Discrepancy measures for sensitivity analysis

R code

Arnald Puy

Contents

1 Preliminary

```
# PRELIMINARY ###########################################################################

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(-5,-5,-5,-5),
          legend.key = element_rect(fill = "transparent",
                                    color = NA),
          strip.background = element_rect(fill = "white"))
}

# Load the packages
sensobol::load_packages(c("sensobol", "data.table", "tidyverse", "parallel",
                          "RcppAlgos", "scales", "doParallel", "benchmarkme",
                          "cowplot", "wesanderson", "microbenchmark"))

# CHECKPOINT ------------------------------------------------------------------

dir.create(".checkpoint")
library("checkpoint")

checkpoint("2023-01-13", R.version ="4.2.1", checkpointLocation = getwd())

# C++ CODE --------------------------------------------------------------------

# 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])
}
```

2 Define functions

```r
# DEFINE FUNCTIONS ###################################################################

# SAVAGE SCORES ------------------------------------------------------------------

savage_scores <- function(x, type) {

  if (type == "ersatz") {

    true.ranks <- rank(x)

  } else {

    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)
}

# SALTELLI ERSATZ DISCREPANCY ----------------------------------------------------

saltelli_ersatz <- function(mat) {

  N <- nrow(mat)

  s <- ceiling(sqrt(N))

  # Create the zero matrix
  mat_zeroes <- matrix(0, s, s)

  # Compute index for x_i
  m <- ceiling(mat[, 1] * s)

  # Compute index for y
  x <- mat[, 2]
  n_norm <- (x-min(x))/(max(x)-min(x)) # Scale y to 0, 1
  n <- ceiling(n_norm * s)

  # Turn y==0 to y == 1
  n <- ifelse(n == 0, 1, n)
```

```
  # Merge and identify which cells are occupied by points
  ind <- cbind(m, n)
  mat_zeroes[ind] <- 1

  # Compute discrepancy
  out <- sum(mat_zeroes==1) / N

  return(out)
}

# DISCREPANCY FUNCTION WRAP-UP ------------------------------------------------

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

# Wrap up 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))

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

    R <- saltelli_ersatz(X)

  }
  return(R)
}

# Final 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)
}

# FUNCTION TO COMPUTE JANSEN T_I ------------------------------------------

jansen_ti <- function(d, N, params) {

  m <- matrix(d, nrow = N)
  k <- length(params)
```

```r
  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)
}
```

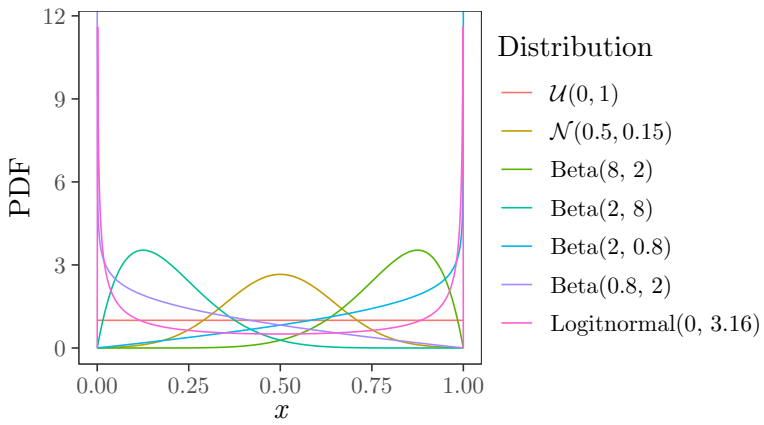
3 Metafunction

```r
# DISTRIBUTIONS OF THE METAFUNCTIONS ##########################################

# 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)
```

```r
  if(!phi == length(names_ff) + 1) {
    out <- sample_distributions[[names_ff[phi]]](X)
  } else {
    temp <- sample(names_ff, ncol(X), replace = TRUE)
    out <- sapply(seq_along(temp), function(x)
      sample_distributions[[temp[x]]](X[, x]))
  }
  return(out)
}
```

```r
# PLOT DISTRIBUTIONS ############################################################

# 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) logitnorm::dlogitnorm(x, 0, 3.16)
)

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)

plot.distributions <- ggplot(dt, aes(x = x, y = value, group = variable)) +
  geom_line(aes(color = variable)) +
  scale_color_discrete(labels = c("$\\mathcal{U}(0, 1)$",
                                  "$\\mathcal{N}(0.5, 0.15)$",
                                  "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") +
```

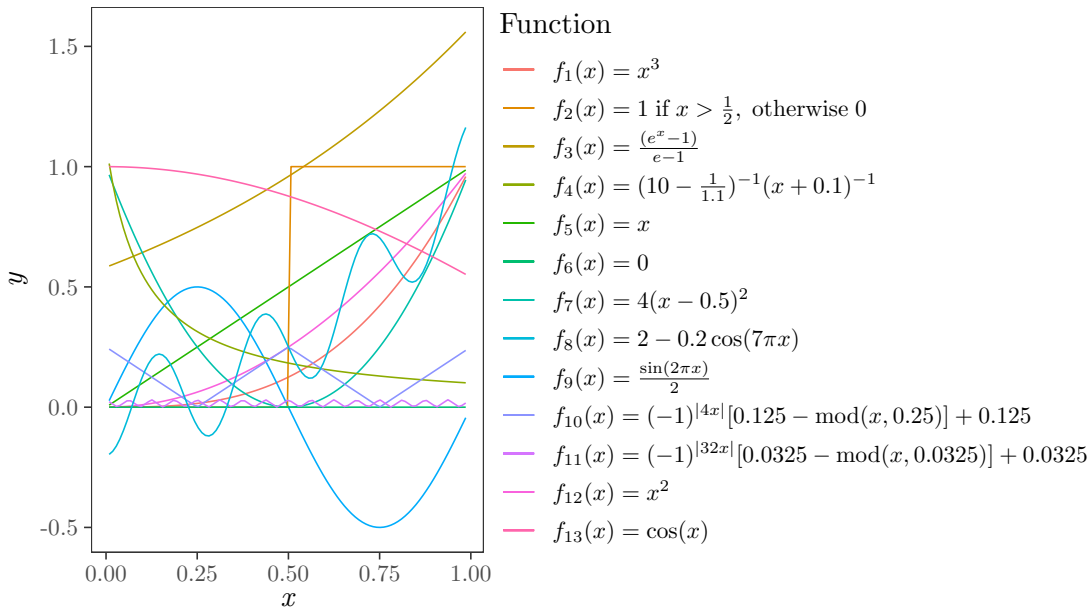
```
  theme_AP() +
  theme(legend.text.align = 0)

plot.distributions
```




```
# PLOT METAFUNCTION ##################################################################

plot.metafunction <- ggplot(data.frame(x = runif(100)), aes(x)) +
  map(1:length(function_list), function(nn) {
    stat_function(fun = function_list[[nn]],
                  geom = "line",
                  aes_(color = factor(names(function_list[nn]))))
  }) +
  labs(color= "Function", linetype = "Function",
       x = expression(italic(x)),
       y = expression(italic(y))) +
  scale_color_discrete(labels = c("$f_1(x) = x^3$",
                                  "$f_2(x) = 1 \\hspace{1mm} \\mbox{if} \\hspace{1mm} x > \\fra
                                  "$f_3(x) = \\frac{(e^x - 1)}{e-1}$",
                                  "$f_4(x) = (10-\\frac{1}{1.1})^{-1}(x + 0.1)^{-1}$",
                                  "$f_5(x) = x$",
                                  "$f_6(x) = 0$",
                                  "$f_7(x) = 4(x - 0.5)^2$",
                                  "$f_8(x) = 2 - 0.2 \\cos(7 \\pi x)$",
                                  "$f_9(x) = \\frac{\\sin(2 \\pi x)}{2}$",
                                  "$f_{10}(x) = (-1)^ {|4x|} [0.125- \\mbox{mod}(x, 0.25)] + 0
                                  "$f_{11}(x) = (-1)^ {|32x|} [0.0325-\\mbox{mod}(x, 0.0325)] +
                                  "$f_{12}(x) = x^2$",
                                  "$f_{13}(x) = \\cos(x)$")) +
  theme_AP() +
  theme(legend.text.align = 0)
```

```
## Warning: `aes_()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation ideoms with `aes()`
```

```
plot.metafunction
```



