

# A comprehensive comparison of total-order estimators for global sensitivity analysis

R code

Arnald Puy

## Contents

<b>1 Functions</b>	<b>3</b>
1.1 Savage scores . . . . .	3
1.2 VARS-TO . . . . .	3
1.3 Sobol' total-order indices . . . . .	5
1.4 Prove that estimators work as expected . . . . .	8
1.5 Differences between random and quasi-random sequences . . . . .	11
<b>2 The metaprocedure</b>	<b>11</b>
<b>3 The model</b>	<b>18</b>
3.1 Settings . . . . .	18
3.2 Sample matrix . . . . .	18
3.3 Define the model . . . . .	19
3.4 Run the model . . . . .	22
3.5 Arrange output . . . . .	23
<b>4 Uncertainty analysis</b>	<b>24</b>
4.1 Boxplots . . . . .	24
4.2 Scatterplots . . . . .	26
4.3 Scatterplots with ratios . . . . .	27
4.4 Negative r values . . . . .	29
4.5 Plot medians . . . . .	35
4.6 Number of simulations per mean Nt/k ratio . . . . .	38
<b>5 Sensitivity analysis</b>	<b>39</b>
5.1 Scatterplots . . . . .	39
5.2 Sobol' indices . . . . .	56
5.3 Plot Sobol' indices . . . . .	57
<b>6 Session information</b>	<b>58</b>

```

# 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", "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("2021-02-05",
           R.version ="4.0.3",
           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 VARS-TO

```
# 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)
}

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

```

```

VY <- var(Y[index.centers])
Ti <- (variogr + covariogr) / VY
output <- data.table::data.table(Ti)
output[, `:=` (parameters = params)]
return(output)
}

```

### 1.3 Sobol' total-order indices

```

# COMPUTATION OF SOBOL' Ti INDICES ----

# SOBOL MATRICES ----

scrambled_sobol <- function(matrices, A, B, C, order, cluster) {
  first <- 1:ncol(A)
  N <- nrow(A)
  if(order == "first") {
    loop <- first
  } else if(order == "second") {
    second <- c(first, utils::combn(1:ncol(A), 2, simplify = FALSE))
    loop <- second
  } else if(order == "third") {
    second <- c(first, utils::combn(1:ncol(A), 2, simplify = FALSE))
    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
  CB.mat <- "CB" %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
}
if(CB.mat == TRUE) {
  Z <- rbind(A, B)
  for(i in loop) {
    CB <- C
    CB[, i] <- B[, i]
    Z <- rbind(Z, CB)
  }
  CB <- Z[(2 * N + 1) : nrow(Z), ]
} else if(CB.mat == FALSE) {
  CB <- NULL
}
final <- rbind(AB, BA, CB)
return(final)
}

sobol_matrices <- function(matrices = c("A", "B", "AB"),
                           N, params, order = "first",
                           method = "QRN", cluster = NULL) {
  k <- length(params)
  n.matrices <- ifelse(any(stringr::str_detect(matrices, "C")) == FALSE, 2, 3)
  if(method == "QRN") {
    df <- randtoolbox::sobol(n = N, dim = k * n.matrices, scrambling = 1)
  } else if(method == "R") {
    df <- replicate(k * n.matrices, stats::runif(N))
  } else if(method == "LHS") {
    df <- lhs::randomLHS(N, n.matrices * k)
  } else {
    stop("method should be either QRN, R or LHS")
  }
  A <- df[, 1:k]
  B <- df[, (k + 1) : (k * 2)]
  if(n.matrices == 3) {
    C <- df[, ((k * 2) + 1):(k * 3)]
  } else {
    C <- NULL
  }
  out <- scrambled_sobol(matrices = matrices,

```

```

      A = A, B = B, C = C,
      order = order, cluster = cluster)
A.mat <- "A" %in% matrices
B.mat <- "B" %in% matrices
C.mat <- "C" %in% matrices
if(A.mat == FALSE) {
  A <- NULL
}
if(B.mat == FALSE) {
  B <- NULL
}
if(C.mat == FALSE) {
  C <- NULL
}
final <- rbind(A, B, C, out)
colnames(final) <- params
return(final)
}

# COMPUTATION OF SOBOL' TOTAL-ORDER INDICES

sobol_Ti <- function(d, N, params, total) {
  m <- matrix(d, nrow = N)
  k <- length(params)
  if(total == "jansen" | total == "homma" | 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 == "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 == "saltelli") {

```

```

Y_A <- m[, 1]
Y_B <- m[, 2]
Y_BA <- m[, -c(1, 2)]
f02 <- (1 / N * sum(Y_A)) ^ 2
VY <- 1 / N * sum(Y_A ^ 2) - f02
Ti <- 1 - ((1 / N * Rfast::colsums(Y_B * Y_BA - f02)) / VY)
}

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 <- Rfast::colsums((Y_B - Y_BA) ^ 2 + (Y_A - Y_AB) ^ 2) /
    Rfast::colsums((Y_A - Y_B) ^ 2 + (Y_BA - Y_AB) ^ 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)
}

```

## 1.4 Prove that estimators work as expected

```

# CHECK THAT ALL TI ESTIMATORS WORK -----
# Settings
estimators <- c("jansen", "homma", "azzini", "monod", "glen", "owen", "saltelli")
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 == "homma" | i == "monod" | i == "glen") {
      matrices <- c("A", "AB")
    } else if(i == "azzini"){
      matrices <- c("A", "B", "AB", "BA")
    } else if(i == "owen") {
      matrices <- c("A", "B", "BA", "CB")
    } else if(i == "saltelli") {
      matrices <- c("A", "B", "BA")
    }
    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 = matrices)
    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.srct dplyr

# 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,

```

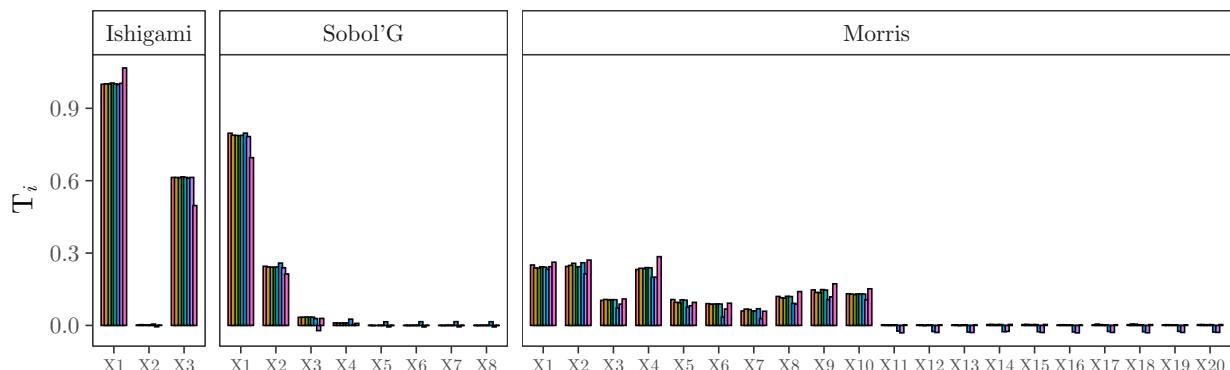
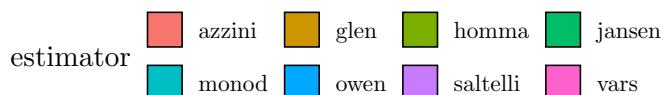
```

        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"] %>%
  setcolororder(., c("estimator", "Function", "Ti", "parameters"))

# 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") +
  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",
        strip.background = element_rect(fill = "white")) +
  guides(fill = guide_legend(nrow = 2,
                             byrow = TRUE))

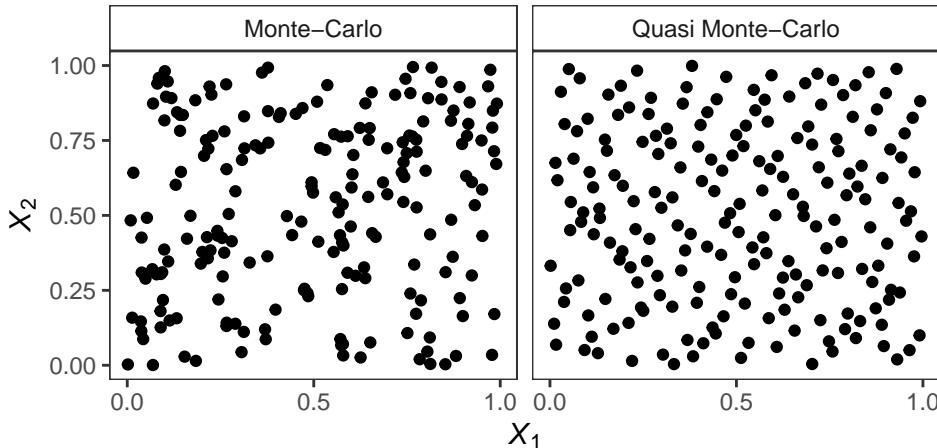
```



## 1.5 Differences between random and quasi-random sequences

```
# 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() +
    theme(strip.background = element_rect(fill = "white"))
```



## 2 The metaprogram

```
# CREATE METAPROGRAM -----
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]))))
  }) +
  labs(color= "Function", linetype = "Function",
       x = expression(italic(x)),
       y = expression(italic(y))) +
  scale_color_discrete(labels = c(expression(f[1](x) == x ^ 3),
                                   expression(paste(f[2](x) == 1, " if ", x>1/2, ", otherwise ",
                                   expression(f[3](x) == (e ^ x - 1) / (e - 1)),
                                   expression(f[4](x) == (10 - 1 / 1.1) ^ -1 * (x + 0.1) ^ -1),
                                   expression(f[5](x) == x),
                                   expression(f[6](x) == 0),
                                   expression(f[7](x) == 4^-~(x - 0.5) ^ 2),
                                   expression(f[8](x) == sin(2^-~pi^-~x) / 2),
                                   expression(f[9](x) == x ^ 2),
                                   expression(f[10](x) == cos(x)))) +
  theme_AP() +
  theme(legend.text.align = 0)

```

```
a
```

```
# CREATE FUNCTION FOR RANDOM DISTRIBUTIONS -----
```

```
# Density function
```

```

sample_distributions_PDF <- list(
  "uniform" = function(x) dunif(x, 0, 1),
  "normal" = function(x) dnorm(x, 0.5, 0.15),
  "beta" = function(x) dbeta(x, 8, 2),
  "beta2" = function(x) dbeta(x, 2, 8),
  "beta3" = function(x) dbeta(x, 2, 0.8),
  "beta4" = function(x) dbeta(x, 0.8, 2),
  "logitnormal" = function(x) dlogitnorm(x, 0, 3.16)
  # Logit-normal, Bates too?
)
```

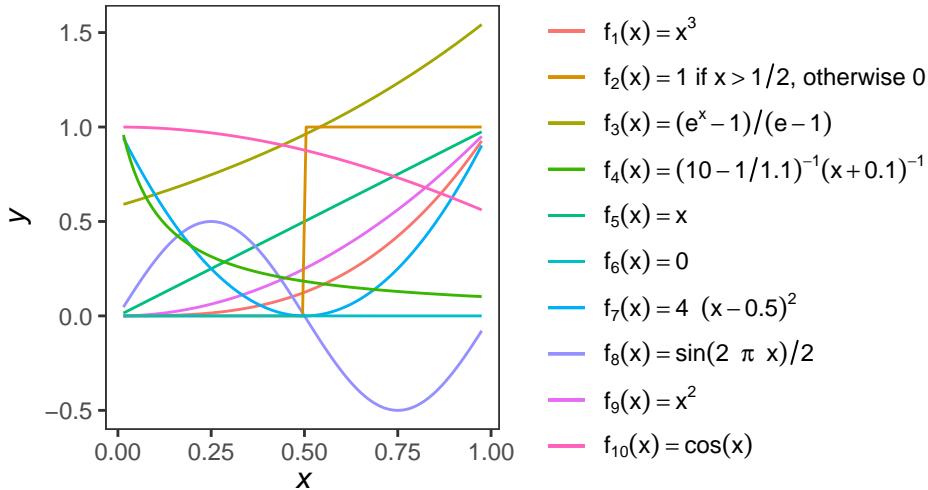


Figure 1: Functions used in the metafunction of Becker (2020).

