Evaluation of Process Flowsheets via Python-ASPEN Integration

Project Scope

Performing manual simulations on ASPEN Plus is often time consuming for complex tasks such as conducting multi-variable sensitivity analyses and extracting stream data from a complicated process flowsheet. The versatility of Python as a data management tool has made it an ideal candidate for automating these tasks. Therefore, this project aims to investigate the procedure of linking Python to ASPEN Plus simulations and explore its uses in evaluating process flowsheets.

Connecting Python to ASPEN

The ASPEN model used is ***ASPEN Plus V12.1*** and ***Anaconda*** is selected as the default Python interpreter on VSC. The following is applicable to Windows operating systems.

To connect a Python script to an ASPEN file, the following libraries must be imported first.

```
#Import libraries
import os                          # Import operating system interface
import win32com.client as win32    # Import COM
```

These are the fundamental libraries that enable Python to communicate with external Windows applications like ASPEN Plus via the Component Object Model (COM) interface. The first library ***os*** allows the script to interact with the operating system for tasks such as navigating directories and handling file paths. The second library ***win32*** is essential to control ASPEN via COM automation for tasks like changing input values, running simulations and extracting results.

After importing the necessary libraries, the next step is to write the path to the desired ASPEN file and launch the simulation. Simply replace the variable under ***File*** with the desired file path in the following lines of code.

```
#Specify file name/path
#Modify file path accordingly
File = r" # file path "
aspen_Path = os.path.abspath(file)

#Launch ASPEN Plus and load the simulation
aspen = win32.Dispatch('Apwn.Document') #Launch ASPEN Plus
aspen.InitFromArchive2(aspen_Path) #Load simulation
```

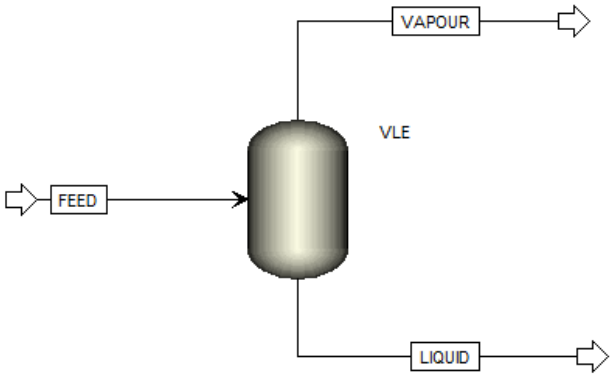
To verify if the linking process is successful, include the following line. Run the Python script and the ASPEN simulation should appear automatically on the screen.

```
# Make Aspen Plus visible (optional, can run hidden)
aspen.Visible = True
```

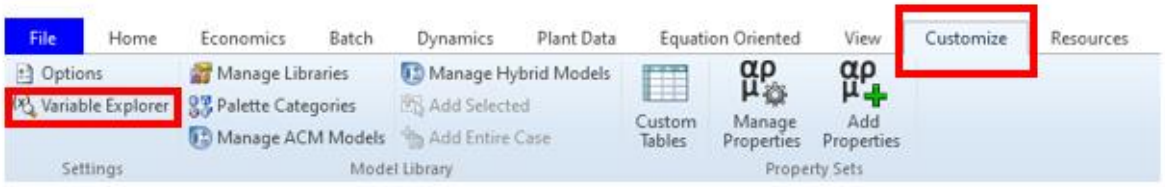
Extracting Output Data

Simulation data on ASPEN is stored in a hierarchal tree structure which can be visualised using the ***Variable Explorer*** feature. By obtaining the pathway to a desired variable, Python can navigate the tree to extract the data relevant to the variable. This will be illustrated with the following example. The complete code can be found in ***VLE.py***.

