Getting Started with ProSimPlus®

Case 7: Integrating Artificial Intelligence (AI) into ProSimPlus

fives
Industry can do it

Introduction

Artificial Intelligence (AI) and process simulation can work together to provide significant advances in the field of process engineering.

By exploiting advanced Machine Learning techniques, AI can analyze existing simulation models, extract key information and create faster predictive models.

These AI-powered models, also known as Surrogate Models, can significantly reduce computing times by optimizing complex process engineering operations.

Introduction

This document outlines the steps involved in creating a Surrogate Model in ProSimPlus.

The steps are as follows:

1. Creation of a dataset by ProSimPlus
2. Model training
3. Model deployment within ProSimPlus

Before delving into this chapter, it is highly recommended to refer to "Getting Started with ProSimPlus, Use Case 1," which introduces the primary features of ProSimPlus

References:
R. Bounaceur, O. Baudouin, "Couplage entre logiciel PSE et modèles fondés sur des algorithmes d'Intelligence Artificielle", tutorial SFGP 2022, Toulouse (2022)
R. Bounaceur *et al.*, "Development of an artificial intelligence model to predict combustion properties, with a focus on auto-ignition delay", J. Eng. Gas Turbines Power., 1-28 (2023)

Prerequisite

- Firstly, installation of the Python software is required.
- A few libraries for Python can be used, including (but not limited to):

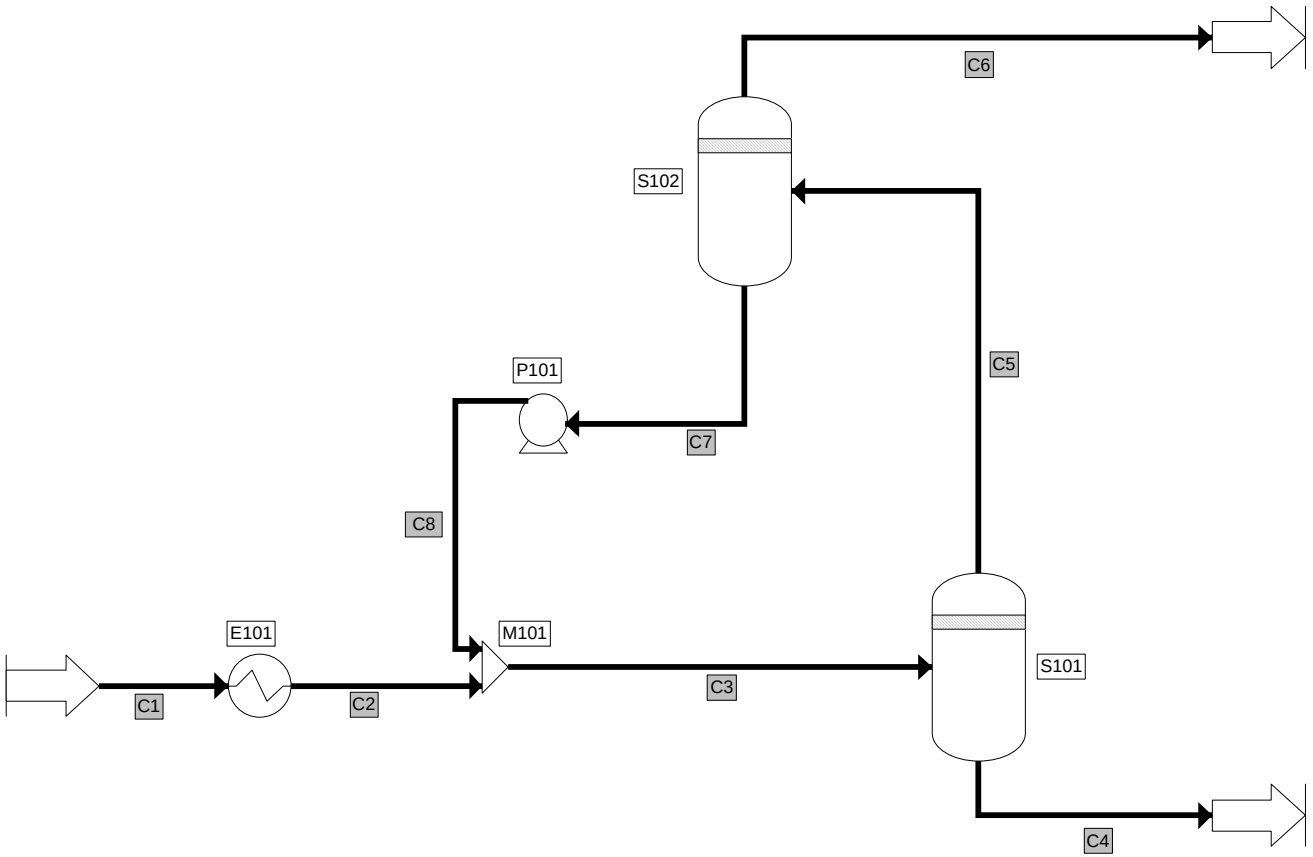
Library	Function
Pywin32	Integrate the surrogate model into ProSimPlus
NumPy	Manipulate matrices or multidimensional arrays in Python
Pandas	Manipulate objects (dataframe)
Matplotlib	Create graphics
Scikit-learn	Open-source Python library for machine learning
Jupyter	Create and execute Python code interactively, combine code, visualizations, explanatory text and results
JobLib	Save trained models and reload them later

