Discrepancy as a sensitivity measure in mathematical modeling

R code

Arnald Puy

Contents

1 Preliminary functions

```
# PRELIMINARY ----------------------------------------------------------------

# 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)
    }
  }
}

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.margin=margin(0, 0, 0, 0),
          legend.box.margin=margin(-7,-7,-7,-7),
          legend.key = element_rect(fill = "transparent",
                                    color = NA),
          strip.background = element_rect(fill = "white"))
}

# Load the packages
loadPackages(c("sensobol", "data.table", "tidyverse", "parallel",
               "RcppAlgos", "scales", "doParallel", "benchmarkme",
               "cowplot"))
```

2 Goal

We aim at checking whether discrepancy measures can be used as a sensitivity measure to reliably rank parameters in terms of their influence in the model output. To that aim, we explore how well the symmetric L2-discrepancy measure ranks the most important parameters of a set of functions, including for the moment the Oakley and O'Hagan 2004 and the Bratley et al. 1992 functions, and compare its efficacy with that of the Jansen estimator. We do the comparison on Savage-score transformed ranks (Iman and Conover 1987).

The discrepancy is computed directly on the plane of the X_i, Y scatterplots where X_i is a coordinate $(0, 1)$ of the input hypercube and Y is the output rescaled in $0, 1$. For each of the k scatterplots, the points are sorted by one of the X_i. The idea is that a ''pattern" (=effect) corresponds to a deviation from a uniform distribution of points (noise) for a non-influential variable: the largest the effect, the larger the discrepancy, e.g., there are areas in the plot where points are denser and areas where they are more rarefied.

One might wonder why use a discrepancy measure when there are the Sobol' indices. The reply is that many do not understand the theory behind them (and the same applies to moment independent methods or VARS), while everyone understands scatterplots. 'Explaining' discrepancy as a measure of the non-uniformity of the scatterplots appears an avenue likely to appeal to some users.

3 Required functions

3.1 Savage ranks function

```
# SAVAGE SCORES FUNCTION ----------------------------------------------------

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)
}
```

3.2 Discrepancy function

```
# DISCREPANCY FUNCTION ------------------------------------------------------

# This is based on the function discrepancyCriteria_cpp of
# the sensitivity package

# Function to rescale -------------------------
rescale_fun <- function(x) (x - min(x)) / (max(x) - min(x))

# Source cpp code -----------------------------
cpp_functions <- c("cpp_functions.cpp", "L2star_functions.cpp",
```

```
                    "L2_functions.cpp", "L2centered_functions.cpp",
                    "L2wraparound_functions.cpp", "L2modified_functions.cpp")

for(i in 1:length(cpp_functions)) {
  Rcpp::sourceCpp(cpp_functions[i])
}

# Function -------------------------------------
discrepancy_fun <- function (design, type) {

  X <- as.matrix(design)
  dimension <- ncol(X)
  n <- nrow(X)

  # Reescale if needed-------

  if (min(X) < 0 || max(X) > 1) {

    X <- apply(X, 2, rescale_fun)
  }

  # Compute discrepancy

  if (type == "symmetric") {

    P <- 1 + 2 * X - 2 * X^2
    s1 <- DisS2_Rowprod(t(P), dimension)
    s2 <- DisS2_Crossprod(c(t(X)), dimension)
    R <- sqrt(((4/3)^dimension) - ((2/n) * s1) + ((2^dimension/n^2) * s2))

  } else if (type == "star") {

    dL2 <- DisL2star_Crossprod(t(X), dimension)
    R <- sqrt(3^(-dimension) + dL2)

  } else if (type == "L2") {

    P <- X * (1 - X)
    s1 <- DisL2_Rowprod(t(P), dimension)
    s2 <- DisL2_Crossprod(c(t(X)), dimension)
    R <- sqrt(12^(-dimension) - (((2^(1 - dimension))/n) * s1) + ((1/n^2) * s2))

  } else if (type == "centered") {

    P <- 1 + 0.5 * abs(X - 0.5) - 0.5 * (abs(X - 0.5)^2)
    s1 <- DisC2_Rowprod(t(P), dimension)
    s2 <- DisC2_Crossprod(c(t(X)), dimension)
```

```
    R <- sqrt(((13/12)^dimension) - ((2/n) * s1) + ((1/n^2) * s2))

  } else if (type == "wraparound") {

    s1 <- DisW2_Crossprod(t(X), dimension)
    R <- sqrt(-(4/3)^dimension + (1/n^2) * s1)

  } else if (type == "modified") {

    P <- 3 - X^2
    s1 <- DisM2_Rowprod(t(P), dimension)
    s2 <- DisM2_Crossprod(c(t(X)), dimension)
    R <- sqrt(((4/3)^dimension) - (((2^(1 - dimension))/n) * s1) + ((1/n^2) * s2))

  }
  return(R)
}

# Discrepancy function --------------------------
discrepancy <- function(mat, y, params, type) {
  value <- sapply(1:ncol(mat), function(j) {
    design <- cbind(mat[, j], y)
    value <- discrepancy_fun(design = design, type = type)
  })
  return(value)
}
```

