The battle of total-order sensitivity estimators

R code

Arnald Puy

Contents

```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("Rcpp", "RcppArmadillo", "tidyverse", "parallel", "foreach",
               "doParallel", "Rfast", "data.table", "scales", "cowplot",
               "benchmarkme", "logitnorm", "sensobol", "ggrepel"))

# 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("2020-04-21",
           R.version ="3.6.3",
           checkpointLocation = getwd())
```

1 Functions

1.1 Savage scores

```
# 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)
}
```

1.2 Sensitivity indices

```
# VARS FUNCTIONS -----------------------------------------------------------------

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 if(method == "LHS") {
    mat <- lhs::randomLHS(star.centers, length(params))
  } 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] # Remove zeroes
    })
    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]])
```

```
  }
  return(do.call(rbind, out))
}

# 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)
}

# Function to compute VARS-TI
vars_ti <- function(Y, star.centers, params, h) {
  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"], ]
  }
  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])])
    }))
  out <- lapply(d, function(x) lapply(x, function(y) matrix(y, nrow = length(y) / 2, byrow = TR
  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)
}

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

sobol_Ti <- function(d, N, params, total) {
```

```
m <- matrix(d, nrow = N)
k <- length(params)
if(total == "jansen" | total == "homma" | total == "sobol" | total == "monod" |
   total == "glen") {
  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)
} else if(total == "glen") {
  Ti <- 1 - (1 / (N - 1) *
              Rfast::colsums(((Y_A - mean(Y_A)) * (Y_AB - Rfast::colmeans(Y_AB))) /
                               sqrt(var(Y_A) * Rfast::colVars(Y_AB))))
}
if(total == "azzini" | total == "lamboni") {
  Y_A <- m[, 1]
  Y_B <- m[, 2]
  Y_AB <- m[, 3:(3 + k - 1)]
  Y_BA <- m[, (ncol(m) - k + 1):ncol(m)]
  f0 <- 1 / (2 * N) * sum(Y_A + Y_B)
  VY <- 1 / (2 * N - 1) * sum((Y_A - f0) ^ 2 + (Y_B - f0) ^ 2)
}
if(total == "azzini") {
  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)))
} else if(total == "lamboni") {
  Ti <- (1 / (4 * N) * colSums((Y_A - Y_AB) ^ 2 + (Y_B - Y_BA) ^ 2, na.rm = TRUE)) / VY
}
if(total == "owen") {
  Y_A <- m[, 1]
  Y_B <- m[, 2]
  Y_BA <- m[, 3:(3 + k - 1)]
  Y_CB <- m[, (ncol(m) - k + 1):ncol(m)]
  VY <- sapply(1:k, function(j)
```

```
      mean(Rfast::rowmeans(m[,c(1, 2, 2 + j, 2 + j + k)]^ 2)) -
        mean(Rfast::rowmeans(m[,c(1, 2, 2 + j, 2 + j + k)])) ^2)
    Ti <- (VY - (1 / N * Rfast::colsums((Y_B - Y_CB) * (Y_BA - Y_A)))) / VY
  }
  output <- data.table(Ti)
  output[, `:=`(parameters = paste("X", 1:k, sep = ""))]
  return(output)
}
```

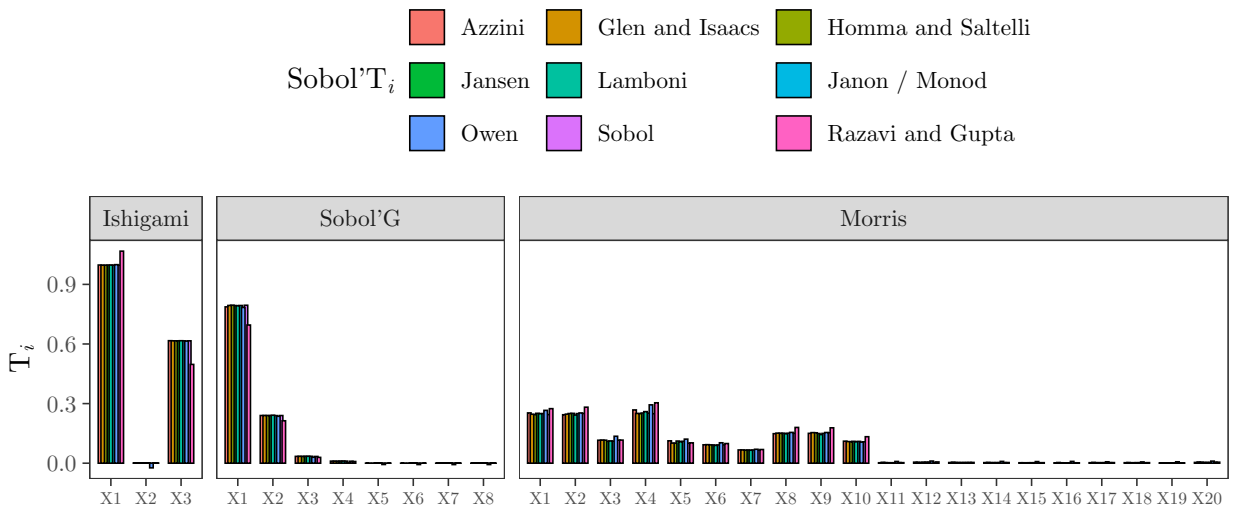
```
# CHECK THAT ALL TI ESTIMATORS WORK ---------------------------------------------

# Settings
estimators <- c("jansen", "sobol", "homma", "azzini", "monod", "lamboni", "glen", "owen")
test_functions <- c("Ishigami", "Sobol'G", "Morris")
N <- 2 ^ 11

# Run model
ind <- Y <- mt <- list()
for(i in estimators) {
  for(j in test_functions) {
    if(i == "jansen" | i == "sobol" | i == "homma" | i == "monod" | i == "glen") {
      matrices <- c("A", "AB")
    } else if(i == "azzini" | i == "lamboni"){
      matrices <- c("A", "B", "AB", "BA")
    } else if(i == "owen") {
      matrices <- c("A", "B", "BA", "CB")
    }
    if(j == "Ishigami") {
      k <- 3
      modelRun <- sensobol::ishigami_Fun
    } 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
```

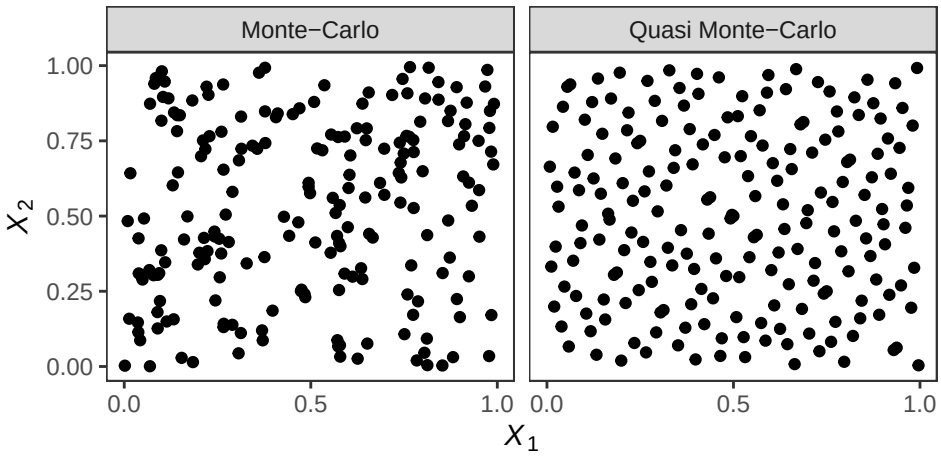
```r
# Run model for VARS
star.centers <- 200
h <- 0.2
vars.ind <- Y <- list()
for(j in test_functions) {
  if(j == "Ishigami") {
    k <- 3
    modelRun <- sensobol::ishigami_Fun
  } else if(j == "Sobol'G") {
    k <- 8
    modelRun <- sensobol::sobol_Fun
  } else if(j == "Morris") {
    k <- 20
    modelRun <- sensitivity::morris.fun
  }
  mt[[j]] <- vars_matrices(star.centers = star.centers,
                           params = paste("X", 1:k, sep = ""),
                           h = h)
  Y[[j]] <- modelRun(mt[[j]])
  vars.ind[[j]] <- vars_ti(Y = Y[[j]], star.centers = star.centers,
                           params = paste("X", 1:k, sep = ""), h = h)
}

vars.ind <- rbindlist(vars.ind, idcol = "Function")[
  , estimator:= "vars"] %>%
  setcolorder(., c("estimator", "Function", "Ti", "parameters"))
```

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

lapply(ind, function(x) rbindlist(x, idcol = "Function")) %>%
  rbindlist(., idcol = "estimator") %>%
  rbind(vars.ind) %>%
  .[, 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") +
  scale_fill_discrete(name = expression(paste("Sobol' ", T[italic(i)])),
                      labels = c("Azzini", "Glen and Isaacs", "Homma and Saltelli",
                                 "Jansen", "Lamboni", "Janon / Monod", "Owen",
                                 "Sobol", "Razavi and Gupta")) +
  facet_grid(~Function,
             scales = "free_x",
             space = "free_x") +
  labs(x = "",
       y = expression(T[italic(i)])) +
```

```
  theme_AP() +
  theme(axis.text.x = element_text(size = 6.5),
        legend.position = "top") +
  guides(fill = guide_legend(nrow = 3,
                             byrow = TRUE))
```




```
# PLOT THE SAMPLING METHODS AVAILABLE ----------------------------------------

method <- c("R", "QRN")
matrices <- "A"
N <- 200
set.seed(666) # For the random sampling
params <- paste("X", 1:2, sep = "")
A <- lapply(method, function(x)
  sobol_matrices(N = N, params = params,
                 matrices = matrices, method = x))

names(A) <- method
lapply(A, data.table) %>%
  rbindlist(., idcol = "method") %>%
  .[, method:= ifelse(method == "R", "Monte-Carlo", "Quasi Monte-Carlo")] %>%
  .[, method:= factor(method, levels = c("Monte-Carlo", "Quasi Monte-Carlo"))] %>%
  ggplot(., aes(X1, X2)) +
  geom_point() +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = expression(italic(X)[1]),
       y = expression(italic(X)[2])) +
  facet_wrap(~method, ncol = 3) +
  theme_AP()
```



2 The metafunction

```
# 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)
)
```

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

```
## Warning: `mapping` is not used by stat_function()
```

```
## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()

## Warning: `mapping` is not used by stat_function()
```

```
a
```

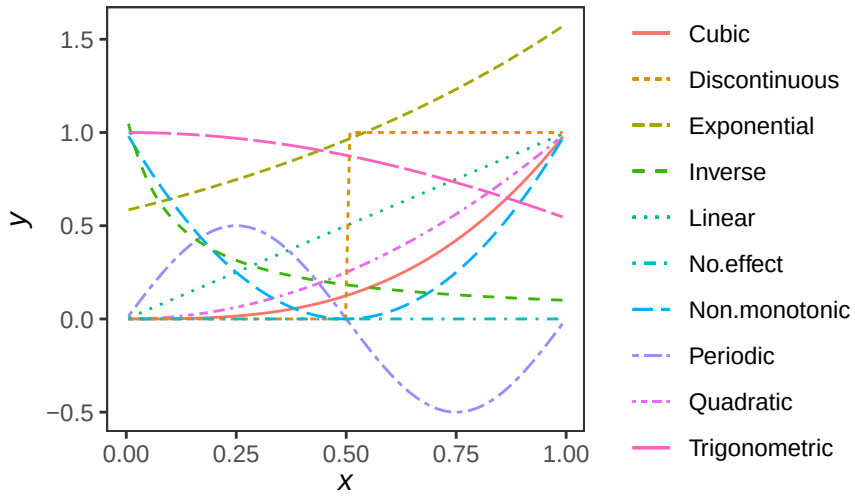


Figure 1: Functions used in the metafunction of Becker (2019).