- To install these libraries, open the command prompt and type:

pip install Library name

Introduction of the process: Simple Example

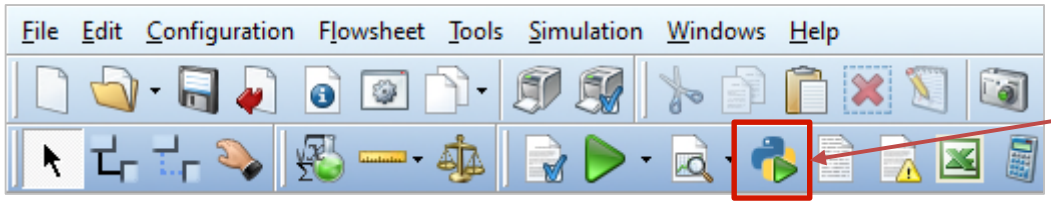
For this study, the example is based on a simple process available in the ProSimPlus samples directory under the name "PSPS_EX_EN-Simple-Example".



1. Creation of a dataset

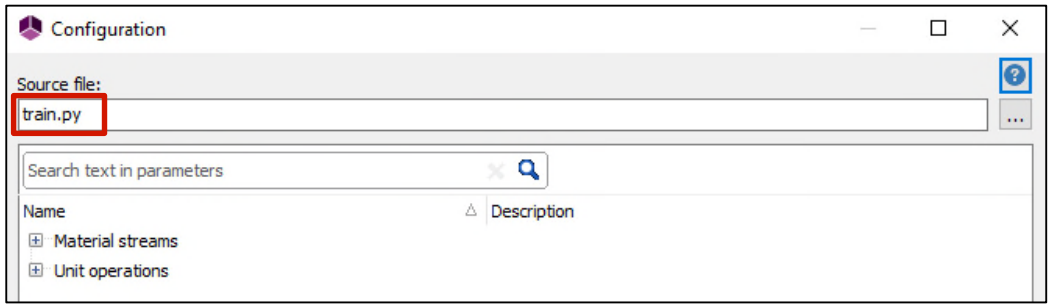
- The first step is to choose the training subject and generate a dataset.

- In the absence of experimental data, we will opt to use ProSimPlus with an external solver to generate the necessary values. An alternative would be to use the sensitivity analysis or the console version, but this would be less practical.



Click on this icon to access the external solver

- The Python source file that will be used to generate the dataset is called “train.py”



The detailed explanation of the parameters to be modified in the Python file can be found directly within the document: “train.py”

1. Creation of a dataset

- We need to specify the input values (module parameters) and run simulations to calculate the output values (results).
- We will ensure to avoid creating an input dataset that is entirely random or excessively linear, as it may not necessarily be representative of our model.
- To achieve this, we will use the Sobol sequences method, which will easily provide us with quasi-random values that adequately cover the selected intervals.

1. Creation of a dataset

- The "sobol.py" python source code is included with this document.
- The invocation of the Sobol sequences method is performed as follows:

```
tab = sobol.i4_sobol_generate(ndim, npoints, npass)
```

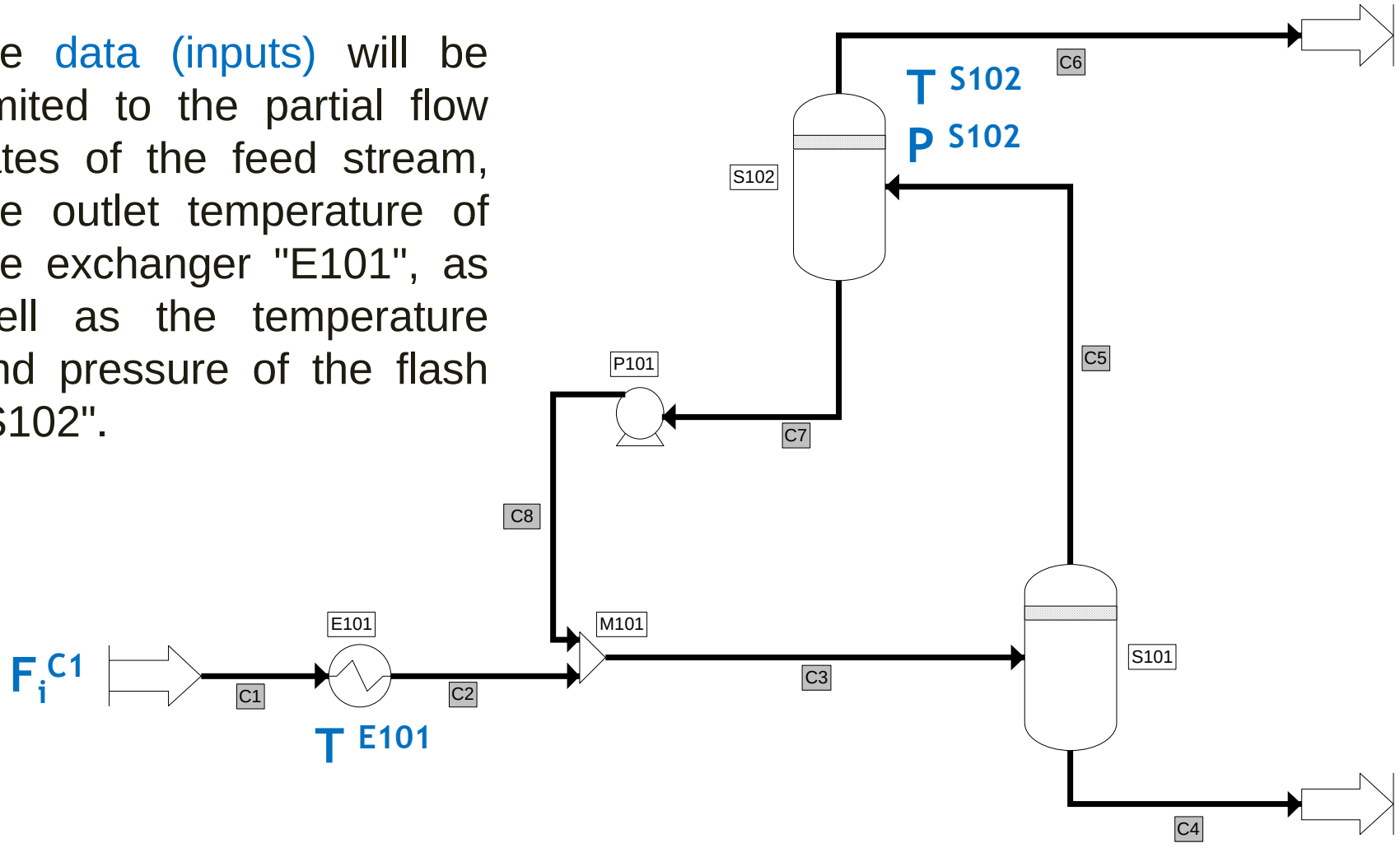
  - **ndim** is the number of dimensions (inputs) (from 1 to 40),
  - **npoints** is the total number of points,
  - **npass** is the number of initial points in the sequence to skip, in order to avoid encountering the first points that are nearly identical.
  - The returned value **tab** is a fixed-size homogeneous NumPy array, with dimensions corresponding to the provided arguments.

