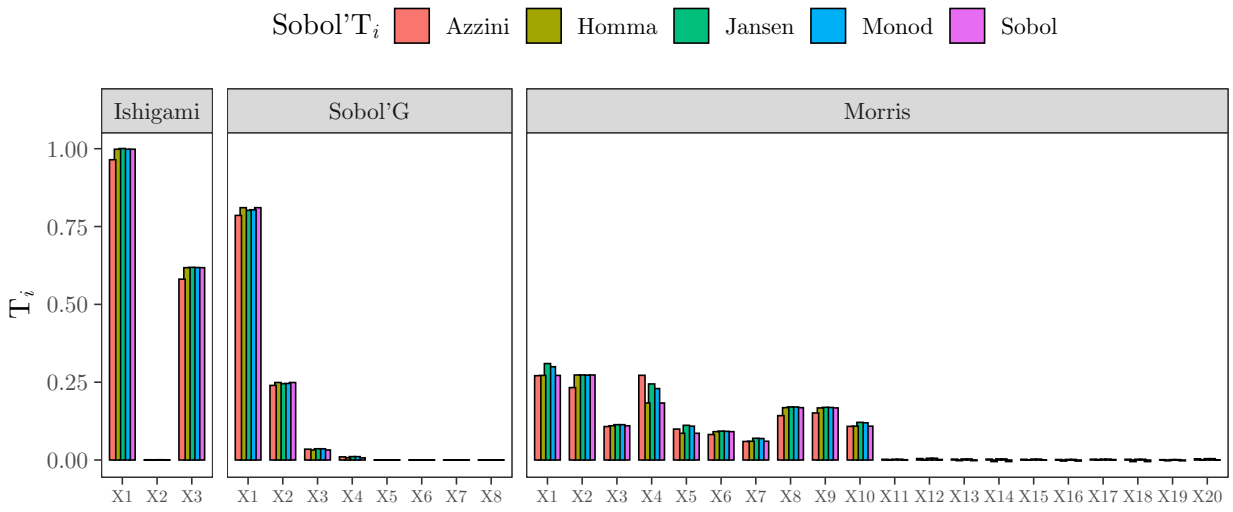
```
  output <- data.table(Ti)
  output[, `:=`(parameters = paste("X", 1:k, sep = ""),
                ranks= rank(-Ti),
                savage.scores = savage_scores(Ti))]
  return(output)
}
```

```
# CHECK THAT ALL TI ESTIMATORS WORK ----------------------------------------

# Settings
estimators <- c("jansen", "sobol", "homma", "azzini", "monod")
test_functions <- c("Ishigami", "Sobol'G", "Morris")
N <- 2^9

# Run model
ind <- Y <- mt <- list()
for(i in estimators) {
  for(j in test_functions) {
    if(!i == "azzini") {
      matrices <- c("A", "AB")
    } else {
      matrices <- c("A", "B", "AB", "BA")
    }
    if(j == "Ishigami") {
      k <- 3
      modelRun <- sensobol::ishigami_Mapply
    } else if(j == "Sobol'G") {
      k <- 8
      modelRun <- sensobol::sobol_Fun
    } else if(j == "Morris") {
      k <- 20
      modelRun <- sensitivity::morris.fun
    }
    mt[[i]][[j]] <- sobol_matrices(N = N, params = paste("X", 1:k, sep = ""), matrices = matri
    Y[[i]][[j]] <- modelRun(mt[[i]][[j]])
    ind[[i]][[j]] <- sobol_Ti(d = Y[[i]][[j]], params = paste("X", 1:k, sep = ""),
                              N = N, total = i)
  }
}
```

```
## Registered S3 method overwritten by 'sensitivity':
##   method    from
##   print.src dplyr
```

```
# PLOT SENSITIVITY INDICES -------------------------------------------------

lapply(ind, function(x) rbindlist(x, idcol = "Function")) %>%
  rbindlist(., idcol = "estimator") %>%
```

```
.[, parameters:= factor(parameters, levels = paste("X", 1:20, sep = ""))] %>%
.[, Function:= factor(Function, levels = test_functions)] %>%
ggplot(., aes(parameters, Ti, fill = estimator)) +
geom_bar(stat = "identity",
         position = position_dodge(0.7),
         color = "black") +
facet_grid(~Function,
           scales = "free_x",
           space = "free_x") +
scale_fill_discrete(name = expression(paste("Sobol' ", T[italic(i)])),
                    labels = c("Azzini", "Homma", "Jansen",
                               "Monod", "Sobol")) +
labs(x = "",
     y = expression(T[italic(i)])) +
theme_AP() +
theme(axis.text.x = element_text(size = 6.5),
      legend.position = "top")
```



