Efficient computation of Sobol' sensitivity indices based on ranking: R code

Contents

1 Introduction

Global Sensitivity Analysis (SA) is the study of how the uncertainty in a given model output is apportioned to the uncertainty in the model inputs (Sobol' and Kucherenko 2005; Saltelli 2002b). Over the last two decades, Sobol' variance-based sensitivity indices have become a method of choice for practitioners. These methods typically require a large number of function evaluations to achieve acceptable convergence and can become impractical for large engineering problems. Although the efficiency of the computation of first-order effects (S_i) has improved significantly (Saltelli 2002a; Sobol' and Myshetskaya 2008; Owen 2013; Kucherenko and Song 2017), fewer advances have been made in the case of total-order (T_i) indices (Sobol' 2001; Saltelli et al. 2010; Kucherenko et al. 2015; Piano et al. 2017).

Here we propose to increase the efficiency of the computation of T_i by monitoring their ranking. Being discrete variables, ranks may converge significantly quicker than the very sensitivity indices. Our approach is inspired by the ideas presented by Kreinin and Iscoe (2018), and also takes advantage of the fact that T_i converges faster than S_i: this is because the estimate for S_i depends upon elementary effects, related to output function values f as products $ff' - f_0^2$, where only one factor X_i is kept fixed between f and f', and f_0 is the sample output mean. Instead, for the estimation of the total sensitivity index T_i, we use differences $f - f'$, but in this case between f and f' all values are fixed but one (Saltelli et al. 2010).

An elementary effect for S_i is therefore hardly ever zero, as f and f' are generally different, and many f and f' need to be averaged to show that, once f_0^2 is substracted, the effect is small or zero. However, an elementary effect for T_i is normally different from zero since only one factor has been shifted. Therefore, if there is a difference between f and f', this can only be attributed to the moved parameter X_i. If there is no difference when the factor is influential it is just because the step separating f from f' is too small or the step is across a point of non-monotonicity.

2 Materials and methods

Hereafter we present the `R` code upon which our paper is based.

2.1 Reproducibility

Let us first create a wrapper function that allows to load all the required `R` libraries in one go: `data.table` (Dowle and Srinivasan 2019), `ggplot2` (Wickham 2016), `sensobol` (Puy 2019a), `scales` (Wickham 2018), `parallel` (R Core Team 2019), `cowplot` (Wilke 2019), `gridExtra`(Auguie 2017), `sensitivity` (Iooss, Janon, and Pujol 2019), `wesanderson` (Ram and Wickham 2018) and `RColorBrewer` (Neuwirth 2014). We then load the package `checkpoint` (Microsoft Corporation 2018), which installs in a local directory the same package versions used in the study. This allows anyone that runs our code to fully reproduce our results anytime.

```
# LOAD PACKAGES ------------------------------------------------------------

# 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)
    }
  }
}

loadPackages(c("data.table", "ggplot2", "sensobol", "scales", "parallel", "grid",
               "cowplot", "gridExtra", "sensitivity", "wesanderson", "RColorBrewer",
               "tikzDevice"))

# SET CHECKPOINT ----------------------------------------------------------

#dir.create(".checkpoint")

#library("checkpoint")

#checkpoint("2019-08-28",
           #R.version ="3.6.1",
           #checkpointLocation = getwd())

# CUSTOM FUNCTION TO DEFINE THE PLOT THEMES -------------------------------

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))
}
```

2.2 Functions

2.2.1 Test functions

Here we define the test functions that we will use later on to benchmark our approach against the normal computation procedure: the Sobol' G (Sobol' 1993), the Bratley, Fox, and Niederreiter (1992), the Oakley and O'Hagan (2004) and the Morris (1991) functions.

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

# Sobol G' function
G_Fun <- function(X) {
  a <- c(0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5)
  y <- 1
  for (j in 1:8) {
    y <- y * (abs(4 * X[, j] - 2) + a[j])/(1 + a[j])
  }
```

```
  return(y)
}

# The function by Bratley et al. (1992) is defined in the sensobol package
# (Function A1 in Kucherenko et al. 2011)

# The function by Oakley and O'Hagan is defined in the sensobol package

# The function by Morris 1991 is included in the sensitivity package
```

2.2.2 Sample matrices

The following code chunk allows to create r replicas of the $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{A}_{\mathbf{B}}^{i}$ matrices using Sobol' quasi-random number sequences (Bratley and Fox 1988). In this report we will stick with just one replica to save model runs in the model execution stage.

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

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)
}

scrambled_sobol <- function(A, B) {
  X <- rbind(A, B)
  for(i in 1:ncol(A)) {
    AB <- A
    AB[, i] <- B[, i]
    X <- rbind(X, AB)
  }
  AB <- X
  return(AB)
}

scrambled_replicas <- function(N, k, version) {
  X <- A <- B <- out <- list()
  df <- randtoolbox::sobol(n = N * version, dim = k * 2)
  indices <- CutBySize(nrow(df), nb = version)
  for(i in 1:nrow(indices)) {
    lower <- indices[i, "lower"]
    upper <- indices[i, "upper"]
    X[[i]] <- df[lower:upper, ]
  }
  for(i in seq_along(X)) {
    A[[i]] <- X[[i]][, 1:k]
```

```
    B[[i]] <- X[[i]][, (k + 1) : (k * 2)]
  }
  for(i in seq_along(A)) {
    out[[i]] <- scrambled_sobol(A[[i]], B[[i]])
  }
  return(out)
}
```

2.2.3 Sobol' indices

We use the estimators by Saltelli et al. (2010) and Jansen (1999) for S_i and T_i respectively.