4 Plots to show discrepancy

```
# LIST OF FUNCTIONS ##################################################################

# NEW FUNCTIONS ------------------------------------------------------------------
f1_fun <- function(x) 10 * x[, 1] + 0.2 * x[, 2]^3

f2_fun <- function(x) 2 * x[, 1] - x[, 2]^2

f3_fun <- function(x) x[, 1]^2 + x[, 2]^4 + x[, 1] * x[, 2] + x[, 2] * x[, 3]^4

f4_fun <- function(x) 0.2 * exp(x[, 1] - 3) + 2.2 * abs(x[, 2]) + 1.3 * x[, 2]^6 -
  2 * x[, 2]^2 - 0.5 * x[, 2]^4 - 0.5 * x[, 1]^4 + 2.5 * x[, 1]^2 + 0.7 * x[, 1]^3 +
  3 / ((8 * x[, 1] - 2)^2 + (5 * x[, 2] - 3)^2 + 1) + sin(5 * x[, 1]) * cos(3 * x[, 1]^2)

f6_fun <- function(x) 0.2 * exp(x[, 1] + 2 * x[, 2])

fun_vec <- paste("f", 1:4, "_fun", sep = "")

f_list <- list(f1_fun, f2_fun, f3_fun, f4_fun)
names(f_list) <- fun_vec

# RUN FUNCTIONS ------------------------------------------------------------------
output <- ind <- list()

for (i in names(f_list)) {
```

```r
  if(i == "f3_fun") {

    k <- 3

  } else {

    k <- 2
  }
  params <- paste("$x_", 1:k, "$", sep = "")
  N <- 100
  mat <- sobol_matrices(N = N, params = params, scrambling = 1)
  y <- f_list[[i]](mat)
  y <- rescale_fun(y)
  ind[[i]] <- sobol_indices(Y = y, N = N, params = params)
  output[[i]] <- plot_scatter(data = mat, N = N, Y = y, params = params) +
    labs(x = "$x$", y = "$y$") +
    scale_x_continuous(breaks = c("0" = 0, "0.5" = 0.5, "1" = 1)) +
    scale_y_continuous(breaks = c("0" = 0, "0.5" = 0.5, "1" = 1)) +
    theme(plot.margin = unit(c(0, 0.2, 0, 0), "cm"))
}
```

```
## Warning in randtoolbox::sobol(n = N, dim = k * n.matrices, ...): scrambling is
## currently disabled.

## Warning in randtoolbox::sobol(n = N, dim = k * n.matrices, ...): scrambling is
## currently disabled.

## Warning in randtoolbox::sobol(n = N, dim = k * n.matrices, ...): scrambling is
## currently disabled.

## Warning in randtoolbox::sobol(n = N, dim = k * n.matrices, ...): scrambling is
## currently disabled.
```

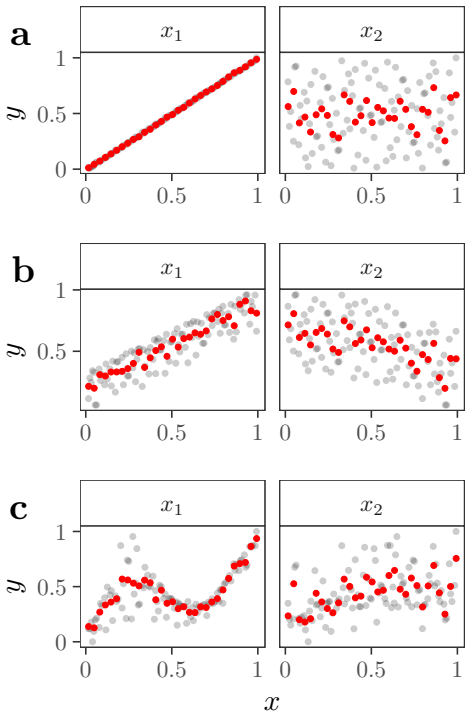
```r
ind
```

```
## $f1_fun
##
## First-order estimator: saltelli | Total-order estimator: jansen
##
## Total number of model runs: 400
##
## Sum of first order indices: 1.062445
##        original sensitivity parameters
## 1: 1.062355e+00          Si      $x_1$
## 2: 8.943357e-05          Si      $x_2$
## 3: 1.064639e+00          Ti      $x_1$
## 4: 3.840524e-04          Ti      $x_2$
##
```

