Hands-on Uncertainty Analysis

Arnald Puy

26th October 2022

Install and load required packages

You should have the packages `sensobol` (Puy, Lo Piano, et al. 2022), `data.table` (Dowle and Srinivasan 2021) and `tidyverse` (Wickham et al. 2019) installed in your R environment to reproduce the results of this session. To install them, remove the hash character from the first line of the code snippet below and run the code.

```r
# install.packages(c("sensobol", "data.table", "tidyverse"))

library("sensobol")
library("data.table")
library("tidyverse")
```

Uncertainty analysis (UA)

An uncertainty analysis (UA) aims at quantifying how the uncertainty in the model inputs (i.e., in parameters, boundary conditions, mathematical formulations) affects the model output. Uncertainty can derive from measurement error, natural variation, inherent randomness or lack of knowledge (epistemic uncertainty). Often an uncertainty analysis goes hand in hand with a sensitivity analysis (SA), which aims at informing which inputs convey the most uncertainty to the output (Saltelli et al. 2008). Both UA/SA increase the transparency and quality of the modelling process and are especially recommended if the model aims at guiding decisions in the real world (Saltelli et al. 2020).

An UA requires the analyst to:

- Define which inputs of the model are treated as uncertain.
- Design a sample matrix. This sample matrix will be your model's uncertain space and will be formed by N rows and k columns. N is a number chosen by the analyst (often $N > 10,000$) and is the sample size, that is, the number of times that the model will run on different combination of values for the uncertain inputs. As for k, it reflects the number of uncertain inputs.
- Run the model in the sample matrix to get a numeric vector with the model output.
- Analyze the uncertainty in the model output with descriptive statistics (average value, median, quantiles, confidence intervals, etc) and appropriate plots.

Here we will show how to conduct a simple UA analysis in the R environment. First, we need to define the main settings of the UA, which will remain the same throughout the exercise unless stated otherwise. We define the sample size as $N = 2^7 = 128$, the required matrix as $\mathbf{A}$ (this is related to SA and hence beyond the scope of the tutorial, check Puy, Lo Piano, et al. (2022) for further information), and the type of sample matrix as `QRN`, meaning that we will use quasi-random numbers to build the sample matrix (we will see why later on).

```
# Sample size
N <- 2^7

# Required matrices
matrices <- "A"

# Type of sample matrix
type <- "QRN"
```

Now let us get into the meat of the matter.

Dummy example (1)

Let us start by assuming that our model has two uncertain inputs, which we name x_1 and x_2:

```
params <- c("x1", "x2")
```

We then create the sample matrix with the function `sobol_matrices`, from the `sensobol` package (Puy, Lo Piano, et al. 2022).

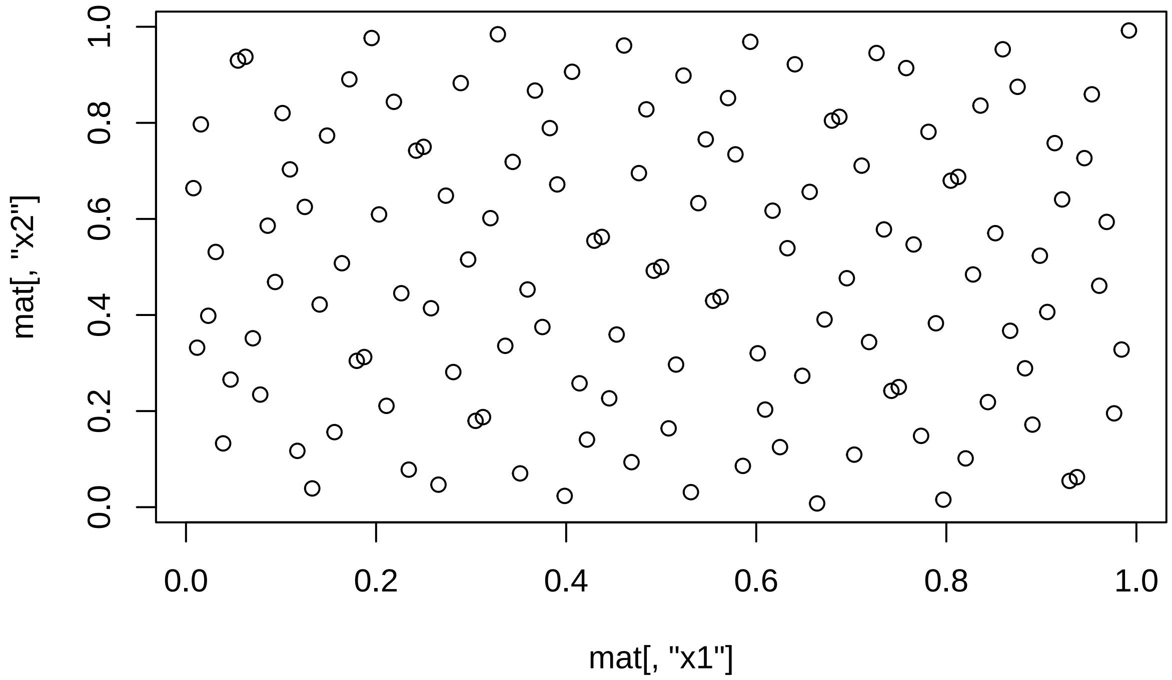
```
mat <- sobol_matrices(matrices = matrices, N = N, params = params, type = type)
```