It is possible to use other sampling methods than Sobol sequences, but they are not explained in this document.

1. Creation of a dataset

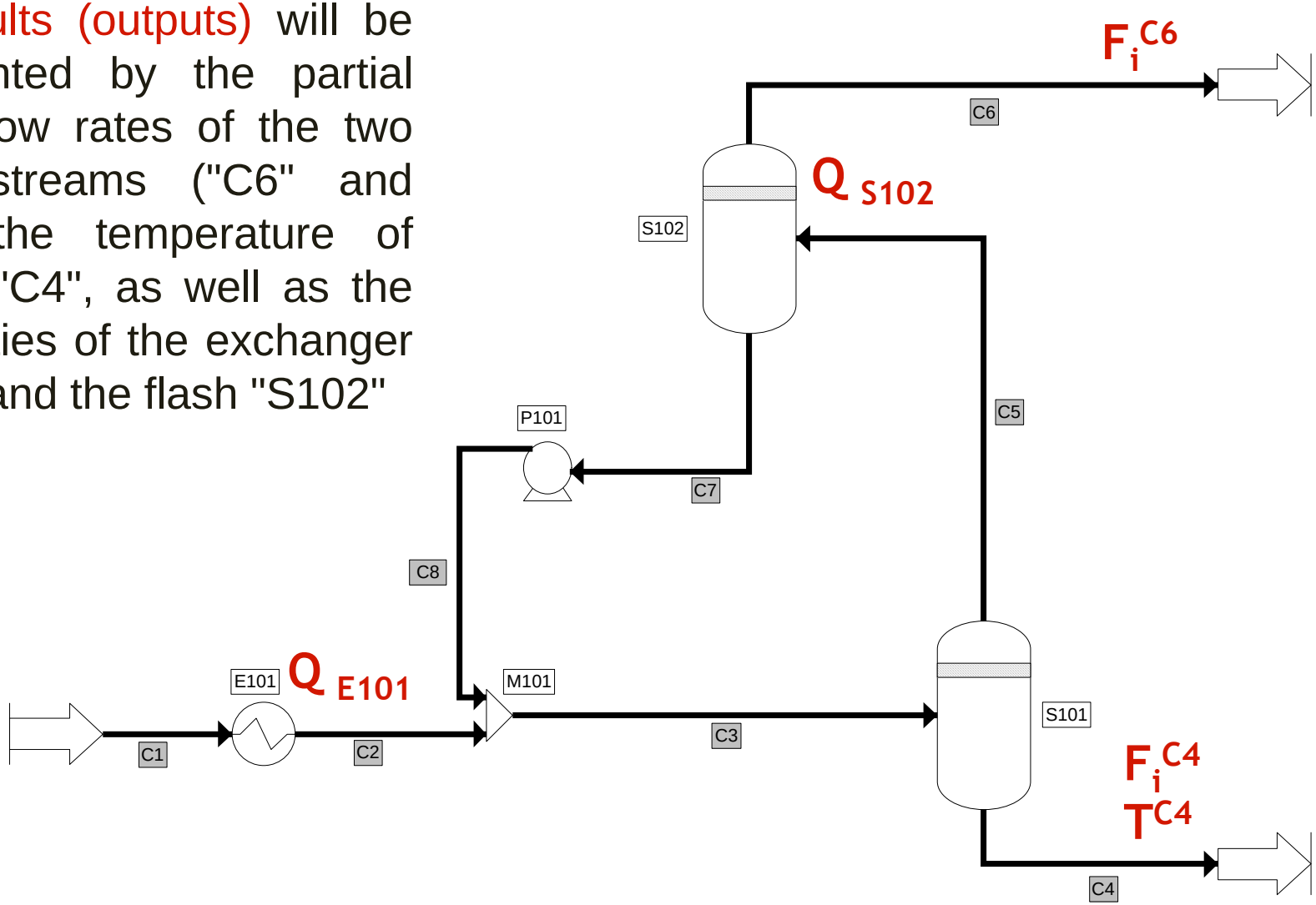
We are going to assume that:

- the data (inputs) will be limited to the partial flow rates of the feed stream, the outlet temperature of the exchanger "E101", as well as the temperature and pressure of the flash "S102".