3.3 Jansen estimator

```
# FUNCTION TO COMPUTE JANSEN TI ------------------------------------------------

jansen_ti <- function(d, N, params) {
  m <- matrix(d, nrow = N)
  k <- length(params)
  Y_A <- m[, 1]
  Y_AB <- m[, -1]
  f0 <- (1 / length(Y_A)) * sum(Y_A)
  VY <- 1 / length(Y_A) * sum((Y_A - f0) ^ 2)
  value <- (1 / (2 * N) * Rfast::colsums((Y_A - Y_AB) ^ 2)) / VY
  return(value)
}
```

3.4 Function to replicate sample matrix

```
# FUNCTION TO CREATE REPLICAS OF SAMPLE MATRIX ---------------------------------

# For discrepancy
```

```r
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)
}

sobol_replicas <- function(matrices, N, params, replicas,...) {
  mat <- sobol_matrices(matrices = matrices, N = N * replicas,
                        params = params,...)
  index <- CutBySize(nrow(mat), block.size = N)
  out <- list()
  for (i in 1:nrow(index)) {
    out[[i]] <- mat[index[i, "lower"]:index[i, "upper"], ]
  }
  return(out)
}

# For Sobol' indices
scrambled_sobol <- function(A, B) {
  X <- rbind(A, B)
  for(i in 1:ncol(A)) {
    AB <- A
    AB[, i] <- B[, i]
    X <- rbind(X, AB)
  }
  AB <- rbind(A, X[((2*nrow(A)) + 1):nrow(X), ])
  return(AB)
}
# Function to create replicas of the A, B and AB matrices
sobol_replicas_A_AB <- function(N, params, replicas) {
  k <- length(params)
  df <- randtoolbox::sobol(n = N * replicas, dim = k * 2)
  indices <- CutBySize(nrow(df), nb = replicas)
  X <- A <- B <- out <- list()
  for(i in 1:nrow(indices)) {
    lower <- indices[i, "lower"]
    upper <- indices[i, "upper"]
    X[[i]] <- df[lower:upper, ]
  }
  for(i in seq_along(X)) {
    A[[i]] <- X[[i]][, 1:k]
    B[[i]] <- X[[i]][, (k + 1) : (k * 2)]
  }
  for(i in seq_along(A)) {
    out[[i]] <- scrambled_sobol(A[[i]], B[[i]])
```

```
  }
  return(out)
}
```

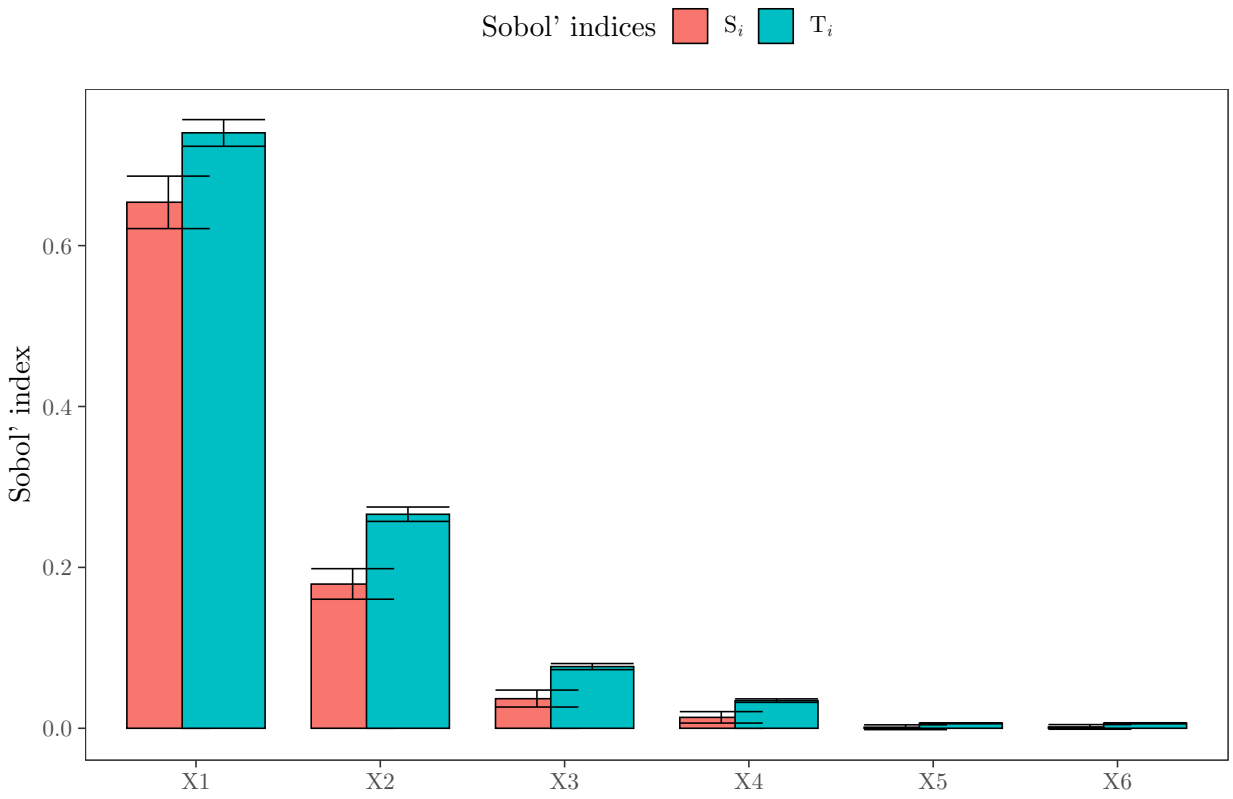
3.5 Function to check correlation between discrepancy and jansen

```
# FUNCTION TO CHECK CORRELAITON BETWEEN DISCREPANCIES AND SAVAGE SCORES --------

cor_fun <- function(N = N, params = params, replicas = replicas,
                    type = "symmetric", approach = NULL,
                    model_fun = NULL, true_scores = NULL,...) {

  if (is.null(approach)) {
    stop("approach should be either discrepancy or sobol")
  }

  if (is.null(model_fun) | is.null(true_scores)) {
    stop("A function and the true ranks should be provided")
  }

  if (approach == "discrepancy") {

    mat <- sobol_replicas(matrices = "A", N = N, params = params,
                          replicas = replicas,...)
    y <- lapply(mat, model_fun)

    out <- list()
    for(i in 1:length(y)) {
      out[[i]] <- discrepancy(mat = mat[[i]], y = y[[i]], params = params,
                              type = type) %>%
        savage_scores()
    }

  } else if (approach == "sobol") {

    mat <- sobol_replicas_A_AB(N = N, params = params, replicas = replicas)
    y <- lapply(mat, model_fun)
    out <- lapply(y, function(x) jansen_ti(d = x, N = N, params = params) %>%
                    savage_scores())
  }

  final <- unlist(lapply(out, function(x) cor(x, true_scores)))

  return(final)
}
```