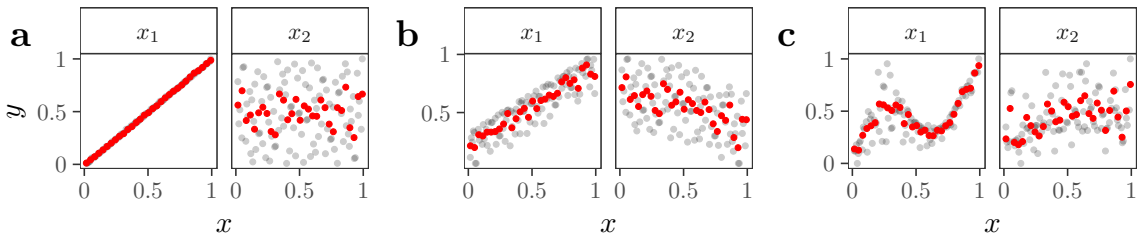
```
## $f2_fun
##
## First-order estimator: saltelli | Total-order estimator: jansen
##
## Total number of model runs: 400
##
## Sum of first order indices: 1.037818
##     original sensitivity parameters
## 1: 0.8303711          Si      $x_1$
## 2: 0.2074472          Si      $x_2$
## 3: 0.8311775          Ti      $x_1$
## 4: 0.2078450          Ti      $x_2$
##
## $f3_fun
##
## First-order estimator: saltelli | Total-order estimator: jansen
##
## Total number of model runs: 500
##
## Sum of first order indices: 0.9844601
##      original sensitivity parameters
## 1: 0.47789926          Si      $x_1$
## 2: 0.45561260          Si      $x_2$
## 3: 0.05094827          Si      $x_3$
## 4: 0.46752749          Ti      $x_1$
## 5: 0.51394544          Ti      $x_2$
## 6: 0.04775163          Ti      $x_3$
##
## $f4_fun
##
## First-order estimator: saltelli | Total-order estimator: jansen
##
## Total number of model runs: 400
##
## Sum of first order indices: 1.11431
##     original sensitivity parameters
## 1: 0.7858947          Si      $x_1$
## 2: 0.3284152          Si      $x_2$
## 3: 0.7622537          Ti      $x_1$
## 4: 0.3619071          Ti      $x_2$
```

```r
# PLOT LIST OF FUNCTIONS ###########################################################

plot_list <- list(output[[1]] + labs(x = "", y = "$y$"),
                  output[[2]] + labs(x = "", y = "$y$"),
                  output[[4]])

scat_plot <- plot_grid(plotlist = plot_list, ncol = 1, labels = "auto")
```

```
scat_plot
```




```
# PLOT LIST OF FUNCTIONS ###########################################################

plot_list <- list(output[[1]] + labs(x = "$x$", y = "$y$"),
                  output[[2]] + labs(x = "$x$", y = ""),
                  output[[4]] + labs(x = "$x$", y = ""))

scat_plot <- plot_grid(plotlist = plot_list, ncol = 3, labels = "auto")
scat_plot
```

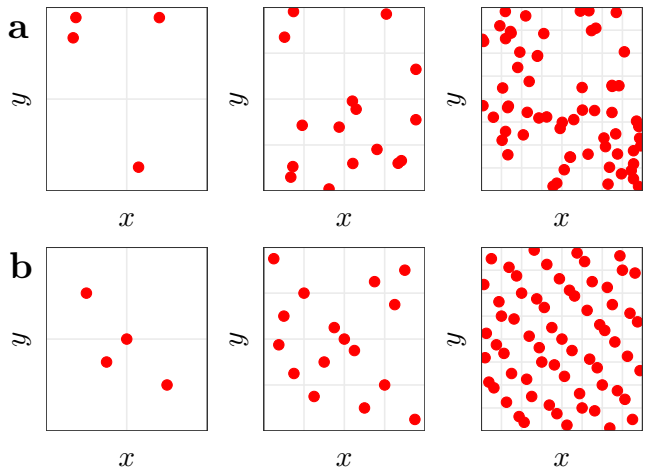


5 Plot sampling points in grid

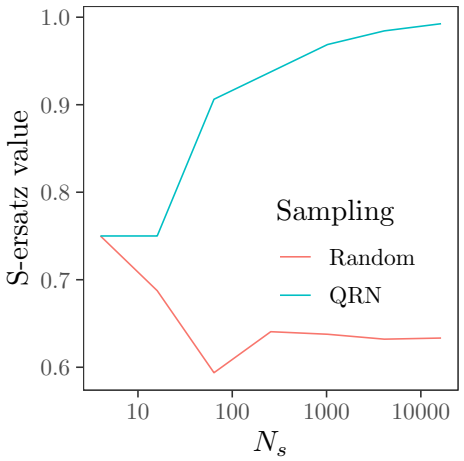
```
# FUNCTION TO DRAW SAMPLING POINTS IN GRID ###################################

plot_grid_fun <- function(N, type, output = "plot") {

  set.seed(2)
  mat <- sobol_matrices(N = N, params = paste("$x_", 1:2, "$", sep = ""),
```

```
                        matrices = "A", type = type)

  n.ersatz <- saltelli_ersatz(mat)

  s <- ceiling(sqrt(N))

  out <- mat %>%
    data.table() %>%
    ggplot(., aes(`$x_1$`, `$x_2$`)) +
    geom_point(color = "red") +
    theme_bw() +
    labs(x = "$x$", y = "$y$") +
    scale_x_continuous(breaks = seq(0, 1, 1/s), expand = c(0, 0), limits = c(0,1)) +
    scale_y_continuous(breaks = seq(0, 1, 1/s), expand = c(0, 0), limits = c(0,1)) +
    theme(panel.grid.minor.x = element_blank(),
          panel.grid.minor.y = element_blank(),
          axis.ticks = element_blank(),
          axis.text = element_blank(),
          plot.margin = unit(c(0.1, 0.1, 0.1, 0.1), "cm"))

  if (output == "plot") {

    return(out)

  } else {

    return(n.ersatz)
  }

}
```

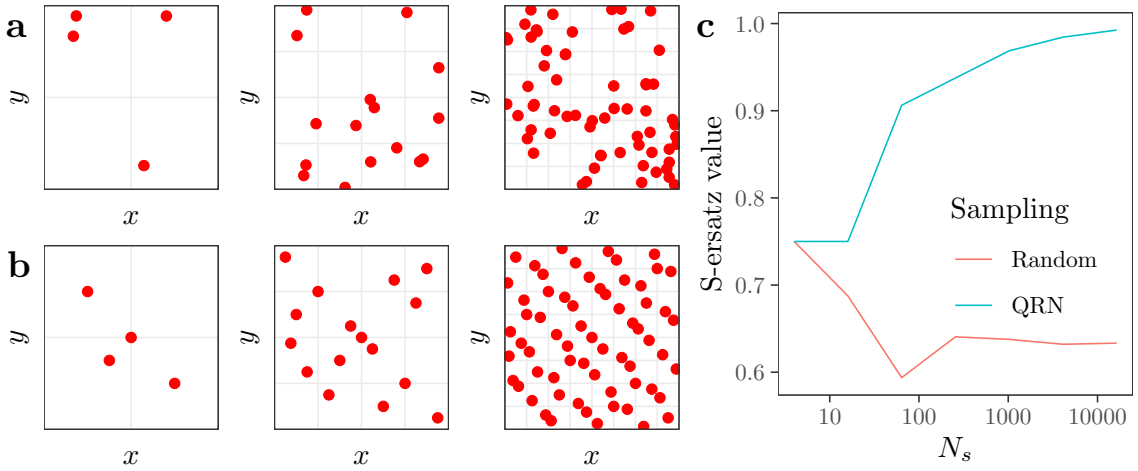
```
# PLOT ########################################################################

# Define loop ------------------------------------------------------------------
type <- c("R", "QRN")
N <- 2^seq(2, 6, 2)

# Plot -------------------------------------------------------------------------
pt <- lapply(type, function(type) lapply(N, function(N) plot_grid_fun(N = N, type = type)))
out <- list()

for (i in 1:length(type)) {
    out[[i]] <- plot_grid(plotlist = pt[[i]], ncol = length(N), labels = "")
}

all.grids <- plot_grid(out[[1]], out[[2]], ncol = 1, labels = "auto")
all.grids
```




```
# EXTRACT ERSATZ MEASURE ###########################################################

N <- 2^seq(2, 14, 2)

ersatz.measure <- lapply(type, function(type) lapply(N, function(N)
  plot_grid_fun(N = N, type = type, output = "ersatz")))

col_names <- c("Random", "QRN")
dt.ersatz <- data.table(do.call(cbind, ersatz.measure))
dt.ersatz <- setnames(dt.ersatz, paste("V", 1:2, sep = ""), col_names)
dt.ersatz <- dt.ersatz[, sample.size:= N]

dt.ersatz[, (col_names):= lapply(.SD, as.numeric), .SDcols = (col_names)]

plot.ersatz <- melt(dt.ersatz, measure.vars = c("Random", "QRN")) %>%
  ggplot(., aes(sample.size, value, color = variable, group = variable)) +
  geom_line() +
  scale_x_log10() +
  labs(x = "$N_s$", y = "S-ersatz value") +
  scale_color_discrete(name = "Sampling") +
  theme_AP() +
  theme(legend.position = c(0.7, 0.35))

plot.ersatz
```




```
plot_grid(all.grids, plot.ersatz, ncol = 2, labels = c("", "c"),
          rel_widths = c(0.6, 0.4))
```