```
# CREATE 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 <- 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 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("U(0, 1)",
                                  "N(0.5, 0.2)",
                                  "Beta(8, 2)",
                                  "Beta(2, 8)",
                                  "Beta(2, 0.5)",
                                  "Beta(0.5, 2)",
                                  "Logitnormal(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
# MERGE METAFUNCTION PLOT AND DISTRIBUTIONS PLOT ---------------------------------

plot_grid(b, a, ncol = 1, labels = "auto", align = "hv")
```

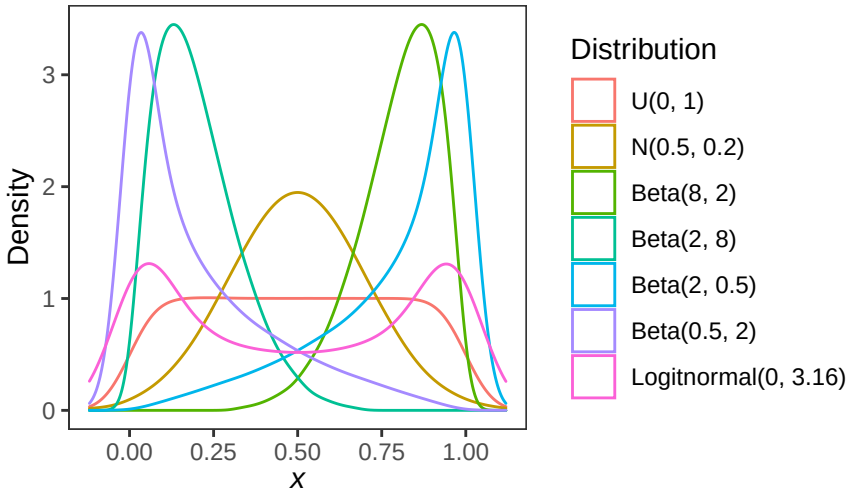
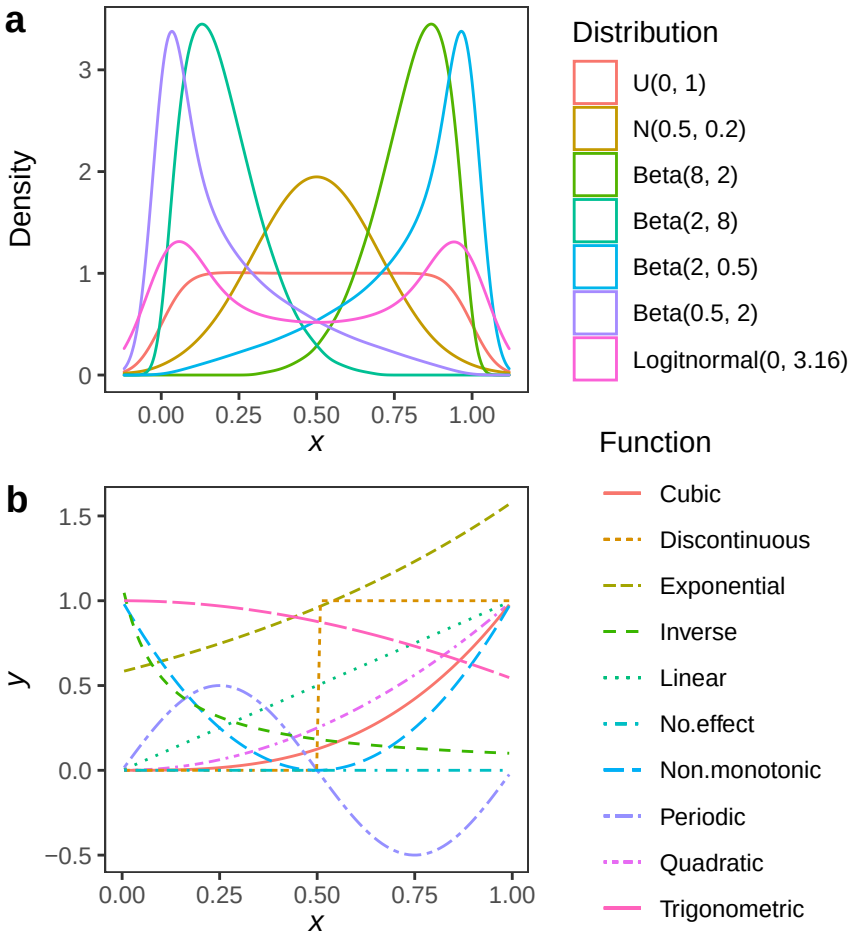
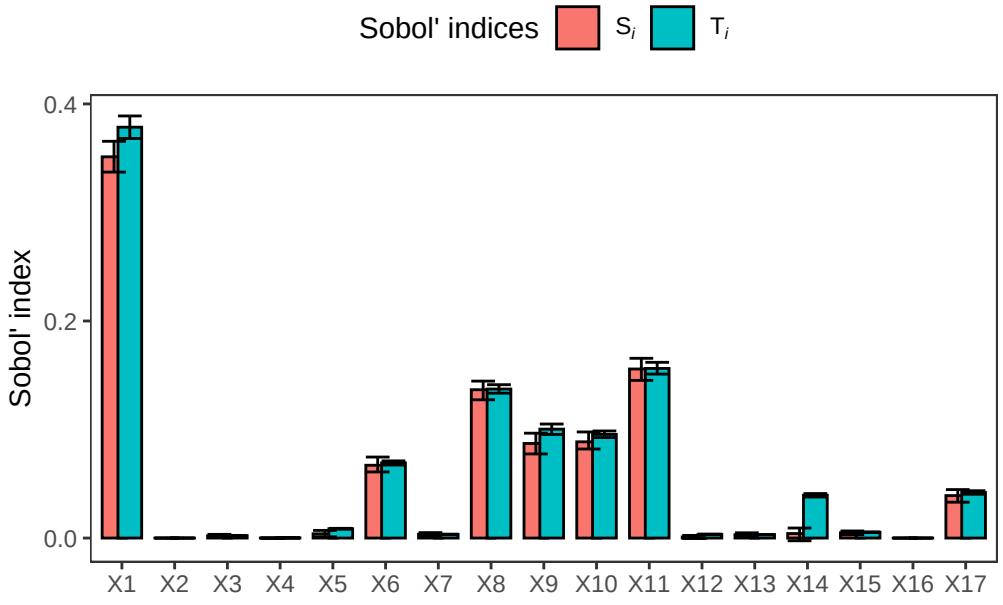


Figure 2: Distributions used in the metafunction of Becker (2019).




```r
# EXEMPLIFY THE METAFUNCTION WITH AN EXAMPLE ------------------------------------

N <- 10000 # Sample size
R <- 100 # Number of bootstrap replications
k <- 17 # Number of model inputs
```

```r
k_2 <- 0.5 # Fraction of active pairwise interactions
k_3 <- 0.2 # Fraction of active three-wise interactions
epsilon <- 666  # to reproduce the results
params <- paste("X", 1:k, sep = "")
A <- sobol_matrices(N = N, params = params)

# Compute
Y <- sensobol::metafunction(data = A, k_2 = k_2, k_3 = k_3, epsilon = epsilon)
indices <- data.table(sensobol::sobol_indices(Y = Y, N = N, params = params, boot = TRUE, R = 1
  .[, parameters:= factor(parameters, levels = paste("X", 1:k, sep = ""))]

ggplot(indices, 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_x_discrete(labels = ggplot2:::parse_safe) +
  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")
```



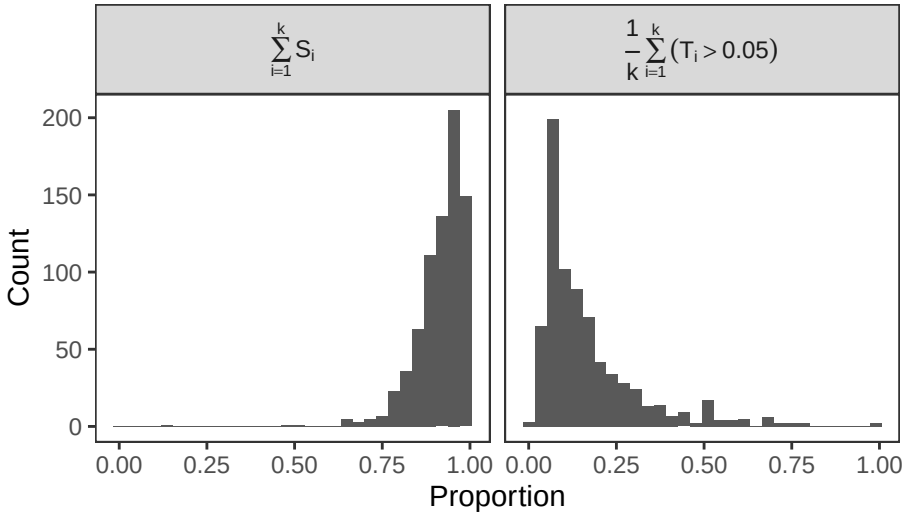

```r
# FUNCTIONS TO COMPUTE SUM OF SI AND PROPORTION OF SI > 0.05 FOR METAFUNCTION -----
```

```r
# Function to replicate in parallel
RepParallel <- function(n, expr, simplify = "array",...) {
  answer <-
    mclapply(integer(n), eval.parent(substitute(function(...) expr)),...)
  if (!identical(simplify, FALSE) && length(answer))
    return(simplify2array(answer, higher = (simplify == "array")))
  else return(answer)
}

# Function to replicate
try_metafunction <- function(x) {
  k <- sample(x)[1]
  params <- paste("X", 1:k, sep = "")
  N <- 500
  A <- sobol_matrices(N = N, params = params)
  Y <- sensobol::metafunction(A)
  ind <- sobol_indices(Y = Y, N = N, params = params)
  out1 <- ind[sensitivity == "Si", sum(original)]
  out2 <- ind[sensitivity == "Ti", sum(original > 0.05)/.N]
  output <- c(out1, out2)
  return(output)
}
```

```r
# COMPUTE SUM OF SI AND PROPORTION OF SI > 0.05 FOR METAFUNCTION --------------------

n <- 10^3
out <- do.call(rbind, RepParallel(n, try_metafunction(3:100), simplify = FALSE))
dt.out <- data.table(out)
```

```r
# PLOT

dt.out[V1 < 1 & V1 > 0] %>%
  setnames(., c("V1", "V2"),
           c("sum(S[i], i==1, k)",
             "frac(1,k)~sum((T[i]>0.05), i == 1, k)")) %>%
  melt(., measure.vars = c("sum(S[i], i==1, k)",
                           "frac(1,k)~sum((T[i]>0.05), i == 1, k)")) %>%
  ggplot(., aes(value)) +
  geom_histogram() +
  facet_wrap(~variable,
             labeller = label_parsed) +
  labs(x = "Proportion", y = "Count") +
  theme_AP()
```

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



3 The model

3.1 Settings

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

N <- 2 ^ 11 # Sample size of sample matrix
h <- 0.2 # step for VARS
R <- 500 # Number of bootstrap replicas
n_cores <- ceiling(detectCores() * 0.5)
order <- "second"
params <- c("k_2", "k_3", "epsilon", "phi", "delta", "tau")
N.high <- 2 ^ 11 # Maximum sample size of the large sample matrix
```

3.2 Sample matrix

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

# Main matrix without Nt and k
tmp <- sobol_matrices(N = N, params = c(params, "N_t", "k"), order = order, matrices = c("A",
A <- tmp[1:N, ]
B <- tmp[(N + 1):(2 * N), ]

first <- 1:length(params)
loop <- c(first, utils::combn(1:length(params), 2, simplify = FALSE))

X <- rbind(A, B)
for(i in loop) {
  AB <- A
  AB[, i] <- B[, i]
  X <- rbind(X, AB)
  }
```

```r
mat <- X

# Matrix with clusters (Nt,k)
AB <- AB2 <- AB3 <- A
AB[, c("N_t", "k")] <- B[, c("N_t", "k")] # (Nt,k)
AB2[, c("k_2", "k_3", "epsilon", "phi")] <-  B[, c("k_2", "k_3", "epsilon", "phi")] # Model un
AB3[, c("delta", "tau")] <- B[, c("delta", "tau")] # Manageable uncertainties

# Final matrix
mat <- rbind(mat, AB, AB2, AB3)

# TRANSFORM MATRIX --------

mat[, 1] <- round(qunif(mat[, 1], 0.3, 0.5), 2) # k_2
mat[, 2] <- round(qunif(mat[, 2], 0.1, 0.3), 2) # k_3
mat[, 3] <- floor(qunif(mat[, 3], 1, 200)) # Epsilon
mat[, 4] <- floor(mat[, 4] * (8 - 1 + 1)) + 1 # Phi
mat[, 5] <- floor(mat[, 5] * (3 - 1 + 1)) + 1 # Delta
mat[, 6] <- floor(mat[, 6] * (2 - 1 + 1)) + 1 # Tau
mat[, 7] <- floor(qunif(mat[, 7], 10, 1000)) # Nt
mat[, 8] <- floor(qunif(mat[, 8], 3, 100)) # k

# Define the base sample matrix
N.all <- apply(mat, 1, function(x) ceiling(x["N_t"] / (x["k"] + 1)))
N.azzini <- apply(mat, 1, function(x) ceiling(x["N_t"] / (2 * x["k"] + 2)))
N.vars <- apply(mat, 1, function(x) floor(x["N_t"] / ((x[["k"]] * ((1 / h) -1) + 1))))

tmp <- data.table(cbind(mat, N.all, N.azzini, N.vars))

# Now constrain N as a function of vars
tmp <- tmp[, N.all:= ifelse(N.vars == 1 | N.vars == 0, ceiling(2 * (4 * k + 1) / (k + 1)), N.al
tmp <- tmp[, N.azzini:= ifelse(N.vars == 1 | N.vars == 0, ceiling(2 * (4 * k + 1) / (2 * k + 2
tmp <- tmp[, N.vars:= ifelse(N.vars == 1 | N.vars == 0, 2, N.vars)]

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

mat <- cbind(tmp, C.all, C.azzini, C.vars)
```

```r
# EXPORT SAMPLE MATRIX ------------------------------------------------------
fwrite(mat, "mat.csv")
```

3.3 Define the model

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

```r
model_Ti <- function(k, N.all, N.azzini, N.vars, h,
                     N.high, k_2, k_3, epsilon, phi, delta, tau) {
  ind <- list()
  estimators <- c("jansen", "sobol", "homma", "monod", "azzini",
                  "lamboni", "glen", "owen", "vars")
  if(tau == 1) {
    method <- "R"
  } else if(tau == 2) {
    method <- "QRN"
  }
  set.seed(epsilon)
  all.but.azzini <- sobol_matrices(N = N.all, params = paste("X", 1:k, sep = ""),
                                   matrices = c("A", "AB"), method = method)
  set.seed(epsilon)
  azzini <- sobol_matrices(N = N.azzini, params = paste("X", 1:k, sep = ""),
                           matrices = c("A", "B", "AB", "BA"), method = method)
  set.seed(epsilon)
  owen.matrix <- sobol_matrices(N = N.azzini, params = paste("X", 1:k, sep = ""),
                           matrices = c("A", "B", "BA", "CB"), method = method)
  set.seed(epsilon)
  vars.matrix <- vars_matrices(star.centers = N.vars, params = paste("X", 1:k, sep = ""),
                               h = h, method = method)
  set.seed(epsilon)
  large.matrix <- sobol_matrices(N = N.high, params = paste("X", 1:k, sep = ""),
                                 matrices = c("A", "AB"), method = method)
  set.seed(epsilon)
  all.matrices <- random_distributions(X = rbind(all.but.azzini, azzini,
                                                 owen.matrix, vars.matrix,
                                                 large.matrix),
                                       phi = phi)
  output <- sensobol::metafunction(data = all.matrices,
                                   k_2 = k_2,
                                   k_3 = k_3,
                                   epsilon = epsilon)
  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"]