2 The metafunction

In this section we code Becker (2019)'s metafunction which reads as follows:

$$
\begin{aligned}
Y = &\sum_{i=1}^{k} \alpha_i f^{ui}(x_i) + \sum_{i=1}^{k_2} \beta_i f^{u_{V_{i,1}}}(x_{V_{i,1}}) f^{u_{V_{i,2}}}(x_{V_{i,2}}) \\
&+ \sum_{i=1}^{k_3} \beta_i f^{u_{W_{i,1}}}(x_{W_{i,1}}) f^{u_{W_{i,2}}}(x_{W_{i,2}}) f^{u_{W_{i,3}}}(x_{W_{i,3}})
\end{aligned}
\tag{2}
$$

We first start by creating a list with the ten univariate functions that the metafunction will include, and plot them.

```r
# CREATE 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),
  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)
)
```

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

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))) +
  theme_AP() +
  theme(legend.position = "right")
```

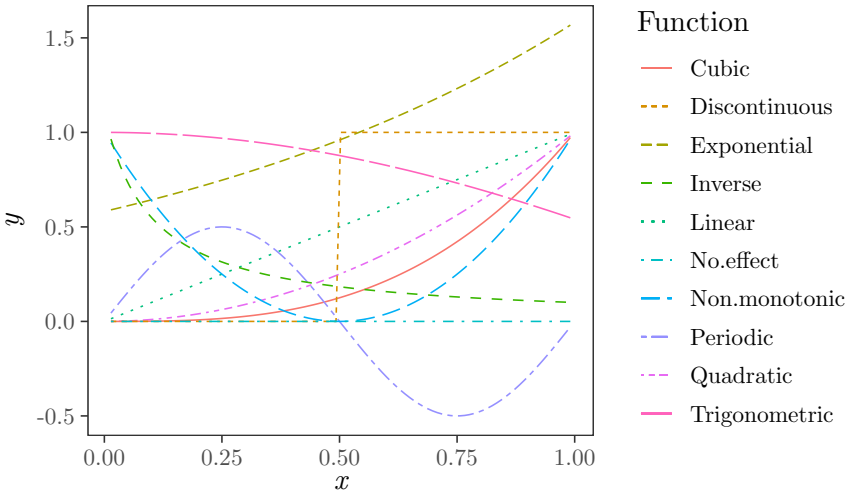


Figure 1: Functions used in the metafunction of Becker (2019).

This code snippet calls a function coded in C++ that allows to multiply a vector by a matrix. It will be included in the metafunction.

```
Rcpp::sourceCpp("vector_multiplication.cpp")
```

And here we code the main behaviour of the metafunction.

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

metafunction <- function(X) {
  k <- ncol(X)
  # Define functions according to k
  all_functions <- sample(names(function_list), k, replace = TRUE)
  # Define coefficients
  components <- sample(1:2, prob = c(0.5, 0.5), size = 200, replace = TRUE)
  mus <- c(0, 0)
  sds <- sqrt(c(0.5, 5))
  coefficients <- rnorm(100) * sds[components] + mus[components]
  # Coefficients for first
  coefD1 <- sample(coefficients, k)
  # Coefficients for pairs
  d2 <- t(utils::combn(1:k, 2))
  d2M <- d2[sample(nrow(d2), size = ceiling(k * 0.5), replace = FALSE), ]
  coefD2 <- sample(coefficients, nrow(d2M), replace = TRUE)
  # Coefficients for triplets
  d3 <- t(utils::combn(1:k, 3))
  d3M <- d3[sample(nrow(d3), size = ceiling(k * 0.2), replace = FALSE), ]
  sample.size <- ifelse(is.vector(d3M) == TRUE, 1, nrow(d3M))
  coefD3 <- sample(coefficients, sample.size, replace = TRUE)
  # Run sampled functions in each column
  output <- sapply(seq_along(all_functions), function(x) function_list[[all_functions[x]]](X[,
  y1 <- Rfast::rowsums(mmult(output, coefD1))
  y2 <- Rfast::rowsums(mmult(output[, d2M[, 1]] *  output[, d2M[, 2]], coefD1))
  if(is.vector(d3M) == TRUE) {
    y3 <- sum(output[, d3M[1]] *  output[, d3M[2]] * output[, d3M[3]] * coefD3)
  } else {
    y3 <- Rfast::rowsums(mmult(output[, d3M[, 1]] *  output[, d3M[, 2]] * output[, d3M[, 3]],
  }
  Y <- y1 + y2 + y3
  return(Y)
}
```

3 The model

This section sets the ground for the battle of the estimators.

