

# A sensitivity analysis of the PAWN sensitivity index

*Arnald Puy, Samuele Lo Piano, Andrea Saltelli*

## Contents

<b>1</b>	<b>Preliminary functions</b>	<b>2</b>
<b>2</b>	<b>Check convergence of Sobol' indices and PAWN</b>	<b>2</b>
2.1	Sample matrix . . . . .	3
2.2	Model output . . . . .	3
2.3	Sobol' indices . . . . .	4
2.4	PAWN . . . . .	6
2.5	Plot convergence . . . . .	7
<b>3</b>	<b>Sensitivity of PAWN to its design parameters</b>	<b>10</b>
3.1	The model . . . . .	10
3.2	Settings . . . . .	11
3.3	Sample matrix . . . . .	12
3.4	Run the model . . . . .	13
3.5	Uncertainty analysis . . . . .	14
3.6	Sensitivity analysis . . . . .	19
<b>4</b>	<b>Sensitivity of Sobol' indices to its design parameters</b>	<b>22</b>
4.1	The model . . . . .	22
4.2	Settings . . . . .	24
4.3	Sample matrix . . . . .	24
4.4	Run the model . . . . .	25
4.5	Uncertainty analysis . . . . .	26
4.6	Sensitivity analysis . . . . .	30
<b>5</b>	<b>Extra plots</b>	<b>33</b>
<b>6</b>	<b>Session information</b>	<b>51</b>

## 1 Preliminary functions

```
# PRELIMINARY FUNCTIONS -----

# Install the development version of the pawnr package
devtools::install_github("arnalduy/pawnr", build_vignettes = TRUE)

# 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("tidyverse", "data.table", "randtoolbox", "sensitivity",
              "boot", "parallel", "doParallel", "scales", "cowplot",
              "overlapping", "pawnr", "sensobol", "sensitivity", "wesanderson"))

# Set checkpoint

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

checkpoint("2019-09-22",
          R.version = "3.6.1",
          checkpointLocation = getwd())
```

## 2 Check convergence of Sobol' indices and PAWN

```
# DEFINE SETTINGS -----

N <- seq(500, 10000, 250) # Sample sizes
n <- 15 # Number of conditioning intervals
k <- c(2, 3, 8, 20) # Vector with number of model inputs
R <- 100 # Bootstrap replicas
n_cores <- floor(detectCores() * 0.75) # Use 75% of the cores available
type <- "norm" # Define the confidence interval method
conf <- 0.95 # Define the ci
models <- c("Liu", "Ishigami", "Sobol' G", "Morris")
params <- lapply(k, function(x) paste("X", 1:x, sep = ""))
names(params) <- models
```

```
# Function to compute the Liu et al. function
```

```
liu <- function(X1, X2) {  
  X1 / X2  
}
```

```
liu_Mapply <- function(X) {  
  X[, 1] <- qchisq(X[, 1], df = 10)  
  X[, 2] <- qchisq(X[, 2], df = 13.978)  
  return(mapply(liu, X[, 1], X[, 2]))  
}
```

## 2.1 Sample matrix

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

```
A <- list()  
for(i in k) {  
  A[[i]] <- mclapply(N, function(N) sobol_matrices(n = N, k = i), mc.cores = n_cores)  
}
```

```
A <- A[!sapply(A, is.null)]  
names(A) <- models
```

```
for(i in names(A)) {  
  names(A[[i]]) <- N  
}
```

## 2.2 Model output

```
# COMPUTE MODEL OUTPUT -----
```

```
Y <- list()  
for(i in names(A)) {  
  if(i == "Liu") {  
    Y[[i]] <- lapply(A[[i]], function(x) liu_Mapply(x))  
  } else if(i == "Ishigami") {  
    Y[[i]] <- lapply(A[[i]], function(x) sensobol::ishigami_Mapply(x))  
  } else if(i == "Sobol' G") {  
    Y[[i]] <- lapply(A[[i]], function(x) sensobol::sobol_Fun(x))  
  } else {  
    Y[[i]] <- lapply(A[[i]], function(x) sensitivity::morris.fun(x))  
  }  
}
```

```
names(Y) <- models  
for(i in names(Y)) {  
  names(Y[[i]]) <- N  
}
```

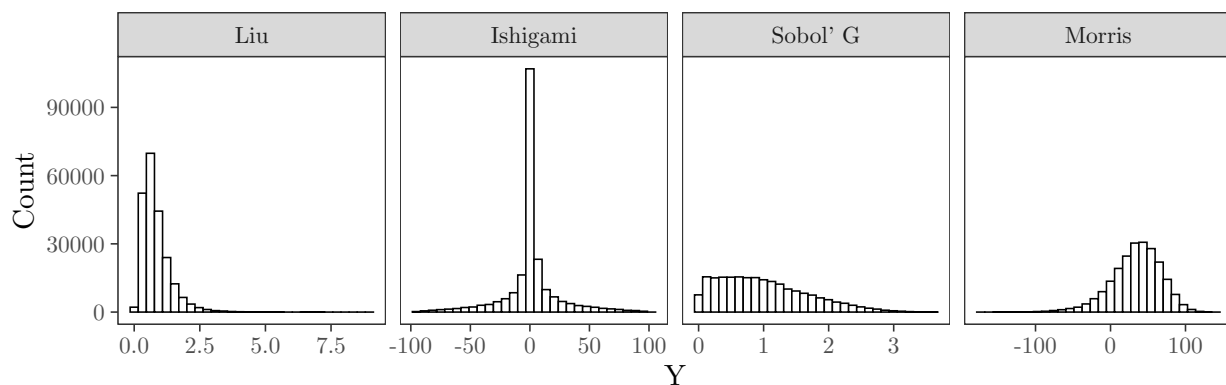
```
}
```

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

lapply(models, function(models) Y[[models]]$`10000`) %>%
  do.call(cbind, .) %>%
  data.table() %>%
  setnames(., 1:4, models) %>%
  melt(., measure.vars = 1:4) %>%
  .[, variable:= factor(variable, levels = models)] %>%
  ggplot(., aes(value)) +
  geom_histogram(color = "black",
                 fill = "white") +
  labs(x = "Y",
       y = "Count") +
  facet_wrap(~ variable,
            scales = "free_x",
            ncol = 4) +
  theme_bw() +
  theme(aspect.ratio = 1,
        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))
```

```
## Warning in (function (... , deparse.level = 1) : number of rows of result is
## not a multiple of vector length (arg 1)
```

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



## 2.3 Sobol' indices

```
# COMPUTE SOBOLE' INDICES AND THEIR CONFIDENCE INTERVALS -----

out <- out.ci <- list()
```

```

for(i in names(A)) {
  for(j in names(A[[i]])) {
    out[[i]][[j]] <- sobol_indices(Y[[i]][[j]],
                                   params = params[[i]],
                                   n = as.numeric(j),
                                   type = "saltelli",
                                   R = R,
                                   parallel = "multicore",
                                   ncpus = n_cores)
    out.ci[[i]][[j]] <- sobol_ci(out[[i]][[j]],
                                 params = params[[i]],
                                 type = type,
                                 conf = conf)
  }
}

# SOBOL INDICES AND CONFIDENCE INTERVALS OF DUMMY PARAMETER -----

sobol.dummy <- sobol.dummy.ci <- list()
for(i in names(A)) {
  for(j in names(A[[i]])) {
    sobol.dummy[[i]][[j]] <- sobol_dummy(Y[[i]][[j]],
                                           params = params[[i]],
                                           R = R,
                                           n = as.numeric(j),
                                           parallel = "multicore",
                                           ncpus = n_cores)
    sobol.dummy.ci[[i]][[j]] <- sobol_ci_dummy(sobol.dummy[[i]][[j]],
                                                type = type,
                                                conf = conf)
  }
}

sobol.dummy.final <- lapply(sobol.dummy.ci, function(x) rbindlist(x, idcol = "N")) %>%
  rbindlist(., idcol = "model") %>%
  .[, model:= factor(model, levels = c("Liu", "Ishigami",
                                       "Sobol' G", "Morris"))]

# SOBOL' CONVERGENCE -----

sobol.convergence <- lapply(out.ci, function(x) rbindlist(x, idcol = "N")) %>%
  rbindlist(., idcol = "model") %>%
  .[, N:= as.numeric(N)] %>%
  .[, diff:= high.ci - low.ci] %>%
  .[, model:= factor(model, levels = c("Liu", "Ishigami",
                                       "Sobol' G", "Morris"))] %>%
  .[, parameters:= factor(parameters,
                          levels = paste("X", 1:20, sep = ""))] %>%

```

```
.[, method:= "Sobol' $S_{Ti}$"] %>%
.[, .(model, N, parameters, original, low.ci, high.ci, diff, method, sensitivity)]
```

## 2.4 PAWN

```
# COMPUTE PAWN INDICES AND THEIR CONFIDENCE INTERVALS -----

# Subset to take only the A matrix and the model output of the A matrix
Y.pawn <- A.pawn <- list()
for(i in names(Y)) {
  for(j in names(Y[[i]])) {
    Y.pawn[[i]][[j]] <- Y[[i]][[j]][1:j]
    A.pawn[[i]][[j]] <- A[[i]][[j]][1:j, ]
  }
}

# Compute PAWN indices and their confidence intervals
pawn.indices <- pawn.ci <- list()
for(i in names(A.pawn)) {
  for(j in names(A.pawn[[i]])) {
    pawn.indices[[i]][[j]] <- pawn_generic(data = A.pawn[[i]][[j]],
                                           Y = Y.pawn[[i]][[j]],
                                           n = n,
                                           test = median,
                                           R = R)
    pawn.ci[[i]][[j]] <- pawn_ci(pawn.indices[[i]][[j]])
  }
}

# PAWN AND CONFIDENCE INTERVALS OF DUMMY PARAMETER -----

pawn.index.dummy <- list()
for(i in names(Y)) {
  for(j in names(Y[[i]])) {
    pawn.index.dummy[[i]][[j]] <- pawn_dummy(Y = Y[[i]][[j]],
                                              n = n,
                                              R = R)
  }
}

pawn.index.dummy <- lapply(pawn.index.dummy, function(x) rbindlist(x, idcol = "N")) %>%
  rbindlist(., idcol = "model") %>%
  .[, model:= factor(model, levels = c(c("Liu", "Ishigami",
    "Sobol' G", "Morris")))]

# PAWN CONVERGENCE -----

pawn.convergence <- lapply(pawn.ci, function(x) rbindlist(x, idcol = "N")) %>%
```

```

rbindlist(., idcol = "model") %>%
.[, N:= as.numeric(N)] %>%
.[, diff:= high.ci - low.ci] %>%
.[, model:= factor(model, levels = c("Liu", "Ishigami",
                                     "Sobol' G", "Morris"))] %>%
.[, parameters:= gsub("V", "X", parameters)] %>%
.[, parameters:= factor(parameters,
                        levels = paste("X", 1:20, sep = ""))] %>%
.[, method:= "PAWN"]

```

## 2.5 Plot convergence

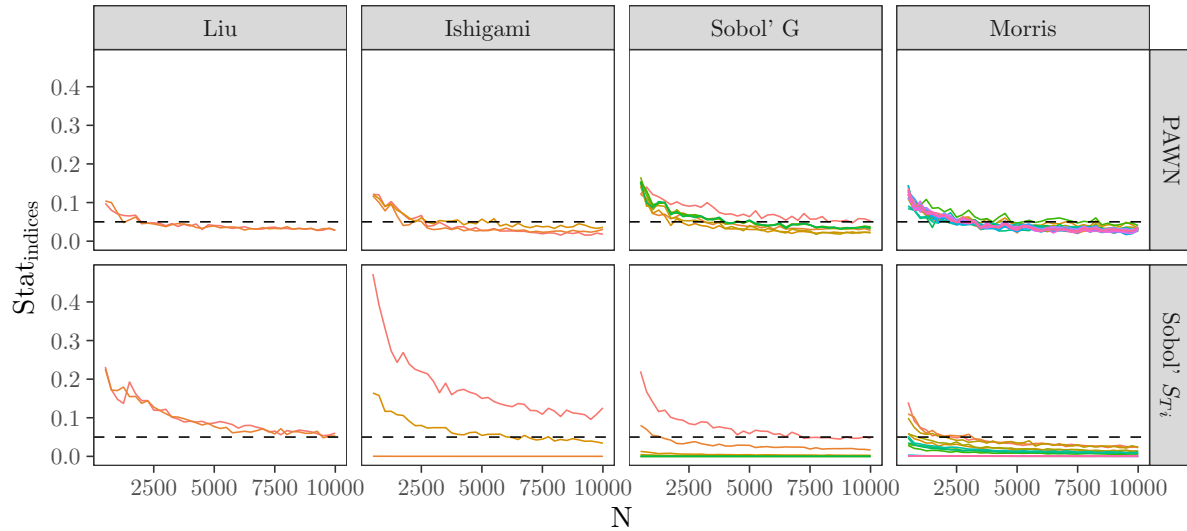
```

# PLOT CONVERGENCE -----
sobol.convergence[sensitivity == "STi"] %>%
.[, sensitivity:= NULL] %>%
rbind(., pawn.convergence) %>%
ggplot(., aes(N, diff,
              group = parameters,
              color = parameters)) +
geom_line() +
geom_hline(yintercept = 0.05,
           lty = 2) +
scale_color_discrete(name = "Model inputs") +
labs(y = expression(Stat[indices]),
     x = "N") +
facet_grid(method~model) +
theme_bw() +
theme(legend.position = "top",
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      legend.background = element_rect(fill = "transparent",
                                       color = NA),
      legend.key = element_rect(fill = "transparent",
                                color = NA))

```

Model inputs

— X1	— X5	— X9	— X13	— X17
— X2	— X6	— X10	— X14	— X18
— X3	— X7	— X11	— X15	— X19
— X4	— X8	— X12	— X16	— X20



```
# PLOT SOBOLO' AND PAWN INDICES -----

# Sobol' indices
a <- plot_sobol(sobol.convergence[N==4000],
               dummy = sobol.dummy.final[N==4000]) +
  facet_grid(~model,
             scales = "free_x",
             space = "free_x") +
  labs(x = "",
       y = "Sobol' index") +
  theme(axis.text.x = element_text(size = 6),
        legend.position = "none")