  # Define indices of Y for estimators
  Nt.all.but.azzini <- N.all * (k + 1)
  Nt.azzini.owen <- N.azzini * ((2 * k) + 2)
  Nt.vars <- N.vars * (k * ((1 / h) - 1) + 1)
  lg.all.but.azzini <- 1:Nt.all.but.azzini
  lg.azzini <- (length(lg.all.but.azzini) + 1):(length(lg.all.but.azzini) + Nt.azzini.owen)
  lg.owen <- (max(lg.azzini) + 1):(max(lg.azzini) + Nt.azzini.owen)
```

```
lg.vars <- (max(lg.owen) + 1): (max(lg.owen) + Nt.vars)

for(i in estimators) {
  if(i == "jansen" | i == "sobol" | i == "homma" | i == "monod" | i == "glen") {
    y <- output[lg.all.but.azzini]
    n <- N.all
  } else if(i == "azzini" | i == "lamboni") {
    y <- output[lg.azzini]
    n <- N.azzini
  } else if(i == "owen") {
    y <- output[lg.owen]
    n <- N.azzini
  } else if(i == "vars") {
    y <- output[lg.vars]
    star.centers <- N.vars
  }
  if(!i == "vars") {
    ind[[i]] <- sobol_Ti(d = y, N = n, params = paste("X", 1:k, sep = ""), total = i)
  }
  if(i == "vars") {
    ind[[i]] <- vars_ti(Y = y, star.centers = star.centers,
                        params = paste("X", 1:k, sep = ""), h = h)
  }
  ind[[i]][, sample.size:= "n"]
  ind[[i]] <- rbind(ind[[i]], full.ind)
}
# Arrange data
out <- rbindlist(ind, idcol = "estimator")
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)
}
```

3.4 Run the model

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

# Define parallel computing
cl <- makeCluster(n_cores)
registerDoParallel(cl)

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

# Stop parallel cluster
stopCluster(cl)
```

3.5 Arrange output

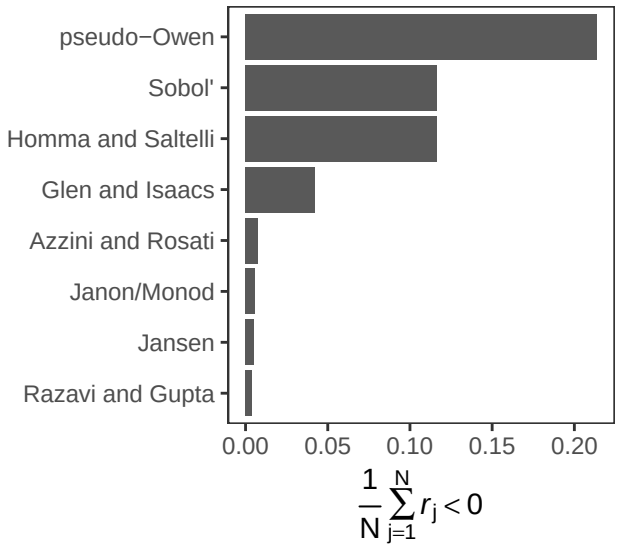
```
# ARRANGE OUTPUT -------------------------------------------------------------

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 == "azzini" | estimator == "lamboni"
                  | estimator == "owen", N.azzini * (2 * k + 2),
                  ifelse(estimator == "vars", N.vars * (k * ((1 / h) -1) + 1), 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",
                                             ifelse(estimator %in% "glen", "Glen and Isaacs",
                                                    ifelse(estimator %in% "lamboni", "Lamboni"
                                                           ifelse(estimator %in% "owen", "pseu
                                                                  ifelse(estimator %in% "vars"
                                                                         "Sobol'")))))))] %>%
  .[!estimator == "Lamboni"]

# Define A matrix (but drop Lamboni)
A <- full_output[!estimator == "Lamboni",.SD[1:N], estimator]

# Define matrix for total variance
VY.dt <- full_output[row %in% c(1:N)][
  , .(VY = 1 / N * sum((correlation - ((1 / N) * sum(correlation))) ^ 2)),
  estimator]
```

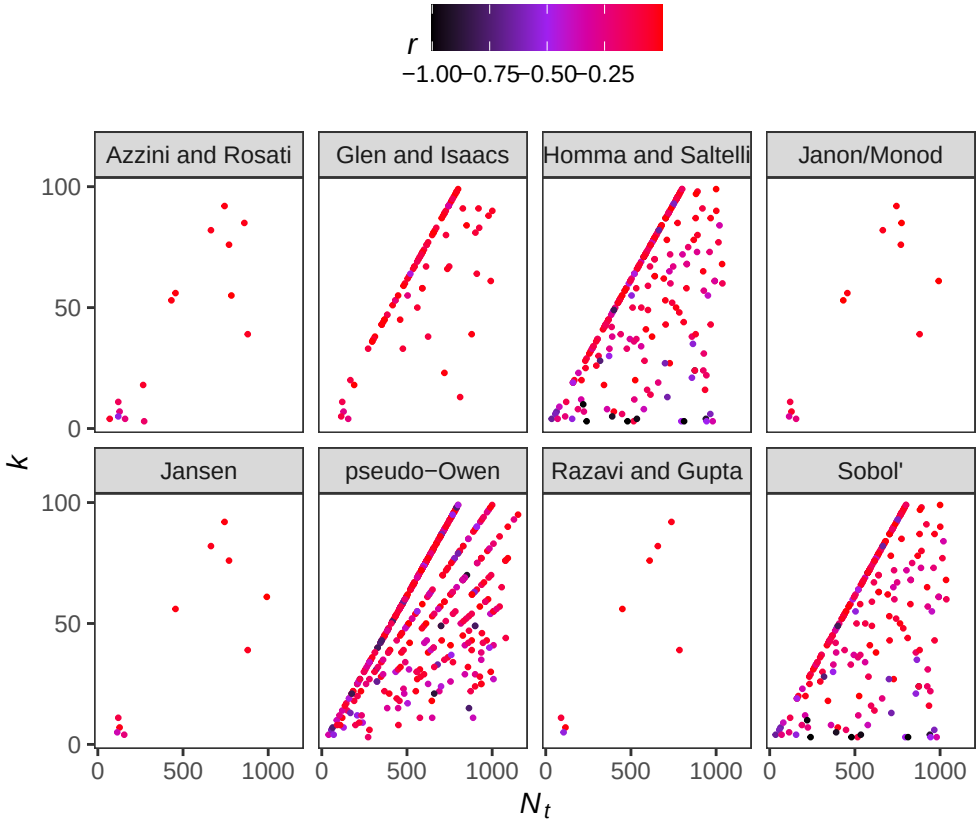
```
# EXPORT OUTPUT ------------------------------------------------------------------

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

4 Uncertainty analysis

```
# PLOT PROPORTION OF NEGATIVE VALUES ---------------------------------------------

A[, sum(correlation < 0)/ .N, estimator] %>%
  ggplot(., aes(reorder(estimator, V1), V1)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(y = expression(frac(1, N)~sum(italic(r)[j] < 0, j==1, N)),
       x = "") +
  theme_AP()
```




```
# MAP VALUES WITH NEGATIVE R ------------------------------------------------------

index.neg <- A[, .I[correlation < 0]]

A[index.neg] %>%
  ggplot(., aes(Nt, k, color = correlation)) +
  geom_point(size = 0.5) +
  scale_colour_gradientn(colours = c("black", "purple", "red"),
                         name = expression(italic(r))) +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  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")
```




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

# 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)]
```

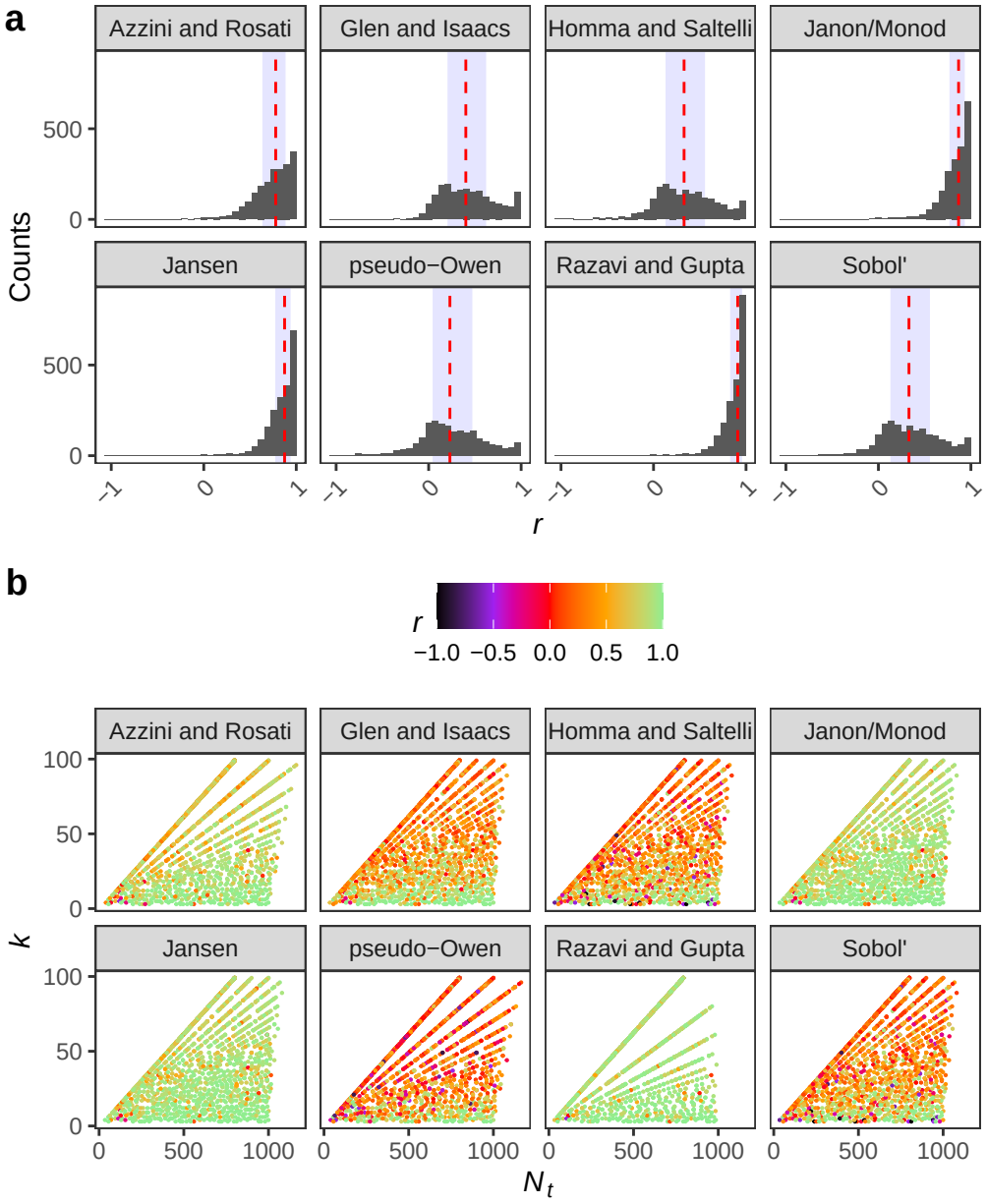
```
##             estimator    median     low.ci   high.ci
## 1:        pseudo-Owen 0.2298777 0.02805562 0.4878208
## 2: Homma and Saltelli 0.3302772 0.11494772 0.5721159
## 3:             Sobol' 0.3302772 0.11494772 0.5721159
## 4:    Glen and Isaacs 0.4019751 0.18866989 0.6371165
## 5:  Azzini and Rosati 0.7778500 0.61887933 0.8996873
## 6:        Janon/Monod 0.8666667 0.75282215 0.9487651
## 7:             Jansen 0.8739175 0.75804357 0.9540298
## 8:   Razavi and Gupta 0.9101595 0.81319190 0.9719266
```

```
a <- 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(axis.text.x = element_text(angle = 45, hjust = 1))

# Scatterplot
b <- ggplot(A, aes(Nt, k, color = correlation)) +
  geom_point(size = 0.1) +
  scale_colour_gradientn(colours = c("black", "purple", "red", "orange", "lightgreen"),
                         name = expression(italic(r))) +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  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")

# Ratio
c <- A[, ratio:= Nt / k] %>%
  ggplot(., aes(ratio, correlation)) +
  geom_point(alpha = 0.1, size = 0.2) +
  facet_wrap(~estimator,
             ncol = 4) +
  geom_smooth() +
  labs(x = expression(italic(N[t]/k)),
       y = expression(italic(r))) +
  scale_x_log10() +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  theme_AP()

# Merge plot
```