Let us inspect the resulting matrix: first, we print the matrix, and then we plot it:

```
mat
```

```
##                 x1        x2
##   [1,] 0.50000000 0.5000000
##   [2,] 0.75000000 0.2500000
##   [3,] 0.25000000 0.7500000
##   [4,] 0.37500000 0.3750000
##   [5,] 0.87500000 0.8750000
##   [6,] 0.62500000 0.1250000
##   [7,] 0.12500000 0.6250000
##   [8,] 0.18750000 0.3125000
##   [9,] 0.68750000 0.8125000
##  [10,] 0.93750000 0.0625000
##  [11,] 0.43750000 0.5625000
##  [12,] 0.31250000 0.1875000
##  [13,] 0.81250000 0.6875000
##  [14,] 0.56250000 0.4375000
##  [15,] 0.06250000 0.9375000
##  [16,] 0.09375000 0.4687500
##  [17,] 0.59375000 0.9687500
##  [18,] 0.84375000 0.2187500
##  [19,] 0.34375000 0.7187500
##  [20,] 0.46875000 0.0937500
##  [21,] 0.96875000 0.5937500
##  [22,] 0.71875000 0.3437500
##  [23,] 0.21875000 0.8437500
##  [24,] 0.15625000 0.1562500
##  [25,] 0.65625000 0.6562500
##  [26,] 0.90625000 0.4062500
##  [27,] 0.40625000 0.9062500
##  [28,] 0.28125000 0.2812500
##  [29,] 0.78125000 0.7812500
## [30,] 0.53125000 0.0312500
## [31,] 0.03125000 0.5312500
## [32,] 0.04687500 0.2656250
## [33,] 0.54687500 0.7656250
## [34,] 0.79687500 0.0156250
## [35,] 0.29687500 0.5156250
## [36,] 0.42187500 0.1406250
## [37,] 0.92187500 0.6406250
## [38,] 0.67187500 0.3906250
## [39,] 0.17187500 0.8906250
## [40,] 0.23437500 0.0781250
## [41,] 0.73437500 0.5781250
## [42,] 0.98437500 0.3281250
## [43,] 0.48437500 0.8281250
## [44,] 0.35937500 0.4531250
## [45,] 0.85937500 0.9531250
## [46,] 0.60937500 0.2031250
## [47,] 0.10937500 0.7031250
## [48,] 0.07812500 0.2343750
## [49,] 0.57812500 0.7343750
## [50,] 0.82812500 0.4843750
## [51,] 0.32812500 0.9843750
## [52,] 0.45312500 0.3593750
## [53,] 0.95312500 0.8593750
## [54,] 0.70312500 0.1093750
## [55,] 0.20312500 0.6093750
## [56,] 0.14062500 0.4218750
## [57,] 0.64062500 0.9218750
## [58,] 0.89062500 0.1718750
## [59,] 0.39062500 0.6718750
## [60,] 0.26562500 0.0468750
## [61,] 0.76562500 0.5468750
## [62,] 0.51562500 0.2968750
## [63,] 0.01562500 0.7968750
## [64,] 0.02343750 0.3984375
## [65,] 0.52343750 0.8984375
## [66,] 0.77343750 0.1484375
## [67,] 0.27343750 0.6484375
## [68,] 0.39843750 0.0234375
## [69,] 0.89843750 0.5234375
## [70,] 0.64843750 0.2734375
## [71,] 0.14843750 0.7734375
## [72,] 0.21093750 0.2109375
## [73,] 0.71093750 0.7109375
## [74,] 0.96093750 0.4609375
## [75,] 0.46093750 0.9609375
## [76,] 0.33593750 0.3359375
## [77,] 0.83593750 0.8359375
## [78,] 0.58593750 0.0859375
## [79,] 0.08593750 0.5859375
## [80,] 0.11718750 0.1171875
## [81,] 0.61718750 0.6171875
## [82,] 0.86718750 0.3671875
## [83,] 0.36718750 0.8671875
##  [84,] 0.49218750 0.4921875
##  [85,] 0.99218750 0.9921875
##  [86,] 0.74218750 0.2421875
##  [87,] 0.24218750 0.7421875
##  [88,] 0.17968750 0.3046875
##  [89,] 0.67968750 0.8046875
##  [90,] 0.92968750 0.0546875
##  [91,] 0.42968750 0.5546875
##  [92,] 0.30468750 0.1796875
##  [93,] 0.80468750 0.6796875
##  [94,] 0.55468750 0.4296875
##  [95,] 0.05468750 0.9296875
##  [96,] 0.03906250 0.1328125
##  [97,] 0.53906250 0.6328125
##  [98,] 0.78906250 0.3828125
##  [99,] 0.28906250 0.8828125
## [100,] 0.41406250 0.2578125
## [101,] 0.91406250 0.7578125
## [102,] 0.66406250 0.0078125
## [103,] 0.16406250 0.5078125
## [104,] 0.22656250 0.4453125
## [105,] 0.72656250 0.9453125
## [106,] 0.97656250 0.1953125
## [107,] 0.47656250 0.6953125
## [108,] 0.35156250 0.0703125
## [109,] 0.85156250 0.5703125
## [110,] 0.60156250 0.3203125
## [111,] 0.10156250 0.8203125
## [112,] 0.07031250 0.3515625
## [113,] 0.57031250 0.8515625
## [114,] 0.82031250 0.1015625
## [115,] 0.32031250 0.6015625
## [116,] 0.44531250 0.2265625
## [117,] 0.94531250 0.7265625
## [118,] 0.69531250 0.4765625
## [119,] 0.19531250 0.9765625
## [120,] 0.13281250 0.0390625
## [121,] 0.63281250 0.5390625
## [122,] 0.88281250 0.2890625
## [123,] 0.38281250 0.7890625
## [124,] 0.25781250 0.4140625
## [125,] 0.75781250 0.9140625
## [126,] 0.50781250 0.1640625
## [127,] 0.00781250 0.6640625
## [128,] 0.01171875 0.3320312
```

```
plot(mat[, "x1"], mat[, "x2"])
```