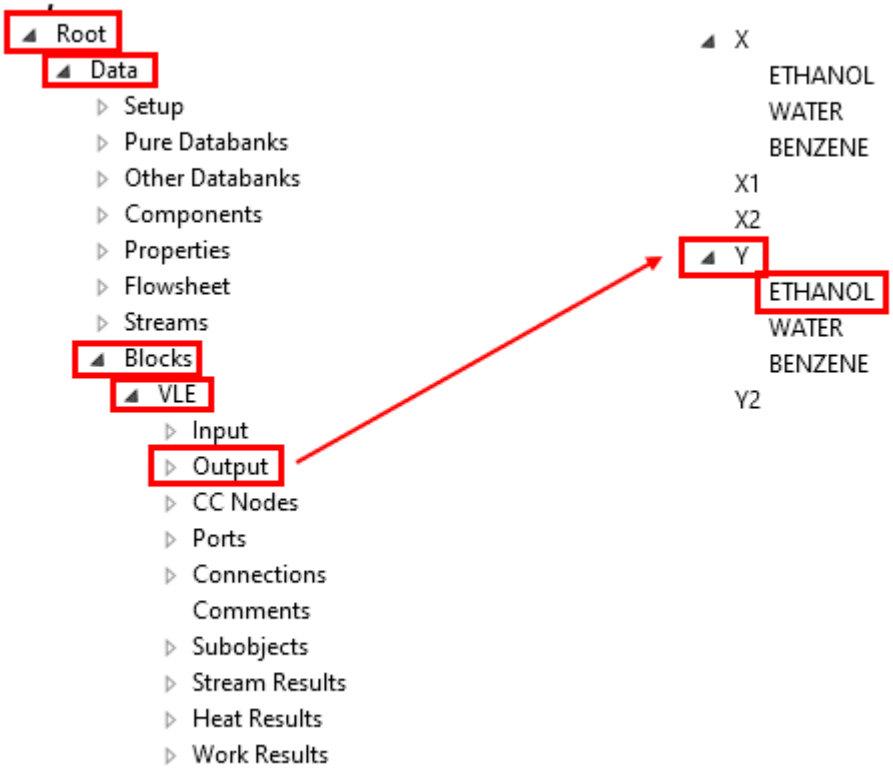
Example: Extracting VLE data from a flash separator simulation



In the above example of a simple flash separator consisting of ethanol, benzene and water components, suppose that we are interested in extracting the output vapour and liquid compositions. The VLE data is stored under nodes in the data tree which we need to obtain the pathway to these nodes to enable Python to locate these variables. Navigate to the ***Customize*** section and click on ***Variable Explorer***.



In the ***Variable Explorer*** interface, expand the tree until the ***Blocks*** branch appears. This is where all blocks in the simulation and its data are stored. Select the relevant block and navigate to the ***Output*** branch to obtain a collection of output data for this block.



The liquid and vapour compositions for this instance are stored under the ***X*** and ***Y*** sub-branch under the ***Output*** branch. To illustrate this example further, we would look at the vapour composition of the ethanol component specifically. By selecting ***ETHANOL*** under ***Y***, a table of attributes for the selected variable will appear on the right side of the interface as shown.

Attribute	Value
Path to Node	Application.Tree.Data.Blocks.VLE.Output.Y.ETHANOL
Call	Application.Tree.FindNode("\Data\Blocks\VLE\Output\Y\ETHANOL")
Dimension	0
Value	0.598615414
Physical Quantity	0
Unit of Measure	0
Basis	MOLE
Record Type	No Attribute
Output	1
Enterable	
Upper Limit	No Attribute
Lower Limit	No Attribute
Default Value	No Attribute
Completion Status	
Port In or Out	No Attribute
Port Gender	No Attribute
Multiport	No Attribute
Port Type	No Attribute
Has Children	0

As seen above, the value of the variable is stored as 0.5986... under ***Value*** in the 4th row of the attribute table which will be extracted by Python. ***Path to Node*** shows the pathway to the variable in the data tree while ***Call*** formats the pathway to be read by external programs. Extract ***Application.Tree.FindNode("\Data\Blocks\VLE\Output\Y\ETHANOL")*** and modify it to suit the Python script's format. Since we have previously defined ***aspen*** as the default program function variable, replace ***Application*** with ***aspen*** and add a ***.Value*** function at the end of the line to instruct Python to extract the value of the variable.

```
y_ETOH = aspen.Tree.FindNode(r"\Data\Blocks\VLE\Output\Y\ETHANOL").Value
```

After learning how to extract the value for a single variable, the above process can be repeated to extract data for other variables such as stream enthalpy.

Manipulating Input Variables

Apart from extracting output data, Python is also capable of manipulating input data of an ASPEN simulation. The following illustrates how the value of a single variable on ASPEN can be modified to obtain new simulation results. The complete code can be found in ***Loop Test.py***.