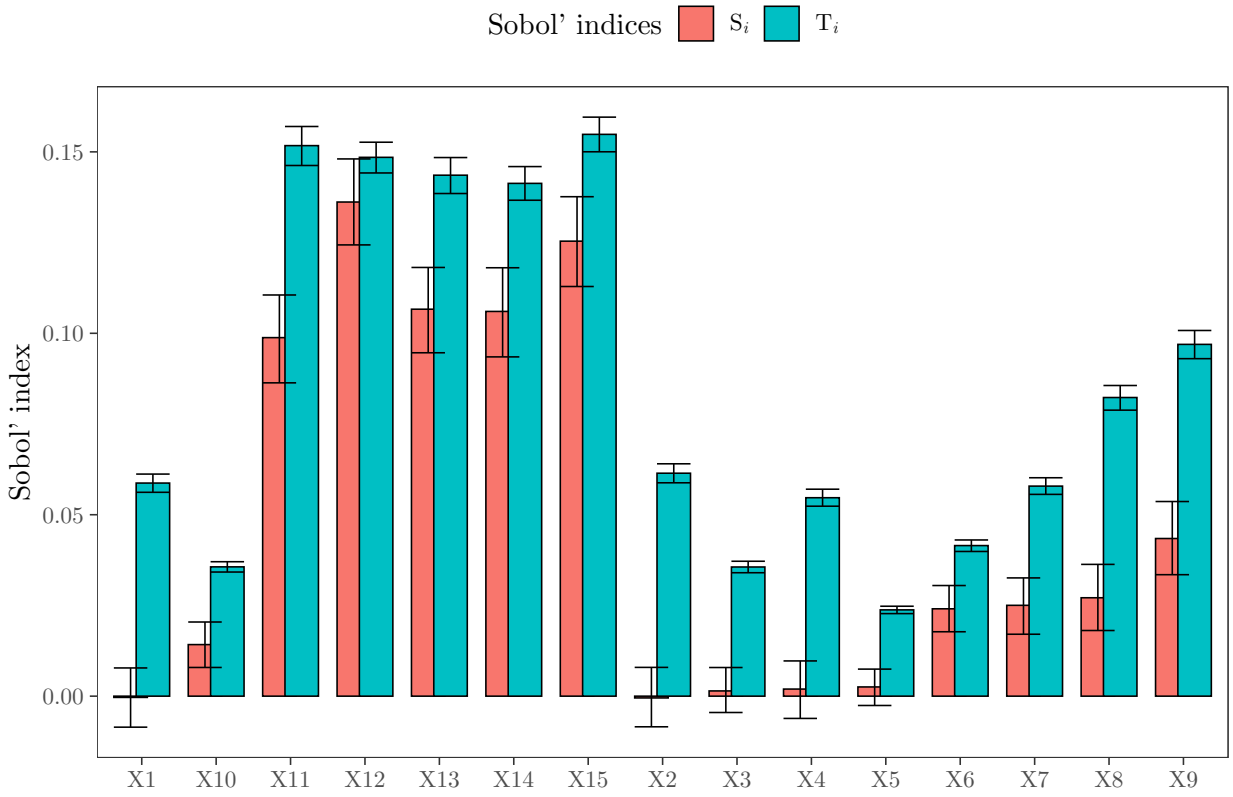
4 The Bratley et al. 1992 and the Oakland and O'Hagan 2004 functions

```
# CHECK SOBOL' INDICES AND SCATTERPLOT FOR SEVERAL FUNCTIONS ------------------

N <- 2^14 # Base sample size
matrices <- c("A", "B", "AB")
R <- 10^3 # Bootstrap replicas

models <- c("Bratley et al. 1992", "Oakland and O'Hagan 2004")
fun_list <- list(bratley1992_Fun, oakley_Fun)
names(fun_list) <- models

out <- ind <- savage.rank <- list()
for (i in models) {

  if (i == "Bratley et al. 1992") {
    params <- paste("X", 1:6, sep = "")

  } else if (i == "Oakland and O'Hagan 2004") {
    params <- paste("X", 1:15, sep = "")
  }

  mat <- sobol_matrices(matrices = matrices, N = N, params = params)
  y <- fun_list[[i]](mat)
  ind[[i]] <- sobol_indices(matrices = matrices, Y = y, N = N, params = params,
                        boot = TRUE, R = R)
  savage.rank[[i]] <- ind[[i]]$results[sensitivity == "Ti"] %>%
    .[, savage_scores(original)]
  out[[i]] <-plot_scatter(data = mat, N = N, Y = y, params = params)
}

lapply(ind, plot)
```

```
## $`Bratley et al. 1992`
```




```
##
## $`Oakland and O'Hagan 2004`
```