6 The model

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

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

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

  # Define vectors to loop through ----------------------------------------

  simulations <- c("discrepancy", "jansen")
  disc.type <- c("symmetric", "star", "L2", "centered", "wraparound",
                 "modified", "ersatz")

  # Select the sampling method ----------------------------------------------

  if (tau == 1) {
```

```
  type <- "R"

} else if (tau == 2) {

  type <- "QRN"
}

# If statements to select matrices and N as a function
# of the estimator used ---------------------------------------------------

mat <- y <- ind <- disc <- output <-  jansen.results <- list()

for (i in simulations) {

  if (i == "discrepancy") {

    matrices <- "A"
    N <- cost.discrepancy

  } else if (i == "jansen") {
    matrices <- c("A", "AB")
    N <- base.sample.size

  }

  # Construct the sample matrix, randomly transform it
  # according to phi and run the metafunction -------------------------------

  if (i == "discrepancy") {

    set.seed(epsilon)
    mat.uniform <- sobol_matrices(matrices = matrices, N = N, params = params,
                                  type = type)

    set.seed(epsilon)
    mat[[i]] <- random_distributions(sobol_matrices(matrices = matrices,
                                                    N = N, params = params,
                                                    type = type), phi = phi)

  } else if (i == "jansen") {

    set.seed(epsilon)
    mat[[i]] <- random_distributions(sobol_matrices(matrices = matrices,
                                                    N = N, params = params,
                                                    type = type), phi = phi)
  }
```

```
  set.seed(epsilon)
  y[[i]] <- sensobol::metafunction(data = mat[[i]], epsilon = epsilon)

  # Calculate first (saltelli) and total order (jansen) indices
  # and discrepancy values ------------------------------------------------

  if (i == "jansen") {

    ind[[i]] <- jansen_ti(d = y[[i]], N = base.sample.size, params = params)

  }

  if (i == "discrepancy") {

    discrepancy.val <- lapply(disc.type, function(x)
      discrepancy(mat = mat.uniform, y = y[[i]], params = params, type = x))
  }
}

# Arrange output ------------------------------------------------------------

names(discrepancy.val) <- disc.type
all.simulations <- c(ind, discrepancy.val)

# Savage scores -------------------------------------------------------------

all.simulations.savage <- list()

for (i in names(all.simulations)) {

  all.simulations.savage[[i]] <- savage_scores(all.simulations[[i]], type = i)
}

# Correlation between indices and discrepancy measures ----------------------

for (i in disc.type) {

  jansen.results[[i]] <- cor(all.simulations.savage$jansen, all.simulations.savage[[i]])

}

# Arrange output ------------------------------------------------------------

output <- unlist(jansen.results)

return(output)
```

```
}
```

7 Sample matrix

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

# DEFINE SETTINGS ------------------------------------------------------------
N <- 2^9
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))

# RE-ARRANGE COST OF ANALYSIS ------------------------------------------------
cost.jansen <- mat[, "base.sample.size"] * (mat[, "k"] + 1)
cost.saltelli <- mat[, "base.sample.size"] * (mat[, "k"] + 2)
cost.discrepancy <- cost.jansen

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

8 Run simulations

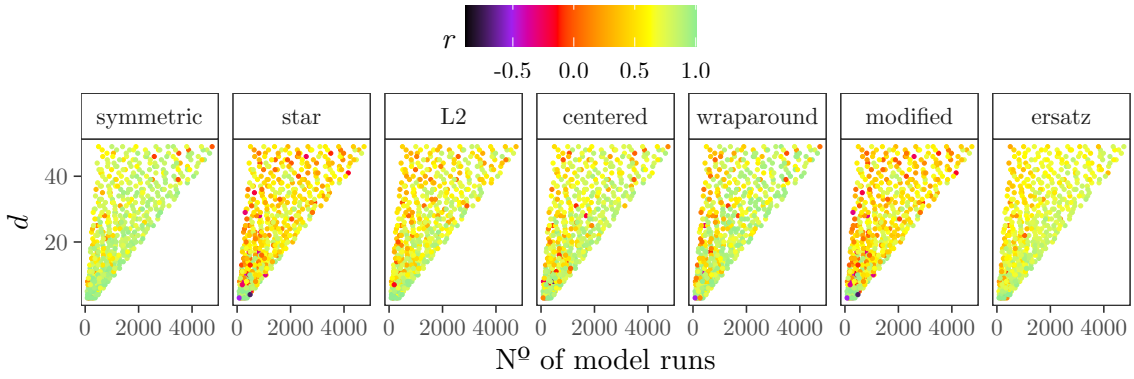
```
# RUN SIMULATIONS ################################################################

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))
```

9 Arrange output

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

final.dt <- do.call(rbind, y) %>%
  cbind(final.mat, .) %>%
```

