Démarrer avec ProSimPlus®

Cas 7 : Intégration de l'Intelligence Artificielle (IA) dans ProSimPlus

fives
Industry can do it

Introduction

L’Intelligence Artificielle (IA) et la simulation des procédés peuvent se combiner pour offrir des avancées significatives dans le domaine du génie des procédés.

En exploitant des techniques avancées d'apprentissage automatique (*Machine Learning*), l'IA peut analyser des modèles de simulation existants, extraire des informations clés et créer des modèles prédictifs plus rapides.

Ces modèles alimentés par l’IA, également appelée modèles de substitution (*Surrogate Models*), permettent de diminuer les temps de calcul en optimisant les processus complexes du génie des procédés.

Introduction

L’objectif de ce document est donc de parcourir les étapes nécessaires à la réalisation d’un modèle de substitution (*Surrogate Model*) dans ProSimPlus.

Les étapes seront :

1. Création d’un jeu de données par ProSimPlus
2. Entrainement d’un modèle
3. Utilisation du modèle dans ProSimPlus

Avant d’aborder ce chapitre, il est fortement recommandé de consulter
« Démarrer avec ProSimPlus, Cas 1 » présentant les principales fonctionnalités de ProSimPlus

Références :
R. Bounaceur, O. Baudouin, "Couplage entre logiciel PSE et modèles fondés sur des algorithmes d’Intelligence Artificielle", tutoriel 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)

Prérequis

- Tout d'abord, l'installation du logiciel Python est requis.
- Un certain nombre de bibliothèques pour Python peut être utilisé, comme par exemple (non exhaustif) :

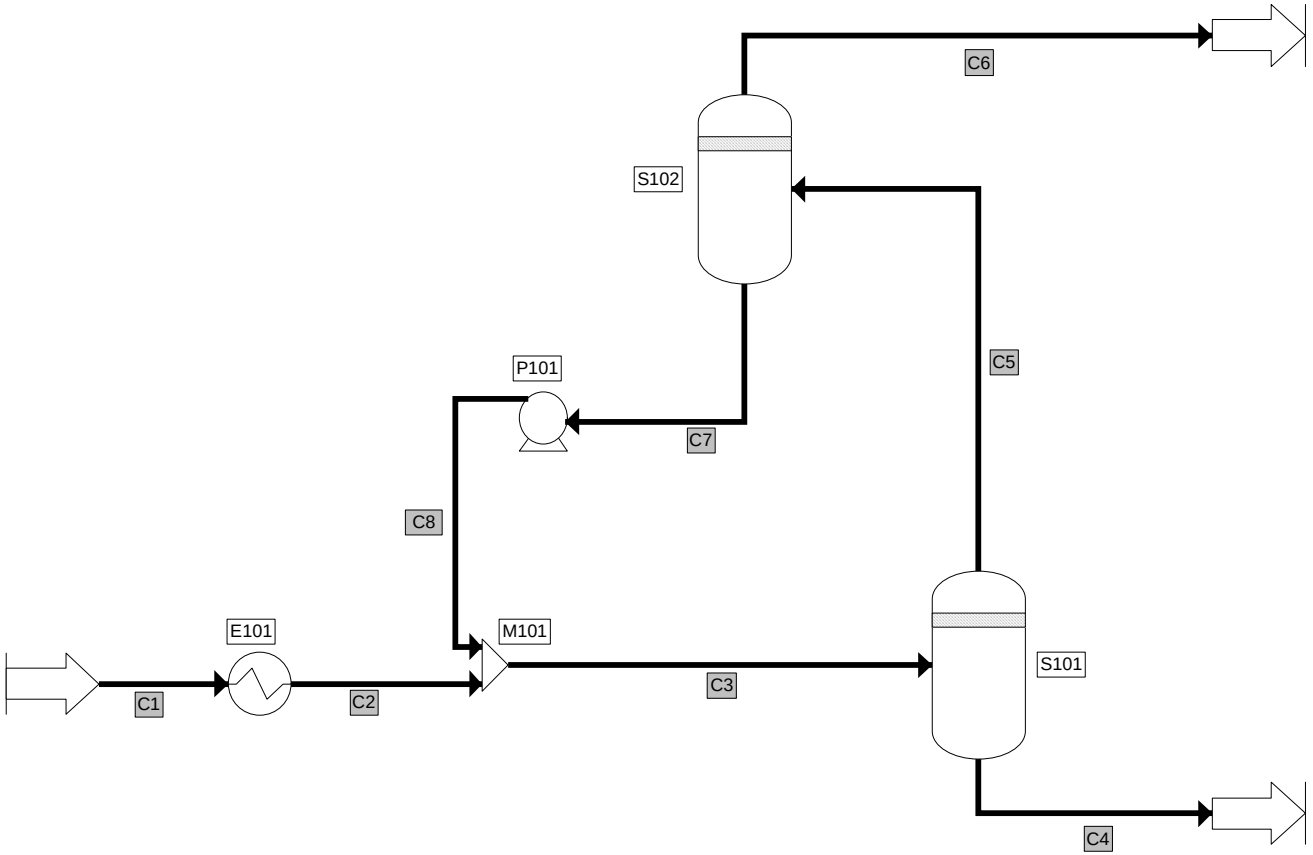
Bibliothèque	Fonction
Pywin32	Intégrer le modèle de substitution dans ProSimPlus
NumPy	Manipuler des matrices ou des tableaux multidimensionnels en Python
Pandas	Manipuler des objets (dataframe)
Matplotlib	Créer des graphiques
Scikit-learn	Bibliothèque open-source en Python pour l'apprentissage automatique
Jupyter	Créer et exécuter du code Python de manière interactive, en combinant du code, des visualisations, des textes explicatifs et des résultats
JobLib	Sauvegarder des modèles entraînés et les recharger ultérieurement

