flood_notebook_M2RI

February 2, 2023

1 Global sensitivity analysis with derivatives. A flood case study.

[ ]:
```
## O. Roustant (INSA Toulouse) and O. Zahm (INRIA Grenoble), with contributions␣
↪from B. Iooss (EDF & IMT)
```

We consider a simplified first model simulating flooding events. The model has 8 input random variables, viewed as random variables:

- $X_1 = Q$, Maximal annual flowrate (m^3/s), Gumbel $\mathcal{G}(1013, 558)$ truncated on $[500, 3000]$
- $X_2 = K_s$, Strickler coefficient, Normal $\mathcal{N}(30, 8^2)$ truncated on $[15, +\infty[$
- $X_3 = Z_v$, River downstream level (m), Triangular $\mathcal{T}(49, 51)$
- $X_4 = Z_m$, River upstream level (m), Triangular $\mathcal{T}(54, 56)$
- $X_5 = H_d$, Dyke height (m), Uniform $\mathcal{U}[7, 9]$
- $X_6 = C_b$, Bank level (m), Triangular $\mathcal{T}(55, 56)$
- $X_7 = L$, River stretch (m), Triangular $\mathcal{T}(4990, 5010)$
- $X_8 = B$, River width (m), Triangular $\mathcal{T}(295, 305)$

We consider two variables of interest. First, the maximal annual overflow S (in meters), obtained from simplified hydro-dynamical equations of Saint-Venant:

$$S = \left(\frac{Q}{BK_s \sqrt{\frac{Z_m - Z_v}{L}}} \right)^{0.6} + Z_v - H_d - C_b \,. \tag{1}$$

Secondly, the cost (in million euros) of the damage on the dyke Y, depending on S, written as:

$$Y = 1_{S>0} + \left[0.2 + 0.8 \left(1 - \exp^{-\frac{1000}{S^4}} \right) \right] 1_{S \leq 0} + \frac{1}{20} \left(H_d 1_{H_d > 8} + 81_{H_d \leq 8} \right) , \tag{2}$$

where $1_A(x)$ is the indicator function which is equal to 1 for $x \in A$ and 0 otherwise. The aim is to perform a global sensitivity analysis for S and Y. In particular, we would like to detect the non-essential input variables and the active subspaces.

[ ]:
```
# Input names
floodInputNames <- c("Q", "Ks", "Zv", "Zm", "Hd", "Cb", "L", "B")
floodOutputNames <- c("Subverse", "Cost")

# Flood model
flood <- function(X, ans = 0){
    # ans = 1 gives Overflow output; ans = 2 gives Cost output; ans=0 gives both
```

```
    mat <- as.matrix(X, ncol = 8)
    if (ans == 0) {
        output <- matrix(NA, nrow(mat), 2)
    } else {
        output <- rep(NA, nrow(mat))
    }
    for (i in 1:nrow(mat)){
        H <- (mat[i, 1] / (mat[i, 2] * mat[i, 8] * sqrt((mat[i, 4] - mat[i, 3])
 ↪/ mat[i, 7])))^0.6
        S <- mat[i, 3] + H - mat[i, 5] - mat[i, 6]
    if (S > 0){
        Cp <- 1
    } else {
        Cp <- 0.2 + 0.8 * (1 - exp(- 1000 / S^4))
    }
    if (mat[i, 5] > 8){
        Cp <- Cp + mat[i, 5] / 20
    } else {
        Cp = Cp + 8 / 20
    }
    if (ans == 0){
        output[i, 1] <- S ;
        output[i, 2] <- Cp ;
    }
    if (ans == 1) output[i] <- S
    if (ans == 2) output[i] <- Cp
    }
    return(output)
}
```

[ ]:
```
# Flood model derivatives (here by finite-differences)
floodDer <- function(X, i, ans, eps){
    Xder <- X
    X1 <- X
    X1[, i] <- X[, i] + eps
    Xder <- (flood(X1, ans) - flood(X, ans)) / eps
    return(Xder)
}
```

[ ]:
```
# Function for flood model inputs sampling
library(sensitivity) # (truncated) Gumbel law and truncated Normal law
library(triangle) # Triangular law

floodSample <- function(size){
    X <- matrix(NA, size, 8)
    X[, 1] <- rgumbel.trunc(size, loc = 1013, scale = 558, min = 500, max =
 ↪3000)
```

```
    X[, 2] <- rnorm.trunc(size, mean = 30, sd = 8, min = 15)
    X[, 3] <- rtriangle(size, a = 49, b = 51)
    X[, 4] <- rtriangle(size, a = 54, b = 56)
    X[, 5] <- runif(size, min = 7, max = 9)
    X[, 6] <- rtriangle(size, a = 55, b = 56)
    X[, 7] <- rtriangle(size, a = 4990, b = 5010)
    X[, 8] <- rtriangle(size, a = 295, b = 305)
    return(X)
}
```

1.1 Preliminary visualization

From now on, we select one output. As a first sensitivity diagnostic, we plot that output versus the input variables.