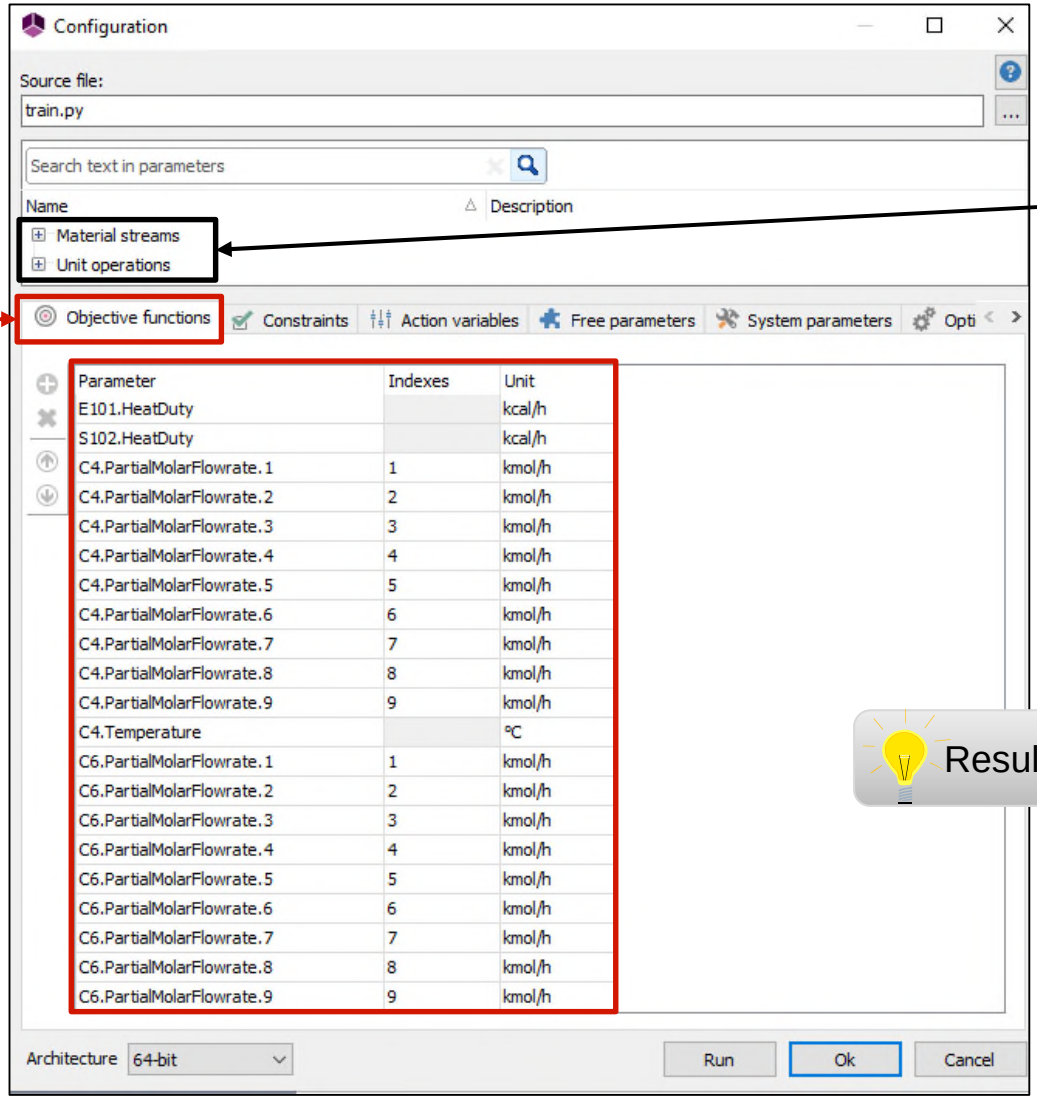
1. Creation of a dataset

- the results (outputs) will be represented by the partial molar flow rates of the two outlet streams ("C6" and "C4"), the temperature of stream "C4", as well as the heat duties of the exchanger "E101" and the flash "S102"



1. Creation of a dataset

To achieve this, we will configure the external solver by incorporating the results (outputs) into the “objective functions”:

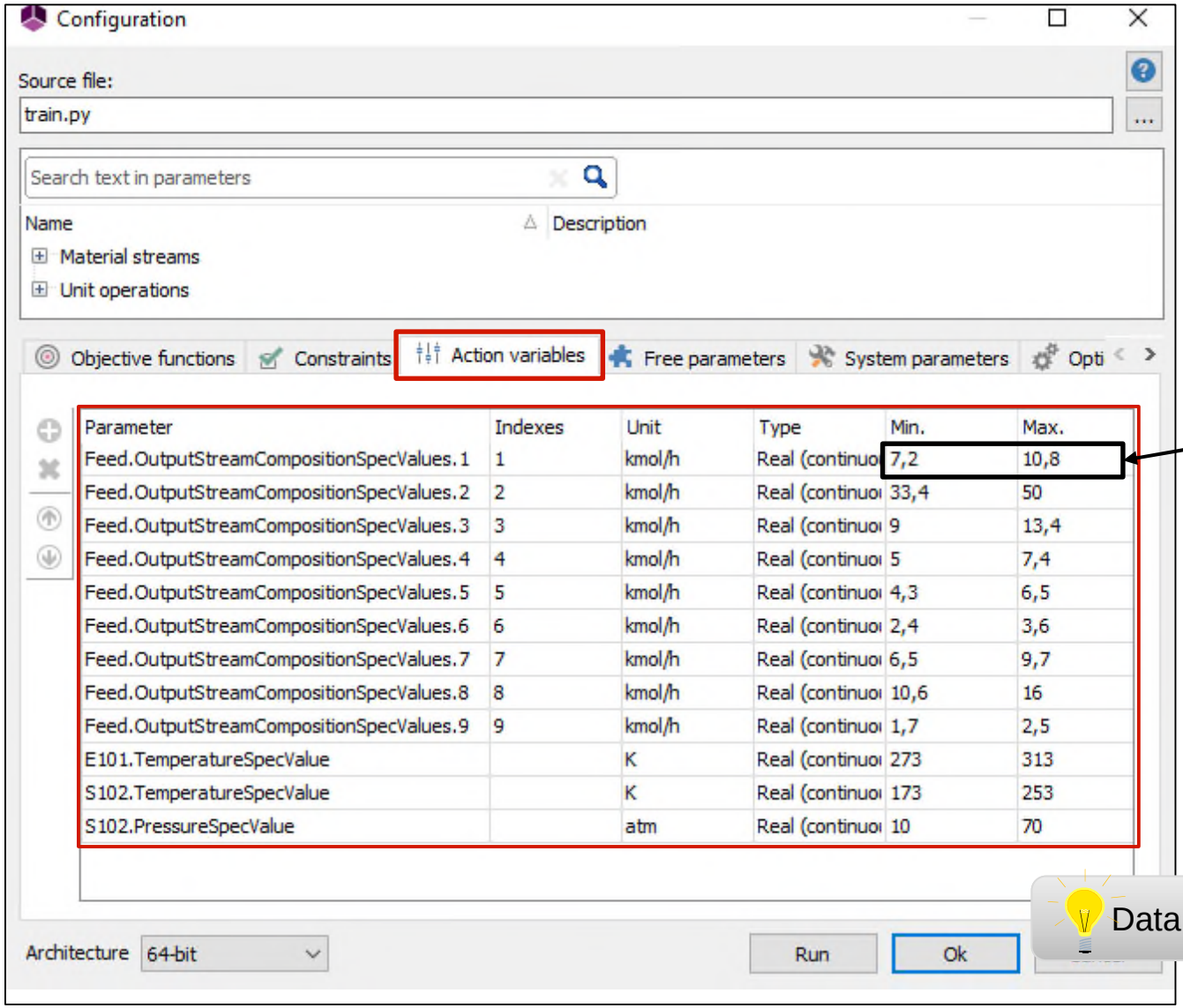


Double-click to access to the various parameters

Results (outputs) = Objective functions

1. Creation of a dataset

And the data (inputs) into the action variables:



Remember to set boundaries for the variables

Data (inputs) = Action variables

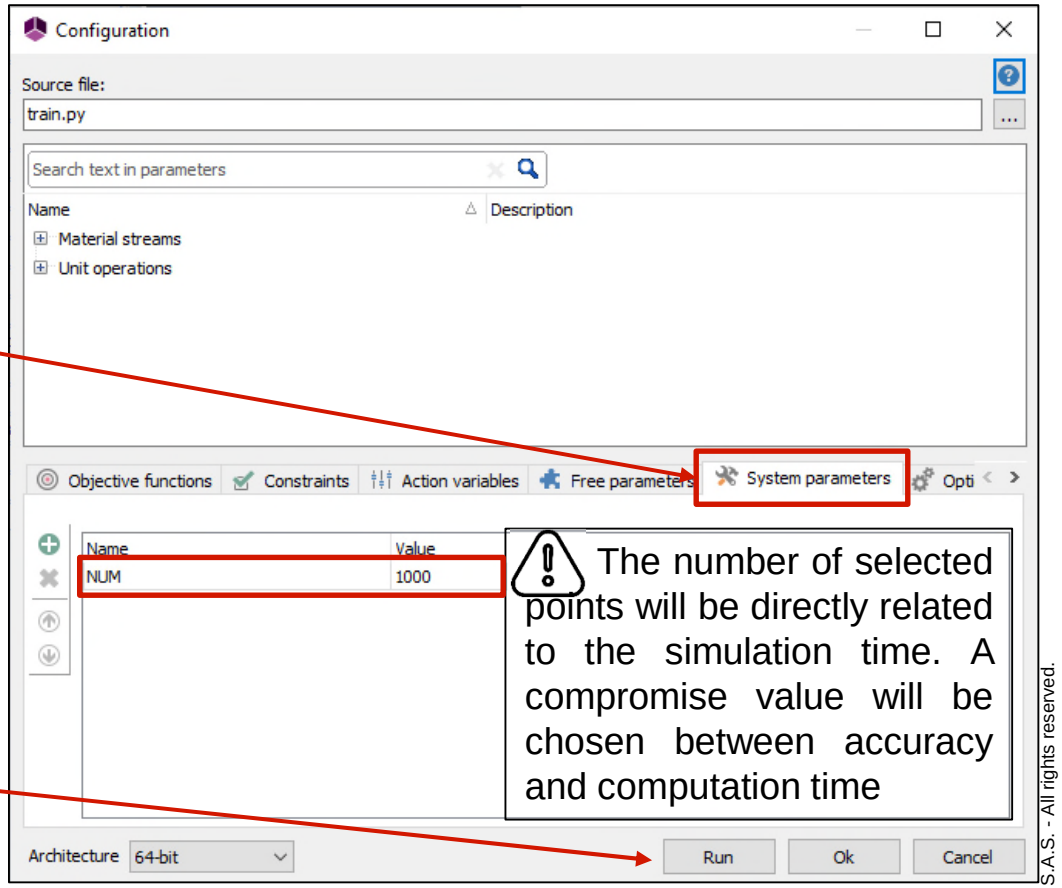
1. Creation of a dataset

- We also include a parameter named "NUM" in the "System parameters" tab to specify the number of points to generate. This parameter is used in the source file as the total number of points generated by the Sobol sequence.

