More complex models produce more uncertain estimates

R code

Arnald Puy

Contents

```r
# PRELIMINARY FUNCTIONS ------------------------------------------------------

# Function to read in all required packages in one go:
loadPackages <- function(x) {
  for(i in x) {
    if(!require(i, character.only = TRUE)) {
      install.packages(i, dependencies = TRUE)
      library(i, character.only = TRUE)
    }
  }
}

# Load the packages
loadPackages(c(
  "data.table", "tidyverse", "parallel", "doParallel", "deSolve",
  "sensobol", "crone", "KScorrect", "cowplot", "wesanderson",
  "Rfast", "RcppAlgos", "scales", "pracma", "grid"))

# Create custom theme
theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent",
                                           color = NA),
          legend.key = element_rect(fill = "transparent",
                                    color = NA),
          strip.background = element_rect(fill = "white"))
}

# Set checkpoint

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

checkpoint("2021-05-10",
           R.version ="4.0.3",
           checkpointLocation = getwd())
```

1 PSACOIN model

```r
# PSACOIN MODEL ------------------------------------------------------------

psacoin0_fun <- function(R_0, X_B, K_DB, K_DG, D_G0, X_G,
                         alpha_G, V_G, W, W_M, t) {

  # WASTE MODEL ------------------------------------------
  #---------------------------------------------------

  # Constants
  I_0 <- 2.035e-5
  lambda <- 1.07e-5
  S <- 1.2e6
  Q <- 2e+08

  tau_0 <- Q / (R_0 * S)
  I <- I_0 * exp(-lambda * t)
  heaviside_out <- ifelse((t - tau_0) < 0, 1, 0)
  R_wf <- R_0 * (1 - heaviside_out)
  Fw <- R_wf * I * S

  # BUFFER MODEL -----------------------------------------
  #---------------------------------------------------

  # Constants
  p_B <- 1.85e3
  e_B <- 0.099
  D_B <- 0.03

  R_B <- 1 + ((1 - e_B) * K_DB * p_B) / e_B
  tau_B <- (X_B^2 * R_B) / (4 * D_B)
  F_B <- ifelse(t < tau_B, 0, Fw * (t - tau_B) * exp(-lambda * tau_B))

  # GEOSPHERE MODEL ---------------------------------------
  #---------------------------------------------------

  # Constants
  p_G <- 2e3
  epsilon_G <- 0.3

  D_G <- D_G0 + alpha_G * V_G
  R_G <- 1 + p_G / epsilon_G * (1 - epsilon_G) * K_DG

  alpha <- X_G
  beta <- 2 * sqrt(D_G / R_G)
  gamma <- V_G / R_G
```

```
  a <- (gamma / beta)^2
  b <- -((2 * alpha * gamma) / beta^2 + 1)
  c <- (alpha / beta)^2

  tau_H <- (-b + sqrt(b^2 - 4 * a * c)) / (2 * a)
  tau_L <- (-b - sqrt(b^2 - 4 * a * c)) / (2 * a)

  t_prim <- ((t - tau_L - tau_B) / (tau_H + tau_0 - tau_L)) * tau_0 + tau_B
  F_G <- ifelse(t < (tau_B + tau_L) |
                  t > (tau_B + tau_H + tau_0), 0,
                tau_0 / (tau_H + tau_0 - tau_L) * F_B * t_prim * exp(-lambda * (t - t_prim)))


  # BIOSPHERE MODEL ------------------------------------
  #-----------------------------------------------------

  # Constant
  A <-  2.04e11
  D <- 2.3e-9

  C <- (F_G * A) / W
  H <- C * W_M * D

  ########################################################

  return(list(Fw, F_B, F_G, H))
}
```

```
# SETTINGS PSACOIN ---------------------------------------------------------

N <- 2^15
R <- 10^3
matrices <- c("A", "B", "AB", "BA")
order <- "third"
first <- "azzini"
total <- "azzini"
n_cores <- detectCores() * 0.75
```

```
# RUN PSACOIN IN EACH COMPARTMENT ------------------------------------------

# BUFFER --------------------

params.buffer <- c("R_0", "X_B", "K_DB")
mat <- sobol_matrices(params = params.buffer, N = N, scrambling = 1,
                      matrices = matrices, order = order)

# Precise distributions
```

```
mat[, "R_0"] <- qlunif(mat[, "R_0"], 10^-2.57, 10^1.11)
mat[, "X_B"] <- qunif(mat[, "X_B"], 0.5, 5)
mat[, "K_DB"] <- qlnorm(mat[, "K_DB"], -2.38, 0.143) # Selenium

y.buffer <- psacoin0_fun(R_0 = mat[, "R_0"],
                         X_B = mat[, "X_B"],
                         K_DB = mat[, "K_DB"],
                         K_DG = NULL,
                         D_GO = NULL,
                         X_G = NULL,
                         alpha_G = NULL,
                         V_G = NULL,
                         W = NULL,
                         W_M = NULL,
                         t = 10^7)[[2]]

# GEOSPHERE --------------------

params.geo <- c("R_0", "X_B", "K_DB", "K_DG", "D_GO", "X_G", "alpha_G", "V_G")
mat <- sobol_matrices(params = params.geo, N = N, scrambling = 1,
                      matrices = matrices, order = order)

mat[, "R_0"] <- qlunif(mat[, "R_0"], 10^-2.57, 10^1.11)
mat[, "X_B"] <- qunif(mat[, "X_B"], 0.5, 5)
mat[, "K_DB"] <- qlnorm(mat[, "K_DB"], -2.38, 0.143) # Selenium
mat[, "K_DG"] <- qlnorm(mat[, "K_DG"], -3.38, 0.3) # Selenium
mat[, "D_GO"] <- qnorm(mat[, "D_GO"], 0.04, 0.001)
mat[, "X_G"] <- qunif(mat[, "X_G"], 10^3, 10^4)
mat[, "alpha_G"] <- qlunif(mat[, "alpha_G"], 10^0.3, 10^2.3)
mat[, "V_G"] <- qlunif(mat[, "V_G"], 10^-3, 10^-1)

y.geo <- psacoin0_fun(R_0 = mat[, "R_0"],
                      X_B = mat[, "X_B"],
                      K_DB = mat[, "K_DB"],
                      K_DG = mat[, "K_DG"],
                      D_GO = mat[, "D_GO"],
                      X_G = mat[, "X_G"],
                      alpha_G = mat[, "alpha_G"],
                      V_G = mat[, "V_G"],
                      W = NULL,
                      W_M = NULL,
                      t = 10^7)[[3]]

# BIOSPHERE --------------------------

params <- c("R_0", "X_B", "K_DB", "K_DG", "D_GO", "X_G",
            "alpha_G", "V_G", "W", "W_M")
```

```r
mat <- sobol_matrices(params = params, N = N, scrambling = 1,
                      matrices = matrices, order = order)

mat[, "R_0"] <- qlunif(mat[, "R_0"], 10^-2.57, 10^1.11)
mat[, "X_B"] <- qunif(mat[, "X_B"], 0.5, 5)
mat[, "K_DB"] <- qlnorm(mat[, "K_DB"], -2.38, 0.143) # Selenium
mat[, "K_DG"] <- qlnorm(mat[, "K_DG"], -3.38, 0.3) # Selenium
mat[, "D_GO"] <- qnorm(mat[, "D_GO"], 0.04, 0.001)
mat[, "X_G"] <- qunif(mat[, "X_G"], 10^3, 10^4)
mat[, "alpha_G"] <- qlunif(mat[, "alpha_G"], 10^0.3, 10^2.3)
mat[, "V_G"] <- qlunif(mat[, "V_G"], 10^-3, 10^-1)
mat[, "W"] <- qunif(mat[, "W"], 5 * 10^5, 5 * 10^6)
mat[, "W_M"] <- qunif(mat[, "W_M"], 0.7, 0.9)

y.bio <- psacoin0_fun(R_0 = mat[, "R_0"],
                      X_B = mat[, "X_B"],
                      K_DB = mat[, "K_DB"],
                      K_DG = mat[, "K_DG"],
                      D_GO = mat[, "D_GO"],
                      X_G = mat[, "X_G"],
                      alpha_G = mat[, "alpha_G"],
                      V_G = mat[, "V_G"],
                      W = mat[, "W"],
                      W_M = mat[, "W_M"],
                      t = 10^7)
```

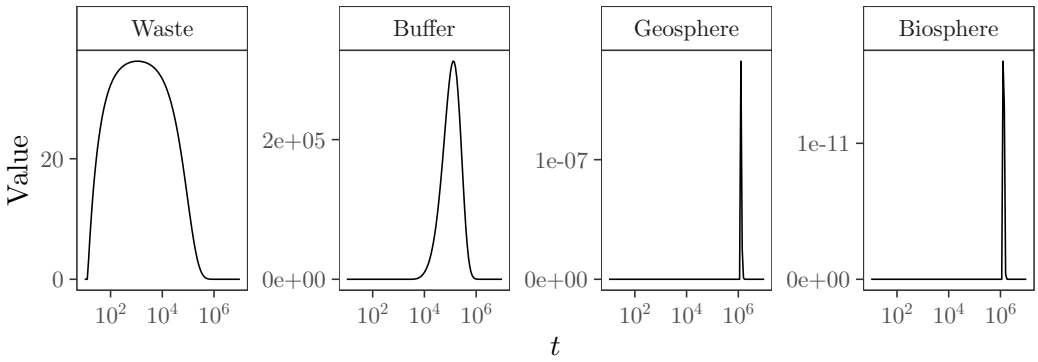
```r
# ARRANGE OUTPUT OF THE COMPARTMENTS ----------------------------------------

model.levels <- c("Waste", "Buffer", "Geosphere", "Biosphere")

names(y.bio) <- model.levels

full.output <- do.call(cbind, y.bio) %>%
  data.table()

A <- full.output %>%
  .[1:N]
```

```r
# RUN MODEL DYNAMICS --------------------------------------------------------

mat <- sobol_matrices(params = params, N = 2^11, matrices = "A")

mat[, "R_0"] <- qlunif(mat[, "R_0"], 10^-2.57, 10^1.11)
mat[, "X_B"] <- qunif(mat[, "X_B"], 0.5, 5)
mat[, "K_DB"] <- qlnorm(mat[, "K_DB"], -2.38, 0.143) # Selenium
mat[, "K_DG"] <- qlnorm(mat[, "K_DG"], -3.38, 0.3) # Selenium
mat[, "D_GO"] <- qnorm(mat[, "D_GO"], 0.04, 0.001)
mat[, "X_G"] <- qunif(mat[, "X_G"], 10^3, 10^4)
```

```r
mat[, "alpha_G"] <- qlunif(mat[, "alpha_G"], 10^0.3, 10^2.3)
mat[, "V_G"] <- qlunif(mat[, "V_G"], 10^-3, 10^-1)
mat[, "W"] <- qunif(mat[, "W"], 5 * 10^5, 5 * 10^6)
mat[, "W_M"] <- qunif(mat[, "W_M"], 0.7, 0.9)

t <- seq(1, 8, 0.05)

y <- mclapply(t, function(t)
  psacoin0_fun(R_0 = mat[, "R_0"],
               X_B = mat[, "X_B"],
               K_DB = mat[, "K_DB"],
               K_DG = mat[, "K_DG"],
               D_G0 = mat[, "D_G0"],
               X_G = mat[, "X_G"],
               alpha_G = mat[, "alpha_G"],
               V_G = mat[, "V_G"],
               W = mat[, "W"],
               W_M = mat[, "W_M"],
               t = 10^t),
  mc.cores = n_cores)
```

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

out <- lapply(y, function(x)
  do.call(cbind, x) %>%
    data.table)

names(out) <- 10^t

out.plot <- rbindlist(out, idcol = "time") %>%
  setnames(., c("time", paste("V", 1:4, sep = "")),
           c("time", model.levels)) %>%
  melt(., measure.vars = model.levels) %>%
  .[, time:= as.numeric(time)] %>%
  .[, mean(value), .(time, variable)] %>%
  .[time >= 10^1 & time <= 10^7]
```

```r
# PLOT DYNAMICS PSACOIN -----------------------------------------------------

plot.psacoin <- ggplot(out.plot, aes(time, V1, group = variable)) +
  geom_line() +
  facet_wrap(~variable, scales = "free_y", ncol = 4) +
  scale_x_log10(labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  scale_y_continuous(breaks = pretty_breaks(n = 2)) +
  labs(x = "$t$", y = "Value") +
  theme_AP()

plot.psacoin
```




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

ind.psacoin <- list()
for(i in 2:4) {
  if(i == 2) {
    y <- y.buffer
    params <- params.buffer
  } else if(i == 3) {
    y <- y.geo
    params <- params.geo
  } else {
    y <- y.bio[[4]]
    params <- c("R_0", "X_B", "K_DB", "K_DG", "D_GO", "X_G",
                "alpha_G", "V_G", "W", "W_M")
  }
  ind.psacoin[[i]] <- sobol_indices(Y = y, N = N, params = params,
                                    first = first, total = total,
                                    R = R, boot = TRUE,
                                    parallel = "multicore", order = order,
                                    ncpus = n_cores, matrices = matrices)
}
```

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

out <- lapply(ind.psacoin, function(x) x$results)[2:4]
names(out) <- c("Buffer", "Geosphere", "Biosphere")

ind.psacoin <- rbindlist(out, idcol = "compartment") %>%
  .[, compartment:= factor(compartment, levels = c("Buffer", "Geosphere", "Biosphere"))]

# Sum of first, second and third-order effects in each compartment
ind.psacoin[!sensitivity == "Ti", sum(original), compartment]
```

```
##    compartment        V1
## 1:      Buffer 1.0000174
## 2:   Geosphere 0.4548629
## 3:   Biosphere 0.3758309
```

```r
psacoin.cv <- A[, lapply(.SD, function(x)
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)), .SDcols = (model.levels)] %>%
  melt(., measure.vars = model.levels, value.name = "cv")

psacoin.kt <- ind.psacoin[sensitivity == "Ti" & original > 0.05] %>%
  .[, .(kt = length(unique(parameters))), compartment] %>%
  rbind(., list("Waste", 1)) %>%
  setnames(., "compartment", "variable")

psacoin.cv.kt <- merge(psacoin.cv, psacoin.kt, by = "variable") %>%
  .[, model:= "PSACOIN"]

print(psacoin.cv.kt)
```

```
##     variable       cv kt   model
## 1:     Waste 1.799618  1 PSACOIN
## 2:    Buffer 2.363077  2 PSACOIN
## 3: Geosphere 7.194928  5 PSACOIN
## 4: Biosphere 9.051070  6 PSACOIN
```

```r
# PLOT PSACOIN CV ------------------------------------------------------------

psacoin.cv.plot <- ggplot(psacoin.cv.kt, aes(variable, cv, group = 1)) +
  geom_line() +
  geom_point(aes(size = kt)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  scale_size_continuous(name = "$k_t$", range = c(1, 4)) +
  labs(x = "", y = "CV") +
  theme_AP() +
  theme(legend.position = "none")
```

```r
# PLOT -----------------------------------------------------------------------

psacoin.ks.plot <- ind.psacoin[!sensitivity == "Ti"] %>%
  .[, sum(original), .(sensitivity, compartment)] %>%
  ggplot(., aes(compartment, V1, fill = sensitivity, order = sensitivity)) +
  geom_col(position = position_stack(reverse = TRUE), color = "black") +
  labs(x = "", y = "$\\sum S_i + \\sum S_{i,j} + \\sum S_{i,j,l}$") +
  scale_fill_manual(name = "Sensitivity",
                    labels = c("$S_i$", "$S_{i,j}$", "$S_{i,j,l}$"),
                    values = wes_palette("Cavalcanti1")) +
  geom_hline(yintercept = 0.99, lty = 2) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme_AP() +
  theme(legend.position = "none",
        axis.title.y = element_text(size = 7)) +
  annotation_custom(textGrob("p", gp = gpar(col = "red")),
                    xmin = 3.5, xmax = 3.5, ymin = 0.98, ymax = 0.98)
```

```r
# PLOT ALL FIGURES PSACOIN ------------------------------------------------------

psacoin.plots <- plot_grid(psacoin.cv.plot,
                           psacoin.ks.plot, ncol = 2, labels = "auto")
```

