Multiobjective Optimization of Chemical Processes with Complete Models using MATLAB and Aspen Plus

Abel Briones-Ramírez[1], Claudia Gutiérrez-Antonio[2]

[1] Exxerpro Solutions, Querétaro, Mexico

[2] Universidad Autónoma de Querétaro, Facultad de Química, Querétaro, Mexico

abel.briones@exxerpro.com, claudia.gutierrez@uaq.mx

Abstract. The design of chemical processes has to consider the delivery of products with high quality, minimum energy requirements and minimum impact to the environment. In order to design chemical processes with the minimum energy requirements several researchers have been focus their efforts in the development of optimization strategies. Regardless the optimization technique used, it is necessary a model of the chemical process. Some works consider the use of reduced models, which are more simple and easy to implement. Another studies employ complete models, usually linking the optimization strategies to chemical processes simulators through Excel®; however, the information to make this link directly between the chemical process simulator and the optimization strategy neither is nor reported yet. Thus, in this work we propose a procedure to perform the link between MATLAB® and Aspen Plus® processes simulator. We present the information requirements, information flows and the communication commands. Also, instructions to generate automatically the bkp files of the optimal designs are described. It is worth to mention that with this procedure any optimization strategy can be used.

Keywords. MATLAB®, Aspen Plus®, linking procedure.

1 Introduction

Nowadays, the design of chemical processes must to consider not just delivering products of high quality, but also performing this task with minimum both energy requirements and environmental impact. According to Yue et al. [30], "concerns about climate change, waste pollution, energy security, and resource depletion are driving society to explore a more sustainable way for development and manufacturing" [30]. In order to design chemical processes with the minimum energy requirements several researches have been focus their efforts in the development of optimization strategies; these strategies include mathematical programming [1, 26, 19, 5, 22, 9, 31] or stochastic techniques [18, 20, 28, 24, 32, 23]. Regardless the optimization technique used, it is necessary a model of the chemical process. Some works employ reduced models, which are more simple and easy to implement; another strategies consider complete models, usually linking the optimization strategies to chemical processes simulators. Next, we give a brief review of some works that consider the last approach, for different purposes.

Lababidi et al. [17] developed a prototype design support system for process engineering, in order to analyze the controllability and dynamics of process during the conceptual design stage. They integrated MATLAB® with Omola®, through object-oriented message passing to get this objective.

Later, Ramzan & Witt [25] proposed a methodology for decision support among conflicting objectives; for this, they used a multiobjective optimization layer based on goal programming using Aspen Plus® and the optimization tool for MATLAB®.

In 2008, Tona Vásquez et al. [27] presented an approach to realize multiscale modeling for the production of perfume microcapsules.

Their strategy connects modules developed in MATLAB® with the operation models developed in

Computación y Sistemas, Vol. 22, No. 4, 2018, pp. 1157–1170
ISSN 1405-5546
doi: 10.13053/CyS-22-4-3087

Aspen Plus®. Gutiérrez-Antonio and Briones-Ramírez [12] proposed the use of a multiobjective genetic algorithm with constraints handling for the optimization of Petlyuk sequences. The optimization tool, developed in MATLAB®, is coupled to Aspen Plus® in order to get the information of the objectives and constraints required. In 2011, Khodadoost and Sadegui [15] reported a dynamic simulation study of distillation column sequences in a Gas Refinery. They linked the module Dynamics of Aspen Plus® and MATLAB® Simulink® software to perform their work. In the same year, Eslick and Miller [8] reported a work where the freshwater consumption was minimized; the study case was a pulverized coal power plant with a capacity of 550 MW. Their framework uses an optimization scheduler, based on NSGA-II [7], which is linked to Aspen Plus® by means of Excel®. In addition, Alabdulkarem et al. [2] optimized the propane pre-cooled mixed refrigerant of a liquefied natural gas plant. In order to realize the optimization, they used a genetic algorithm, taken from the MATLAB® optimization toolbox, along with a computer model in Aspen HYSYS®. Bhattacharyya et al. [3] made a study of the control performance during load, following in a coal-fed integrated gasification combined cycle power plant. They developed a steady-state model, which was exported to the module Dynamics of Aspen Plus®, and integrated with MATLAB®. In the same year, Kiss et al. [16] proposed a novel biodiesel process based on a reactive dividing wall column. As a design tool, they used simulated annealing as optimization technique, which was coupled to Aspen Plus® through Excel®.