Each row in the matrix just printed corresponds to a point in the plot. There are 128 rows, so we have 128 points. These points are your sampling points in the model's uncertain space, and can be thought of as coordinates. If you have a look at the first row of the matrix printed above, you will see that $x_1 = x_2 = 0.5$: this means that the model will compute the output in that row by assuming that $x_1 = x_2 = 0.5$. In the second row, the model will assume that $x_1 = 0.75$ and $x_2 = 0.25$, and so on until the 128th row. At the end of the day, you will know which values your model output gets in each one of the sampling points.

In this example we can graphically represent the model's uncertain space with a plane because our model has two dimensions and therefore is a two-dimensional model. The $x-$ and the y-axis in the plot represent the first and the second columns of the sample matrix. When dealing with k-dimensional models, where $k > 3$, a graphical representation of the model's uncertain space like the one above is not possible because we live in a three-dimensional world.

We have built this sample matrix using Quasi-Random Numbers (QRN), a specific method to sample the unit hypercube that differs from other methods such as random numbers (R) or Latin Hypercube Sampling (LHS). Let us compare the three methods to see how they differ.

First, we define a vector with the three different sampling methods available in `sensobol`.

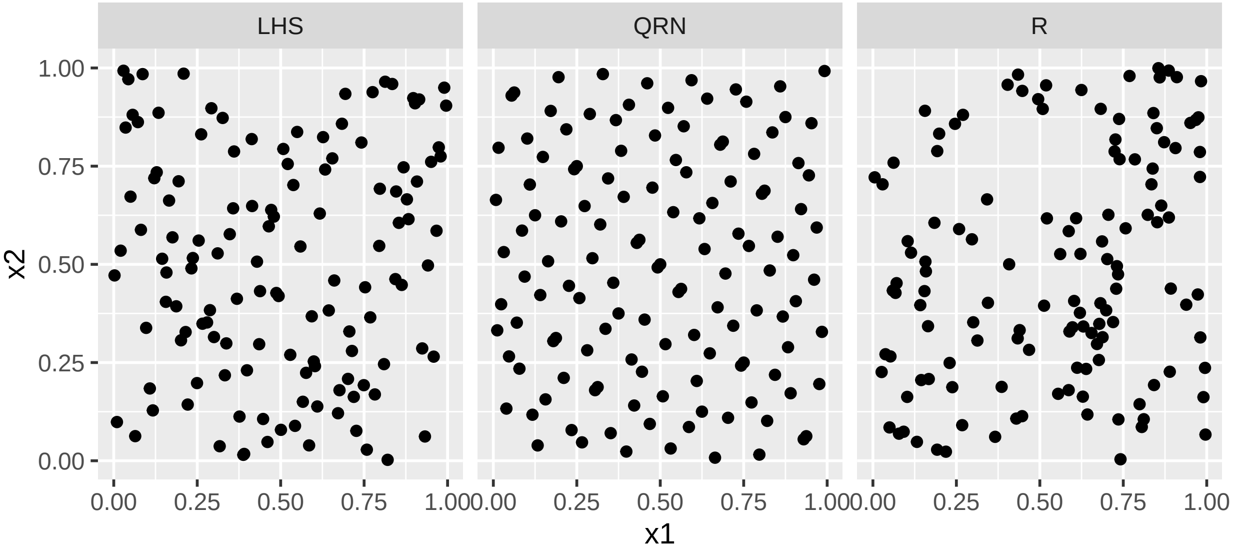
```
all_types <- c("QRN", "LHS", "R")
```