The following code has been adapted from Puy (2019b). The setting `boot = TRUE` in `sobol_compute` allows the function to randomly sample with replacement the row indices of the $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{A}_{\mathbf{B}}^{i}$ matrices, and compute S_i and T_i in the resulting sample. This setting is required to create $2n$ bootstrap replicas of the ranks during the execution of the `smart_rank` algorithm (see below). If `boot = FALSE`, `sobol_compute` directly calculates S_i and T_i on the original matrices, without bootstrapping. Finally, the function returns the sensitivity indices or the ranks depending on whether `ranks = FALSE` or `ranks = TRUE` respectively.

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

sobol_computeS <- function(Y_A, Y_B, Y_AB) {
  n <- length(Y_A[!is.na(Y_A)])
  f0 <- (1 / (2 * n)) * sum(Y_A + Y_B, na.rm = TRUE)
  VY <- 1 / n * sum((Y_A - f0) ^ 2, na.rm = TRUE)
  Si <- (1 / n) * sum(Y_B * (Y_AB - Y_A), na.rm = TRUE) / VY
  STi <- ((1 / (2 * n)) * sum((Y_A - Y_AB) ^ 2, na.rm = TRUE)) / VY
  return(c(Si, STi))
}

sobol_MapplyS <- function(d) {
  return(mapply(sobol_computeS,
                d[, "Y_A"],
                d[, "Y_B"],
                d[, "Y_AB"]))
}

sobol_compute <- function(Y, params, boot = FALSE, ranks = FALSE) {
  k <- length(params)
  p <- length(1:(length(Y) / (k + 2)))
  Y_A <- Y[1:p]
  Y_B <- Y[(p + 1) : (2 * p)]
  Y_AB <- Y[(2 * p + 1):((length(Y) / (k + 2)) * (k + 2))]
  parameters <- rep(params, each = length(Y_A))
  vec <- cbind(Y_A, Y_B, Y_AB)
  out <- data.table::data.table(vec, parameters)
  if(boot == TRUE) {
```

```
    indices <- out[, sample(.I, replace = TRUE), parameters][, V1]
    output <- out[indices, sobol_MapplyS(.SD), by = parameters][
      , sensitivity:= rep(c("Si", "STi"), times = k)]
  }
  if(boot == FALSE) {
    output <- out[, sobol_MapplyS(.SD), by = parameters][
      , sensitivity:= rep(c("Si", "STi"), times = k)]
  }
  if(ranks == FALSE) {
    final <- output[, V1, sensitivity][, V1]
  }
  if(ranks == TRUE) {
    final <- output[, rank(-V1), sensitivity][, V1]
  }
  return(final)
}
```

2.3 The new algorithm

Here we describe the new algorithm to increase the efficiency of the computation of T_i. Its behaviour is graphically represented in Fig. 1. It works as follows:

1. It runs sequentially at sample sizes $N = 2^2, 2^3, ..., 2^m$, where m is defined by the user. After constructing a $(N, 2k)$ Sobol' matrix, for $i = 1, 2, ...k$ parameters, it computes the model output, the sensitivity indices S_{Ti} and T_i and the ranks of the parameters T_{Ri}.
2. At each sample size $N \neq 2^2$, it checks the following statement:
   (a) Condition 1: $T_{Ri}^{(N)}, T_{Rj}^{(N)}, ... \equiv T_{Ri}^{(N-1)}, T_{Rj}^{(N-1)}, ...$
3. If true, the algorithm considers that the ranks have converged and computes the set-sensitivity of the parameters with $T_i < 0.05$. The total cost C' of the analysis at convergence is therefore $C' = N'(k+3)$, where N' is the size of the sample matrix at convergence. At rank convergence, and for the next sample sizes, the algorithm computes the T_i of the influential parameters only, as well as the T_s, leading to $(N - N')(k - k' + 3)$ model runs, where k' is the number of non-influential parameters. Overall, the total cost C is $C = N'(k+3) + (N - N')(k - k' + 3) = N(k - k' + 3) + N'k'$.
4. If $T_{Ri}^{(N)}, T_{Rj}^{(N)}, ... \neq T_{Ri}^{(N-1)}, T_{Rj}^{(N-1)}, ...$, the algorithm checks for two more conditions before considering that the ranks have not converged, and that is necessary to increase the sample size. This is to ensure that the lack of convergence at N' is not caused by small scillations betwen non-important parameters, i.e. with $T_i < 0.05$, which are negligible for ranking purposes.
   (a) Condition 2: let $T_i^{(N)}, T_j^{(N)}, ...$, and $T_i^{(N-1)}, T_j^{(N-1)}, ...$ be the total sensitivity indices of the parameters whose ranks at N and $N - 1$ do not converge. The algorithm then checks for the following condition: $|(T_i^{(N)} - T_j^{(N)} - ...)|/(T_i^{(N)} + T_j^{(N)} + ...) < 0.1$, and $|(T_i^{(N-1)} - T_j^{(N-1)} - ...)|/(T_i^{(N-1)} + T_j^{(N-1)} + ...) < 0.1$. If the condition is true for both vectors, the algorithm considers that the oscillation of ranks is negligible from a factor fixing standpoint. The ranks have thus converged, and it applies the computations

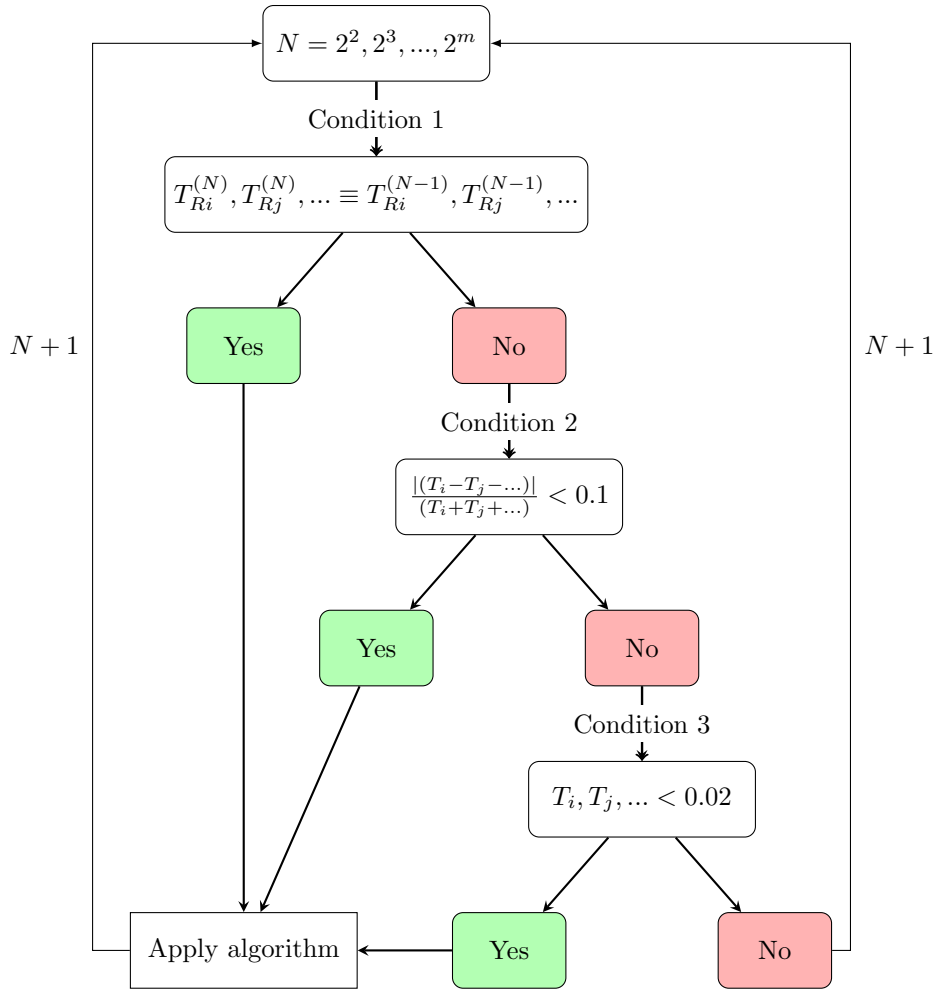

Figure 1: Diagram of the new algorithm

described in point 3 above. If the condition is false for at least one vector, it checks condition 3.

(b) Condition 3: the algorithm checks whether $T_i^{(N)}, T_j^{(N)}, ... < 0.02$ and $T_i^{(N-1)}, T_j^{(N-1)}, ... < 0.02$. If true for both vectors, it considers that the swapping of ranks is due to insignificant oscillations between parameters – the ranks have thus converged, and applies the computations described in point 3 above. If false, it definetely considers that the sample size is not enough to guarantee a robust ranking, and the algorithm jumps to the next sample size.