5 Run the model

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

matrices <- "A"
replicas <- 150 # Number of replicas of the sample matrix

output <-  list()

for(i in models) {

  if (i == "Bratley et al. 1992") {
    params <- paste("X", 1:6, sep = "")

  } else if (i == "Oakland and O'Hagan 2004") {
    params <- paste("X", 1:15, sep = "")

  }

  for (j in c("discrepancy", "sobol")) {

    if (j == "discrepancy") {

      if(i == "Bratley et al. 1992") {
        sample.sizes <- seq(4, 150, by = 4)
        final.sample.sizes <- sample.sizes
      } else {
        sample.sizes <- seq(4, 320, by = 4)
        final.sample.sizes <- sample.sizes
      }

    } else if (j == "sobol") {
      sample.sizes <- 2:20
      final.sample.sizes <- sample.sizes * (length(params) + 1)

    }
    output[[i]][[j]] <- mclapply(sample.sizes, function(x)
      cor_fun(N = x, params = params, replicas = replicas, model_fun = fun_list[[i]],
              true_scores = savage.rank[[i]], approach = j, type = "symmetric",
              scrambling = 1),
      mc.cores = detectCores() * 0.75)

    names(output[[i]][[j]]) <- final.sample.sizes

  }
}
```

```r
# Arrange output -------------------
results <- lapply(output, function(x) lapply(x, function(y)
  lapply(y, function(z) data.table(z))))  %>%
  lapply(., function(x) lapply(x, function(y) rbindlist(y, idcol = "Model.runs"))) %>%
  lapply(., function(x) rbindlist(x, idcol = "Approach")) %>%
  rbindlist(., idcol = "Function") %>%
  .[, Model.runs:= as.numeric(Model.runs)] %>%
  .[, Approach:= ifelse(Approach == "discrepancy", "Discrepancy", "Jansen estimator")] %>%
  na.omit()
```

```r
# Plot the results ------------------------------------------------------------

# New facet label names for supp variable
supp.labs <- c("Symmetric $L2$ discrepancy", "Jansen estimator")
names(supp.labs) <- c("Discrepancy", "Jansen estimator")