- Pour installer ces bibliothèques, ouvrir l'invite de commande et taper :

pip install Nom de bibliothèque

Introduction du procédé : Exemple Simple

Pour cette étude, l'exemple est fondé sur un procédé simple disponible dans le répertoire d'exemples de ProSimPlus sous le nom « PSPS_EX_FR-Exemple-Simple.pmp3 »

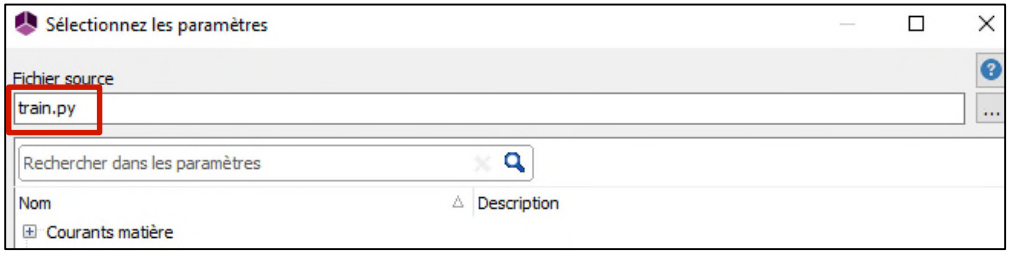


1. Création d'un jeu de données

- La première étape consiste à choisir le sujet de l'entrainement et de générer un jeu de données.

- En l'absence de données expérimentales, nous opterons pour l'utilisation de ProSimPlus avec un solveur externe pour générer les valeurs nécessaires. Une alternative serait d'utiliser l'analyse de sensibilité ou la version console, mais cela serait moins pratique.

- Le fichier source python qui va nous servir à générer le jeu de données s'appelle « train.py ».



L'explication sur les paramètres à modifier dans le fichier python est détaillée directement dans le fichier « train.py »

1. Création d’un jeu de données

- Nous avons besoin de spécifier les valeurs d’entrées (paramètres de modules) et de réaliser des simulations pour calculer les valeurs de sortie (résultats).

- Nous veillerons à éviter la création d'un jeu de données d'entrée complètement aléatoire ou excessivement linéaire, qui ne serait pas nécessairement représentatif de notre modèle.

- Pour cela, nous utiliserons la méthode des suites de Sobol, qui nous fournira aisément des valeurs quasi-aléatoires couvrant de manière appropriée les intervalles sélectionnés.

1. Création d'un jeu de données

- Le code source python « sobol.py » se trouve avec ce document.
- L'appel à la méthode des suites de Sobol s'effectue de la façon suivante :

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

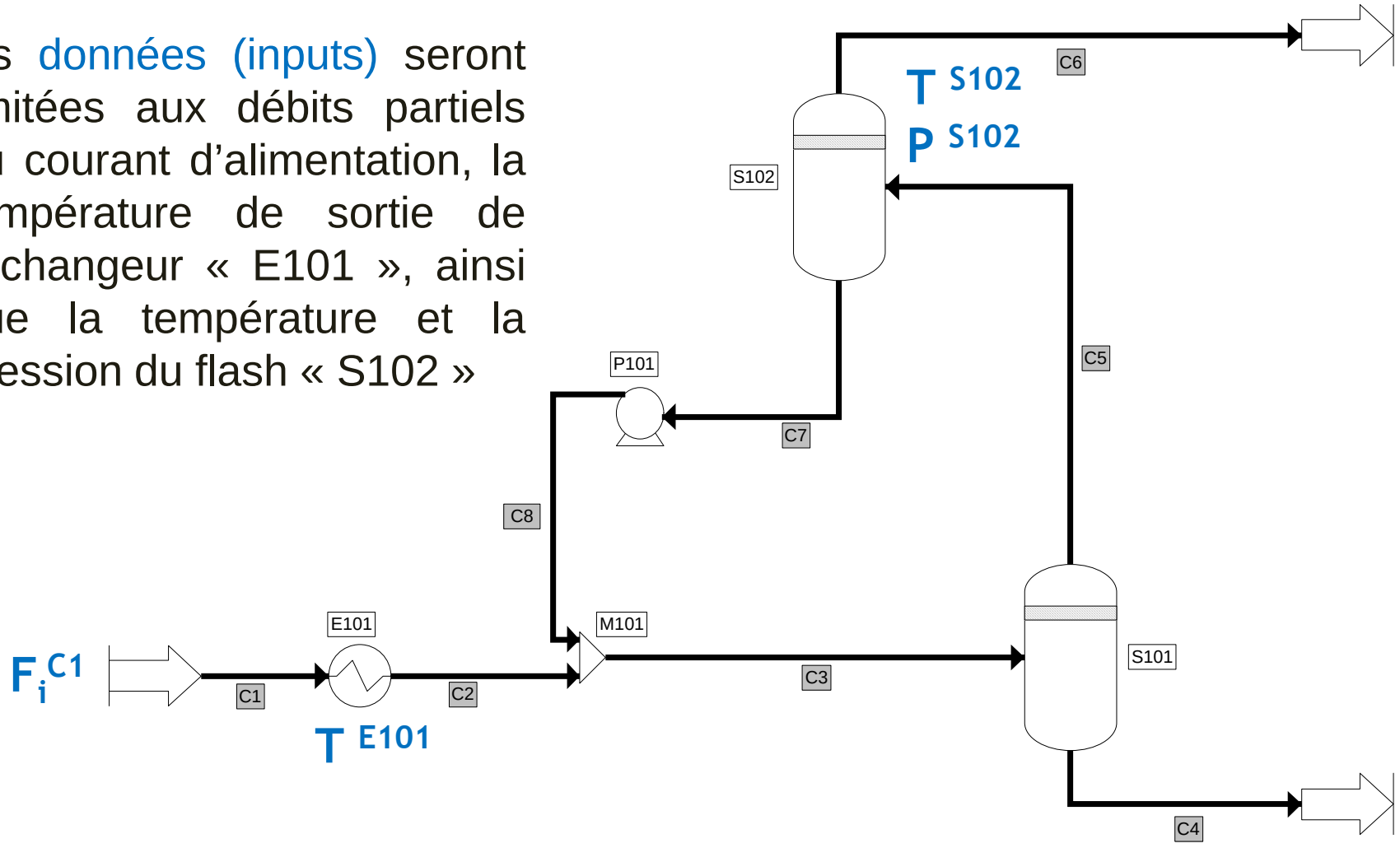
  - **ndim** est le nombre de dimensions (inputs) (de 1 à 40),
  - **npoints** est le nombre total de points,
  - **npasse** est le nombre de points de départ de la suite à passer pour éviter de tomber sur les premiers points qui sont quasiment identiques.
  - La valeur retournée **tab** est un tableau NumPy homogène de taille fixe dont les dimensions correspondent aux arguments fournis.