⚠ Adapt the contents of the "train.py" file to the parameters defined previously

- Click on the "Run" button to perform simulations on the dataset.

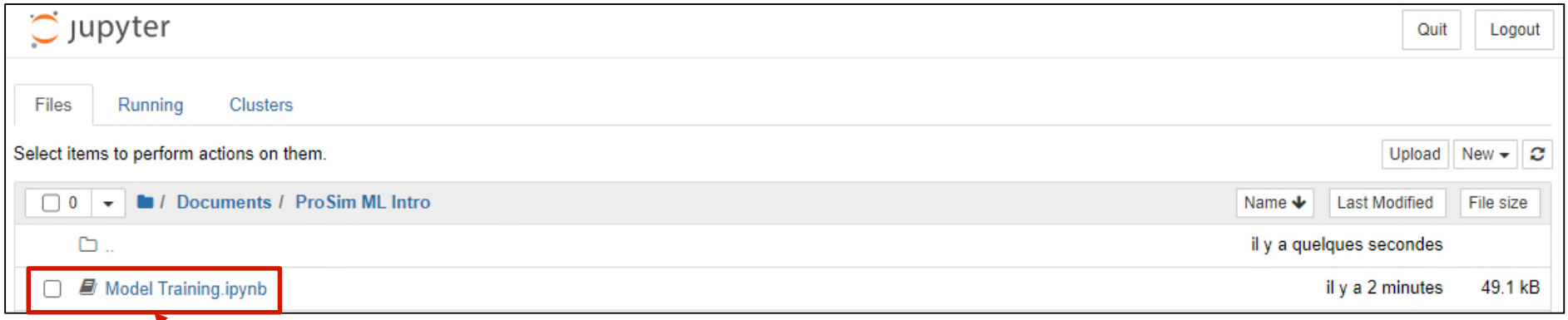
- If everything went well, we now have a dataset available in "traindata.txt".



2. Model training

To make things clearer and more understandable, this part will be carried out in Jupyter Notebook

- Open the command prompt and type the following: Jupyter notebook
- Open the notebook titled "Model Training.ipynb"



The explanation regarding the parameters to be modified for the model training process is detailed in this file

2. Model training

1. Data verification and cleaning

Data cleaning is a crucial step before creating a machine learning model. It involves taking measures to process and correct raw data so that the model can learn effectively and produce high-quality results.

The main steps of the process are as follows:

- **Data Collection**

- **Data Processing** (Handling incorrectly labeled data, dealing with “NaN” values…)

- **Normalization or Scaling**

If the various data inputs have significantly different scales, it is advisable to normalize or scale them to prevent certain inputs from excessively dominating the model.

- **Data Splitting**

Divide the data into **training** and **test** sets to evaluate the model's performance in an unbiased way. It is possible to choose the percentage associated to each set.

2. Model training

2. Machine Learning Algorithm

Various learning methods are available, and in this study, we are using a **KernelRidge regression model**, which is a regularized regression technique utilizing kernel methods for predictions. Other learning models could also be employed, such as for instance:

- kNN (k-Nearest Neighborhood)
- DecisionTree
- RandomForest
- MLP (Multi-Layer Perceptron) – Artificial Neural Networks

There is no one-size-fits-all solution to determine the best regression algorithm for all scenarios. The most recommended approach is to experiment with several models, evaluate them in terms of performance, and select the one that achieves the best results on the available dataset.

2. Model training

3. Evaluation parameters

Here are some commonly used metrics to evaluate the performance of learning models, especially in the case of regression:

- **Coefficient of Determination (R2 score)**

A high R2 score (≈1) indicates that the model effectively explains the data variation, while an R2 score close to 0 or negative suggests that the model does not accurately represent the data.

- **Mean Absolute Error (MAE)**

The MAE quantifies the average difference between the model's predicted values and the actual values.

- **Mean Squared Error (MSE)**

The MSE measures the average of the squared differences between predicted values and actual values. A lower MSE indicates that the model's predictions are closer to the actual values.

The obtained metrics should be in the same order of magnitude for the training and test sets.

2. Model training

4. Saving with Joblib

- After we have trained our model, we can utilize Joblib, a tool that facilitates saving data structures to a file so that they can be reloaded later (for instance, in ProSimPlus).

```
# To save, we use joblib, although there are other available options.
import joblib
joblib.dump(bestmodel, "model.joblib")
```

- Additionally, we store the minimum and maximum values of both the input data and the output results. As our model is normalized, these values are needed for the transformation of results back into their original scale.