```
# DEFINE ALGORITHM ------------------------------------------------------------

# Compute normal
compute_normal <- function(N, params, test_F) {
  A <- sobol_matrices(n = N, k = length(params))
  Y <- test_F(A)
  indices <- sobol_compute(Y, params = params)
  ranks <- sobol_compute(Y, params = params, ranks = TRUE)
```

```
  dt <- data.table(cbind(indices, ranks))
  dt[, parameters:= rep(params, times = 2)][
    , sensitivity:= rep(c("Si", "STi"), each = length(params))][
    , method:= "Normal approach"][
    , N:= N][
    , model.runs:= N * (length(params) + 2)]
  return(dt)
}

# Algorithm to save runs
compute_saving <- function(dt, params, N, test_F, eps = 0.05) {
  dt2 <- copy(dt)
  set <- dt[sensitivity == "STi" & indices <= 0.05][, parameters]
  if(length(set) > 1) {
    Ti.set <- match(set, params)
    Ti.important <- match(setdiff(params, set), params)
    A.important <- sobol_matrices(N, k = length(params), cluster = Ti.important)
    A.set <- sobol_matrices(N, k = length(params), cluster = list(Ti.set))
    A.full <- rbind(A.important, A.set[(2 * N + 1):nrow(A.set), ])
    Y <- test_F(A.full)
    indices <- sobol_compute(Y, params = c(params[Ti.important], "set"))
    ranks <- sobol_compute(Y, params = c(params[Ti.important], "set"), ranks = TRUE)
    dt <- data.table(cbind(indices, ranks))
    dt[, parameters:= rep(c(params[Ti.important], "set"), times = 2)][
      , sensitivity:= rep(c("Si", "STi"), each = length(c(params[Ti.important], "set")))][
      , method:= "New approach"][
      , N:= N][
      , model.runs:= N * (length(params) - length(set) + 3)]
    final <- rbind(dt2, dt)
  } else {
    final <- dt
  }
  return(final)
}

# Full algorithm
full_algorithm <- function(max.exponent, params, test_F) {
  N <- sapply(2:max.exponent, function(x) 2 ^ x)
  dt <- list()
  for(i in seq_along(N)) {
    dt[[i]] <- compute_normal(N[i], params = params, test_F = test_F)
    if(i > 1) {
      a <- dt[[i]][sensitivity == "STi" & method == "Normal approach"]
      b <- dt[[i-1]][sensitivity == "STi" & method == "Normal approach"]
      # CHECK CONDITION 1---------------------------------
      condition1 <- identical(a[, ranks], b[, ranks])
      # CHECK CONDITION 2---------------------------------
```