```r
# EXPORT PSACOIN SENSITIVITY INDICES --------------------------------------------

fwrite(ind.psacoin, "ind.psacoin.csv")
```

2 The Bateman equations

```
# BATEMAN EQUATIONS ------------------------------------------------------------

bateman <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dx1 <- - lambda1 * x1
    dx2 <- - lambda2 * x2 + lambda1 * x1
    dx3 <- - lambda3 * x3 + lambda2 * x2
    dx4 <- - lambda4 * x4 + lambda3 * x3
    dx5 <- - lambda5 * x5 + lambda4 * x4
    dx6 <- - lambda6 * x6 + lambda5 * x5
    dx7 <- - lambda7 * x7 + lambda6 * x6
    dx8 <- - lambda8 * x8 + lambda7 * x7
    dx9 <- - lambda9 * x9 + lambda8 * x8
    dx10 <- - lambda10 * x10 + lambda9 * x9
    list(c(dx1, dx2, dx3, dx4, dx5, dx6, dx7, dx8, dx9, dx10))
  })
}
```

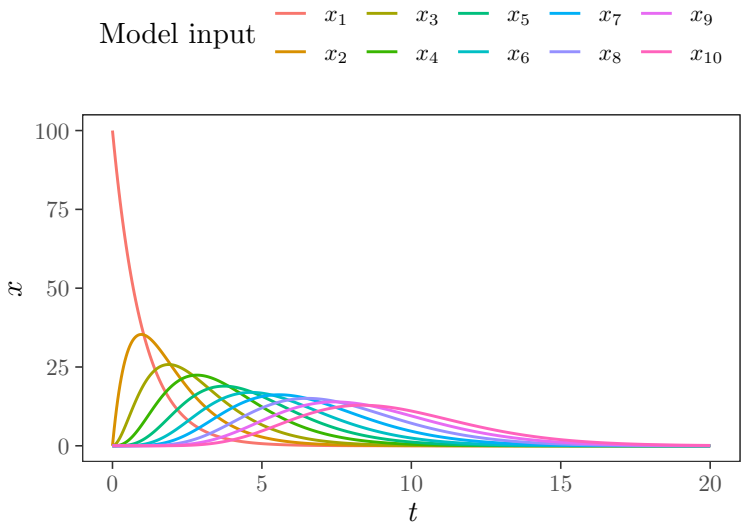
```
# BATEMAN SETTINGS -------------------------------------------------------------

# settings
t <- 20
x1 <- 100
sample.size <- 2 ^ 12
times <- seq(0, 20, 0.01)
params <- c(100, rep(0,  9))
names(params) <- paste("x", 1:10, sep = "")
```

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

mat <- sensobol::sobol_matrices(matrices = "A", N = sample.size,
                                params = paste("lambda", 1:10, sep = ""), type = "R")
mat <- apply(mat, 2, function(x) KScorrect::qlunif(x, 0.001, 10))
```

```
# RUN DIFFERENTIAL EQUATIONS ---------------------------------------------------

Y <- data.table(ode(y = params,
                    times = times,
                    func = bateman,
                    parms = colMeans(mat)))
```

```
# PLOT BATEMAN DYNAMICS --------------------------------------------------------

plot.bateman <- melt(Y, measure.vars = paste("x", 1:10, sep = "")) %>%
  ggplot(., aes(time, value, color = variable)) +
  geom_line(size = 1) +
  labs(x = expression(italic(t)), y = "$x$") +
```

```
  scale_color_discrete(name = "Model input",
                       labels = paste("$x_{", 1:10, "}$", sep = "")) +
  theme_AP() +
  theme(legend.position = "top")

plot.bateman
```




```
# DEFINE BATEMAN EQUATIONS FOR K ----------------------------------------------

# Define Bateman equations for xk
bateman_fun <- function(x1, lambda, t) {
  out <- list()
  for(i in 1:length(lambda)) {
    out[[i]] <- (lambda / (lambda - lambda[i]))
  }
  alpha <- unlist(lapply(out, function(x) prod(x[!is.infinite(x)])))
  xk <- x1 / lambda[length(lambda)] * sum(lambda * alpha * exp(1) ^ (-lambda * t))
  return(xk)
}

# Wrapper to run bateman equations rowwise
bateman_rowwise <- function(x1, mat, t) {
  out <- vector()
  for(i in 1:nrow(mat)) {
    out[i] <- bateman_fun(x1 = x1, lambda = mat[i, ], t = t)
  }
  return(out)
}
```

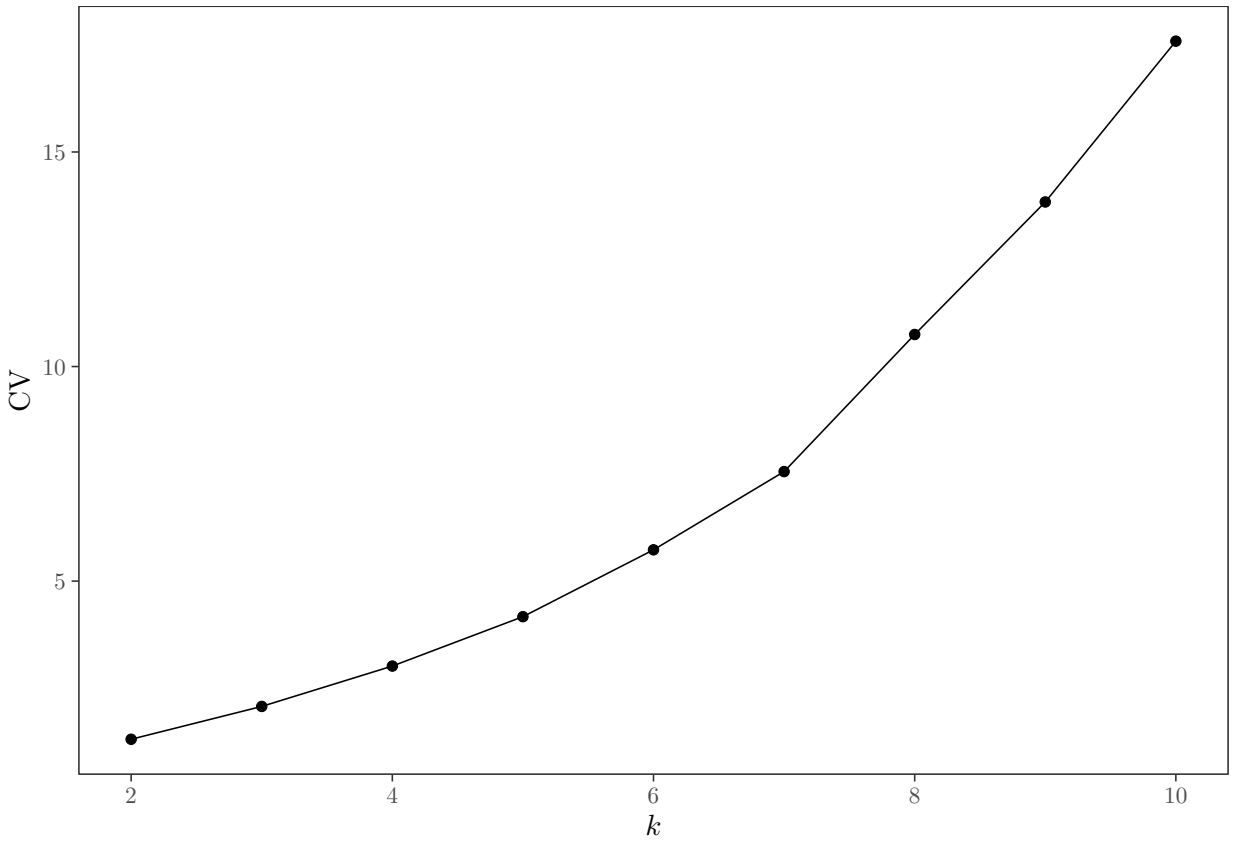
```
# DEFINE SETTINGS -------------------------------------------------------------

t <- 10
x1 <- 100
```

```r
k <- seq(2, t, 1)
sample.size <- 2 ^ 15
matrices <- "A"
```

```r
# RUN BATEMAN IN PARALLEL ------------------------------------------------------

# Define parallel computing
n_cores <- detectCores() * 0.75
cl <- makeCluster(n_cores)
registerDoParallel(cl)

y <- foreach(i = 1:length(k),
             .packages = c("sensobol", "data.table", "KScorrect")) %dopar%
  {
    params <- paste("x", 1:k[i], sep = "")
    set.seed(2)
    tmp <- sensobol::sobol_matrices(matrices = matrices, N = sample.size, params = params,
                                    type = "R")
    mat <- apply(tmp, 2, function(x) KScorrect::qlunif(x, 0.001, 10))
    out <- bateman_rowwise(x1 = x1, mat = mat, t = k[i])
  }

# Stop parallel cluster
stopCluster(cl)
```

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

# Arrange data
tmp <- lapply(y, data.table::data.table)
names(tmp) <- k
dt <- data.table::rbindlist(tmp, idcol = "k")
dt <- dt[, k:= as.numeric(k)]
A <- dt[, .SD[1:(2 * sample.size)], k]

# Compute coefficient of variation and median absolute deviation
dt.stat <- A[, .(cv = sd(V1, na.rm = TRUE) / mean(V1, na.rm = TRUE)), k]
```

```r
# COEFFICIENT OF VARIATION -----------------------------------------------------

cv.bateman <- ggplot(dt.stat, aes(k, cv)) +
  geom_line() +
  geom_point() +
  labs(x = "$k$", y = "CV") +
  theme_AP()

cv.bateman
```

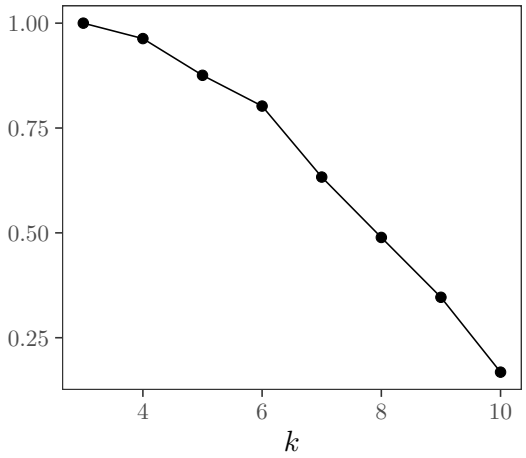



```r
# SENSITIVITY ANALYSIS ------------------------------------------------------

order <- "third"
k <- 3:10
sample.size <- 2 ^ 15
matrices <- c("A", "B", "AB", "BA")
first <- "azzini"
second <- "azzini"
#---------------------------------

# Define parallel computing
n_cores <- detectCores() * 0.75
cl <- makeCluster(n_cores)
registerDoParallel(cl)

ind.bateman <- foreach(i = 1:length(k),
              .packages = c("sensobol", "data.table", "KScorrect")) %dopar%
  {
    params <- paste("X", 1:k[i], sep = "")
    mat <- sensobol::sobol_matrices(matrices = matrices, N = sample.size,
                                    params = params, order = order,
                                    scrambling = 1)
    mat <- apply(mat, 2, function(x) KScorrect::qlunif(x, 0.001, 10))
    y <- bateman_rowwise(x1 = x1, mat = mat, t = k[i])
```

```
    sobol_indices(matrices = matrices, Y = y, N = sample.size,
                  params = params, order = order, first = first,
                  total = total, R = R, boot = TRUE)
  }

# Stop parallel cluster
stopCluster(cl)
```

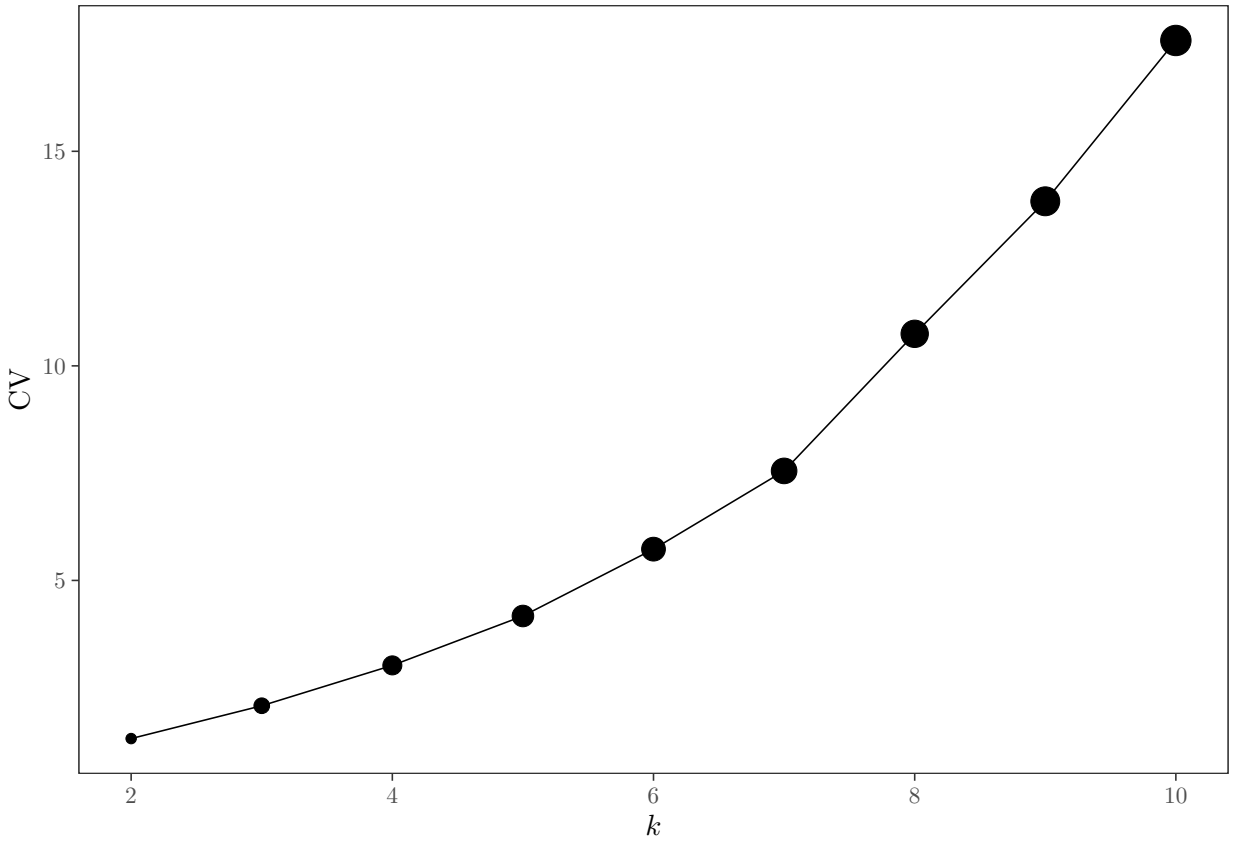
```
# ARRANGE BATEMAN SENSITIVITY ANALYSIS ----------------------------------------