We then loop over all these methods to create a matrix for each sampling method. The we name the slots and plot the resulting matrices for a better visualization. The code is the following:

```
# Loop
prove <- lapply(all_types, function(type)
  sobol_matrices(matrices = matrices, N = N, params = params, type = type))

# Name
names(prove) <- all_types

# Plot
lapply(prove, data.table) %>%
  rbindlist(., idcol = "Method") %>%
  ggplot(., aes(x1, x2)) +
  geom_point() +
  facet_wrap(~Method)
```



Which differences do you see between the sampling space created by LHS, QRN and R?

Take-home information: QRN is the preferred sampling method in UA/SA because it leaves smaller unexplored volumes and hence maps more effectively the uncertain space. However, note that random methods might be better to compute sensitivity indices when the model under examination has important high-order terms (Kucherenko et al. 2011). We will explore SA later on.

Dummy example (2)

In this example we will use a very simple model with three uncertain parameters. Let us first name our parameters x_1, x_2 and x_3:

```
params <- c("x1", "x2", "x3")
```

And create again the sample matrix with the function `sobol_matrices` from the `sensobol` package:

```
mat <- sobol_matrices(matrices = matrices, N = N, params = params, type = type)
```

We now define the model. We will use a very simple polynomial:

$$y = 3x_1^2 + 2x_1x_2 - 2x_3$$

In the next code snippet we code the polynomial in a function, which we label `dummy_fun`. Note that the function needs to be coded as to run rowwise throughout the sample matrix.

```
dummy_fun <- function(mat) 3 * mat[, "x1"]^2 + 2 * mat[, "x1"] * mat[, "x2"] - 2 * mat[, "x3"]
```

We execute the model in the sample matrix and print the output.

```
y <- dummy_fun(mat)
y
```