```
      ind <- which(!a[, ranks] == b[, ranks])
      condition2 <- condition2 <- all(abs(a[ind][, indices] - b[ind][, indices]) /
                          (a[ind][, indices] + b[ind][, indices]) < 0.1)
      # CHECK CONDITION 3 ---------------------------------
      condition3 <- all(c(a[ind][, indices], b[ind][, indices]) < 0.02)
      if(condition1 == TRUE |
         condition1 == FALSE & condition2 == TRUE |
         condition1 == FALSE & condition2 == FALSE & condition3 == TRUE) {
        dt[[i]] <- compute_saving(dt[[i]], params = params, N = N[i], test_F = test_F)
      } else {
        dt[[i]] <- compute_normal(N[i], params = params, test_F = test_F)
      }
    }
  }
  # Compute total number of model runs for the new algorithm
  final <- rbindlist(dt)
  N_convergence <- final[method == "New approach" & sensitivity == "STi", min(N)]
  a <- final[N == N_convergence & method == "Normal approach"
             & sensitivity == "STi"][, parameters]
  b <- final[N == N_convergence & !parameters == "set" &
               method == "New approach" & sensitivity == "STi"][, parameters]
  k_noninfluential <- length(setdiff(a, b))
  final[, model.runs:= ifelse(method %in% "New approach",
                              model.runs + N_convergence * k_noninfluential,
                              model.runs)]
  return(final)
}
```

We test the performance of the algorithm with the Sobol' G (Sobol' 1993), the Bratley, Fox, and Niederreiter (1992) the Oakley and O'Hagan (2004) and the Morris (1991) functions at each N, for $N = 2^2, 2^3, ..., 2^{16}$.

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

max.exponent <- 16
test_functions <- c("G_Fun", "bratley1992_Fun", "oakley_Fun", "morris_Fun")
out <- list()

for(i in test_functions) {
  if(i == "G_Fun") {
    params <- paste("X", 1:8, sep = "")
    test_F <- G_Fun
  } else if(i == "bratley1992_Fun") {
    params <- paste("X", 1:8, sep = "")
    test_F <- bratley1992_Fun
  } else if(i == "oakley_Fun") {
    params <- paste("X", 1:15, sep = "")
    test_F <- oakley_Fun
```

```
  } else {
    params <- paste("X", 1:20, sep = "")
    test_F <- sensitivity::morris.fun
  }
  out[[i]] <- full_algorithm(max.exponent = max.exponent,
                             params = params,
                             test_F = test_F)
}
```

We now arrange the results for better plotting:

```
# ARRANGE RESULTS ------------------------------------------------------------

# Arrange results
model_names <- c("Sobol' G", "Bratley et al. 1994",
                 "Oakley and O'Hagan 2004", "Morris 1991")
names(out) <- model_names
results <- rbindlist(out, idcol = "model") %>%
  .[, model:= factor(model, levels = model_names)] %>%
  .[, parameters:= factor(parameters,
                          levels = c(paste("X", 1:20, sep = ""), "set"))] %>%
  .[, method:= ifelse(method %in% "New approach", "New.approach", "Normal.approach")]

# Compare number of model runs and savings
tmp <- results[, .(model.runs = unique(model.runs)), .(model, N, method)]
savings.dt <- dcast(tmp, model + N ~ method, value.var = "model.runs") %>%
  .[, New.approach:= ifelse(New.approach %in% NA, Normal.approach, New.approach)] %>%
  .[, saving:= 1 - (New.approach / Normal.approach)]

# Compute cumulative number of runs
cumulative.runs <- savings.dt[, cumsum(.SD),
                              .SDcols = c("New.approach", "Normal.approach"), model][
                                , saving:= 1 - (New.approach / Normal.approach)]
```

```
# EXPORT MODEL RUNS ----------------------------------------------------------

fwrite(results, "results.csv")
fwrite(savings.dt, "savings.dt.csv")
```

3 Results

Fig. 2a shows the evolution of the T_i and the T_s along different sample sizes. The onset of the lines marks the sample size at which our algorithm kicks in, i.e. the sample size at which the ranks, according to the conditions set in the previous chapter, are considered converged. The first dot thus indicates the smallest sample size needed to obtain a robust screening of the parameters: note that, once the ranks converge, they do not "disconverge" and the algorithm is activated at every sample size. The computation of the T_s shows that grouping non-important factors might not have a dramatic effect in the model output if they are fixed to reduce model complexity. All $T_s \approx 0.05$

except in the case of the Oakley and O'Hagan (2004) function, for which $T_s \approx 0.15$.

It is also worth stressing that the cardinality of the T_s, i.e. the number of elements in the set, is constant throughout the range of sample sizes tested (Fig. 2b), and that the T_s always includes the same set of parameters regardless of the sample size used (not shown in the plot but checked in `R`). This suggests that, once the algorithm kicks in, there is no need to increase the sample size in order to achieve more precise results for a factor fixing setting. However, some influential parameters with very similar T_i values might still swap ranks, and a larger sample size might be needed to ensure their full stabilization.

Under this scenario, our algorithm allows to search for a full robust ranking of all parameters through different sample sizes while saving a significant number of model runs in the process. This is shown in Fig. 2c: the x axis presents the total number of model runs once the algorithm is activated, whereas the y axis the number of model runs saved as a percentage over the total $N(k+2)$. Note that there is a drop in the savings at rank convergence compared to the traditional approach: this is because the algorithm adds N' model runs to compute the T_s. Once converged, the total number of runs is $N(k-k'+3)+N'k'$, which yields to significant savings in the computation cost, i.e. up to 40% in the case of the Bratley, Fox, and Niederreiter (1992) and the Morris (1991) functions.