# Get legend
legend <- get_legend(a + theme(legend.position = "top"))

# PAWN indices
b <- pawn.convergence[N==4000] %>%
  plot_pawn(.) +
  geom_rect(data = pawn.index.dummy[N==4000],
           aes(ymin = 0,
               ymax = high.ci,
               xmin = -Inf,
               xmax = Inf),
           fill = "black",
           alpha = 0.1,
```



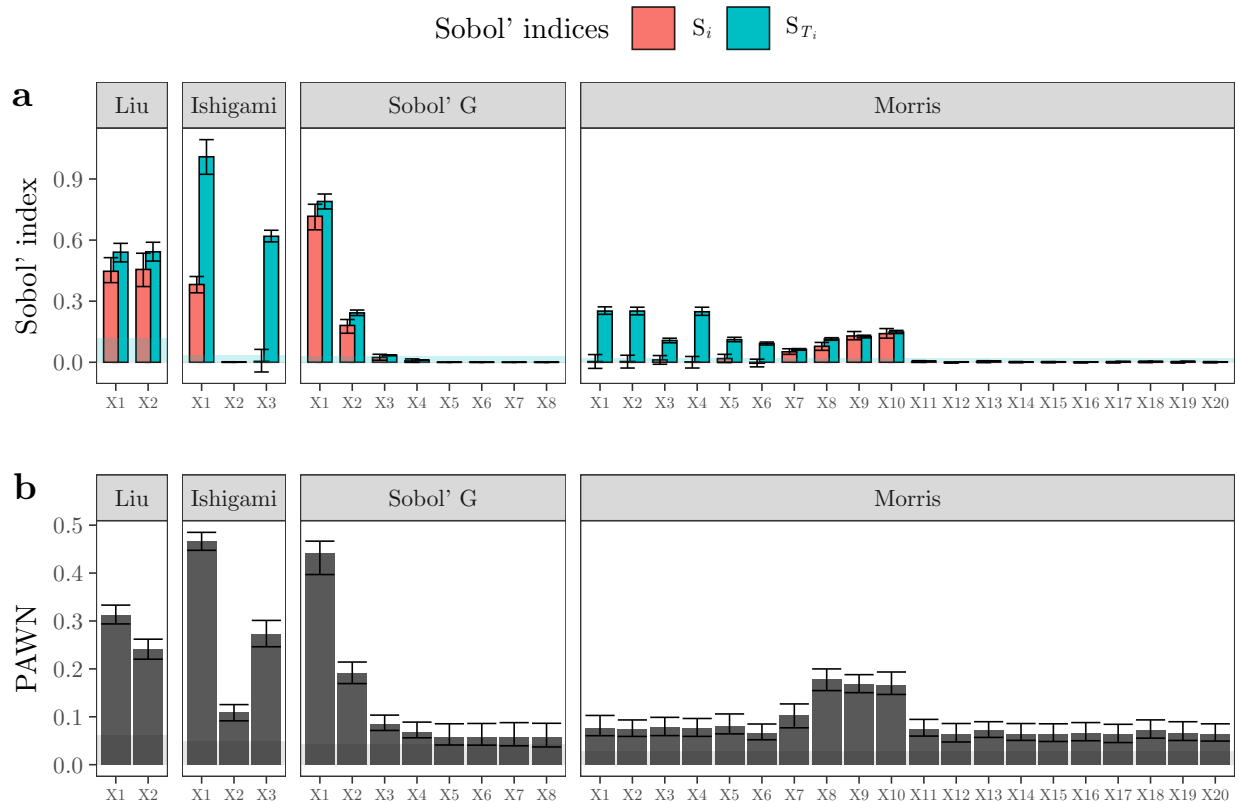
```

        inherit.aes = FALSE) +
labs(x = "",
     y = "PAWN") +
facet_grid(~ model,
           scales = "free_x",
           space = "free_x") +
theme(axis.text.x = element_text(size = 6))

# Merge
bottom <- plot_grid(a, b,
                    ncol = 1,
                    labels = "auto",
                    align = "h")

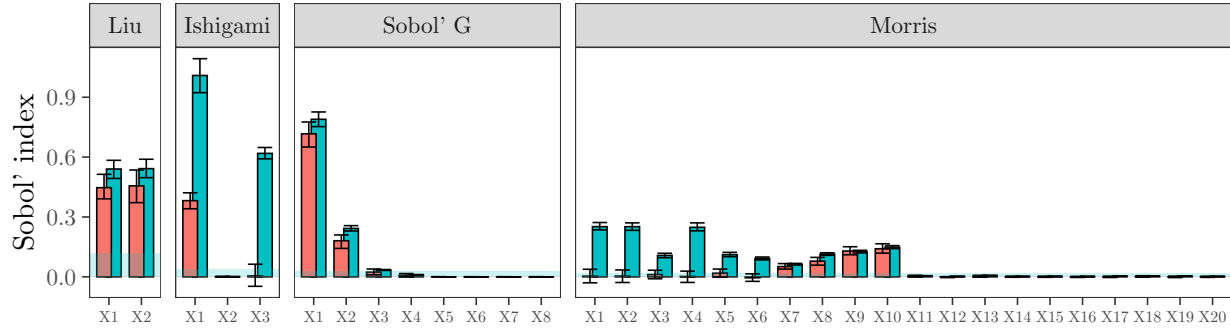
plot_grid(legend, bottom,
          labels = c("", ""),
          ncol = 1,
          align = "",
          rel_heights = c(0.1, 1))

```

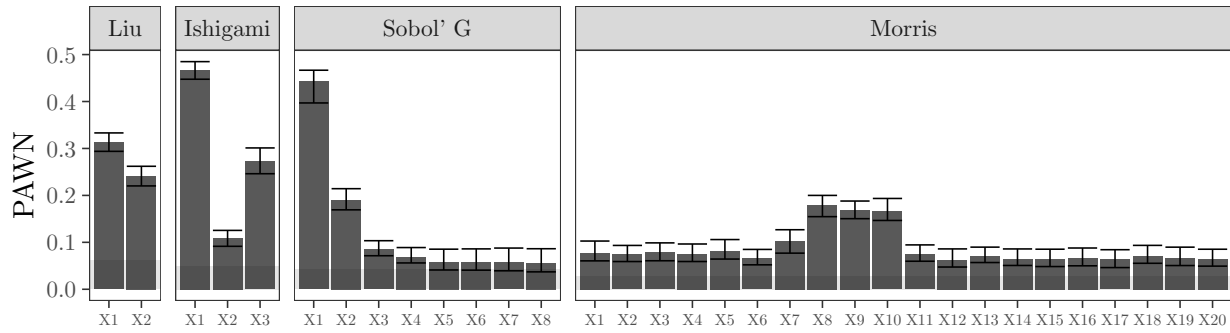


*# PLOT SOBOLE' AND PAWN INDICES (INDIVIDUAL PLOTS) -----*

**a**



b



### 3 Sensitivity of PAWN to its design parameters

#### 3.1 The model

```
# THE MODEL -----

# Function to divide a vector into chunks
chunks <- function(x,n) split(x, cut(seq_along(x), n, labels = FALSE))

# The model
model_pawn <- function(Model, N, n, epsilon, theta) {
  # Check which model to apply to set the number of
  # parameters
  if(Model == 1) {
    k <- 2
  } else if(Model == 2) {
    k <- 3
  } else if(Model == 3) {
    k <- 8
  } else {
    k <- 20
  }
  # Create the Sobol' matrix
  data <- randtoolbox::sobol(n = N, dim = k)
  # Transform distribution:
  if(Model == 1) {
```

```

ModelRun <- liu_Mapply
} else if(Model == 2) {
  ModelRun <- sensobol::ishigami_Mapply
} else if(Model == 3) {
  ModelRun <- sensobol::sobol_Fun
} else {
  ModelRun <- sensitivity::morris.fun
}
# Run the model
Y <- ModelRun(data)
# Set seed to fix the random number generator
# for the sample function below
set.seed(epsilon)
# Sample the unconditional model output
index <- sample(1:nrow(data),
               size = floor(nrow(data) / n),
               # Without replacement
               replace = FALSE)
# Bind model inputs and model output
dt <- data.table::data.table(cbind(data, Y))
# Subset and obtain the unconditional model output
Y_unc <- dt[index, Y]
# Create the intervals
melted <- data.table::melt(dt,
                        measure.vars = 1:(ncol(dt) - 1),
                        variable.name = "parameters")
# Compute PAWN indices
out <- melted[order(parameters, value)][
  , .(chunks(Y, n)), parameters][
  , Y_unc:= .(rep(. (Y_unc), times = n * ncol(data)))][
  , ID:= .I][
  , results:= .(.(mapply(stats::ks.test, Y_unc, V1))), ID][
  , statistic:= sapply(results, function(x) x[, 1]$statistic)]
if(theta == 1) {
  final <- out[, mean(statistic), parameters][, V1]
} else if(theta == 2) {
  final <- out[, median(statistic), parameters][, V1]
} else {
  final <- out[, max(statistic), parameters][, V1]
}
return(final)
}

```

## 3.2 Settings

```

# DEFINE SETTINGS -----

```

```

# Set sample size
n <- 2 ^ 13

# Define N.min and N.max
N.min <- 200
N.max <- 2000

# Set parameters
parameters <- c("N", "n", "epsilon", "theta")

# Vector with name of functions
models <- c("Liu", "Ishigami", "Sobol' G", "Morris")

```

### 3.3 Sample matrix

```

# CREATION OF THE MATRICES -----

# Create the A, B and AB matrices, also for the
# computation of second and third-order indices
tmp <- lapply(1:4, function(x)
  sobol_matrices(n = n,
    k = length(parameters),
    second = TRUE,
    third = TRUE) %>%
  data.table())

# Name the slots
names(tmp) <- 1:4

# Rename columns
tmp <- lapply(tmp, setnames, parameters) %>%
  rbindlist(., idcol = "Model")

# Create two copies of the sample matrix and list the
# original and the copies. One would be to run the
# calculations in the max in theta setting; the
# other one for the max not in theta setting,
# and the other in the optimum setting
max <- copy(tmp)
A <- list(tmp, max, copy(tmp))

# Name the slots
names(A) <- c("max", "no.max", "optimum")

# Transform all distributions
for(i in names(A)) {
  if(i == "max") {

```

```

# where 1=mean, 2=median, 3=max in the model
A[[i]][, N:= floor(qunif(N, N.min, N.max))]
A[[i]][, n:= floor(qunif(n, 5, 20))]
A[[i]][, theta:= floor(theta * (3 - 1 + 1)) + 1]
} else if(i == "no.max") {
  A[[i]][, N:= floor(qunif(N, N.min, N.max))]
  A[[i]][, n:= floor(qunif(n, 5, 20))]
  A[[i]][, theta:= floor(theta * (2 - 1 + 1)) + 1]
} else {
  A[[i]][, N:= floor(qunif(N, N.max, 4000))]
  A[[i]][, n:= floor(qunif(n, 15, 20))]
  A[[i]][, theta:= floor(theta * (2 - 1 + 1)) + 1]
}
}

# Transform all the other distributions
A.pawn <- rbindlist(A, idcol = "setting")[
  , epsilon:= floor(qunif(epsilon, 1, 1000))][
  , Model:= as.numeric(Model)]

print(A.pawn)

```

```

##          setting Model      N  n epsilon theta
##          1:      max      1 1100 12      500      2
##          2:      max      1 1550  8      750      1
##          3:      max      1  650 16      250      3
##          4:      max      1  875 10      625      1
##          5:      max      1 1775 18      125      2
##          ---
## 1572860: optimum      4 2500 18      508      2
## 1572861: optimum      4 3500 16          9      1
## 1572862: optimum      4 3000 19      258      2
## 1572863: optimum      4 2000 17      758      1
## 1572864: optimum      4 2000 19      889      2

```

### 3.4 Run the model

```

# RUN MODEL -----

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

# Compute
Y.pawn <- foreach(i=1:nrow(A.pawn),
  .packages = "data.table") %dopar%
{

```

```

    model_pawn(epsilon = A.pawn[[i, "epsilon"]],
               N = A.pawn[[i, "N"]],
               n = A.pawn[[i, "n"]],
               theta = A.pawn[[i, "theta"]],
               Model = A.pawn[[i, "Model"]])
  }

# Stop parallel cluster
stopCluster(cl)

# EXTRACT DATA -----

rowNumber <- lapply(1:4, function(x) A.pawn[, .I[Model == x]])
names(rowNumber) <- models

out <- list()
for(i in models) {
  out[[i]] <- Y.pawn[rowNumber[[i]]]
}

dt.models <- list()
for(i in seq_along(1:4)) {
  dt.models[[i]] <- cbind(A.pawn[Model == i], data.table(do.call(rbind, out[[i]])))
}

```

### 3.5 Uncertainty analysis

```

# DATASET FOR UNCERTAINTY ANALYSIS -----

AB.pawn <- lapply(dt.models, function(x) {
  x[, .SD[1: (2 * (2 ^ 13))], setting] %>%
    melt(., measure.vars = patterns("V"),
         variable.name = "parameter")
}) %>%
rbindlist() %>%
.[, Model:= ifelse(Model == 1, models[1],
                   ifelse(Model == 2, models[2],
                           ifelse(Model == 3, models[3], models[4])))] %>%
.[, parameter:= gsub("V", "X", parameter)] %>%
.[, parameter:= factor(parameter,
                       levels = paste("X", 1:20, sep = ""))] %>%
.[, Model:= factor(Model,
                   levels = c("Liu", "Ishigami", "Sobol' G", "Morris"))] %>%
.[, setting:= ifelse(setting == "max", "$max \\in \\theta$",
                     ifelse(setting == "no.max", "$max \\notin \\theta$", "Optimum"))]

```

```

# CHECK OVERLAP -----

overlap.dt <- split(AB.pawn, AB.pawn$setting)

overlap.results <- mclapply(overlap.dt, function(x) {
  split(x, x$Model, drop = TRUE) %>%
    lapply(., function(x) split(x, x$parameter, drop = TRUE)) %>%
    lapply(., function(x) lapply(x, function(y) y[, value])) %>%
    lapply(., function(x) overlap(x))},
  mc.cores = n_cores)

tmp <- lapply(overlap.results, function(x) lapply(x, function(y) {
  cbind(y$OV) %>%
    data.frame() %>%
    setDT(., keep.rownames = TRUE)
})))

pawn.overlap.results <- lapply(tmp, function(x)
  rbindlist(x, idcol = "Model") %>%
  rbindlist(., idcol = "setting") %>%
  setnames(., ".", "overlap"))

par.overlap <- paste("X", 1:6, sep = "")

final.overlap <- lapply(models, function(x) pawn.overlap.results[Model==x, .SD, setting]) %>%
  lapply(., function(x) x[, "overlap" := round(.SD, 3), .SDcols = "overlap"])

final.overlap

## [[1]]
##           setting Model    rn overlap
## 1:    $max \\in \\theta$ Liu X1-X2  0.253
## 2:    $max \\notin \\theta$ Liu X1-X2  0.127
## 3:           Optimum   Liu X1-X2  0.013
##
## [[2]]
##           setting   Model    rn overlap
## 1:    $max \\in \\theta$ Ishigami X1-X2  0.009
## 2:    $max \\in \\theta$ Ishigami X1-X3  0.051
## 3:    $max \\in \\theta$ Ishigami X2-X3  0.095
## 4:    $max \\notin \\theta$ Ishigami X1-X2  0.001
## 5:    $max \\notin \\theta$ Ishigami X1-X3  0.016
## 6:    $max \\notin \\theta$ Ishigami X2-X3  0.038
## 7:           Optimum Ishigami X1-X2  0.000
## 8:           Optimum Ishigami X1-X3  0.000
## 9:           Optimum Ishigami X2-X3  0.000
##

```