[ ]:
```
myOutput <- 1  # 1 for subverse, 2 for cost
```

[ ]:
```
N <- 100
X <- ?  # create a sample of size N of the input variables
Y <- ?  # evaluation the output on X
par(mfrow = c(2, 4))
options(repr.plot.width = 15, repr.plot.height = 6)
for (i in 1:8){
    plot(Y ~ X[, i], xlab = floodInputNames[i], ylab =
 ↪floodOutputNames[myOutput],
         pch = 19, col = "grey", cex = 0.5, cex.lab = 1.5)
    ss <- ?  # add an estimation of the conditional expectation with function
 ↪smooth.spline
    lines(ss, col = "blue", lwd = 3)
}
```

Q Which input variables seem influential/inactive? What represents the blue line? Which quantities canNOT be visualized with these plots?

Q Adapt the code to plot the main effects.

1.2 Computation of Sobol indices

[ ]:
```
library(boot)
# we use a pick-freeze formula (Sobol Jansen), which requires two samples
N <- 100 # a large sample size is preferable
X1 <- ?   # create a sample of size N for input variables
X2 <- ?   # create a second sample of size N, independent of the first one
X1 <- data.frame(X1); names(X1) <- floodInputNames
X2 <- data.frame(X2); names(X2) <- floodInputNames
```

```r
# sensitivity analysis
x <- ?   # use function soboljansen to estimate the total effects
# use nboot = 100, in order to quantify the estimation error
totalIndex <- x$T$original; names(totalIndex) <- floodInputNames
sobolIndex <- x$S$original; names(sobolIndex) <- floodInputNames

library(ggplot2)
ggplot(x)
```

Q Interpret the result. Is it consistent with the output~input plots? What other information can you get now?

1.3 Computation of DGSM

[ ]:
```r
d <- 8
n <- 100  # we choose here a small sample size
eps <- 1e-7 # finite-differences for derivatives
X <- ?  # create a sample of size n for the input variables
yy <- ? # evaluate the output at X
y <- scale(yy, center = TRUE, scale = FALSE)[, 1]    # we choose to remove the␣
 ↪global mean
dy <- NULL
for (i in 1:d){
    dy <- cbind(dy, ?) # here add a column vector containing the partial␣
 ↪derivative of the output / X_i
}
DGSM <- ?  # Monte Carlo estimate of the DGSM (use function colMeans)
names(DGSM) <- floodInputNames
DGSMw <- DGSM  # for the second question: multiply each term by the input␣
 ↪variance
par(mfrow = c(1, 2))
barplot(DGSMw / sum(DGSMw), ylim = c(0, 1), ylab = "DGSM", xlab =␣
 ↪paste("Estimation with n = ", n))
barplot(totalIndex, ylim = c(0, 1), ylab = "Total Sobol index", xlab =␣
 ↪paste("Estimation with n = ", N))
```

Q What can you say about the ranking of influential variables obtained by (normalized) DGSM and total Sobol index?

Q Redo the comparison by using a weighted DGSM obtained by multiplying the DGSM by the variance of the corresponding input.

1.4 Computation of upper bounds of Sobol indices, based on DGSM

[ ]:
```
method <- "quad"
out_1 <- PoincareOptimal(distr = list("gumbel", 1013, 558), min = 500, max =␣
↪3000,
                         only.values = FALSE, der = TRUE, method = method)
out_2 <- PoincareOptimal(distr = list("norm", 30, 8), min = 15, max = 200,
                         only.values = FALSE, der = TRUE, method = method)
out_3 <- PoincareOptimal(distr = list("triangle", 49, 51, 50),
                         only.values = FALSE, der = TRUE, method = method)
out_4 <- PoincareOptimal(distr = list("triangle", 54, 56, 55),
                         only.values = FALSE, der = TRUE, method = method)
out_5 <- PoincareOptimal(distr = list("unif", 7, 9),
                         only.values = FALSE, der = TRUE, method = method)
out_6 <- PoincareOptimal(distr = list("triangle", 55, 56, 55.5),
                         only.values = FALSE, der = TRUE, method = method)
out_7 <- PoincareOptimal(distr = list("triangle", 4990, 5010, 5000),
                         only.values = FALSE, der = TRUE, method = method)
out_8 <- PoincareOptimal(distr = list("triangle", 295, 305, 300),
                         only.values = FALSE, der = TRUE, method = method)
out_ <- list(out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8)
```

[ ]:
```
PoinConst <- sapply(out_, function(x) x$opt)
D <- ? # global variance
upperBound <- ?  # the upper bound of the total index computed with DGSM
par(mfrow = c(1, 2))
barplot(upperBound, ylim = c(0, 1), ylab = "upper bound based on DGSM", xlab =␣
↪paste("Estimation with n = ", n))
barplot(totalIndex, ylim = c(0, 1), ylab = "Total Sobol index", xlab =␣
↪paste("Estimation with n = ", N))
```

Q On the left, the barplot represents the upper bounds of the total Sobol index obtained from DGSM by using Poincaré inequalities. Comment the results.