4 Preliminary conclusions

- Our algorithm finds the minimum sample size needed to ensure a robust screening of parameters. This will allow researchers to save computer time when searching for an ideal sample size to ensure consistent results in a factor fixing setting.
- If what is aimed at is at ensuring full rank convergence, our algorithm allows to explore different sample sizes while providing significant and incremental benefits in terms of savings over the total cumulative number of runs.
- A necessary pre-condition for the algorithm to allow to save model runs is the existence of at least two parameters with $T_i < 0.05$. This is not uncommon in real case studies, where sensitivity indices tend to follow a Pareto distribution (i.e. the indices of 80% of the parameters are responsible for less than 20% of the variance and those of the remainder form more than 80%).
- It should be noted that the results are fully dependent on some arbitrarily selected values: 1) 0.05 as a threshold to distinguish influential from non-influential parameters, 2) 0.1 to define Sergei's 'condition 2' (see code in page 13 and 14), and 3) 0.02 in 'condition 3' (see code in page 14). The value of 0.05 might be the most disputed choice as other thresholds and screening methods have been proposed in the literature.

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

results <- results[, Type:= ifelse(parameters %in% "set", "Set", "Individual")]

formatter <- function(...) function(x) format(round(x, 2), ...)

results[!sensitivity == "Si" & method == "New.approach"] %>%
  ggplot(., aes(N, indices,
                color = Type,
```

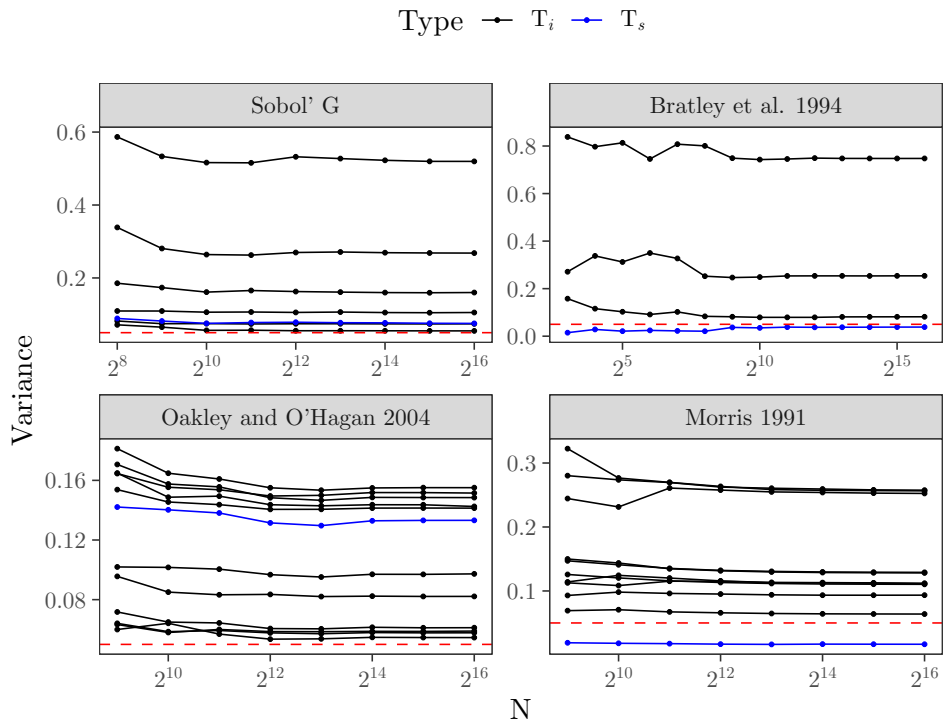

Figure 2: Evolution of the T_i and the T_s along different sample sizes. The horizontal, red dotted line is at 0.05.

```
                group = parameters)) +
  geom_line() +
  scale_x_continuous(trans="log",
                     breaks = trans_breaks("log2", function(x) 2 ^ x),
                     labels = trans_format("log2", math_format(2^.x))) +
  geom_point(size = 0.5) +
  scale_color_manual(values = c("black", "blue"),
                     labels = c(expression(T[italic(i)]),
                                expression(T[italic(s)]))) +
  labs(x = "N",
       y = "Variance") +
  geom_hline(yintercept = 0.05,
             lty = 2,
             color = "red") +
  facet_wrap(~model,
             ncol = 2,
             scales = "free") +
  theme_AP() +
  theme(legend.position = "top")
```