# Compute coefficient of variation and median absolute deviation
bateman.cv <- A[, .(cv = sd(V1, na.rm = TRUE) / mean(V1, na.rm = TRUE)), k]

out <- lapply(ind.bateman, function(x) x$results)
names(out) <- k

ind.bateman <- rbindlist(out, idcol = "dim")
```

```
# SUM UP TO THIRD-ORDER EFFECTS-----------------------------------------------

ind.bateman[!sensitivity == "Ti", sum(original), dim] %>%
  .[, dim:= as.numeric(dim)] %>%
  ggplot(., aes(dim, V1)) +
  geom_line() +
  labs(x = "$k$", y = "") +
  geom_point() +
  theme_AP()
```



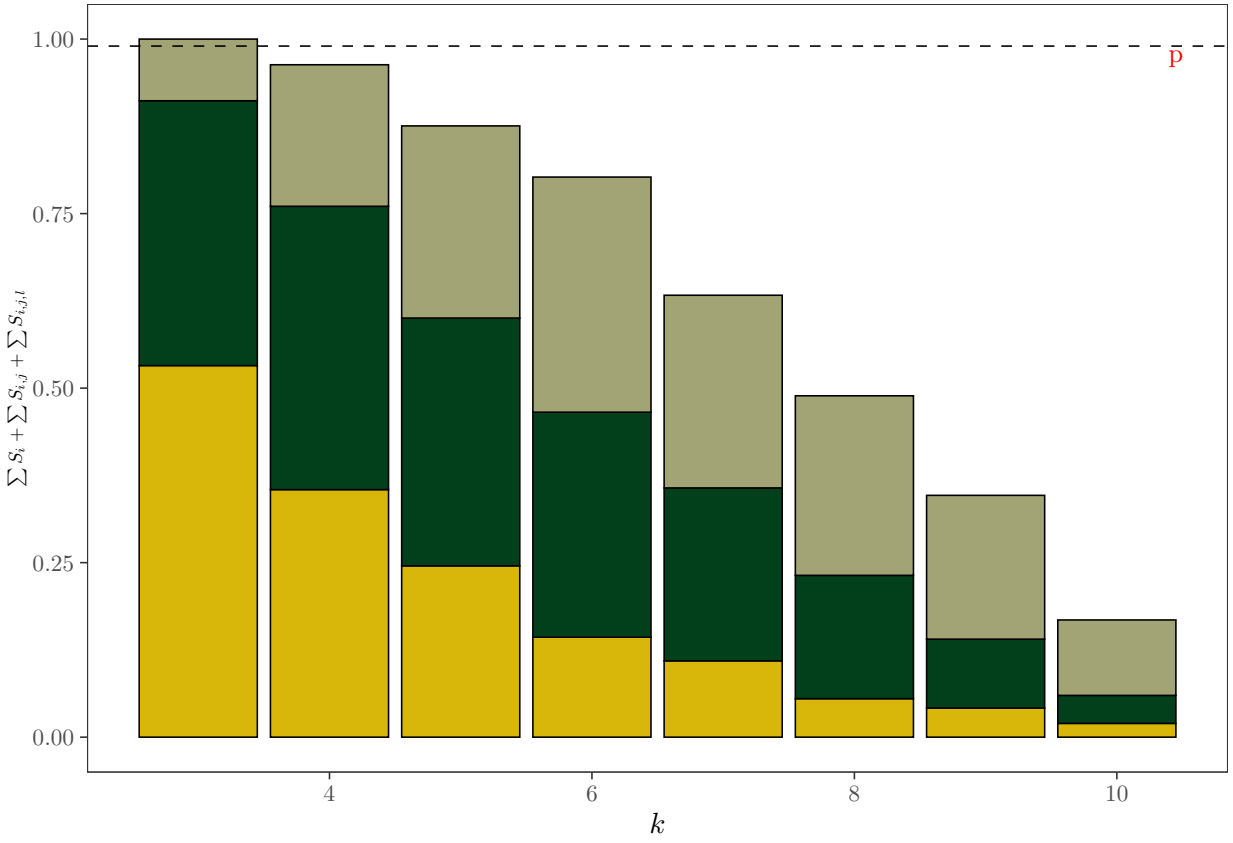

```
# PLOT BATEMAN CV -------------------------------------------------------------

bateman.kt <- ind.bateman[sensitivity == "Ti" & original > 0.05] %>%
  .[, .(kt= length(unique(parameters))), dim] %>%
  setnames(., "dim", "variable") %>%
  .[, variable:= as.numeric(variable)]
```

```r
bateman.cv <- bateman.cv[, kt:= c(2, bateman.kt$kt)]

bateman.cv.plot <- ggplot(bateman.cv, aes(k, cv)) +
  geom_line() +
  geom_point(aes(size = kt)) +
  geom_point() +
  scale_size_continuous(name = "$k_t$", range = c(1, 5)) +
  labs(x = "$k$", y = "CV") +
  theme_AP() +
  theme(legend.position = "none")

legend.kt <- get_legend(bateman.cv.plot + theme(legend.position = "top"))

bateman.cv.plot
```




```r
# PLOT BATEMAN ------------------------------------------------------------------

high.order.bateman <- ind.bateman[!sensitivity == "Ti"] %>%
  .[, sum(original), .(sensitivity, dim)] %>%
  .[, dim:= as.numeric(dim)] %>%
  ggplot(., aes(dim, V1, fill = sensitivity)) +
  geom_col(position = position_stack(reverse = TRUE), color = "black") +
  labs(x = "$k$", y = "$\\sum S_i + \\sum S_{i,j} + \\sum S_{i,j,l}$") +
  geom_hline(yintercept = 0.99, lty = 2) +
```

```
  scale_fill_manual(name = "",
                    labels = c("$S_i$", "$S_{i,j}$", "$S_{i,j,l}$"),
                    values = wes_palette("Cavalcanti1")) +
  annotation_custom(textGrob("p", gp = gpar(col = "red")),
                    xmin = 10.45, xmax = 10.45, ymin = 0.98, ymax = 0.98) +
  theme_AP() +
  theme(legend.position = "none",
        axis.title.y = element_text(size = 7))


high.order.bateman
```



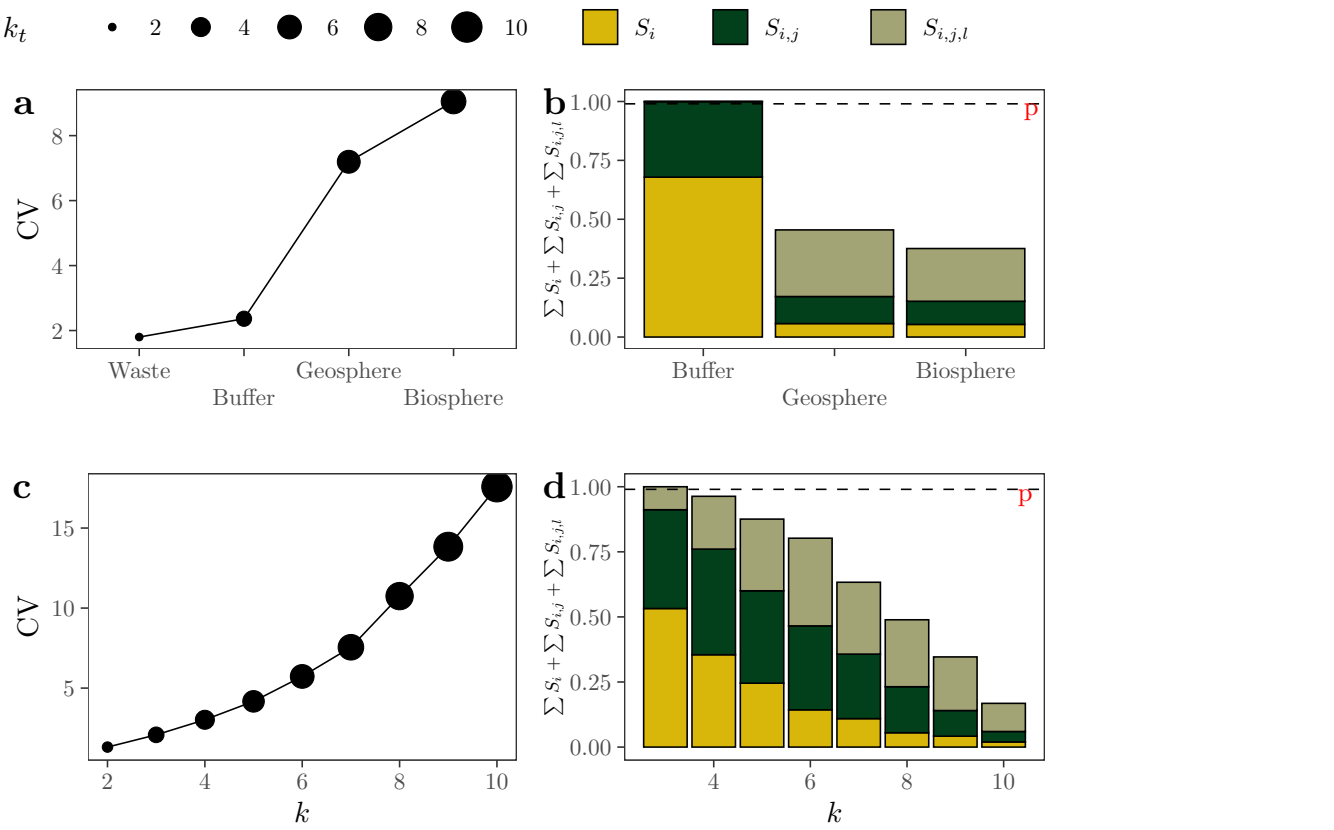

```
ind.bateman[sensitivity == "Ti" & original > 0.05, length(unique(parameters)), dim]
```

```
##    dim V1
## 1:   3  3
## 2:   4  4
## 3:   5  5
## 4:   6  6
## 5:   7  7
## 6:   8  8
## 7:   9  9
## 8:  10 10
```

```
legend.ks <- get_legend(high.order.bateman + theme(legend.position = "top",
                                                   legend.key.size = unit(1, 'lines'),
                                                   legend.spacing.x = unit(0.2, "cm")))
```

```
# PLOT BATEMAN2  ------------------------------------------------------------

bateman.plots <- plot_grid(bateman.cv.plot, high.order.bateman,
                           ncol = 2, labels = c("c", "d"))
```

```
# MERGE ALL PLOTS -----------------------------------------------------------

all.legends <- plot_grid(legend.kt, legend.ks, ncol = 2)
all.plots <- plot_grid(psacoin.plots, bateman.plots, ncol = 1)
plot_grid(all.legends, all.plots, ncol = 1, rel_heights = c(0.1, 0.8))
```




```
# EXPORT BATEMAN SENSITIVITY INDICES ----------------------------------------

fwrite(ind.bateman, "ind.bateman.csv")
```

3 COVID-19 models

```
# PRELIMINARIES ----------------------------------------------------------------

# Define timesteps (5 years)
times <- seq(1, 5 * 52, 1)

# Read input data
R0.listOrig <- fread("NY.csv")
R0.list.a <- R0.listOrig$hku1

shift <- 0
R0.list.shifted <- circshift(R0.list.a, -shift)
R0.list <- rep(R0.list.shifted, length = length(times))
```

```
# COVID MODELS -----------------------------------------------------------------

# SIR(S) with seasonality
sir_s_fun <- function(t, state, parameters) {

  with(as.list(c(state, parameters)), {

    beta <- R0.list[t] * gamma

    dSP <- mu * N - (beta * SP * (IP + alpha * IS) / N) - mu * SP
    dIP <- (beta * SP * (IP + alpha * IS) / N) - (gamma + mu) * IP
    dR <- gamma * (IP + IS) - (delta + mu) * R
    dSS <- delta * R - epsilon * (beta * SS * (IP + alpha * IS) / N) - mu * SS
    dIS <- epsilon * (beta * SS * (IP + alpha * IS) / N) - (gamma + mu) * IS
    list(c(dSP, dIP, dR, dSS, dIS))
  })
}

# SIR(S) with seasonality and vaccination
sir_s_vaccination_fun <- function(t, state, parameters) {

  with(as.list(c(state, parameters)), {

    beta <- R0.list[t] * gamma

    svax <- 0
    if(t > tvax){
      svax <- 1
    }
    dSP <- mu * N - (beta * SP * (IP + alpha * IS) / N) - mu * SP - svax * nu * SP
    dIP <- (beta * SP * (IP + alpha * IS) / N) - (gamma + mu) * IP
    dR <- gamma * (IP + IS) - (delta + mu) * R
    dSS <- delta * R - epsilon * (beta * SS * (IP + alpha * IS) / N) - mu * SS +
```

```
      delta_vax * V - svax * nu * SS
    dIS <- epsilon * (beta * SS * (IP + alpha * IS) / N) - (gamma + mu) * IS
    dV <- svax * nu * (SP + SS) - delta_vax * V - mu * V
    list(c(dSP, dIP, dR, dSS, dIS, dV))
  })
}

# Extendend SIR(S)
sir_s_extended <- function(t, state, parameters) {

  with(as.list(c(state, parameters)), {
    beta <- R0.list[t] * gamma

    svax <- 0
    if(t > tvax){
      svax <- 1
    }

    dSP <- mu - (beta * SP * (IP + alpha * IS + alphaV * IV +
                                alpha1 * IS1 + alpha2 * IS2)) - mu * SP - svax * nu * SP
    dIP <- (beta * SP * (IP + alpha * IS + alphaV *
                           IV + alpha1 * IS1 + alpha2 * IS2)) - (gamma + mu) * IP
    dR <- gamma * (IP + IS + IV + IS1 + IS2) - (delta + mu) * R
    dSS <- delta * R - epsilon * (beta * SS *(IP + alpha * IS + alphaV * IV + alpha1 * IS1 +
                                                alpha2 * IS2)) - mu * SS - svax * nu * SS
    dIS <- epsilon * (beta * SS * (IP + alpha * IS + alphaV *
                                     IV + alpha1 * IS1 + alpha2 * IS2)) - (gamma + mu) * IS
    dV1 <- svax * nu * (SP + d * SS) - epsilonV1 *
      beta * V1 * (IP + alpha * IS + alphaV * IV + alpha1 * IS1 + alpha2 * IS2) -
      (omega + rho1 + mu) * V1
    dV2 <- (1 - d) * svax * nu * SS + omega * V1 - epsilonV2 * beta * V2 *
      (IP + alpha * IS + alphaV * IV + alpha1 * IS1 + alpha2 * IS2) - (rho2 + mu) * V2
    dIV <- (epsilonV1 * beta * V1 + epsilonV2 *
              beta * V2) * (IP + alpha * IS + alphaV *
                              IV + alpha1 * IS1 + alpha2 * IS2) - (gamma + mu) * IV
    dSS1 <- rho1 * V1 - epsilon1 * beta * SS1 *
      (IP + alpha * IS + alphaV * IV + alpha1 * IS1 + alpha2 *IS2) - mu * SS1
    dSS2 <- rho2 * V2 - epsilon2 * beta * SS2 *
      (IP + alpha * IS + alphaV * IV + alpha1 * IS1 + alpha2 * IS2) - mu * SS2
    dIS1 <- epsilon1 * beta * SS1 *
      (IP + alpha * IS + alphaV * IV + alpha1 * IS1 + alpha2 * IS2) - (gamma + mu) * IS1
    dIS2 <- epsilon2 * beta * SS2 *
      (IP + alpha * IS + alphaV * IV + alpha1 * IS1 + alpha2 * IS2) - (gamma + mu) * IS2
    list(c(dSP, dIP, dR, dSS, dIS, dV1, dV2, dIV, dSS1, dSS2, dIS1, dIS2))
  })
}
```

```r
# MERGE ALL MODELS ------------------------------------------------------------

covid_models <- list(sir_s_fun, sir_s_vaccination_fun, sir_s_extended)
type_model <- c("SIR(S)", "SIR(S)vaccination", "SIR(S)extended")
names(covid_models) <- type_model
```

```r
# INITIAL VALUES --------------------------------------------------------------

N <- 1 # Total population
I0 <- 1e-9
SP <- 1 - I0 # Fully susceptible individuals
IP <- I0 # Individuals infected
R <- 0 # Immune individuals due to recovery
SS <- 0 # Individuals whose immunity has waned at rate delta
IS <- 0 # Individuals with secondary infection
V <- 0  # Individuals vaccinated
V1 <- V2 <- IV <- SS1 <- SS2 <- IS1 <- IS2 <- 0

# Define list of initial values
all_y <- list(
  c(SP = SP, IP = IP, R = R, SS = SS, IS = IS),
  c(SP = SP, IP = IP, R = R, SS = SS, IS = IS, V = V),
  c(SP = SP, IP = IP, R = R, SS = SS, IS = IS, V1 = V1,
    V2 = V2, IV = IV, SS1 = SS1, SS2 = SS2, IS1 = IS1, IS2 = IS2)
  )