Q Compare the Poincaré constant with the input variance. Explain.

To assess the uncertainty on the estimates, a bootstrap procedure is applied.

[ ]:
```
# Bootstrap to assess the uncertainty on the upper bound
nBoot <- 100
UB_boot <- matrix(NA, nrow = nBoot, ncol = d)
for (b in 1:nBoot){
    bootSample <- sample(1:n, replace = TRUE)
    y_boot <- y[bootSample]   # we resample the output
    dy_boot <- dy[bootSample, ]
    DGSM_boot <- ?   # Monte Carlo estimate of the DGSM for this new sample
    D_boot <- ?  # global variance for the new sample
    UB_boot[b, ] <- ? # upper bound for the new sample
```

```
}
# Plot the result
par(mfrow = c(1, 2))
plot(upperBound, type = "p", pch = 19, ylim = c(0, 1), ylab = "upper bound␣
 ↪based on DGSM",
     xlab = paste("Estimation with n = ", n), xaxt = "n")
axis(1, at = 1:d, labels = floodInputNames)
for (i in 1:d){
    arrows(x0 = i, y0 = quantile(UB_boot[, i], 0.025),
           x1 = i, y1 = quantile(UB_boot[, i], 0.975), code = 0)
}
plot(totalIndex, type = "p", pch = 19, ylim = c(0, 1), ylab = "Total Sobol␣
 ↪index",
     xlab = paste("Estimation with n = ", N), xaxt = "n")
axis(1, at = 1:d, labels = floodInputNames)
for (i in 1:d){
    arrows(x0 = i, y0 = x$T[i, 4],
           x1 = i, y1 = x$T[i, 5], code = 0)
}
```

Q The two plots represent 95% confidence intervals computed by boostrap (with 100 replicates) for the total Sobol index (right) and a upper bound based on DGSM (left). When a single point is visible, it means that the width of the confidence interval is too small to be visible. Are the conclusion modified?

Q Replace D_boot by D (estimated on the larger sample used to compute the total Sobol index) in the code above, in order to visualize the part of variability of the confidence intervals due to the global variance. Conclusion ?

Q. Actually the comparison is unfair, because the pick-freeze estimator uses (d+2)N=10N function evaluations, versus N for the DGSM upper bound (if the gradient is provided). Check that the size of the confidence intervals for the total Sobol indices, estimated by Sobol Jansen, is approximately 3 times larger.

1.5 Estimation of Sobol indices with Poincaré chaos

[ ]:
```
# In this basic example, we just use the first-order terms in Poincaré chaos␣
 ↪with interactions.
# Better results can be obtained with a larger degree.
c2 <- c2der <- c2tot <- c2totder <- rep(0, d)

for (i in 1:d){
  m <- diag(1, d, d) ; m[, i] <- 1

  for (j in 1:d){
    cc <- PoincareChaosSqCoef(PoincareEigen = out_, multiIndex = m[j,],
```

```
                                design = X, output = y, outputGrad = NULL,␣
 ↪inputIndex = i, der = FALSE)
    c2tot[i] <- c2tot[i] + cc
    if (j == i) c2[i] <- cc

    cc <- PoincareChaosSqCoef(PoincareEigen = out_, multiIndex = m[j,],
                                design = X, output = y, outputGrad = dy,␣
 ↪inputIndex = i, der = TRUE)
    c2totder[i] <- c2totder[i] + cc
    if (j == i) c2der[i] <- cc
  }
}
```