```

)

# Quantile function
sample_distributions <- list(
  "uniform" = function(x) x,
  "normal" = function(x) qnorm(x, 0.5, 0.15),
  "beta" = function(x) qbeta(x, 8, 2),
  "beta2" = function(x) qbeta(x, 2, 8),
  "beta3" = function(x) qbeta(x, 2, 0.8),
  "beta4" = function(x) qbeta(x, 0.8, 2),
  "logitnormal" = function(x) qlogitnorm(x, 0, 3.16)
  # Logit-normal, Bates too?
)

random_distributions <- function(X, phi) {
  names_ff <- names(sample_distributions)
  if(!phi == length(names_ff) + 1) {
    out <- 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)
}

# PLOT DISTRIBUTIONS -----
names_ff <- names(sample_distributions)
x <- seq(0, 1, .001)

out <- matrix(rep(x, length(names_ff)), ncol = length(names_ff))

```

```

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

dt <- setnames(dt, paste("V", 1:length(names_ff), sep = ""), names_ff) %>%
  .[, x:= x] %>%
  melt(., measure.vars = names_ff)

b <- ggplot(dt, aes(x = x, y = value, group = variable)) +
  geom_line(aes(color = variable)) +
  scale_color_discrete(labels = c("U(0, 1)",
    expression(N[T](0.5, 0.15, 0, 1)),
    "Beta(8, 2)",
    "Beta(2, 8)",
    "Beta(2, 0.8)",
    "Beta(0.8, 2)",
    "Logitnormal(0, 3.16)"),
    name = "Distribution") +
  labs(x = expression(italic(x)),
    y = "PDF") +
  theme_AP() +
  theme(legend.text.align = 0)

```

b

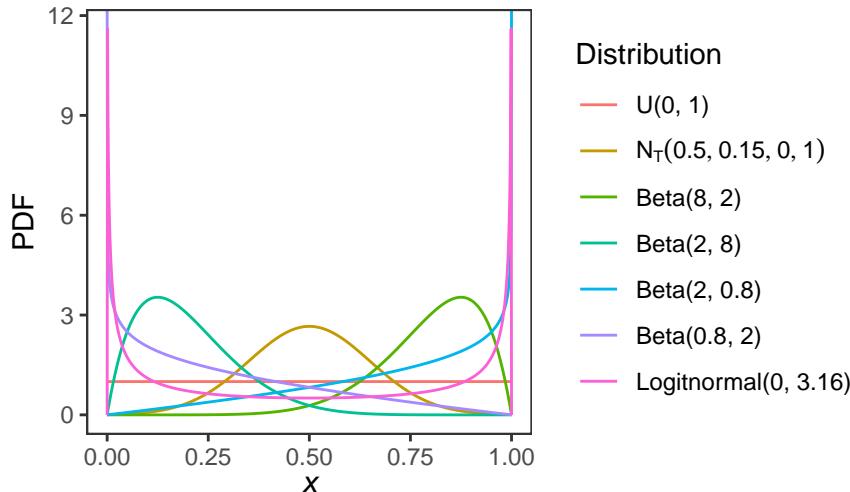
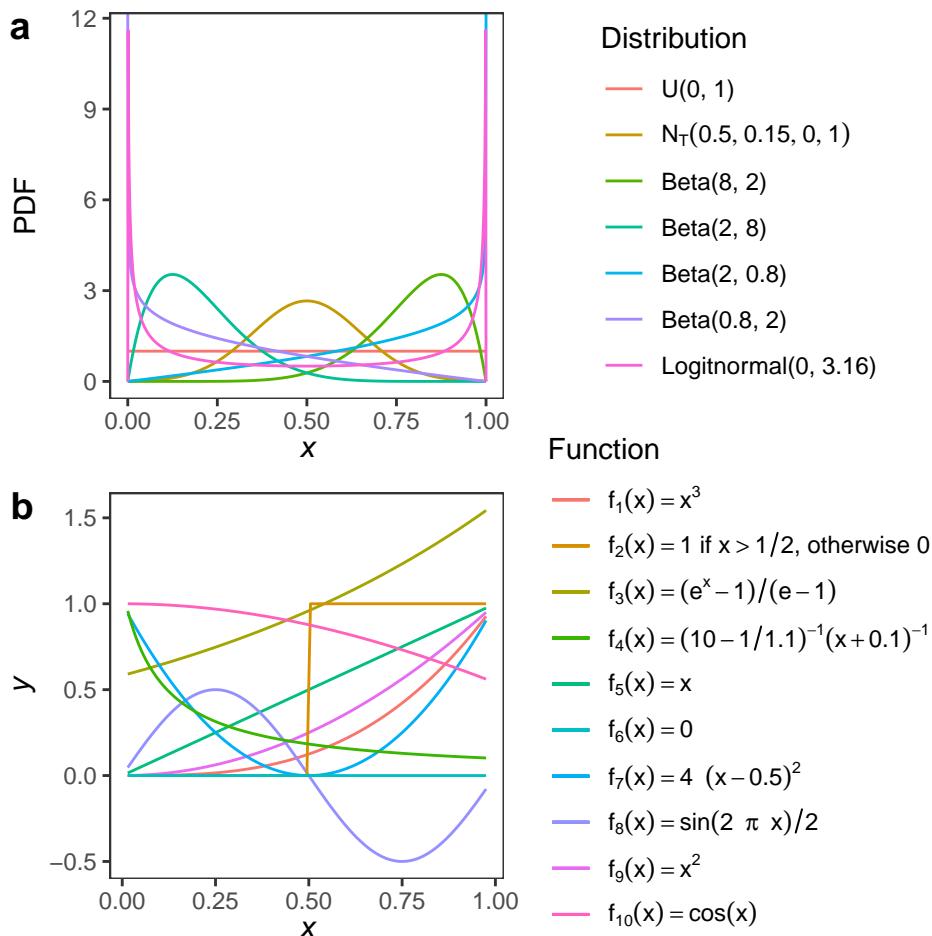


Figure 2: Distributions used in the metafunction of Becker (2020).

---

# MERGE METAFUNCTION PLOT AND DISTRIBUTIONS PLOT -----

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



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

```

N <- 10^4 # Sample size
R <- 10^2 # Number of bootstrap replications
k <- 17 # Number of model inputs
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 = R,
  .[, 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,

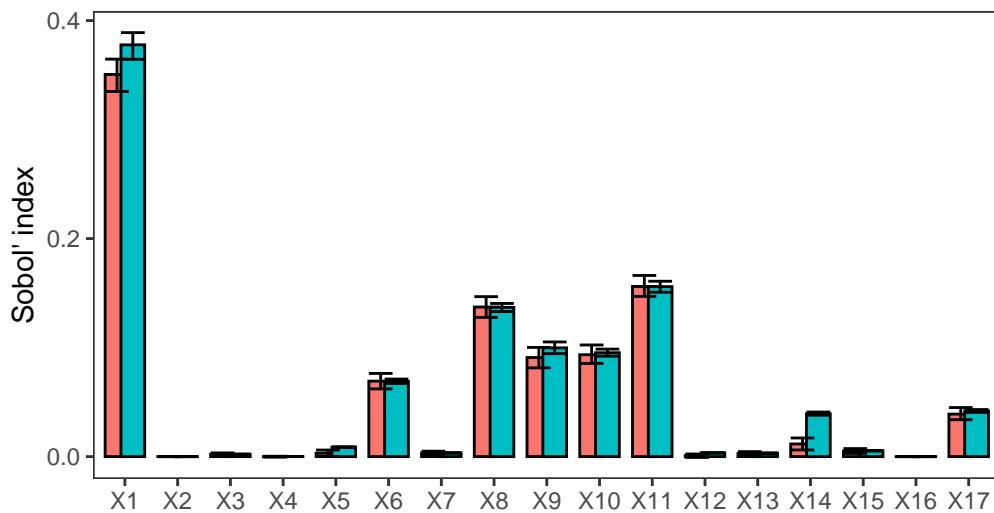
```

```

        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")

```

Sobol' indices  $S_i$   $T_i$



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

```

# 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)
}

```

```

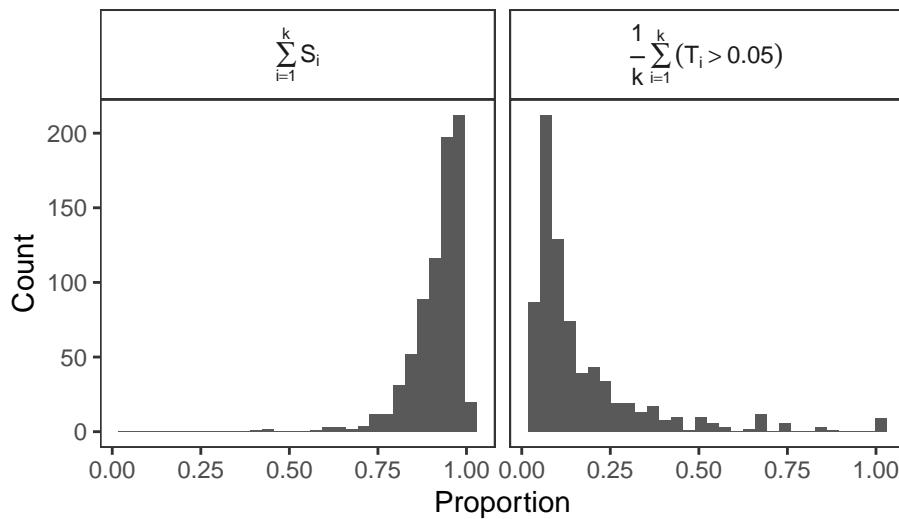
ind <- sobol_indices(Y = Y, N = N, params = params)
out1 <- ind[sensitivity == "Si", sum(original)]
out2 <- ind[sensitivity == "Ti", sum(original > 0.05)/.N]
output <- c(out1, out2)
return(output)
}

# 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)

# 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() +
  theme(strip.background = element_rect(fill = "white"))

```

## `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 <- "first"
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 ----

# Create sample matrix
mat <- sensobol::sobol_matrices(N = N, params = c(params, "N_t", "k"), order = order)
Nt.k.rows <- which(rep(c("A", "B", params, "N_t", "k"), each = N) %in% c("N_t", "k"))
mat <- mat[-Nt.k.rows, ] # Exclude AB matrices for Nt and k

# Create A and B matrices to create the clusters
A <- mat[1:N, ]
B <- mat[(N + 1):(2 * N), ]

# Matrix with clusters (Nt,k), (k_2, k_3, epsilon, phi), (delta, tau)
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 matrices
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] * (2 - 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.saltelli <- apply(mat, 1, function(x) ceiling(x["N_t"] / (x["k"] + 2)))
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.all)]
tmp <- tmp[, N.saltelli:= ifelse(N.vars == 1 | N.vars == 0, ceiling(2 * (4 * k + 1) / (k + 2)), N.saltelli)]
tmp <- tmp[, N.azzini:= ifelse(N.vars == 1 | N.vars == 0, ceiling(2 * (4 * k + 1) / (2 * k + 2)), N.azzini)]
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.saltelli <- apply(tmp, 1, function(x) x["N.saltelli"] * (x["k"] + 2))
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.saltelli, C.azzini, C.vars)

# CREATE MATRICES FOR RANKS AND MAE

mat.ranks <- mat # Define matrix for ranks
rows.delta <- which(rep(c("A", "B", params), each = N) %in% "delta") # Rows for delta
rows.nt.k <- (nrow(mat) - N + 1):nrow(mat) # Rows for cluster Nt_k
mat.mae <- mat[-c(rows.delta, rows.nt.k)] # Define matrix for mae

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

```

### 3.3 Define the model

```

# DEFINE MODEL ----

model_Ti <- function(k, N.all, N.azzini, N.saltelli, N.vars, h,
                      N.high, k_2, k_3, epsilon, phi, delta, tau,
                      ranks = TRUE) {
  ind <- list()
  estimators <- c("jansen", "homma", "monod", "azzini", "glen", "owen", "vars", "saltelli")
  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)
saltelli <- sobol_matrices(N = N.saltelli, params = paste("X", 1:k, sep = ""),
                           matrices = c("A", "B", "BA"), 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,
                                                 saltelli,
                                                 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)
Nt.saltelli <- N.saltelli * (k + 2)
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)
lg.saltelli <- (max(lg.vars) + 1):(max(lg.vars) + Nt.saltelli)

```

```

for(i in estimators) {
  if(i == "jansen" | i == "homma" | i == "monod" | i == "glen") {
    y <- output[lg.all.but.azzini]
    n <- N.all
  } else if(i == "saltelli") {
    y <- output[lg.saltelli]
    n <- N.saltelli
  } else if(i == "azzini") {
    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)
# Replace Na
for (i in seq_along(out.wide))
  set(out.wide, i=which(is.na(out.wide[[i]])), j = i, value = 0)

# COMPUTE RANKS
if(ranks == TRUE) {
  if(delta == 1) { # kendall tau
    final <- out.wide[, .(value = pcaPP::cor.fk(N, n)), estimator]
  } else { # Savage ranks
    final <- out.wide[, lapply(.SD, savage_scores), .SDcols = c("N", "n"), estimator][
}

```

```

        , .(value = cor(N, n)), estimator]
    }

    # COMPUTE MAE
} else if(ranks == FALSE) {
    final <- out.wide[, .(value = mean(abs(N - n))), estimator]
}
return(final)
}

```

### 3.4 Run the model

```

# DEFINE THE PARALLEL COMPUTING -----
# Define parallel computing
cl <- makeCluster(n_cores)
registerDoParallel(cl)

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

# RUN MODEL FOR MAE -----
# Compute for MAE
Y.mae <- foreach(i=1:nrow(mat.mae),
                     .packages = c("sensobol", "data.table", "pcaPP",
                                  "logitnorm", "dplyr")) %dopar%
{

```

```

model_Ti(k = mat.mae[[i, "k"]],  

         k_2 = mat.mae[[i, "k_2"]],  

         k_3 = mat.mae[[i, "k_3"]],  

         epsilon = mat.mae[[i, "epsilon"]],  

         phi = mat.mae[[i, "phi"]],  

         delta = mat.mae[[i, "delta"]],  

         tau = mat.mae[[i, "tau"]],  

         N.all = mat.mae[[i, "N.all"]],  

         N.azzini = mat.mae[[i, "N.azzini"]],  

         N.saltelli = mat.mae[[i, "N.saltelli"]],  

         N.vars = mat.mae[[i, "N.vars"]],  

         N.high = N.high,  

         h = h,  

         ranks = FALSE)  

}  
  

# STOP PARALLEL COMPUTING -----  
  

stopCluster(cl)

```

### 3.5 Arrange output

```

# ARRANGE OUTPUT -----  
  

out.ranks <- rbindlist(Y.ranks, idcol = "row")[, type:= "r"]  

out.mae <- rbindlist(Y.mae, idcol = "row")[, type:= "MAE"]  
  

full.output <- rbind(merge(mat.ranks[, row:= 1:N], out.ranks),  

                      merge(mat.mae[, row:= 1:N], out.mae)) %>%  

  .[, Nt:= ifelse(estimator == "azzini" | estimator == "owen", N.azzini * (2 * k + 2),  

                  ifelse(estimator == "vars", N.vars * (k * ((1 / h) - 1) + 1),  

                      ifelse(estimator == "saltelli", N.saltelli * (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",  

                                      ifelse(estimator %in% "glen", "Glen and Isaacs",  

                                          ifelse(estimator %in% "owen", "pseudo-Owen",  

                                              ifelse(estimator %in% "saltelli",  
  

# Define A matrices  

A.ranks <- full.output[type == "r", .SD[1:N], estimator][, ratio:= Nt / k]  

A.mae <- full.output[type == "MAE", .SD[1:N], estimator][, ratio:= Nt / k]  

A.both <- rbind(A.ranks, A.mae)  
  

# Define matrix for total variance  

VY.dt <- full.output[row %in% c(1:N)][

```

```
, .(VY = 1 / N * sum((value - ((1 / N) * sum(value))) ^ 2)),  
estimator]
```