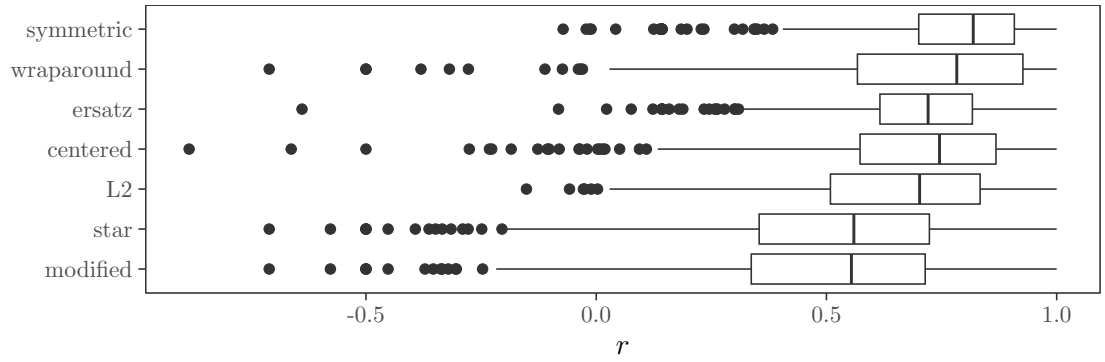
```
  data.table()

# EXPORT RESULTS ------------------------------------------------------------
fwrite(final.dt, "final.dt.csv")
```

```
# PLOT ########################################################################

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

# SCATTERPLOT ---------------------------------------------------------------
scatter <- melt(final.dt, measure.vars = disc.type) %>%
  ggplot(., aes(cost.discrepancy, k, color = value)) +
  geom_point(size = 0.4) +
  scale_colour_gradientn(colours = c("black", "purple", "red", "orange",
                                     "yellow", "lightgreen"),
                         name = expression(italic(r)),
                         breaks = pretty_breaks(n = 3)) +
  facet_grid(~variable) +
  labs(x = "Nº of model runs", y = "$d$") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  theme_AP() +
  theme(legend.position = "none")

legend <- get_legend(scatter + theme(legend.position = "top"))

scatter.plot <- plot_grid(legend, scatter, ncol = 1, rel_heights = c(0.19, 0.81))
scatter.plot
```

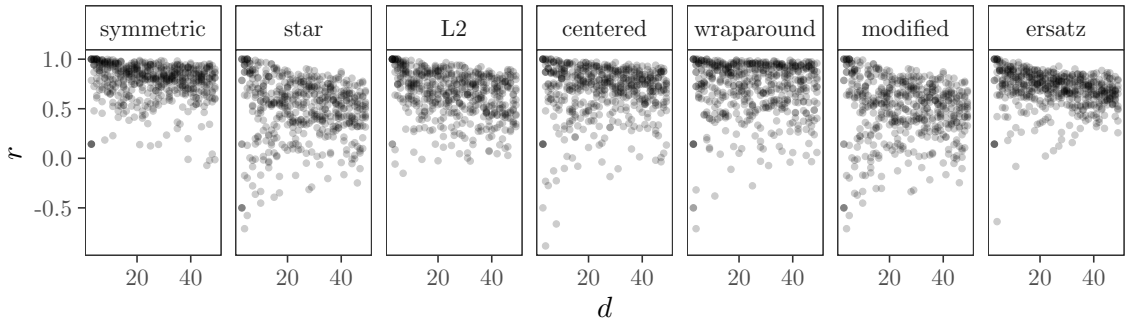



```
# BOXPLOT -------------------------------------------------------------------
boxplots <-  melt(final.dt, measure.vars = disc.type) %>%
  ggplot(., aes(reorder(variable, value), value)) +
  geom_boxplot() +
  coord_flip() +
  labs(y = "$r$", x = "") +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
```

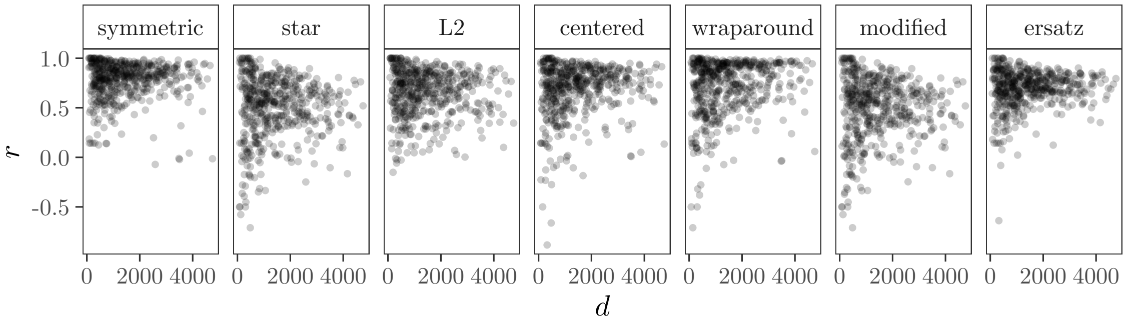
```
  theme_AP()

boxplots
```




```
# PLOT SCATTERPLOTS OF Y AGAINST X_i ##########################################

scatter.d <- melt(final.dt, measure.vars = disc.type) %>%
  ggplot(., aes(k, value)) +
  geom_point(alpha = 0.2, size = 0.8) +
  labs(x = "$d$", y = "$r$") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  facet_wrap(~variable, ncol = 7) +
  theme_AP()

scatter.d
```



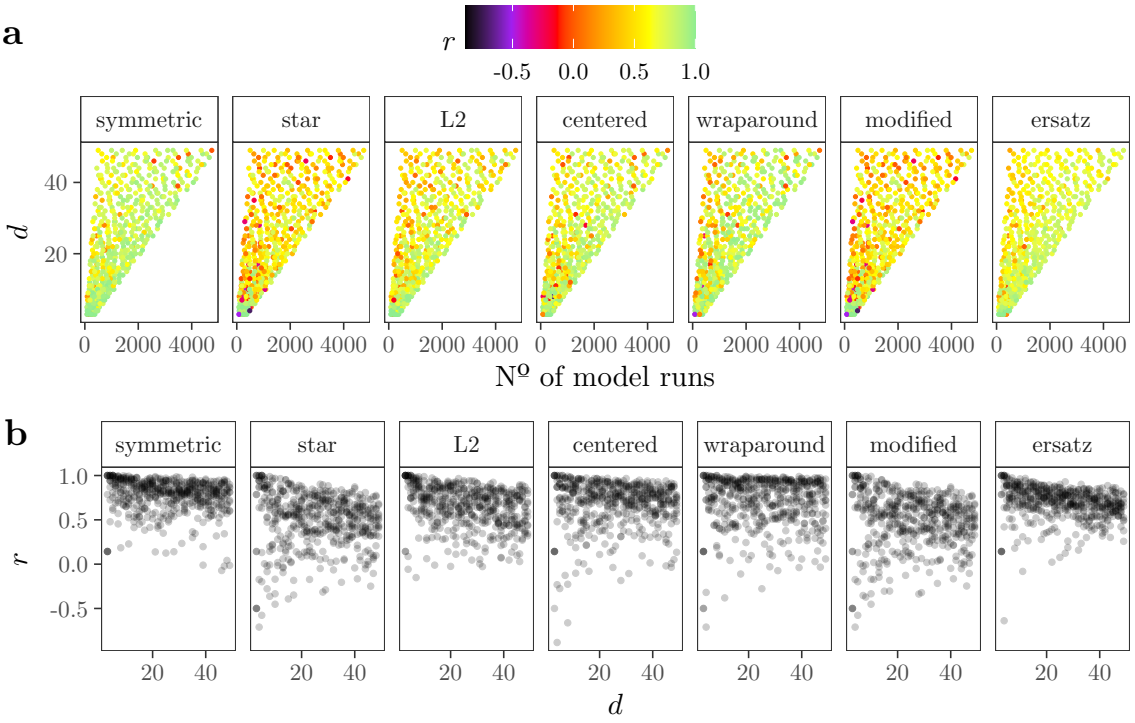

```
scatter.n <- melt(final.dt, measure.vars = disc.type) %>%
  ggplot(., aes(cost.discrepancy, value)) +
  geom_point(alpha = 0.2, size = 0.8) +
  labs(x = "$d$", y = "$r$") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  facet_wrap(~variable, ncol = 7) +
  theme_AP()

scatter.n
```




```
plot_grid(scatter.plot, scatter.d, ncol = 1,
          labels = "auto", rel_heights = c(0.55, 0.45))
```

```
## 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.
```