# Name them
names(all_y) <- type_model
```

```r
# CONSTANTS -------------------------------------------------------------------

gamma <- 7 / 5 # Recovery rate from primary and secondary infections ]
delta <- 1 / (1 * 52) # Wane rate of full immunity from natural infection
mu <- 1 / (50 * 52) # birth rate at which individuals enter the susceptible class SP

d <- 0.5
epsilonV1 <- 0.1 # First level of immune protection
epsilonV2 <- 0.05 # Second level of immune protection
omega <- 0
rho2 <- rho1 <- 0 # Vaccinal immunity wanes at rho1 and rho2
delta_vax <- 1
epsilon1 <- epsilon2 <- 0.7 # Effect of vaccine 1 and 2; fixed
# after observing that their uncertainty (epsilon1=U(0.5, 0.9);
# epsilon2 = U(0.7, 0.9) does not contribute output uncertainty)
```

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

sample.size <- 2^10
```

```r
matrices <- c("A", "B", "AB", "BA")
order <- "third"
first <- "azzini"
total <- "azzini"
R <- 10^3
```

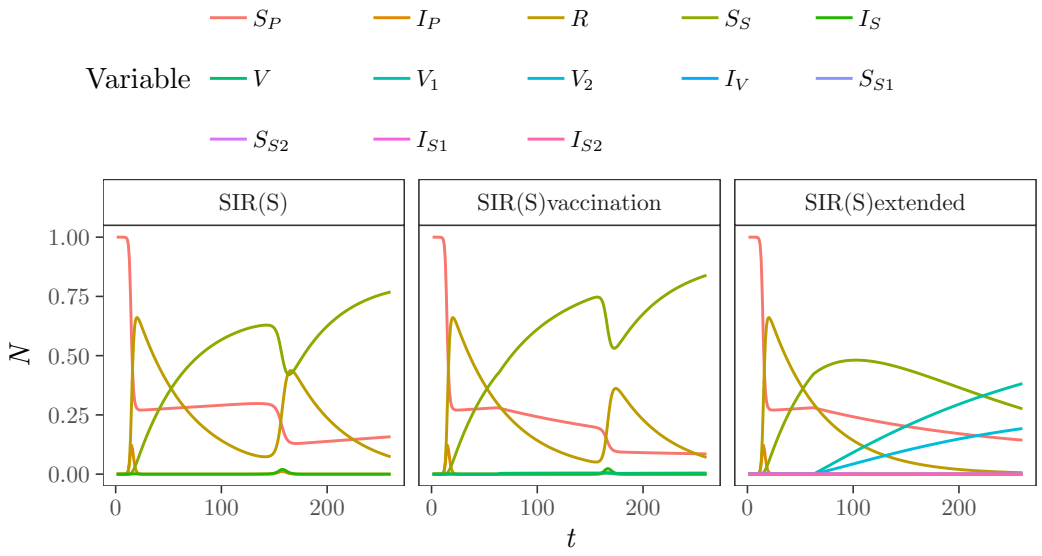
```r
# DEFINE MATRICES ----------------------------------------------------------------

# SIR(S) with seasonality
#-------------------------------------
params <- c("epsilon", "alpha")
# order = second because there are only two uncertain parameters
mat1 <- sobol_matrices(N = sample.size, params = params, scrambling = 1,
                       matrices = matrices, order = "second")

mat1[, "epsilon"] <- qunif(mat1[, "epsilon"], 0.4, 1)
mat1[, "alpha"] <- qunif(mat1[, "alpha"], 0.8, 1)

mat.dt1 <- data.table(mat1)[, `:=` (alpha1 = alpha, alpha2 = alpha, alphaV = alpha)]
mat.dt1 <- cbind(mat.dt1, gamma, delta, mu, epsilonV1, epsilonV2, omega)

# SIR(S) with vaccination
#-------------------------------------
# params <- c("epsilon", "alpha", "nu", "delta_vax", "tvax")
params <- c("epsilon", "alpha", "nu", "tvax")
mat2 <- sobol_matrices(N = sample.size, params = params, scrambling = 1,
                       matrices = matrices, order = order)

mat2[, "epsilon"] <- qunif(mat2[, "epsilon"], 0.4, 1)
mat2[, "alpha"] <- qunif(mat2[, "alpha"], 0.8, 1)
mat2[, "nu"] <- qunif(mat2[, "nu"], 0.001, 0.009)
# mat2[, "delta_vax"] <- qunif(mat2[, "delta_vax"], 0.5, 2)
mat2[, "tvax"] <- floor(qunif(mat2[, "tvax"], 48, 78))

mat.dt2 <- data.table(mat2)[, `:=` (alpha1 = alpha, alpha2 = alpha, alphaV = alpha)]
mat.dt2 <- cbind(mat.dt2, gamma, delta, mu, epsilonV1, epsilonV2, omega)

# SIR(S) extended
#-------------------------------------
# params <- c("epsilon", "alpha", "nu", "delta_vax", "tvax", "epsilon1", "epsilon2")
params <- c("epsilon", "alpha", "nu", "tvax")
mat <- sobol_matrices(N = sample.size, params = params, scrambling = 1,
                      matrices = matrices, order = order)

mat[, "epsilon"] <- qunif(mat[, "epsilon"], 0.4, 1)
mat[, "alpha"] <- qunif(mat[, "alpha"], 0.8, 1)
mat[, "nu"] <- qunif(mat[, "nu"], 0.001, 0.009)
```

```r
# mat[, "delta_vax"] <- qunif(mat[, "delta_vax"], 0.5, 2)
mat[, "tvax"] <- floor(qunif(mat[, "tvax"], 48, 78))
# mat[, "epsilon1"] <- qunif(mat[, "epsilon1"], 0.5, 0.9)
# mat[, "epsilon2"] <- qunif(mat[, "epsilon2"], 0.5, 0.7)

# MERGE ALL MATRICES --------------------

all.mat <- list(mat1, mat2, mat)
all.mat <- lapply(all.mat, data.table) %>%
  lapply(., function(x)
    x[, `:=` (alpha1 = alpha, alpha2 = alpha, alphaV = alpha)]) %>%
  lapply(., function(x) cbind(x, gamma, delta, mu, epsilonV1, epsilonV2, omega,
                              delta_vax, epsilon1, epsilon2))

names(all.mat) <- type_model
```

```r
# RUN MODELS FOR DYNAMICS ------------------------------------------------------

out <- list()
for (i in names(covid_models)) {
  out[[i]] <- data.table(ode(y = all_y[[i]],
                             times = times,
                             func = covid_models[[i]],
                             parms = colMeans(all.mat[[i]])))
}
```

```r
# PLOT DYNAMICS ----------------------------------------------------------------

bottom <- lapply(out, function(x)
  melt(x, measure.vars = colnames(x)[-1])) %>%
  rbindlist(., idcol = "Model") %>%
  .[, Model:= factor(Model, levels = type_model)] %>%
  ggplot(., aes(time, value, color = variable)) +
  geom_line(size = 1) +
  labs(x = expression(italic(t)),
       y = expression(italic(N))) +
  facet_wrap(~ Model) +
  scale_color_discrete(name = "Variable",
                       labels = c("$S_P$", "$I_P$", "$R$", "$S_S$",
                                  "$I_S$", "$V$", "$V_1$", "$V_2$",
                                  "$I_V$", "$S_{S1}$", "$S_{S2}$", "$I_{S1}$",
                                  "$I_{S2}$")) +
  theme_AP() +
  theme(legend.position = "none")

legend <- get_legend(bottom + theme(legend.position = "top",
                                    legend.text = element_text(
    margin = margin(l = -5, unit = "pt"))) +
```

```
                        guides(color = guide_legend(nrow = 3, byrow=TRUE)))
dynamics.covid <- plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.3, 0.7))

dynamics.covid
```

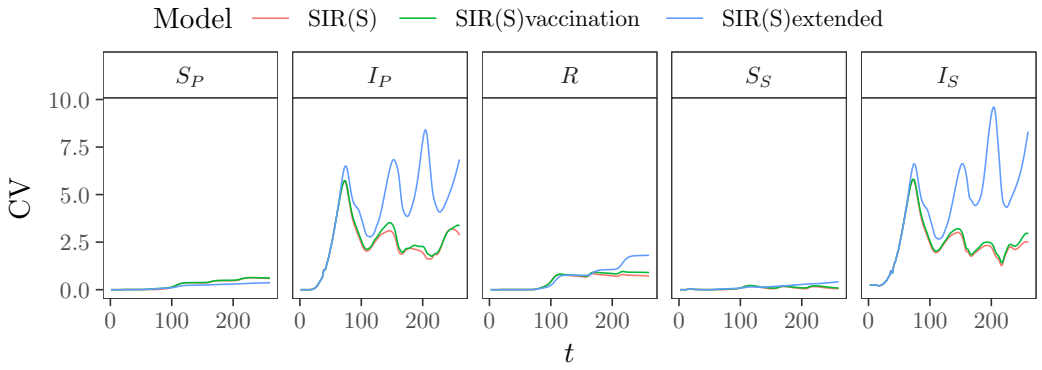



```
# RUN MODEL IN PARALLEL ------------------------------------------------------

# Define parallel computing
n_cores <- makeCluster(floor(detectCores() * 0.75))
registerDoParallel(n_cores)

y <- foreach(j = type_model) %:%
  foreach(i = 1:nrow(all.mat[[j]]), .combine = "rbind",
          .packages = "deSolve") %dopar%
  {
    .GlobalEnv$R0.list <- R0.list
    .GlobalEnv$N <- N
    .GlobalEnv$d <- d
    .GlobalEnv$rho1 <- rho1
    .GlobalEnv$rho2 <- rho2
    ode(y = all_y[[j]],
        times = times,
        func = covid_models[[j]],
        parms = all.mat[[j]][i, ])
  }

# Stop clusters
stopCluster(n_cores)
```

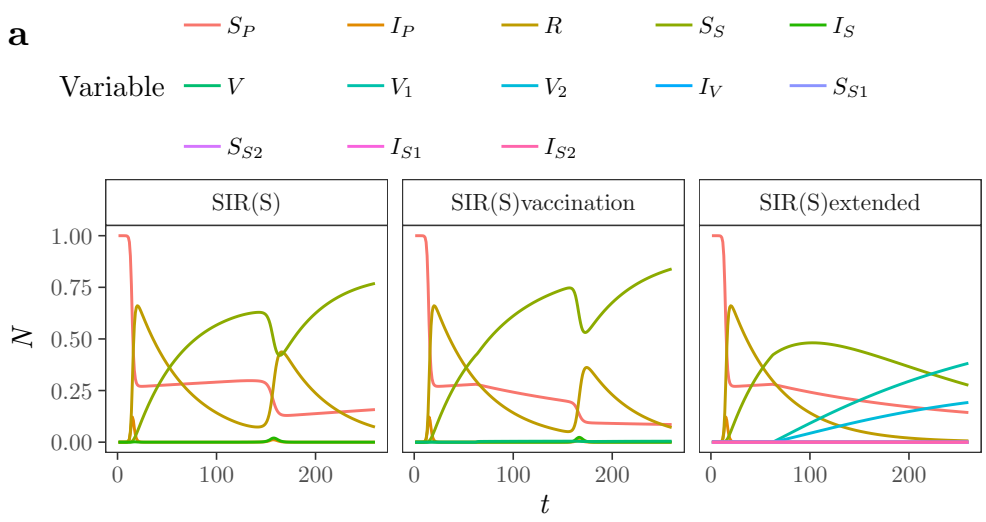
```
# ARRANGE DATA ---------------------------------------------------------------

common.compartments <- c("SP", "IP", "R", "SS", "IS")
```

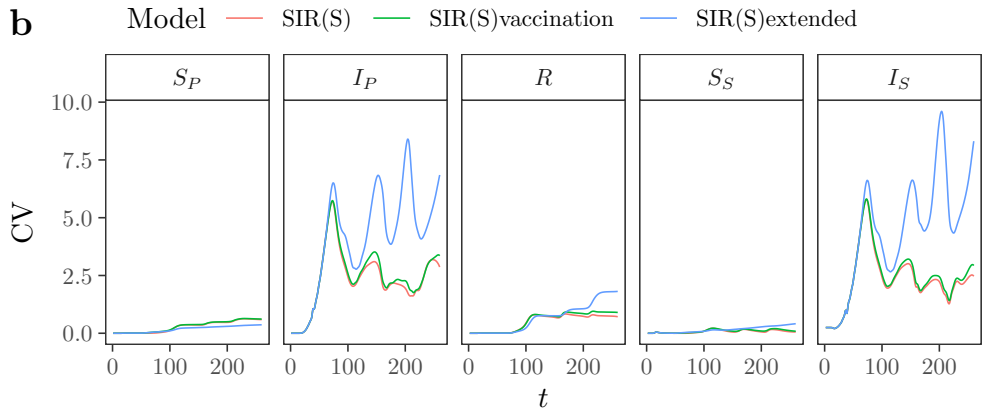
```r
dt.covid <- lapply(y, function(x) data.table(x) %>%
                     .[, .SD, .SDcols = c("time", common.compartments)])

names(dt.covid) <- type_model

# Dataset for uncertainty analysis
A <- rbindlist(dt.covid, idcol = "model") %>%
  .[, .SD[1:((sample.size * 2) * length(times))], model] %>%
  .[, model:= factor(model, levels = type_model)] %>%
  melt(., measure.vars = common.compartments)

# Full dataset
out <- rbindlist(dt.covid, idcol = "model") %>%
  melt(., measure.vars = common.compartments) %>%
  .[, model:= factor(model, levels = type_model)]
```

```r
# PLOT CV ----------------------------------------------------------------------

state_names <- c(
  "SP" = "$S_P$",
  "IP" = "$I_P$",
  "R" = "$R$",
  "SS" = "$S_S$",
  "IS" = "$I_S$"
  )

covid.cv <- A[, .(cv = sd(value) / mean(value)), .(model, time, variable)] %>%
  ggplot(., aes(time, cv, group = model, color = model)) +
  geom_line() +
  facet_wrap(~variable, ncol = 5, labeller = as_labeller(state_names)) +
  labs(x = "$t$", y = "CV") +
  theme_AP() +
  scale_color_discrete(name = "Model") +
  theme(legend.position = "none")

legend <- get_legend(covid.cv + theme(legend.position = "top"))
cv.covid <- plot_grid(legend, covid.cv, ncol = 1, rel_heights = c(0.1, 0.9))

cv.covid
```




```
# MERGE DYNAMICS AND CV COVID MODEL ------------------------------------------

plot_grid(dynamics.covid, cv.covid, labels = "auto", ncol = 1,
          rel_heights = c(0.55, 0.45))
```





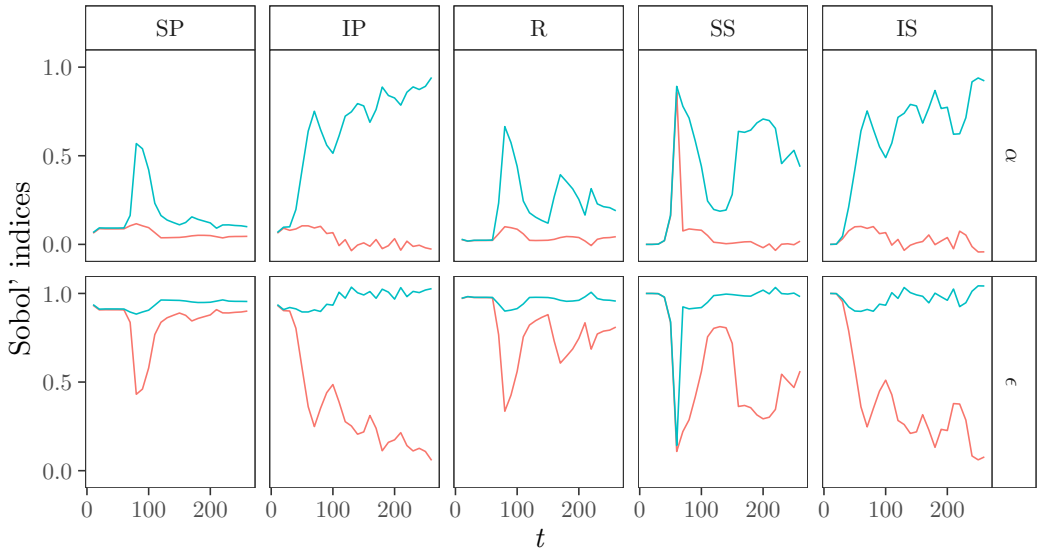

```
# SENSITIVITY ANALYSIS -------------------------------------------------------

timeSteps <- seq(10, 260, 10)

prove <- lapply(dt.covid, function(x)
  melt(x, measure.vars = common.compartments) %>%
```

```
    .[time %in% timeSteps])

names(prove) <- type_model

ind <- list()
for(i in names(prove)) {
  if(i == "SIR(S)") {
    params <- c("$\\epsilon$", "$\\alpha$")
    order <- "second"
  } else if(i == "SIR(S)vaccination") {
    params <- c("$\\epsilon$", "$\\alpha$", "$\\nu$", "$t_{vax}$")
    order <- "third"
  } else if(i == "SIR(S)extended") {
    params <- c("$\\epsilon$", "$\\alpha$", "$\\nu$", "$t_{vax}$")
    order <- "third"
  }
 ind[[i]] <- prove[[i]][, sobol_indices(Y = value, N = sample.size, matrices = matrices,
                                        params = params, order = order,
                                        first = first, total = total, boot = TRUE,
                                        R = R, parallel = "multicore",
                                        ncpus = detectCores() * 0.75)$results,
             .(time, variable)]
}
```

```
## [1] "All values of t are equal to  0.999999999960785 \n Cannot calculate confidence interval
## [1] "All values of t are equal to  0.999999999967852 \n Cannot calculate confidence interval
## [1] "All values of t are equal to  0.999999999960825 \n Cannot calculate confidence interval
## [1] "All values of t are equal to  0.99999999996784 \n Cannot calculate confidence intervals
## [1] "All values of t are equal to  0.999999999960821 \n Cannot calculate confidence interval
## [1] "All values of t are equal to  0.999999999967854 \n Cannot calculate confidence interval
```