# EXPORT OUTPUT -----

```
fwrite(full.output, "full.output.csv")  
fwrite(A.ranks, "A.ranks.csv")  
fwrite(A.mae, "A.mae.csv")
```

## 4 Uncertainty analysis

### 4.1 Boxplots

# QUANTILES -----

```
tmp <- A.both[, .(median = median(value),  
v.low.co = quantile(value, 0.025),  
low.ci = quantile(value, 0.25),  
high.ci = quantile(value, 0.75),  
v.high.ci = quantile(value, 0.975)), .(estimator, type)]
```

```
tmp[tmp[, do.call(order, .SD)], .SDcols = c("type", "median")]]
```

	estimator	type	median	v.low.co	low.ci	high.ci
## 1:	Janon/Monod	MAE	0.01004761	0.003319945	0.006388092	0.01754499
## 2:	Jansen	MAE	0.01031266	0.003355846	0.006335840	0.01859782
## 3:	Azzini and Rosati	MAE	0.01295522	0.004452327	0.008123290	0.02143318
## 4:	Razavi and Gupta	MAE	0.03480095	0.003706873	0.011714348	0.16019528
## 5:	Glen and Isaacs	MAE	0.04350877	0.019484431	0.033749134	0.05920599
## 6:	Homma and Saltelli	MAE	0.05778854	0.019501235	0.036073114	0.10447928
## 7:	pseudo-Owen	MAE	0.52212030	0.072787352	0.243572464	0.91950509
## 8:	Saltelli	MAE	0.76276603	0.053651085	0.307395459	2.02673511
## 9:	pseudo-Owen	r	0.23493101	-0.200000000	0.064951715	0.47490203
## 10:	Homma and Saltelli	r	0.31006634	-0.117014702	0.113120636	0.54092139
## 11:	Saltelli	r	0.31249111	-0.136020104	0.118126877	0.53034034
## 12:	Glen and Isaacs	r	0.37681104	-0.014843481	0.176284001	0.57857354
## 13:	Azzini and Rosati	r	0.84522052	0.591437851	0.763476795	0.91851852
## 14:	Janon/Monod	r	0.89914630	0.671730560	0.834336880	0.95490787
## 15:	Jansen	r	0.90325312	0.673011739	0.836494387	0.95701282
## 16:	Razavi and Gupta	r	0.90739346	0.639853935	0.835005530	0.96448563
	v.high.ci					
## 1:	0.05513422					
## 2:	0.07563105					
## 3:	0.06693832					
## 4:	15.09093976					
## 5:	0.12025340					
## 6:	0.55294638					

```

## 7: 1.88420331
## 8: 16.08920143
## 9: 0.89412248
## 10: 0.92840515
## 11: 0.92341341
## 12: 0.97179194
## 13: 1.00000000
## 14: 1.00000000
## 15: 1.00000000
## 16: 1.00000000

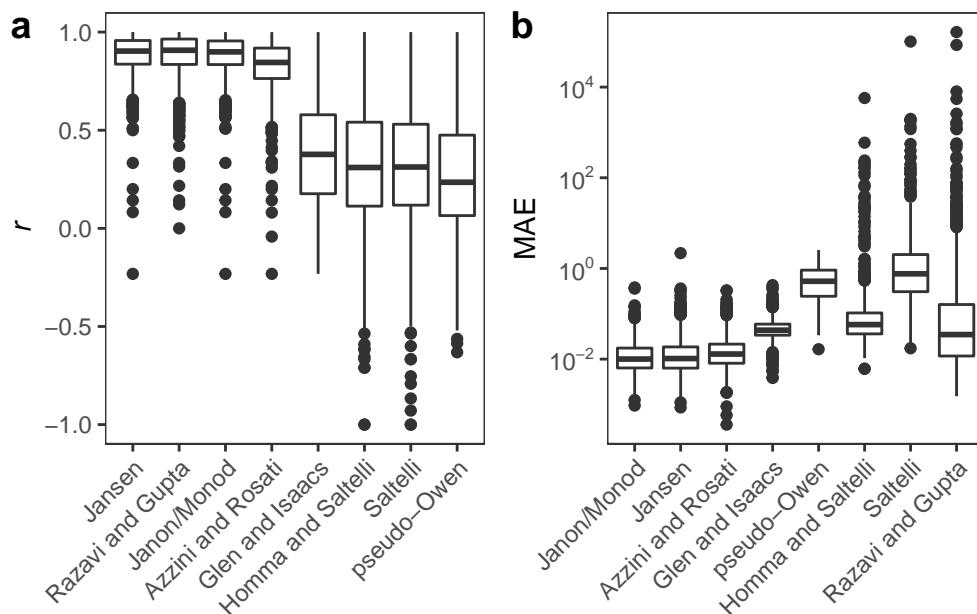
# PLOT BOXPLOT ----

a <- ggplot(A.ranks, aes(x = reorder(estimator, -value), value)) +
  geom_boxplot() +
  labs(x = "",
       y = expression(italic(r))) +
  theme_AP() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

b <- ggplot(A.mae, aes(x = reorder(estimator, value), value)) +
  geom_boxplot() +
  labs(x = "",
       y = "MAE") +
  scale_y_log10(breaks = scales::trans_breaks("log10", function(x) 10^x),
                 labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  theme_AP() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

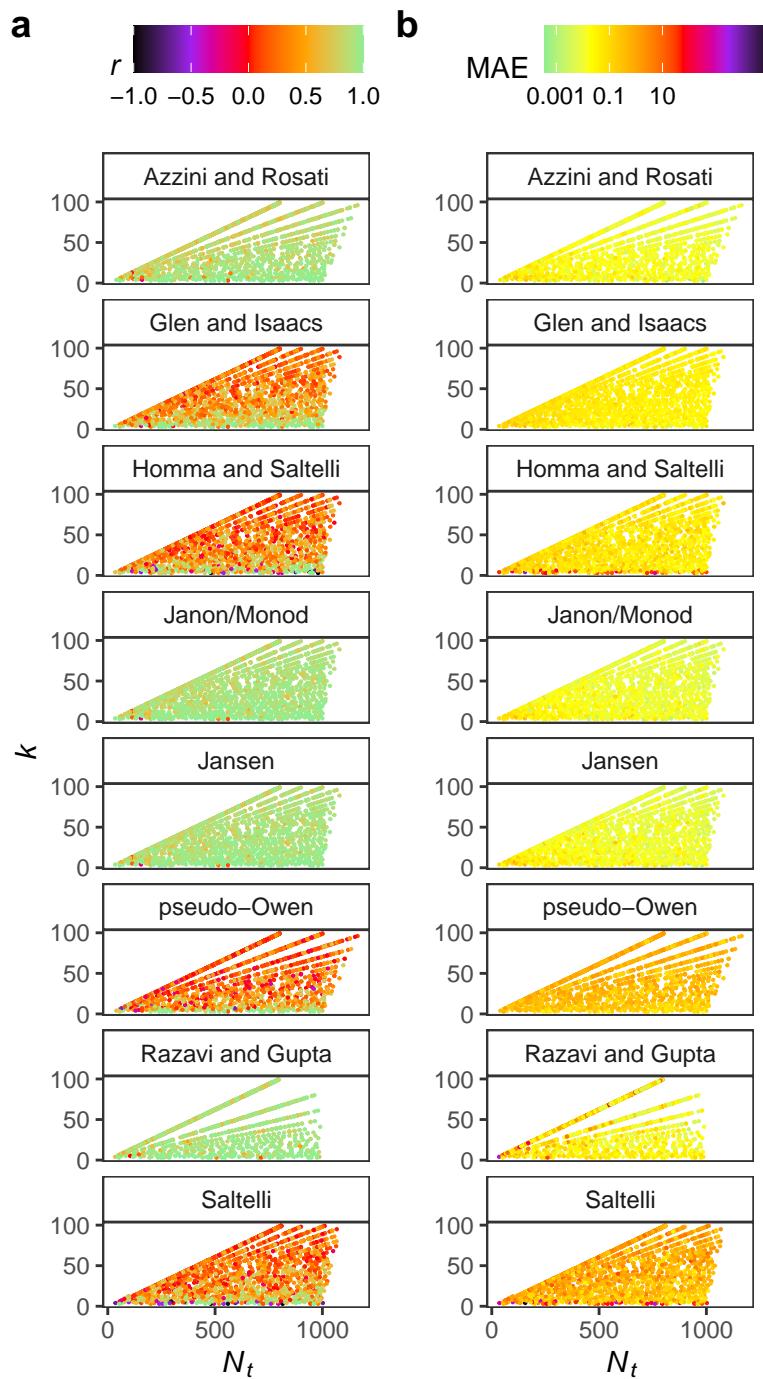
plot_grid(a, b, ncol = 2, labels = "auto")

```



## 4.2 Scatterplots

```
# PLOT SCATTERPLOTS -----  
  
a <- ggplot(A.ranks, aes(Nt, k, color = value)) +  
  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 = 1) +  
  theme_AP() +  
  theme(legend.position = "top",  
        strip.background = element_rect(fill = "white"))  
  
b <- ggplot(A.mae, aes(Nt, k, color = value)) +  
  geom_point(size = 0.1) +  
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +  
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +  
  scale_colour_gradientn(colours = c("lightgreen", "yellow", "orange", "red", "purple", "black",  
                                    name = "MAE",  
                                    trans = "log",  
                                    breaks = c(0.001, 0.1, 10),  
                                    labels = c(0.001, 0.1, 10)) +  
  labs(x = expression(italic(N[t])),  
       y = "") +  
  facet_wrap(~estimator,  
            ncol = 1) +  
  theme_AP() +  
  theme(legend.position = "top",  
        strip.background = element_rect(fill = "white"))  
  
plot_grid(a, b, ncol = 2, labels = "auto")
```



### 4.3 Scatterplots with ratios

```
# PLOT SCATTERPLOTS WITH RATIOS -----
a <- A.ranks[, ratio:= Nt / k] %>%
  ggplot(., aes(ratio, value)) +
  geom_point(alpha = 0.1, size = 0.2) +
  facet_wrap(~estimator,
```

```

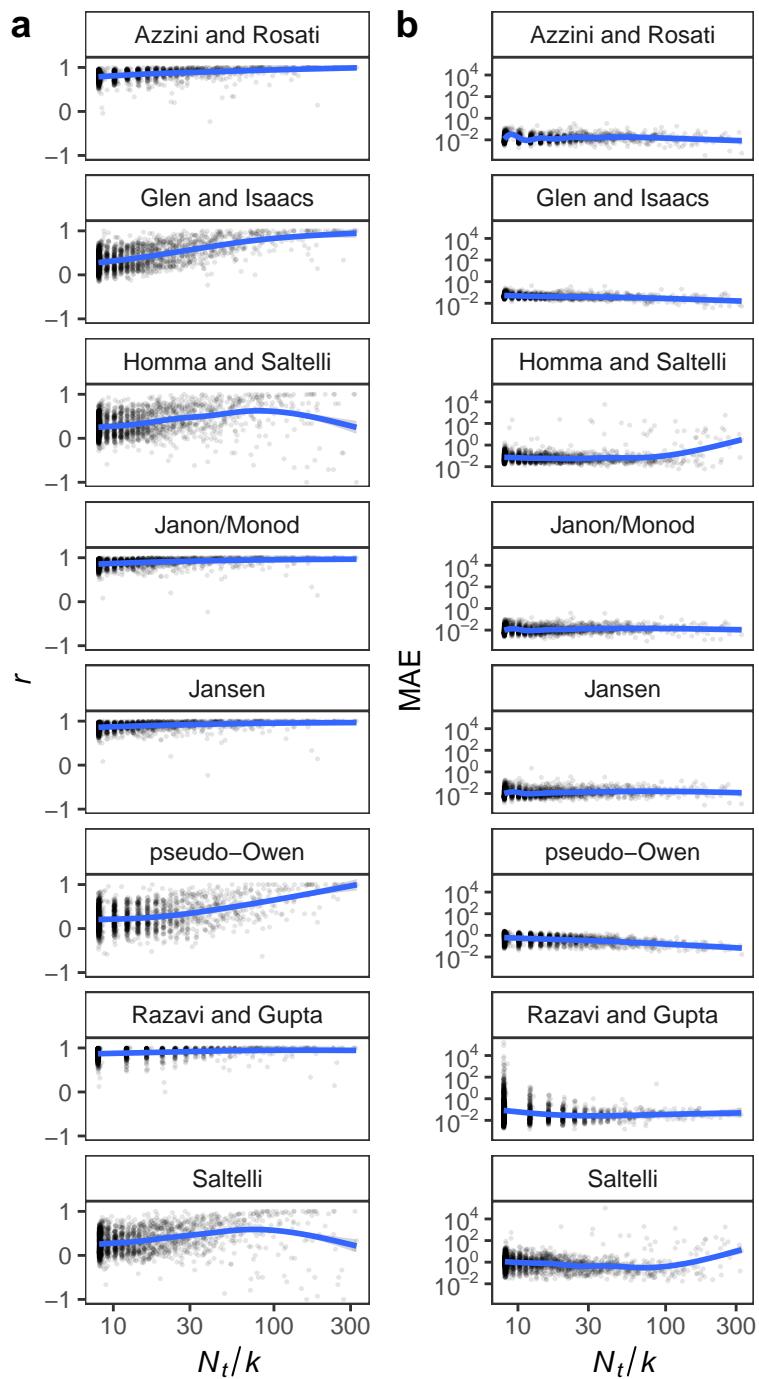
        ncol = 1) +
geom_smooth() +
labs(x = expression(italic(N[t])/k)),
y = expression(italic(r))) +
scale_x_log10() +
scale_y_continuous(breaks = pretty_breaks(n = 3)) +
theme_AP() +
theme(strip.background = element_rect(fill = "white"))

b <- A.mae[, ratio:= Nt / k] %>%
ggplot(., aes(ratio, value)) +
geom_point(alpha = 0.1, size = 0.2) +
facet_wrap(~estimator,
        ncol = 1) +
geom_smooth() +
labs(x = expression(italic(N[t])/k)),
y = "MAE") +
scale_x_log10() +
scale_y_log10(breaks = scales::trans_breaks("log10", function(x) 10^x),
              labels = scales::trans_format("log10", scales::math_format(10^.x))) +
theme_AP() +
theme(strip.background = element_rect(fill = "white"))

plot_grid(a, b, ncol = 2, labels = "auto")

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

