SENSITIVITY ANALYSIS, UNCERTAINTY PROPAGATION AND INTENSIVE OPTIMIZATION WITH FAST SIMULATORS

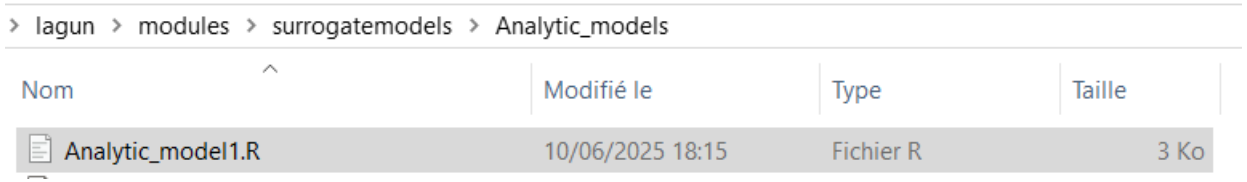
CONTEXT

- Your simulator is sufficiently fast/cheap to be called intensively for Monte-Carlo sampling
  - For intensive exploration
  - For sensivity analysis
  - For uncertainty propagation
  - For “intensive” optimization : e.g. multi-objective optimization, robust optimization …

IDEA

Define your fast simulator as a user-defined surrogate model

- The user-defined Surrogate Models should be located in a folder in modules/surrogatemodels
  Here : *Analytic_models/Analytic_model1.R*



- Add the new folder (here *Analytic_models)* to the list of surrogates in the file «modules/surrogatemodels/SurrogateFolders.R »

DEFINE YOUR FAST SIMULATOR AS A USER-DEFINED SURROGATE MODEL

1. The « build » function is used only to define the model structure

```
Analytic_model1.build <- function(Xmodel, y, Ytype){
# Build the metamodel and return the output list
# Here: the model is fixed (no learning step)
    obj <- list()

    # This example is a 2d model with 2 linear outputs each defined by 3 coefficients
    # output1 = 1 + 2*x1 + 3*x2
    # output2 = -x1 + 2*x2
    coef0 = c(1,0)
    coef1 = c(2,-1)
    coef2 = c(3,2)

    # Store model coefficients in a structure to be used by predict function
    # the model structure can contain more complex information as the name of a code to be called

    obj$model <- list(coef0=coef0,coef1=coef1,coef2=coef2)

    # Returns directly y as predictions and Q2=1 because the model is fixed (no learning phase)
    obj$yloo <- y

    # In case of multiple outputs of the models, it is useful to store names of the expected output in model obj for prediction
    function
    obj$yname <- colnames(y)
  obj$Q2loo <- 1

  return(obj)
}
```

DEFINE YOUR FAST SIMULATOR AS A USER-DEFINED SURROGATE MODEL

2. The « predict » function calls the fast simulator based on the information stored in model structure (defined in « build » function)

```
Analytic_model1.predict <- function(obj, Xmodel, computesd){

  #
  npred <- nrow(Xmodel)
  ysd <- NULL #no variance of prediction

  model = obj$model

  # In case of multiple outputs of the models, retrieve the expected output (colnames of obj initialized in
  Analytic_model1.build function)
    if (obj$yname == "output1"){
          ymean = model$coef0[1] + model$coef1[1]*Xmodel[,1] + model$coef2[1]*Xmodel[,2]
    } else if (obj$yname == "output2"){
      ymean = model$coef0[2] + model$coef1[2]*Xmodel[,1] + model$coef2[2]*Xmodel[,2]
    }

  Outputs <- list(ymean = ymean, ysd=ysd)
  return(Outputs)
}
```

Your simulator !