Il est possible d'utiliser d'autres méthodes d'échantillonnage que les suites de Sobol, mais celles-ci ne sont pas expliquées dans ce document

1. Création d'un jeu de données

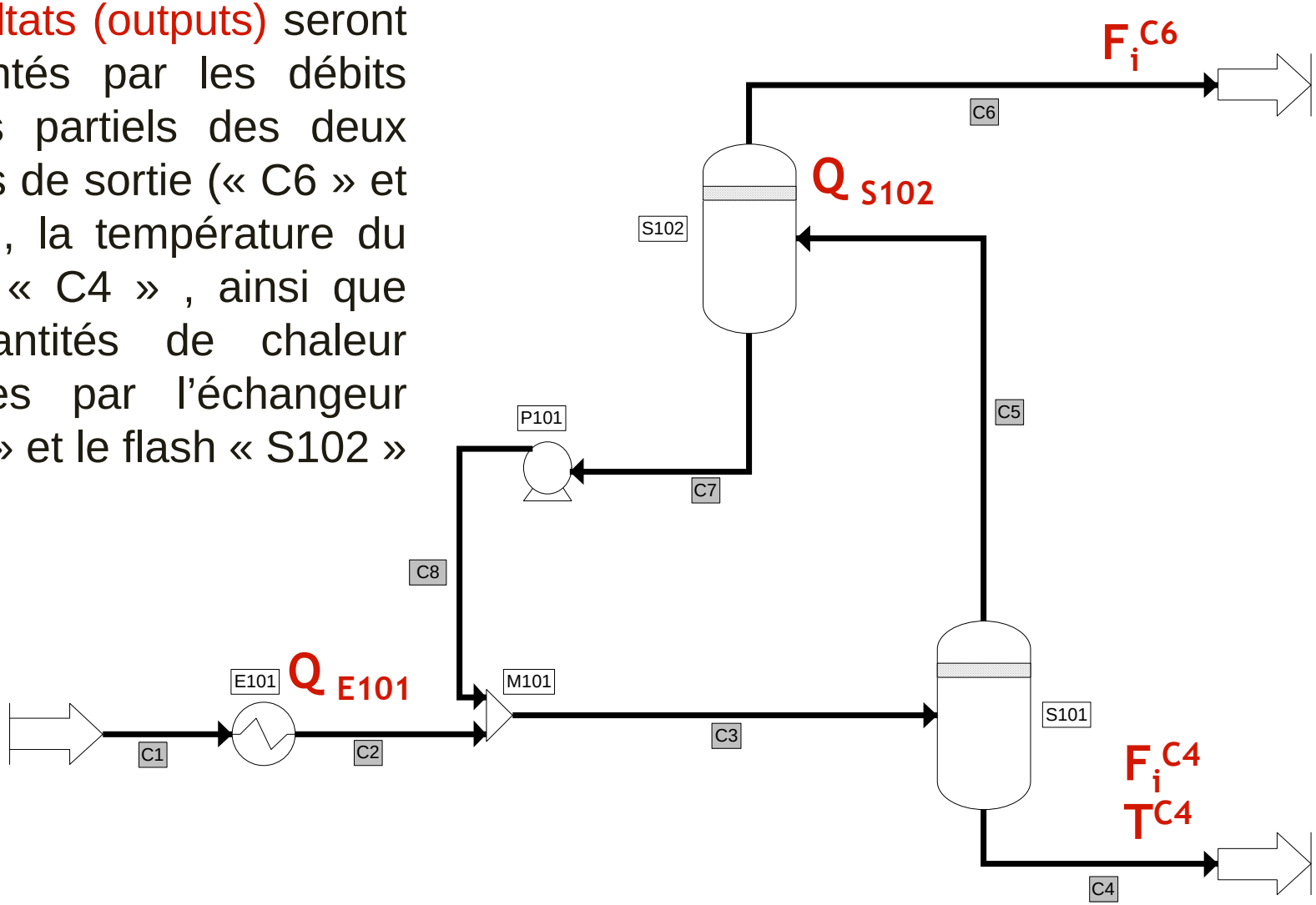
Nous allons partir du principe que :

- les données (inputs) seront limitées aux débits partiels du courant d'alimentation, la température de sortie de l'échangeur « E101 », ainsi que la température et la pression du flash « S102 »