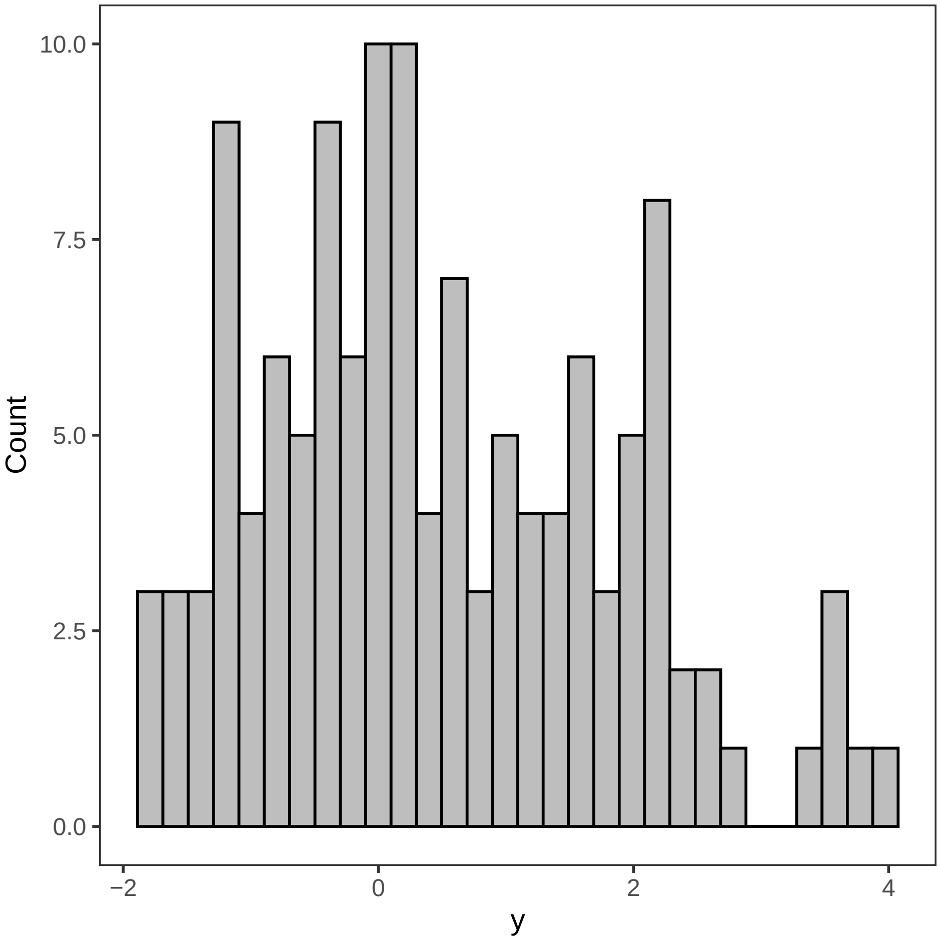
```
##   [1]  0.25000000  0.56250000  0.06250000 -0.54687500  3.57812500
##   [6]  0.57812500 -1.54687500 -0.40234375  0.91015625  1.62890625
##  [11]  0.94140625 -1.46484375  2.22265625  1.06640625 -1.24609375
##  [16] -1.57324219  1.52050781  2.31738281 -0.33886719 -0.19042969
##  [21]  2.02832031  0.60644531  0.07519531 -0.94042969  2.09082031
##  [26]  2.63769531 -0.33105469  0.08300781  1.73925781 -0.93261719
##  [31] -0.77636719 -1.18725586  1.51586914  1.21118164 -1.14819336
##  [36]  0.18383789  2.26196289 -0.08959961 -0.57397461 -1.39233398
##  [41]  1.87329102  3.45922852  0.41235352 -0.13061523  2.01000977
##  [46]  0.01782227 -0.15405273 -0.47631836  0.32055664  1.82836914
##  [51]  0.93774414 -0.83959961  3.58227539  1.35571289 -0.90991211
##  [56]  0.02172852  1.25610352  1.02954102  0.32641602 -1.16967773
##  [61]  2.18969727  0.19750977 -1.88061523 -0.87030029 -0.12811279
##  [66]  0.63360596  0.18829346 -1.14569092  2.72149658  1.47540283
##  [71] -0.84490967 -0.04315186  1.26153564  1.89044189  0.75762939
##  [76] -0.45135498  3.47833252  0.61505127 -1.39276123 -1.25946045
##  [81]  1.57647705  2.06475830 -0.78680420  1.13311768  3.84405518
##  [86]  0.43389893 -0.04266357 -1.74676514  1.52667236  2.24151611
##  [91] -0.42254639 -0.31512451  1.33331299  0.19659424 -0.09246826
##  [96] -1.84442139  0.69464111  2.11260986 -0.59832764  0.11846924
## [101]  2.28253174  0.22393799  0.13800049 -1.12860107  2.47296143
## [106]  2.25811768 -0.64031982  0.18585205  1.91241455 -0.26336670
## [111] -0.53680420 -0.35760498  0.52520752  0.26348877 -0.22869873
## [116] -0.37518311  3.88262939  1.44122314 -1.17596436 -0.73358154
## [121]  0.08673096  1.55157471  0.74688721 -1.13397217  2.56134033
## [126]  0.89337158 -1.03631592 -1.56211853
```

As you can see, the model output is a vector whose length equals the number of rows of the sample matrix, 128 in this case. The first number in the vector is the model output produced when x_1, x_2 and x_3 take the values defined in the first row of the sampling matrix; the second number is the model output produced when x_1, x_2 and x_3 take the values defined in the second row of the sampling matrix; and so on.

With this vector we can conduct a proper UA of the model output. We can first visualize the distribution of the output with an histogram. This is easy to do with the `plot_uncertainty` function of the `sensobol` package:

```
plot_uncertainty(Y = y, N = N) +
  geom_histogram(fill = "grey", color = "black") # This last line of code is just to fill
```

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



```
# the histogram with grey colour.
```

Once we get an idea of the shape of the distribution, we can compute some statistical measures to describe the data. Which descriptive statistics will you select in this case?

Take-home information: When the output distribution is not normal (i.e., when it is not Gaussian, not symmetric around the mean; in other words, when the distribution does not look like a “bell curve”), the mean or the standard deviation may not be the most appropriate measures of central tendency and spread. Other options, such as the median, quartiles, the interquartile range (the difference between the 75th and the 25th quartile), may be better suited. Always plot your data to see how it is distributed.

We can compute the mean, median, interquartile range and quantiles of the model output as follows:

```
mean(y)
```

```
## [1] 0.4782439
```

```
median(y)
```

```
## [1] 0.197052
```

```
IQR(y)
```

```
## [1] 2.056351
```

```
quantile(y)
```

```
##          0%        25%        50%        75%       100%
## -1.8806152 -0.5393219  0.1970520  1.5170288  3.8826294
```

An hydrological example: calculation of irrigation water withdrawals

In this section we will conduct a full-fledged uncertainty analysis of one of the formulae used to model irrigation water withdrawals. This model requires information on the extension of irrigation, crop coefficient, evapotranspiration, precipitation and irrigation efficiency (Puy, Sheikholeslami, et al. 2022). It reads as follows:

$$y = \frac{A(ET_c - P)}{E} \,,$$

where y represent irrigation water withdrawals [m^3], A is the extension of irrigation [m^2], ET_c is the crop evapotranspiration [m], P is the precipitation [m] and E is the irrigation efficiency [-].