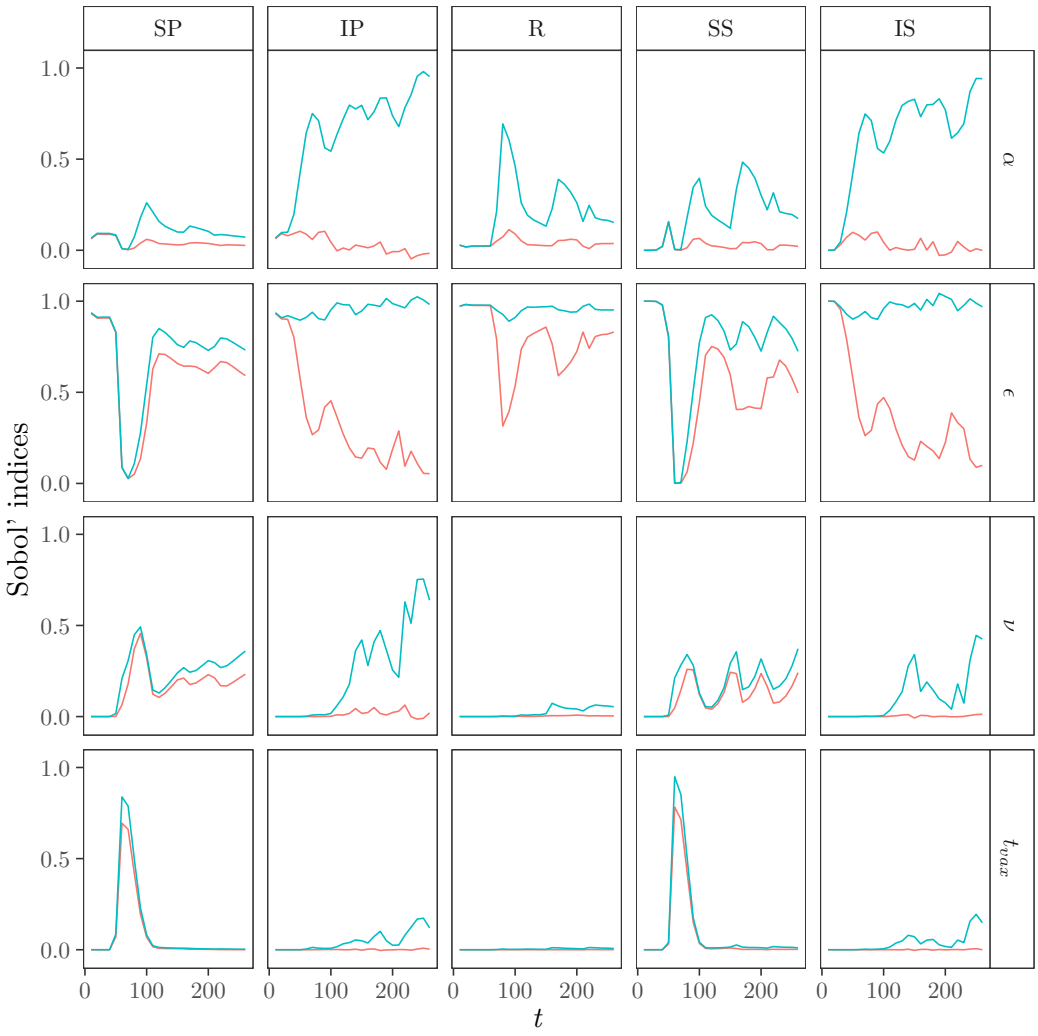
```
# PLOT SENSITIVITY INDICES ------------------------------------------------------

plot.ind <- list()
for(i in names(ind)) {
  plot.ind[[i]] <- ggplot(ind[[i]][sensitivity %in% c("Si", "Ti")],
                          aes(time, original, fill = sensitivity,
                              color = sensitivity, group = sensitivity)) +
    geom_line() +
    guides(linetype = FALSE, color = FALSE) +
    facet_grid(parameters ~ variable) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 3)) +
    labs(x = expression(italic(t)),
         y = "Sobol' indices") +
    theme_AP() +
    theme(legend.position = "top")
}
```

```
plot.ind[[1]]
```

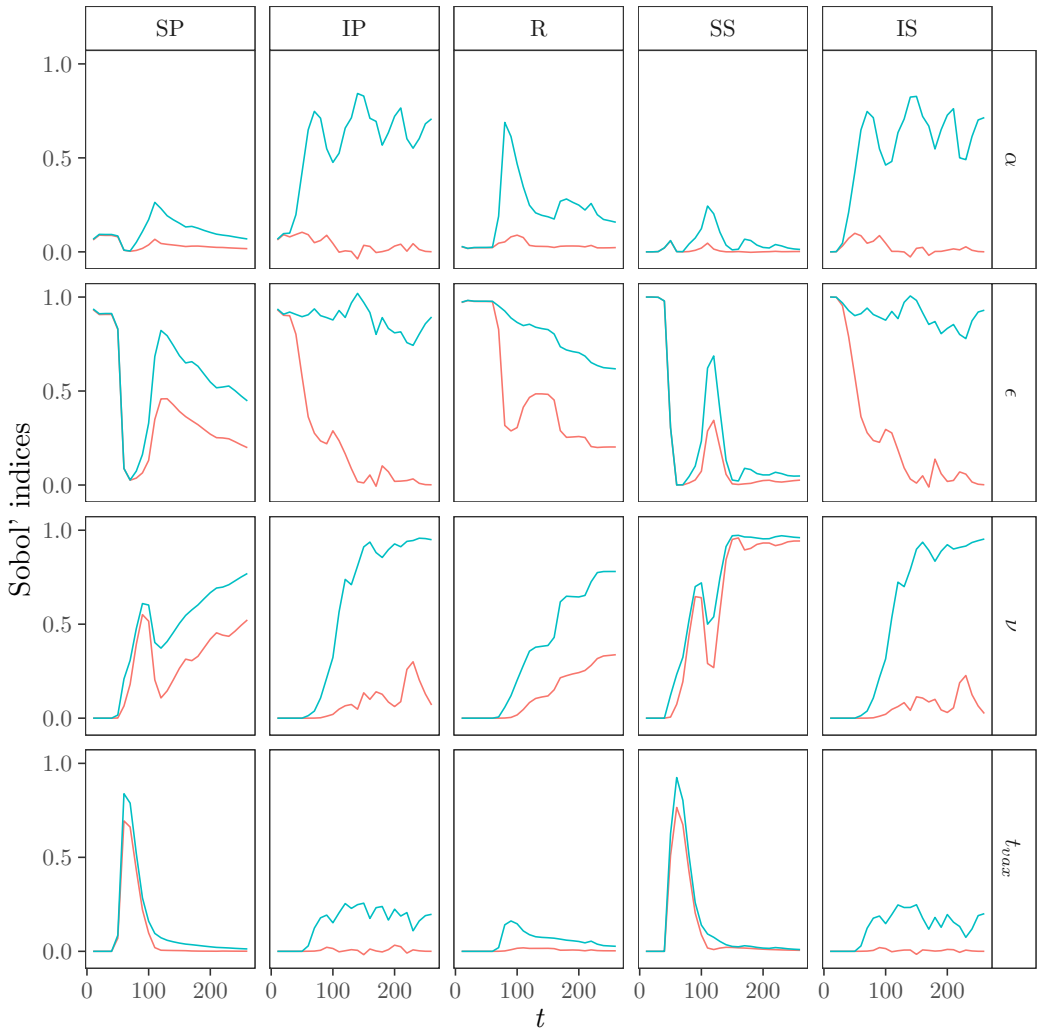



```
# PLOT SENSITIVITY INDICES ------------------------------------------------------

plot.ind <- list()
for(i in names(ind)) {
  plot.ind[[i]] <- ggplot(ind[[i]][sensitivity %in% c("Si", "Ti")],
                          aes(time, original, fill = sensitivity,
                              color = sensitivity, group = sensitivity)) +
    geom_line() +
    guides(linetype = FALSE, color = FALSE) +
    facet_grid(parameters ~ variable) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 3)) +
    labs(x = expression(italic(t)),
         y = "Sobol' indices") +
    theme_AP() +
    theme(legend.position = "top")
}

lapply(2:3, function(x) plot.ind[[x]])
```

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



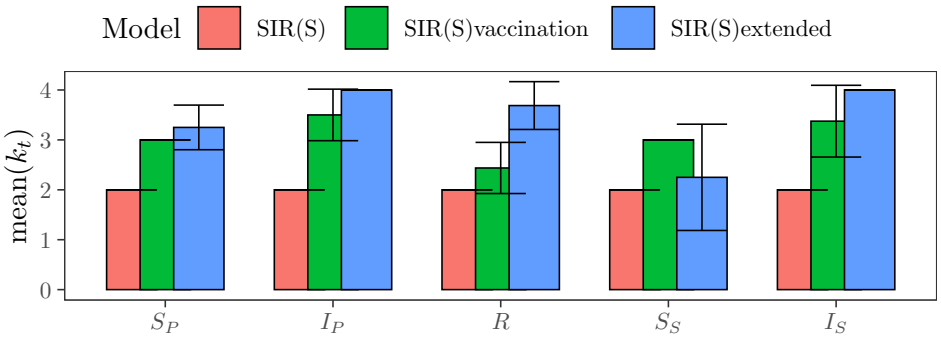

```
##
## [[2]]
```




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

covid.indices <- rbindlist(ind, idcol = "model")
fwrite(covid.indices, "ind.covid.csv")
```

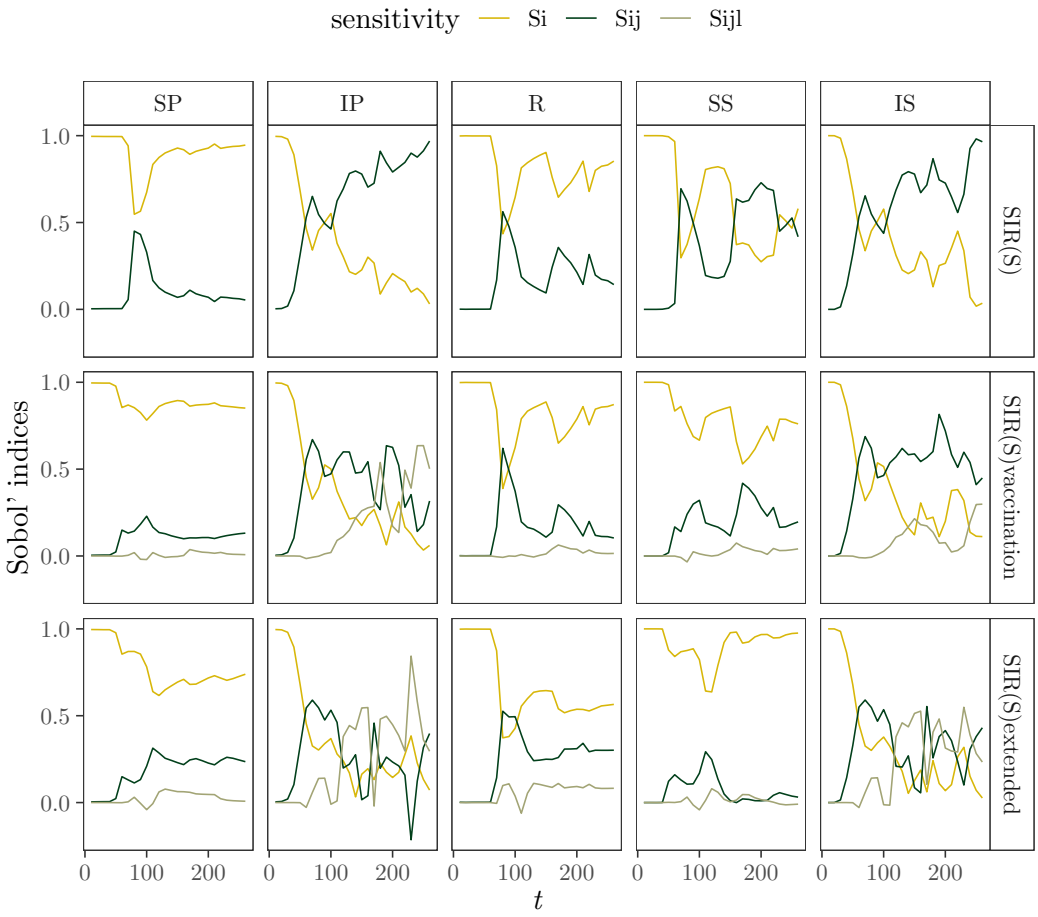
```
# PLOT COVID KT -----------------------------------------------------------------

x.labels <- c("$S_P$", "$I_P$", "$R$", "$S_S$", "$I_S$")

covid.kt <- covid.indices[sensitivity == "Ti" & original > 0.05,
                          length(unique(parameters)), .(model, time, variable)] %>%
  .[time > 100, .(mean = mean(V1), sd = sd(V1)), .(model, variable)] %>%
  .[, model:= factor(model, levels = type_model)] %>%
  ggplot(., aes(variable, mean, fill = model)) +
  geom_bar(stat = "identity", position = position_dodge(0.6), color = "black") +
  geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd),
                position = position_dodge(0.6)) +
  scale_fill_discrete(name = "Model") +
  scale_x_discrete(labels = x.labels) +
```

```
  theme_AP() +
  labs(x = "", y = "mean($k_t$)") +
  theme(legend.position = "none")

legend.covid.kt <- get_legend(covid.kt + theme(legend.position = "top"))
plot.covid.kt <- plot_grid(legend.covid.kt, covid.kt, ncol = 1, rel_heights = c(0.15, 0.85))
plot.covid.kt
```



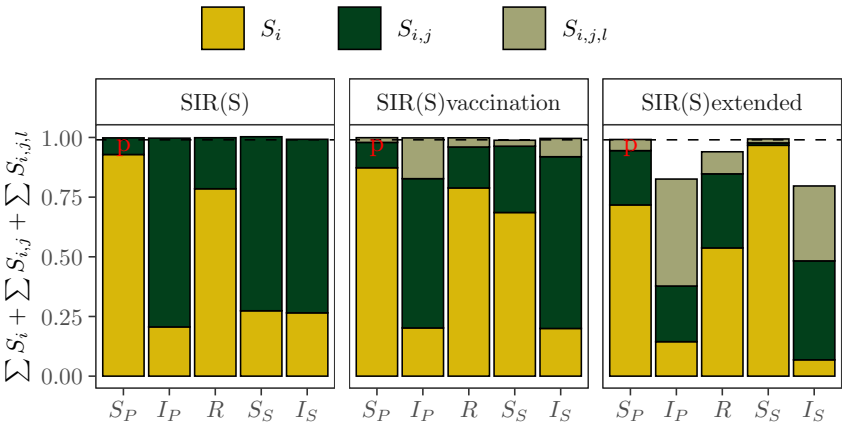