```
## [[3]]
##          setting      Model      rn overlap
##  1:      $max \\in \\theta$ Sobol' G X1-X2   0.101
##  2:      $max \\in \\theta$ Sobol' G X1-X3   0.010
##  3:      $max \\in \\theta$ Sobol' G X1-X4   0.011
##  4:      $max \\in \\theta$ Sobol' G X1-X5   0.006
##  5:      $max \\in \\theta$ Sobol' G X1-X6   0.006
##  6:      $max \\in \\theta$ Sobol' G X1-X7   0.006
##  7:      $max \\in \\theta$ Sobol' G X1-X8   0.009
##  8:      $max \\in \\theta$ Sobol' G X2-X3   0.192
##  9:      $max \\in \\theta$ Sobol' G X2-X4   0.138
## 10:      $max \\in \\theta$ Sobol' G X2-X5   0.108
## 11:      $max \\in \\theta$ Sobol' G X2-X6   0.107
## 12:      $max \\in \\theta$ Sobol' G X2-X7   0.100
## 13:      $max \\in \\theta$ Sobol' G X2-X8   0.103
## 14:      $max \\in \\theta$ Sobol' G X3-X4   0.702
## 15:      $max \\in \\theta$ Sobol' G X3-X5   0.592
## 16:      $max \\in \\theta$ Sobol' G X3-X6   0.568
## 17:      $max \\in \\theta$ Sobol' G X3-X7   0.544
## 18:      $max \\in \\theta$ Sobol' G X3-X8   0.544
## 19:      $max \\in \\theta$ Sobol' G X4-X5   0.839
## 20:      $max \\in \\theta$ Sobol' G X4-X6   0.808
## 21:      $max \\in \\theta$ Sobol' G X4-X7   0.779
## 22:      $max \\in \\theta$ Sobol' G X4-X8   0.777
## 23:      $max \\in \\theta$ Sobol' G X5-X6   0.952
## 24:      $max \\in \\theta$ Sobol' G X5-X7   0.920
## 25:      $max \\in \\theta$ Sobol' G X5-X8   0.913
## 26:      $max \\in \\theta$ Sobol' G X6-X7   0.951
## 27:      $max \\in \\theta$ Sobol' G X6-X8   0.950
## 28:      $max \\in \\theta$ Sobol' G X7-X8   0.961
## 29: $max \\notin \\theta$ Sobol' G X1-X2   0.003
## 30: $max \\notin \\theta$ Sobol' G X1-X3   0.001
## 31: $max \\notin \\theta$ Sobol' G X1-X4   0.002
## 32: $max \\notin \\theta$ Sobol' G X1-X5   0.001
## 33: $max \\notin \\theta$ Sobol' G X1-X6   0.001
## 34: $max \\notin \\theta$ Sobol' G X1-X7   0.001
## 35: $max \\notin \\theta$ Sobol' G X1-X8   0.002
## 36: $max \\notin \\theta$ Sobol' G X2-X3   0.095
## 37: $max \\notin \\theta$ Sobol' G X2-X4   0.089
## 38: $max \\notin \\theta$ Sobol' G X2-X5   0.076
## 39: $max \\notin \\theta$ Sobol' G X2-X6   0.078
## 40: $max \\notin \\theta$ Sobol' G X2-X7   0.075
## 41: $max \\notin \\theta$ Sobol' G X2-X8   0.076
## 42: $max \\notin \\theta$ Sobol' G X3-X4   0.648
## 43: $max \\notin \\theta$ Sobol' G X3-X5   0.569
## 44: $max \\notin \\theta$ Sobol' G X3-X6   0.548
## 45: $max \\notin \\theta$ Sobol' G X3-X7   0.527
## 46: $max \\notin \\theta$ Sobol' G X3-X8   0.525
```



```

## 47: $max \\notin \\theta$ Sobol' G X4-X5 0.852
## 48: $max \\notin \\theta$ Sobol' G X4-X6 0.821
## 49: $max \\notin \\theta$ Sobol' G X4-X7 0.797
## 50: $max \\notin \\theta$ Sobol' G X4-X8 0.793
## 51: $max \\notin \\theta$ Sobol' G X5-X6 0.952
## 52: $max \\notin \\theta$ Sobol' G X5-X7 0.927
## 53: $max \\notin \\theta$ Sobol' G X5-X8 0.919
## 54: $max \\notin \\theta$ Sobol' G X6-X7 0.956
## 55: $max \\notin \\theta$ Sobol' G X6-X8 0.953
## 56: $max \\notin \\theta$ Sobol' G X7-X8 0.958
## 57: Optimum Sobol' G X1-X2 0.000
## 58: Optimum Sobol' G X1-X3 0.000
## 59: Optimum Sobol' G X1-X4 0.000
## 60: Optimum Sobol' G X1-X5 0.000
## 61: Optimum Sobol' G X1-X6 0.000
## 62: Optimum Sobol' G X1-X7 0.000
## 63: Optimum Sobol' G X1-X8 0.000
## 64: Optimum Sobol' G X2-X3 0.002
## 65: Optimum Sobol' G X2-X4 0.001
## 66: Optimum Sobol' G X2-X5 0.001
## 67: Optimum Sobol' G X2-X6 0.001
## 68: Optimum Sobol' G X2-X7 0.001
## 69: Optimum Sobol' G X2-X8 0.001
## 70: Optimum Sobol' G X3-X4 0.276
## 71: Optimum Sobol' G X3-X5 0.191
## 72: Optimum Sobol' G X3-X6 0.183
## 73: Optimum Sobol' G X3-X7 0.184
## 74: Optimum Sobol' G X3-X8 0.174
## 75: Optimum Sobol' G X4-X5 0.542
## 76: Optimum Sobol' G X4-X6 0.523
## 77: Optimum Sobol' G X4-X7 0.516
## 78: Optimum Sobol' G X4-X8 0.490
## 79: Optimum Sobol' G X5-X6 0.957
## 80: Optimum Sobol' G X5-X7 0.947
## 81: Optimum Sobol' G X5-X8 0.907
## 82: Optimum Sobol' G X6-X7 0.953
## 83: Optimum Sobol' G X6-X8 0.919
## 84: Optimum Sobol' G X7-X8 0.954
##      setting      Model      rn overlap
##
## [[4]]
##      setting      Model      rn overlap
## 1: $max \\in \\theta$ Morris X1-X2 0.907
## 2: $max \\in \\theta$ Morris X1-X3 0.940
## 3: $max \\in \\theta$ Morris X1-X4 0.890
## 4: $max \\in \\theta$ Morris X1-X5 0.887
## 5: $max \\in \\theta$ Morris X1-X6 0.814
## ---

```

```
## 566:          Optimum Morris X17-X19    0.884
## 567:          Optimum Morris X17-X20    0.971
## 568:          Optimum Morris X18-X19    0.756
## 569:          Optimum Morris X18-X20    0.680
## 570:          Optimum Morris X19-X20    0.890
```

```
# Export results
```

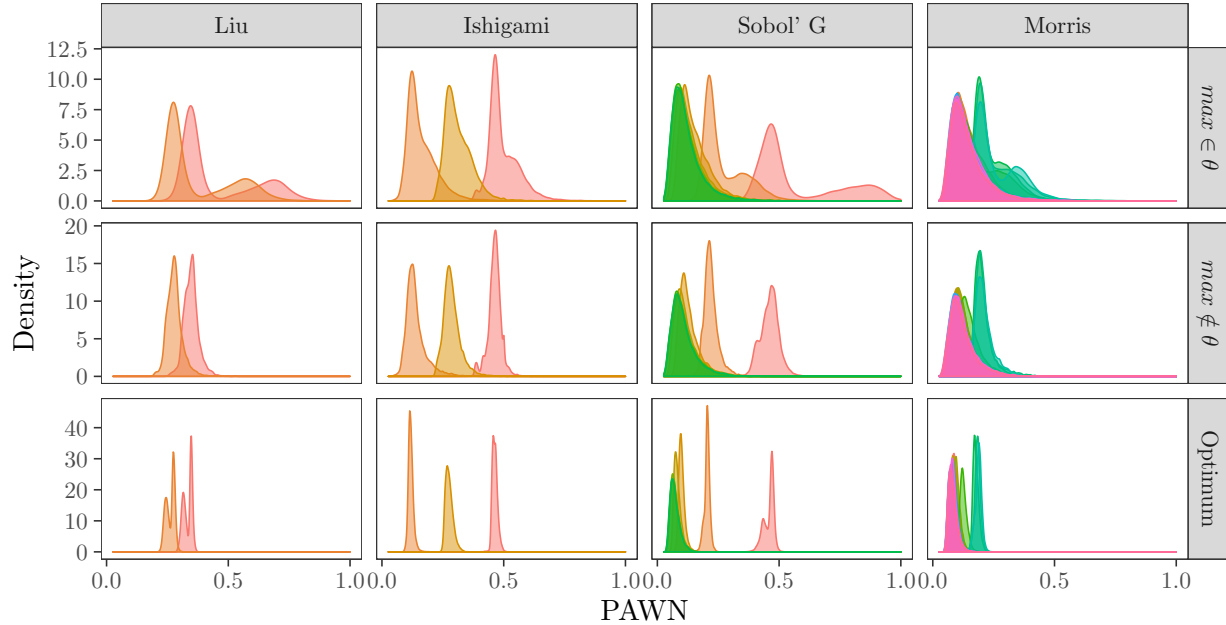
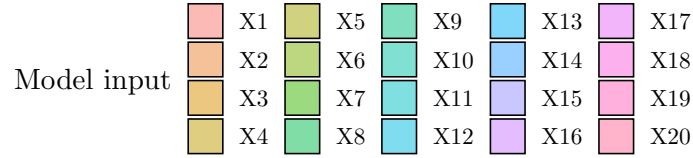
```
rbindlist(final.overlap) %>%
  fwrite(., "pawn.overlap.csv")
```

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

```
plot.uncertainty.pawn <- ggplot(AB.pawn, aes(value,
                                              fill = parameter,
                                              color = parameter)) +

  geom_density(alpha = 0.5,
               position = "identity") +
  facet_grid(setting~Model,
             scales = "free_y") +
  scale_fill_discrete(name = "Model input") +
  scale_color_discrete(guide = FALSE) +
  labs(x = "PAWN",
       y = "Density") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  theme_bw() +
  theme(legend.position = "top",
        legend.box = "horizontal",
        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))

plot.uncertainty.pawn
```



```
# EXPORT AB MATRIX FOR PAWN -----
```

```
fwrite(AB.pawn, "AB.pawn.csv")
```

### 3.6 Sensitivity analysis

```
# DATASET FOR SENSITIVITY ANALYSIS -----
```

```
dt.pawn.sens <- lapply(dt.models, function(x)
  melt(x, measure.vars = patterns("V"), variable.name = "model.input")) %>%
  rbindlist() %>%
  .[, Model:= ifelse(Model == 1, models[1],
    ifelse(Model == 2, models[2],
      ifelse(Model == 3, models[3], models[4])))] %>%
  .[, model.input:= gsub("V", "X", model.input)] %>%
  .[, model.input:= factor(model.input,
    levels = paste("X", 1:20, sep = ""))] %>%
  .[, Model:= factor(Model,
    levels = c("Liu", "Ishigami", "Sobol' G", "Morris"))] %>%
  setnames(., "value", "Y")
```

```
# EXPORT AB MATRIX FOR SENSITIVITY -----
```

```
fwrite(dt.pawn.sens, "dt.pawn.sens.csv")
```

```

# SENSITIVITY ANALYSIS -----

pawn.sensitivity <- dt.pawn.sens[, sobol_indices(Y,
                                                params = parameters,
                                                R = R,
                                                n = 2 ^ 13,
                                                parallel = "multicore",
                                                second = TRUE,
                                                third = TRUE,
                                                ncpus = n_cores),
                              .(setting, Model, model.input)]

# CONFIDENCE INTERVALS -----

# Arrange data
tmp3 <- split(pawn.sensitivity, pawn.sensitivity$setting) %>%
  lapply(., function(x) split(x, x$Model)) %>%
  lapply(., function(x) lapply(x, function(y) split(y, y$model.input, drop = TRUE)))

# Compute confidence intervals
pawn.ci <- list()
for(i in names(tmp3)) {
  for(j in names(tmp3[[i]])) {
    for(k in names(tmp3[[i]][[j]])) {
      pawn.ci[[i]][[j]][[k]] <- sobol_ci(tmp3[[i]][[j]][[k]],
                                          params = parameters,
                                          type = type,
                                          conf = conf,
                                          second = TRUE,
                                          third = TRUE)
    }
  }
}

# Rearrange data
final.pawn.ci <- lapply(pawn.ci, function(x)
  lapply(x, function(y) rbindlist(y, idcol = "model.input"))) %>%
  lapply(., function(x) rbindlist(x, idcol = "model")) %>%
  rbindlist(., idcol = "setting") %>%
  .[, model:= factor(model, levels = c("Liu", "Ishigami",
                                       "Sobol' G", "Morris"))] %>%
  .[, model.input:= factor(model.input, levels = paste("X", 1:20, sep = ""))] %>%
  .[, parameters:= gsub("epsilon", "$\\\\\\varepsilon$", parameters)] %>%
  .[, parameters:= gsub("theta", "$\\\\\\theta$", parameters)] %>%
  .[, setting:= ifelse(setting == "max", "$max \\in \\theta$",
                      ifelse(setting == "no.max", "$max \\notin \\theta$", "Optimum"))]

```

```

# EXPORT DATA -----

fwrite(final.pawn.ci, "final.pawn.ci.csv")

# PLOT AGGREGATED SOBOL' INDICES -----

a <- final.pawn.ci[sensitivity == "Si" | sensitivity == "STi"] %>%
  ggplot(., aes(parameters, original,
                fill = sensitivity)) +
  geom_boxplot(outlier.size = 0.2) +
  labs(x = "",
       y = "Sobol' index") +
  scale_fill_discrete(name = "Sobol' indices",
                     labels = c(expression(S[italic(i)]),
                                expression(S[italic(T[i])])))) +
  theme_bw() +
  facet_wrap(~ setting) +
  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),
        legend.position = "none")

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

# PLOT SUM OF SOBOL' SI -----

b <- final.pawn.ci[sensitivity == "Si"] [
  , sum(original), .(setting, model, model.input)] %>%
  ggplot(., aes(setting, V1)) +
  geom_boxplot(outlier.size = 0.2) +
  labs(x = "",
       y = expression(paste("Sum of"-S[i]))) +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())

# MERGE AGGREGATED SOBOL' AND SUM OF SOBOL' -----

up <- plot_grid(legend, NULL,
                ncol = 2)

bottom <- plot_grid(a, b,
                   ncol = 2,
                   align = "hv",