```



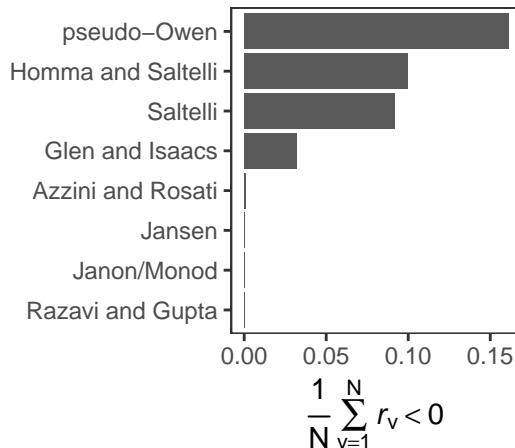
#### 4.4 Negative r values

```
# PLOT PROPORTION OF NEGATIVE R -----
A.ranks[, sum(value < 0)/ .N, estimator] %>%
  ggplot(., aes(reorder(estimator, V1), V1)) +
  geom_bar(stat = "identity") +
  coord_flip()
```

```

  labs(y = expression(frac(1, N)~sum(italic(r)[v] < 0, v==1, N)),
       x = "") +
  theme_AP()

```




---

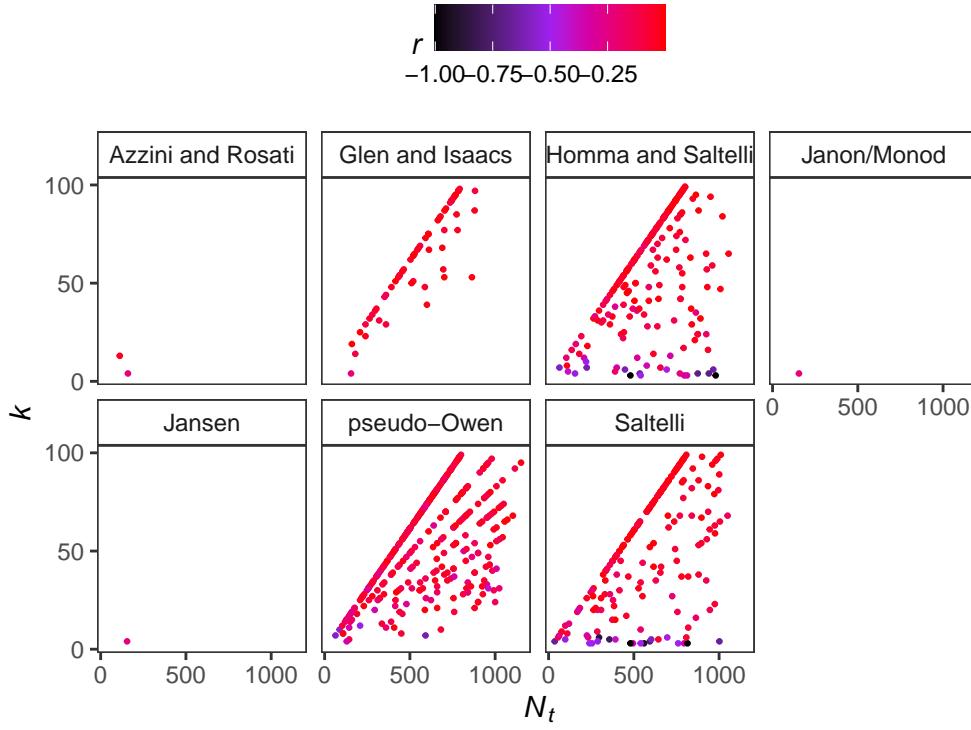
# MAP VALUES WITH NEGATIVE R -----

```

index.neg <- A.ranks[, .I[value < 0]]

A.ranks[index.neg] %>%
  ggplot(., aes(Nt, k, color = value)) +
  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",
        strip.background = element_rect(fill = "white"))

```



```
# MODEL TO RETRIEVE SIMULATIONS THAT YIELDED R < 0 ----

model_Ti2 <- function(k, N.all, N.azzini, N.saltelli, N.vars, h,
                      N.high, k_2, k_3, epsilon, phi, delta, tau) {
  ind <- list()
  estimators <- c("jansen", "homma", "monod", "azzini", "glen", "owen", "vars", "saltelli")
  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)
  saltelli <- sobol_matrices(N = N.saltelli, params = paste("X", 1:k, sep = ""),
                            matrices = c("A", "B", "BA"), 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,
                                                 saltelli,
                                                 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)
Nt.saltelli <- N.saltelli * (k + 2)
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)
lg.saltelli <- (max(lg.vars) + 1):(max(lg.vars) + Nt.saltelli)

for(i in estimators) {
  if(i == "jansen" | i == "homma" | i == "monod" | i == "glen") {
    y <- output[lg.all.but.azzini]
    n <- N.all
  } else if(i == "saltelli") {
    y <- output[lg.saltelli]
    n <- N.saltelli
  } else if(i == "azzini") {
    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")
  return(out.wide)
}

A.ranks <- A.ranks[, ID:= .I]
rowsToExtract <- A.ranks[value < 0 & estimator %in% c("Saltelli", "pseudo-Owen", "Homma and S
                           "Glen and Isaacs")][, ID]

prove <- A.ranks[ID %in% rowsToExtract]

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

# Compute
Y.ti2 <- foreach(i=1:nrow(prove),
                  .packages = c("sensobol", "data.table", "pcaPP",
                               "logitnorm", "dplyr")) %dopar%
{
  model_Ti2(k = prove[[i, "k"]],
             k_2 = prove[[i, "k_2"]],
             k_3 = prove[[i, "k_3"]],
             epsilon = prove[[i, "epsilon"]],
             phi = prove[[i, "phi"]],
             delta = prove[[i, "delta"]],
             tau = prove[[i, "tau"]],
             N.all = prove[[i, "N.all"]],
             N.azzini = prove[[i, "N.azzini"]],
             N.saltelli = prove[[i, "N.saltelli"]],
             N.vars = prove[[i, "N.vars"]],
             N.high = N.high,
             h = h)
}

```

```

}

# Stop parallel cluster
stopCluster(cl)

# PLOT DISTRIBUTION OF R<0 AND R>1 ----

# Arrange output
names(Y.ti2) <- rowsToExtract

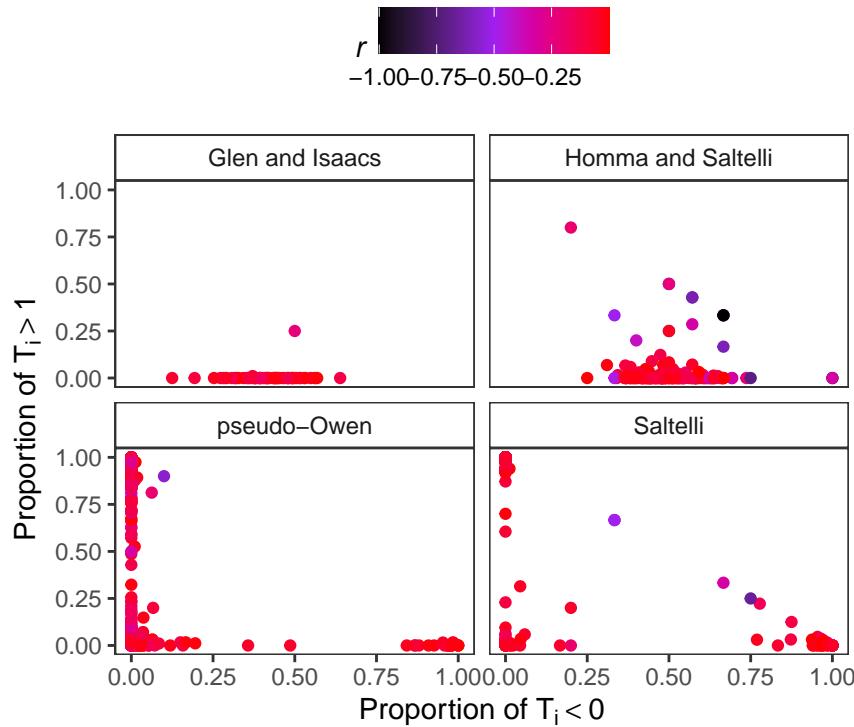
out.neg <- rbindlist(Y.ti2, idcol = "ID") %>%
  .[, ID:= as.numeric(ID)]

out.dt <- out.neg[, 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 Gitterman",
                                                     ifelse(estimator %in% "owen", "Owen and Hedges",
                                                       ifelse(estimator %in% "margossian", "Margossian et al.", "Margossian et al."))))))]

dd <- out.dt[, .(prop.negative = sum(n < 0) / .N,
                 prop.above.one = sum(n > 1) / .N), .(estimator, ID)]

merge(dd, prove, by = c("ID", "estimator")) %>%
  ggplot(., aes(prop.negative, prop.above.one, color = value)) +
  geom_point() +
  scale_colour_gradientn(colours = c("black", "purple", "red"),
                         name = expression(italic(r))) +
  labs(y = expression(paste("Proportion of ", T[i] > 1)),
       x = expression(paste("Proportion of ", T[i] < 0))) +
  facet_wrap(~estimator) +
  theme_AP() +
  theme(legend.position = "top",
        strip.background = element_rect(fill = "white"))

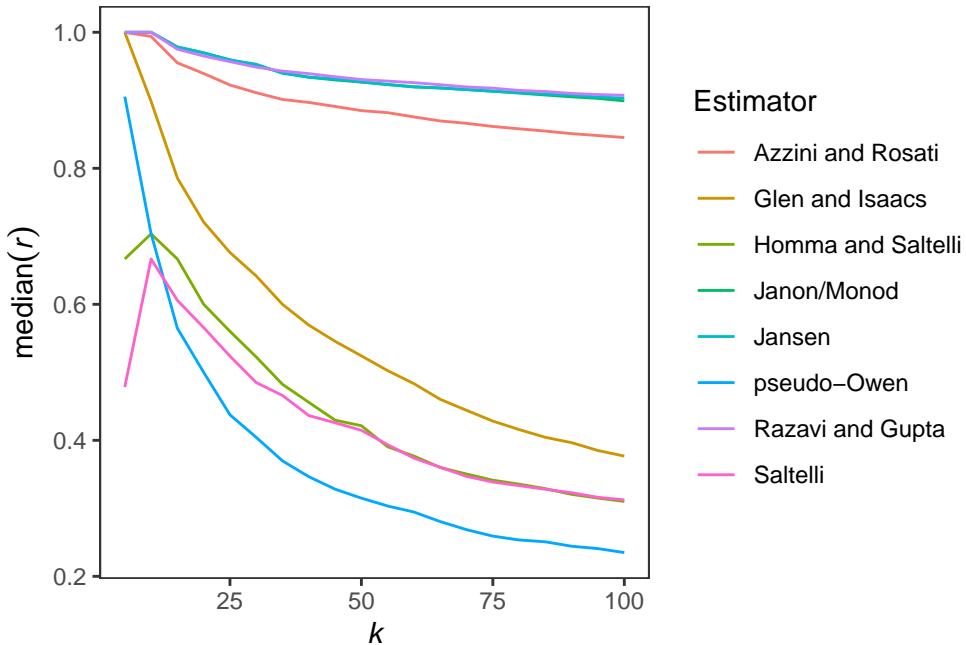
```



## 4.5 Plot medians

```
# PLOT MEDIANS -----
# FOR RANKS-----
vv <- seq(5, 100, 5)
dt <- lapply(vv, function(x) A.ranks[k <= x, median(value), 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() +
  theme(strip.background = element_rect(fill = "white"))
```



```

# Median Nt/k
dt.tmp <- A.ranks[, .(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.ranks <- list()
for(i in 1:nrow(indices)) {
  out.ranks[[i]] <- A.ranks[ratio > indices[i, 1] & ratio < indices[i, 2]]
}

names(out.ranks) <- rowmeans(indices)

# Plot
median.ranks <- lapply(out.ranks, function(x) x[, median(value, na.rm = TRUE), estimator]) %>%
  rbindlist(., idcol = "N") %>%
  .[, N:= as.numeric(N)] %>%
  .[, type:= "r"]

# FOR MAE-----
dt <- lapply(vv, function(x) A.mae[k <= x, median(value), estimator])
names(dt) <- vv

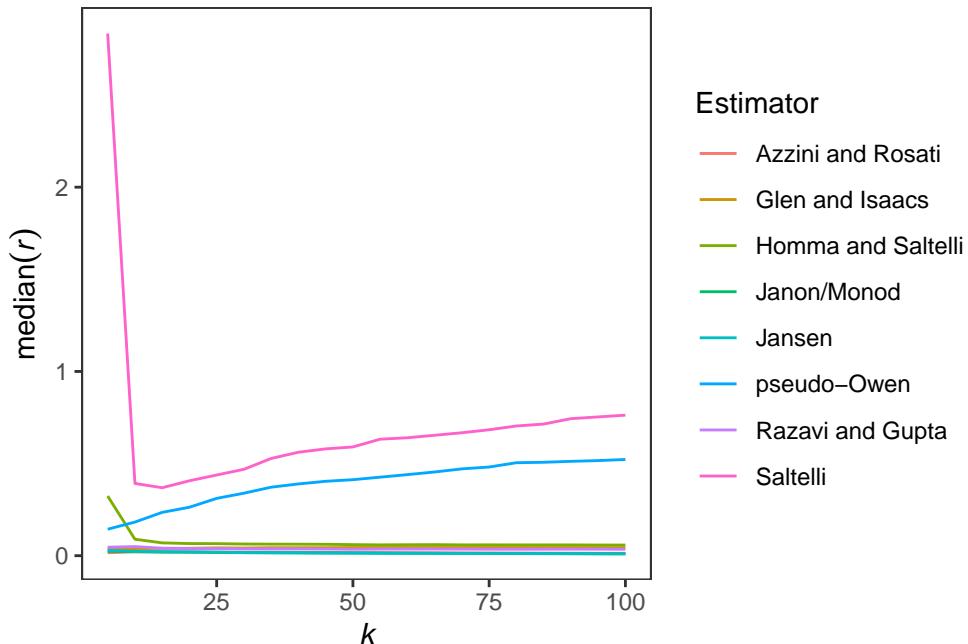
rbindlist(dt, idcol = "k") %>%
  .[, k:= as.numeric(k)] %>%
  ggplot(., aes(k, V1, color = estimator)) +
  scale_color_discrete(name = "Estimator") +
  labs(x = expression(italic(k)),