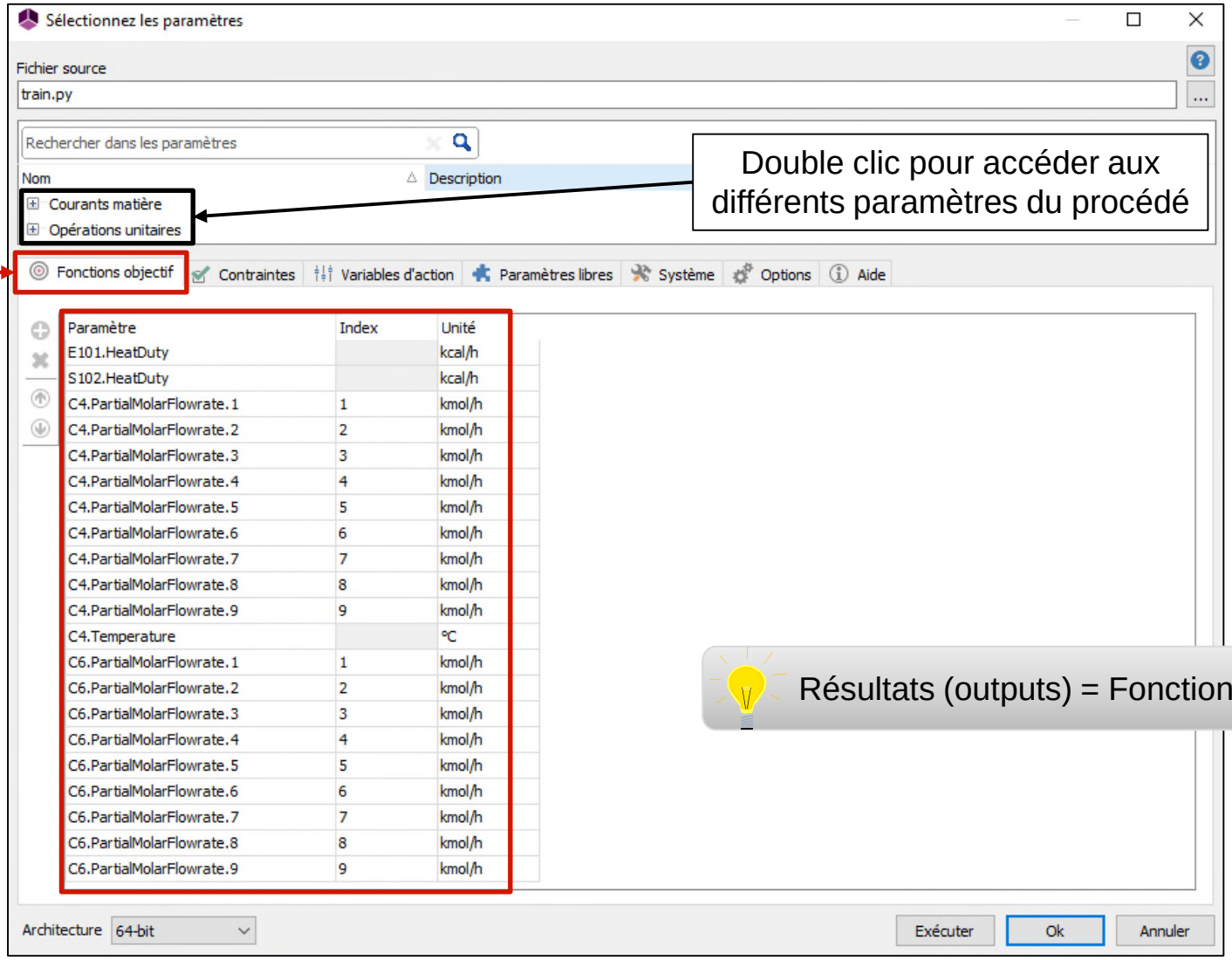
1. Création d'un jeu de données

- les résultats (outputs) seront représentés par les débits molaires partiels des deux courants de sortie (« C6 » et « C4 »), la température du courant « C4 » , ainsi que les quantités de chaleur dégagées par l'échangeur « E101 » et le flash « S102 »