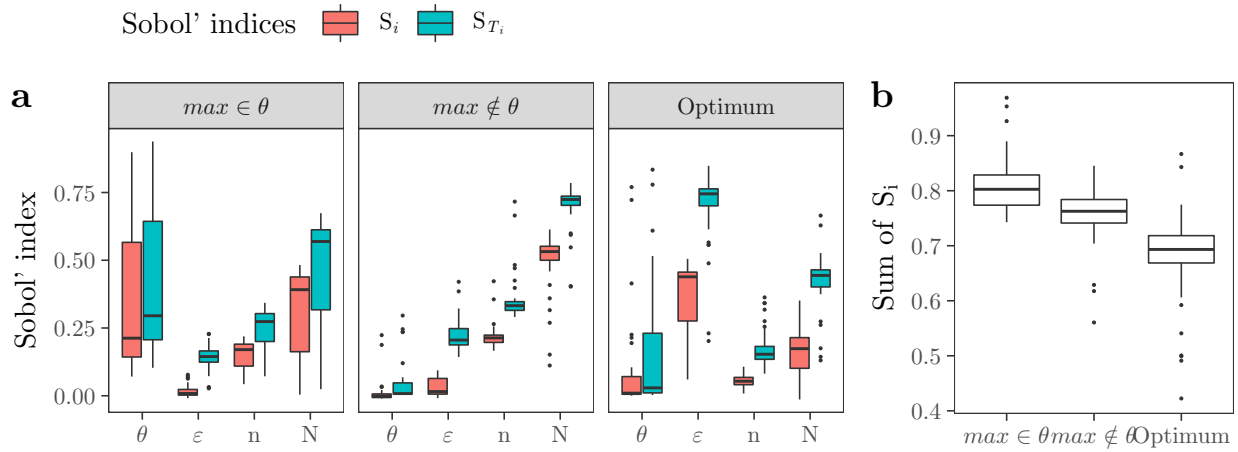
```

```
labels = "auto",
rel_widths = c(2.2, 1))
```

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

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

```
plot_grid(up, bottom,
          ncol = 1,
          align = "hv",
          rel_heights = c(0.21, 1))
```



## 4 Sensitivity of Sobol' indices to its design parameters

### 4.1 The model

```
# THE MODEL -----

# Functions to create A and AB matrices to compute Ti
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)
}

sobol_matrix <- function(n, k) {
  df <- randtoolbox::sobol(n = n, dim = k * 2)
  A <- df[, 1:k]
```

```

B <- df[, (k + 1) : (k * 2)]
out <- scrambled_sobol(A = A, B = B)
return(out)
}

# Functions to estimate Ti
sobol_all <- function(Y_A, Y_AB, type) {
  n <- length(Y_A[!is.na(Y_A)])
  f0 <- (1 / n) * sum(Y_A)
  VY <- 1 / n * sum((Y_A - f0) ^ 2)
  if(type == "jansen") {
    STi <- ((1 / (2 * n)) * sum((Y_A - Y_AB) ^ 2)) / VY
  } else if(type == "homma") {
    STi <- (VY - (1 / n) * sum(Y_A * Y_AB) + f0 ^ 2) / VY
  } else if(type == "sobol") {
    STi <- ((1 / n) * sum(Y_A * (Y_A - Y_AB))) / VY
  } else {
    stop("type should be either jansen, sobol or homma")
  }
  return(STi)
}

sobol_Ti_Mapply <- function(d, type) {
  return(mapply(sobol_all,
                MoreArgs = list(type = type),
                d[, "Y_A"],
                d[, "Y_AB"]))
}

sobol_Ti <- function(Y, params, type) {
  k <- length(params)
  p <- length(1:(length(Y) / (k + 1)))
  Y_A <- Y[1:p]
  Y_AB <- Y[(p + 1):length(Y)]
  parameters <- rep(params, each = length(Y_A))
  vec <- cbind(Y_A, Y_AB)
  out <- data.table(vec, parameters)
  output <- out[, sobol_Ti_Mapply(.SD, type = type), parameters][, V1]
  return(output)
}

# The model
model_sobol <- function(Model, N, k, Theta) {
  data <- sobol_matrix(n = N, k = k)
  if(Model == 1) {
    Y <- liu_Mapply(data)
  } else if(Model == 2) {

```

```

    Y <- sensobol::ishigami_Mapply(data)
  } else if(Model == 3) {
    Y <- sensobol::sobol_Fun(data)
  } else {
    Y <- sensitivity::morris.fun(data)
  }
  out <- sobol_Ti(Y, params = paste("X", 1:k, sep = ""), type = Theta)
  return(out)
}

```

## 4.2 Settings

```

# DEFINE SETTINGS -----

# Set parameters
parameters.sobol <- c("N", "Theta")

```

## 4.3 Sample matrix

```

# CREATION OF THE MATRICES -----

# Create the A and AB matrices, also for the
# computation of second and third-order indices
tmp <- lapply(models, function(x)
  sobol_matrices(n = n, k = length(parameters.sobol)) %>%
  data.table())

# Rename columns and transform distributions
A <- lapply(tmp, setnames, parameters.sobol) %>%
  rbindlist(., idcol = "Model")

# Create two copies of the sample matrix and list the
# original and the copies. One would be to run the
# calculations with uncertainty in N and Theta,
# the other with uncertainty in N only.
N.only <- copy(A)
A.DT <- list(A, N.only)
names(A.DT) <- c("N.Theta", "N")

A <- rbindlist(A.DT, idcol = "setting")

A.sobol <- A[, k:= ifelse(Model == 1, 2, ifelse(Model == 2, 3, ifelse(Model == 3, 8, 20)))] [
  , N:= floor(qunif(N, N.min, N.max))] [
  , Model:= as.numeric(Model)] [
  , Theta:= floor(Theta * (3 - 1 + 1)) + 1] [
  , Theta:= ifelse(Theta == 1, "jansen", ifelse(Theta == 2, "honna", "sobol"))] [
  , Theta:= ifelse(setting == "N", "jansen", Theta)]

```



```
print(A.sobol)
```

```
##          setting Model      N  Theta  k
##      1: N.Theta        1 1100  homma  2
##      2: N.Theta        1 1550  jansen  2
##      3: N.Theta        1  650  sobol  2
##      4: N.Theta        1  875  homma  2
##      5: N.Theta        1 1775  sobol  2
##      ---
## 262140:      N      4  650  jansen 20
## 262141:      N      4 1550  jansen 20
## 262142:      N      4 1100  jansen 20
## 262143:      N      4  200  jansen 20
## 262144:      N      4  200  jansen 20
```

```
print(n)
```

```
## [1] 8192
```

#### 4.4 Run the model

```
# RUN MODEL -----

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

# Compute
Y.sobol <- foreach(i=1:nrow(A.sobol),
                   .packages = "data.table") %dopar%
{
  model_sobol(N = A.sobol[[i, "N"]],
             Theta = A.sobol[[i, "Theta"]],
             Model = A.sobol[[i, "Model"]],
             k = A.sobol[[i, "k"]])
}

# Stop parallel cluster
stopCluster(cl)

# EXTRACT MODEL OUTPUT -----

rowNumber <- lapply(1:4, function(x) A.sobol[, .I[Model == x]])
names(rowNumber) <- models

out <- list()
for(i in models) {
  out[[i]] <- Y.sobol[rowNumber[[i]]]
```

```

}

dt.models <- list()
for(i in seq_along(1:4)) {
  dt.models[[i]] <- cbind(A[Model == i], data.table(do.call(rbind, out[[i]])))
}

```

## 4.5 Uncertainty analysis

```

# DATASET FOR UNCERTAINTY ANALYSIS -----

AB.sobol <- lapply(dt.models, function(x) {
  x[, .SD[1: (2 * (2 ^ 13))], setting] %>%
  melt(., measure.vars = patterns("V"),
       variable.name = "parameter") %>%
  rbindlist(.) %>%
  .[, Model:= ifelse(Model == 1, models[1],
                    ifelse(Model == 2, models[2],
                          ifelse(Model == 3, models[3], models[4])))] %>%
  .[, k:= NULL] %>%
  .[, parameter:= gsub("V", "X", parameter)] %>%
  .[, parameter:= factor(parameter,
                        levels = paste("X", 1:20, sep = ""))] %>%
  .[, Model:= factor(Model,
                    levels = c("Liu", "Ishigami", "Sobol' G", "Morris"))] %>%
  .[, setting:= ifelse(setting == "N.Theta", "$N,\\theta$", setting)]

# EXPORT AB MATRIX FOR SOBOL' -----

fwrite(AB.sobol, "AB.sobol.csv")

# PLOT UNCERTAINTY -----

AB.sobol %>%
  ggplot(., aes(value,
               fill = parameter,
               color = parameter)) +
  geom_density(alpha = 0.5,
               position = "identity") +
  facet_grid(setting~Model) +
  scale_fill_discrete(name = "Model input") +
  scale_color_discrete(guide = FALSE) +
  labs(x = "Y",
       y = "Count") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  scale_y_continuous(limits = c(0, 35)) +
  theme_bw() +

```

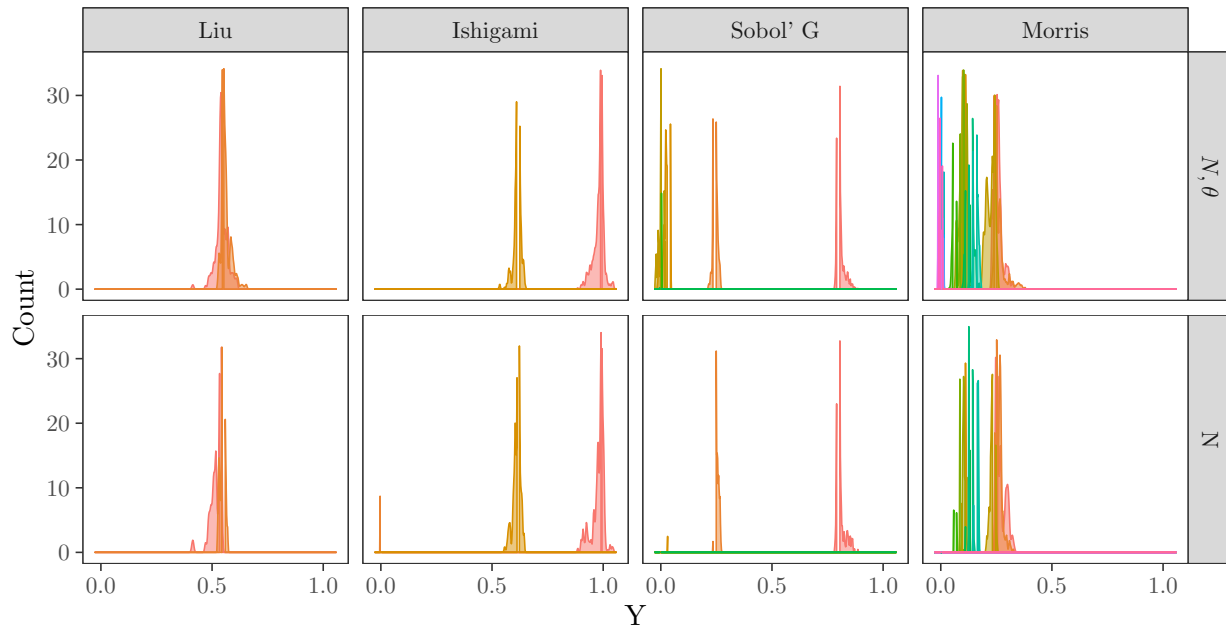
```

theme(legend.position = "top",
      legend.box = "horizontal",
      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))

```

Model input

<span style="display:inline-block; width:10px; height:10px; background-color: #f08080; border:1px solid black;"></span> X1	<span style="display:inline-block; width:10px; height:10px; background-color: #b8860b; border:1px solid black;"></span> X5	<span style="display:inline-block; width:10px; height:10px; background-color: #3cb371; border:1px solid black;"></span> X9	<span style="display:inline-block; width:10px; height:10px; background-color: #1e90ff; border:1px solid black;"></span> X13	<span style="display:inline-block; width:10px; height:10px; background-color: #e619ff; border:1px solid black;"></span> X17
<span style="display:inline-block; width:10px; height:10px; background-color: #f4a460; border:1px solid black;"></span> X2	<span style="display:inline-block; width:10px; height:10px; background-color: #8db600; border:1px solid black;"></span> X6	<span style="display:inline-block; width:10px; height:10px; background-color: #40e0d0; border:1px solid black;"></span> X10	<span style="display:inline-block; width:10px; height:10px; background-color: #6495ed; border:1px solid black;"></span> X14	<span style="display:inline-block; width:10px; height:10px; background-color: #ff69b4; border:1px solid black;"></span> X18
<span style="display:inline-block; width:10px; height:10px; background-color: #d2b48c; border:1px solid black;"></span> X3	<span style="display:inline-block; width:10px; height:10px; background-color: #32cd32; border:1px solid black;"></span> X7	<span style="display:inline-block; width:10px; height:10px; background-color: #00ced1; border:1px solid black;"></span> X11	<span style="display:inline-block; width:10px; height:10px; background-color: #9370db; border:1px solid black;"></span> X15	<span style="display:inline-block; width:10px; height:10px; background-color: #ff69b4; border:1px solid black;"></span> X19
<span style="display:inline-block; width:10px; height:10px; background-color: #d2b48c; border:1px solid black;"></span> X4	<span style="display:inline-block; width:10px; height:10px; background-color: #3cb371; border:1px solid black;"></span> X8	<span style="display:inline-block; width:10px; height:10px; background-color: #1e90ff; border:1px solid black;"></span> X12	<span style="display:inline-block; width:10px; height:10px; background-color: #9370db; border:1px solid black;"></span> X16	<span style="display:inline-block; width:10px; height:10px; background-color: #ff69b4; border:1px solid black;"></span> X20



```

# CHECK OVERLAP -----

overlap.dt <- split(AB.sobol, AB.sobol$setting)

overlap.results <- mclapply(overlap.dt, function(x) {
  split(x, x$Model, drop = TRUE) %>%
    lapply(., function(x) split(x, x$parameter, drop = TRUE)) %>%
    lapply(., function(x) lapply(x, function(y) y[, value])) %>%
    lapply(., function(x) overlap(x))},
  mc.cores = n_cores)

tmp <- lapply(overlap.results, function(x) lapply(x, function(y) {
  cbind(y$OV) %>%
    data.frame() %>%
    setDT(., keep.rownames = TRUE)

```

```
}))
```

```
sobol.overlap.results <- lapply(tmp, function(x)
  rbindlist(x, idcol = "Model")) %>%
  rbindlist(., idcol = "setting") %>%
  setnames(., ".", "overlap")
```

```
par.overlap <- paste("X", 1:6, sep = "")
```

```
final.overlap <- lapply(models, function(x) sobol.overlap.results[Model==x, .SD, setting]) %>%
  lapply(., function(x) x[, "overlap"]:= round(.SD, 3), .SDcols = "overlap")
```