[ ]:
```
# Compute confidence intervals by bootstrap
c2_boot <- c2der_boot <- c2tot_boot <- c2totder_boot <- matrix(0, nBoot, d)
D_boot <- rep(0, nBoot)

for (b in 1:nBoot){
    bootSample <- sample(1:n, replace = TRUE)
    X_boot <- X[bootSample, ]
    y_boot <- y[bootSample]
    dy_boot <- dy[bootSample, ]
    D_boot[b] <- var(y_boot)

    for (i in 1:d){
        m <- diag(1, d, d) ; m[, i] <- 1

        for (j in 1:d){
            cc <- PoincareChaosSqCoef(PoincareEigen = out_, multiIndex = m[j, ],
                                design = X_boot, output = y_boot, outputGrad =␣
 ↪NULL, inputIndex = i, der = FALSE)
            c2tot_boot[b, i] <- c2tot_boot[b, i] + cc
            if (j == i) c2_boot[b, i] <- cc

            cc <- PoincareChaosSqCoef(PoincareEigen = out_, multiIndex = m[j, ],
                                design = X_boot, output = y_boot, outputGrad =␣
 ↪dy_boot, inputIndex = i, der = TRUE)
            c2totder_boot[b, i] <- c2totder_boot[b, i] + cc
            if (j == i) c2der_boot[b, i] <- cc
        }
    }
}
```

[ ]:
```
# Representation of the Sobol indices and total Sobol indices
par(mfrow = c(2, 2))
plot(c2der / D, type = "p", pch = 19, ylim = c(0, 1), ylab = "Sobol index",␣
 ↪main = "Poincaré chaos approach",
```

```
     xlab = paste("Estimation with n = ", n), xaxt = "n")
axis(1, at = 1:d, labels = floodInputNames)
for (i in 1:d){
    arrows(x0 = i, y0 = quantile(c2der_boot[, i]/D_boot, 0.025),
           x1 = i, y1 = quantile(c2der_boot[, i]/D_boot, 0.975), code = 0)
}
plot(sobolIndex, type = "p", pch = 19, ylim = c(0, 1), ylab = "Sobol index",␣
 ↪main = "Standard way",
     xlab = paste("Estimation with n = ", N), xaxt = "n")
axis(1, at = 1:d, labels = floodInputNames)
for (i in 1:d){
    arrows(x0 = i, y0 = x$S[i, 4],
           x1 = i, y1 = x$S[i, 5], code = 0)
}
plot(c2totder / D, type = "p", pch = 19, ylim = c(0, 1), ylab = "total Sobol␣
 ↪index", main = "Poincaré chaos approach",
     xlab = paste("Estimation with n = ", n), xaxt = "n")
axis(1, at = 1:d, labels = floodInputNames)
for (i in 1:d){
    arrows(x0 = i, y0 = quantile(c2totder_boot[, i]/D_boot, 0.025),
           x1 = i, y1 = quantile(c2totder_boot[, i]/D_boot, 0.975), code = 0)
}
plot(totalIndex, type = "p", pch = 19, ylim = c(0, 1), ylab = "total Sobol␣
 ↪index", main = "Standard way",
     xlab = paste("Estimation with n = ", N), xaxt = "n")
axis(1, at = 1:d, labels = floodInputNames)
for (i in 1:d){
    arrows(x0 = i, y0 = x$T[i, 4],
           x1 = i, y1 = x$T[i, 5], code = 0)
}
```

Here above, we compare the estimation of Sobol and total Sobol index, either by Poincaré chaos (using derivatives) or by classical estimators (here Sobol Jansen). Notice that the truncation in the Poincaré chaos corresponds to keeping only the 1st order terms with interactions. This is a strong limitation, and may underestimate the Sobol indices too much. A more advanced implementation, based on sparse chaos, has been done in UQLab (Matlab code). Notice that the comparison is unfair here, because the Sobol indices computed with Sobol Jansen estimator uses more function evaluations (assuming that the gradient is available).

Q Try other estimation methods for Sobol indices (sobol2002, sobol2007, etc.), implemented in package sensitivity. Adapt the sample sizes for a fair comparison in terms of function evaluations.

1.6 Active subspaces

[ ]:
```
N <- 100
X <- floodSample(N)
Y <- flood(X, ans = myOutput)
dy <- NULL
for (i in 1:d) dy <- cbind(dy, floodDer(X, i, ans = myOutput, eps))

sdX <- ?  # standard deviation of each input
diagSdX <- ? # diagonal matrix containing the standard deviations of the inputs
H <- ?  # the active subspace matrix for the normalized input variables
E <- eigen(H)  # diagonalization of H
approxBound <- rev(cumsum(rev(E$values)))   # eigenvalues
barplot(approxBound)
E
R <- E$vectors  # eigenvectors
```

[ ]:
```
RX <- ? # linear combinations associated to the eigen vectors for the scaled␣
 ↪inputs
par(mfrow = c(2, 4))
options(repr.plot.width = 15, repr.plot.height = 6)
for (i in 1:d){
    plot(Y ~ RX[, i], xlab = paste("Component", i), ylab =␣
 ↪floodOutputNames[myOutput],
         pch = 19, col = "grey", cex = 0.5, cex.lab = 1.5)
    ss <- ?  # estimation of the conditional expectation
    lines(ss, col = "blue", lwd = 3)
}
```

[ ]:
```

```