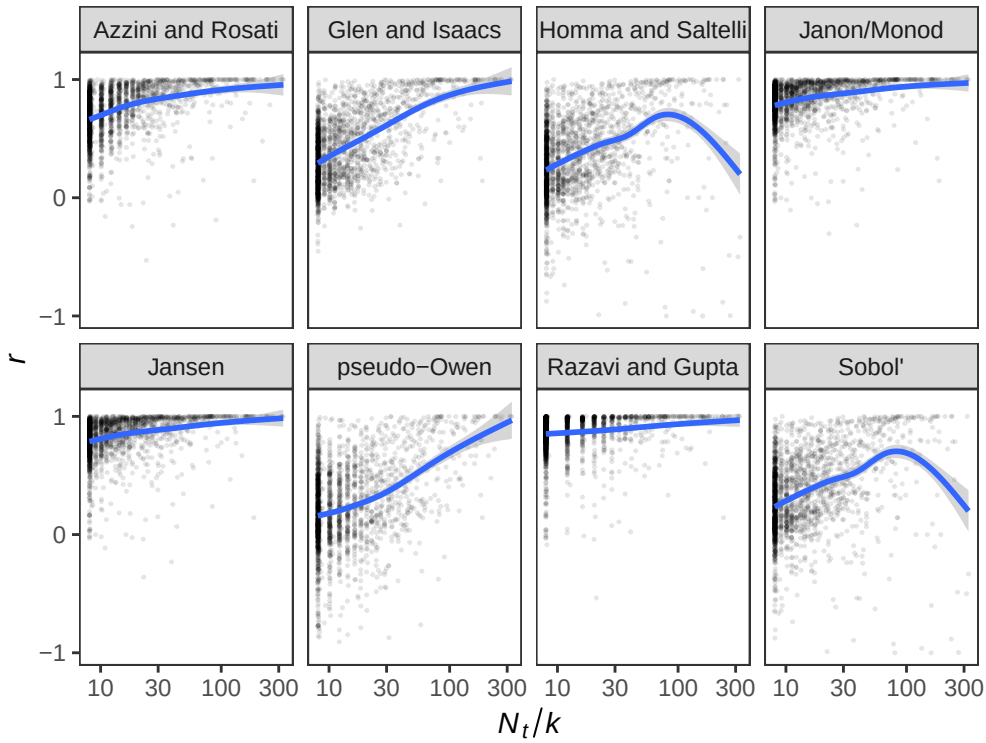
```
plot_grid(a, b, ncol = 1, labels = "auto", rel_heights = c(0.85, 1))
```

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

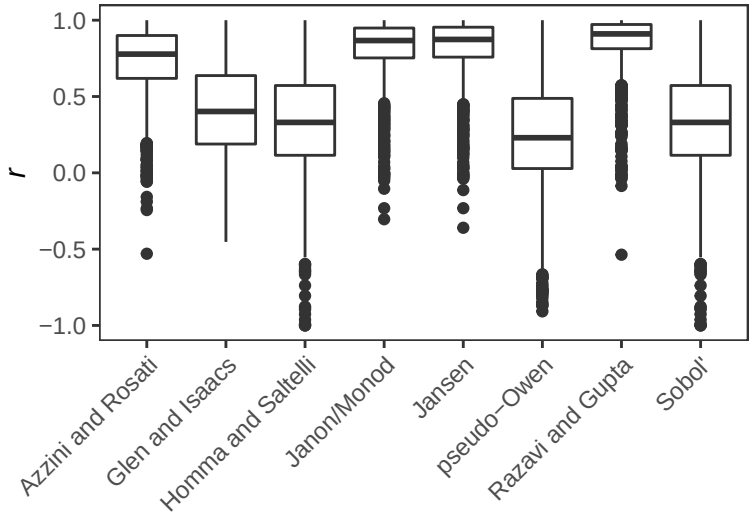



```
# DISPLAY THE RATIO NT/K FOR EACH SIMULATION AND ESTIMATOR ------------------------

c
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

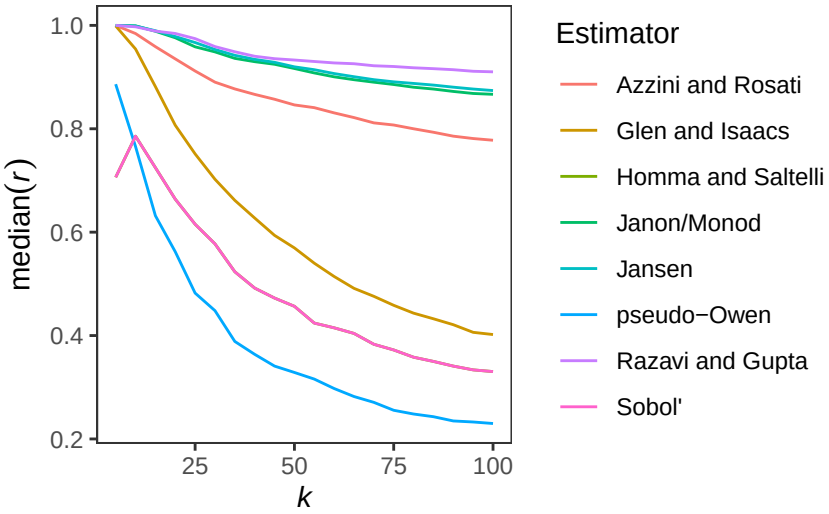



```
# PLOT BOXPLOT ------------------------------------------------------------------

ggplot(A, aes(estimator, correlation)) +
  geom_boxplot() +
  labs(x = "",
       y = expression(italic(r))) +
  theme_AP() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



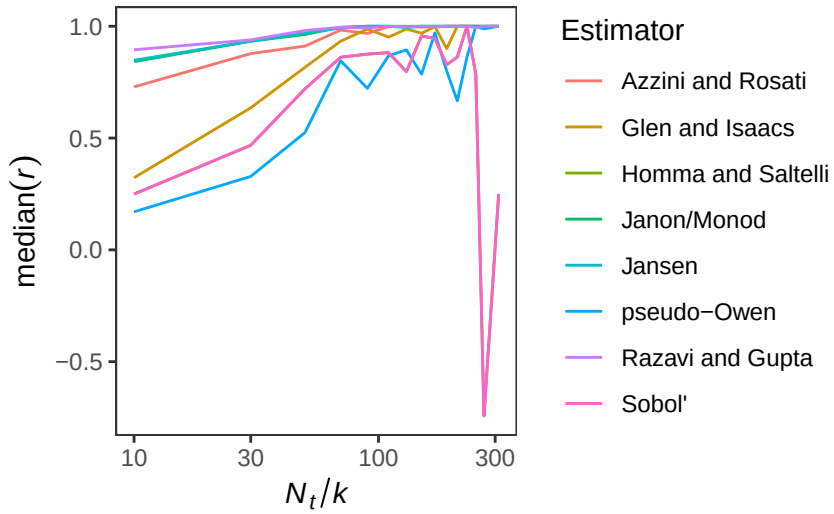

```
# PLOT MEDIANS ------------------------------------------------------------------

vv <- seq(5, 100, 5)
```

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

names(dt) <- vv

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()
```

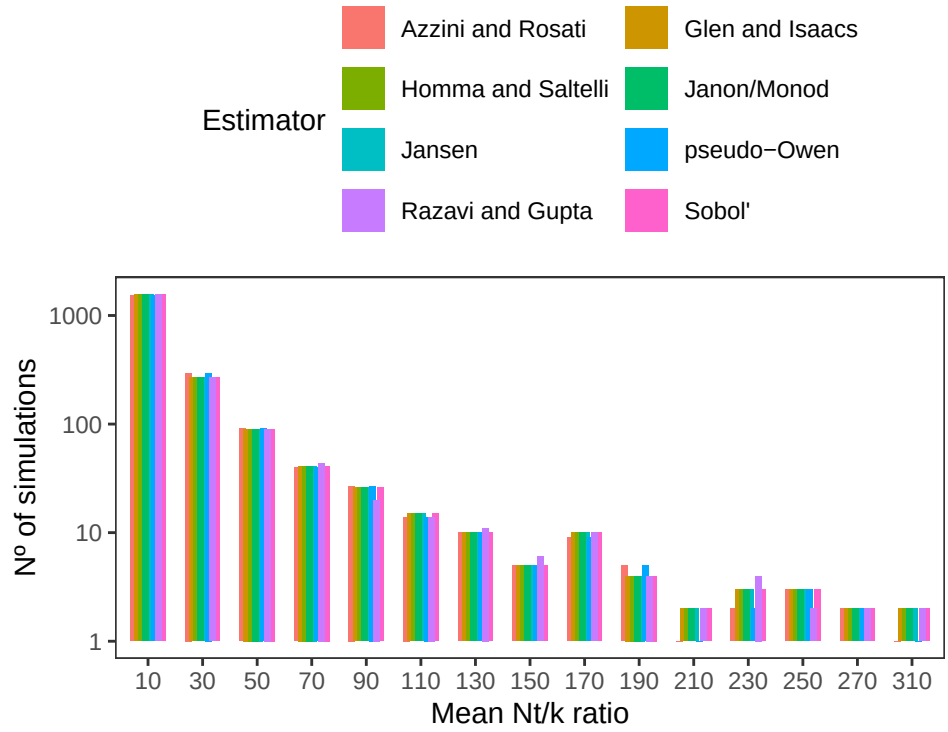



```r
# Median Nt/k
dt.tmp <- A[, .(min = min(ratio), max = max(ratio))]

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) <- rowmeans(indices)

# Plot
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 = expression(median(italic(r)))) +
  scale_color_discrete(name = "Estimator") +
  scale_x_log10() +
  theme_AP()
```




```
# PLOT NUMBER OF SIMULATIONS AGAINST NT/K RATIO ---------------------------------

rbindlist(out, idcol = "samples")[, .N, .(estimator, samples)] %>%
  .[, samples:= factor(samples, levels = rowmeans(indices))] %>%
  ggplot(., aes(samples, N, fill = estimator)) +
  scale_y_log10() +
  scale_fill_discrete(name = "Estimator") +
  labs(x = "Mean Nt/k ratio",
       y = "Nº of simulations") +
  geom_bar(stat = "identity",
           position = position_dodge(0.6)) +
  theme_AP() +
  theme(legend.position = "top") +
  guides(fill = guide_legend(nrow = 4, byrow = TRUE))
```



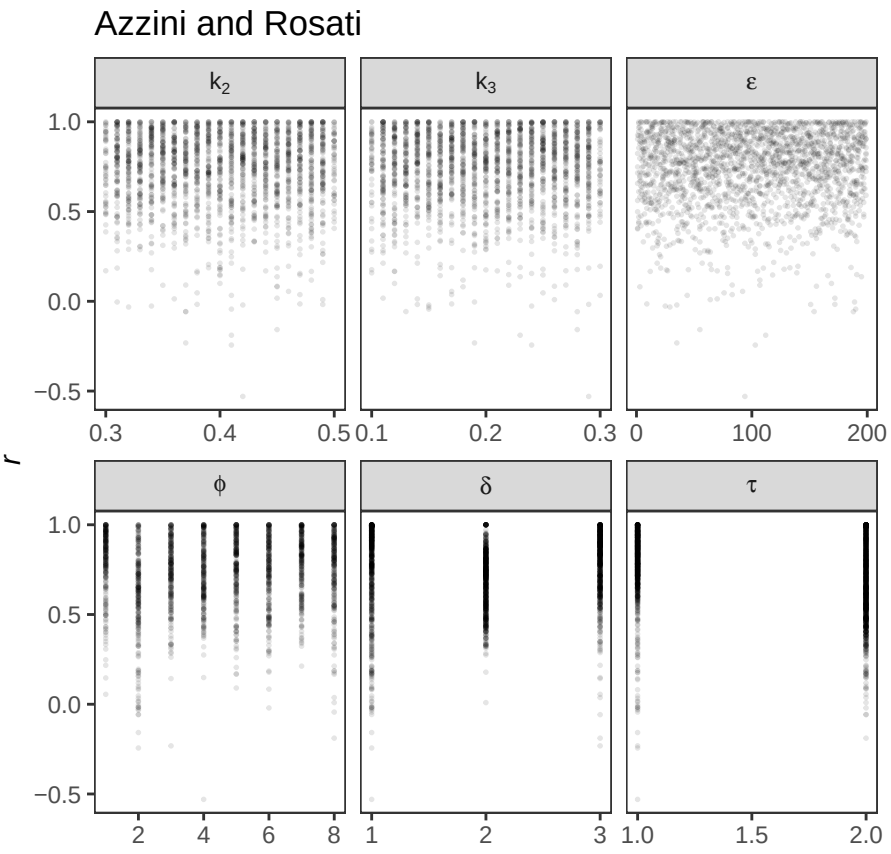
5 Sensitivity analysis

5.1 Scatterplots

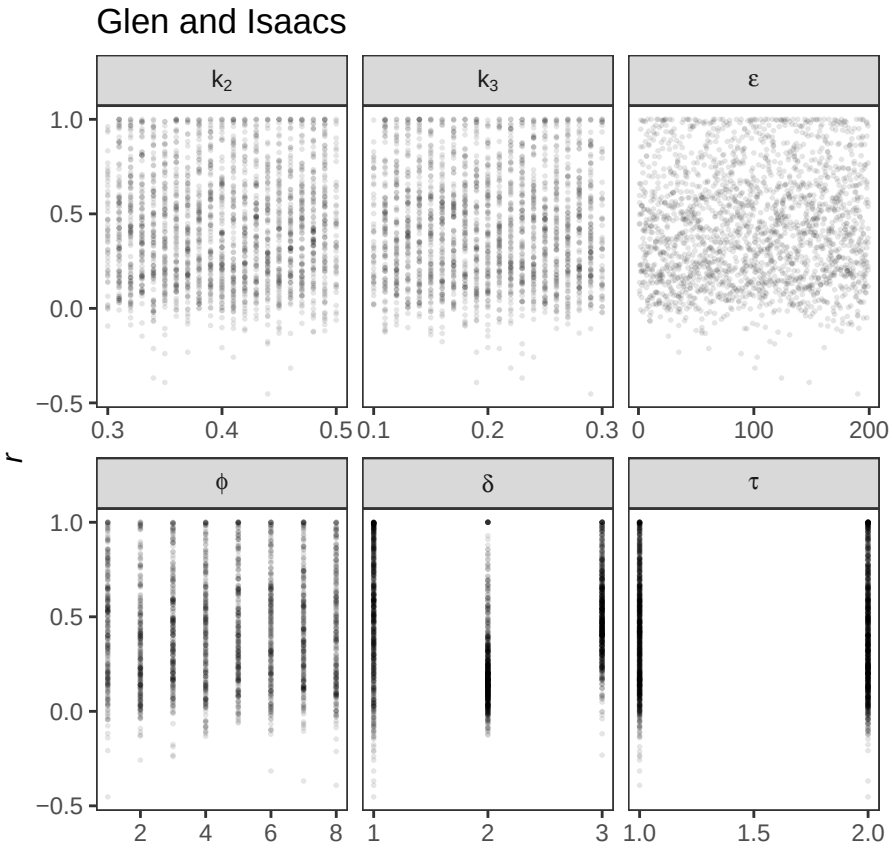
```
# SCATTERPLOTS OF MODEL OUTPUT AGAINST PARAMETERS ----------------------------------