10 Computing time

```
######## COMPUTING TIME ############################################################

# SAMPLE SIZE-------------------------------------------------------------------

N <- 2^9
params <- c("N", "k")

# SAMPLE MATRIX ----------------------------------------------------------------

dt <- sobol_matrices(matrices = "A", N = N, params = params)
dt[, 1] <- floor(qunif(dt[, 1], 10, 10^3 + 1))
dt[, 2] <- floor(qunif(dt[, 2], 3, 50))

# RUN SIMULATIONS --------------------------------------------------------------

y <- mclapply(1:nrow(dt), function(i) {

  params <- paste("x", 1:dt[i, "k"], sep = "")
  N <- dt[i, "N"]
```

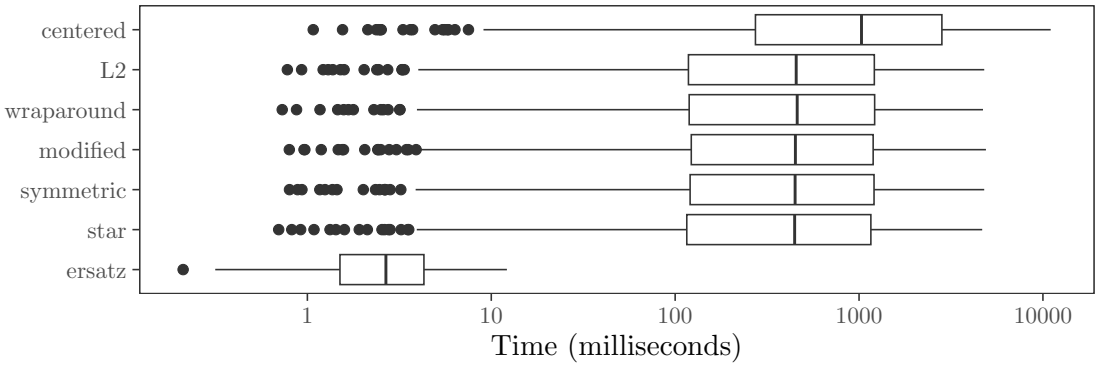
```
  output[[i]] <- lapply(disc.type, function(x) {

    mat <- sobol_matrices(matrices = "A", N = N, params = params)
    y <- metafunction(data = mat)
    da <- microbenchmark(discrepancy(mat = mat, y = y, params = params, type = x),
                         times = 1)$time

  })

}, mc.cores = detectCores())
```

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

da <- list()

for (i in 1:length(y)) {

  names(y[[i]]) <- disc.type
  da[[i]] <- data.frame(do.call(rbind, y[[i]])) %>%
    rownames_to_column(., var = "discrepancy") %>%
    data.table() %>%
    setnames(., colnames(.), c("discrepancy", "time")) %>%
    .[, time:= time / 10^6] # From nano to milliseconds

}

timing.dt <- rbindlist(da, idcol = "row")
fwrite(timing.dt, "timing.dt.csv")
fwrite(dt, "dt.csv")
```

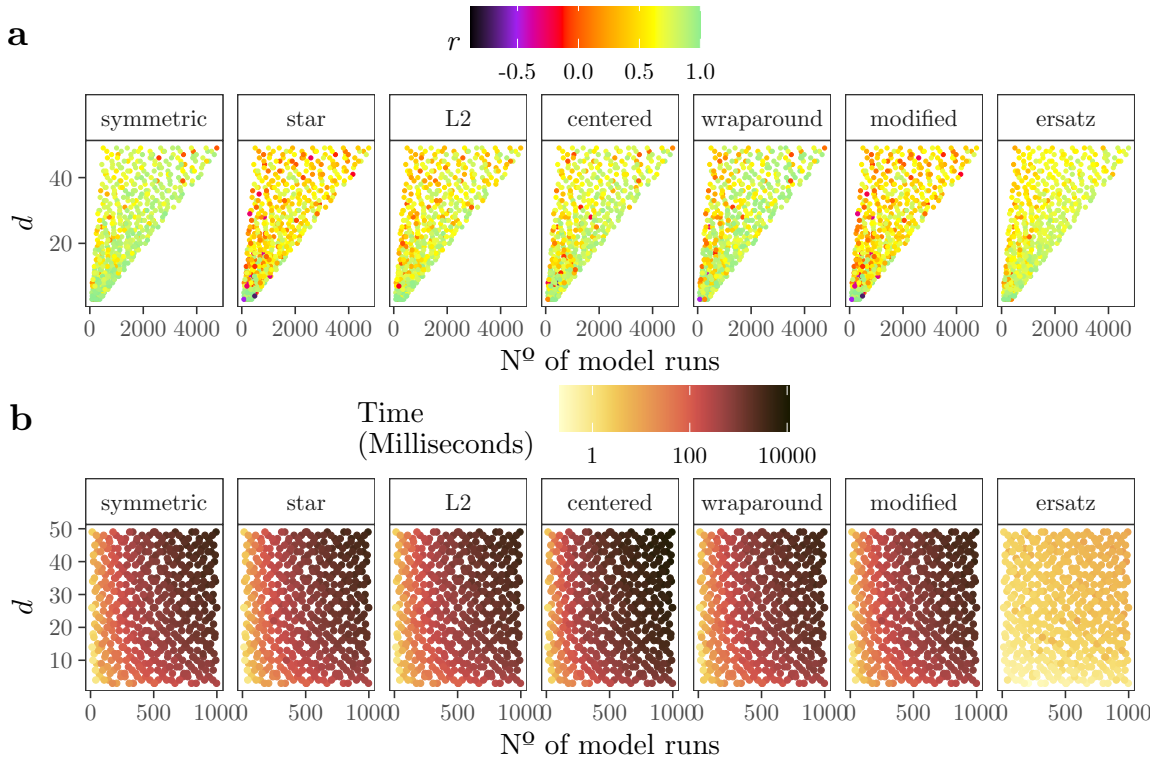
```
## x being coerced from class: matrix to data.table
```

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

# SCATTERPLOT ------------------------------------------------------------------
scatter.timing <- dcast(timing.dt, row ~ discrepancy, value.var = "time") %>%
  cbind(dt, .) %>%
  melt(., measure.vars = disc.type) %>%
  ggplot(., aes(N, k, color = value)) +
  geom_point(size = 0.8) +
  scico::scale_colour_scico(palette = "lajolla",
                            breaks = c(1, 10^2, 10^4),
                            trans = "log",
                            name = "Time \n (Milliseconds)") +
  facet_grid(~variable) +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  labs(x = "Nº of model runs", y = "$d$") +
  theme_AP() +
```