3.1 Settings

We first define the settings of the analysis, including the sample size of the base sample matrix. This sample matrix will initially have two columns: k (which will define the dimensionality of the metafunction) and C (which will define the total cost or the total number of runs of the

metafunction). `N.high` defines the size of the sample matrix that will be used to compute the "true" ranks for each run of the metafunction.

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

N <- 2 ^ 10 # Sample size of sample matrix
params <- c("k", "C")
N.high <- 2 ^ 13 # Maximum sample size of the large sample matrix
```

3.2 Sample matrix

Here we create the sample matrix using Sobol' quasi-random number sequences, and transform each column to its appropriate distribution: the first column will define the dimensionality k of the metafunction, and we will describe it as $k \sim \mathcal{U}(3, 200)$. The second column will define the total cost C of the analysis and will be described as $k \sim \mathcal{U}(10, 2000)$. Finally, we create two extra columns that will define the initial sample size of the matrix needed to compute the all estimators but Azzini and Rosati (2019) (`N.all`), and Azzini and Rosati (2019) (`N.azzini`), given the value of C and k in each row of the sample matrix.

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

mat <- randtoolbox::sobol(N, length(params))
mat[, 1] <- floor(qunif(mat[, 1], 3, 200))
mat[, 2] <- floor(qunif(mat[, 2], 10, 2000))
colnames(mat) <- params

N.all <- apply(mat, 1, function(x) ceiling(x["C"] / (x["k"] + 1)))
N.azzini <- apply(mat, 1, function(x) ceiling(x["C"] / (2 * x["k"] + 2)))

tmp <- cbind(mat, N.all, N.azzini)
sel <- c("N.all", "N.azzini")

mat <- as.matrix(data.table(tmp)[, (sel):= lapply(.SD, function(x)
  ifelse(x == 1, 2, x)), .SDcols = (sel)])
```

3.3 Define the model

This section codes the model, which works as follows:

- Create the 'estimators' vector, which includes the name of all estimators.
- Create a ('N.all', k) sample matrix formed by an $\mathbf{A}$ and an $\mathbf{A}_B^{(i)}$ matrix, which will be used to compute all T_i estimators except the Azzini, which requires an $\mathbf{A}$, $\mathbf{B}$, $\mathbf{A}_B^{(i)}$ and a $\mathbf{B}_A^{(i)}$ matrix.
- Create a ('N.high', k) sample matrix formed by an $\mathbf{A}$ and an $\mathbf{A}_B^{(i)}$ matrix. This will be the "large" sample matrix used to compute the "true" total-order indices. It will have an $\mathbf{A}$ and an $\mathbf{A}_B^{(i)}$ matrix because the reference estimator to compute the true total-order indices will be the Jansen estimator.
- We then bind all three sample matrices and call the metafunction throughout. Binding is

mandatory here as the metafunction needs to be called just once: if we call the metafunction in each sample matrix separately, its randomness will make it different each time and the comparison will not be possible.

- We compute the "true" total-order indices for the "large" sample matrix only using the Jansen estimator.
- We compute the total-order indices using each estimator. This works with a ‘for‘ loop.