A <- setnames(A, c("N_t", "k_2", "k_3"), c("N[t]", "k[2]", "k[3]"))

scatter.dt <- melt(A, measure.vars = c("k[2]", "k[3]",
                                       "epsilon", "phi", "delta", "tau")) %>%
  split(., .$estimator)

gg <- list()
for(i in names(scatter.dt)) {
  gg[[i]] <- ggplot(scatter.dt[[i]], aes(value, correlation)) +
    geom_point(alpha = 0.1, size = 0.3) +
    facet_wrap(~ variable,
               scales = "free_x",
               labeller = label_parsed,
               ncol = 3) +
    scale_color_manual(values = c("#00BFC4", "#F8766D")) +
    scale_x_continuous(breaks = pretty_breaks(n = 3)) +
    theme_AP() +
    labs(x = "", y = expression(italic(r))) +
    theme() +
    ggtitle(names(scatter.dt[i]))
```

```
}

gg
```

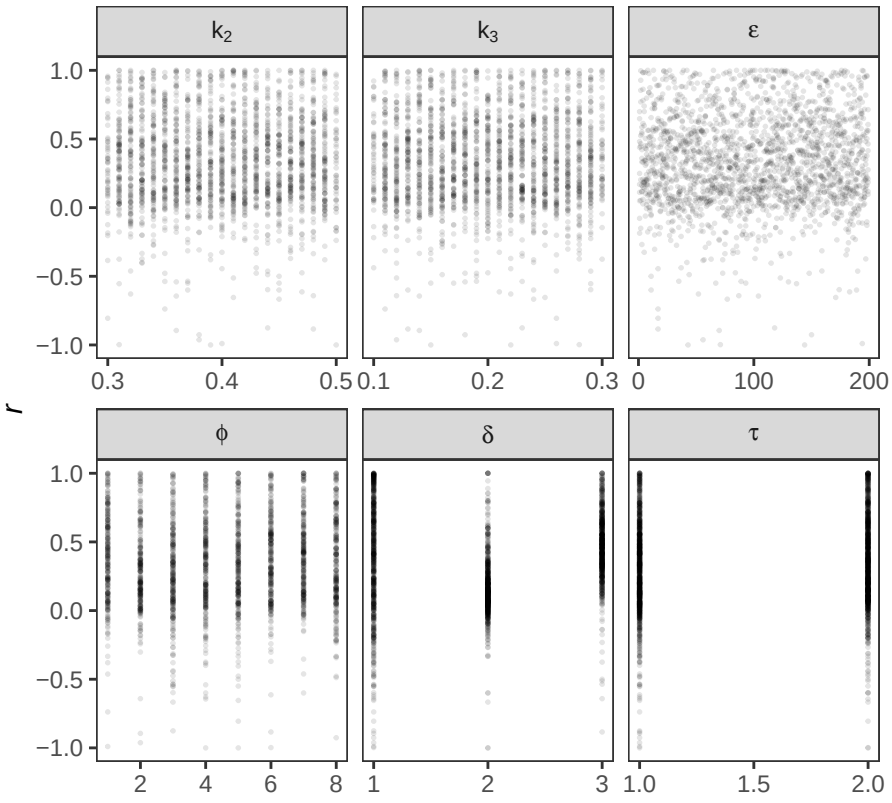
```
## $`Azzini and Rosati`
```



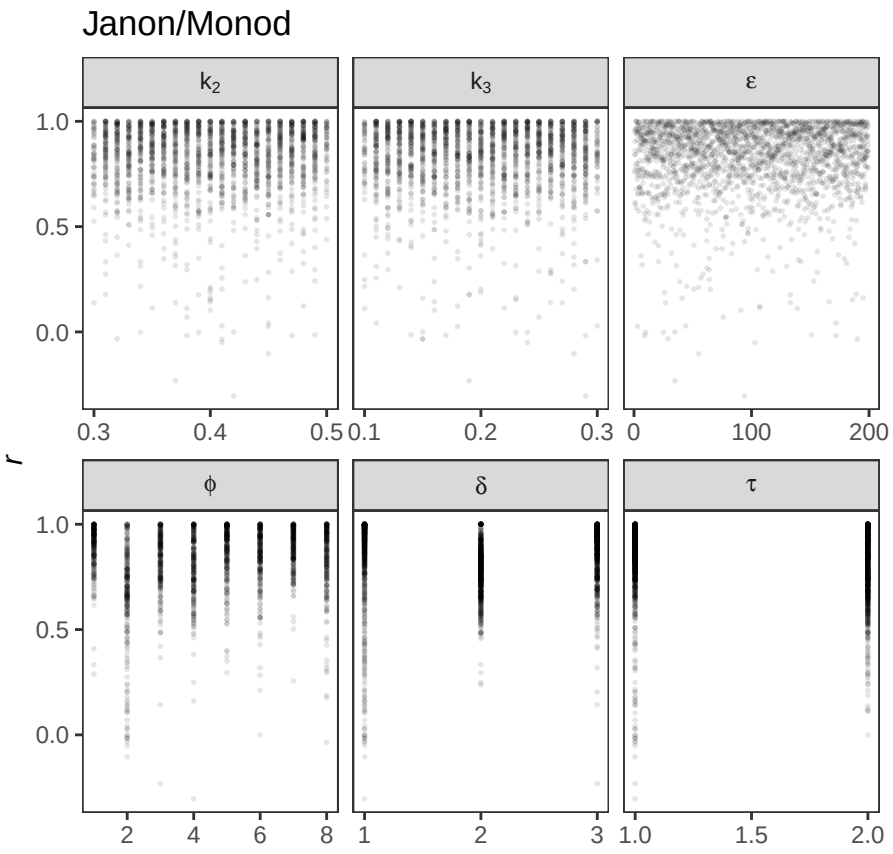