```
final.overlap
```

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

```
##      setting Model      rn overlap
## 1: $N,\\theta$ Liu X1-X2  0.340
## 2:           N   Liu X1-X2  0.185
```

```
##
```

```
## [[2]]
```

```
##      setting      Model      rn overlap
## 1: $N,\\theta$ Ishigami X1-X2      0
## 2: $N,\\theta$ Ishigami X1-X3      0
## 3: $N,\\theta$ Ishigami X2-X3      0
## 4:           N Ishigami X1-X2      0
## 5:           N Ishigami X1-X3      0
## 6:           N Ishigami X2-X3      0
```

```
##
```

```
## [[3]]
```

```
##      setting      Model      rn overlap
## 1: $N,\\theta$ Sobol' G X1-X2  0.000
## 2: $N,\\theta$ Sobol' G X1-X3  0.000
## 3: $N,\\theta$ Sobol' G X1-X4  0.000
## 4: $N,\\theta$ Sobol' G X1-X5  0.000
## 5: $N,\\theta$ Sobol' G X1-X6  0.000
## 6: $N,\\theta$ Sobol' G X1-X7  0.000
## 7: $N,\\theta$ Sobol' G X1-X8  0.000
## 8: $N,\\theta$ Sobol' G X2-X3  0.000
## 9: $N,\\theta$ Sobol' G X2-X4  0.000
## 10: $N,\\theta$ Sobol' G X2-X5  0.000
## 11: $N,\\theta$ Sobol' G X2-X6  0.000
## 12: $N,\\theta$ Sobol' G X2-X7  0.000
## 13: $N,\\theta$ Sobol' G X2-X8  0.000
## 14: $N,\\theta$ Sobol' G X3-X4  0.033
## 15: $N,\\theta$ Sobol' G X3-X5  0.001
## 16: $N,\\theta$ Sobol' G X3-X6  0.001
## 17: $N,\\theta$ Sobol' G X3-X7  0.001
```

```

## 18: $N,\\theta$ Sobol' G X3-X8 0.001
## 19: $N,\\theta$ Sobol' G X4-X5 0.025
## 20: $N,\\theta$ Sobol' G X4-X6 0.036
## 21: $N,\\theta$ Sobol' G X4-X7 0.026
## 22: $N,\\theta$ Sobol' G X4-X8 0.026
## 23: $N,\\theta$ Sobol' G X5-X6 0.779
## 24: $N,\\theta$ Sobol' G X5-X7 0.209
## 25: $N,\\theta$ Sobol' G X5-X8 0.666
## 26: $N,\\theta$ Sobol' G X6-X7 0.258
## 27: $N,\\theta$ Sobol' G X6-X8 0.588
## 28: $N,\\theta$ Sobol' G X7-X8 0.224
## 29:      N Sobol' G X1-X2 0.000
## 30:      N Sobol' G X1-X3 0.000
## 31:      N Sobol' G X1-X4 0.000
## 32:      N Sobol' G X1-X5 0.000
## 33:      N Sobol' G X1-X6 0.000
## 34:      N Sobol' G X1-X7 0.000
## 35:      N Sobol' G X1-X8 0.000
## 36:      N Sobol' G X2-X3 0.000
## 37:      N Sobol' G X2-X4 0.000
## 38:      N Sobol' G X2-X5 0.000
## 39:      N Sobol' G X2-X6 0.000
## 40:      N Sobol' G X2-X7 0.000
## 41:      N Sobol' G X2-X8 0.000
## 42:      N Sobol' G X3-X4 0.000
## 43:      N Sobol' G X3-X5 0.000
## 44:      N Sobol' G X3-X6 0.000
## 45:      N Sobol' G X3-X7 0.000
## 46:      N Sobol' G X3-X8 0.000
## 47:      N Sobol' G X4-X5 0.000
## 48:      N Sobol' G X4-X6 0.000
## 49:      N Sobol' G X4-X7 0.000
## 50:      N Sobol' G X4-X8 0.000
## 51:      N Sobol' G X5-X6 0.372
## 52:      N Sobol' G X5-X7 0.603
## 53:      N Sobol' G X5-X8 0.678
## 54:      N Sobol' G X6-X7 0.269
## 55:      N Sobol' G X6-X8 0.264
## 56:      N Sobol' G X7-X8 0.684
##      setting      Model      rn overlap
##
## [[4]]
##      setting      Model      rn overlap
## 1: $N,\\theta$ Morris      X1-X2      0.447
## 2: $N,\\theta$ Morris      X1-X3      0.000
## 3: $N,\\theta$ Morris      X1-X4      0.291
## 4: $N,\\theta$ Morris      X1-X5      0.000
## 5: $N,\\theta$ Morris      X1-X6      0.000

```

```
## ---
## 376:      N Morris X17-X19    0.060
## 377:      N Morris X17-X20    0.000
## 378:      N Morris X18-X19    0.026
## 379:      N Morris X18-X20    0.000
## 380:      N Morris X19-X20    0.000
```

```
# Export results
rbindlist(final.overlap) %>%
  fwrite(., "sobol.overlap.csv")
```

## 4.6 Sensitivity analysis

```
# DATASET FOR SENSITIVITY ANALYSIS -----
```

```
full.dataset.sobol <- lapply(dt.models, function(x)
  melt(x, measure.vars = patterns("V"),
    variable.name = "parameter")) %>%
  rbindlist(.) %>%
  .[, Model:= ifelse(Model == 1, models[1],
    ifelse(Model == 2, models[2],
      ifelse(Model == 3, models[3], models[4])))] %>%
  .[, k:= NULL] %>%
  .[, parameter:= gsub("V", "X", parameter)] %>%
  .[, parameter:= factor(parameter,
    levels = paste("X", 1:20, sep = ""))] %>%
  .[, Model:= factor(Model,
    levels = c("Liu", "Ishigami", "Sobol' G", "Morris"))] %>%
  .[, setting:= ifelse(setting == "N.Theta", "$N,\\theta$", setting)]
```

```
# EXPORT SENSITIVITY MATRIX -----
```

```
fwrite(full.dataset.sobol, "full.dataset.sobol.csv")
```

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

```
sobol.sensitivity <- full.dataset.sobol[, sobol_indices(value,
  type = "jansen",
  params = parameters.sobol,
  n = 2 ^ 13,
  R = R,
  parallel = "multicore",
  ncpus = n_cores),
.(Model, parameter, setting)]
```

```
# CONFIDENCE INTERVALS -----
```

```
# Arrange data
```

[illegible]

```
## [1] "All values of t are equal to 1 \n Cannot calculate confidence intervals"
## [1] "All values of t are equal to 1 \n Cannot calculate confidence intervals"
## [1] "All values of t are equal to 1 \n Cannot calculate confidence intervals"

# ARRANGE DATA -----

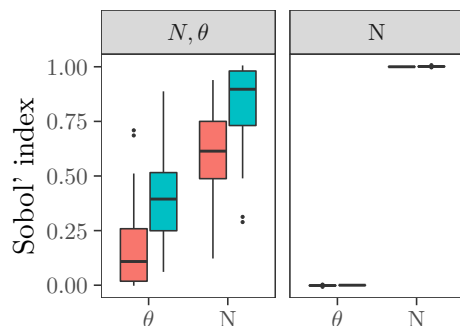
final.sobol <- lapply(out, function(x)
  lapply(x, function(y) rbindlist(y, idcol = "model.input"))) %>%
  lapply(., function(x) rbindlist(x, idcol = "Model")) %>%
  rbindlist(., idcol = "setting") %>%
  .[, Model:= factor(Model, levels = c("Liu", "Ishigami", "Sobol' G", "Morris"))] %>%
  .[, model.input:= factor(model.input, levels = paste("X", 1:20, sep = ""))] %>%
  .[, parameters:= gsub("Theta", "$\\\\\\theta$", parameters)]

# EXPORT DATA -----

fwrite(final.sobol, "final.sobol.csv")

# PLOT SOBOL INDICES -----

ggplot(final.sobol, aes(parameters, original,
  fill = sensitivity)) +
  geom_boxplot(outlier.size = 0.2) +
  labs(x = "",
  y = "Sobol' index") +
  scale_fill_discrete(name = "Sobol' indices",
  labels = c(expression(S[italic(i)]),
  expression(S[italic(T[i])])))) +
  theme_bw() +
  facet_wrap(~setting) +
  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),
  legend.position = "none")
```





## 5 Extra plots

```
# MERGE UNCERTAINTY IN PAWN AND SOBOLE'-----

a <- plot.uncertainty.pawn +
  theme(legend.position = "none")

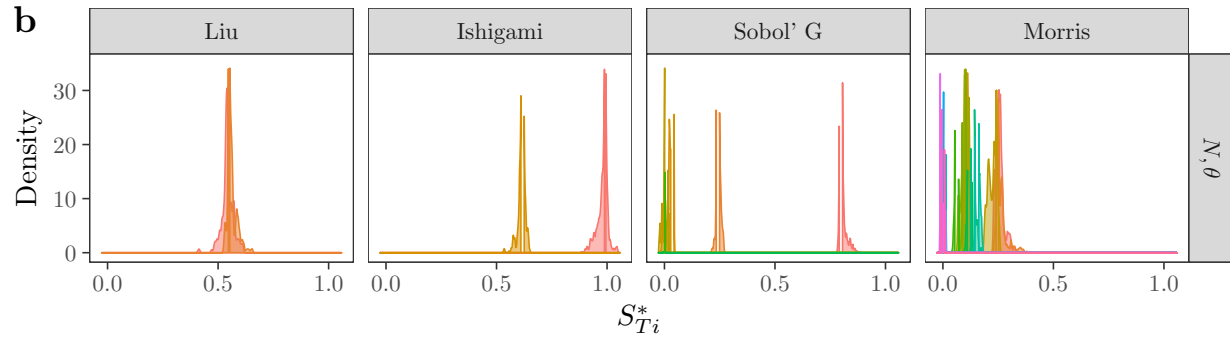
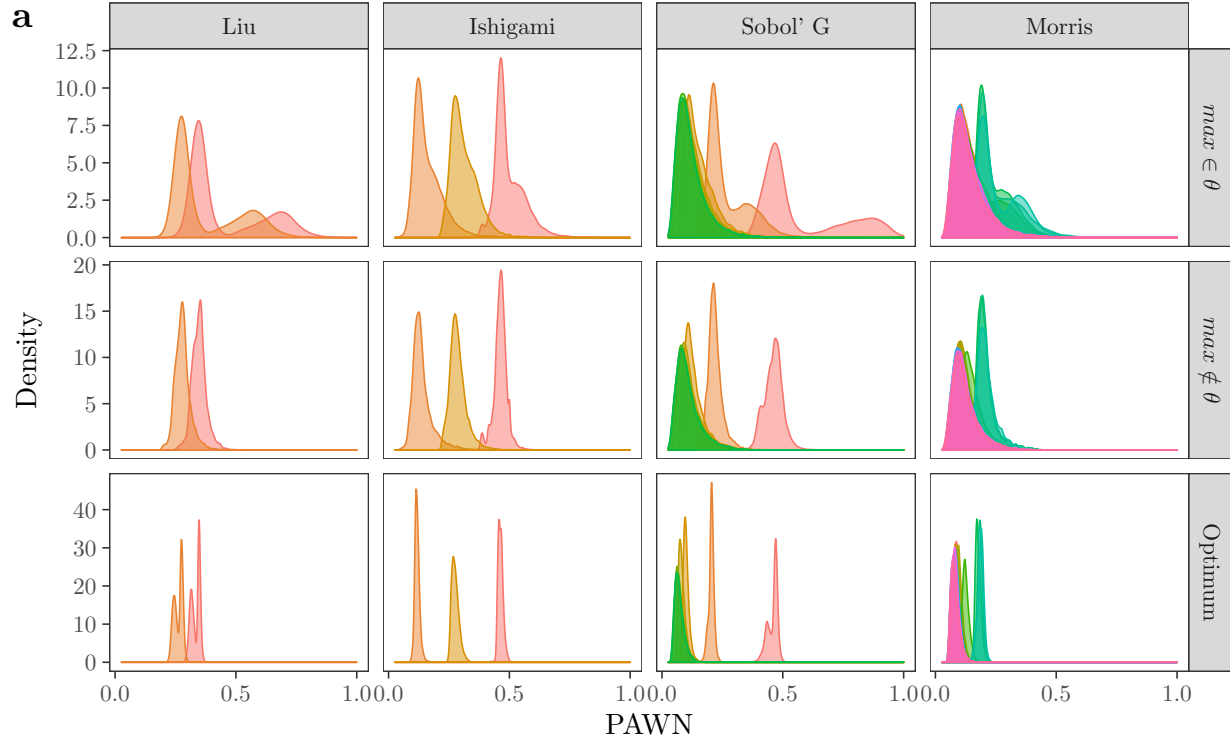
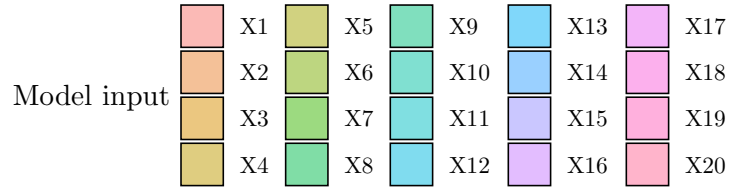
b <- AB.sobol[!setting == "N"] %>%
  ggplot(., aes(value,
                fill = parameter,
                color = parameter)) +
  geom_density(alpha = 0.5,
               position = "identity") +
  facet_grid(setting~Model) +
  scale_fill_discrete(name = "Model input") +
  scale_color_discrete(guide = FALSE) +
  labs(x = "$S_{Ti}~*$",
       y = "Density") +
  scale_x_continuous(breaks = pretty_breaks(n = 3)) +
  scale_y_continuous(limits = c(0, 35)) +
  theme_bw() +
  theme(legend.position = "none",
        legend.box = "horizontal",
        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))

# Get legend
legend <- get_legend(a + theme(legend.position = "top"))

# Merge
bottom <- plot_grid(a, b,
                    ncol = 1,
                    labels = "auto",
                    align = "h",
                    rel_heights = c(1, 0.46))
```

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

```
plot_grid(legend, bottom,
          labels = c("", ""),
          ncol = 1,
          align = "",
          rel_heights = c(0.2, 1))
```



# PLOT AGGREGATED SOBOL' INDICES -----

```
a <- final.pawn.ci[sensitivity == "Si" | sensitivity == "STi"] %>%
  ggplot(., aes(parameters, original,
                fill = sensitivity)) +
  geom_boxplot(outlier.size = 0.2) +
  labs(x = "",
       y = "Sobol' index") +
  scale_fill_discrete(name = "Sobol' indices",
                     labels = c(expression(S[italic(i)])),
```

```

                                expression(S[italic(T[i])])))) +
theme_bw() +
facet_wrap(~ setting) +
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),
      legend.position = "none")