DEFINE YOUR FAST SIMULATOR AS A USER-DEFINED SURROGATE MODEL

3. The « update » function is useless but mandatory

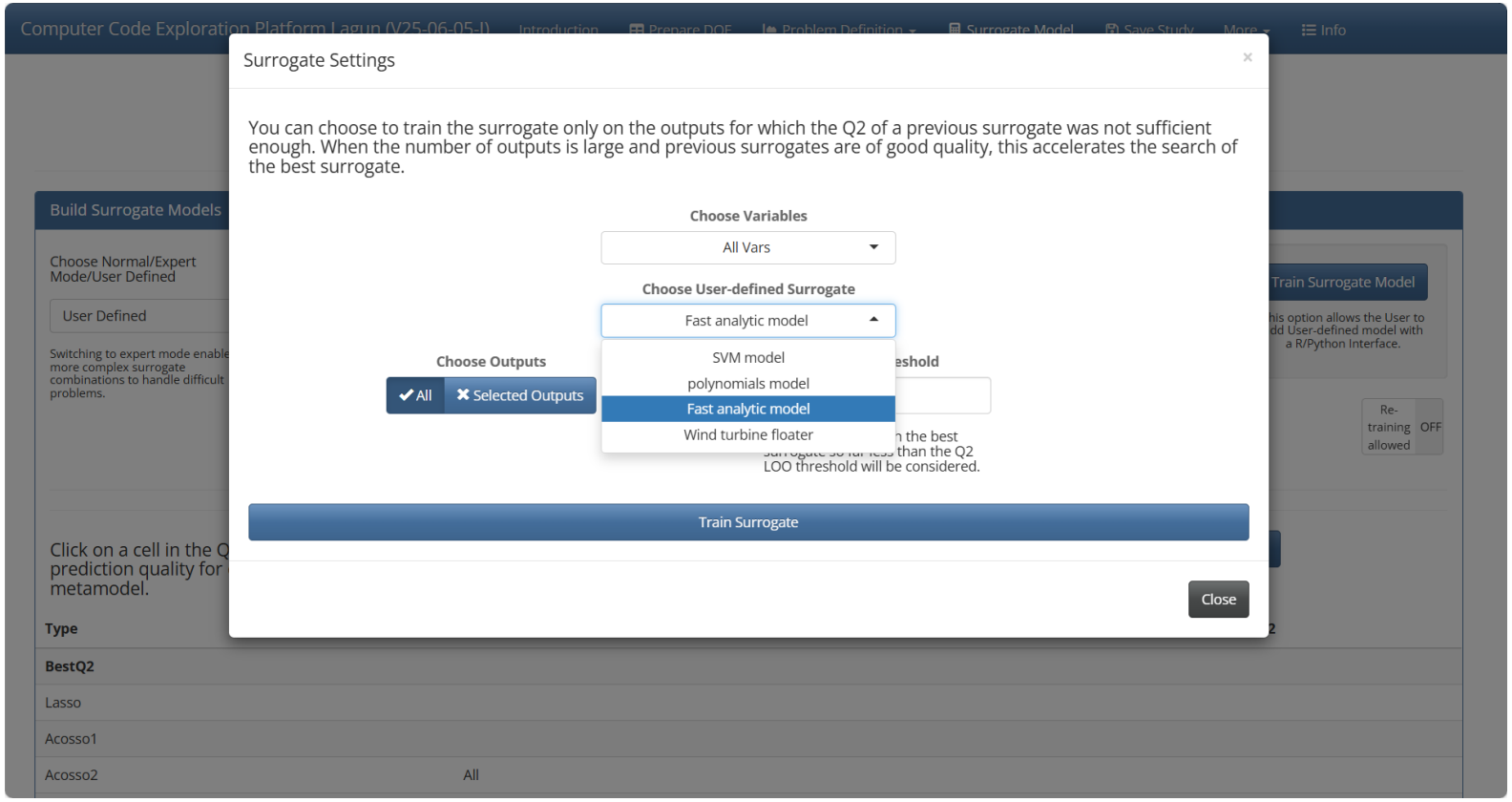
```
Analytic_model1.update <- function(obj, Xmodel, y){

    obj$yloo <- y
    obj$Q2loo <- 1

  return(obj)
}
```

4. Define the name of your model to be displayed in Lagun and its characteristics

```
Analytic_model1.description <- function(){
# Returns the characteristics and parameters of the surrogate model
#
# Display Name: SURROGATE
#
# OptimTags=list( "regression", "classification", "predict.sd", "CategoricalInputs")
#
# Warnings=list()
#
    DisplayName="Fast analytic model"

    Description="To illustrate how to plug in a fast simulator to perform SA, UQ ..."

    SurrogateTags=list(classification=F, regression=T, predict.sd = F, CategoricalInputs = F)
    Warnings=list()


    return(list(dispname=DisplayName, descr=Description, tags=SurrogateTags, warn=Warnings))
}
```

USE YOUR FAST SIMULATOR IN LAGUN

- Import a DOE with at least two different points to define the inputs and outputs of your simulator
- Surrogate model panel: select “user-defined”
- Select your fast simulator (here “Fast analytic model”)



REMARK

- You can use different types of simulators:
  - Python script : via reticulate package (See scipy example for python)
  - Executable (.exe)
  - ???

- For optimization (without uncertainties) based on your fast simulator:
  rather use the simulation-based optimization functionality (Tutorial 12), there is a longer list of available optimizers and functionalities 😉