```
# PLOT COVID CV AND COVID KT ----------------------------------------------------

covid.cv.kt <- plot_grid(cv.covid, plot.covid.kt, labels = "auto", ncol = 1)
```

```
# COVID SUM FIRST, SECOND AND THIRD -ORDER ---------------------------------------

covid.indices %>%
  .[!sensitivity == "Ti"] %>%
  .[, sum(original), .(sensitivity, time, variable, model)] %>%
  .[, model:= factor(model, levels = type_model)] %>%
  ggplot(., aes(time, V1, color = sensitivity, group = sensitivity)) +
  geom_line() +
  facet_grid(model ~ variable) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 3)) +
  scale_color_manual(values = wes_palette("Cavalcanti1")) +
  labs(x = expression(italic(t)),
       y = "Sobol' indices") +
  theme_AP() +
  theme(legend.position = "top")
```

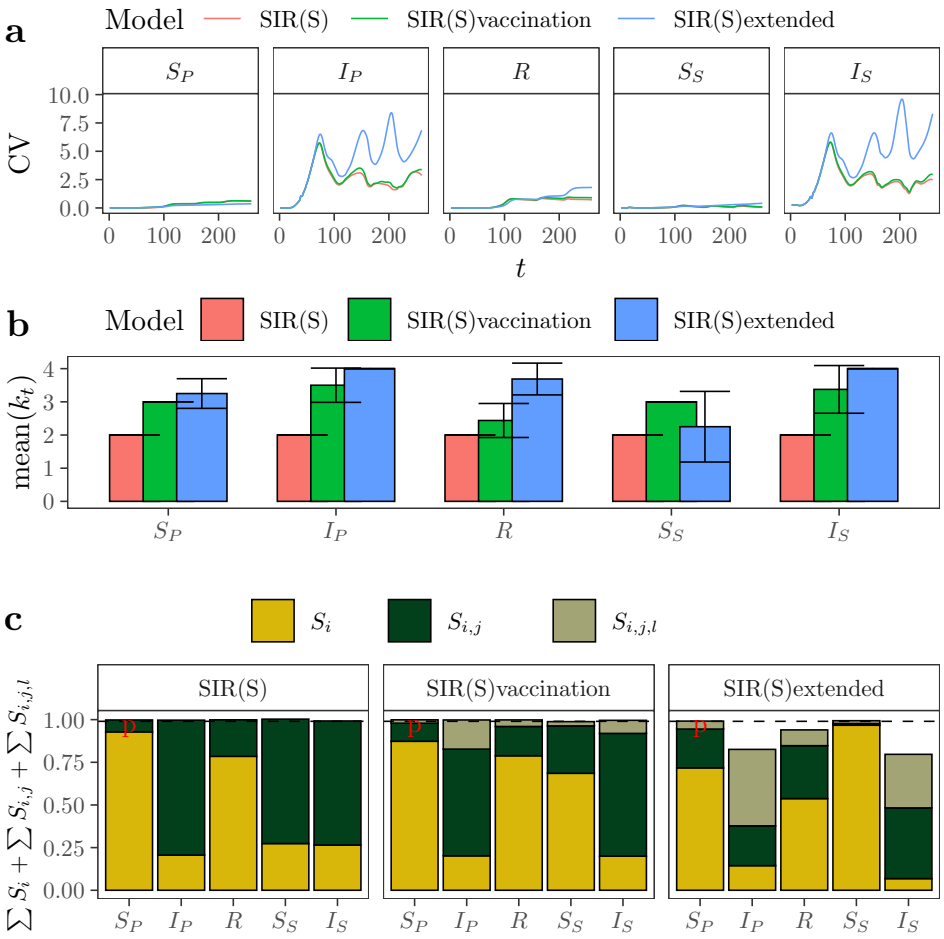



```
# PLOT SUM OF SI, SIJ AND SIJL AT T = 200 ----------------------------------------

time.selected <- 200

covid.ks <-
  rbindlist(ind, idcol = "model") %>%
  .[time == time.selected] %>%
  .[!sensitivity == "Ti"] %>%
  .[, sum(original), .(model, variable, sensitivity)] %>%
  .[, model:= factor(model, levels = type_model)] %>%
  ggplot(., aes(variable, V1, fill = sensitivity)) +
  geom_col(position = position_stack(reverse = TRUE), color = "black") +
  labs(x = "", y = "$\\sum S_i + \\sum S_{i,j} + \\sum S_{i,j,l}$") +
  geom_hline(yintercept = 0.99, lty = 2) +
  scale_fill_manual(name = "",
                    labels = c("$S_i$", "$S_{i,j}$", "$S_{i,j,l}$"),
                    values = wes_palette("Cavalcanti1")) +
  annotation_custom(textGrob("p", gp = gpar(col = "red")),
                    xmin = 1, xmax = 1, ymin = 0.98, ymax = 0.98) +
  facet_wrap(~model) +
  scale_x_discrete(labels = c("SP" = "$S_P$",
                              "IP" = "$I_P$",
```

```
                                "R" = "$R$",
                                "SS" = "$S_S$",
                                "IS" = "$I_S$")) +
  theme_AP() +
  theme(legend.position = "none",
        axis.title.y = element_text(size = 9))

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

covid.ks <- plot_grid(legend.covid.ks, covid.ks, ncol = 1, rel_heights = c(0.15, 0.85))

covid.ks
```




```
plot_grid(covid.cv.kt, covid.ks, ncol = 1,
          labels = c("", "c"), rel_heights = c(0.6, 0.4))
```



4 A model on irrigation water withdrawals

```
# DEFINE THE MODELS ------------------------------------------------------------

run_model <- function(trigger, output, mat) {

  # Priestley Taylor
  if(trigger == "priestley") {
    et0 <- mat[, "alpha"] * ((mat[, "Delta"] * mat[, "A"]) / (mat[, "gamma"] + mat[, "Delta"]))

    # Penman Monteith
  } else if(trigger == "penman") {
    et0 <- (0.408 * mat[, "Delta"] * mat[, "A"] +
              mat[, "gamma"] * (900 / (mat[, "T_a"] + 273)) * mat[, "w"] * mat[, "v"]) /
      mat[, "Delta"] + mat[, "gamma"] * (1 + 0.34 * mat[, "w"])
  }

  if(output == "et0") {
    final <- et0
```

```
  } else if(output == "etc") {
    final <- mat[, "k_c"] * et0

  } else if(output == "water") {
    final <- (mat[, "I_a"] * (mat[, "k_c"] * et0 - mat[, "P"])) /
      (mat[, "E_a"] * mat[, "E_c"] * mat[, "M_f"])
    final <- final / 10^3 # Output is in m3 ha
  }

  return(final)
}

# Model to run rowwise
run_model_rowwise <- function(I_a, Delta, A, gamma, T_a, w, v, output,
                        k_c, P, E_a, E_c, M_f, alpha, X1) {

  # Priestley Taylor
  if(X1 == 1) {
    et0 <- alpha * ((Delta * A) / (gamma + Delta))

    # Penman Monteith
  } else if(X1 == 2) {
    et0 <- (0.408 * Delta * A + gamma * (900 / (T_a + 273)) * w * v) /
      Delta + gamma * (1 + 0.34 * w)

  } else {
    stop("trigger should be either 1 or 2")
  }

  crop.reference <- k_c * et0
  water.withdrawal <- (I_a * (crop.reference - P)) / (E_a * E_c * M_f)

  if(output == "et0") {
    final <- et0
  } else if(output == "etc") {
    final <- crop.reference
  } else if(output == "water") {
    final <- water.withdrawal / 10^3 # Output is in m3 ha
  }
  return(final)
}


# FUNCTION TO CREATE MATRICES ----------------------------------------

create_matrices <- function(trigger, output) {
```

```r
  if(trigger == "priestley") {
    params <- c("alpha", "Delta", "A", "gamma")

  } else if(trigger == "penman") {
    params <- c("Delta", "A", "gamma", "T_a", "w", "v")

  } else if(trigger == "all") {
    params <- c("Delta", "A", "gamma", "T_a", "w", "v", "alpha", "X1")
  }

  if(output == "et0") {
    params <- params

  } else if(output == "etc") {
    params <- c(params, "k_c")

  } else if(output == "water") {
    params <- c(params, "k_c", "I_a", "E_a", "E_c", "M_f", "P")
  }

  mat <- sobol_matrices(N = sample.size, params = params, order = order,
                        matrices = matrices, scrambling = 1)
  return(mat)
}


# FUNCTION TO TRANSFORM MATRICES -------------------------------------

Delta <- 0.08725467
Gamma <- 0.06658597
T_air <- 11
v <- 0.4115359

transform_matrices <- function(trigger, mat, output) {

  if(trigger == "priestley") {
    mat[, "alpha"] <- qunif(mat[, "alpha"], 1.13, 1.38)
    mat[, "Delta"] <- qunif(mat[, "Delta"], Delta + Delta * -0.005, Delta + Delta * 0.005)
    mat[, "gamma"] <- qunif(mat[, "gamma"], Gamma + Gamma * -0.001, Gamma + Gamma * 0.001)
    mat[, "A"] <- qunif(mat[, "A"], 350 + 350 * -0.15, 350 + 350 * 0.15)

  } else if(trigger == "penman") {
    mat[, "Delta"] <- qunif(mat[, "Delta"], Delta + Delta * -0.005, Delta + Delta * 0.005)
    mat[, "gamma"] <- qunif(mat[, "gamma"], Gamma + Gamma * -0.001, Gamma + Gamma * 0.001)
    mat[, "A"] <- qunif(mat[, "A"], 350 + 350 * -0.15, 350 + 350 * 0.15)
    mat[, "T_a"] <- qunif(mat[, "T_a"], T_air + T_air * -0.01, T_air + T_air * 0.01)
    mat[, "w"] <- qunif(mat[, "w"], 2.81 + 2.81 * -0.05, 2.81 + 2.81 * 0.05)
```

```
    mat[, "v"] <- qunif(mat[, "v"], v + v * -0.04, v + v * 0.04)

  } else if(trigger == "all") {
    mat[, "Delta"] <- qunif(mat[, "Delta"], Delta + Delta * -0.005, Delta + Delta * 0.005)
    mat[, "gamma"] <- qunif(mat[, "gamma"], Gamma + Gamma * -0.001, Gamma + Gamma * 0.001)
    mat[, "A"] <- qunif(mat[, "A"], 350 + 350 * -0.15, 350 + 350 * 0.15)
    mat[, "T_a"] <- qunif(mat[, "T_a"], T_air + T_air * -0.01, T_air + T_air * 0.01)
    mat[, "w"] <- qunif(mat[, "w"], 2.81 + 2.81 * -0.05, 2.81 + 2.81 * 0.05)
    mat[, "v"] <- qunif(mat[, "v"], v + v * -0.04, v + v * 0.04)
    mat[, "alpha"] <- qunif(mat[, "alpha"], 1.13, 1.38)
    mat[, "X1"] <- floor(mat[, "X1"] * (2 - 1 + 1)) + 1

  }

  if(output == "et0") {
    mat <- mat

  } else if(output == "etc") {
    mat[, "k_c"] <- qunif(mat[, "k_c"], 0.4564315, 1.144222)

  } else if(output == "water") {
    mat[, "k_c"] <- qunif(mat[, "k_c"], 0.4564315, 1.144222)
    mat[, "I_a"] <- qunif(mat[, "I_a"], 42.932, 144.5515)
    mat[, "E_a"] <- qunif(mat[, "E_a"], 0.49, 0.88)
    mat[, "E_c"] <- qunif(mat[, "E_c"], 0.64, 0.96)
    mat[, "M_f"] <- qunif(mat[, "M_f"], 0.5, 0.97)
    mat[, "P"] <- qunif(mat[, "P"], 0, 0.1)

  }
  return(mat)
}

# MERGE ALL FUNCTIONS ----------------------------------------------

full_model <- function(trigger, output) {
  mat <- create_matrices(trigger = trigger, output = output)
  mat <- transform_matrices(trigger = trigger, mat = mat, output = output)
  out <- run_model(trigger = trigger, output = output, mat = mat)
  return(out)
}
```

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

sample.size <- 2^12
order <- "third"
matrices <- c("A", "B", "AB")
first <- "saltelli"
```

```r
total <- "jansen"
R <- 10^3
```

```r
# RUN MODELS ------------------------------------------------------------------

et0.formulae <- c("priestley", "penman")
outputs <- c("et0", "etc", "water")

# Define parallel computing
n_cores <- makeCluster(floor(detectCores() * 0.75))
registerDoParallel(n_cores)

y <- foreach(j = et0.formulae) %:%
  foreach(l = outputs,
          .packages = "sensobol") %dopar%
  {
    full_model(trigger = j, output = l)
  }

# Stop parallel computing
stopCluster(n_cores)
```

```r
# RUN MODELS (all) ------------------------------------------------------------
trigger <- "all"
output <- "water"

mat <- create_matrices(trigger = trigger, output = output)
mat <- data.table(transform_matrices(trigger = trigger, mat = mat, output = output))

# Define parallel computing
n_cores <- makeCluster(floor(detectCores() * 0.75))
registerDoParallel(n_cores)

y.all <- foreach(j = c("et0", "etc", "water")) %:%
  foreach(i = 1:nrow(mat), .combine = "rbind") %dopar%
  {
    run_model_rowwise(I_a = mat[i, "I_a"],
                Delta = mat[i, "Delta"],
                A = mat[i, "A"],
                gamma = mat[i, "gamma"],
                T_a = mat[i, "T_a"],
                w = mat[i, "w"],
                v = mat[i, "v"],
                k_c = mat[i, "k_c"],
                P = mat[i, "P"],
                E_a = mat[i, "E_a"],
                E_c = mat[i, "E_c"],
                alpha = mat[i, "alpha"],
```

```
                M_f = mat[i, "M_f"],
                X1 = mat[i, "X1"],
                output = j)
  }

# Stop parallel computing
stopCluster(n_cores)
```

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

for(i in 1:2) {
  names(y[[i]]) <- outputs
}

names(y) <- et0.formulae
names(y.all) <- outputs
```

```
# CREATION OF THE A MATRIX ---------------------------------------------------

# First round of simulations
A <- list()
for(i in names(y)) {
  for(j in names(y[[i]])) {
    A[[i]][[j]] <- data.table(y[[i]][[j]][1:(2 * sample.size)])
  }
}

A <- lapply(A, function(x) rbindlist(x, idcol = "output")) %>%
  rbindlist(., idcol = "et0.formulae")

# Second round of simulations (all)
out.all <- lapply(y.all, function(x) data.table(x)) %>%
  rbindlist(., idcol = "output") %>%
  .[, et0.formulae:= "all"]

out.all <- setcolorder(out.all, c("et0.formulae", "output", "V1"))

A.all <- out.all[, .SD[1:(sample.size * 2)], output]

A.water <- rbind(A, A.all)
```

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

ind <- list()

for(i in names(y)) {
  for(j in names(y[[i]])) {
    if(i == "priestley") {
```

```
      params <- c("alpha", "Delta", "A", "gamma")
      } else if(i == "penman") {
        params <- c("Delta", "A", "gamma", "T_a", "w", "v")
      }
    if(j == "et0") {
      params <- params
    } else if(j == "etc") {
      params <- c(params, "k_c")
    } else if(j == "water") {
      params <- c(params, "k_c", "I_a", "E_a", "E_c", "M_f", "P")
    }
    ind[[i]][[j]] <- sobol_indices(matrices = matrices, Y = y[[i]][[j]],
                                   N = sample.size, params = params, order = order,
                                   first = first, total = total,
                                   R = R, boot = TRUE, parallel = "multicore")$results
  }
}

ind.all <- out.all[, sobol_indices(matrices = matrices,
                                   Y = V1, params = colnames(mat), N = sample.size,
                                   order = order, R = R, boot = TRUE, parallel = "multicore",
                                   first = first, total = total)$results, .(et0.formulae, outpu
```