Let us assume that we do not know the exact value of any of these inputs, but that we have done our research and agreed that their uncertainty can be fairly approximated with the distributions presented in the Table 1 below.

Table 1: Table 1. Distribution of the uncertain inputs for the irrigation water withdrawal model.

Parameter	Description	Distribution
A	Irrigated area	$\mathcal{U}(15, 20)$
ET_c	Crop evapotranspiration	$\mathcal{N}(0.4, 0.05)$
P	Precipitation	$\mathcal{U}(0, 0.1)$
E	Irrigation efficiency	$\mathcal{U}(0.4, 0.6)$

In this example we will increase the sample size to $N = 2^{12}$ to ensure a thorough sampling of the uncertain space:

```r
# Sample size
N <- 2^12
```

As in the previous examples, we first define a vector with the name of our uncertain inputs:

```r
params <- c("A", "ET_c", "P", "E")
```

We then create the sample matrix:

```r
mat <- sobol_matrices(matrices = matrices, N = N, params = params, type = type)
```

Let us inspect the first five rows of the resulting matrix:

```r
head(mat)
```

```
##          A  ET_c     P     E
## [1,] 0.500 0.500 0.500 0.500
## [2,] 0.750 0.250 0.750 0.250
## [3,] 0.250 0.750 0.250 0.750
## [4,] 0.375 0.375 0.625 0.125
## [5,] 0.875 0.875 0.125 0.625
## [6,] 0.625 0.125 0.375 0.375
```

Note that all the inputs are uniformly distributed in $[0, 1]$. Since our inputs have other distributions (see table above), we need to apply a quantile transformation to bring each input to its appropriate distribution. This is done in `R` as follows:

```
mat[, "A"] <- qunif(mat[, "A"], min = 15, max = 20) # for a uniform distribution.
mat[, "ET_c"] <- qnorm(mat[, "ET_c"], mean = 0.4, sd = 0.05) # for a normal distribution.
mat[, "P"] <- qunif(mat[, "P"], min = 0, max = 0.1)
mat[, "E"] <- qunif(mat[, "E"], min = 0.4, max = 0.6)
```

Now let us plot the first five rows to inspect the resulting transformed matrix:

```
head(mat)
```

```
##            A      ET_c      P     E
## [1,] 17.500 0.4000000 0.0500 0.500
## [2,] 18.750 0.3662755 0.0750 0.450
## [3,] 16.250 0.4337245 0.0250 0.550
## [4,] 16.875 0.3840680 0.0625 0.425
## [5,] 19.375 0.4575175 0.0125 0.525
## [6,] 18.125 0.3424825 0.0375 0.475
```

The inputs are no longer distributed in $\mathcal{U}(0, 1)$, but follow the distributions listed in Table 1.

We can now create a function that codes the irrigation model:

```
irrigation_fun <- function(mat) (mat[, "A"] * (mat[, "ET_c"] - mat[, "P"])) / mat[, "E"]
```