```
##
## $`Glen and Isaacs`
```



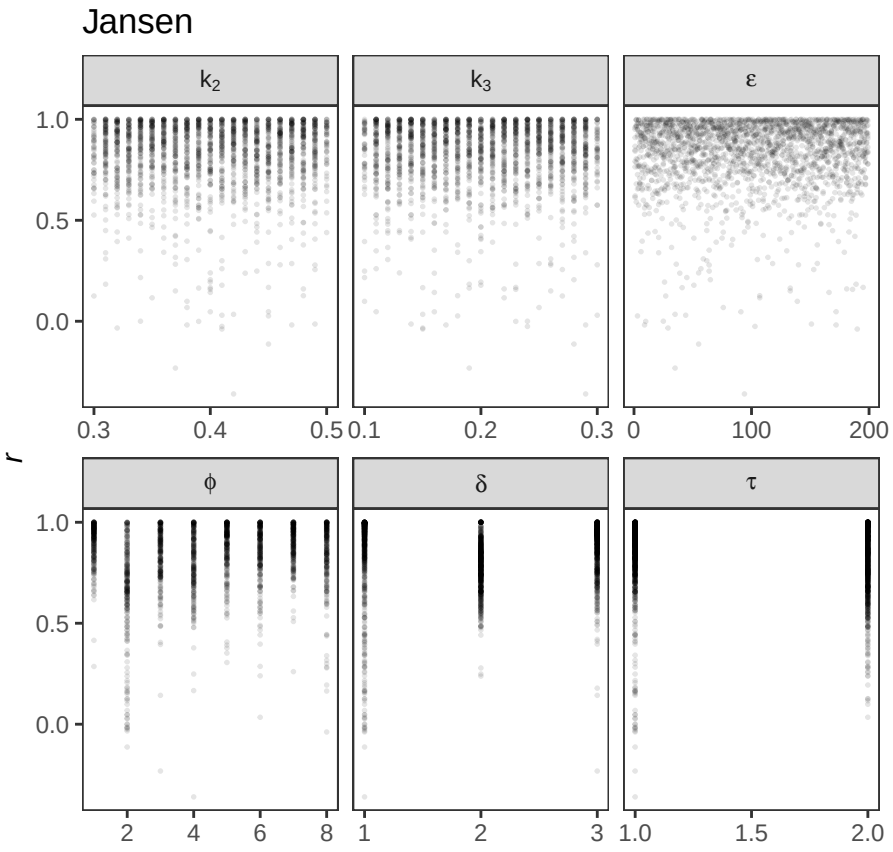

```
##
## $`Homma and Saltelli`
```


Homma and Saltelli
k2
k3
ε
ϕ
δ
τ
r
1.0
0.5
0.0
−0.5
−1.0
0.3
0.4
0.5
0.1
0.2
0.3
0
100
200
2
4
6
8
1
2
3
1.0
1.5
2.0

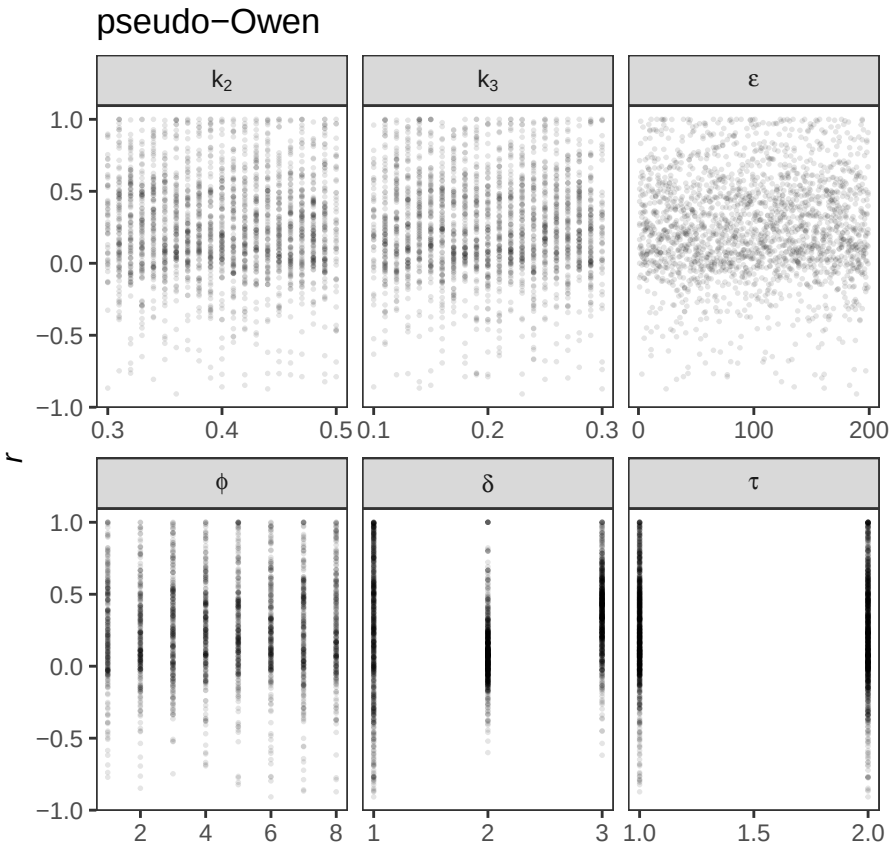
```
##
## $`Janon/Monod`
```


Janon/Monod
k2
k3
ε
ϕ
δ
τ
r
1.0
0.5
0.0
0.3
0.4
0.5
0.1
0.2
0.3
0
100
200
2
4
6
8
1
2
3
1.0
1.5
2.0

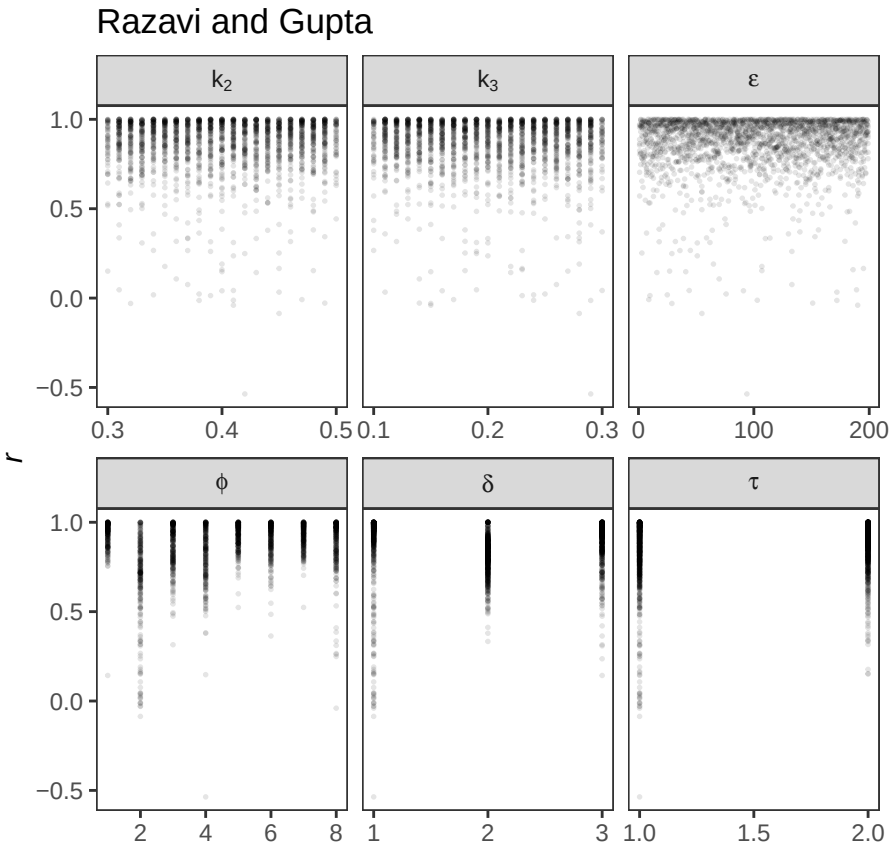
```
##
## $Jansen
```


Jansen
k2
k3
ε
ϕ
δ
τ
r
1.0
0.5
0.0
0.3
0.4
0.5
0.1
0.2
0.3
0
100
200
2
4
6
8
1
2
3
1.0
1.5
2.0

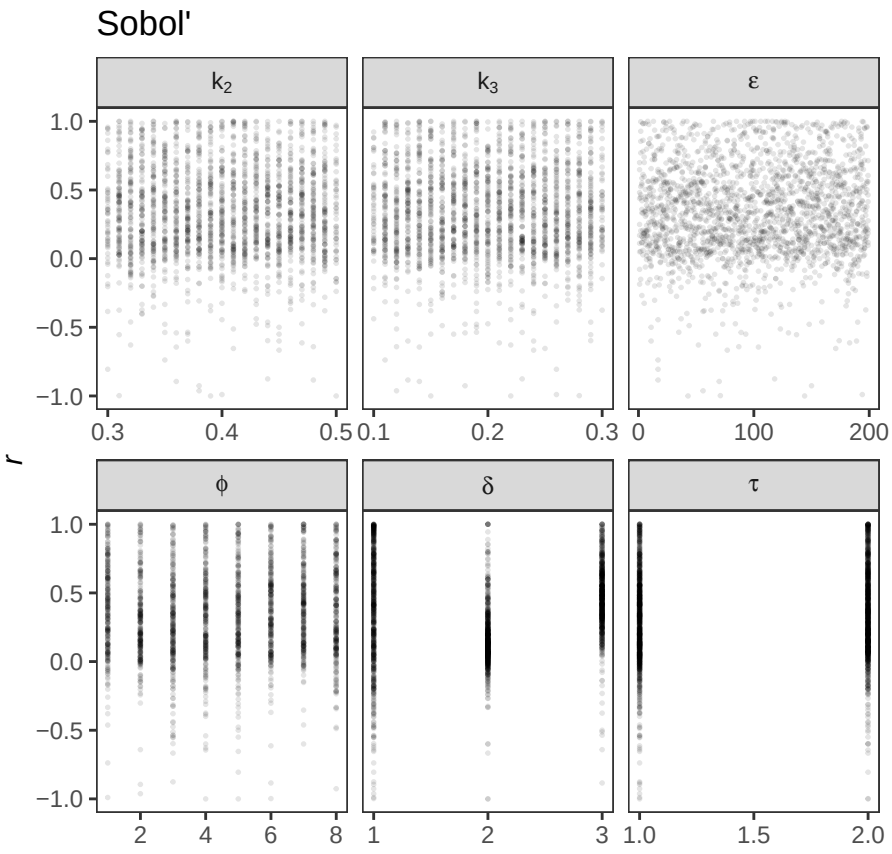
```
##
## $`pseudo-Owen`
```




```
##
## $`Razavi and Gupta`
```


Razavi and Gupta
k2
k3
ε
ϕ
δ
τ
r
1.0
0.5
0.0
−0.5
0.3
0.4
0.5
0.1
0.2
0.3
0
100
200
2
4
6
8
1
2
3
1.0
1.5
2.0

```
##
## $`Sobol'`
```



5.2 Sobol' indices

```
# SENSITIVITY ANALYSIS ----------------------------------------------------------

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

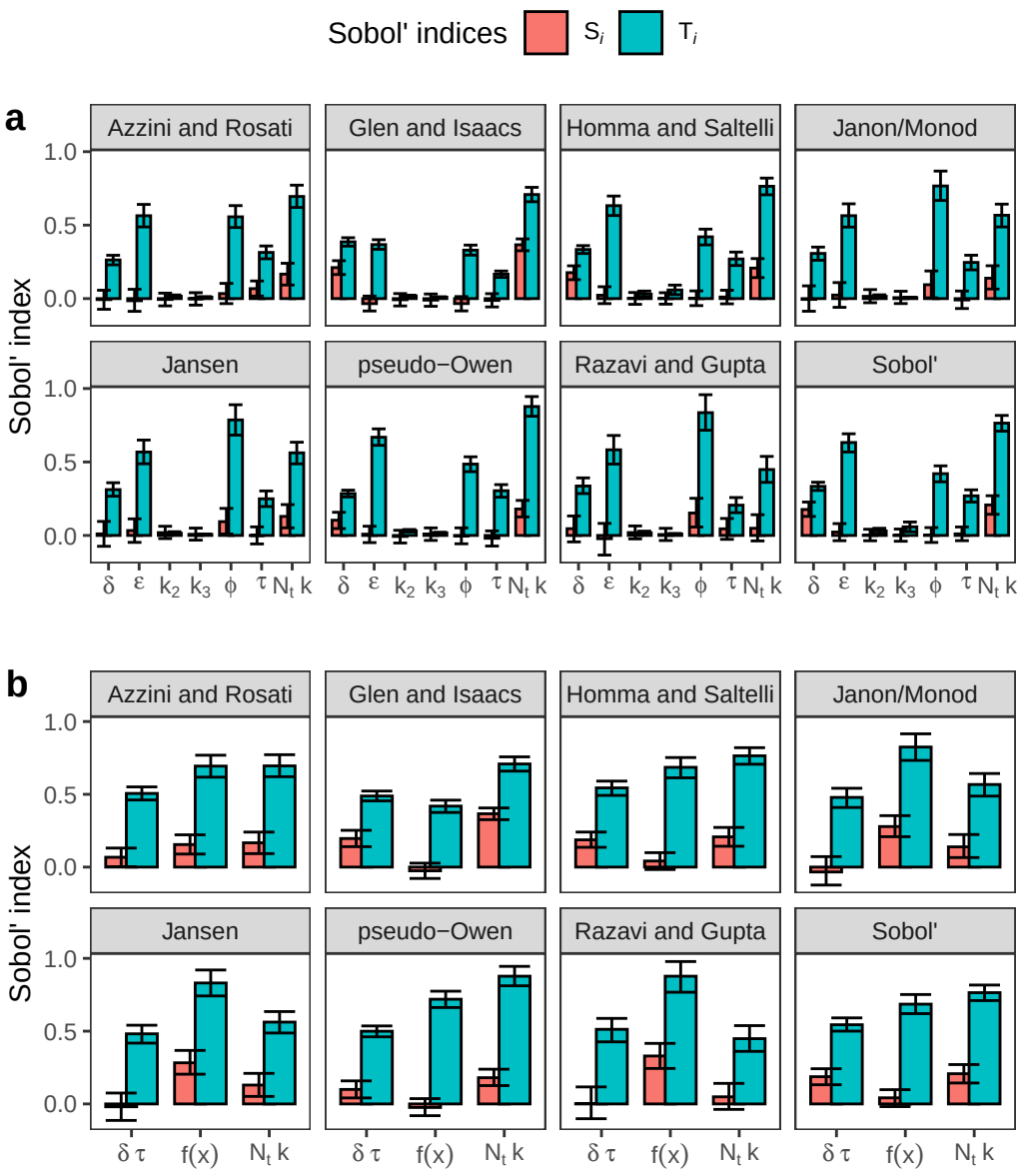
```
## Empty data.table (0 rows and 18 cols): row,k_2,k_3,epsilon,phi,delta...
```

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

# New vector of parameters to plot better
params.new <- c("k[2]", "k[3]", "epsilon", "phi", "delta", "tau")

# Compute Sobol' indices except for the cluster Nt,k
indices <- full_output[!row %in% mat[, tail(.I, N)]][
  , sobol_indices(Y = correlation,
                  N = N,
                  params = params.new,
                  first = "jansen",
                  boot = TRUE,
                  R = R,
                  order = order),
```

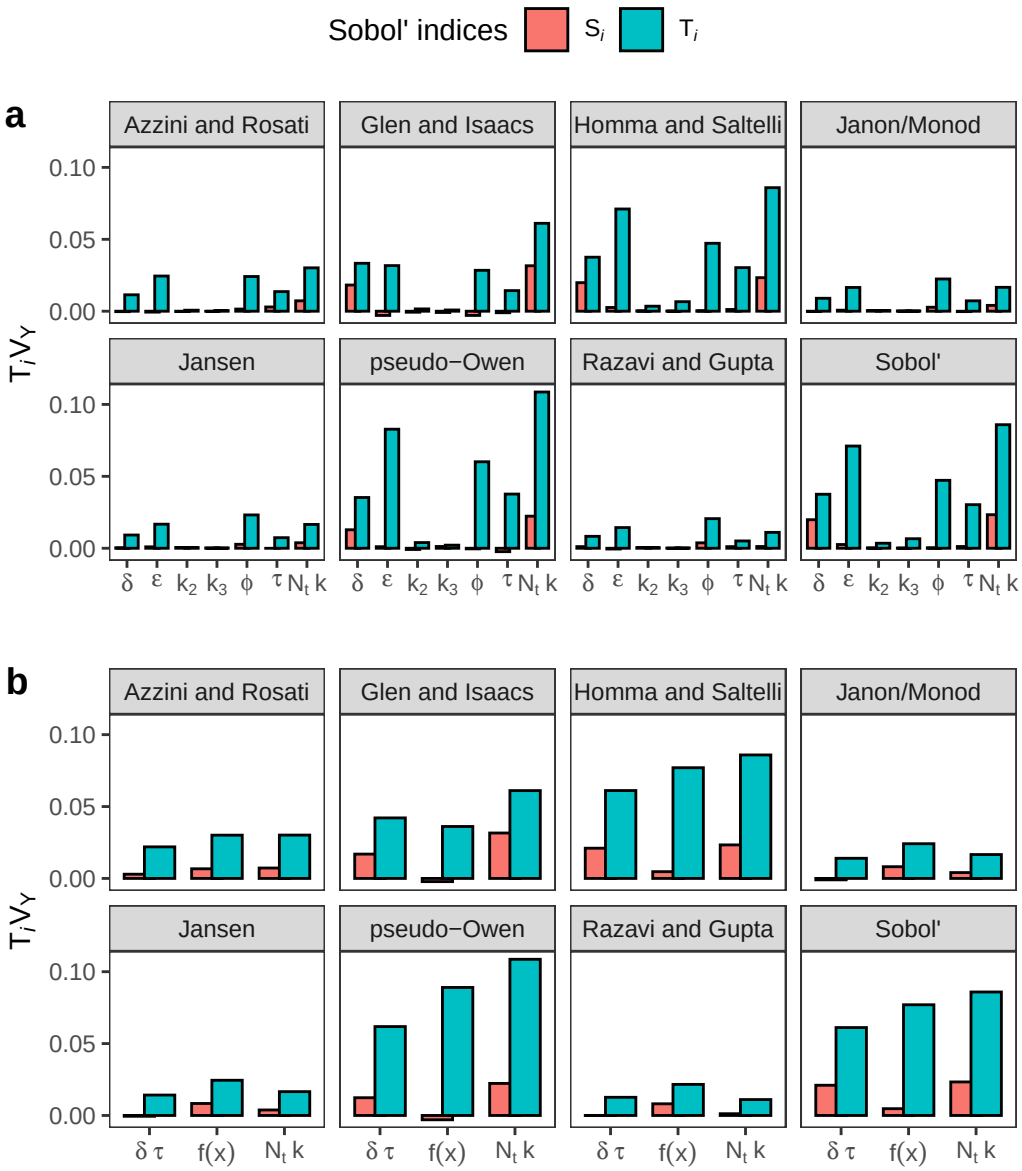
```
  estimator]

# Compute Sobol' indices for cluster
index.clusters <- mat[, tail(.I, 3 * N)]

indicesC <- rbind(full_output[row %in% 1:(2 * N)], full_output[row %in% index.clusters])[
  , sobol_indices(Y = correlation,
                  N = N,
                  params = c("N[t]~k", "f(x)", "delta~tau"),
                  first = "jansen",
                  boot = TRUE,
                  R = R,
                  order = "first"),
  estimator]