ggplot(results, aes(Model.runs, z, group = Model.runs)) +
  geom_boxplot(position = "identity", outlier.size = 0.5) +
  labs(x = "Nº of model runs", y = "Correlation") +
  facet_grid(Approach~Function,
             scale = "free_x", space = "free",
             labeller = labeller(Approach = supp.labs)) +
  theme_AP()
```

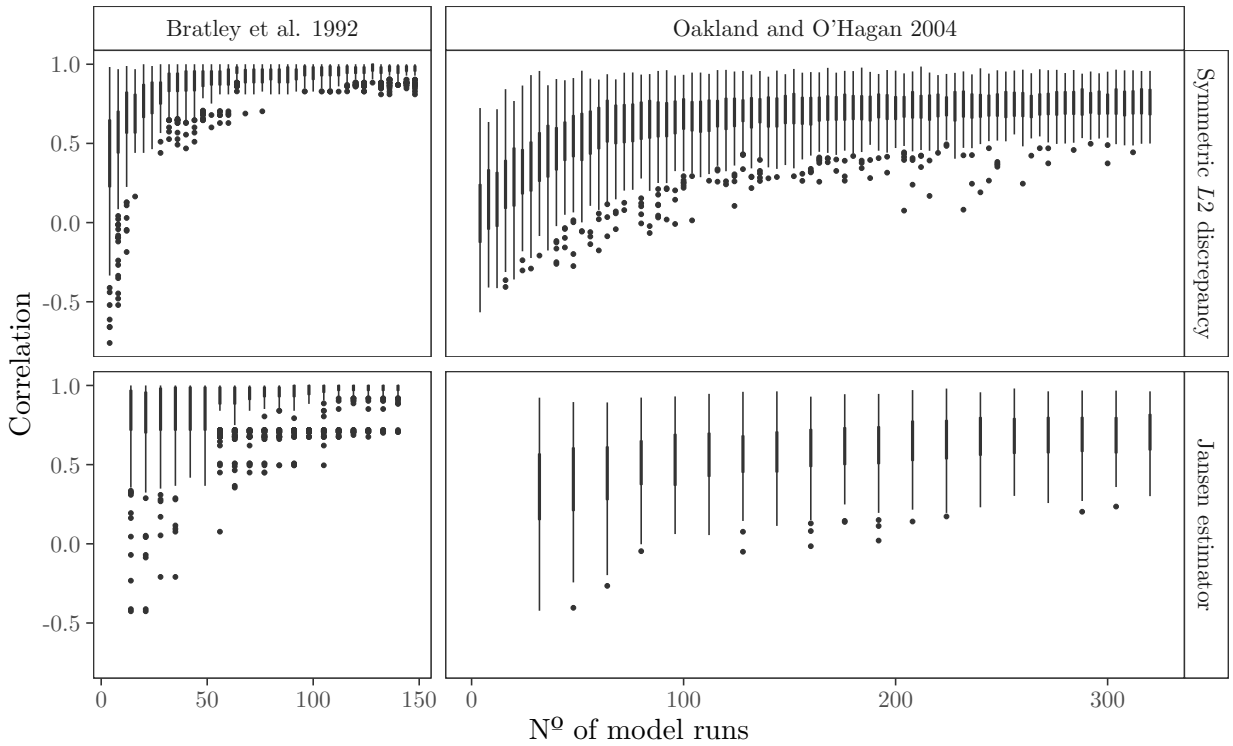


Figure 1: Correlation between the Savage scores obtained with T_i (Jansen estimator) and those obtained with discrepancy.

6 The metamodel

```r
# RANDOM FUNCTIONS AND DISTRIBUTIONS ----------------------------------------

# Functions -----------------------------------
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),
  Piecewise.large = function(x) ((-1) ^ as.integer(4 * x) *
                                   (0.125 - (x %% 0.25)) + 0.125),
  Piecewise.small = function(x) ((-1) ^ as.integer(32 * x) *
                                   (0.03125 - 2 * (x %% 0.03125)) + 0.03125) / 2,
  Oscillation = function(x) x ^ 2 - 0.2 * cos(7 * pi * x)
)

# Random distributions  -------------------------
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) logitnorm::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)
}
```

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

model_fun <- function(tau, epsilon, base.sample.size, cost.discrepancy, phi, k) {

  params <- paste("X", 1:k, sep = "")

  if (tau == 1) {

    type <- "R"

  } else if (tau == 2) {

    type <- "QRN"
  }
  set.seed(epsilon)
  discrepancy.mat <- sobol_matrices(matrices = "A", N = cost.discrepancy,
                                    params = params,
                                    type = type)

  set.seed(epsilon)
  jansen.mat <- sobol_matrices(matrices = c("A", "AB"), N = base.sample.size,
                               params = params,
                               type = type)

  all.matrices <- rbind(discrepancy.mat, jansen.mat)
  set.seed(epsilon)
  transformed.all.matrices <- random_distributions(X = all.matrices, phi = phi)

  y <- sensobol::metafunction(data = transformed.all.matrices, epsilon = epsilon)

  type <- c("symmetric", "star", "L2", "centered", "wraparound", "modified")

  discrepancy.value <- lapply(type, function(x)
    discrepancy(mat = all.matrices[1:cost.discrepancy, ],
                y = y[1:cost.discrepancy], params = params, type = x))


  jansen.value <- jansen_ti(d = y[(cost.discrepancy + 1):length(y)],
                            N = base.sample.size, params = params)

  savage.discrepancy <- lapply(discrepancy.value, savage_scores)
  jansen.discrepancy <- savage_scores(jansen.value)

  out <- lapply(savage.discrepancy, function(x)
    cor(x, jansen.discrepancy))

  return(out)
```

```
}
```

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

N <- 2^10
params <- c("epsilon", "phi", "k", "tau", "base.sample.size")
mat <- sobol_matrices(matrices = "A", N = N, params = params)