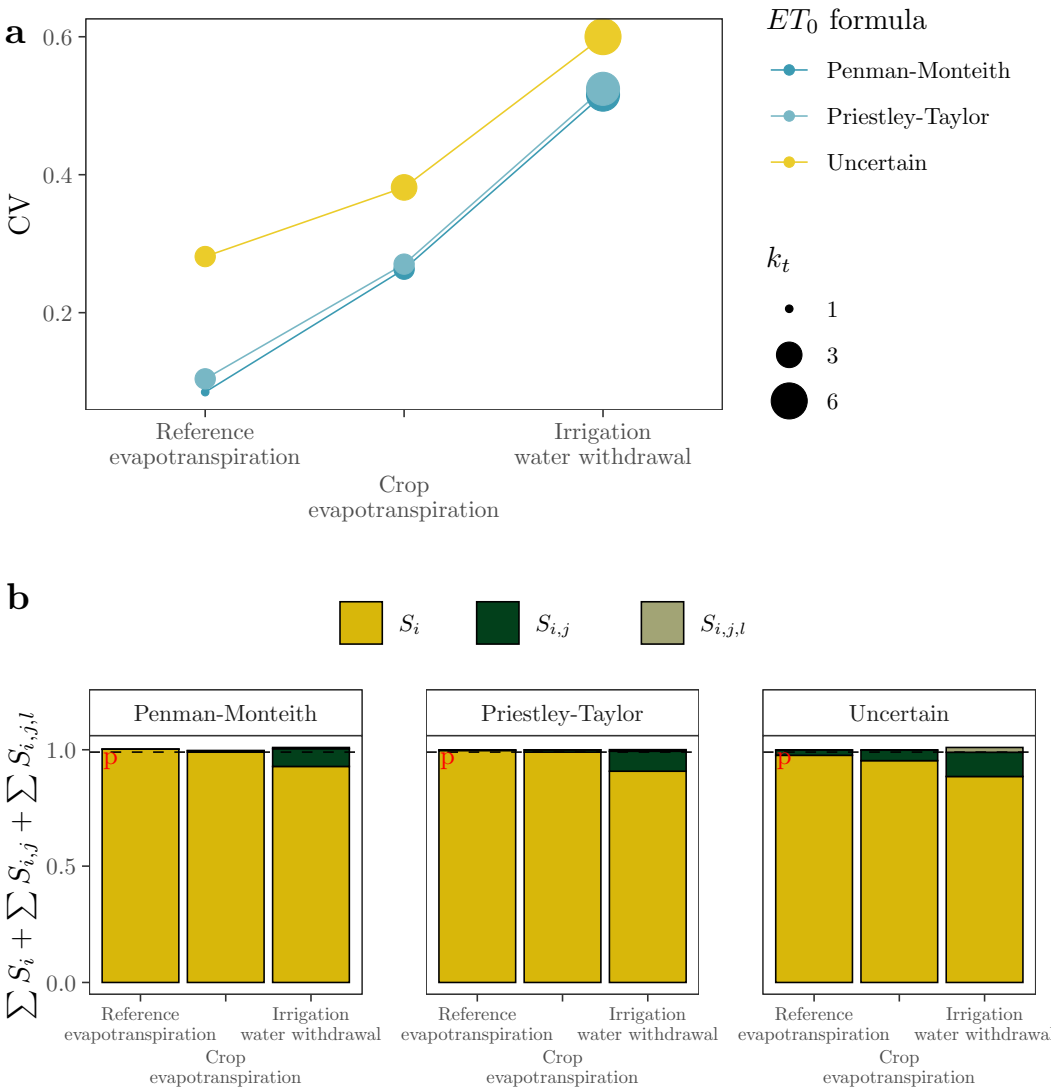
```
# ARRANGE SENSITIVITY DATA ------------------------------------------

ind.dt <- lapply(ind, function(x) rbindlist(x, idcol = "output")) %>%
  rbindlist(., idcol = "et0.formulae")

# merge
all.ind <- rbind(ind.dt, ind.all)[, original:= ifelse(original < 0, 0, original)]
```

```
# CREATE PLOTS ------------------------------------------------------

water.kt <- all.ind[sensitivity == "Ti" & original > 0.05] %>%
  .[, .(kt = length(unique(parameters))), .(et0.formulae, output)]

water.kt
```

```
##    et0.formulae output kt
## 1:    priestley    et0  2
## 2:    priestley    etc  2
## 3:    priestley  water  5
## 4:       penman    et0  1
## 5:       penman    etc  2
## 6:       penman  water  5
## 7:          all    et0  2
## 8:          all    etc  3
## 9:          all  water  6
```

```r
water.cv <- A.water[, .(cv = sd(V1) / mean(V1)), .(et0.formulae, output)]

# KT and CV plot
# KT and CV plot
a <- merge(water.kt, water.cv, by = c("et0.formulae", "output")) %>%
  .[, et0.formulae:= factor(et0.formulae, levels = c("penman", "priestley", "all"))] %>%
  ggplot(., aes(output, cv, color = et0.formulae, group = et0.formulae)) +
  geom_line() +
  geom_point(aes(size = kt)) +
  scale_color_manual(name = "$ET_0$ formula",
                     values = wes_palette("Zissou1"),
                     labels = c("Penman-Monteith",
                                "Priestley-Taylor",
                                "Uncertain")) +
  scale_size_continuous(name = "$k_t$",
                        breaks = c(1, 3, 6)) +
  scale_x_discrete(labels = c("Reference \n evapotranspiration",
                              "Crop \n evapotranspiration",
                              "Irrigation \n water withdrawal"),
                   guide = guide_axis(n.dodge = 2)) +
  labs(x = "", y = "CV") +
  theme_AP()

# ks plots

facet_labels <- c(
  "penman" = "Penman-Monteith",
  "priestley" = "Priestley-Taylor",
  "all" = "Uncertain"
)

b <- all.ind[!sensitivity == "Ti", sum(original), .(et0.formulae, output, sensitivity)] %>%
  .[, et0.formulae:= factor(et0.formulae, levels = c("penman", "priestley", "all"))] %>%
  ggplot(., aes(output, V1, fill = sensitivity)) +
  geom_col(position = position_stack(reverse = TRUE), color = "black") +
  scale_fill_manual(name = "",
                    labels = c("$S_i$", "$S_{i,j}$", "$S_{i,j,l}$"),
                    values = wes_palette("Cavalcanti1")) +
  scale_x_discrete(labels = c("Reference \n evapotranspiration",
                              "Crop \n evapotranspiration",
                              "Irrigation \n water withdrawal"),
                   guide = guide_axis(n.dodge = 2)) +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  geom_hline(yintercept = 0.99, lty = 2) +
  labs(x = "", y = "$\\sum S_i + \\sum S_{i,j} + \\sum S_{i,j,l}$") +
  annotation_custom(textGrob("p", gp = gpar(col = "red")),
                    xmin = 0.3, xmax = 1, ymin = 0.98, ymax = 0.98) +
```

```
  facet_wrap(~et0.formulae, labeller = as_labeller(facet_labels), ncol = 3) +
  theme_AP() +
  theme(legend.position = "top",
        panel.spacing = unit(2, "lines"),
        axis.text.x = element_text(size = 7))

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




```
all.ind[!sensitivity == "Ti", round(sum(original), 3), .(et0.formulae, output, sensitivity)]
```

```
##     et0.formulae output sensitivity    V1
##  1:    priestley    et0          Si 0.997
##  2:    priestley    et0         Sij 0.002
##  3:    priestley    et0        Sijl 0.000
##  4:    priestley    etc          Si 0.990
##  5:    priestley    etc         Sij 0.010
##  6:    priestley    etc        Sijl 0.000
```

```
##  7:    priestley  water          Si 0.908
##  8:    priestley  water         Sij 0.086
##  9:    priestley  water        Sijl 0.007
## 10:       penman    et0          Si 1.002
## 11:       penman    et0         Sij 0.000
## 12:       penman    et0        Sijl 0.000
## 13:       penman    etc          Si 0.989
## 14:       penman    etc         Sij 0.007
## 15:       penman    etc        Sijl 0.000
## 16:       penman  water          Si 0.928
## 17:       penman  water         Sij 0.076
## 18:       penman  water        Sijl 0.005
## 19:          all    et0          Si 0.976
## 20:          all    et0         Sij 0.023
## 21:          all    et0        Sijl 0.000
## 22:          all    etc          Si 0.953
## 23:          all    etc         Sij 0.045
## 24:          all    etc        Sijl 0.001
## 25:          all  water          Si 0.885
## 26:          all  water         Sij 0.104
## 27:          all  water        Sijl 0.021
##     et0.formulae output sensitivity    V1
```

```
# EXPORT WATER SENSITIVITY INDICES ---------------------------------------------

fwrite(all.ind, "ind.water.csv")
```

5 The Sobol' G function

```
# DEFINE MODELS ----------------------------------------------------------------

# Analytical indices of G function -----------
g_analytical <- function(a) {
  # Si
  Vi <- (1 / 3) / (1 + a)^2
  V <- prod((1 + Vi)) - 1
  Si.analytical <- Vi / V

  # Ti
  Vt <- vector()
  for(i in 1:length(Vi)) {
    Vt[i] <- Vi[i] * (prod(1 + Vi[-i]))
  }
  Ti.analytical <- Vt / V
  out <- c(Si.analytical, Ti.analytical)
  return(out)
}
```

```r
# G Function --------------------------------
g_fun <- function (X, a, epsilon) {
  set.seed(epsilon)
  a <- sample(a, size = ncol(X), prob = prob, replace = TRUE)
  y <- 1
  for (j in 1:ncol(X)) {
    y <- y * (abs(4 * X[, j] - 2) + a[j])/(1 + a[j])
  }
  analytical <- g_analytical(a)
  output <- list(y, analytical)
  names(output) <- c("output", "analytical")
  return(output)
}

# Model with G function
model_g <- function(N, k, epsilon, order) {
  params <- paste("X", 1:k, sep = "")
  mat <- sobol_matrices(N = N, params = params, order = order, scrambling = 1)
  y <- g_fun(X = mat, a = a, epsilon = epsilon)
  ind <- sobol_indices(Y = y$output, N = N, params = params, order = order)
  mae <- ind$results[sensitivity %in% c("Si", "Ti")][
    , abs(mean(original - y$analytical)), sensitivity][, V1]
  cv <- sd(y$output) / mean(y$output)
  sum.si <- ind$si.sum
  k.t <- ind$results[sensitivity == "Ti" & original > 0.05, length(unique(parameters))]
  Sij <- ind$results[sensitivity == "Sij", sum(original)]
  Sijl <- ind$results[sensitivity == "Sijl", sum(original)]
  return(c(cv, sum.si, k.t, Sij, Sijl, mae))
}
```

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

# Settings
params <- c("k", "epsilon")
N <- 2^10
N.internal <- 2^12
order <- "third"
matrices <- "A"
a <- c(0, 1, 4.5, 9, 99)
prob <- c(0.4, 0.3, 0.2, 0.05, 0.05)

# Creation of sample matrix
mat <- sobol_matrices(N = N, params = params, matrices = matrices, scrambling = 1)

# maximum number of explored inputs
max.k <- 20
```

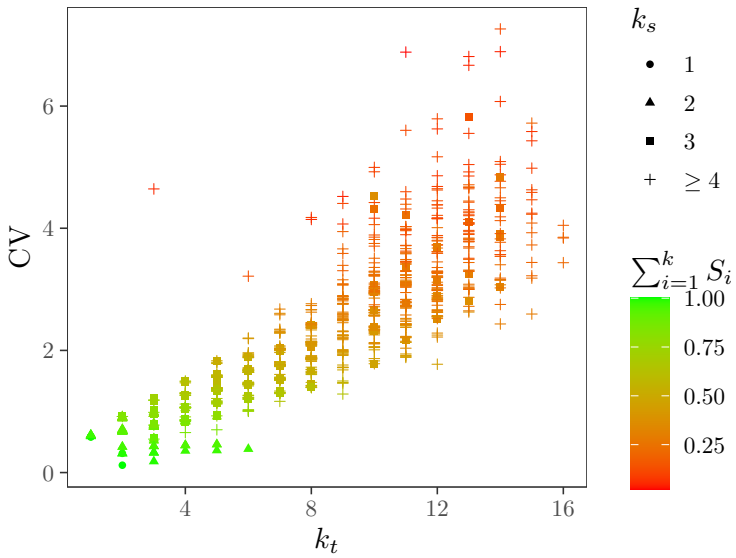
```r
# Transformation to appropriate distributions
mat[, "k"] <- floor(mat[, "k"] * (max.k - 3 + 1) + 3)
mat[, "epsilon"] <- floor(mat[, "epsilon"] * (N - 1 + 1) + 1)
```

```r
# RUN SOBOL' G MODEL -------------------------------------------------------

# Define parallel computing
n_cores <- detectCores() * 0.75
cl <- makeCluster(n_cores)
registerDoParallel(cl)

# Run model
Y <- foreach(i=1:nrow(mat), .packages = "sensobol",
             .combine = "rbind") %dopar%
  {
   model_g(N = N.internal,
           k = mat[[i, "k"]],
           epsilon = mat[[i, "epsilon"]],
           order = order)
  }

# Stop parallel cluster
stopCluster(cl)
```

```r
# ARRANGE SOBOL' G DATA ----------------------------------------------------

dt <- data.table(Y) %>%
  setnames(., paste("V", 1:7, sep = ""),
           c("CV", "sum.si", "k.t", "sij", "sijl", "mae.si", "mae.ti")) %>%
  cbind(mat, .) %>%
  .[, k.s:= ifelse(sum.si >= 0.99, 1,
                   ifelse(sum.si + sij >= 0.99, 2,
                          ifelse((sum.si + sij + sijl) >= 0.99, 3, 4)))] %>%
  .[, k.s:= factor(k.s)] %>%
  .[, up.second:= sum.si + sij] %>%
  .[, up.third:= sum.si + sij + sijl]

fwrite(dt, "dt.sobol.csv")
```

```r
# PLOT ---------------------------------------------------------------------

sobolg.plot <- ggplot(dt, aes(k.t, CV, color = sum.si, shape = k.s)) +
  geom_point(size = 1) +
  labs(x = "$k_t$", y = "CV") +
  scale_color_gradient(low = "red", high = "green",
                       name = "$\\sum_{i=1}^{k} S_i$") +
  scale_shape_discrete(name = "$k_s$",
                       labels = c("1", "2", "3", "$\\geq 4$")) +
```

```
  theme_AP()

sobolg.plot
```

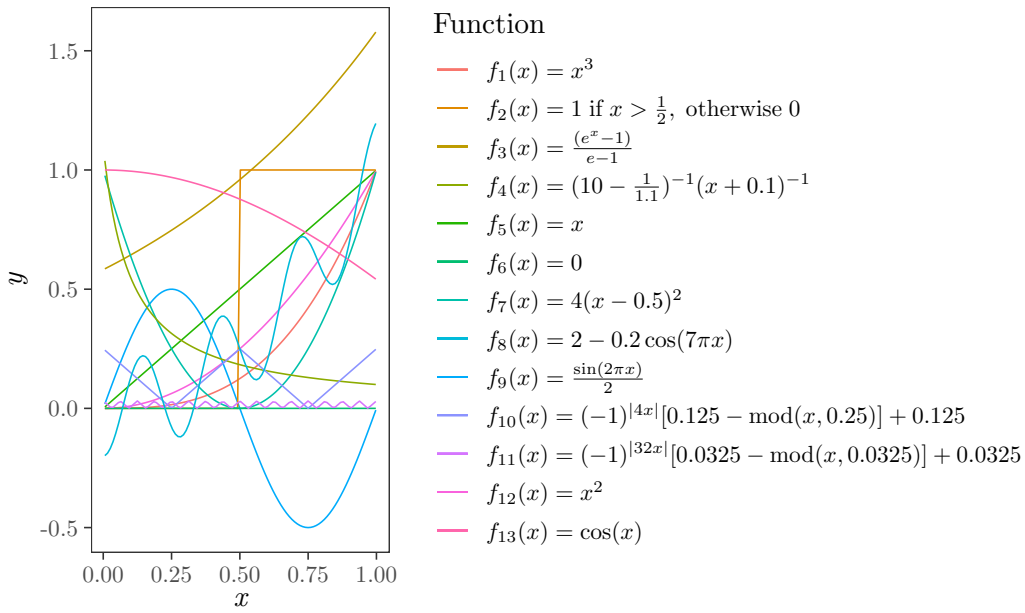


6 The metafunction

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

# Settings
params <- c("k", "epsilon", "n")
N <- 2^11
matrices <- "A"

# Creation of sample matrix
mat <- sobol_matrices(N = N, params = params, matrices = matrices, scrambling = 1)

# maximum number of explored inputs
max.k <- 15

# Transformation to appropriate distributions
mat[, "k"] <- floor(mat[, "k"] * (max.k - 2 + 1) + 2)
mat[, "epsilon"] <- floor(mat[, "epsilon"] * (N - 1 + 1) + 1)
set.seed(666)
mat[, "n"] <- sapply(mat[, "k"], function(x) sample(2:x, 1))
mat[, "n"] <- ifelse(mat[, "n"] == 1, 2, mat[, "n"]) # Correct and force 1 to be 2
```

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

function_list <- list(
    Linear = function(x) x,
    Quadratic = function(x) x ^ 2,
```

```
    Cubic = function(x) x ^ 3,
    Exponential = function(x) exp(1) ^ x / (exp(1) - 1),
    Periodic = function(x) sin(2 * pi * x) / 2,
    Discontinuous = function(x) ifelse(x > 0.5, 1, 0),
    Non.monotonic = function(x) 4 * (x - 0.5) ^ 2,
    Inverse = function(x) (10 - 1 / 1.1) ^ -1 * (x + 0.1) ^ - 1,
    No.effect = function(x) x * 0,
    Trigonometric = function(x) cos(x),
    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)
  )

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)

plot.metafunction
```