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

model_Ti <- function(k, N.all, N.azzini, N.high) {
  ind <- list()
  estimators <- c("jansen", "sobol", "homma", "monod", "azzini")
  all.but.azzini <- sobol_matrices(N = N.all, params = paste("X", 1:k, sep = ""),
                                   matrices = c("A", "AB"))
  azzini <- sobol_matrices(N = N.azzini, params = paste("X", 1:k, sep = ""),
                           matrices = c("A", "B", "AB", "BA"))
  large.matrix <- sobol_matrices(N = N.high, params = paste("X", 1:k, sep = ""),
                                 matrices = c("A", "AB"))
  output <- metafunction(rbind(all.but.azzini, azzini, large.matrix))
  full.ind <- sobol_Ti(d = tail(output, nrow(large.matrix)),
                       N = N.high,
                       params = paste("X", 1:k, sep = ""),
                       total = "jansen")
  full.ind[, sample.size:= "N"]
  for(i in estimators) {
    if(!i == "azzini") {
      y <- output[1:nrow(all.but.azzini)]
      n <- N.all
    } else {
      y <- output[-c(1:nrow(all.but.azzini),
                     (nrow(all.but.azzini) + nrow(azzini) + 1):length(output))]
      n <- N.azzini
    }
    ind[[i]] <- sobol_Ti(d = y, N = n, params = paste("X", 1:k, sep = ""), total = i)
    ind[[i]][, sample.size:= "n"]
    ind[[i]] <- rbind(ind[[i]], full.ind)
  }
  return(ind)
}
```

3.4 Run the model

This code snippet runs the model. In my computer (see section System Information) it took approximately 4 h.

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

Y.ti <- list()
```

```
for(i in 1:nrow(mat)) {
  Y.ti[[i]] <- model_Ti(k = mat[[i, "k"]],
                        N.all = mat[[i, "N.all"]],
                        N.azzini = mat[[i, "N.azzini"]],
                        N.high = N.high)
}
```

3.5 Arrange output

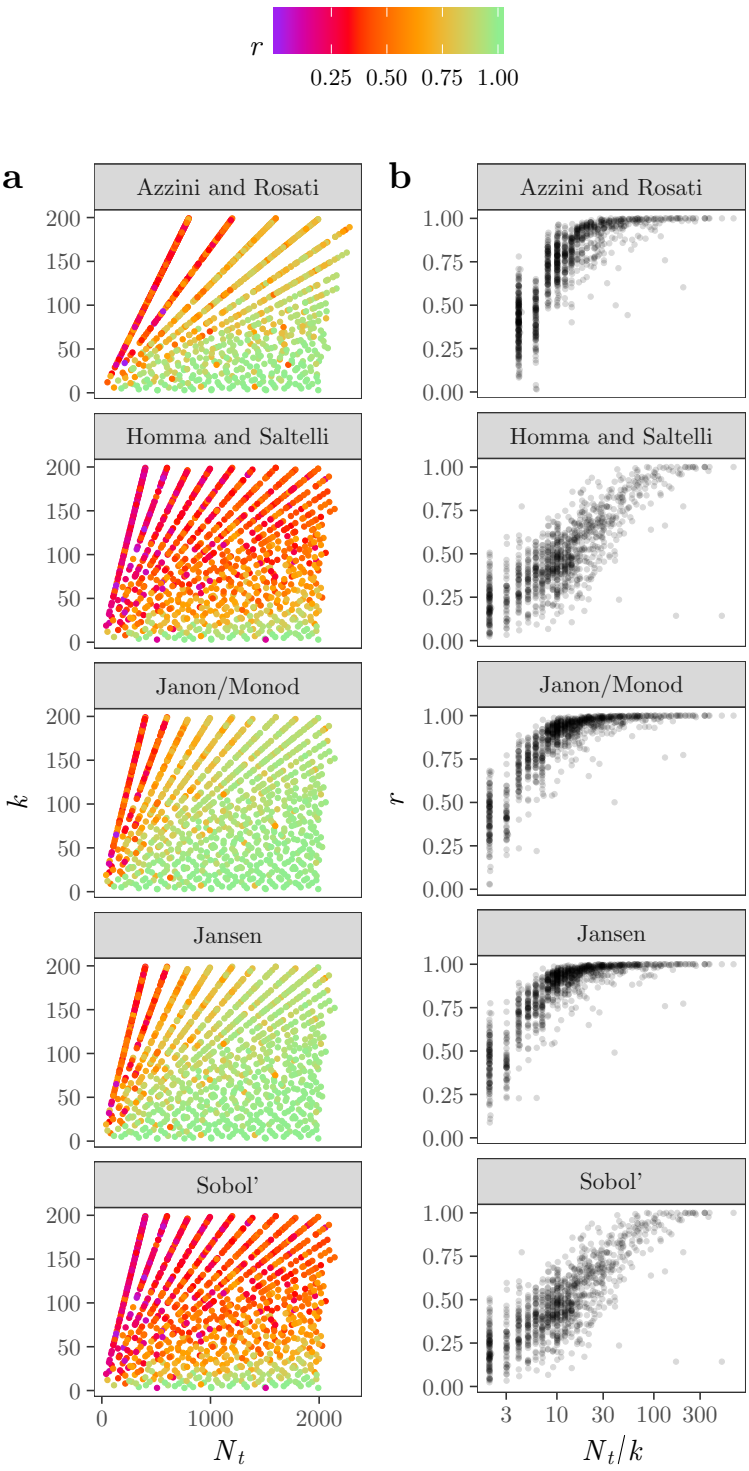
This section arranges the output to plot the results.