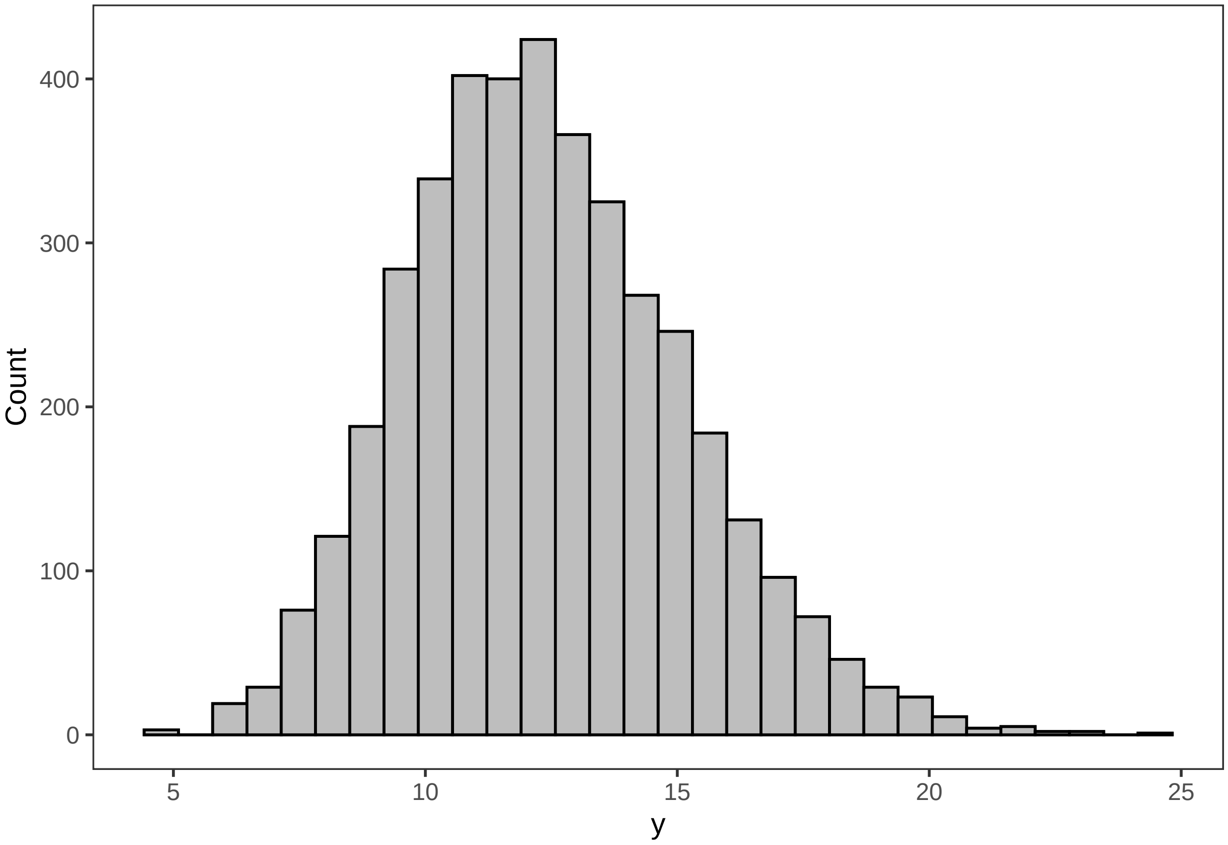
and run it throughout the sample matrix:

```
y <- irrigation_fun(mat)
```

Let us observe the distribution of the model output:

```
plot_uncertainty(N = N, Y = y) +
  geom_histogram(fill = "grey", color = "black") # This last line of code is just to fill
```

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



```
# the histogram with grey colour
```

. . . and compute a few statistics: we wrap the functions `min`, `max`, `mean` and `median` in a vector to make the code less verbose and execute one after the other:

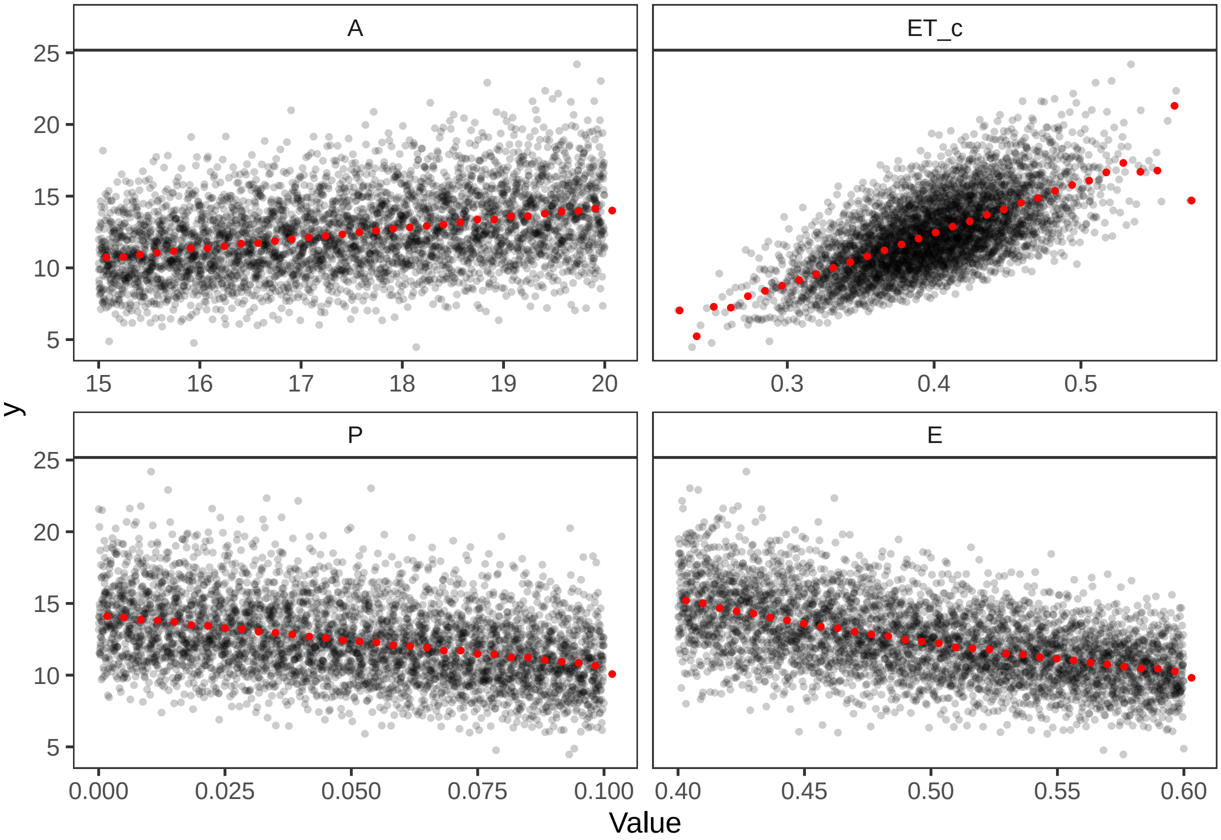
```
f <- c(min, max, mean, median)
sapply(f, function(f) f(y))
```

```
## [1]  4.471539 24.192795 12.416927 12.181732
```