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

b <- final.sobol[!setting == "N"] %>%
  ggplot(., aes(parameters, original, fill = sensitivity)) +
  geom_boxplot(outlier.size = 0.2) +
  labs(x = "",
       y = "") +
  scale_fill_discrete(name = expression(paste("Sobol'"~T[i])),
                     labels = c(expression(S[italic(i)]),
                                expression(S[italic(T[i])])))) +

  facet_wrap(~ setting) +
  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),
        legend.position = "none")

bottom <- plot_grid(a, b,
                   ncol = 2,
                   align = "hv",
                   labels = "auto",
                   rel_widths = c(2.58, 1))

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

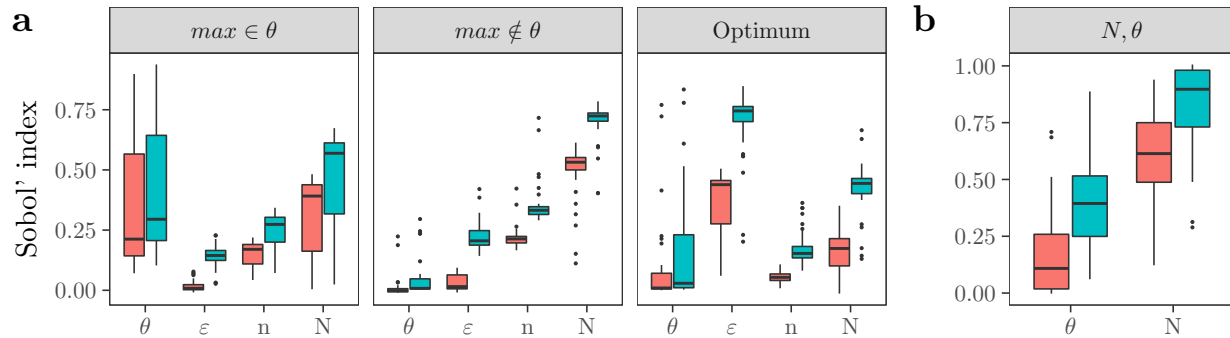
plot_grid(legend, bottom,
          ncol = 1,
          align = "hv",
          rel_heights = c(0.21, 1))

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

```

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

Sobol' indices ■  $S_i$  ■  $S_{T_i}$

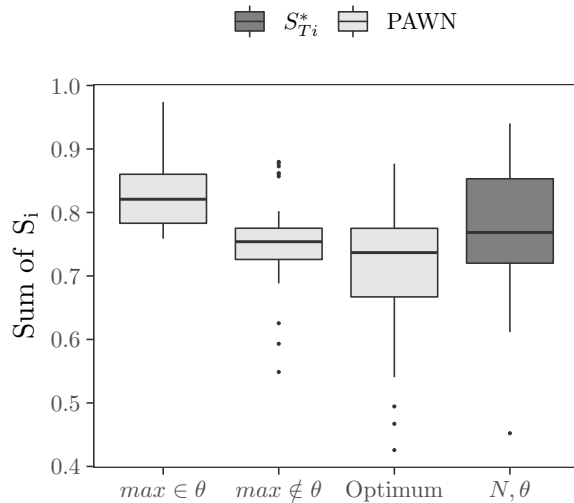


# PLOT AGGREGATED SUM OF SI -----

```
final.sobol2 <- setnames(final.sobol, "Model", "model")

rbind(final.pawn.ci[sensitivity == "Si"][, type:= "PAWN"],
      final.sobol2[!setting == "N" & sensitivity == "Si"][, type:= "$S_{Ti}~*$")] %>%
  .[, sum(original), .(setting, model, model.input, type)] %>%
  .[, setting:= factor(setting, levels = c("$max \\in \\theta$",
                                           "$max \\notin \\theta$",
                                           "Optimum",
                                           "$N, \\theta$"))] %>%

ggplot(., aes(setting, V1, fill = type)) +
  scale_fill_grey(start = 0.5, end = 0.9,
                  name = "") +
  geom_boxplot(outlier.size = 0.2) +
  labs(x = "",
       y = expression(paste("Sum of"~S[i]))) +
  theme_bw() +
  theme(legend.position = "top",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```



```
# ARRANGE TO PLOT SOBOLO' INDICES FOR EACH FUNCTION -----

tmp <- split(final.pawn.ci, final.pawn.ci$model)
gg <- list()
for(i in names(tmp)) {
  for(j in 1:3) {
    gg[[i]][[j]] <- plot_sobol(tmp[[i]], type = j) +
      scale_y_continuous(breaks = pretty_breaks(n = 3)) +
      facet_grid(model.input ~ setting) +
      labs(x = "",
           y = "Sobol' indices") +
      theme(legend.position = "none")
  }
}

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

# PLOT SOBOLO' INDICES FOR LIU, ISHIGAMI AND SOBOLO' G -----

all <- lapply(1:3, function(x) {
  left <- plot_grid(gg[[1]][[x]], gg[[2]][[x]],
                    labels = c("a", "b"),
                    align = "h",
                    ncol = 1)
  plot_grid(left, gg[[3]][[x]],
            labels = c("", "c"),
            ncol = 2)
})
```

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

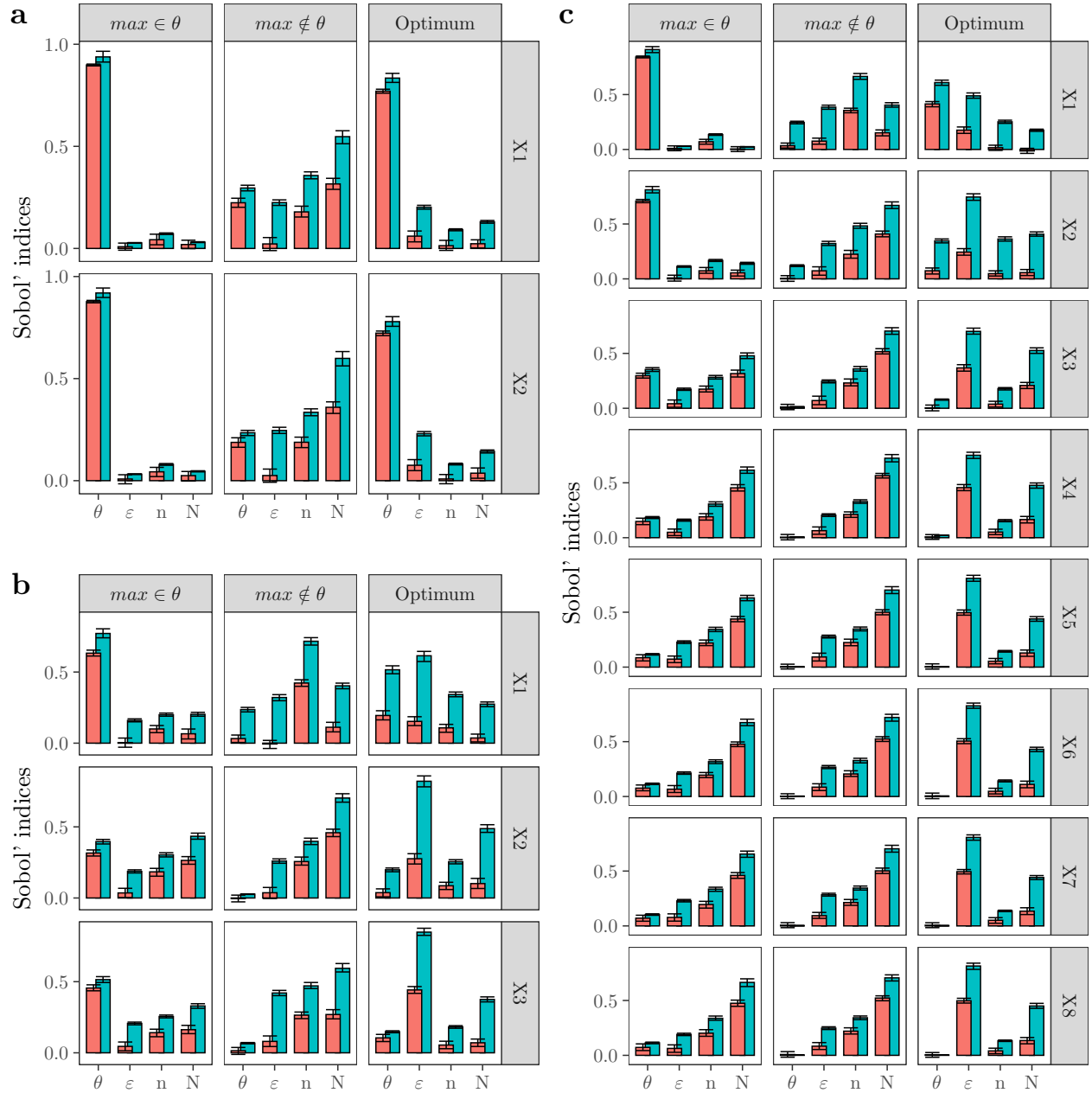
```
## Warning: Graphs cannot be horizontally aligned unless the axis parameter is
```

## set. Placing graphs unaligned.

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

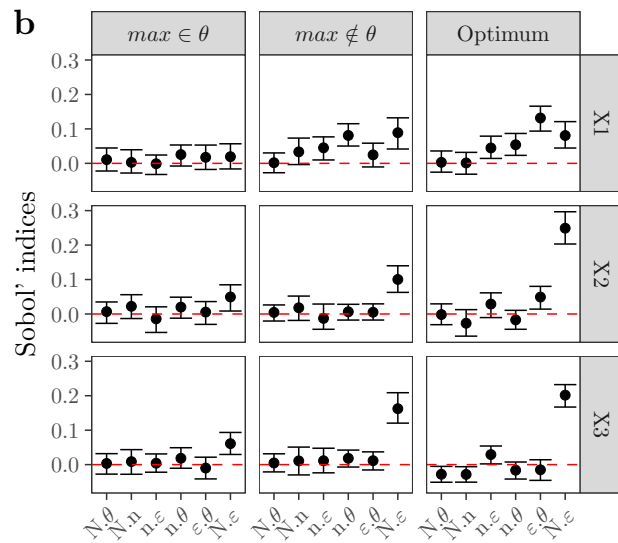
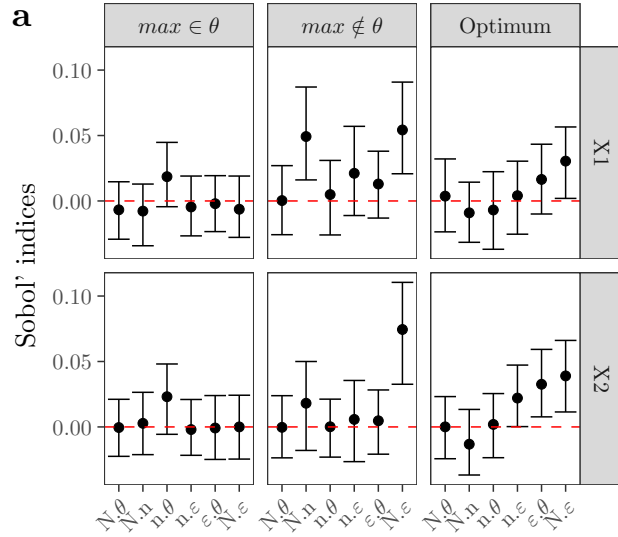
```
plot_grid(legend, all[[1]],
          ncol = 1,
          rel_heights = c(0.1, 1))
```