# Final
indices <- rbind(indices, indicesC[parameters == "N[t]~k"]) %>%
  merge(., VY.dt, by = "estimator") %>%
  .[, unnormalized:= original * VY]

# Export
fwrite(indices, "indices.csv")
fwrite(indicesC, "indicesC.csv")
```

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

all.indices <- list(indices, indicesC)

gg <- list()
for(i in seq_along(all.indices)) {
  gg[[i]] <- ggplot(all.indices[[i]][sensitivity == "Si" |
                                       sensitivity == "Ti"],
                    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_x_discrete(labels = ggplot2:::parse_safe) +
    scale_y_continuous(breaks = pretty_breaks(n = 3)) +
    facet_wrap(~estimator,
               ncol = 4) +
    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 = "none")
}

# Get legend
legend <- get_legend(gg[[1]] + theme(legend.position = "top"))

top <- plot_grid(gg[[1]], gg[[2]] + theme(legend.position = "none"),
                 align = "hv", labels = "auto", ncol = 1)

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



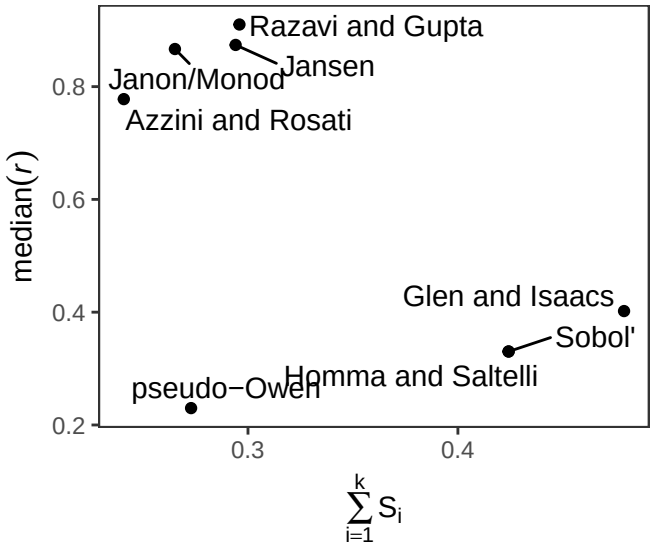
```r
# PLOT UNNORMALIZED SOBOL' INDICES ----------------------------------------

indicesC <- indicesC %>%
  merge(., VY.dt, by = "estimator") %>%
  .[, unnormalized:= original * VY]

all.indicesC <- list(indices, indicesC)

hh <- list()
for(i in seq_along(all.indicesC)) {
  hh[[i]] <- ggplot(all.indicesC[[i]][sensitivity == "Si" |
                                        sensitivity == "Ti"],
                    aes(parameters, unnormalized, fill = sensitivity)) +
    geom_bar(stat = "identity",
             position = position_dodge(0.6),
             color = "black") +
    scale_x_discrete(labels = ggplot2:::parse_safe) +
    scale_y_continuous(breaks = pretty_breaks(n = 3)) +
    facet_wrap(~estimator,
               ncol = 4) +
    labs(x = "",
         y = expression(T[italic(i)] * V[Y])) +
    scale_fill_discrete(name = "Sobol' indices",
                        labels = c(expression(S[italic(i)]),
                                   expression(T[italic(i)]))) +
    theme_AP() +
    theme(legend.position = "none")
}

# Get legend
legend <- get_legend(hh[[1]] + theme(legend.position = "top"))

top <- plot_grid(hh[[1]], hh[[2]] + theme(legend.position = "none"),
                 align = "hv", labels = "auto", ncol = 1)

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




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

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

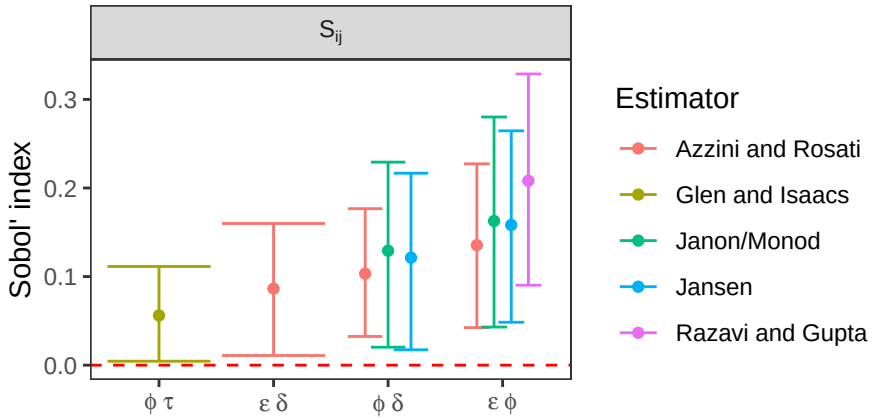
```
##             estimator        V1
## 1:  Azzini and Rosati 0.2408583
## 2:    Glen and Isaacs 0.4791346
## 3: Homma and Saltelli 0.4241118
## 4:        Janon/Monod 0.2652128
## 5:             Jansen 0.2941218
## 6:   Razavi and Gupta 0.2959356
## 7:             Sobol' 0.4241816
## 8:        pseudo-Owen 0.2729050
```

```r
# Plot
merge(indices[sensitivity == "Si", sum(original), estimator],
      dt_median, by = "estimator") %>%
  ggplot(., aes(V1, median)) +
  geom_point() +
  labs(x = expression(sum(S[i], i==1, k)),
       y = expression(median(italic(r)))) +
  geom_text_repel(aes(label = estimator)) +
  theme_AP()
```




```r
# PLOT SECOND-ORDER EFFECTS ------------------------------------------------------------

indices[sensitivity == "Sij"] %>%
  .[low.ci > 0] %>%
  .[, sensitivity:= ifelse(sensitivity %in% "Sij", "S[ij]", sensitivity)] %>%
  .[, parameters:= gsub(parameters,
                        pattern = ".",
                        replacement = "~",
                        fixed = TRUE)] %>%
  ggplot(., aes(reorder(parameters, original), original, color = estimator)) +
  geom_point(position = position_dodge(0.6)) +
  geom_errorbar(aes(ymax = high.ci, ymin = low.ci),
                position = position_dodge(0.6)) +
  geom_hline(yintercept = 0,
             lty = 2,
             color = "red") +
  facet_grid(~sensitivity,
             scales = "free_x",
             space = "free_x",
             labeller = label_parsed) +
  scale_x_discrete(labels = ggplot2:::parse_safe) +
  scale_color_discrete(name = "Estimator") +
```

```
  labs(x = "",
       y = "Sobol' index") +
  theme_AP()
```

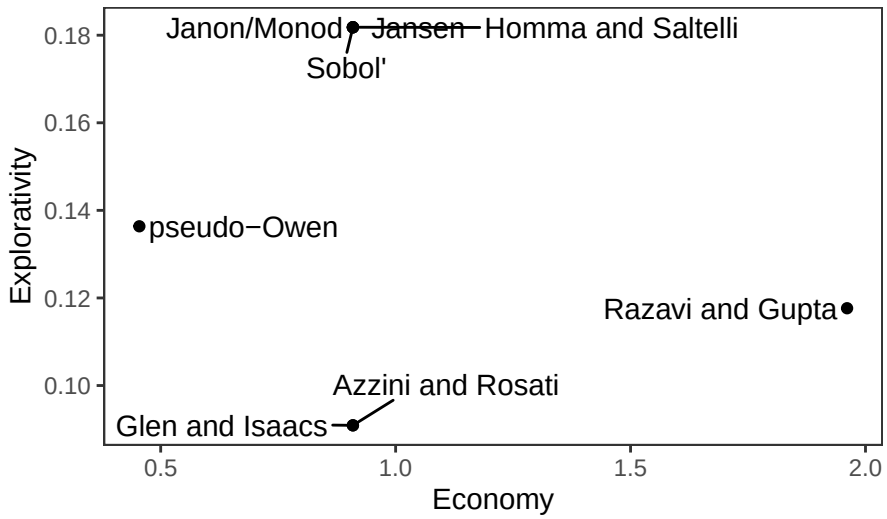



```
# EXPORT SOBOL' INDICES ----------------------------------------------------------------

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

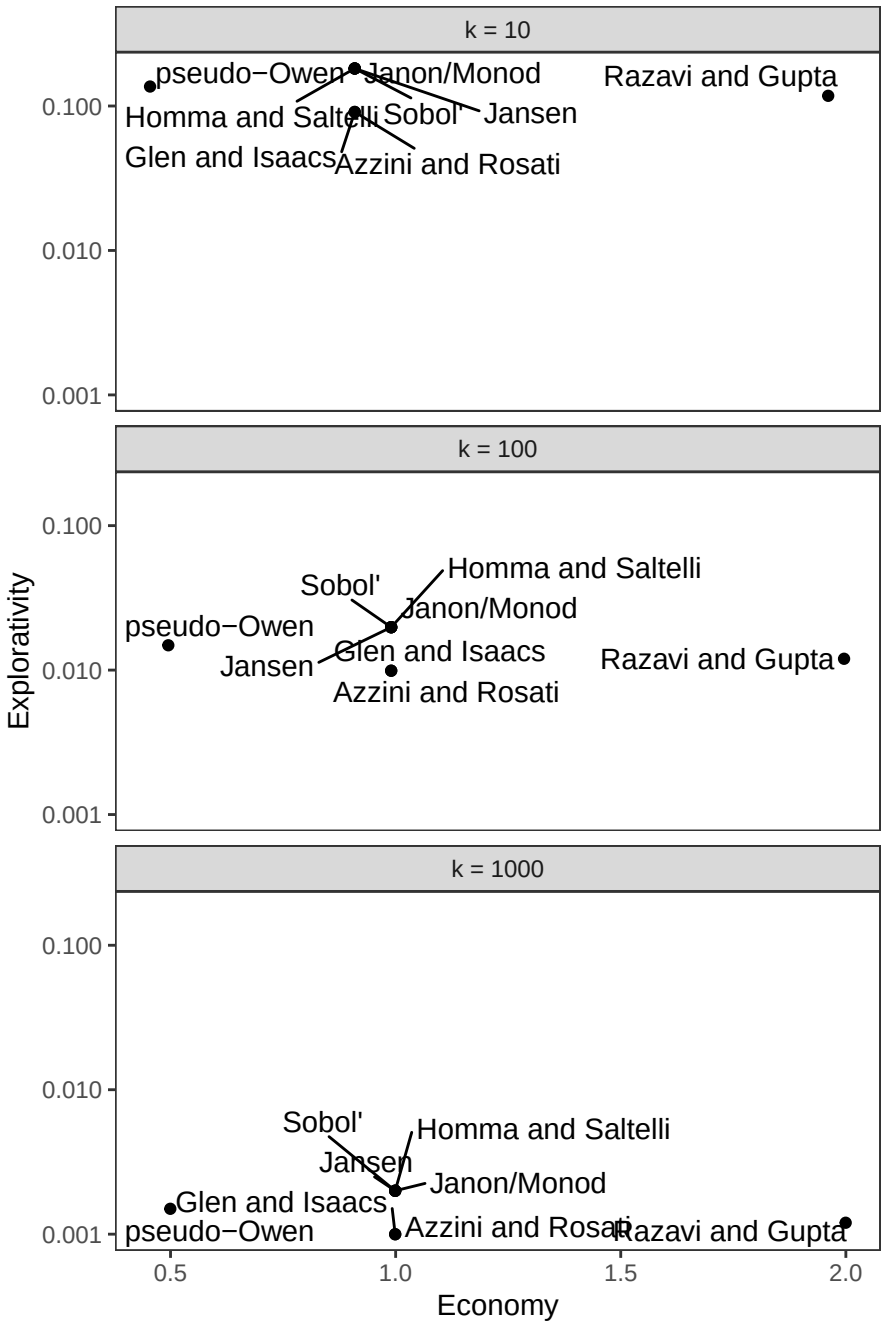
```
# CALCULATE EXPLORATIVITY AND ECONOMY --------------------------------------------

# Create functions
economy_fun <- function(n, k, h) {
  Owen <- k / (2 * (k + 1))
  Razavi <- (2 / h * k) / (1 + 1 / h * k)
  Azzini <- k / (k + 1)
  Glen <- k / (k + 1)
  Sobol <- k / (k + 1)
  Homma <- k / (k + 1)
  Jansen <- k / (k + 1)
  Janon <- k / (k + 1)
  eff <- c(Owen, Razavi, Azzini, Glen, Sobol,
           Homma, Jansen, Janon)
  return(eff)
}

exploravity_fun <- function(n, k, h) {
  Owen <- 3 / ( 2 * (1 + k))
  Razavi <- (1 + 1 / h) / (1 + 1 / h * k)
  Azzini <- 1 / (k + 1)
  Glen <- 1 / (k + 1)
  Sobol <- 2 / (k + 1)
  Homma <- 2 / (k + 1)
  Jansen <- 2 / (k + 1)
  Janon <- 2 / (k + 1)
  ec <- c(Owen, Razavi, Azzini, Glen,
          Sobol, Homma, Jansen, Janon)
```