```

```

y = expression(median(italic(r))) +
geom_line() +
theme_AP()

```



```

# Median Nt/k
dt.tmp <- A.mae[, .(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.mae <- list()
for(i in 1:nrow(indices)) {
  out.mae[[i]] <- A.mae[ratio > indices[i, 1] & ratio < indices[i, 2]]
}

names(out.mae) <- rowmeans(indices)

# Plot
median.mae <- lapply(out.mae, function(x) x[, median(value, na.rm = TRUE), estimator]) %>%
  rbindlist(., idcol = "N") %>%
  .[, N:= as.numeric(N)] %>%
  .[, type:= "MAE"]

# PLOT ALL

a <- ggplot(median.ranks, aes(N, V1, group = estimator, color = estimator)) +
  geom_line() +
  labs(x = expression(italic(N[t])/k)),

```

```

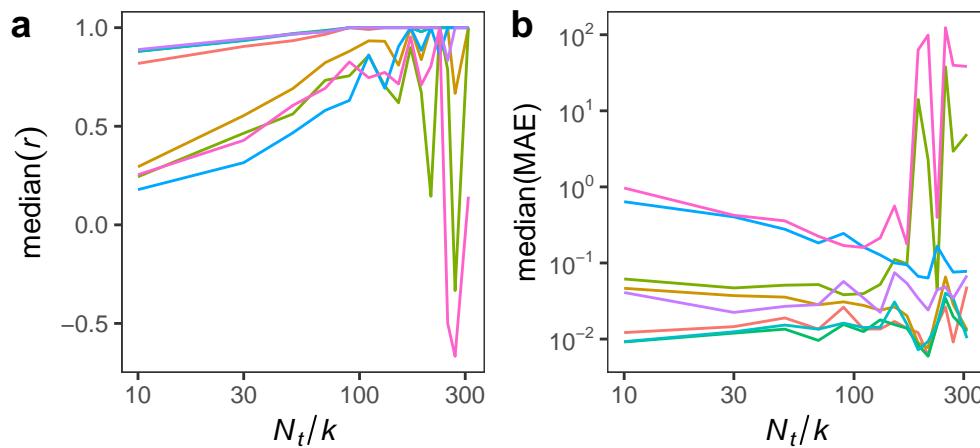
y = expression(median(italic(r)))) +
scale_color_discrete(name = "Estimator") +
scale_x_log10() +
theme_AP() +
theme(legend.position = "none")

b <- ggplot(median.mae, aes(N, V1, group = estimator, color = estimator)) +
geom_line() +
labs(x = expression(italic(N[t]/k)),
y = "median(MAE)") +
scale_color_discrete(name = "Estimator") +
scale_x_log10() +
scale_y_log10(breaks = scales::trans_breaks("log10", function(x) 10^x),
labels = scales::trans_format("log10", scales::math_format(10^.x))) +
theme_AP() +
theme(legend.position = "none")

legend <- get_legend(a + theme(legend.position = "top") +
guides(color = guide_legend(nrow = 3, byrow = TRUE)))
bottom <- plot_grid(a, b, ncol = 2, labels = "auto")
plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.3, 0.7))

    — Azzini and Rosati — Glen and Isaacs — Homma and Saltelli
Estimator — Janon/Monod — Jansen — pseudo-Owen
    — Razavi and Gupta — Saltelli

```



## 4.6 Number of simulations per mean $N_t/k$ ratio

```

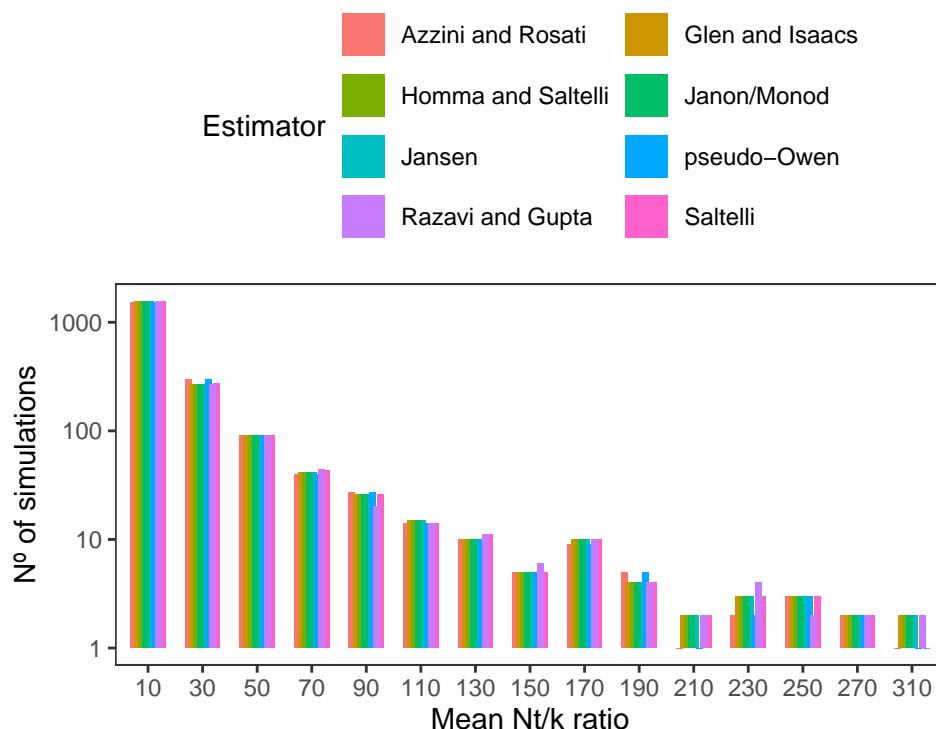
# PLOT MEAN NT/K RATIO AS A FUNCTION OF N° SIMULATIONS -----
rbindlist(out.ranks, idcol = "samples")[, .N, .(estimator, samples)] %>%
  .[, samples:= factor(samples, levels = rowmeans(indices))] %>%

```

```

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

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

scatter.mae <- melt(A.mae, measure.vars = c("k[2]", "k[3]", "epsilon", "phi", "delta", "tau"),
                     value.name = "value.parameters") %>%

```

```

split(., .$estimator)

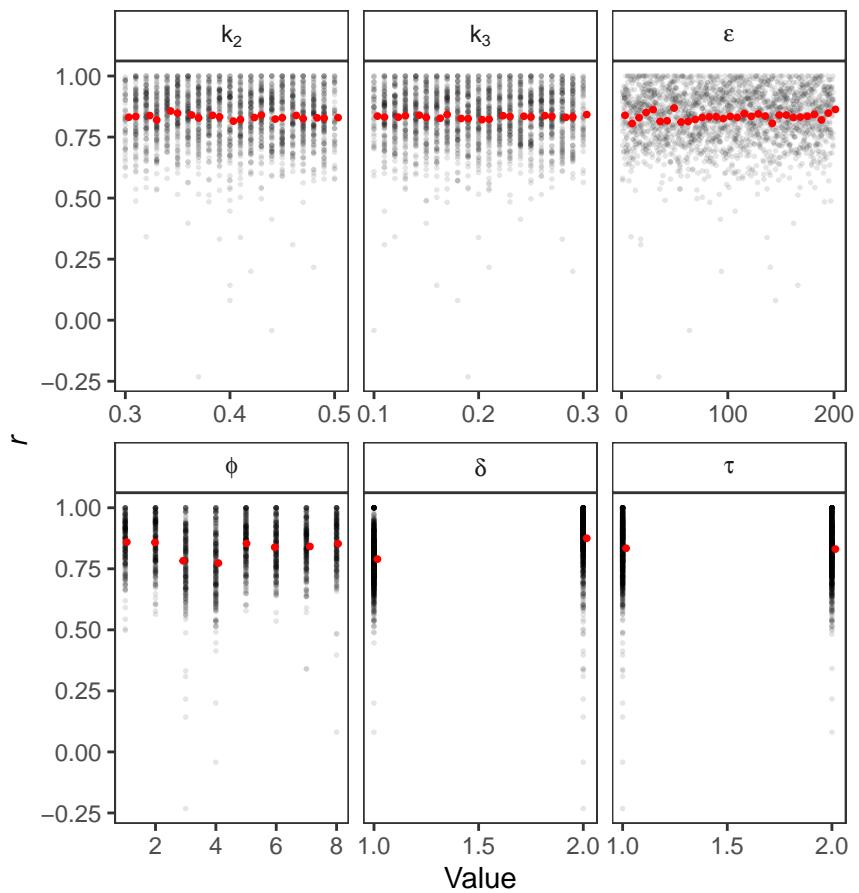
gg.ranks <- list()
for(i in names(scatter.ranks)) {
  gg.ranks[[i]] <- ggplot(scatter.ranks[[i]], aes(value.parameters, value)) +
    geom_point(alpha = 0.1, size = 0.3) +
    facet_wrap(~ variable,
               scales = "free_x",
               labeller = label_parsed,
               ncol = 3) +
    scale_color_manual(values = c("#00BFC4", "#F8766D")) +
    scale_x_continuous(breaks = pretty_breaks(n = 3)) +
    stat_summary_bin(fun = "mean", geom = "point",
                     colour = "red", size = 0.7) +
    theme_AP() +
    labs(x = "Value", y = expression(italic(r))) +
    theme(strip.background = element_rect(fill = "white")) +
    ggttitle(names(scatter.ranks[i]))
}

gg.mae <- list()
for(i in names(scatter.mae)) {
  gg.mae[[i]] <- ggplot(scatter.mae[[i]], aes(value.parameters, value)) +
    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)) +
    stat_summary_bin(fun = "mean", geom = "point",
                     colour = "red", size = 0.7) +
    theme_AP() +
    scale_y_log10() +
    labs(x = "Value", y = "MAE") +
    theme(strip.background = element_rect(fill = "white")) +
    ggttitle(names(scatter.mae[i]))
}

gg.ranks

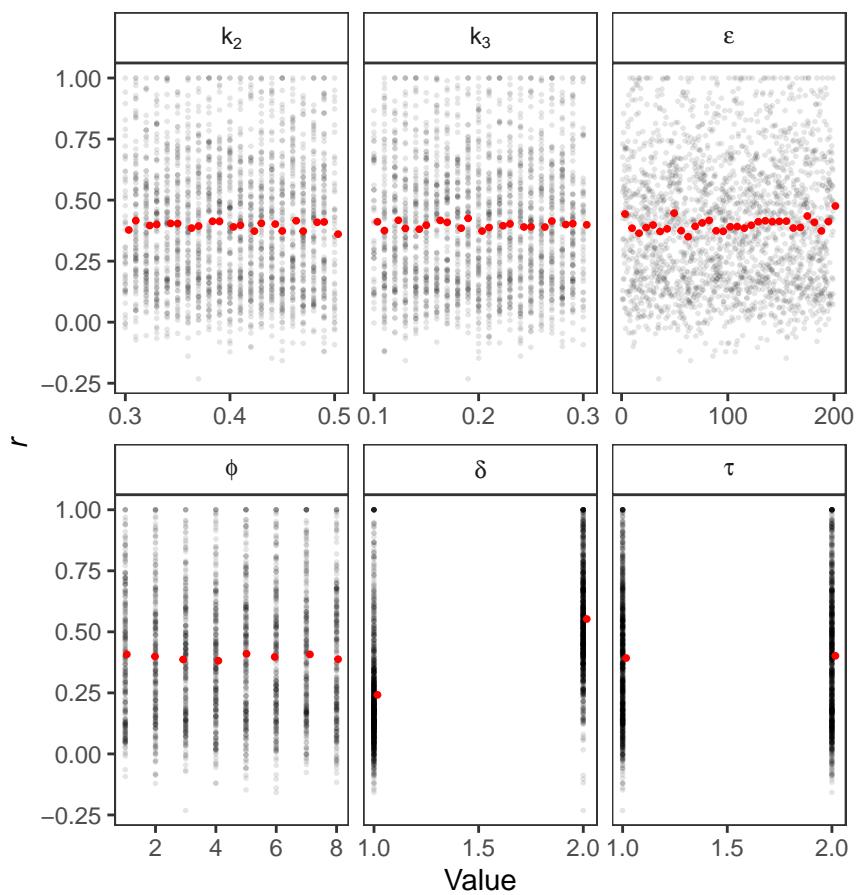
## $`Azzini and Rosati`
```