# Define distributions ---------------------------

mat[, "epsilon"] <- floor(qunif(mat[, "epsilon"], 1, 200))
mat[, "phi"] <- floor(mat[, "phi"] * 8) + 1
mat[, "k"] <- floor(qunif(mat[, "k"], 3, 50))
mat[, "tau"] <- floor(mat[, "tau"] * 2) + 1
mat[, "base.sample.size"] <- floor(qunif(mat[, "base.sample.size"], 10, 100))

cost.jansen <- mat[, "base.sample.size"] * (mat[, "k"] + 1)
cost.discrepancy <- cost.jansen

final.mat <- cbind(mat, cost.jansen, cost.discrepancy)
```

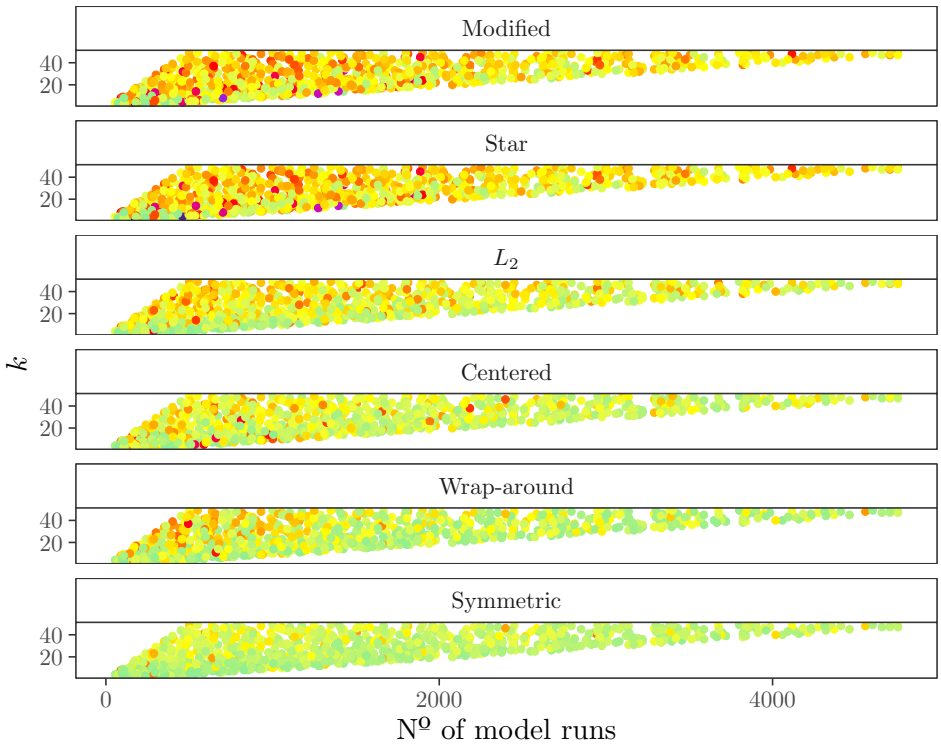
```
# RUN MODEL  -------------------------------------------------------------

y <- mclapply(1:nrow(final.mat), function(i) {
  model_fun(tau = final.mat[i, "tau"],
            epsilon = final.mat[i, "epsilon"],
            base.sample.size = final.mat[i, "base.sample.size"],
            cost.discrepancy = final.mat[i, "cost.discrepancy"],
            phi = final.mat[i, "phi"],
            k = final.mat[i, "k"])},
  mc.cores = floor(detectCores() * 0.75))
```

```
# ARRANGE DATA -----------------------------------------------------------

y <- lapply(y, unlist)
output <- data.table(do.call(rbind, y))
discrepancy_methods <- c("symmetric", "star", "L2", "centered",
                         "wraparound", "modified")
colnames(output) <- discrepancy_methods

final.output <- data.table(cbind(final.mat, output))

# Write model output
fwrite(final.output, "final.output.csv")
```

```
# PLOT UNCERTAINTY -------------------------------------------------------

# New facet label names for supp variable
supp.labs <- c("Symmetric", "Star", "$L_2$", "Centered", "Wrap-around", "Modified")
```

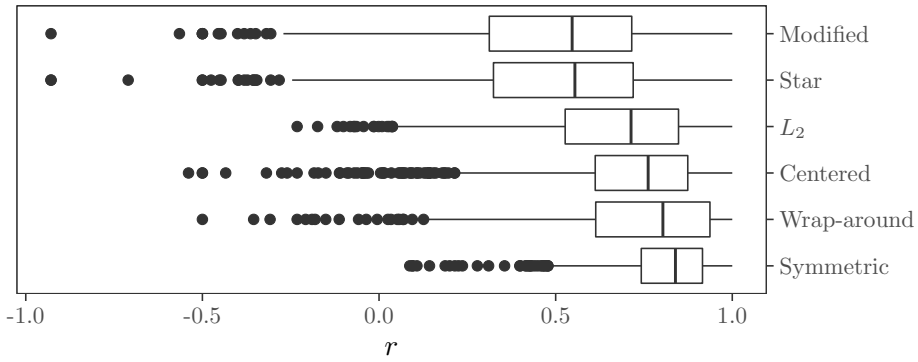
```
names(supp.labs) <- discrepancy_methods

a <- melt(final.output, measure.vars = discrepancy_methods) %>%
  .[, variable:= factor(variable, levels = c("modified", "star", "L2",
                                             "centered", "wraparound",
                                             "symmetric"))] %>%
  ggplot(., aes(cost.discrepancy, k, color = value)) +
  geom_point(size = 1) +
  scale_colour_gradientn(colours = c("black", "purple", "red", "orange",
                                     "yellow", "lightgreen"),
                         name = expression(italic(r)),
                         breaks = pretty_breaks(n = 3)) +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = "Nº of model runs", y = "$k$") +
  facet_wrap(~variable, ncol = 1, labeller = labeller(variable = supp.labs)) +
  theme_AP() +
  theme(legend.position = "none",
        strip.background = element_rect(fill = "white"))

b <- melt(final.output, measure.vars = discrepancy_methods) %>%
  ggplot(., aes(reorder(variable, -value), value)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "", y = "$r$") +
  scale_x_discrete(position = "top",
                   labels = supp.labs) +
  theme_AP()
```