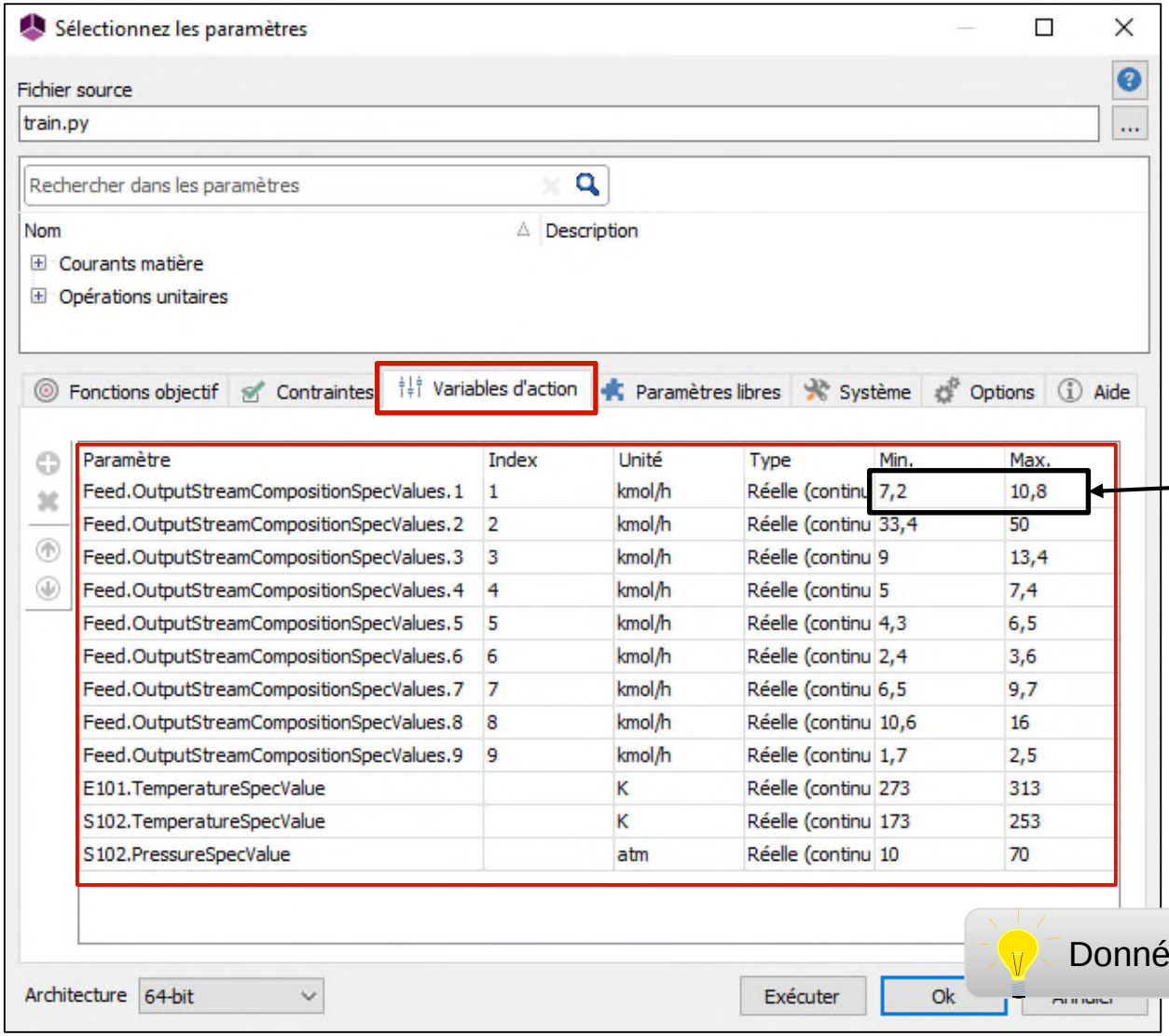
1. Création d'un jeu de données

Pour ce faire, nous allons paramétrer le solveur externe en ajoutant les résultats (outputs) dans l'onglet « fonctions objectif » :



1. Création d'un jeu de données

Et les données (inputs) dans l'onglet « variables d'action » :



Ne pas oublier de borner les variables

Données (inputs) = Variables d'action

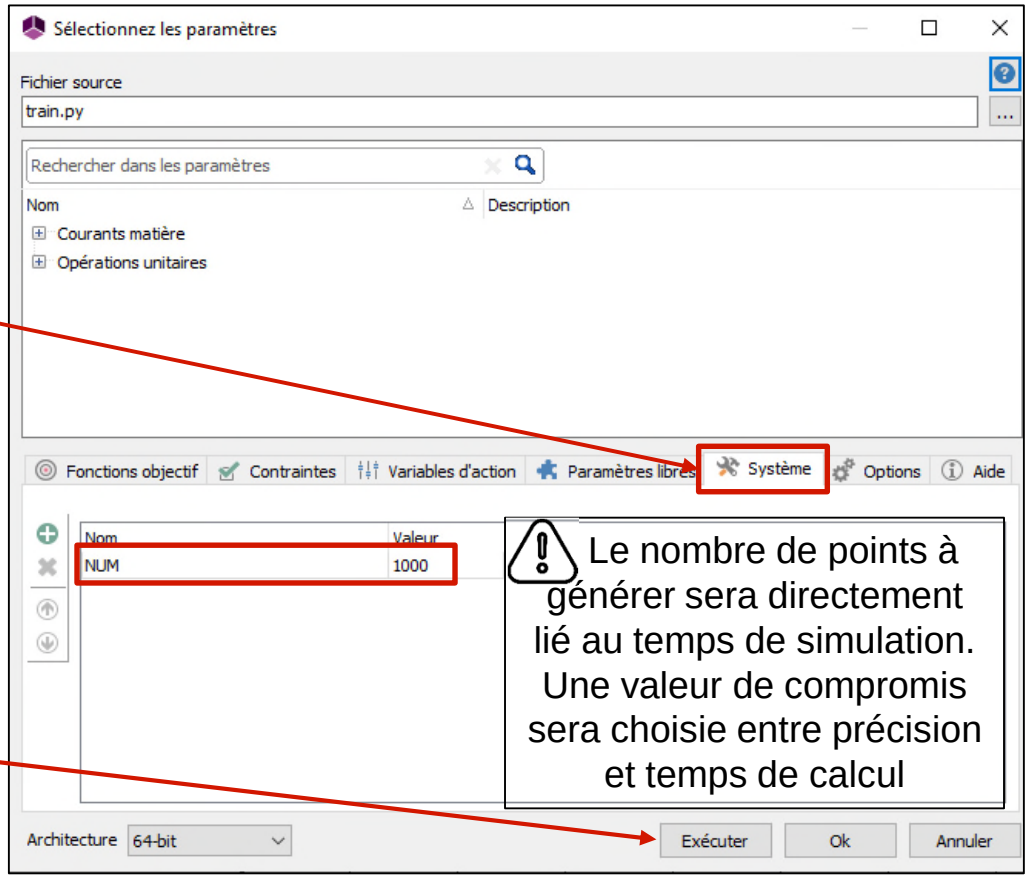
1. Création d'un jeu de données

- Nous ajoutons également un paramètre « NUM » dans l'onglet « Système » afin de pouvoir spécifier le nombre de points à générer. Ce paramètre est utilisé dans le fichier source comme le nombre total de points généré par la suite de Sobol.

> ⚠ Adapter le contenu du fichier « train.py » en accord avec les paramètres sélectionnés précédemment

- Cliquer sur « Exécuter » pour effectuer les simulations du jeu de données.