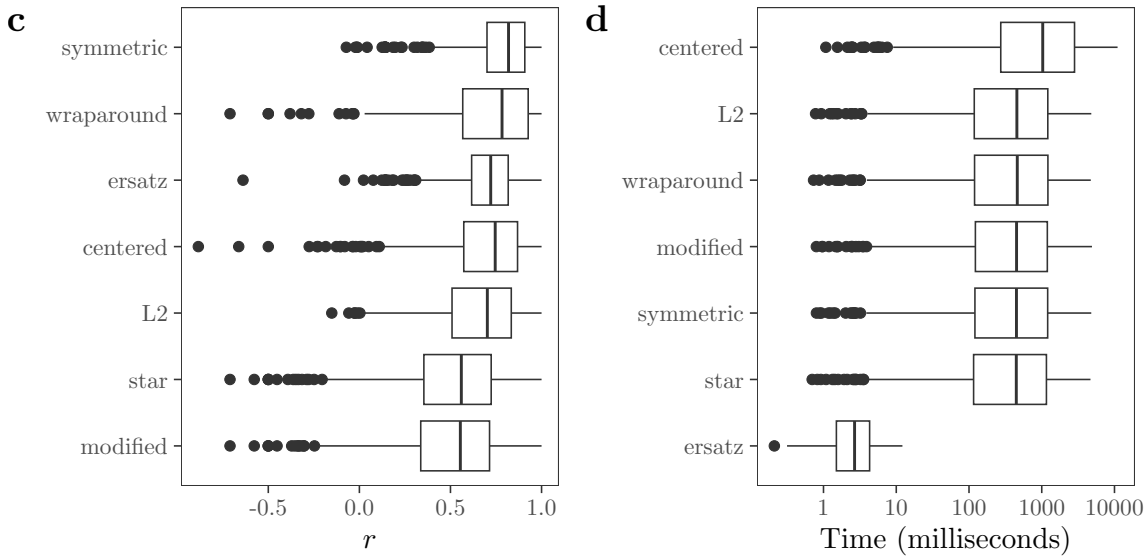
```
  theme(legend.position = "none")

legend <- get_legend(scatter.timing + theme(legend.position = "top"))

time.plot <- plot_grid(legend, scatter.timing, ncol = 1, rel_heights = c(0.19, 0.81))

# BOXPLOT ----------------------------------------------------------------------
boxplot.timing <- ggplot(timing.dt, aes(reorder(discrepancy, time), time)) +
  geom_boxplot() +
  coord_flip() +
  labs(x = "", y = "Time (milliseconds)") +
  scale_y_log10() +
  theme_AP()

boxplot.timing
```



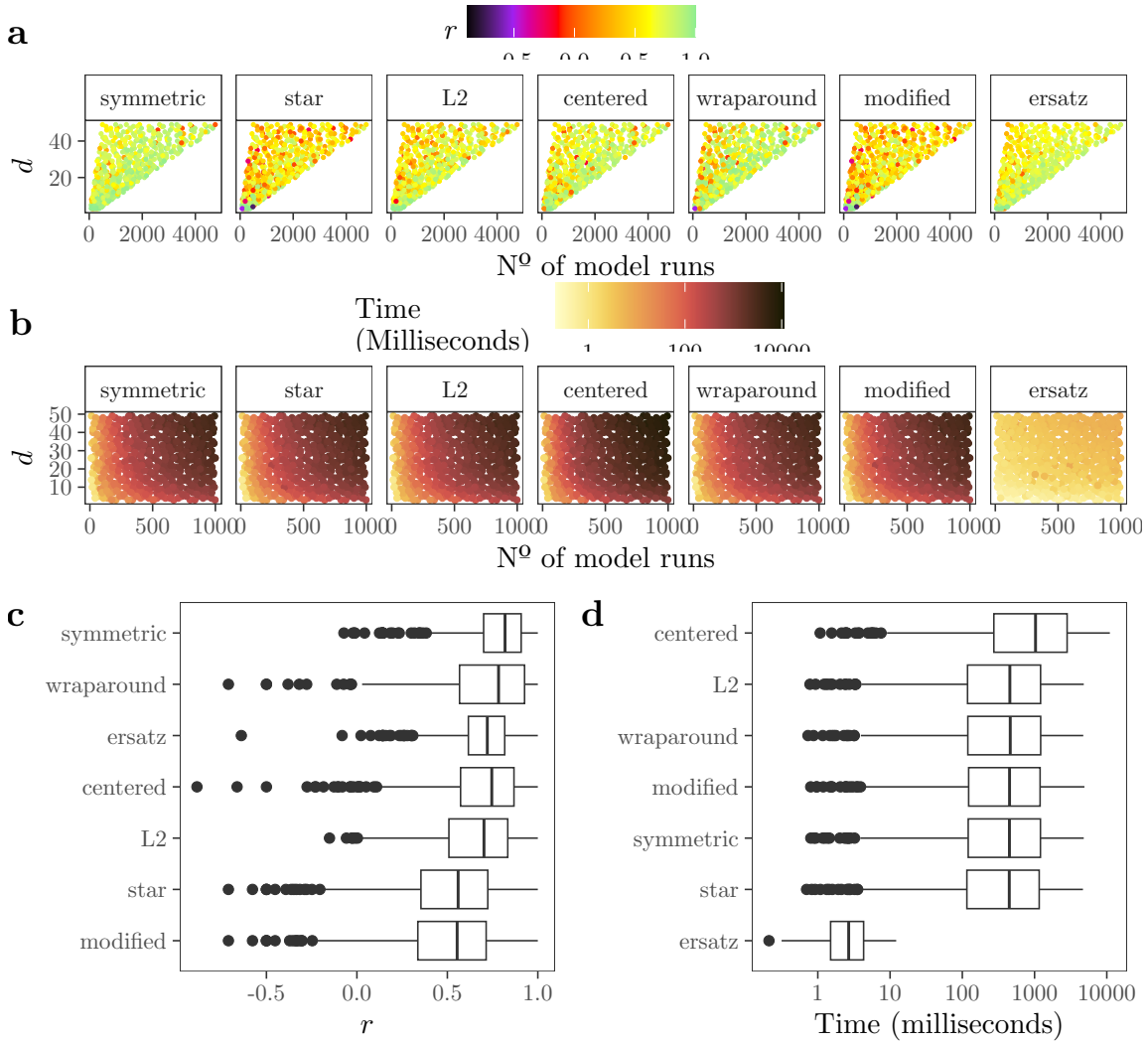

```
# MERGE PLOTS ##################################################################

all.scatter <- plot_grid(scatter.plot, time.plot, ncol = 1, labels = "auto")
all.scatter
```