```
# PLOT SCATTERPLOT ------------------------------------------------------------

a
```




```
# PLOT BOXPLOTS ----------------------------------------------------------------
```

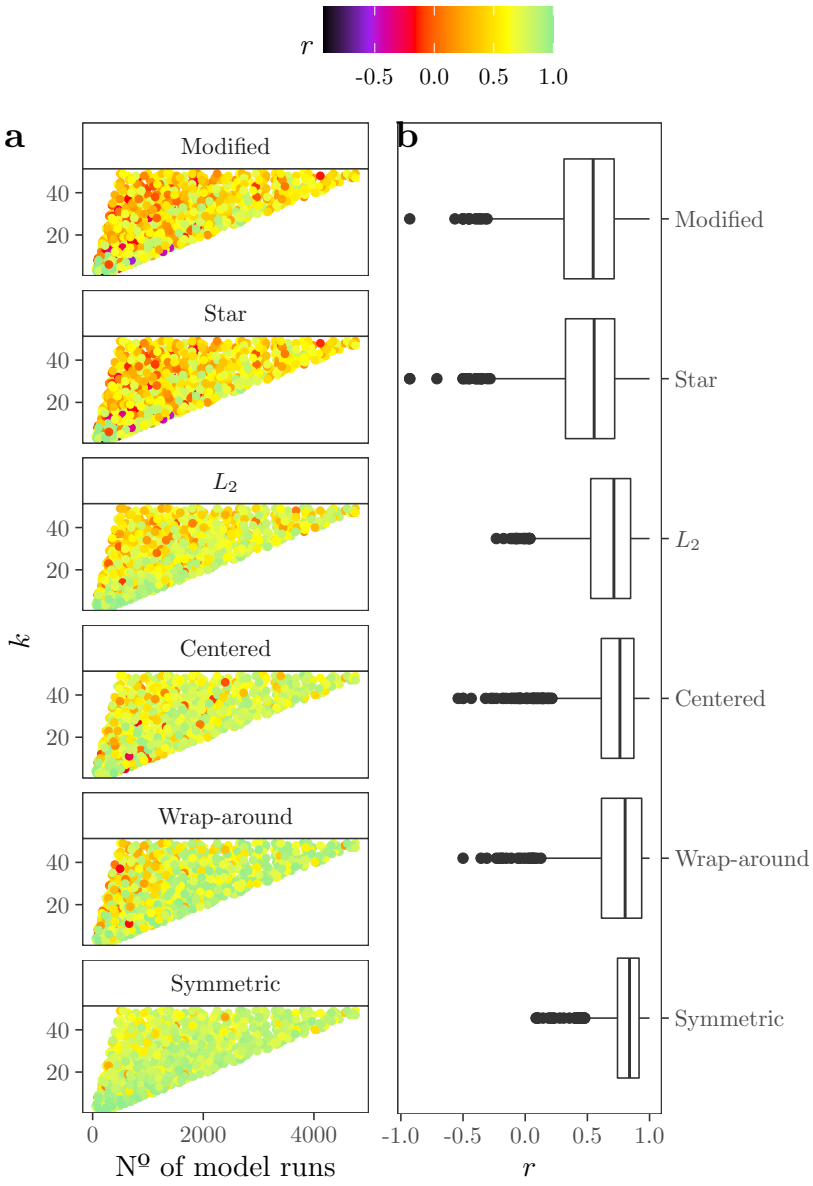
b




```
# MERGE PLOTS ------------------------------------------------------------------
legend <- get_legend(a + theme(legend.position = "top",
                               legend.margin=margin(0, 0, 0, 0),
                               legend.box.margin=margin(-7,-7,-7,-7)))
bottom <- plot_grid(a, b, ncol = 2, labels = "auto",
                    rel_widths = c(0.45, 0.55))

plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.1, 0.9),
          rel_widths = c(0.25, 0.75))
```

```
## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
```

```
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.