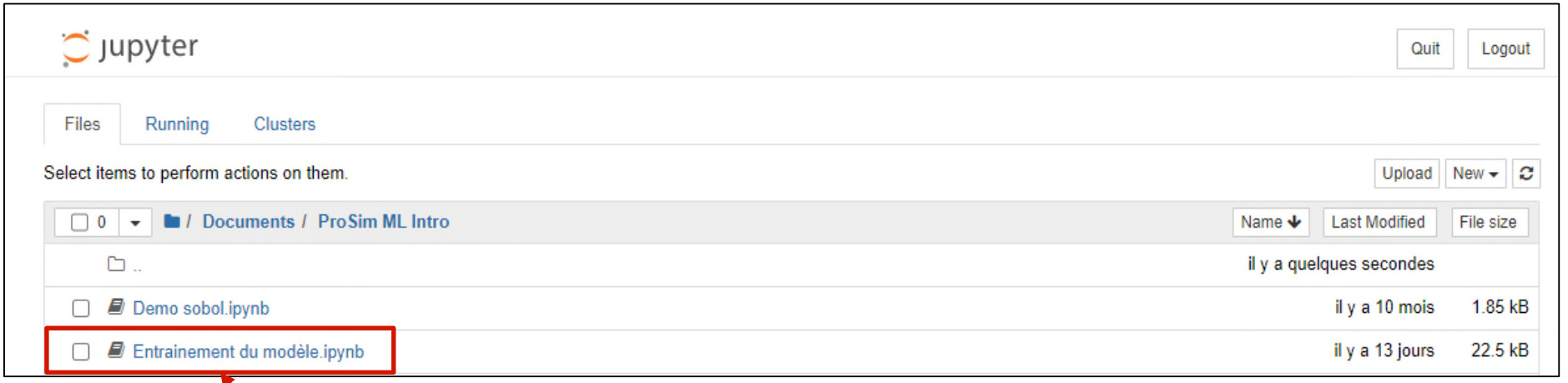
- Si tout s'est bien passé, nous avons maintenant un jeu de données disponible dans « traindata.txt ».



2. Entrainement du modèle

Cette partie sera réalisée dans Jupyter Notebook pour plus de facilité

- Ouvrir l'invite de commande et taper : Jupyter notebook
- Ouvrir le notebook intitulé « Entrainement du modèle.ipynb »



L'explication sur les paramètres à modifier pour la démarche de l'entrainement de modèle est détaillée dans ce fichier

2. Entrainement du modèle

1. Vérification, nettoyage des données

Le nettoyage des données est une étape essentielle avant de créer un modèle d'apprentissage. Cela implique de prendre des mesures pour traiter et corriger les données brutes, afin que le modèle puisse apprendre de manière efficace et produire des résultats de haute qualité.
Les principales étapes du processus sont :

- **Collecte des données**

- **Traitement des données** (données mal annotées, présence de valeurs NaN …)

- **Normalisation ou mise à l'échelle**

Si les différentes données (inputs) ont des échelles très différentes, il est conseillé de les normaliser ou de les mettre à l'échelle pour éviter que certains inputs dominent excessivement le modèle.

- **Séparation des données**

Diviser les données en ensembles d'entraînement (**training set**) et de test (**test set**) pour évaluer les performances du modèle de manière impartiale. Il est possible de choisir le pourcentage associé à chaque ensemble.

2. Entrainement du modèle

2. Apprentissage du modèle

Diverses méthodes d'apprentissage sont disponibles, et dans cette étude, nous utilisons un **modèle de régression KernelRidge (***Kernel Ridge Regression***),** une régression régularisée qui utilise les méthodes à noyau pour effectuer des prédictions. D'autres modèles d'apprentissage pourraient également être utilisés, comme par exemple :

- kNN (k plus proches voisins)
- DecisionTree
- RandomForest
- MLP (Multi-Layer Perceptron) - Réseaux de Neurones Artificiels

Il n'existe pas de solution unique pour déterminer le meilleur algorithme de régression adapté à tous les scénarios. La démarche la plus recommandée consiste à expérimenter plusieurs modèles, les évaluer en termes de performances, et sélectionner celui qui obtient les meilleurs résultats sur le jeu de données disponible.

2. Entrainement du modèle

3. Paramètres d'évaluation

Voici quelques métriques couramment utilisées pour évaluer les performances des modèles d'apprentissage, en particulier dans le cas de la régression :

- **Coefficient de détermination (R2 score)**

Un R2 score élevé (≈1) indique que le modèle a une bonne capacité à expliquer la variation des données, tandis qu'un R2 score proche de 0 ou négatif suggère que le modèle ne parvient pas à bien représenter les données.

- **Erreur absolue moyenne (*Mean Absolute Error* - MAE)**

La MAE mesure l'écart moyen des erreurs de prédiction du modèle.

- **Erreur quadratique moyenne (*Mean Squared Error* - MSE)**

La MSE mesure la moyenne des carrés des différences entre les valeurs prédites et les valeurs réelles. Une MSE plus faible indique que les prédictions du modèle sont plus proches des valeurs réelles.

Les métriques obtenues doivent être du même ordre de grandeur pour les ensembles d'entrainement et de test.

2. Entrainement du modèle

4. Sauvegarde joblib

- Une fois que nous avons entraîné notre modèle, nous pouvons utiliser Joblib, qui permet de sauvegarder des structures de données dans un fichier afin de pouvoir les recharger par la suite (dans ProSimPlus par exemple).

```
# Pour la sauvegarde, nous utilisons joblib mais il y a d'autres solutions
import joblib
joblib.dump(bestmodel, "model.joblib")
```

- Nous enregistrons également les valeurs minimales et maximales des données (entrées) ainsi que des résultats (sorties). Comme notre modèle est normalisé, nous avons besoin de ces valeurs minimales et maximales pour pouvoir transformer les résultats en valeurs réelles.