From the previous works, it is clear that the linkage of a processes simulator is very useful; since it allows considering the complete model of the chemical processes, taking advantage of the complete thermodynamic properties data base of the processes simulator. Nevertheless, none of the above works have presented detailed information on how to make the link of MATLAB® with the process simulator directly, Aspen Plus®. The availability of this procedure allows the use of complete models for the chemical processes, taking advantage of all the modeling capabilities of the process simulators, no matter the kind of optimization technique used.

Thereby, in this work we propose a procedure to perform the link between MATLAB® and Aspen Plus® processes simulator. We present the information requirements, information flows and the communication commands. Also, instructions to generate automatically the bkp files of the optimal designs are presented.

2 Selection of the Chemical Process

In this section, we present the first step required to perform the optimization rigorously, which is the selection of the chemical process. For the selected chemical process, we have to choose the objectives to optimize, and the constraints involved, if they are any. The objectives and/or constraints are the information that guides the optimization process, and also it is the information that is generated in the simulation of the chemical process in Aspen Plus®.

In addition, we must to list the process variables required for the simulation; with this information objectives and/or constraints can be calculated in the simulation. These variables are manipulated in the optimization strategy. It is desirable to record variables, objectives and constraints as vectors in a database; this will allow having all the required information of the optimal solution.

Thus, the problem statement can be established as:

$$\begin{aligned} &\text{Optimize } (\textit{Objective\_1, Objective\_2,…, Objective\_i}) \\ &\text{Subject to} \\ &\textit{Constraint\_1}\geq a\ ,\ \textit{Constraint\_2}\leq b,\ \dots\ ,\ \textit{Constraint\_j}=c \\ &\text{Manipulating} \\ &\textit{Variable\_1, Variable\_2,…,Variable\_k} \end{aligned} \tag{1}$$

3 Procedure to Link MATLAB® and Aspen Plus®

In this section, the linking procedure is presented. In order to illustrate this procedure, we will use a multiobjective genetic algorithm with constraints' handling written in MATLAB® [12]; however, the procedure can be used with any other optimization technique, considering or not constraints.

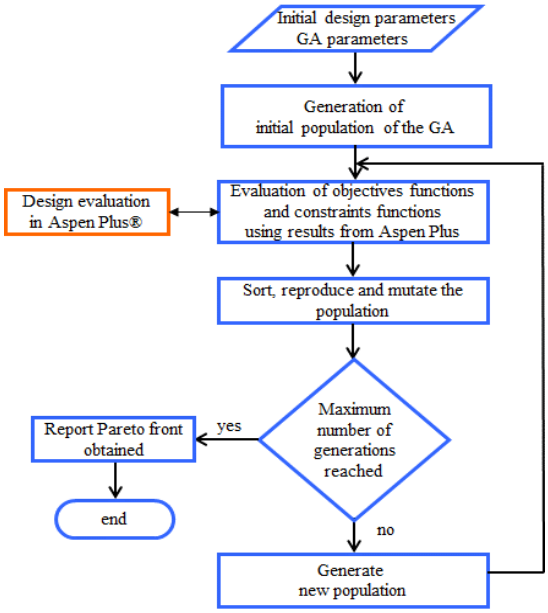

Fig. 1. General flowsheet of the linking procedure between MATLAB® and Aspen Plus®