```
# PLOT SAVINGS ------------------------------------------------------------

melt(savings.dt, measure.vars = c("New.approach", "Normal.approach")) %>%
  .[, saving:= saving * 100] %>%
  setnames(., "model", "Model") %>%
```

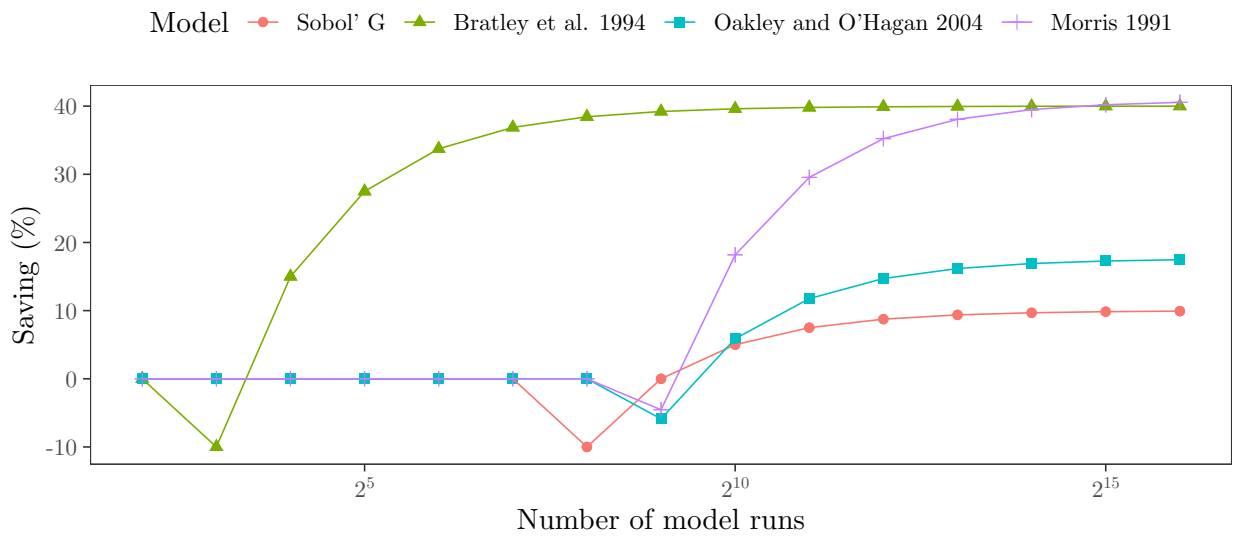

Figure 3: Evolution of ranks for first and total-order indices across different base sample sizes

```
  ggplot(., aes(N, saving, color = Model, shape = Model, group = Model)) +
  geom_point() +
  geom_line() +
  scale_x_continuous(trans="log",
                     breaks = trans_breaks("log2", function(x) 2 ^ x),
                     labels = trans_format("log2", math_format(2^.x))) +
  labs(x = "Number of model runs",
       y = "Saving (\\%)") +
  theme_AP() +
  theme(legend.position = "top") +
  guides(color = guide_legend(nrow = 1,byrow = TRUE))
```

```
# PLOT RANKS ------------------------------------------------------------------

results[method == "Normal.approach"] %>%
  .[, sensitivity:= ifelse(sensitivity %in% "Si", "$S_i$", "$T_i$")] %>%
  ggplot(., aes(N, ranks, group = parameters,
                color = parameters)) +
  geom_point(size = 0.5) +
  geom_line() +
  labs(x = "Base sample size",
       y = "Rank") +
  scale_color_discrete(name = "Parameters") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x))) +
  facet_grid(model ~ sensitivity,
             scales = "free_y") +
  theme_bw() +
  theme(legend.position = "top",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
```

```
        legend.background = element_rect(fill = "transparent",
                                         color = NA),
        legend.key = element_rect(fill = "transparent",
                                  color = NA))
```

5 Computation of the MAE

```
# COMPUTATION OF MEAN ABSOLUTE ERROR ------------------------------------------

# The analytical values for the Oakley and O'Hagan function
dt.analytics <- data.table(parameters = paste("X", 1:15, sep = ""),
                           analytical = c(0.059, 0.063, 0.036, 0.055, 0.024, 0.041,
                                          0.058, 0.082, 0.097, 0.036, 0.151, 0.148,
                                          0.142, 0.141, 0.155))

# Select rows
AE.test <- results[model == "Oakley and O'Hagan 2004" &
                     sensitivity == "STi" &
                     N >= 512]

# Compute the Absolute Error (AE) for the non-clustered parameters
# Compute the Absolute Error (AE) for the non-clustered parameters
AE.ns <- merge(AE.test, dt.analytics, by = "parameters", all.x = TRUE) %>%
  .[, AE_ns:= abs(indices - analytical)]

# Extract vector with the clustered parameters
pars <- AE.ns[method =="Normal.approach" &
                indices < 0.05][, unique(parameters)]

# Compute the sum of the analytical values of the clustered parameters
tmp <- AE.ns[parameters %in% pars][, .(tmp = sum(indices)), .(N, model.runs)]

# Compute the AE for the set of clustered parameters (AE.s)
AE.s <- AE.ns[parameters == "set"] %>%
  .[tmp, on = "N"] %>%
  .[, .(AE_s = abs(indices - tmp)), .(N, model.runs)]

# Compute MAE, MAE_s, and MAE average
full.AE <- AE.ns[AE.s, on = "N"] %>%
  .[, .(MAE_S = (sum(AE_ns, na.rm = TRUE) + AE_s) / 15,
        MAE = mean(abs(indices - analytical), na.rm = TRUE)), .(N, model.runs, method)] %>%
  .[, average.MAE:= (MAE_S + MAE) / 15]