## Azzini and Rosati



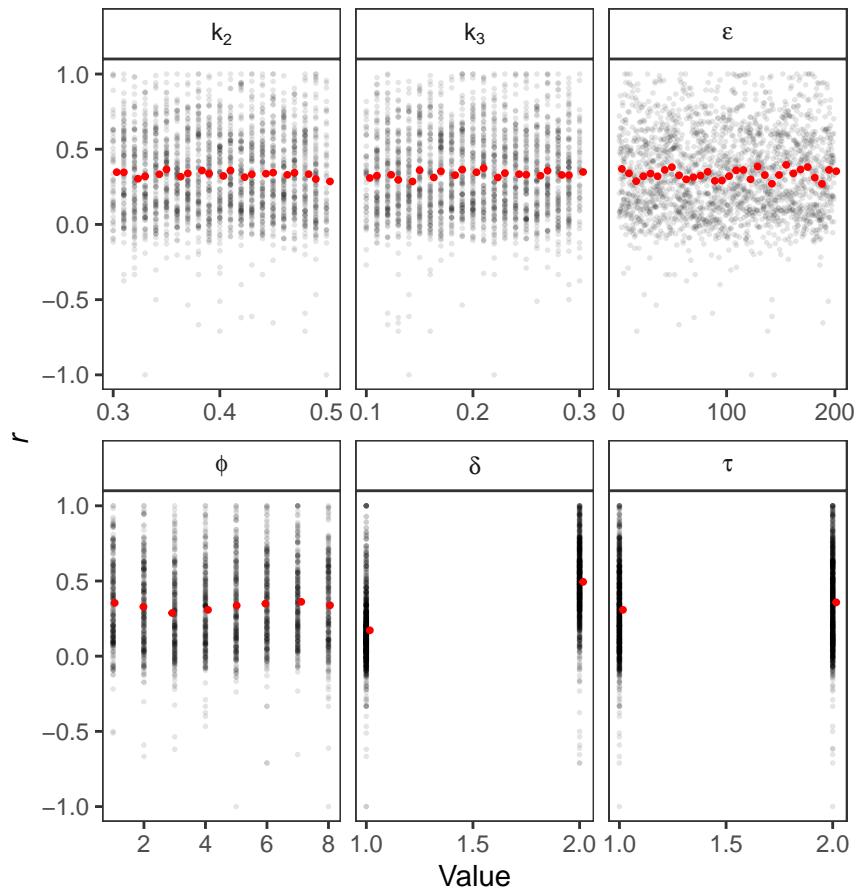
```
##  
## $`Glen and Isaacs`
```

## Glen and Isaacs



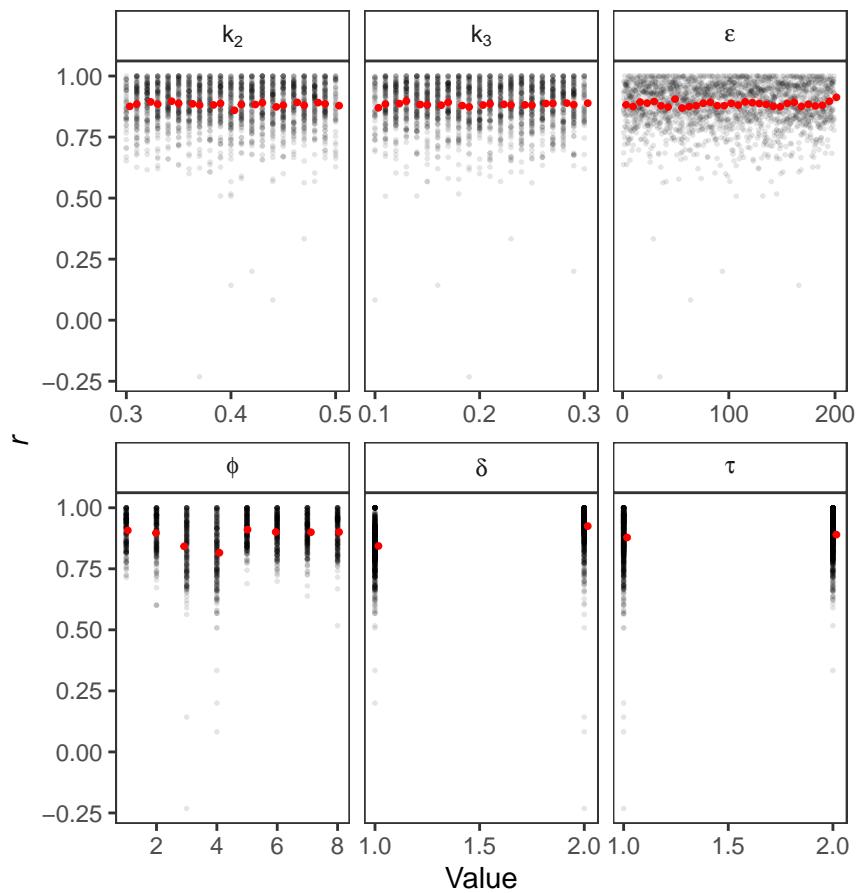
```
##  
## $`Homma and Saltelli`
```

## Homma and Saltelli



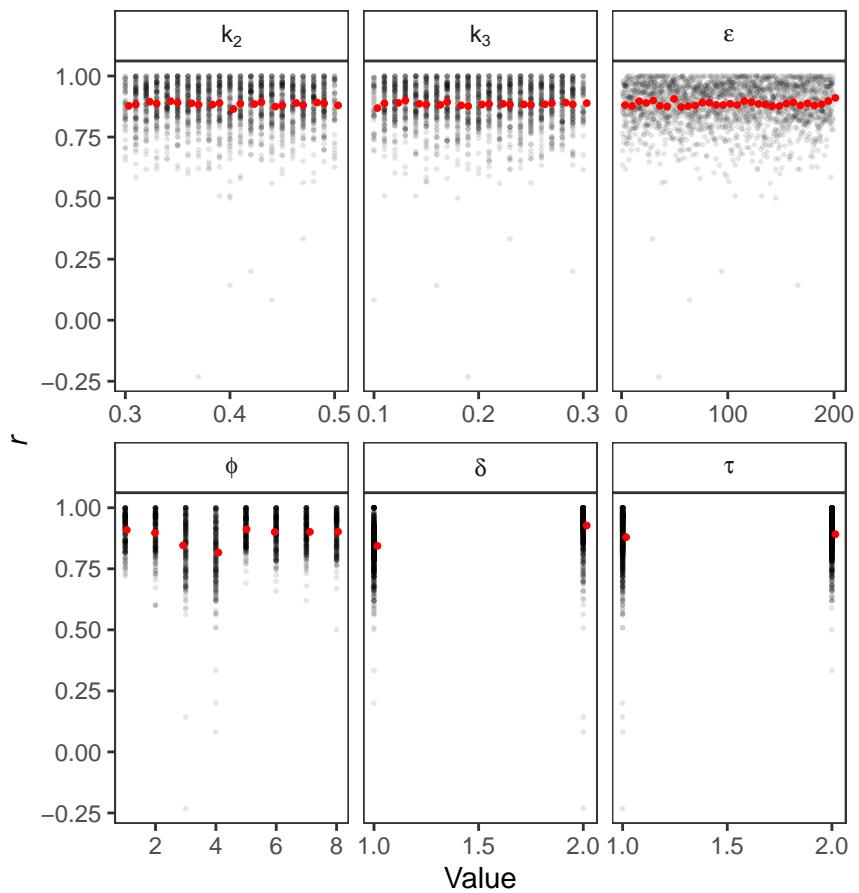
```
##  
## $`Janon/Monod`
```

### Janon/Monod



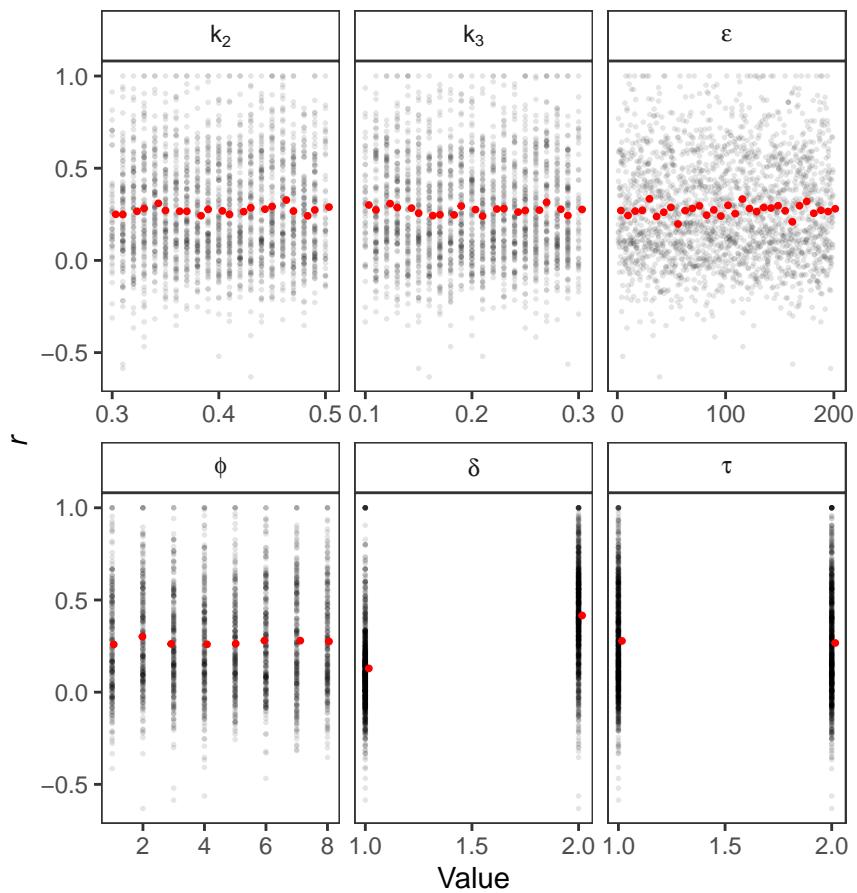
```
##  
## $Jansen
```

Jansen



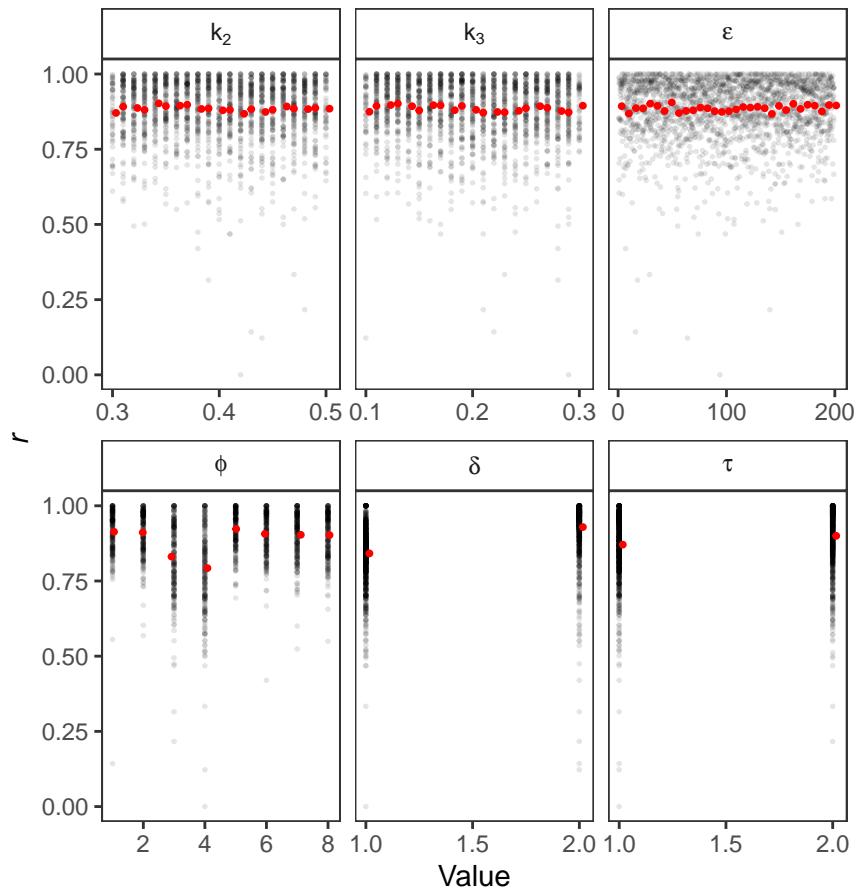
```
##  
## $`pseudo-Owen`
```

### pseudo-Owen



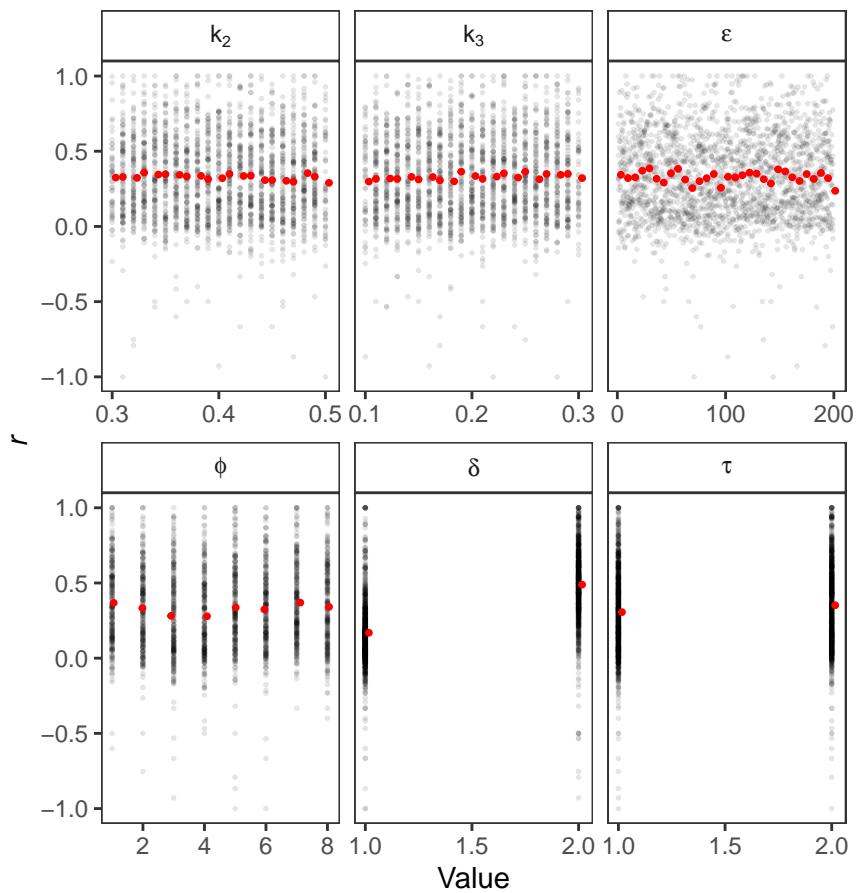
```
##  
## $`Razavi and Gupta`
```

## Razavi and Gupta



```
##  
## $Saltelli
```

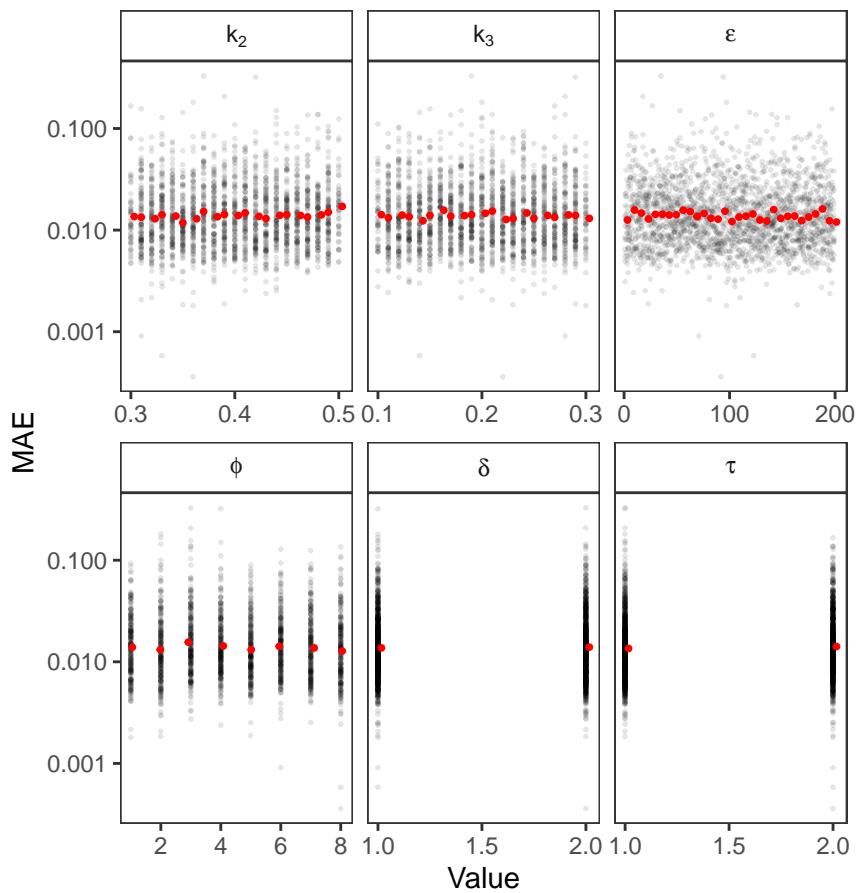
## Saltelli



```
gg.mae
```

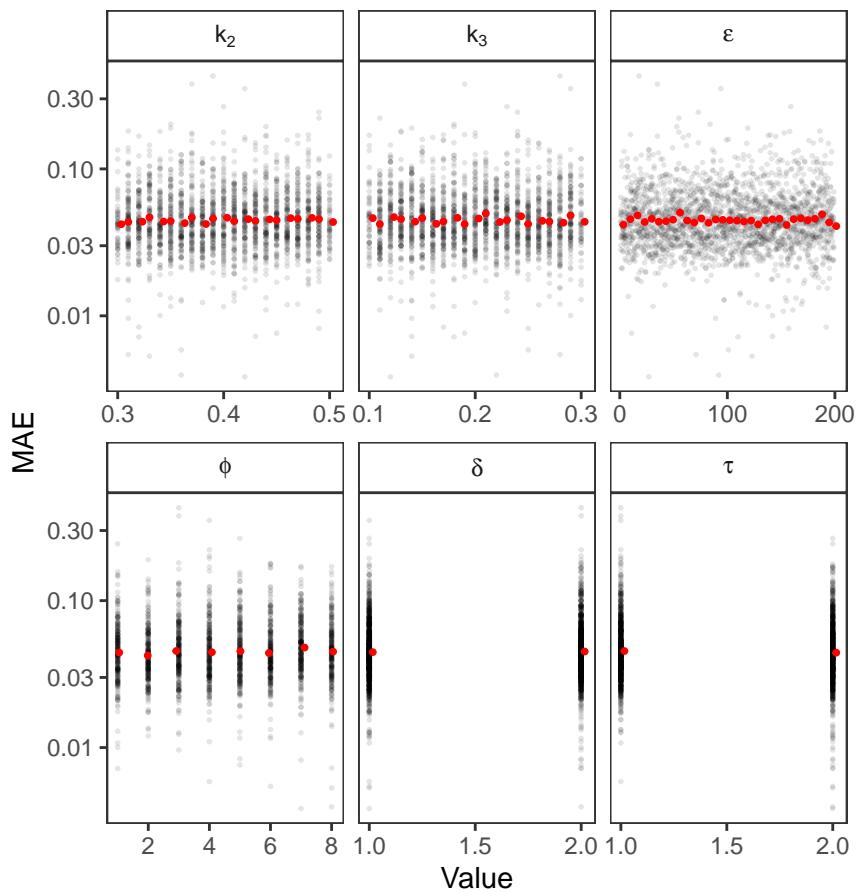
```
## $`Azzini and Rosati`
```

## Azzini and Rosati



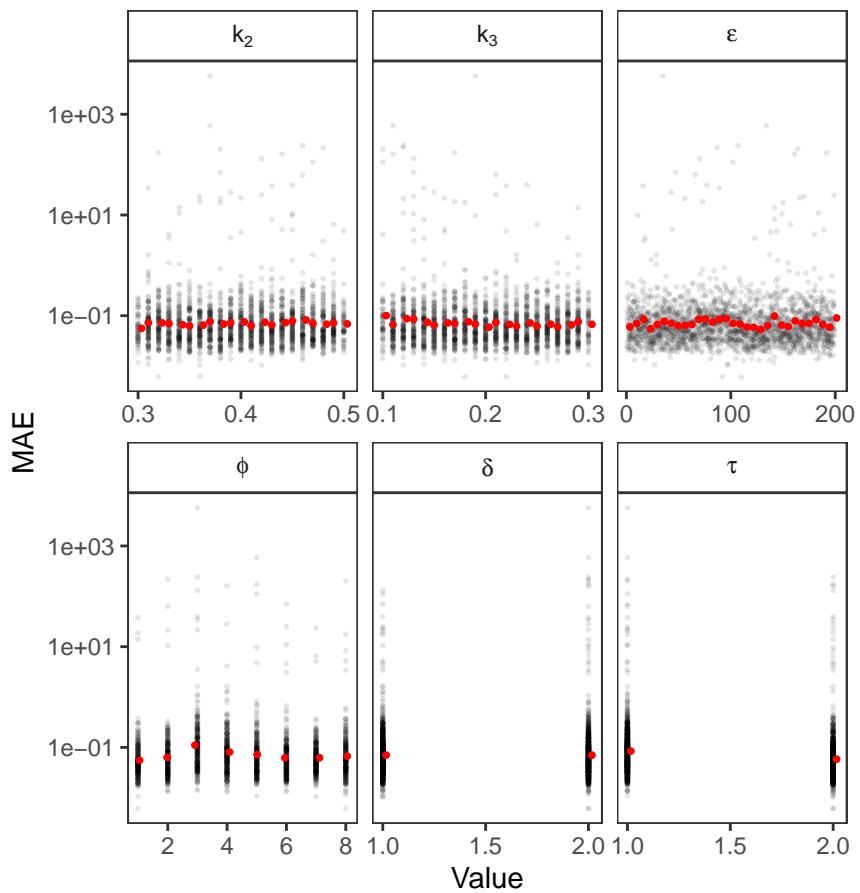
```
##  
## $`Glen and Isaacs`
```

### Glen and Isaacs



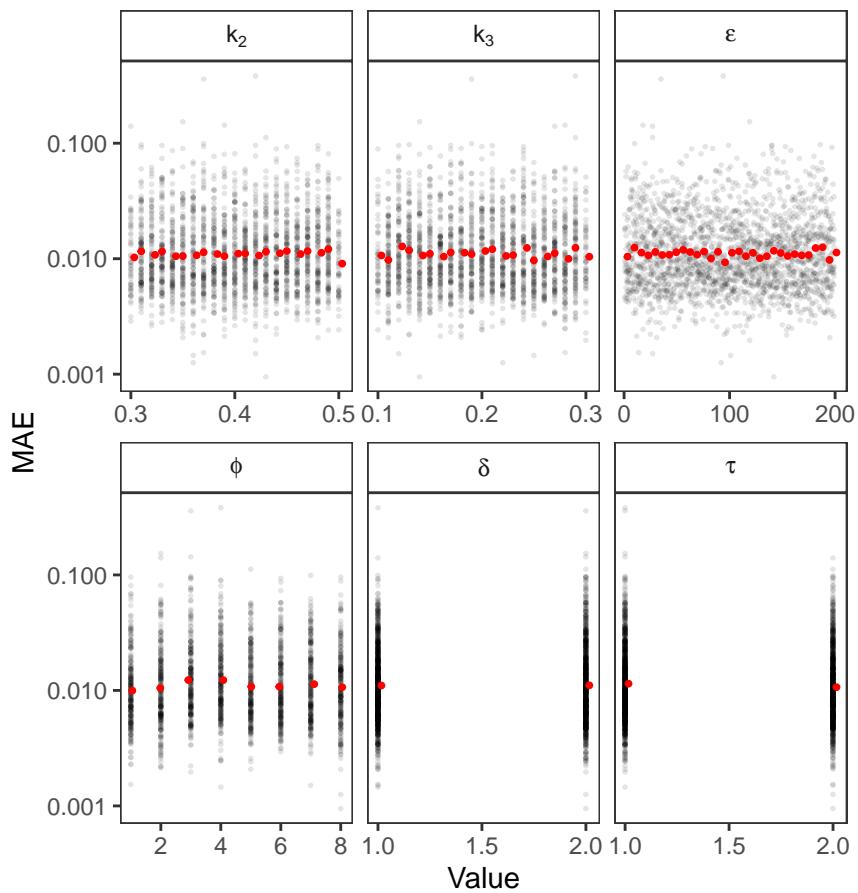
```
##  
## $`Homma and Saltelli`
```

## Homma and Saltelli



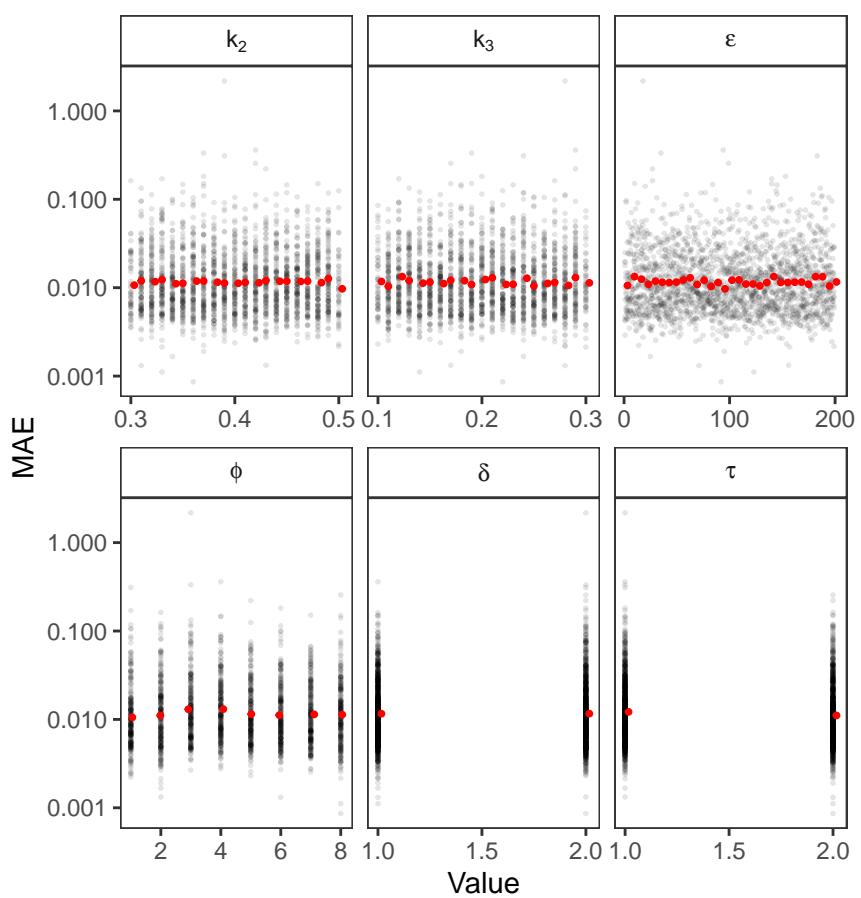
```
##  
## $`Janon/Monod`
```

### Janon/Monod

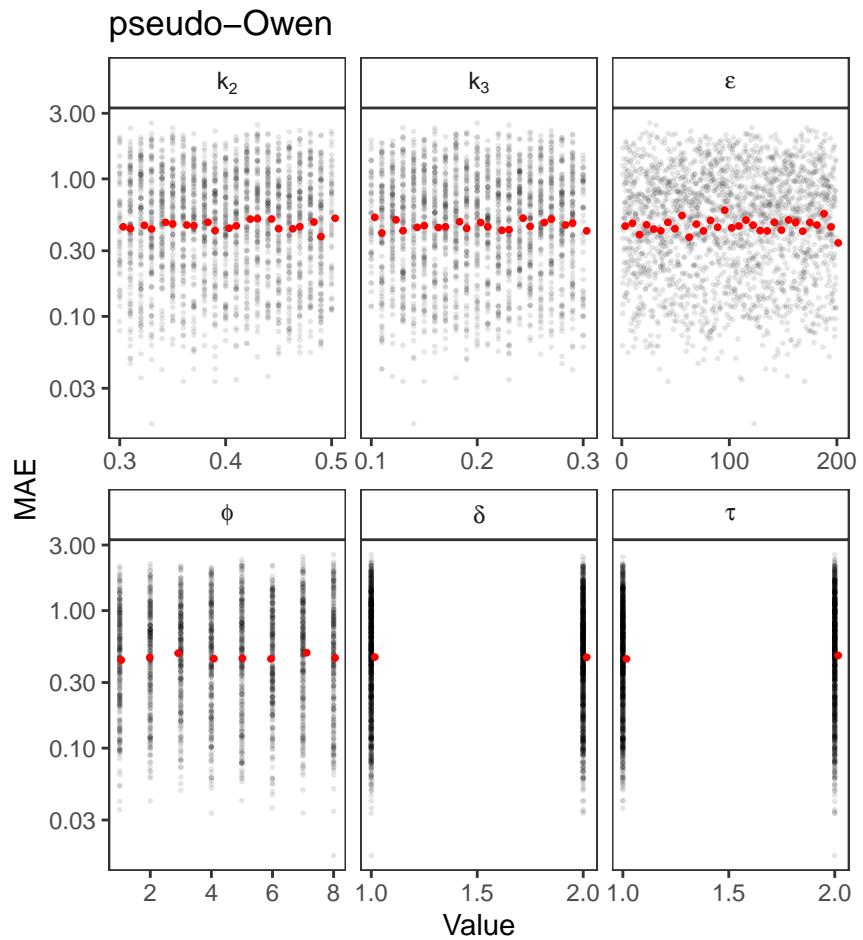


```
##  
## $Jansen
```

Jansen

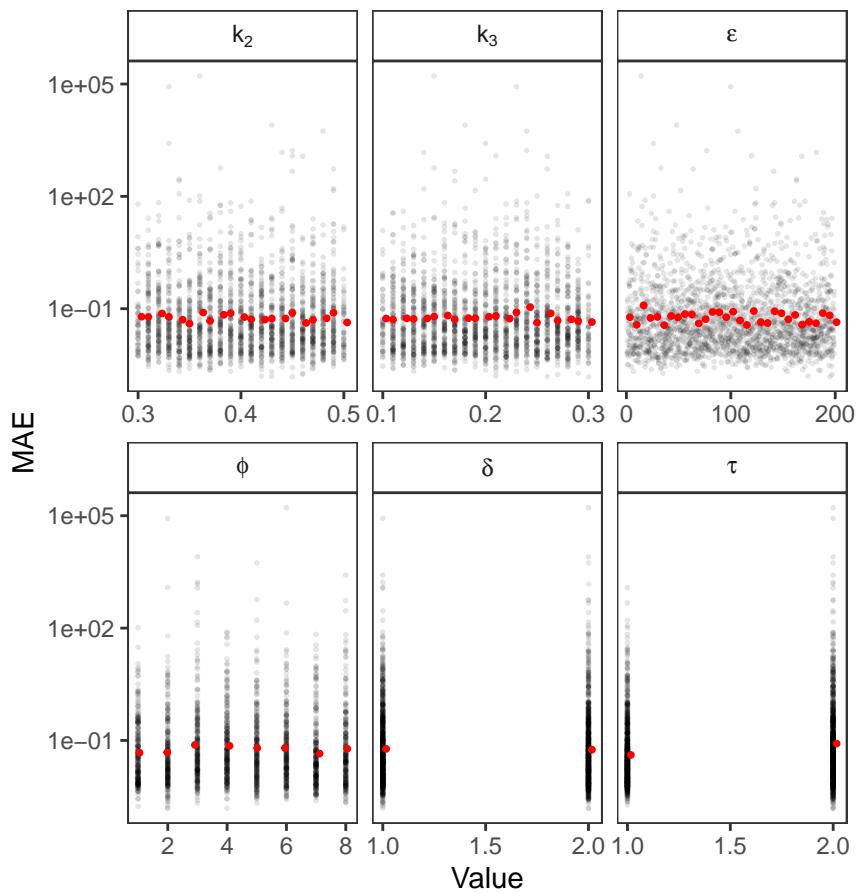


```
##  
## $`pseudo-Owen`
```

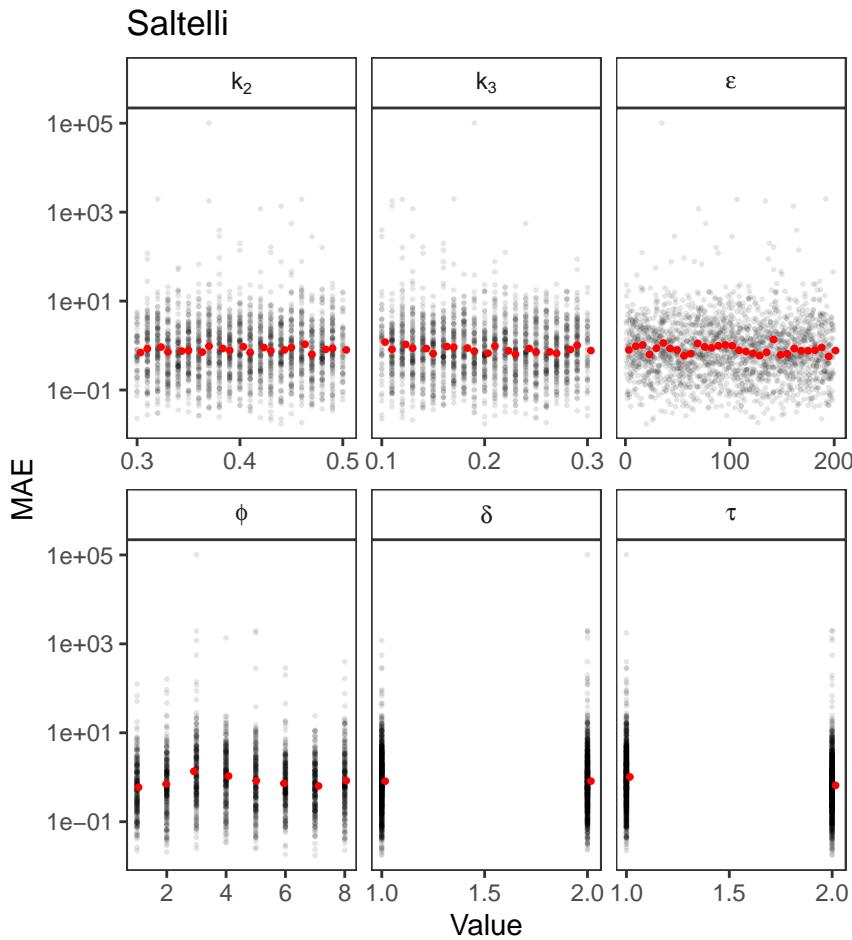


```
##  
## $`Razavi and Gupta`
```

## Razavi and Gupta



```
##  
## $Saltelli
```



## 5.2 Sobol' indices

```
# SOBOL' INDICES ----

out <- list()
for(i in c("r", "MAE")) {
  if(i == "r") {
    params <- c("k[2]", "k[3]", "epsilon", "phi", "delta",
               "tau", "N[t]^k", "f(x)", "delta~tau")
  } else {
    params <- c("k[2]", "k[3]", "epsilon", "phi",
               "tau", "N[t]^k", "f(x)")
  }
  out[[i]] <- full.output[, sobol_indices(Y = value,
                                         N = N,
                                         params = params,