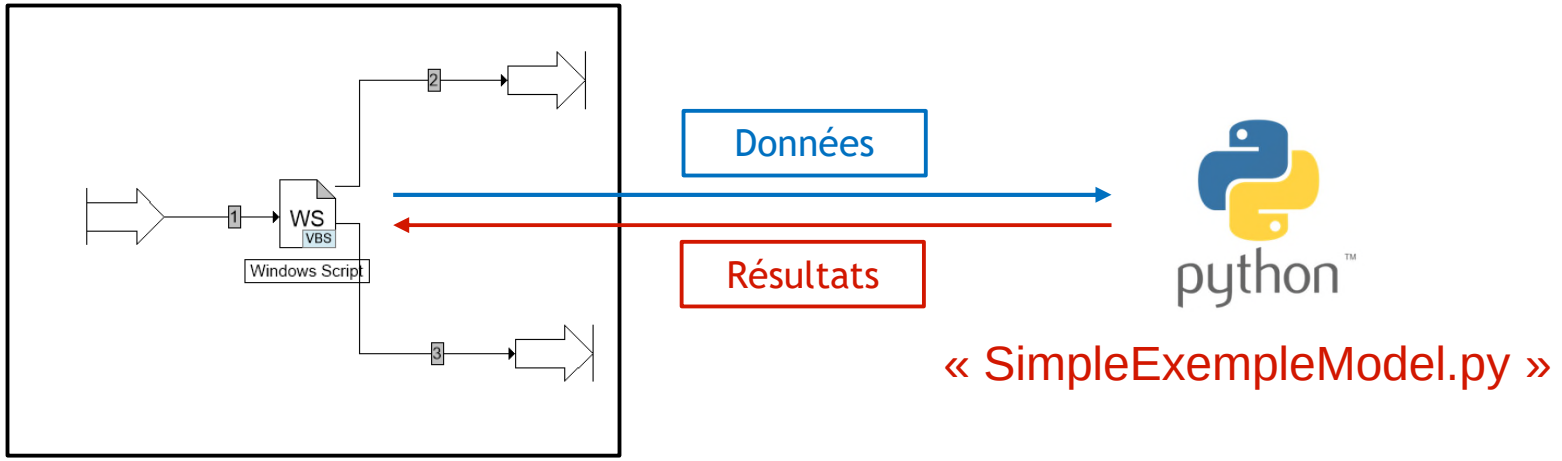
```
# Sauvegarder le dictionnaire à l'aide de joblib
joblib.dump(minmax, "parameters.joblib")
```

3. Utilisation dans ProSimPlus

Après la phase d'entrainement du modèle sur Jupyter notebook, nous avons généré un modèle qui est enregistré dans le fichier « model.joblib », il nous faut maintenant mettre en place la structure pour l'utiliser dans un procédé sous ProSimPlus.

Cette structure est relativement simple, elle va être constituée par :

- Une opération unitaire de type Windows Script
- Un programme Python qui va faire la prédiction en passant les données et retourner les résultats



« SimpleExempleModel.py »

3. Utilisation dans ProSimPlus

Nous allons donc créer un composant en Python au standard Microsoft COM que nous utiliserons depuis le code VBScript. L'avantage principal de cette méthode est que le programme Python pourra rester en mémoire pendant toute la durée de la simulation.

Le code source de ce composant se trouve dans le fichier « SimpleExempleModel.py ».

La description du code est faite directement sous forme de commentaires.

3. Utilisation dans ProSimPlus

Voici quelques explications sur les codes :

- **_reg_progid_**

Le ProgID (Programmatic Identifier) est une chaîne de caractères utilisés pour identifier une classe d'objet COM (Component Object Model) pouvant être instanciée et utilisée dans un script. Il est obligatoire et servira lors de la création d'une instance en VBScript (CreateObject("Mon.progid")).

```
_reg_progid_ = "ProSim.SimpleExempleModel"
```

- **_reg_clsid_**

Le GUID (Globally Unique Identifier) est un identifiant unique utilisé pour référencer la classe.

```
#   https://guidgenerator.com/online-guid-generator.aspx par exemple. Et n'oubliez pas les accolades !
_reg_clsid_ = "{16b4ef0e-ed04-481f-8e31-99e6e17fb0ad}"
```

- **_public_methods_**

Il s'agit ici d'un tableau de chaînes de caractères définissant les méthodes (fonctions ou routines) qui seront disponibles sur notre classe. Pour notre exemple, nous allons proposer juste deux méthodes : une pour calculer (prédire), l'autre pour imprimer des résultats dans le rapport.

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

3. Utilisation dans ProSimPlus

- **Utilisation de « model.joblib »**

La méthode « Calculate » va nous servir à prédire le résultat.

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

Pour cela, le processus consiste à récupérer les données d'entrée (inputs) et prédire les résultats (ouputs) à travers un simple appel à la méthode *'predict'*. L'utilisation de cette méthode '*predict*' devient réalisable après avoir chargé le modèle qui a été préalablement enregistré à l'aide du fichier joblib.

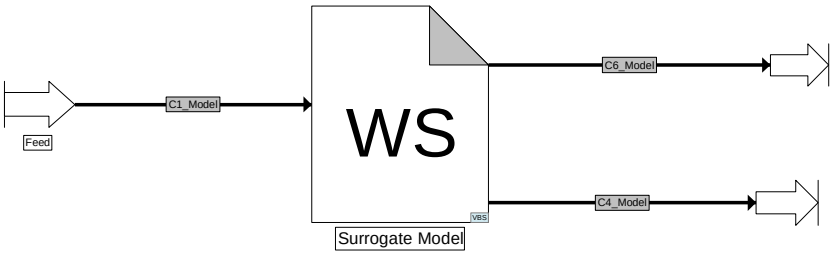
```
features = pandas.DataFrame(data, columns=["F1","F2","F3","F4","F5","F6","F7","F8","F9","T_Ech","T_S102","P_S102"])

res = model.predict(features)
```

Le résultat est retourné sous la forme d'un tableau à deux dimensions :

```
return res
```

3. Utilisation dans ProSimPlus

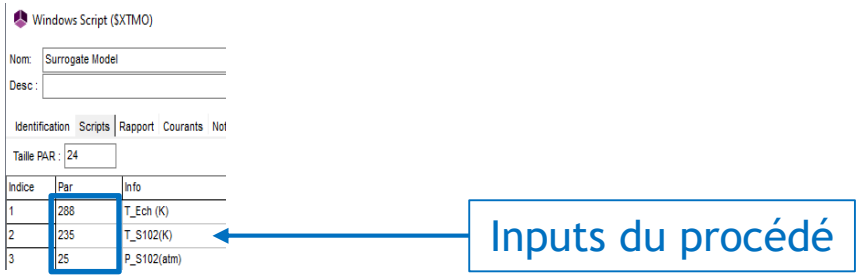
Un fichier de test ProSimPlus faisant usage de ce composant est disponible également sous le nom « SimpleExample.pmp3 ».