# Plot results
full.AE <- setnames(full.AE,
                    c("MAE_S", "MAE", "average.MAE"),
```

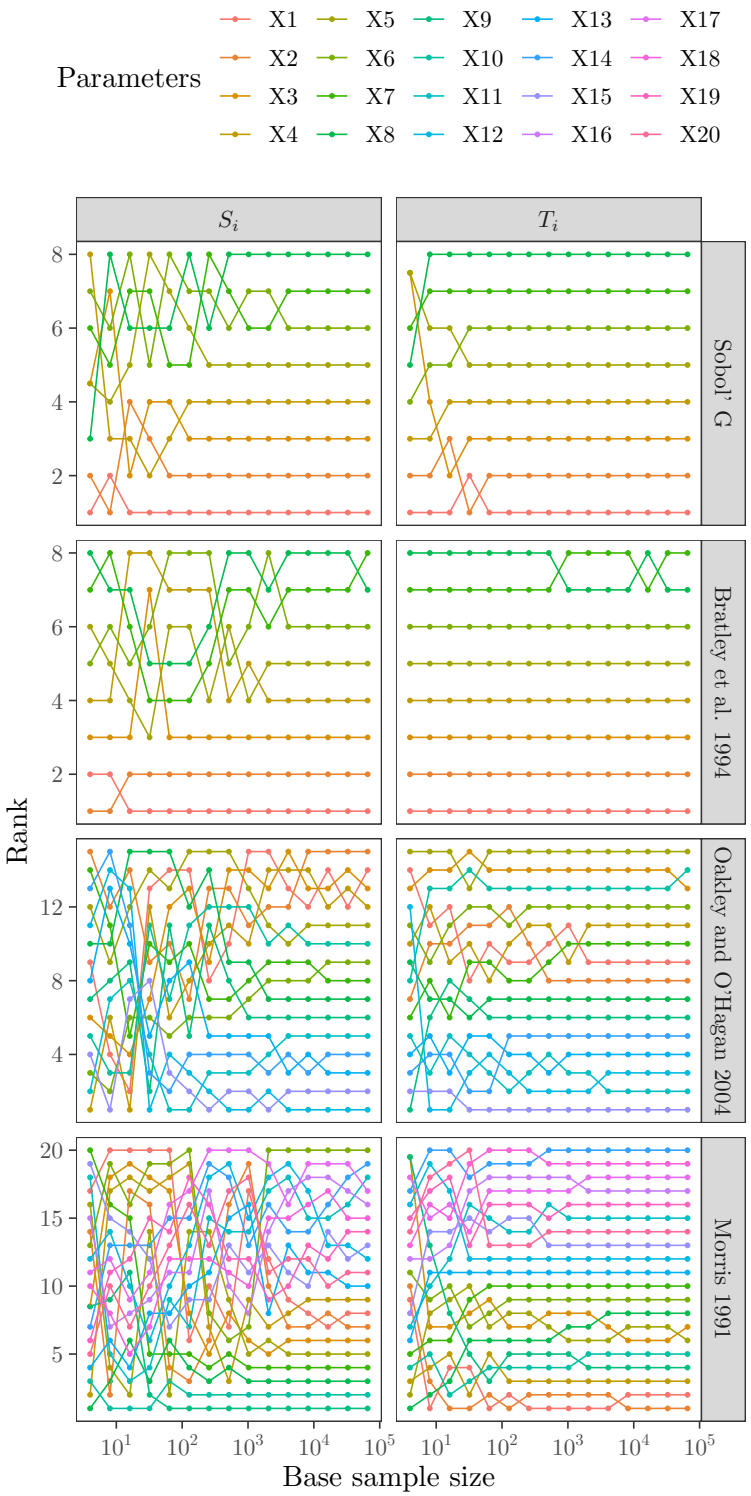

Figure 4: Percentage of saving over the total accumulated model runs.

```
                    c("$T_s$", "$T_i$", "$(T_i + T_s) / k$"))

melt(full.AE, measure.vars = c("$T_s$", "$T_i$", "$(T_i + T_s) / k$")) %>%
  ggplot(., aes(model.runs, value, color = method)) +
  geom_point() +
  geom_line() +
  labs(x = "Number of model runs",
       y = "MAE") +
  scale_x_continuous(trans="log",
                     breaks = trans_breaks("log2", function(x) 2 ^ x),
                     labels = trans_format("log2", math_format(2^.x))) +
  scale_color_discrete(name = "Method",
                       labels = c("New approach", "Traditional approach")) +
  facet_wrap(~ variable,
             ncol = 1,
             scales = "free_y") +
  theme_AP() +
  theme(legend.position = "top") +
  guides(color = guide_legend(nrow = 2, byrow = TRUE))
```

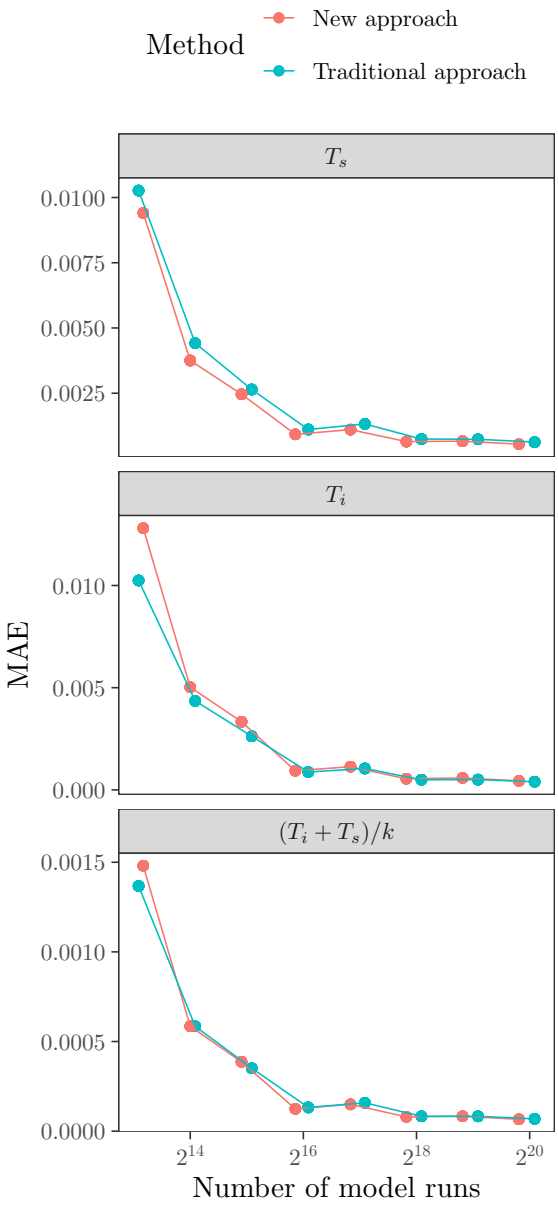

Figure 5: Mean Absolute Error for the Oakley and O'Hagan function.

6 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 10.15.1
##
## 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] grid      parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] tikzDevice_0.12.3  RColorBrewer_1.1-2 wesanderson_0.3.6
##  [4] sensitivity_1.16.2 gridExtra_2.3      cowplot_1.0.0
##  [7] scales_1.0.0       sensobol_0.2.1     ggplot2_3.2.1
## [10] data.table_1.12.2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.2       highr_0.8        pillar_1.4.2     compiler_3.6.1
##  [5] tools_3.6.1      boot_1.3-23      digest_0.6.21    evaluate_0.14
##  [9] tibble_2.1.3     gtable_0.3.0     pkgconfig_2.0.3  rlang_0.4.0
## [13] filehash_2.4-2   bibtex_0.4.2     yaml_2.2.0       xfun_0.9
## [17] withr_2.1.2      dplyr_0.8.3      stringr_1.4.0    knitr_1.25
## [21] gbRd_0.4-11      tidyselect_0.2.5 glue_1.3.1       R6_2.4.0
## [25] Rdpack_0.11-0    rmarkdown_1.15   purrr_0.3.2      magrittr_1.5
## [29] codetools_0.2-16 htmltools_0.4.0  assertthat_0.2.1 colorspace_1.4-1
## [33] labeling_0.3     tinytex_0.16     stringi_1.4.3    lazyeval_0.2.2
## [37] munsell_0.5.0    crayon_1.3.4
```