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

all.boxplots <- plot_grid(boxplots, boxplot.timing, ncol = 2, labels = c("c", "d"))
all.boxplots
```




```
plot_grid(all.scatter, all.boxplots, ncol = 1, rel_heights = c(0.5, 0.4))
```

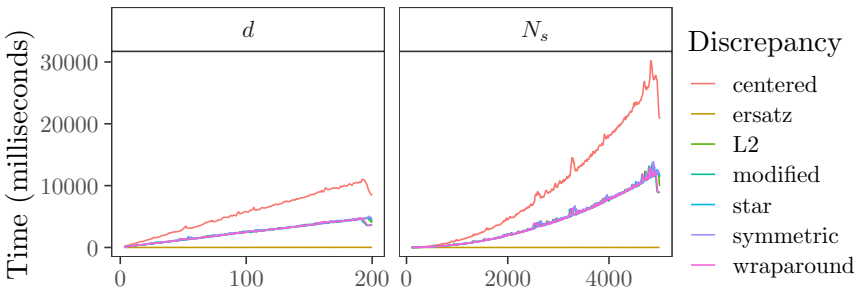



```
# CALCULATE TIME COMPLEXITY ###########################################################

# Setting ------------------------------------------------------------------------
sample.size <- seq(100, 5000, 10)
dimensions <- seq(3, 200, 1)

# Function -----------------------------------------------------------------------
timing_fun <- function(N, k) {

  params <- paste("X", 1:k, sep = "")
  mat <- sobol_matrices(matrices = "A", N = N, params = params)
  y <- metafunction(mat)
  out <- lapply(disc.type, function(x)
    microbenchmark(discrepancy(mat = mat, y = y, params = params, type = x),
                   times = 1)$time)
  names(out) <- disc.type

  return(out)
```

```
}

# Run model ------------------------------------------------------------------
timing.N <- mclapply(sample.size, function(N)
  timing_fun(N = N, k = 5), mc.cores = detectCores())

timing.dimensions <- mclapply(dimensions, function(k)
  timing_fun(N = 500, k = k), mc.cores = detectCores())
```

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

names(timing.N) <- sample.size
names(timing.dimensions) <- dimensions

timing.N.dt <- lapply(timing.N, function(x) lapply(x, function(y) data.table(y))) %>%
  lapply(., function(x) rbindlist(x, idcol = "Discrepancy")) %>%
  rbindlist(., idcol = "approach") %>%
  .[, approach:= as.numeric(approach)] %>%
  .[, method:= "$N_s$"] %>%
  .[, y:= y / 10^6]

timing.dimensions.dt <- lapply(timing.dimensions, function(x) lapply(x, function(y) data.table
  lapply(., function(x) rbindlist(x, idcol = "Discrepancy")) %>%
  rbindlist(., idcol = "approach") %>%
  .[, approach:= as.numeric(approach)] %>%
  .[, method:= "$d$"] %>%
  .[, y:= y / 10^6]

full.timing.dt <- rbind(timing.N.dt, timing.dimensions.dt)

fwrite(full.timing.dt, "full.timing.dt.csv")
```

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

time.complexity <- ggplot(full.timing.dt, aes(approach, y,
                                              group = Discrepancy,
                                              color = Discrepancy)) +
  geom_line() +
  theme_AP() +
  facet_wrap(~method, scales = "free_x") +
  labs(x = "", y = "Time (milliseconds)") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  theme(legend.key.width = unit(0.4, "cm"),
        legend.key.height = unit(0.4, "cm"))

time.complexity
```


d
N_s
Time (milliseconds)
30000
20000
10000
0
0
100
200
0
2000
4000
Discrepancy
centered
ersatz
L2
modified
star
symmetric
wraparound

11 Session information

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

sessionInfo()
```

```
## R version 4.2.0 (2022-04-22)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-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] wesanderson_0.3.6 cowplot_1.1.1     benchmarkme_1.0.7 doParallel_1.0.17
##  [5] iterators_1.0.14  foreach_1.5.2     scales_1.2.0      RcppAlgos_2.5.3
##  [9] forcats_0.5.1     stringr_1.4.0     dplyr_1.0.9       purrr_0.3.4
## [13] readr_2.1.2       tidyr_1.2.0       tibble_3.1.7      ggplot2_3.3.6
## [17] tidyverse_1.3.1   data.table_1.14.2 sensobol_1.1.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.8.3          lattice_0.20-45       lubridate_1.8.0
##  [4] gmp_0.6-5             assertthat_0.2.1      digest_0.6.29
##  [7] utf8_1.2.2            R6_2.5.1              cellranger_1.1.0
## [10] backports_1.4.1       reprex_2.0.1          evaluate_0.15
## [13] httr_1.4.3            pillar_1.7.0          Rdpack_2.3
## [16] rlang_1.0.2           readxl_1.4.0          rstudioapi_0.13
## [19] Matrix_1.4-1          tikzDevice_0.12.3.1   rmarkdown_2.14
## [22] munsell_0.5.0         broom_0.8.0           compiler_4.2.0
## [25] modelr_0.1.8          xfun_0.31             pkgconfig_2.0.3
## [28] htmltools_0.5.2       tidyselect_1.1.2      codetools_0.2-18
## [31] fansi_1.0.3           crayon_1.5.1          tzdb_0.3.0
## [34] dbplyr_2.1.1          withr_2.5.0           rbibutils_2.2.8
## [37] grid_4.2.0            jsonlite_1.8.0        gtable_0.3.0
## [40] lifecycle_1.0.1       DBI_1.1.2             magrittr_2.0.3
## [43] cli_3.3.0             stringi_1.7.6         fs_1.5.2
## [46] benchmarkmeData_1.0.4 xml2_1.3.3            ellipsis_0.3.2
## [49] generics_0.1.2        vctrs_0.4.1           tools_4.2.0
## [52] glue_1.6.2            hms_1.1.1             fastmap_1.1.0
## [55] yaml_2.3.5            colorspace_2.0-3      filehash_2.4-3
```

```
## [58] rvest_1.0.2           knitr_1.39            haven_2.5.0
```

```
## 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 = FALSE))
```

```
## Num threads:
```

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