Extrait du script :

```
dim SimpleExemple

Sub OnSimulationStart()
  set SimpleExemple = createobject("ProSim.SimpleExempleModel")
End Sub
```

Il est nécessaire de fournir les autres paramètres (inputs) du procédé :

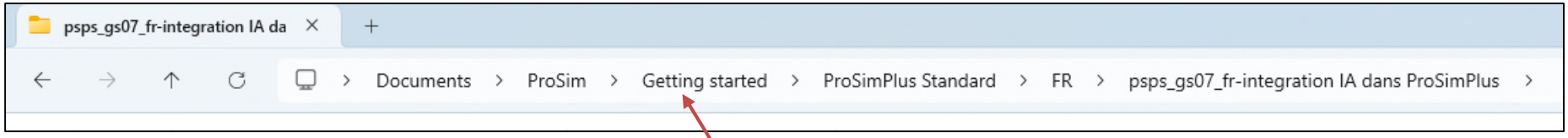


En résumé, le script réalise les actions suivantes : il crée une instance pour récupérer le ProgID de la classe, récupère les propriétés du courant d'entrée, prend en compte les paramètres (inputs) de procédé, génère les courants de sortie et récupère les résultats renvoyés par le modèle d'entrainement.

3. Utilisation dans ProSimPlus

La dernière étape consiste à simuler le procédé à travers le surrogate model que nous avons construit. Pour cela :

- Ouvrir le répertoire du fichier :



Cliquer dans la barre d'adresse et taper : cmd

- La fenêtre de l'invite de commande s'ouvre.
  Exécuter le programme avec la commande :

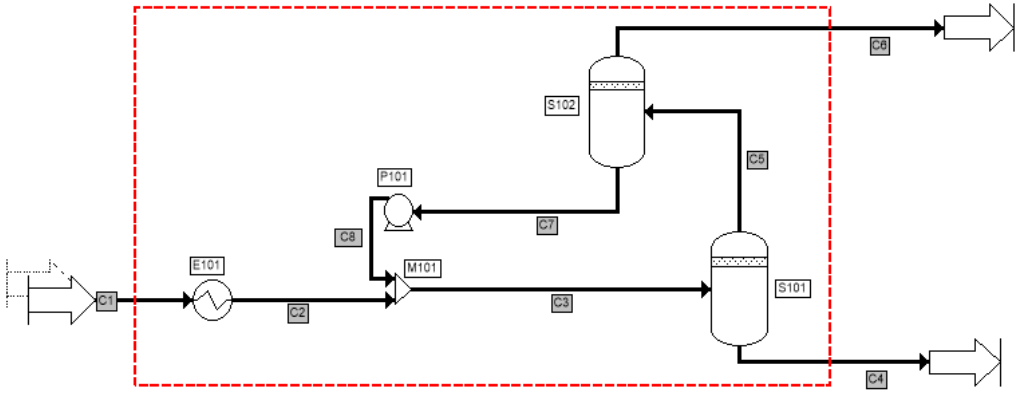
  python SimpleExempleModel.py --register

Cette étape n'est à effectuer qu'une seule fois, avant la première utilisation.
L'explication de cette étape est détaillée dans le fichier python : « SimpleExempleModel.py »

- Lancer la simulation complète.

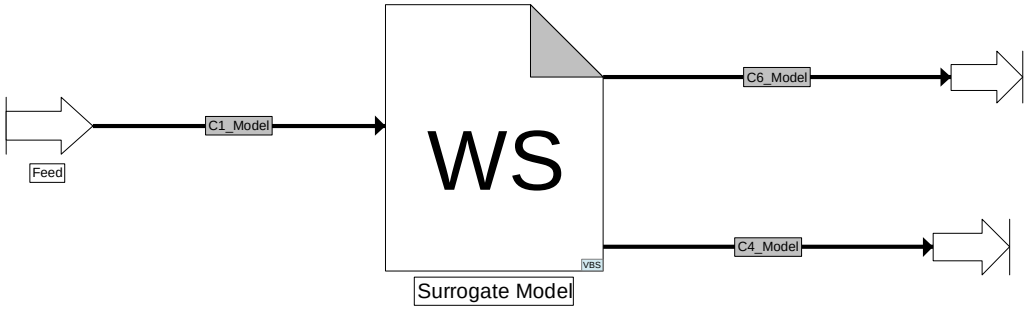
3. Utilisation dans ProSimPlus

Exemple de résultat :



Simulation sur ProSimPlus

Courants		C1	C4	C4_Model	C6	C6_Model
De		Stream dupli...	S101	Surrogate M...	S102	Surrogate M...
Vers		E101	Process out...	Process out...	Process out...	Process out...
Débits partiels (molaires)		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
Débit total (massique)	kg/h	4332.2	3575.5	3573.6	756.65	758.52
Etat physique		Liq./Vap.	Liquide	Liquide	Vapeur	Liq./Vap.
Température	°C	40	14.561	15.598	-38	-38.15
Pression	atm	75	74.7	75.994	25	25
Fraction vaporisée molaire		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