References

Auguie, B. 2017. “gridExtra: Miscellaneous Functions for "Grid" Graphics.”

Bratley, P., and B. L. Fox. 1988. “ALGORITHM 659: implementing Sobol’s quasirandom sequence generator.” *ACM Transactions on Mathematical Software (TOMS)* 14 (1): 88–100.

Bratley, P., B. L. Fox, and H. Niederreiter. 1992. “Implementation and tests of low-discrepancy sequences.” *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2 (3): 195–213.

Dowle, M., and A. Srinivasan. 2019. “data.table: Extension of ‘data.frame‘.”

Iooss, B., A. Janon, and G. Pujol. 2019. “sensitivity: Global Sensitivity Analysis of Model Outputs.” https://cran.r-project.org/package=sensitivity.

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.

Kreinin, A., and A. Iscoe. 2018. “Sensitivity ranks by Monte Carlo.” In *MCQMC*. Rennes.

Kucherenko, S., B. Delpuech, B. Iooss, and S. Tarantola. 2015. “Application of the control variate technique to estimation of total sensitivity indices.” *Reliability Engineering and System Safety* 134 (July): 251–59. https://doi.org/10.1016/j.ress.2014.07.008.

Kucherenko, S., and S. Song. 2017. “Different numerical estimators for main effect global sensitivity indices.” *Reliability Engineering and System Safety* 165: 222–38. https://doi.org/10.1016/j.ress.2017.04.003.

Microsoft Corporation. 2018. “checkpoint: Install Packages from Snapshots on the Checkpoint Server for Reproducibility.” https://cran.r-project.org/package=checkpoint.

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

Neuwirth, E. 2014. “RColorBrewer: ColorBrewer Palettes.”

Oakley, J.E., and A. O’Hagan. 2004. “Probabilistic sensitivity analysis of complex models: a Bayesian approach.” *Journal of the Royal Statistical Society B* 66 (3): 751–69. https://doi.org/10.1111/j.1467-9868.2004.05304.x.

Owen, A. B. 2013. “Better estimation of small sobol’ sensitivity indices.” *ACM Transactions on Modeling and Computer Simulation* 23 (2): 1–17. https://doi.org/10.1145/2457459.2457460.

Piano, S. Lo, F. Ferretti, A. Puy, D. Albrecht, S. Tarantola, and A. Saltelli. 2017. “Is it possible to improve existing sample-based algorithm to compute the total sensitivity index?” *Reliability Engineering & System Safety*, March. http://arxiv.org/abs/1703.05799.

Puy, A. 2019a. “sensobol: Computation of High-Order Sobol’ Sensitivity Indices.”

———. 2019b. “sensobol: Computation of High-Order Sobol’ Sensitivity Indices.” R package. http://github.com/arnaldpuy/sensobol.

Ram, K., and H. Wickham. 2018. “wesanderson: A Wes Anderson Palette Generator.”

R Core Team. 2019. “R: A language and environment for statistical computing. R Foundation for Statistical Computing.” Vienna.

Saltelli, A. 2002a. "Making best use of model valuations to compute sensitivity indices." *Computer Physics Communications* 145: 280–97. https://doi.org/10.1016/S0010-4655(02)00280-1.

———. 2002b. "Sensitivity analysis for importance assessment." *Risk Analysis* 22 (3): 579–90. https://doi.org/10.1111/0272-4332.00040.

Saltelli, A., P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola. 2010. "Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index." *Computer Physics Communications* 181 (2). Elsevier B.V.: 259–70. https://doi.org/10.1016/j.cpc.2009.09.018.

Sobol', I.M. 2001. "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates." *Mathematics and Computers in Simulation* 55 (1-3): 271–80. https://doi.org/10.1016/S0378-4754(00)00270-6.

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 S.S. Kucherenko. 2005. "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates." *Wilmott Magazine* 1: 56–61. https://doi.org/10.1016/S0378-4754(00)00270-6.

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.

Wickham, H. 2016. *Elegant Graphics for Data Analysis*. New York: Springer-Verlag.

———. 2018. "scales: Scale Functions for Visualization." https://cran.r-project.org/package=scales.

Wilke, C. O. 2019. "cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'."