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

# Define metafunction
meta_fun <- function(data, epsilon, n) {

  # Define list of functions included in metafunction
  function_list <- list(
    Linear = function(x) x,
    Quadratic = function(x) x ^ 2,
    Cubic = function(x) x ^ 3,
    Exponential = function(x) exp(1) ^ x / (exp(1) - 1),
    Periodic = function(x) sin(2 * pi * x) / 2,
    Discontinuous = function(x) ifelse(x > 0.5, 1, 0),
    Non.monotonic = function(x) 4 * (x - 0.5) ^ 2,
    Inverse = function(x) (10 - 1 / 1.1) ^ -1 * (x + 0.1) ^ - 1,
    No.effect = function(x) x * 0,
    Trigonometric = function(x) cos(x),
    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)) +
    Oscillation = function(x) x ^ 2 - 0.2 * cos(7 * pi * x)
  )

  # Sample list of functions
  set.seed(epsilon)
  all_functions <- sample(names(function_list), ncol(data), replace = TRUE)

  # Compute model output first order effects
  mat.y <- sapply(seq_along(all_functions), function(x)
    function_list[[all_functions[x]]](data[, x]))
```

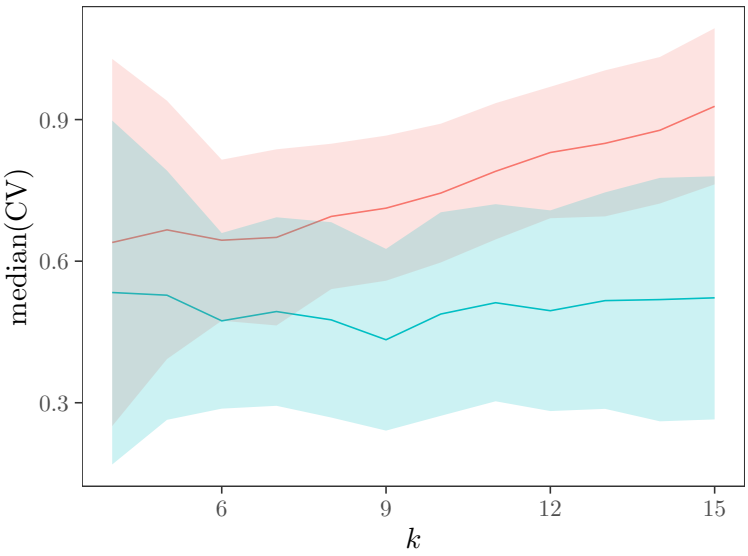
```
  # Compute first-order effects
  y1 <- Rfast::rowsums(mat.y)

  # Define matrix with all possible interactions up to the n-th order
  interactions <- lapply(2:n, function(x) RcppAlgos::comboGeneral(1:n, x, nThreads = 4))

  out <- lapply(1:length(interactions), function(x) {
    lapply(1:nrow(interactions[[x]]), function(y) {
      Rfast::rowprods(mat.y[, interactions[[x]][y, ]])
    })
  })

  y2 <- lapply(out, function(x) do.call(cbind, x)) %>%
    do.call(cbind, .) %>%
    Rfast::rowsums(.)

  y <- y1 + y2

  return(y)
}

# Merge metafunction with model
model <- function(data, epsilon, n) {

  k <- ncol(data)

  if (n > k) {
    stop("level_interactions should be smaller or equal than \n
         the number of parameters")
  }
  y <- meta_fun(data = data, epsilon = epsilon, n = n)

  return(y)

}

# Define final model
model_fun <- function(k, epsilon, N, n) {
  params <- paste("X", 1:k, sep = "")
  mat <- sobol_matrices(N = N, params = params,
                        matrices = "A", scrambling = 1)
  y <- model(data = mat, epsilon = epsilon, n = n)
  cv <- sd(y) / mean(y)
  return(cv)
}
```

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

# Define parallel computing
n_cores <- detectCores() * 0.75
cl <- makeCluster(n_cores)
registerDoParallel(cl)

N.internal <- 2^9
high.order <- c("n", "k")

# Compute
Y <- foreach(j = high.order) %:%
  foreach(i=1:nrow(mat), .packages = c("Rfast", "sensobol", "dplyr", "RcppAlgos"),
          .combine = "rbind") %dopar%
  {
    model_fun(k = mat[[i, "k"]],
              epsilon = mat[[i, "epsilon"]],
              n = mat[[i, j]],
              N = N.internal)
  }


# Run by fixing the model dimensionality at 15
Y.k <- foreach(i=1:nrow(mat), .packages = c("Rfast", "sensobol", "dplyr", "RcppAlgos"),
               .combine = "rbind") %dopar%
  {
    model_fun(k = max.k,
              epsilon = mat[[i, "epsilon"]],
              n = mat[[i, "n"]],
              N = N.internal)
  }

# Stop parallel cluster
stopCluster(cl)
```

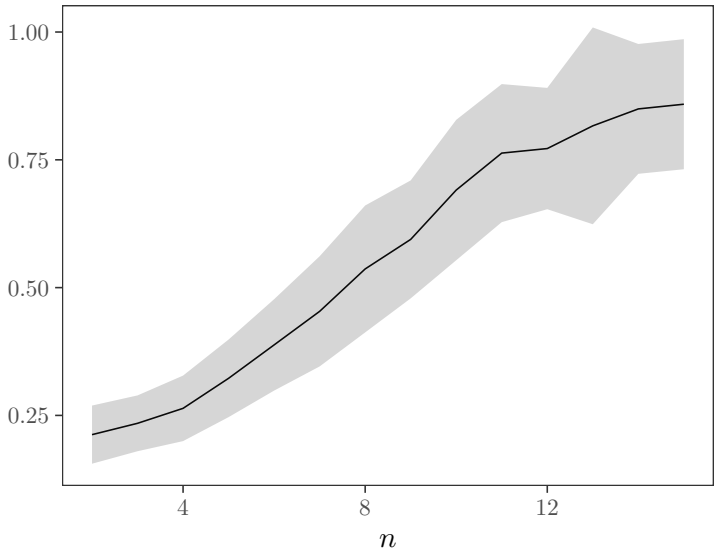
```r
# ARRANGE MODEL OUTPUT -------------------------------------------------------

names(Y) <- high.order

dt <- lapply(Y, data.table) %>%
  lapply(., function(x) cbind(mat, x)) %>%
  rbindlist(., idcol = "high.order") %>%
  .[, n:= ifelse(high.order == "n", n, k)] %>%
  setnames(., "V1", "CV") %>%
  setnames(., "n", "order.interactions") %>%
  .[, ID:= 1:.N]
```

```r
dt.k <- cbind(mat, data.table(Y.k)) %>%
  setnames(., "V1", "CV")

fwrite(dt, "dt.csv")
fwrite(dt.k, "dt.k.csv")
```

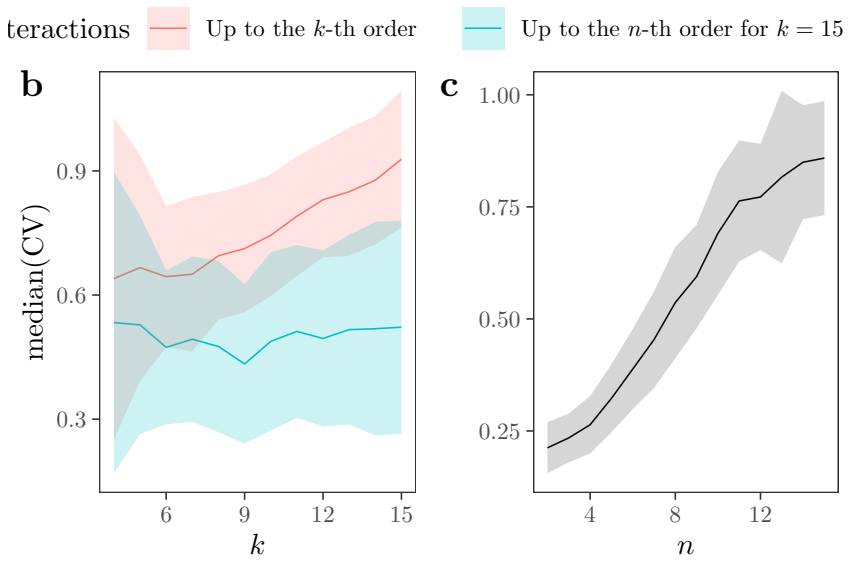
```r
# PLOTS ------------------------------------------------------------------------

a <- dt[k > 3, .(median = median(CV, na.rm = TRUE), sd = sd(CV, na.rm = TRUE)), .(k, high.order
  ggplot(., aes(k, median, fill = high.order)) +
  geom_line(aes(color = high.order)) +
  geom_ribbon(aes(y = median, ymin = median - sd, ymax = median + sd), alpha = 0.2) +
  labs(x = "$k$", y = "median(CV)") +
  scale_color_discrete(name = "Interactions",
                       labels = c("Up to the $k$-th order",
                                  "Up to the $n$-th order for $k=15$")) +
  scale_fill_discrete(name = "Interactions",
                      labels = c("Up to the $k$-th order",
                                 "Up to the $n$-th order for $k=15$")) +
  theme_AP() +
  theme(legend.position = "none")

b <- dt.k[, .(median = median(CV, na.rm = TRUE), sd = sd(CV, na.rm = TRUE)), n] %>%
  ggplot(., aes(n, median)) +
  geom_line() +
  geom_ribbon(aes(y = median, ymin = median - sd, ymax = median + sd), alpha = 0.2) +
  labs(x = "$n$", y = "") +
  theme_AP()
a
```



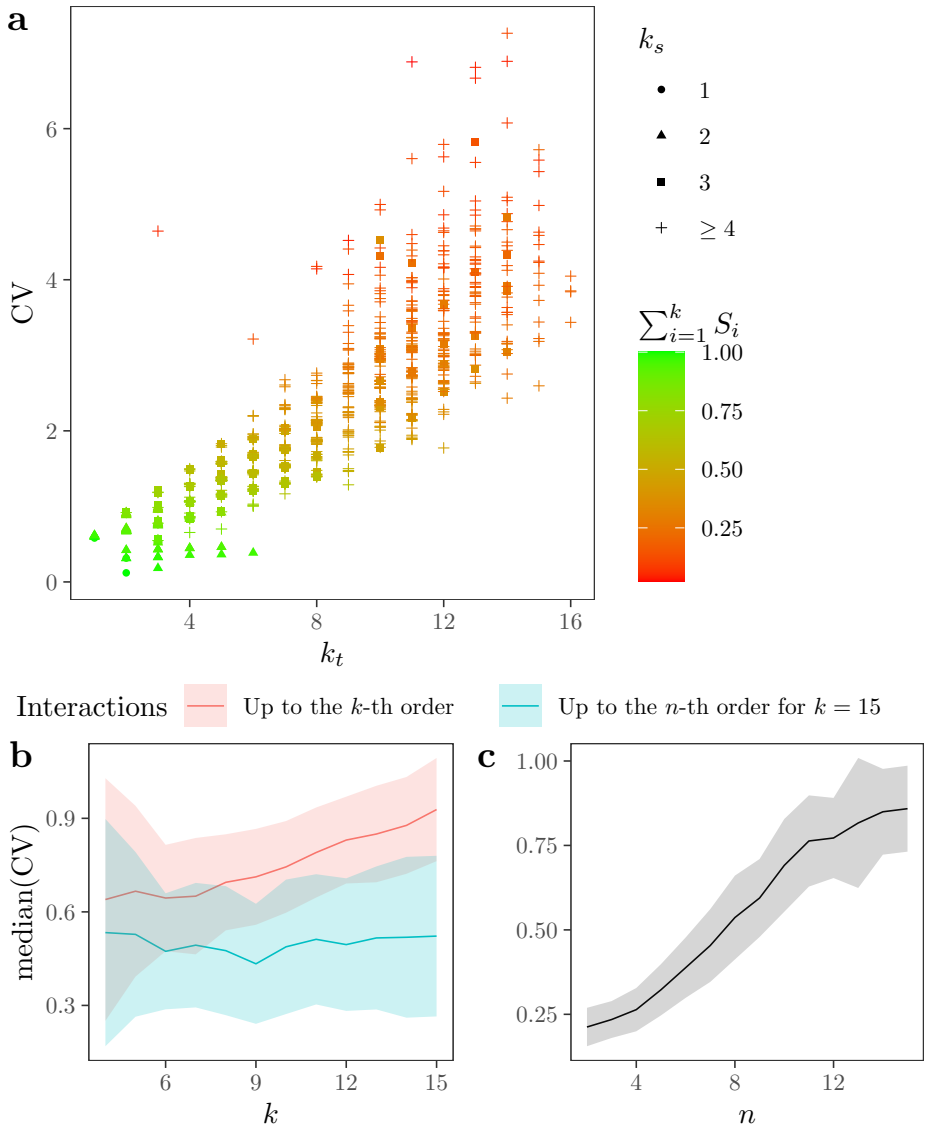
```
b
```




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

legend <- get_legend(a + theme(legend.position = "top"))
bottom <- plot_grid(a, b, ncol = 2, labels = c("b", "c"))
meta.plot <- plot_grid(legend, bottom, ncol = 1, rel_heights = c(0.1, 0.9))
meta.plot
```




```
# MERGE WITH SOBOL' G PLOT -----------------------------------------------------

plot_grid(sobolg.plot, meta.plot, ncol = 1, labels = c("a", ""),
          rel_heights = c(0.6, 0.4))
```




```
plot_grid(plot.metafunction, meta.plot, rel_heights = c(0.65, 0.35),
          rel_widths = c(0.7, 0.3), ncol = 1)
```

tion

$f_1(x) = x^3$

$f_2(x) = 1$ if $x > \frac{1}{2}$, otherwise 0

$f_3(x) = \frac{(e^x - 1)}{e - 1}$

$f_4(x) = (10 - \frac{1}{1.1})^{-1}(x + 0.1)^{-1}$

$f_5(x) = x$

$f_6(x) = 0$

$f_7(x) = 4(x - 0.5)^2$

$f_8(x) = 2 - 0.2\cos(7\pi x)$

$f_9(x) = \frac{\sin(2\pi x)}{2}$

$f_{10}(x) = (-1)^{\lfloor 4x \rfloor}[0.125 - \text{mod}(x, 0.25)] + 0.125$

$f_{11}(x) = (-1)^{\lfloor 32x \rfloor}[0.0325 - \text{mod}(x, 0.0325)] + 0.0325$

$f_{12}(x) = x^2$

$f_{13}(x) = \cos(x)$