## Warning in (function (texString, cex = 1, face = 1, engine =
## getOption("tikzDefaultEngine"), : Attempting to calculate the width of a Unicode
## stringusing the pdftex engine. This may fail! See the Unicodesection of ?
## tikzDevice for more information.
```




```
# SOME STATISTICS ------------------------------------------------------------

final.stat <- melt(final.output, measure.vars = discrepancy_methods)

final.stat[, .(mean = mean(value), median = median(value), sd = sd(value),
        max = max(value), min = min(value)), variable]
```

```
##      variable      mean    median        sd max         min
## 1:  symmetric 0.8115546 0.8391197 0.1421400   1  0.08695652
## 2:       star 0.5000740 0.5545182 0.3018096   1 -0.92857143
## 3:         L2 0.6652102 0.7133971 0.2334243   1 -0.23188406
## 4:   centered 0.7028626 0.7618533 0.2420824   1 -0.53925540
## 5: wraparound 0.7400277 0.8036229 0.2385884   1 -0.50000000
## 6:   modified 0.4916879 0.5466692 0.3005648   1 -0.92857143
```

```
final.stat[, .(quantile = quantile(value)), variable]
```

```
##       variable    quantile
##  1:  symmetric  0.08695652
##  2:  symmetric  0.74236643
##  3:  symmetric  0.83911972
##  4:  symmetric  0.91564479
##  5:  symmetric  1.00000000
##  6:       star -0.92857143
##  7:       star  0.32392821
##  8:       star  0.55451819
##  9:       star  0.71948894
## 10:       star  1.00000000
## 11:         L2 -0.23188406
## 12:         L2  0.52693571
## 13:         L2  0.71339707
## 14:         L2  0.84781675
## 15:         L2  1.00000000
## 16:   centered -0.53925540
## 17:   centered  0.61202010
## 18:   centered  0.76185327
## 19:   centered  0.87396248
## 20:   centered  1.00000000
## 21: wraparound -0.50000000
## 22: wraparound  0.61346360
## 23: wraparound  0.80362291
## 24: wraparound  0.93680110
## 25: wraparound  1.00000000
## 26:   modified -0.92857143
## 27:   modified  0.31214845
## 28:   modified  0.54666924
## 29:   modified  0.71532890
## 30:   modified  1.00000000
##       variable    quantile
```

7 Session information

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

sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.3.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1-arm64/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] cowplot_1.1.1     benchmarkme_1.0.7 doParallel_1.0.17 iterators_1.0.14
##  [5] foreach_1.5.2     scales_1.1.1      RcppAlgos_2.5.3   forcats_0.5.1
##  [9] stringr_1.4.0     dplyr_1.0.8       purrr_0.3.4       readr_2.1.2
## [13] tidyr_1.2.0       tibble_3.1.6      ggplot2_3.3.5     tidyverse_1.3.1
## [17] data.table_1.14.2 sensobol_1.1.0
##
## loaded via a namespace (and not attached):
##  [1] matrixStats_0.61.0    fs_1.5.2              lubridate_1.8.0
##  [4] httr_1.4.2            tools_4.1.2           backports_1.4.1
##  [7] utf8_1.2.2            R6_2.5.1              DBI_1.1.2
## [10] colorspace_2.0-3      withr_2.5.0           tidyselect_1.1.2
## [13] compiler_4.1.2        cli_3.2.0             rvest_1.0.2
## [16] xml2_1.3.3            labeling_0.4.2        digest_0.6.29
## [19] rmarkdown_2.13        benchmarkmeData_1.0.4 pkgconfig_2.0.3
## [22] htmltools_0.5.2       dbplyr_2.1.1          fastmap_1.1.0
## [25] highr_0.9             rlang_1.0.2           readxl_1.4.0
## [28] rstudioapi_0.13       generics_0.1.2        farver_2.1.0
## [31] tikzDevice_0.12.3.1   jsonlite_1.8.0        magrittr_2.0.3
## [34] Matrix_1.4-1          Rcpp_1.0.8.3          munsell_0.5.0
## [37] fansi_1.0.3           lifecycle_1.0.1       RcppZiggurat_0.1.6
## [40] stringi_1.7.6         yaml_2.3.5            grid_4.1.2
## [43] crayon_1.5.1          lattice_0.20-45       haven_2.4.3
## [46] hms_1.1.1             knitr_1.38            pillar_1.7.0
## [49] boot_1.3-28           randtoolbox_1.31.1    codetools_0.2-18
## [52] reprex_2.0.1          glue_1.6.2            evaluate_0.15
## [55] modelr_0.1.8          png_0.1-7             vctrs_0.4.0
```

```
## [58] tzdb_0.3.0            Rdpack_2.3            cellranger_1.1.0
## [61] gtable_0.3.0          assertthat_0.2.1      xfun_0.30
## [64] rbibutils_2.2.7       Rfast_2.0.6           broom_0.7.12
## [67] rngWELL_0.10-7        filehash_2.4-3        tinytex_0.38
## [70] gmp_0.6-5             ellipsis_0.3.2
```

```
## Return the machine CPU
cat("Machine:     "); print(get_cpu()$model_name)
```

```
## Machine:
```

```
## [1] "Apple M1 Max"
```

```
## Return number of true cores
cat("Num cores:   "); print(detectCores(logical = FALSE))
```

```
## Num cores:
```

```
## [1] 10
```

```
## Return number of threads
cat("Num threads: "); print(detectCores(logical = TRUE))
```

```
## Num threads:
```

```
## [1] 10
```

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

```
## RAM:
```

```
## 68.7 GB
```

References

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