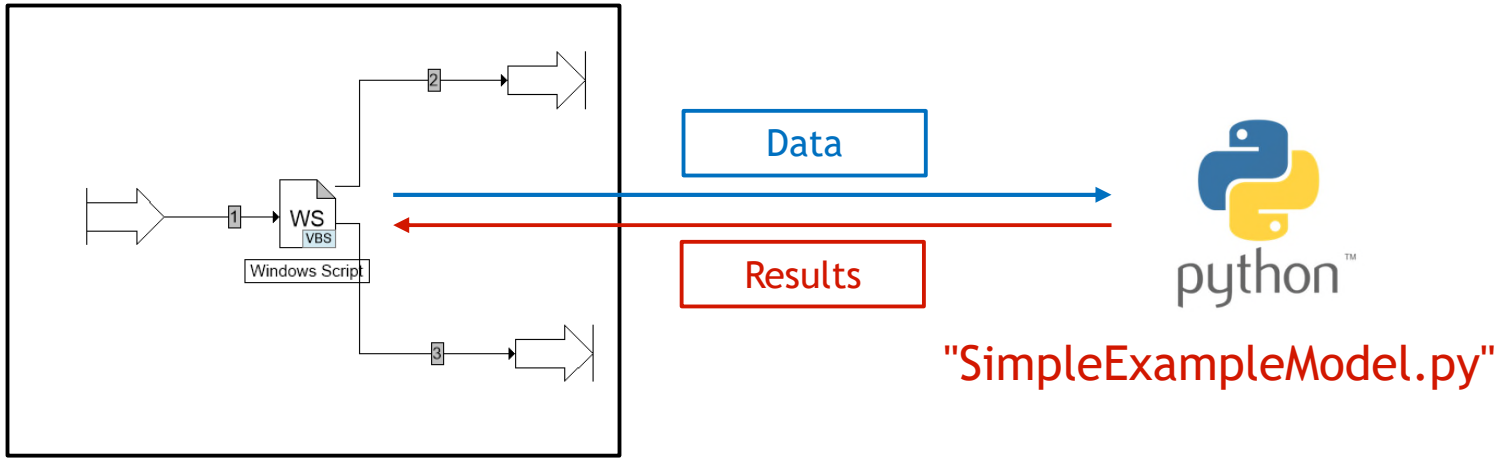
```
# Saving it with joblib
joblib.dump(minmax, "parameters.joblib")
```

3. Model deployment within ProSimPlus

Following the training phase of the model in Jupyter Notebook, a model is created and stored in the "model.joblib" file. The next step involves setting up the necessary framework to integrate it into ProSimPlus.

This structure is relatively simple and will consist of:

- A Windows Script unit operation
- A Python program able to make predictions by processing the data and providing the output results



3. Model deployment within ProSimPlus

We will now create a Python component following the Microsoft COM standard, which will be used from the VBScript code. The primary advantage of this method is that the Python program can remain in memory throughout the entire simulation.

The source code for this component can be found in the file "SimpleExampleModel.py".

The code is thoroughly described through comments within the file.

3. Model deployment within ProSimPlus

Here are some explanations about the codes:

- **_reg_progid_**

The ProgID (Programmatic Identifier) is a character string used to uniquely identify a class of COM (Component Object Model) objects that can be created and utilized within a script. This identifier is crucial and plays a pivotal role when instantiating an object in VBScript (CreateObject("My.progid")).

```
_reg_progid_ = "ProSim.SimpleExampleModel"
```

- **_reg_clsid_**

The GUID(Globally Unique Identifier) is a unique identifier used to reference the class.

```
#   https://guidgenerator.com/online-guid-generator.aspx for example. And don't forget the braces!
_reg_clsid_ = "{f8084d10-d072-4b8e-851f-4adeaac4d371}"
```

- **_public_methods_**

In this context, we are dealing with an array of strings that define the methods (functions or routines) that will be accessible within our class. For our example, we are introducing only two methods: one for performing calculations (predictions) and another for printing results in the report.

```
_public_methods_ = ["Calculate", "PrintResults"]
```

3. Model deployment within ProSimPlus

- **Using the "model.joblib" file**

The "Calculate" method will be used for making predictions.

```
def Calculate(self, F1, F2, F3, F4, F5, F6, F7, F8, F9, T_Ech, T_S102, P_S102)
```

To do this, the process involves retrieving the input data and predicting the results by simply calling the *'predict*' method. The use of this *'predict*' method becomes possible after loading the model that was previously saved using the joblib file.

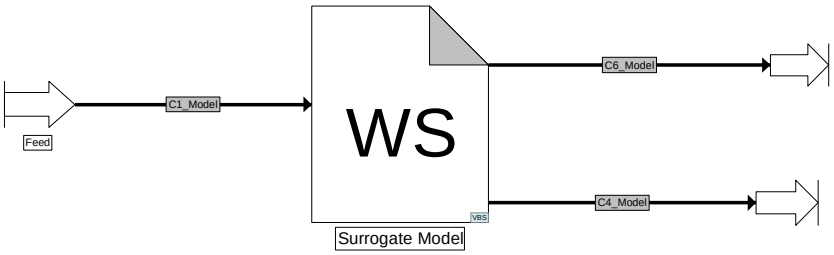
```
features = pandas.DataFrame(data, columns=["F1","F2","F3","F4","F5","F6","F7","F8","F9","T_Ech","T_S102","P_S102"])
res = model.predict(features)
```

The result is returned in the form of a two-dimensional array:

```
return res
```

3. Model deployment within ProSimPlus

A ProSimPlus test file utilizing this component is also available, named "SimpleExample.pmp3".