                                         first = "jansen",
                                         boot = TRUE,
                                         R = R,
                                         order = order),
                           estimator]
```

```

}

ind <- rbindlist(out, idcol = "type")
ind <- ind[, group:= ifelse(parameters %in% c("N[t]~k", "f(x)", "delta~tau"),
                           "Cluster", "Individual")] %>%
  .[, type:= factor(type, levels = c("r", "MAE"))]

```

### 5.3 Plot Sobol' indices

```

# PLOT SOBOL' INDICES ----

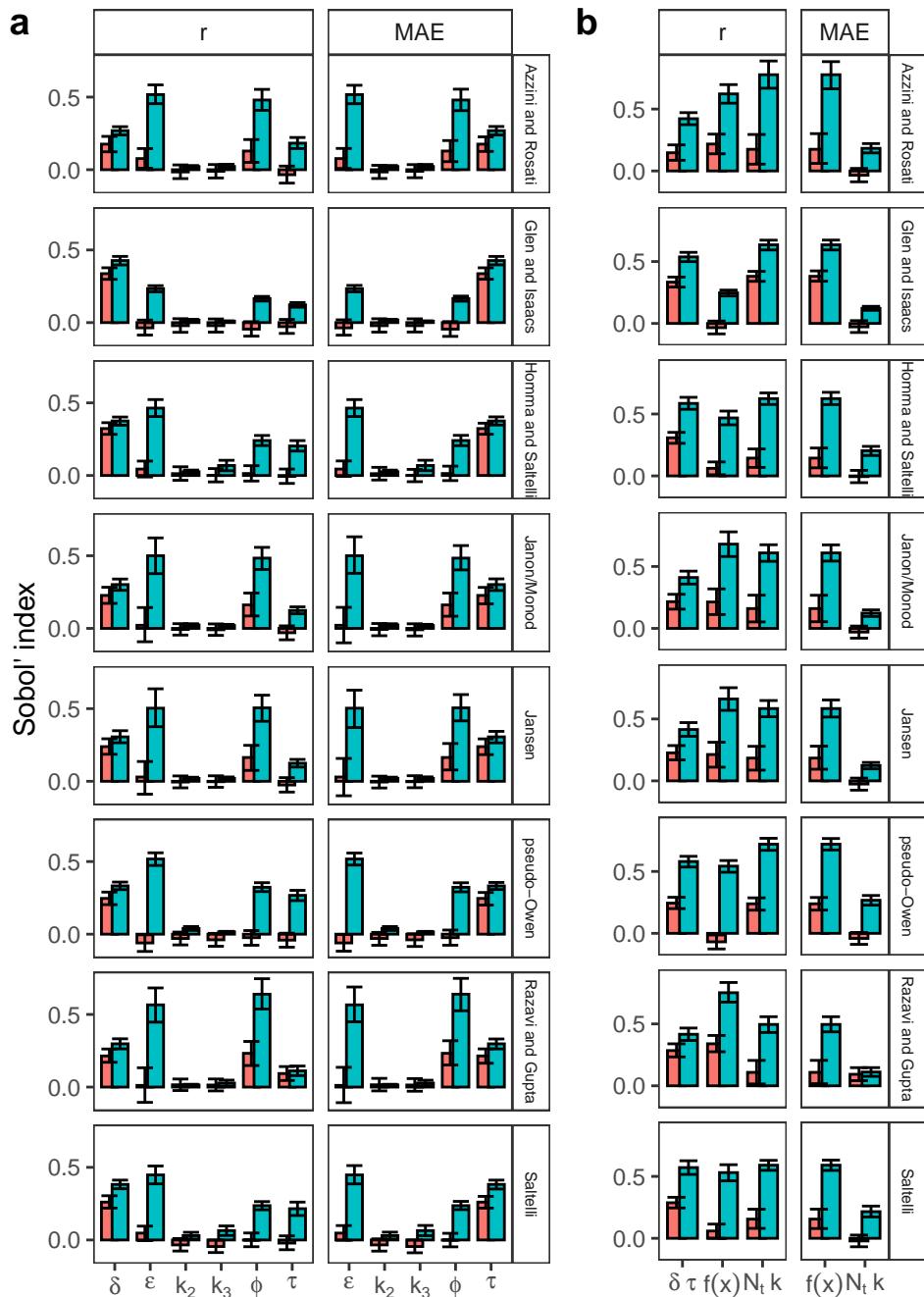
a <- lapply(c("Individual", "Cluster"), function(x)
  plot_sobel(ind[group == x]) +
    facet_grid(estimator~type,
               space = "free_x",
               scales = "free_x") +
    scale_x_discrete(labels = ggplot2:::parse_safe) +
    theme(strip.text.y = element_text(size = 6),
          legend.position = "none"))

# PLOT ----

legend <- get_legend(a[[1]] + theme(legend.position = "top"))
bottom <- plot_grid(a[[1]], a[[2]] + labs(x = "", y = ""), labels = "auto",
                     rel_widths = c(0.6, 0.4))
plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.1, 0.9))

```

Sobol' indices S<sub>i</sub> T<sub>i</sub>



## 6 Session information

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

```
sessionInfo()
```

```

## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.7
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel stats      graphics grDevices utils      datasets  methods
## [8] base
##
## other attached packages:
## [1] checkpoint_0.4.10      ggrepel_0.9.1           sensobol_1.0.2
## [4] logitnorm_0.8.38       benchmarkme_1.0.7      cowplot_1.1.1
## [7] scales_1.1.1           data.table_1.14.0      Rfast_2.0.1
## [10] RcppZiggurat_0.1.6     doParallel_1.0.16      