Sobol' indices ■  $S_i$  ■  $S_{T_i}$

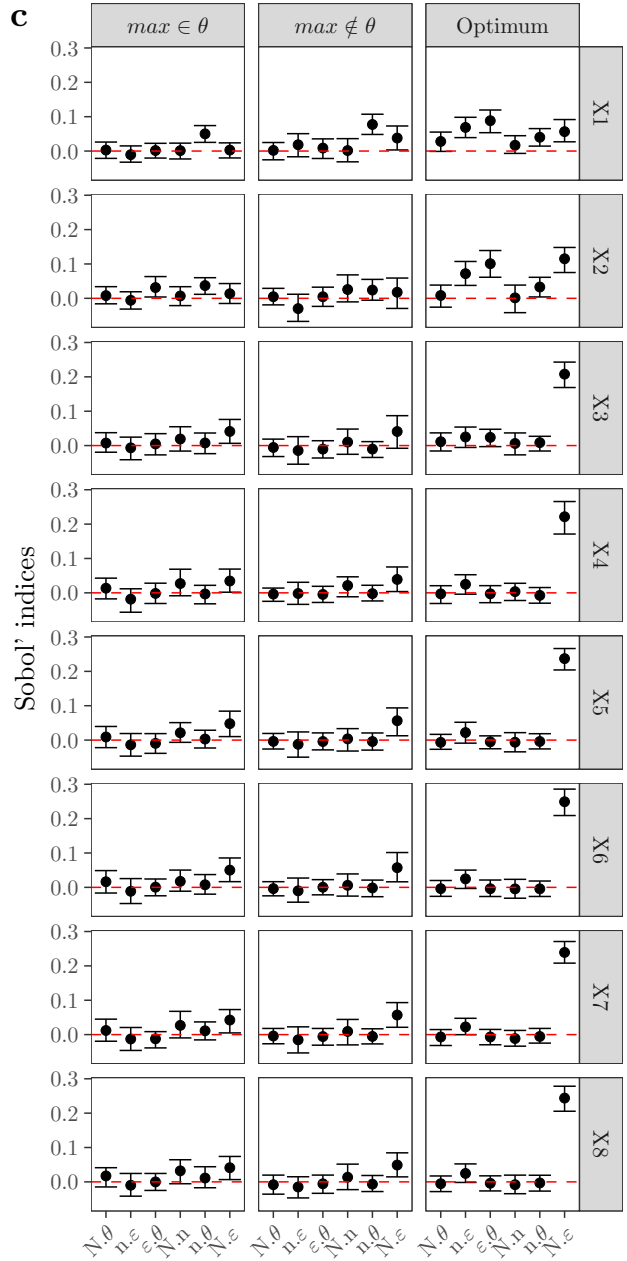


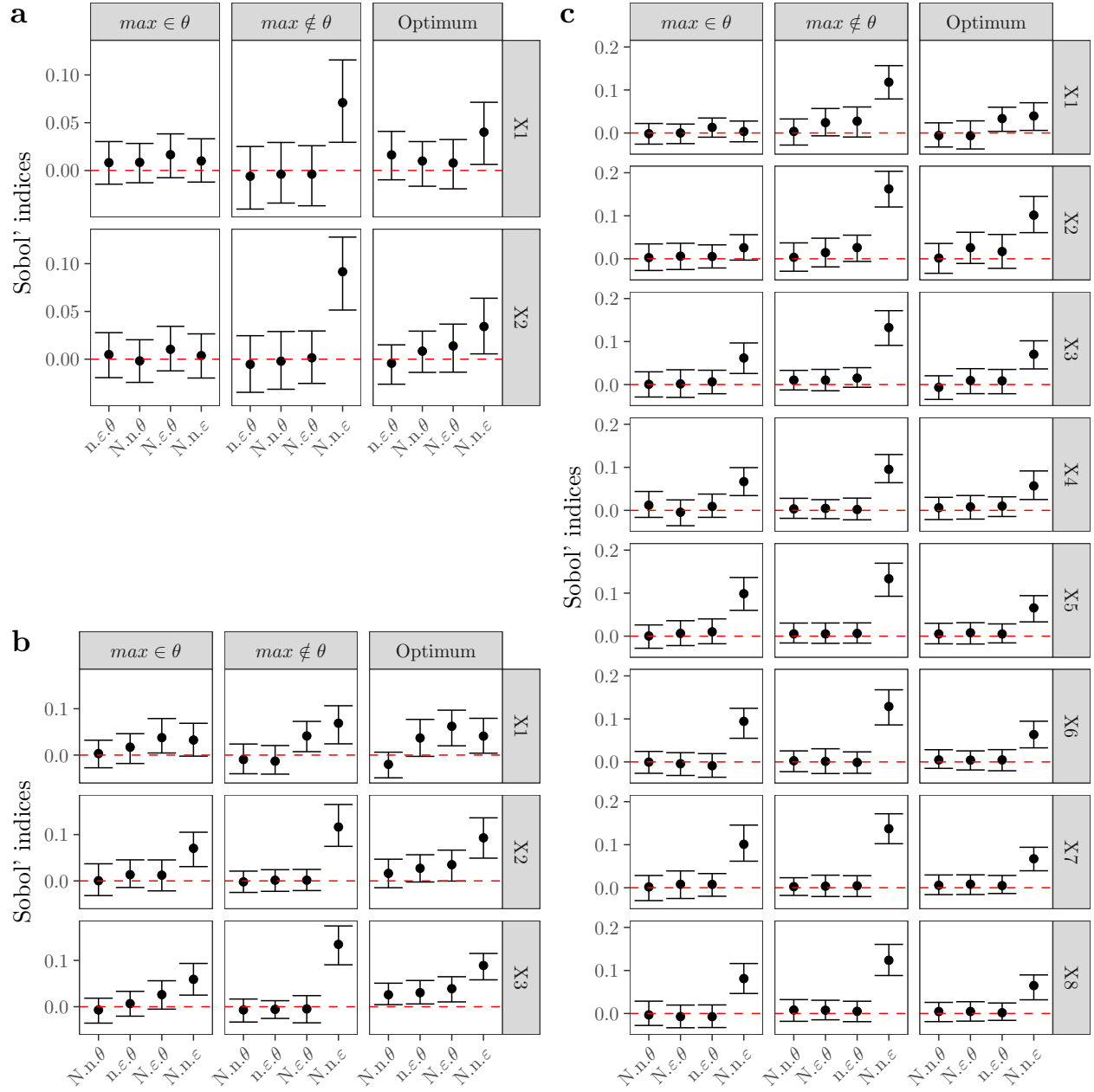
```
lapply(2:3, function(x) all[x])
```

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



```
##
##
## [[2]]
## [[2]] [[1]]
```



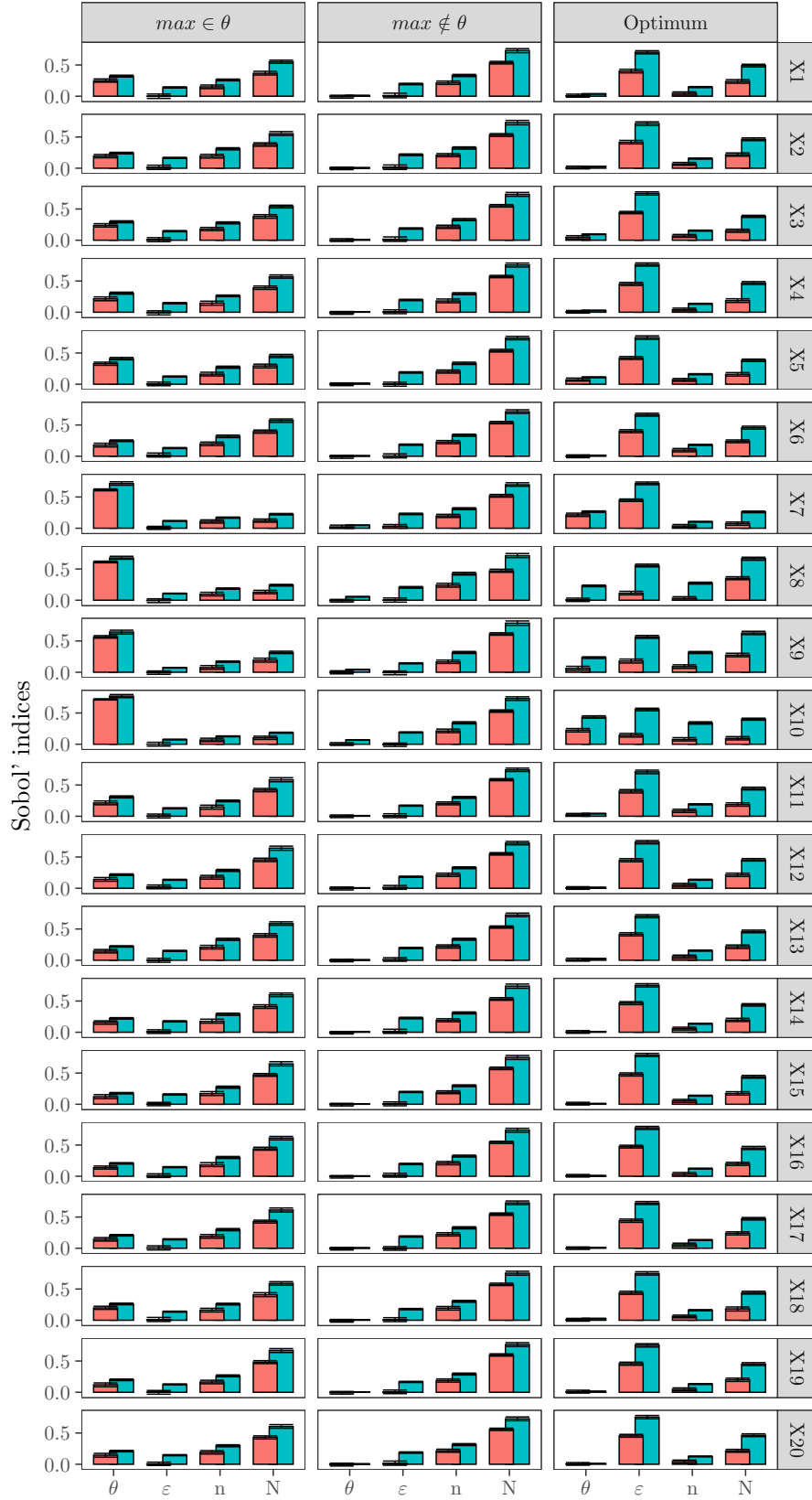


*# PLOT SOBOL' INDICES FOR THE MORRIS FUNCTION* -----

```
plot_grid(legend, gg[[4]][[1]],
          ncol = 1,
          rel_heights = c(0.07, 1))
```

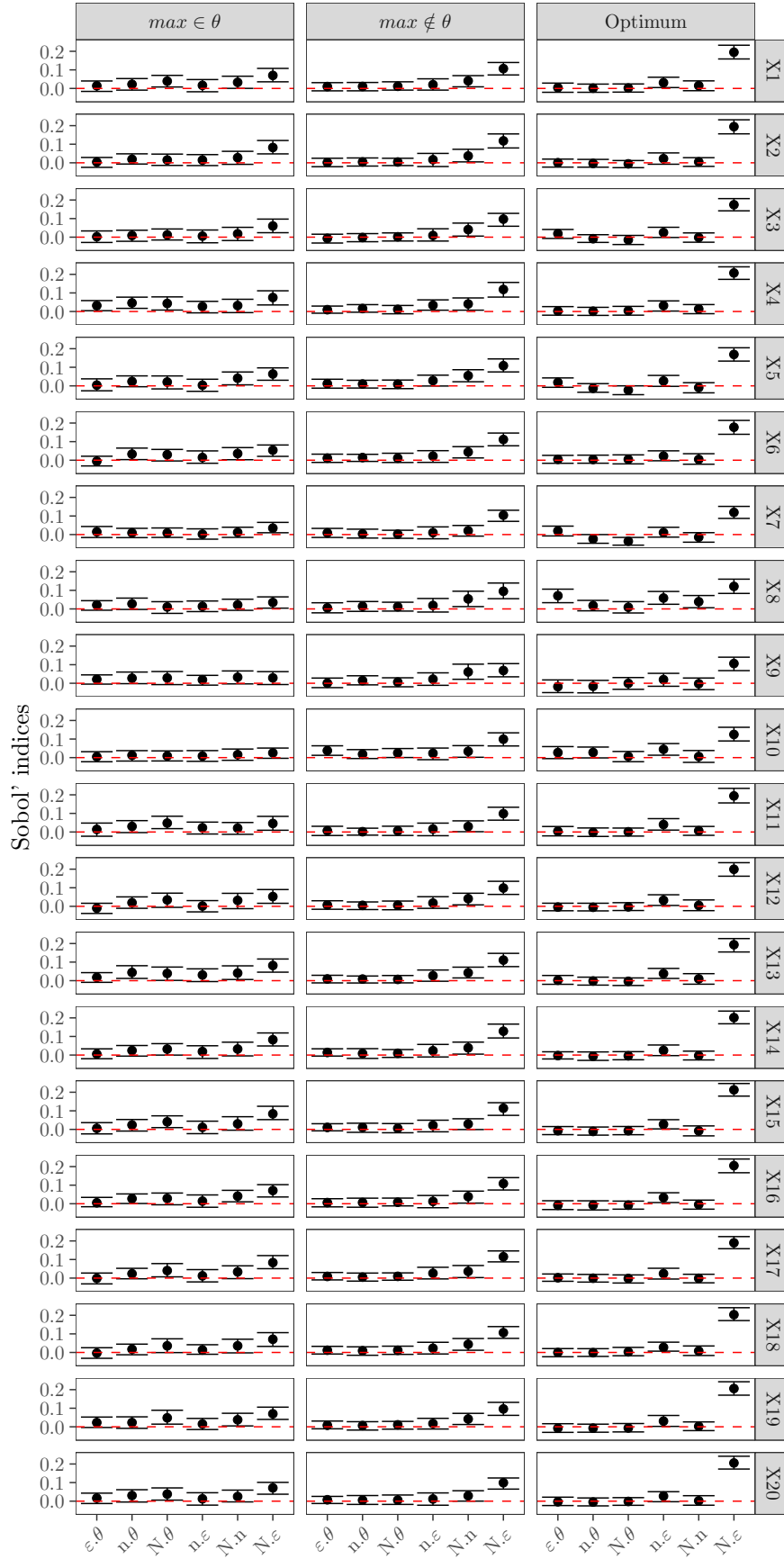


Sobol' indices ■  $S_i$  ■  $S_{T_i}$

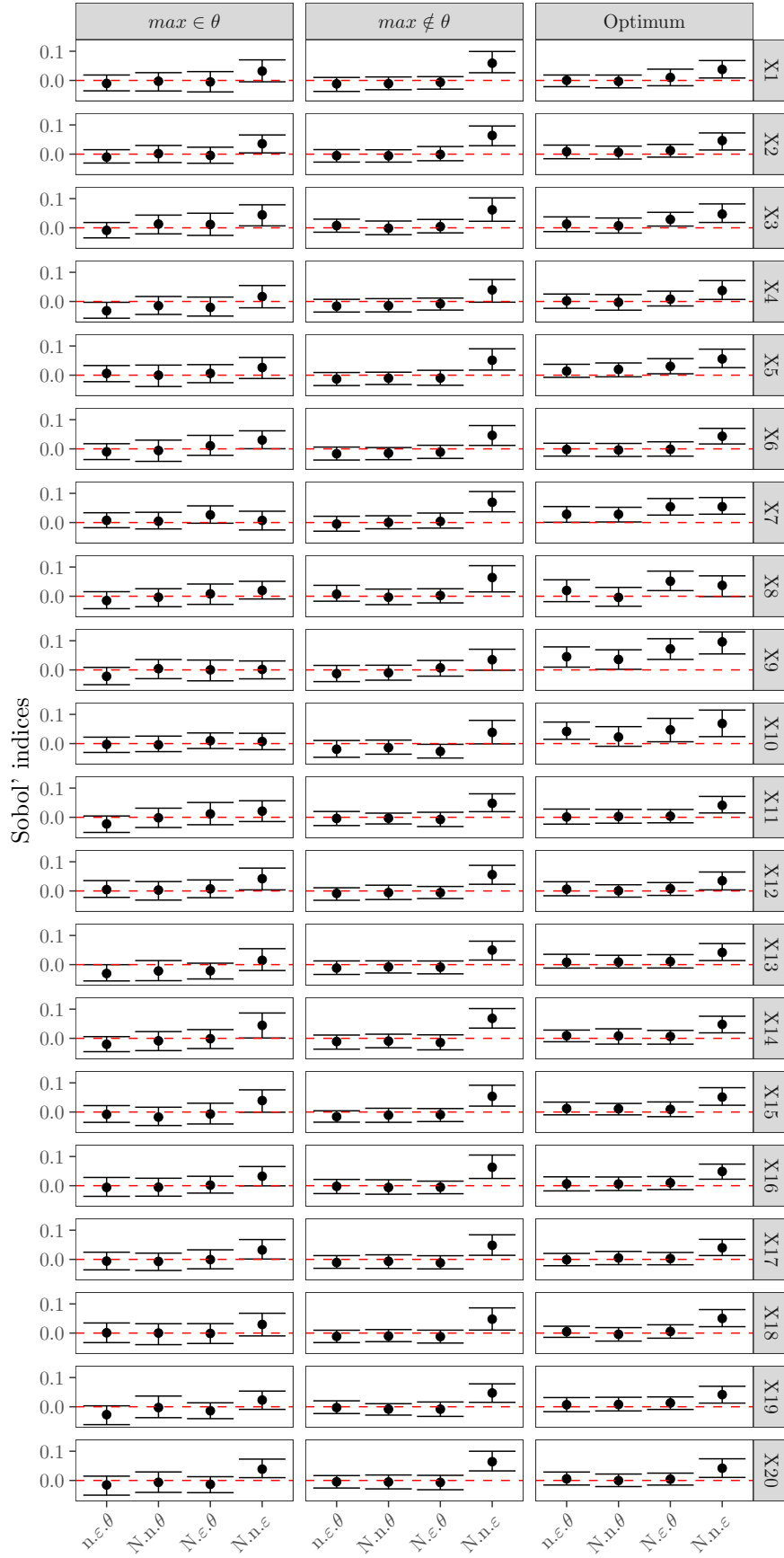


```
lapply(2:3, function(x) gg[[4]][[x]])
```