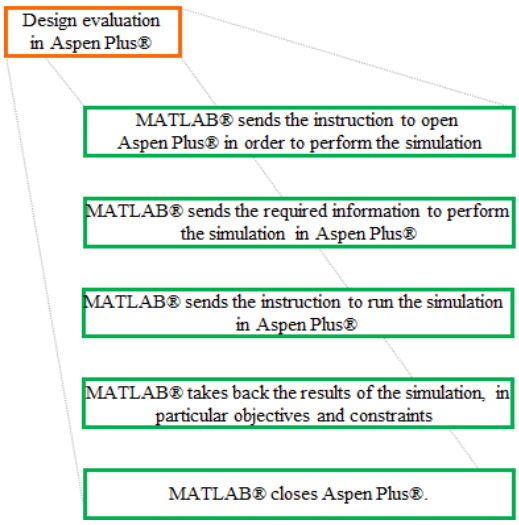

Fig. 2. Design evaluation in Aspen Plus® block

The details of the multiobjective optimization strategy can be consulted in a previous contribution [12].

Figure 1 shows the general procedure of the link between MATLAB® and Aspen Plus®.

According to Figure 1 we can observe that the first step is giving the initial design parameters of the chemical process, information that was presented in section 2. In addition, the parameters of the optimization strategy must be defined. With this information the optimization strategy begins with the generation of the initial population. The entire population is composed of individuals, which are sent one by one to Aspen Plus® in order to be

evaluated in terms of objectives and constraints. Once all individuals are evaluated the optimization strategy can continue with the sort, reproduction and mutation steps. If the maximum number of generations is reached, then the optimization strategy reports the Pareto front; otherwise, the procedure continues until that criterion is satisfied.

On the other hand, the Figure 2 shows the details of the Design evaluation in Aspen Plus® orange block. As can be seen, the first step is that MATLAB® sends the instruction to open Aspen Plus®; after that, MATLAB® sends to Aspen Plus® the required information to perform the simulation, and the instruction to perform the simulation is also sent. Once the simulation has been performed, MATLAB® takes back the results of the objectives and constraints, in order to be fed to the optimization strategy.

Finally, MATLAB® sends the instruction to close Aspen Plus®, in order to reduce the use of RAM memory. In the present contribution, the details of how to perform the operations presented in Figure 2 are given, considering a study case for illustrative purposes. It is worth to mention that this procedure was developed with MATLAB® 2007 and Aspen Plus® V7.1, and the code was validated from these versions until MATLAB® 2012 and Aspen Plus® V9.1. However, the instructions can be applied to further versions if the main core of instructions of both software does not change.

Before entering to the detailed instructions of the operations showed in Figure 2, we need to create a file with the simulation of the process of interest in Aspen Plus®. Usually, this file is saved with extension apw; however, the file employed in the linking process is the one with extension bkp. In the simulation file, it is desirable that the names of the blocks and streams follow a logic methodology; this is especially useful in complex chemical processes, and also it helps to the standardization of the optimization code for future cases.

Also, we recommend ordering the components according with the decreasing relative volatility. It is important to initialize the recycled streams (if there are any), i.e. interconnection or recycle flows; this will help to improve the convergence of the schemes where recycle streams are presented.

We are going to illustrate the process with a Petlyuk sequence, which is showed in Figure 3.

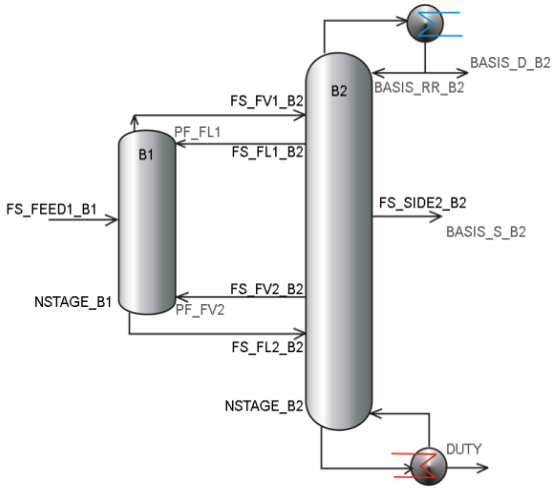

Fig. 3. Petlyuk sequence and its variables

We consider that we are interested in minimizing the number of stages in prefractionator, NSTAGE_B1, the number of stages in main column, NSTAGE_B2, and the heat duty in main column, DUTY. The constraints are the recoveries of the three components presented in the feed stream.