Extract of the script:

```
dim SimpleExample

Sub OnSimulationStart()
  set SimpleExample = createobject("ProSim.SimpleExampleModel")
End Sub
```

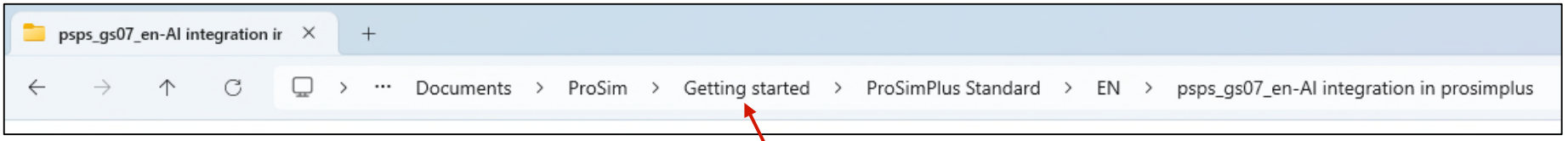
It's necessary to include the remaining process parameters (inputs):

In summary, the script performs the following actions: it creates an instance to obtain the ProgID of the class, retrieves the properties of the current input, incorporates the process parameters (inputs), generates the output streams, and retrieves the results returned by the training model.

3. Model deployment within ProSimPlus

The final step consists of simulating the process using the surrogate model we have built. To achieve this:

- Open the file directory:



Click on the bar address and type: cmd

- The command prompt window opens.

  Run the program with the command:

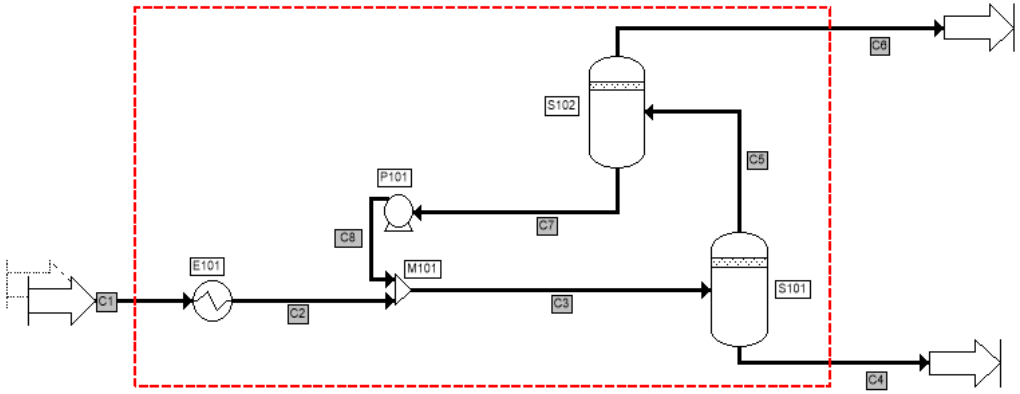
  python SimpleExampleModel.py --register

You only need to perform this step once.
This step is explained in detail in the python file: "SimpleExampleModel.py"

- Run the full simulation.

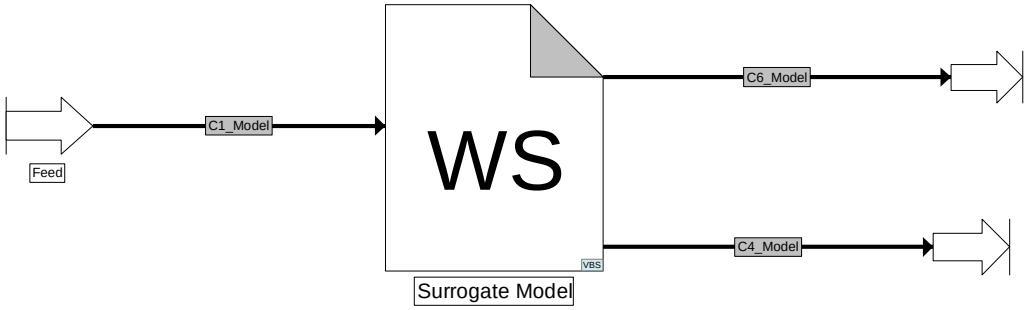
3. Model deployment within ProSimPlus

Example of the result:



Simulation with ProSimPlus

Streams		C1	C4	C4_Model	C6	C6_Model
From		Stream dupli...	S101	Surrogate M...	S102	Surrogate M...
To		E101	Process out...	Process out...	Process out...	Process out...
Partial flows (molar)		kmol/h	kmol/h	kmol/h	kmol/h	kmol/h
NITROGEN		9	1.6583	1.6401	7.3417	7.36
METHANE		41.7	15.493	15.23	26.207	26.471
ETHANE		11.2	8.1088	8.1265	3.0912	3.0727
PROPANE		6.2	5.5336	5.591	0.66639	0.60812
n-BUTANE		5.4	5.2748	5.2774	0.12516	0.12263
n-PENTANE		3	2.9895	2.9863	0.01049	0.013583
n-HEXANE		8.1	8.0981	8.0962	0.0018615	0.0035282
n-HEPTANE		13.3	13.3	13.3	0.00033901	0.00067467
n-OCTANE		2.1	2.1	2.0998	5.2835E-006	1.0028E-005
Total flow (mass)	kg/h	4332.2	3575.5	3573.6	756.65	758.52
Physical state		Liq./Vap.	Liquid	Liquid	Vapor	Liq./Vap.
Temperature	°C	40	14.561	15.598	-38	-38.15
Pressure	atm	75	74.7	75.994	25	25
Molar vapor fraction		0.43991	0	0	1	0.99974



***Surrogate* Model**

fives
Industry can do it

Fives ProSim S.A.S.
51, rue Ampère
Immeuble Stratège A
F-31670 Labège
France

Tel: +33 (0) 5 62 88 24 30

ProSim, Inc.
325 Chestnut Street,
Suite 800
Philadelphia, PA 19106
USA

Tel: +1 215 600 3759

www.fives-prosim.com

fives-prosim.info@fivesgroup.com