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

out <- lapply(Y.ti, function(x) rbindlist(x, idcol = "estimator")) %>%
  rbindlist(., idcol = "row")

out_wide <- spread(out[, .(sample.size, savage.scores, parameters, estimator, row)],
                   sample.size, savage.scores)

out_cor <- out_wide[, .(correlation = cor(N, n)), .(estimator, row)]
mt.dt <- data.table(mat) %>%
  .[, row:= 1:.N]

full_output <- merge(mt.dt, out_cor) %>%
  .[, Nt:= ifelse(estimator == "azzini", N.azzini * (2 * k + 2), N.all * (k + 1))] %>%
  .[, estimator:= ifelse(estimator %in% "azzini", "Azzini and Rosati",
                         ifelse(estimator %in% "homma", "Homma and Saltelli",
                                ifelse(estimator %in% "monod", "Janon/Monod",
                                       ifelse(estimator %in% "jansen", "Jansen", "Sobol'"))))]
```

```
# EXPORT OUTPUT -------------------------------------------------------------

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

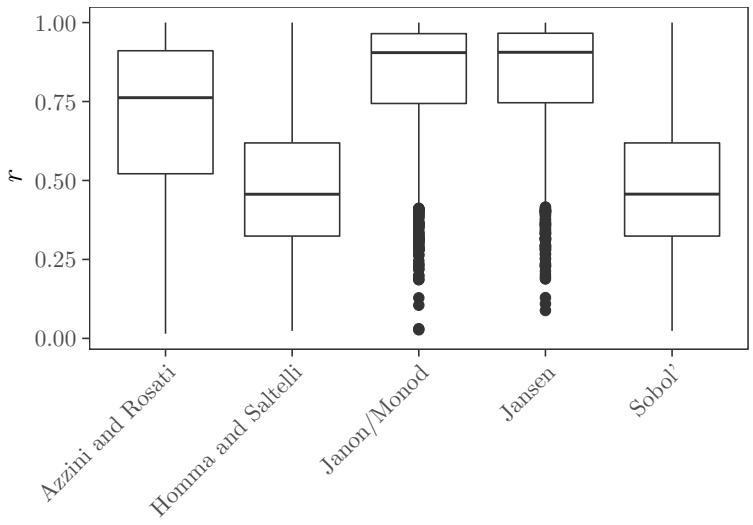
```
# PLOT OUTPUT ---------------------------------------------------------------

full.output <- full_output[, ratio:= Nt / k]

# Scatterplot
a <- ggplot(full_output[correlation > 0], aes(Nt, k, color = correlation)) +
  geom_point(size = 0.6) +
  scale_colour_gradientn(colours = c("purple", "red", "orange", "lightgreen"),
                         name = expression(italic(r))) +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = expression(italic(N[t])),
       y = expression(italic(k))) +
  facet_wrap(~estimator,
             ncol = 1) +
  theme_AP() +
```

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

# Get legend
legend <- get_legend(a + theme(legend.position = "top"))

# Ratio
b <- ggplot(full_output[correlation > 0], aes(ratio, correlation)) +
  geom_point(alpha = 0.15, size = 0.6) +
  facet_wrap(~estimator,
             ncol = 1) +
  labs(x = expression(italic(N[t]/k)),
       y = expression(italic(r))) +
  scale_x_log10() +
  theme_AP()

# Merge plot
bottom <- plot_grid(a, b, ncol = 2, labels = "auto")
plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.15, 1))
```




```
# PLOT BOXPLOT ------------------------------------------------------------------

ggplot(full_output[correlation > 0], aes(estimator, correlation)) +
  geom_boxplot() +
  labs(x = "",
       y = expression(italic(r))) +
```