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



```
##  
## [[2]]
```



```

# MERGE SECOND AND THIRD-ORDER EFFECTS -----

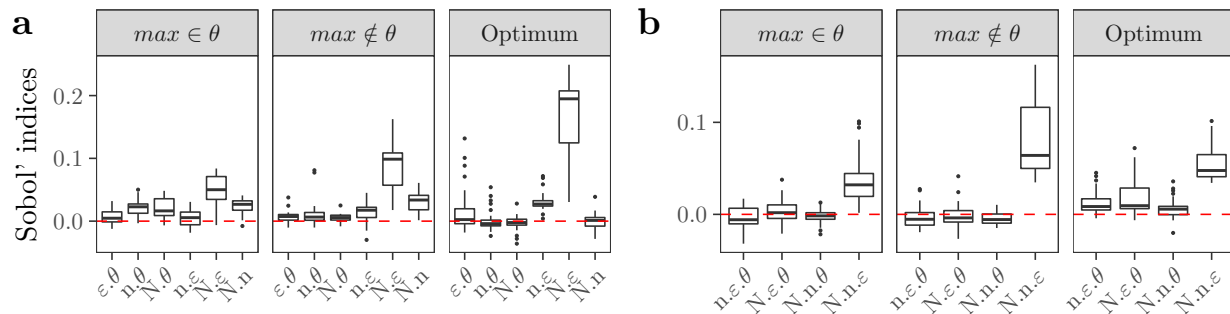
gg <- list()
second.third <- c("Sij", "Sijk")
for(i in second.third) {
  gg[[i]] <- final.pawn.ci[sensitivity == i] %>%
    ggplot(., aes(parameters, original)) +
    geom_boxplot(outlier.size = 0.2) +
    labs(x = NULL,
         y = "Sobol' indices") +
    scale_fill_discrete(name = "Sobol' indices",
                        labels = c(expression(S[italic(i)]),
                                      expression(S[italic(T[i])])))) +

    theme_bw() +
    geom_hline(yintercept = 0,
               lty = 2,
               color = "red") +
    scale_y_continuous(breaks = pretty_breaks(n = 3)) +
    facet_wrap(~ setting) +
    theme(legend.position = "none",
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          axis.text.x = element_text(angle = 45,
                                       hjust = 1))
}

# PLOT SECOND AND THIRD-ORDER EFFECTS -----

plot_grid(gg[[1]],
          gg[[2]] + labs(x = "", y = ""),
          ncol = 2,
          labels = "auto",
          align = "hv")

```



```

# PLOT AGGREGATED SOBOLE' INDICES AFTER WEIGHTING -----

final.pawn.ci[sensitivity == "Si" | sensitivity == "STi"] %>%
  # For each function, setting and design parameter, compute

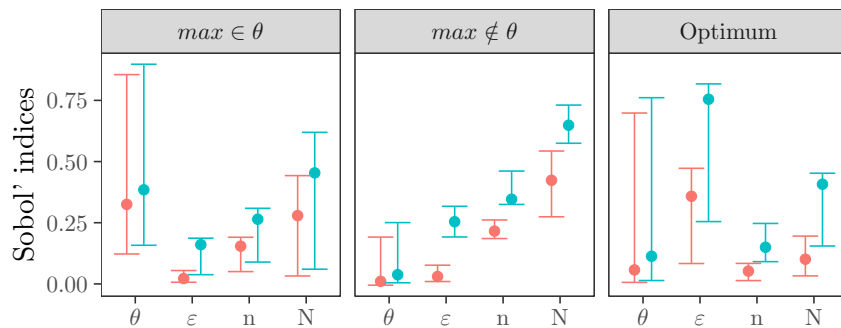
```

```

# the median value of  $S_i$  and  $ST_i$ 
[, .(Median = median(original)),
  by = .(setting, model, sensitivity, parameters)] %>%
# Compute the aggregated median and the percentiles
[, .(aggregated.median = median(Median),
  low.ci = quantile(Median, probs = 0.025),
  high.ci = quantile(Median, probs = 0.975)),
  by = .(setting, sensitivity, parameters)] %>%
ggplot(., aes(parameters, aggregated.median,
  color = sensitivity)) +
geom_point(position = position_dodge(0.6)) +
geom_errorbar(aes(ymin = low.ci,
  ymax = high.ci),
  position = position_dodge(0.6)) +
labs(x = "",
  y = "Sobol' indices") +
scale_color_discrete(name = "Sobol' indices",
  labels = c(expression(S[italic(i)]),
    expression(S[italic(T[i])])))) +
theme_bw() +
facet_wrap(~ setting) +
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),
  legend.position = "top")

```

Sobol' indices —  $S_i$  —  $ST_i$



# PLOT DESIGN PARAMETERS FOR SOBOLE: ALL FUNCTIONS -----

```

tmp <- final.sobol[!setting == "N"] %>%
  split(., .$Model)
gg <- list()
for(i in names(tmp)) {
  gg[[i]] <- plot_sobol(tmp[[i]], type = 1) +

```

```

    scale_y_continuous(breaks = pretty_breaks(n = 3)) +
    facet_grid(model.input ~.) +
    labs(x = "",
         y = "Sobol' indices") +
    theme(legend.position = "none")
}

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

# PLOT SOBOLO' INDICES FOR LIU, ISHIGAMI AND SOBOLO' G -----

left <- plot_grid(gg[[1]], gg[[2]],
                  labels = c("a", "b"),
                  align = "h",
                  ncol = 1)

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

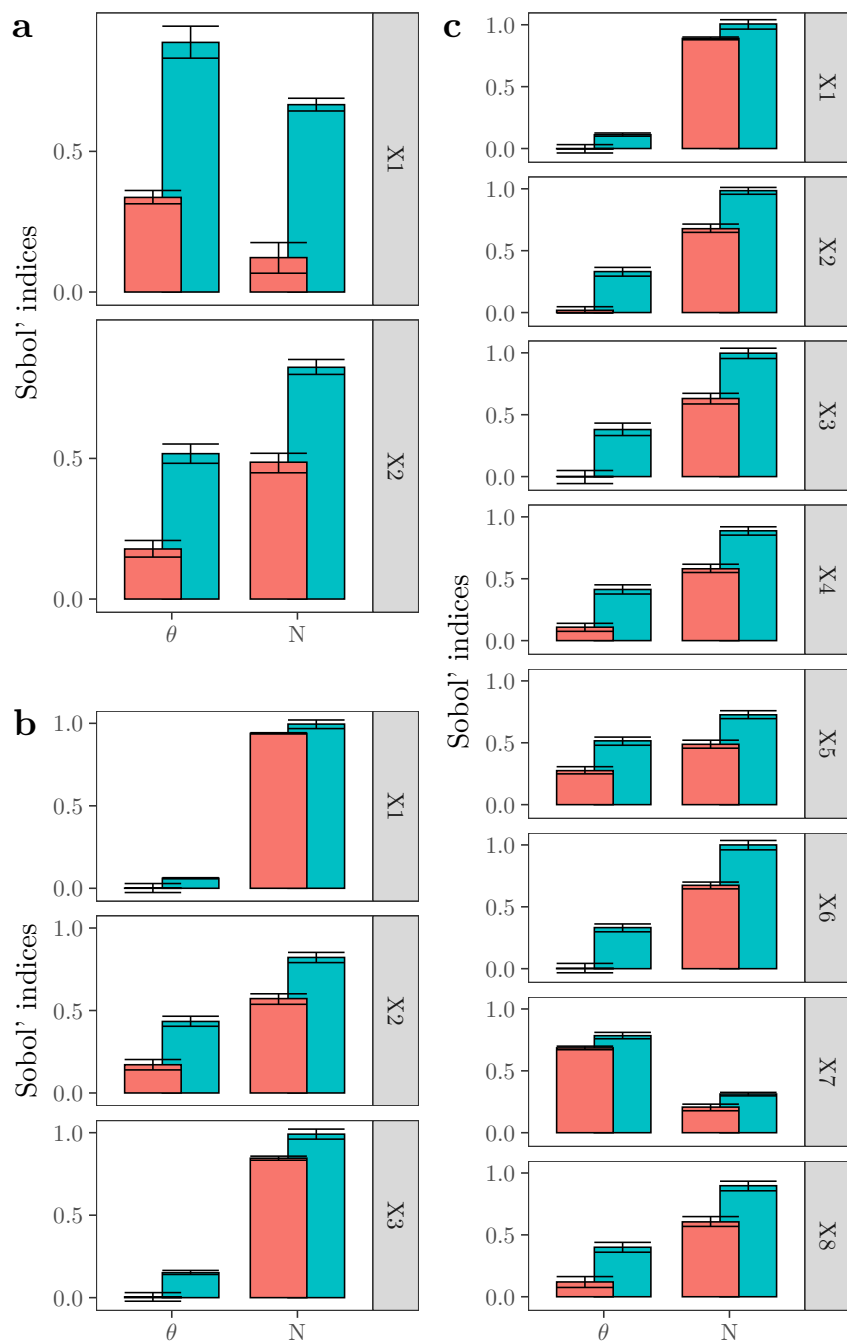
all <- plot_grid(left, gg[[3]],
                 labels = c("", "c"),
                 ncol = 2)

plot_grid(legend, all,
          ncol = 1,
          rel_heights = c(0.1, 1))

```

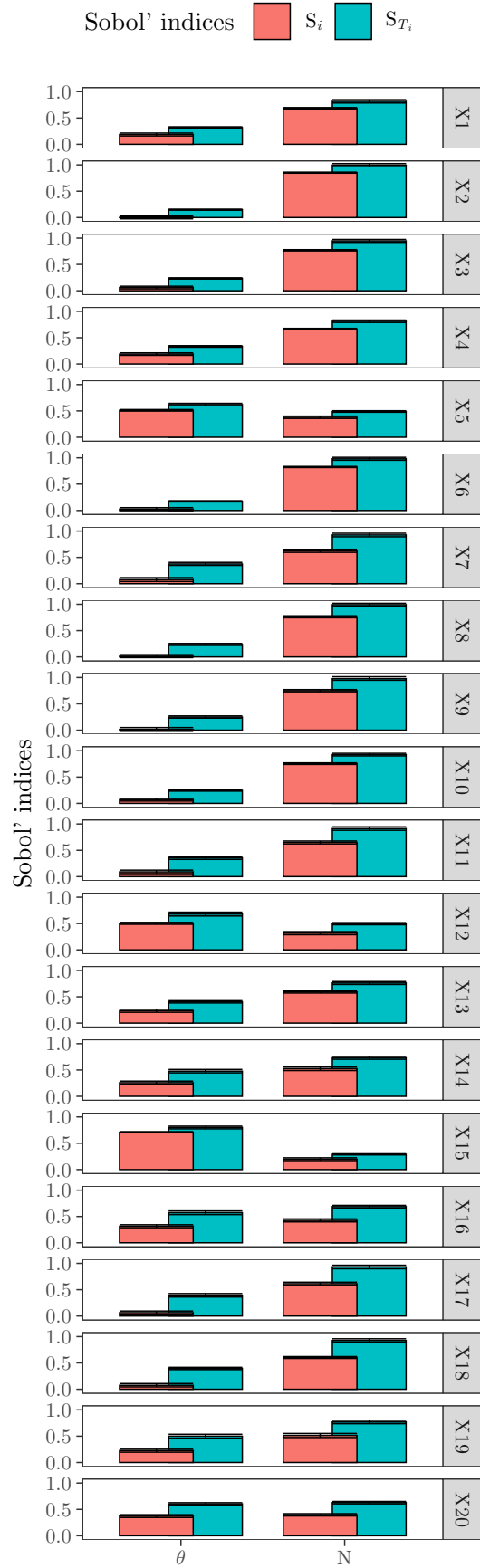


Sobol' indices ■  $S_i$  ■  $S_{T_i}$



# PLOT SOBOL' INDICES FOR THE MORRIS FUNCTION -----

```
plot_grid(legend, gg[[4]],
          ncol = 1,
          rel_heights = c(0.07, 1))
```



## 6 Session information

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

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel stats graphics grDevices utils datasets methods
## [8] base
##
## other attached packages:
## [1] checkpoint_0.4.7 wesanderson_0.3.6 sensobol_0.2.1
## [4] pawnr_0.0.0.9000 overlapping_1.5.4 testthat_2.2.1
## [7] cowplot_1.0.0 scales_1.0.0 doParallel_1.0.15
## [10] iterators_1.0.12 foreach_1.4.7 boot_1.3-23
## [13] sensitivity_1.16.1 randtoolbox_1.30.0 rngWELL_0.10-5
## [16] data.table_1.12.2 forcats_0.4.0 stringr_1.4.0
## [19] dplyr_0.8.3 purrr_0.3.2 readr_1.3.1
## [22] tidyr_1.0.0 tibble_2.1.3 ggplot2_3.2.1
## [25] tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
## [1] nlme_3.1-141 fs_1.3.1 usethis_1.5.1
## [4] lubridate_1.7.4 devtools_2.2.0 httr_1.4.1
## [7] rprojroot_1.3-2 tools_3.6.1 backports_1.1.4
## [10] R6_2.4.0 DT_0.9 lazyeval_0.2.2
## [13] colorspace_1.4-1 withr_2.1.2 tidyselect_0.2.5
## [16] prettyunits_1.0.2 processx_3.4.1 curl_4.1
## [19] compiler_3.6.1 cli_1.1.0 rvest_0.3.4
## [22] xml2_1.2.2 desc_1.2.0 labeling_0.3
## [25] callr_3.3.2 digest_0.6.21 rmarkdown_1.15
## [28] pkgconfig_2.0.3 htmltools_0.3.6 bibtex_0.4.2
## [31] sessioninfo_1.1.1 htmlwidgets_1.3 rlang_0.4.0
## [34] readxl_1.3.1 rstudioapi_0.10 generics_0.0.2
## [37] tikzDevice_0.12.3 jsonlite_1.6 magrittr_1.5
## [40] Rcpp_1.0.2 munsell_0.5.0 lifecycle_0.1.0
## [43] stringi_1.4.3 yaml_2.2.0 gbrd_0.4-11
```

## [46]	pkgbuild_1.0.5	grid_3.6.1	crayon_1.3.4
## [49]	lattice_0.20-38	haven_2.1.1	hms_0.5.1
## [52]	zeallot_0.1.0	knitr_1.25	ps_1.3.0
## [55]	pillar_1.4.2	codetools_0.2-16	pkgload_1.0.2
## [58]	glue_1.3.1	evaluate_0.14	remotes_2.1.0
## [61]	modelr_0.1.5	vctrs_0.2.0	Rdpack_0.11-0
## [64]	cellranger_1.1.0	gtable_0.3.0	assertthat_0.2.1
## [67]	xfun_0.9	broom_0.5.2	filehash_2.4-2
## [70]	tinytex_0.16	memoise_1.1.0	ellipsis_0.3.0