Note how the range goes from 4 m^3 to 24 m^3, with most values concentrating around 10 m^3 and 15 m^3. The extreme values on the right tail of the distribution are produced only by a few model runs, and we may be interested in knowing more about which settings in A, ET_c, P and E give rise to such extreme irrigation water withdrawals.

To that aim, we use the function `plot_scatter`, of the `sensobol` package. This function displays the model output y against the x_i parameter while showing the mean y value.

```
plot_scatter(data = mat, N = N, Y = y, params = params)
```



Note how the highest y values are caused by the combination of high values in A (the area under irrigation) and ET_c (crop evapotranspiration), and low values in P (precipitation) and E (irrigation efficiency). This makes sense, as larger, poorly-efficient irrigated areas in drier environments with water stressed crops will tend to demand more water.

Scatterplots of y against x_i also inform on which input may convey the most uncertainty to the model output, which is one of the aims of SA. Which input do you think is the most influential in this case?

Take-home message: Scatterplots of y against x_i help the analyst identify patterns denoting sensitivity. In general, inputs showing more "shape" than the rest have a higher influence on y.

Scatterplots are often followed by a thorough SA, which is beyond the scope of this tutorial. Those interested in SA will find in Saltelli et al. (2008) and Puy, Lo Piano, et al. (2022) the information needed to become acquainted with variance-based SA (considered the gold standard in SA), and in Razavi et al. (2021) and Pianosi et al. (2016) an overview of the state of the art.

References

Dowle, M., and A. Srinivasan. 2021. *data.table: Extension of "Data.frame"*. Manual. https://CRAN.R-project.org/package=data.table.

Kucherenko, S., B. Feil, N. Shah, and W. Mauntz. 2011. "The Identification of Model Effective Dimensions Using Global Sensitivity Analysis." *Reliability Engineering & System Safety* 96 (4): 440–49. https://doi.org/10.1016/j.ress.2010.11.003.

Pianosi, Francesca, Keith Beven, Jim Freer, Jim W. Hall, Jonathan Rougier, David B. Stephenson, and Thorsten Wagener. 2016. "Sensitivity Analysis of Environmental Models: A Systematic Review with Practical Workflow." *Environmental Modelling and Software* 79: 214–32. https://doi.org/10.1016/j.envsoft.2016.02.008.

Puy, Arnald, Samuele Lo Piano, Andrea Saltelli, and Simon A. Levin. 2022. "Sensobol: An R Package to Compute Variance-Based Sensitivity Indices." *Journal of Statistical Software* 102 (5): 1–37. https:

//doi.org/10.18637/jss.v102.i05.

Puy, Arnald, Razi Sheikholeslami, Hoshin V. Gupta, Jim W. Hall, Bruce Lankford, Samuele Lo Piano, Jonas Meier, et al. 2022. "The Delusive Accuracy of Global Irrigation Water Withdrawal Estimates." *Nature Communications* 13: 3183. https://doi.org/10.1038/s41467-022-30731-8.

Razavi, Saman, Anthony Jakeman, Andrea Saltelli, Clémentine Prieur, Bertrand Iooss, Emanuele Borgonovo, Elmar Plischke, et al. 2021. "The Future of Sensitivity Analysis: An Essential Discipline for Systems Modeling and Policy Support." *Environmental Modelling & Software* 137: 104954. https://doi.org/10.1016/j.envsoft.2020.104954.

Saltelli, Andrea, Gabriele Bammer, Isabelle Bruno, Erica Charters, Monica Di Fiore, Emmanuel Didier, Wendy Nelson Espeland, et al. 2020. "Five Ways to Ensure That Models Serve Society: A Manifesto." *Nature* 582 (7813): 482–84. https://doi.org/10.1038/d41586-020-01812-9.

Saltelli, Andrea, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. 2008. *Global Sensitivity Analysis. The Primer*. Chichester, UK: John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470725184.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.