iterators_1.0.13
## [13] foreach_1.5.1         forcats_0.5.1           stringr_1.4.0
## [16] dplyr_1.0.6             purrr_0.3.4            readr_1.4.0
## [19] tidyverse_1.3.1         tibble_3.1.1           ggplot2_3.3.3
## [22] tidyverse_1.3.1         RcppArmadillo_0.10.4.0.0 Rcpp_1.0.6
##
## loaded via a namespace (and not attached):
## [1] lattice_0.20-44        lubridate_1.7.10        assertthat_0.2.1
## [4] digest_0.6.27          utf8_1.2.1              R6_2.5.0
## [7] cellranger_1.1.0       backports_1.2.1        reprex_2.0.0
## [10] evaluate_0.14          httr_1.4.2              pillar_1.6.0
## [13] Rdpack_2.1.1           rlang_0.4.11           readxl_1.3.1
## [16] rstudioapi_0.13        Matrix_1.3-3            rmarkdown_2.8
## [19] munsell_0.5.0          broom_0.7.6             compiler_4.0.3
## [22] modelr_0.1.8           xfun_0.22               pkgconfig_2.0.3
## [25] htmltools_0.5.1.1      tidyselect_1.1.1       codetools_0.2-18
## [28] fansi_0.4.2            crayon_1.4.1            dbplyr_2.1.1
## [31] withr_2.4.2            rbibutils_2.1.1        grid_4.0.3
## [34] jsonlite_1.7.2          gtable_0.3.0            lifecycle_1.0.0
## [37] DBI_1.1.1               magrittr_2.0.1           cli_2.5.0
## [40] stringi_1.5.3           fs_1.5.0                benchmarkmeData_1.0.4
## [43] xml2_1.3.2              ellipsis_0.3.2          generics_0.1.0
## [46] vctrs_0.3.8              tools_4.0.3              glue_1.4.2
## [49] hms_1.0.0                yaml_2.2.1              colorspace_2.0-1
## [52] rvest_1.0.0              knitr_1.33              haven_2.4.1
##
## 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

```

Becker, William. 2020. “Metafunctions for benchmarking in sensitivity analysis.” *Reliability Engineering and System Safety* 204: 107189. <https://doi.org/10.1016/j.ress.2020.107189>.