```
  theme_AP() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



4 Session information

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

sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Catalina 10.15.3
##
## 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] checkpoint_0.4.8   benchmarkme_1.0.3  cowplot_1.0.0      scales_1.1.0
##  [5] data.table_1.12.8  Rfast_1.9.8        RcppZiggurat_0.1.5 doParallel_1.0.15
##  [9] iterators_1.0.12   foreach_1.4.7      forcats_0.4.0      stringr_1.4.0
## [13] dplyr_0.8.3        purrr_0.3.3        readr_1.3.1        tidyr_1.0.0
## [17] tibble_2.1.3       ggplot2_3.2.1      tidyverse_1.3.0    Rcpp_1.0.3
##
## loaded via a namespace (and not attached):
```

```
##  [1] lubridate_1.7.4       lattice_0.20-38       assertthat_0.2.1
##  [4] zeallot_0.1.0         digest_0.6.23         R6_2.4.1
##  [7] cellranger_1.1.0      backports_1.1.5       reprex_0.3.0
## [10] evaluate_0.14         httr_1.4.1            pillar_1.4.3
## [13] rlang_0.4.2           lazyeval_0.2.2        readxl_1.3.1
## [16] rstudioapi_0.10       Matrix_1.2-18         tikzDevice_0.12.3
## [19] rmarkdown_2.1         labeling_0.3          tinytex_0.19
## [22] munsell_0.5.0         broom_0.5.3           compiler_3.6.1
## [25] modelr_0.1.5          xfun_0.12             pkgconfig_2.0.3
## [28] htmltools_0.4.0       tidyselect_0.2.5      codetools_0.2-16
## [31] fansi_0.4.1           crayon_1.3.4          dbplyr_1.4.2
## [34] withr_2.1.2           grid_3.6.1            nlme_3.1-143
## [37] jsonlite_1.6          gtable_0.3.0          lifecycle_0.1.0
## [40] DBI_1.1.0             magrittr_1.5          cli_2.0.1
## [43] stringi_1.4.5         farver_2.0.3          fs_1.3.1
## [46] benchmarkmeData_1.0.3 xml2_1.2.2            generics_0.0.2
## [49] vctrs_0.2.1           tools_3.6.1           glue_1.3.1
## [52] hms_0.5.3             yaml_2.2.0            colorspace_1.4-1
## [55] filehash_2.4-2        rvest_0.3.5           knitr_1.27
## [58] haven_2.2.0
```

```
## 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 = TRUE))
```

```
## Num threads:
```

```
## [1] 16
```

```
## Return the machine RAM
cat("RAM:         "); print (get_ram()); cat("\n")
```

```
## RAM:
```

```
## 34.4 GB
```

References

Azzini, Ivano, and Rossana Rosati. 2019. "The IA-Estimator for Sobol' sensitivity indices." In *Ninth International Conference on Sensitivity Analysis of Model Output*. Barcelona.

Becker, William. 2019. "Sensitivity analysis on a shoestring : screening model inputs at low sample size."

Homma, T., and A. Saltelli. 1996. "Importance measures in global sensitivity analysis of nonlinear models." *Reliability Engineering & System Safety* 52: 1–17. https://doi.org/10.1016/0951-8320(96)00002-6.

Iman, Ronald L., and W. J. Conover. 1987. "A measure of top-down correlation." *Technometrics* 29 (3): 351–57.

Ishigami, T., and T. Homma. 1990. "An importance quantification technique in uncertainty analysis for computer models." *Proceedings. First International Symposium on Uncertainty Modeling and Analysis* 12: 398–403.

Janon, Alexandre, Thierry Klein, Agnès Lagnoux, Maëlle Nodet, and Clémentine Prieur. 2014. "Asymptotic normality and efficiency of two Sobol index estimators." *ESAIM - Probability and Statistics* 18 (Toulouse 3): 342–64. https://doi.org/10.1051/ps/2013040.

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.

Morris, M. 1991. "Factorial sampling plans for preliminary computational experiments." *Technometrics* 33 (2): 161–74.

Sobol', I. M. 1993. "Sensitivity analysis for nonlinear mathematical models." *Mathematical Modeling and Computational Experiment* 1 (4): 407–14. https://doi.org/10.18287/0134-2452-2015-39-4-459-461.

Sobol', I. M., and E. E. Myshetskaya. 2008. "Monte Carlo estimators for small sensitivity indices." *Monte Carlo Methods and Applications* 13 (5-6). https://doi.org/10.1515/mcma.2007.023.