```
  return(ec)
}

full_fun <- function(n, k, h) {
  arrangement <- data.table(c("pseudo-Owen",
                              "Razavi and Gupta",
                              "Azzini and Rosati",
                              "Glen and Isaacs",
                              "Sobol'",
                              "Homma and Saltelli",
                              "Jansen",
                              "Janon/Monod"))
  ec <- economy_fun(n, k, h)
  ex <- exploravity_fun(n, k, h)
  out <- data.table(cbind(ec, ex))
  final <- cbind(arrangement, out)
  setnames(final, "V1", "Arrangement")
  return(final)
}

vec.k <- c(10, 100, 1000)
data.ex <- lapply(vec.k, function(k) full_fun(n = 4, k = k, h = 0.2))
names(data.ex) <- vec.k

plot.data.ex <- rbindlist(data.ex, idcol = "k") %>%
  .[, k:= as.numeric(k)]
```

```
# PLOT EXPLORATIVITY ------------------------------------------------------------

ggplot(plot.data.ex[k == 10], aes(ec, ex)) +
  geom_point() +
  labs(x = "Economy",
       y = "Exploravity") +
  geom_text_repel(aes(label = Arrangement)) +
  theme_AP()
```




```
# PLOT EXPLORATIVITY ------------------------------------------------------------

plot.data.ex[, k:= paste("k = ", k, sep = "")] %>%
  ggplot(., aes(ec, ex)) +
  geom_point() +
  labs(x = "Economy",
       y = "Explorativity") +
  geom_text_repel(aes(label = Arrangement)) +
  facet_wrap(~k, ncol = 1) +
  scale_y_log10() +
  theme_AP()
```


k = 10
pseudo−Owen
Janon/Monod
Razavi and Gupta
Homma and Saltelli
Sobol'
Jansen
Glen and Isaacs
Azzini and Rosati
0.100
0.010
0.001
k = 100
Sobol'
Homma and Saltelli
Janon/Monod
pseudo−Owen
Glen and Isaacs
Jansen
Razavi and Gupta
Azzini and Rosati
Explorativity
k = 1000
Sobol'
Homma and Saltelli
Jansen
Janon/Monod
Glen and Isaacs
pseudo−Owen
Azzini and Rosati
Razavi and Gupta
0.5
1.0
1.5
2.0
Economy

6 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 10.15.4
##
## 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] pcaPP_1.9-73              checkpoint_0.4.9
##  [3] ggrepel_0.8.2             sensobol_0.3
##  [5] logitnorm_0.8.37          benchmarkme_1.0.3
##  [7] cowplot_1.0.0             scales_1.1.0
##  [9] data.table_1.12.8         Rfast_1.9.9
## [11] RcppZiggurat_0.1.5        doParallel_1.0.15
## [13] iterators_1.0.12          foreach_1.5.0
## [15] forcats_0.5.0             stringr_1.4.0
## [17] dplyr_0.8.5               purrr_0.3.4
## [19] readr_1.3.1               tidyr_1.0.2
## [21] tibble_3.0.1              ggplot2_3.3.0
## [23] tidyverse_1.3.0           RcppArmadillo_0.9.860.2.0
## [25] Rcpp_1.0.4.6
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.1            jsonlite_1.6.1        modelr_0.1.6
##  [4] Rdpack_0.11-1         assertthat_0.2.1      cellranger_1.1.0
##  [7] yaml_2.2.1            remotes_2.1.1         pillar_1.4.3
## [10] backports_1.1.6       lattice_0.20-41       glue_1.4.0
## [13] digest_0.6.25         rvest_0.3.5           colorspace_1.4-1
## [16] htmltools_0.4.0       Matrix_1.2-18         pkgconfig_2.0.3
## [19] bibtex_0.4.2.2        broom_0.5.6           haven_2.2.0
## [22] mvtnorm_1.1-0         farver_2.0.3          generics_0.0.2
## [25] ellipsis_0.3.0        withr_2.2.0           cli_2.0.2
## [28] magrittr_1.5          crayon_1.3.4          readxl_1.3.1
## [31] evaluate_0.14         fs_1.4.1              fansi_0.4.1
```

```
## [34] nlme_3.1-147          xml2_1.3.1            tools_3.6.3
## [37] hms_0.5.3             gbRd_0.4-11           lifecycle_0.2.0
## [40] munsell_0.5.0         reprex_0.3.0          compiler_3.6.3
## [43] rlang_0.4.5           grid_3.6.3            rstudioapi_0.11
## [46] labeling_0.3          rmarkdown_2.1         gtable_0.3.0
## [49] codetools_0.2-16      DBI_1.1.0             curl_4.3
## [52] benchmarkmeData_1.0.3 R6_2.4.1              lubridate_1.7.8
## [55] knitr_1.28            stringi_1.4.6         vctrs_0.2.4
## [58] dbplyr_1.4.3          tidyselect_1.0.0      xfun_0.13
```

```
## 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:
```

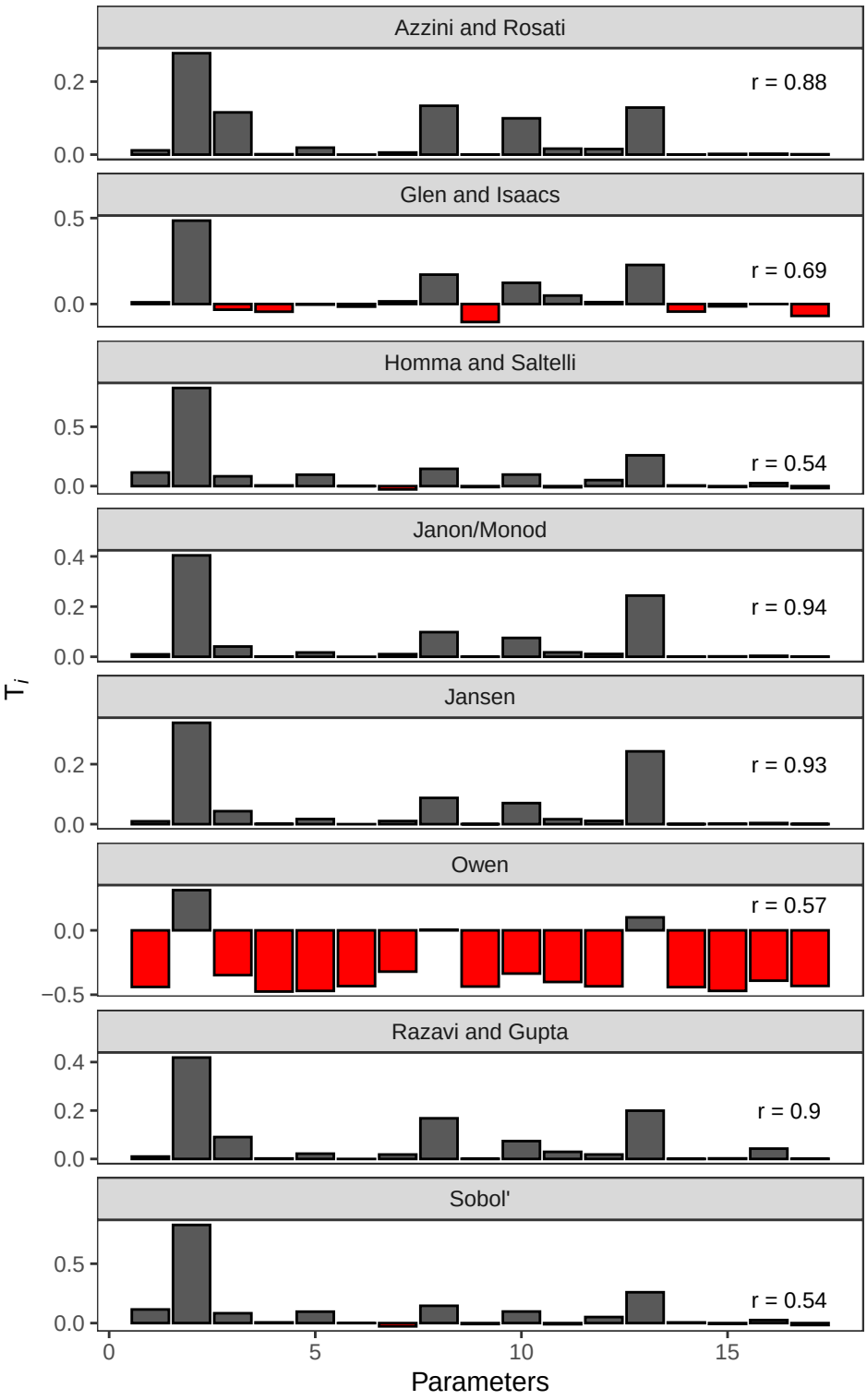
```
## [1] 16
```

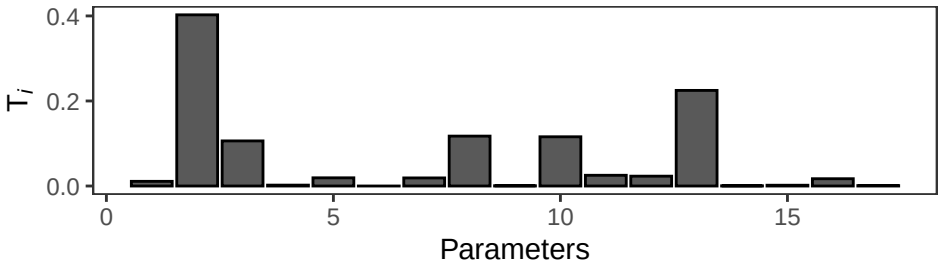
```
## Return the machine RAM
cat("RAM:         "); print (get_ram()); cat("\n")
```

```
## RAM:
```

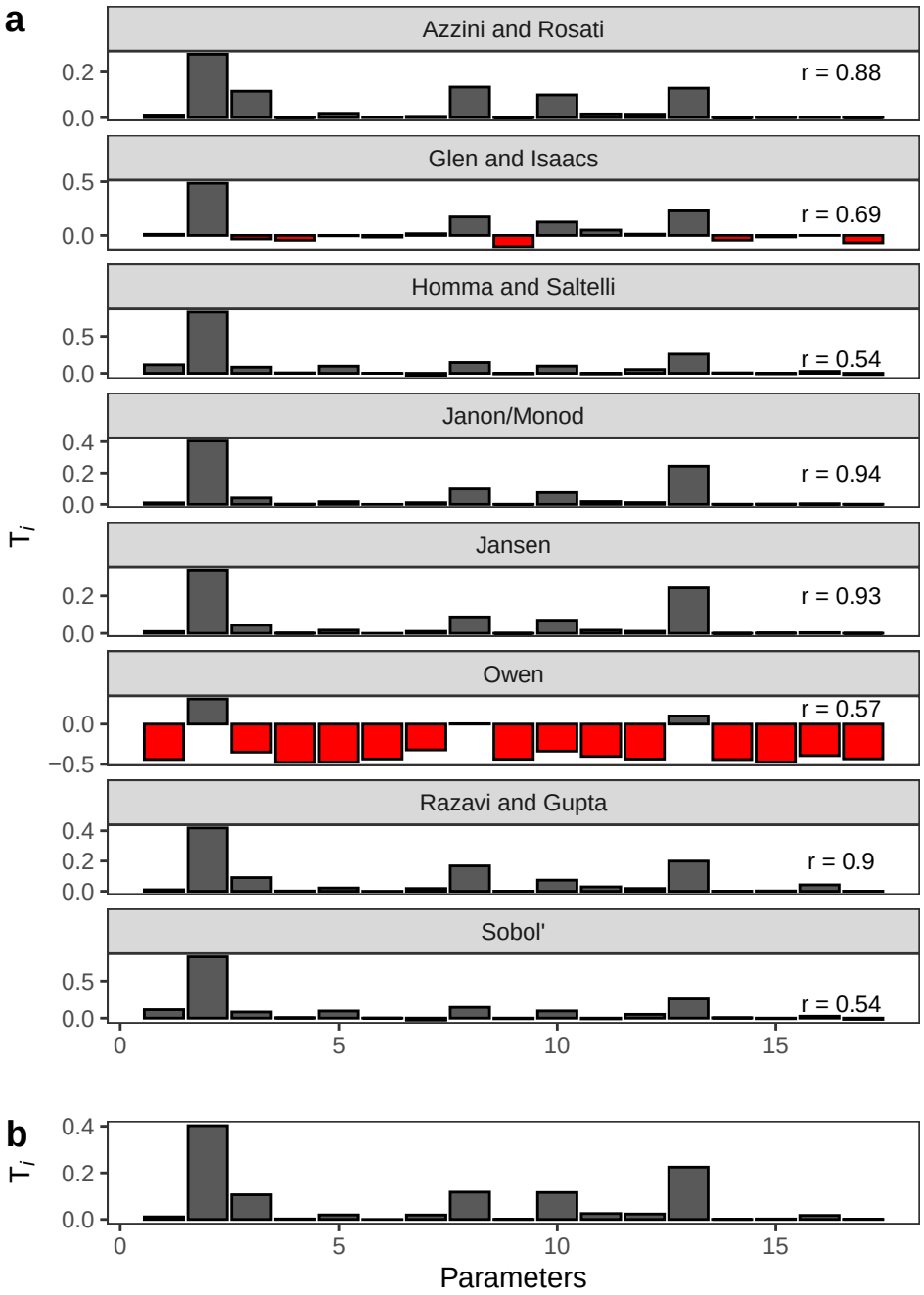
```
## 34.4 GB
```

7 References


Azzini and Rosati
r = 0.88
Glen and Isaacs
r = 0.69
Homma and Saltelli
r = 0.54
Janon/Monod
r = 0.94
Jansen
r = 0.93
Owen
r = 0.57
Razavi and Gupta
r = 0.9
Sobol'
r = 0.54
T_i
Parameters




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



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