From the previous flash separator example, this time we are interested in observing how varying the column's pressure affects the resultant VLE. Using the same logic from extracting output data, we can obtain the pathway that writes the column's pressure, change its value and instruct ASPEN to re-run the simulation with the newly defined pressure.

```
#Reinitialise simulation
aspen.Engine.Reinit()

#Pressure
aspen.Tree.FindNode(r"\Data\Blocks\VLE\Input\PRES").Value = 1 #atm

#Re-run the simulation
aspen.Engine.Run2()
```

In the code above, the simulation must be reinitialised first to remove output data from former runs. Next, obtain the pathway to the column's input pressure in ***Variable Explorer*** and define a new pressure of 1 atm for instance. The last line instructs ASPEN to run the simulation again at 1 atm. The new output VLE can be returned using the techniques in the previous section.

Iterating Over a Range of Input Variables

The previous section provides insight into the fundamentals of manipulating input data. However, it is practically infeasible to obtain simulation results for a range of pressures by keying in a single pressure value for each run. Therefore, we can utilise iteration techniques

on Python to generate VLE data for a range of pressures. This is identical to the sensitivity analysis feature available on ASPEN which allows the user to define a range of values for a certain input variable.

The code below depicts how the vapour composition of the three components can be obtained for a range of pressures. In this case, the range of pressures is defined from 0.1 atm to 1 atm with 10 data points. Three empty arrays are also created to store the resulting vapour compositions for the three components. A ***for*** loop is constructed to iterate the aforementioned process for the range of defined pressures. After running the code, the results can be exported for visualisation using libraries such as ***matplotlib***.

```
 #Input data
#Pressure
P_range = np.linspace(0.1, 1, 10) #atm

#Results array
y_ETOH_array = []
y_H2O_array = []
y_BENZENE_array = []

for P in P_range:
    aspen.Engine.Reinit() #Reinitialise simulation
    aspen.Tree.FindNode(r"\Data\Blocks\VLE\Input\PRES").Value = P
    aspen.Engine.Run2() #run simulation

    #Read compositions
    y_ETOH=aspen.Tree.FindNode(r"\Data\Blocks\VLE\Output\Y\ETHANOL").Value
    y_H2O=aspen.Tree.FindNode(r"\Data\Blocks\VLE\Output\Y\WATER").Value
    y_BENZENE=aspen.Tree.FindNode(r"\Data\Blocks\VLE\Output\Y\BENZENE").Value

    #Store output data in results array
    y_ETOH_array.append(y_ETOH)
    y_H2O_array.append(y_H2O)
    y_BENZENE_array.append(y_BENZENE)
```

Advantages of Python Iteration Compared to Sensitivity Analysis

A major limitation of the sensitivity analysis tool in ASPEN is its infeasibility for large scale studies. When conducting a sensitivity analysis for a given variable, other variables have to be fixed. This makes the simulation task manually intensive as numerous runs have to be conducted for multi-variable studies. If we were to repeat the above flash example with 20 data points of temperature and pressure each, this will require 20 independent runs using ASPEN's sensitivity analysis tool. This is overcome with iteration techniques on Python where

a single run is sufficient through embedding the ***for*** loop for a given variable within the ***for*** loop of another variable.

Application in Stream Data Extraction

This section demonstrates how the techniques above can be applied in stream data extraction for a chemical looping hydrogen production process. The objective is to extract the inlet and outlet temperatures as well as the duties of all the utility heat exchangers in the simulation. The complete code can be found in ***HX_H2.py***.

In the Python code, stream extraction is performed via a series of tasks. First, the code identifies and lists out all the heat exchanger blocks in the simulation. For this task, the ***.Elements*** function is used to access all the available blocks in the simulation. The names of the blocks are then passed through a statement which segregates out heat exchanger blocks if their names begin with the letter E (Exchanger), K (Kettle Reboiler) or F (Furnace). It is acknowledged that this method is not universal as it relies on the naming system and has to be modified according to the simulation. A better approach is to extract the blocks based on their types (reactor, separator, exchanger) but there is currently no recognisable Python function available to obtain block type. After identifying all the heat exchanger blocks, the next step is to extract the heat exchanger duties and the temperatures of the connecting streams. This is easily done using the ***.Value*** function for extracting the values of data. Afterwards, the extracted temperatures are passed through another statement to separate the streams into hot and cold streams. Lastly, the arrays containing extracted data are exported to Excel using the ***pandas*** library.

Another feature has been included in the code to pair heat exchangers based on matching duties. The code sorts the heat exchangers into heaters and coolers and pair them up if they have identical duties. Following that, it finds the temperature difference at the cold and hot end of the process-to-process heat exchanger. The extracted data is useful for heat integration processes and the design of an optimal MER HEN.