In order to perform this optimization, we manipulate 12 variables which are: feed stage in column B1 (FS_FEED_B1), number of stages in column B1 (NSTAGE_B1), reflux ratio in column B2 (BASIS_RR_B2), number of stages in column B2 (NSTAGE_B2), distillate stream flow of column B2 (BASIS_D_B2), side stream flow of column B2 (BASIS_S_B2), product stage of liquid interconnection flow FL1 in column B2 (FS_FL1_B2), feed stage of vapor interconnection flow FV1 in column B2 (FS_FV1_B2), vapor interconnection flow (PF_FV2), feed stage of liquid interconnection flow FL2 in column B2 (FS_FL2_B2), product stage of vapor interconnection flow FV2 from column B2 (FS_FV2_B2), and the liquid interconnection flow (PF_FL1).

As was mentioned before, we have to identify the objectives and/or constraints of the optimization problem, along with all involved variables, manipulated and required for the simulation.

The objectives, constraints and manipulated variables are vectors, as described next.

Algorithm 1. Definition of objectives, constraints and variables

```
1: Objectives(1)= NSTAGE_B1
2: Objectives(2)= NSTAGE_B2
3: Objectives(3)=DUTY
4: Constraints(1)=Rec1
5: Constraints(2)=Rec2
6: Constraints(3)=Rec3
7: Variable(1)= FS_FEED_B1
8: Variable(2)= NSTAGE_B1
9: Variable(3)= BASIS_RR_B2
10: ⋮
11: Variable(12)= PF_FL1
```

The variables presented in Algorithm 1 represent the initial values for the optimization strategy. Nevertheless, in all optimization strategies, new values have to be generated for all variables that are manipulated and fed to Aspen Plus® to perform the simulation.

These variables must be declared in the optimization code, and they will be used to assign the values from MATLAB® into the simulation file in Aspen Plus®:

Algorithm 2. Assignation of names for the variables

```
1: NameVar1=Variable(1)
2: NameVar2=Variable(2)
3: NameVar3=Variable(3)
4: ⋮
5: NameVark=Variable(12)
```

It is worth to mention that NameVark represents the vector of variables that are manipulated during the optimization strategy.

At this point, we have identified objectives, constraints, manipulated variables and required variables for the simulation in Aspen Plus®, which file has already been created; also, the values generated by the algorithm are declared and stored for its posterior use. Now, the following step is opening Aspen Plus® from MATLAB®.

We initiate this process by giving to MATLAB® the route where the bkp file, that contains the process is located, and also we give some instructions to delete all files that Aspen Plus® could generated in a previous simulation.

This procedure is shown in Algorithm 3.

Algorithm 3. Location of the Aspen Plus® file and delete previous files

```
1: strFileAspenOr  = [ fileStr
'Folder1\Folder2\' strColType '\' strMezcla '\'
strColType  ' ' strMezcla '.bkp'  ];
2: strFileAspen  = [ 'Y:' '\' strColType  ' '
strMezcla '.bkp'  ];
3: try
4:    fileattrib(strFileAspenOr);
5: catch
6:    err = lasterror;
7:    rethrow(err) ;
8: end
9: try
10:   fileattrib('Y:\')
11:   % Delete files generated in previous
simulations in virtual disk Y
12:   delete('Y:\*.*')
13:   delete(strFileAspen)
14:   copyfile(strFileAspenOr,strFileAspen)
15: catch
16:   strFileAspen=strFileAspenOr;
17: end
```

The code instructions shown in Algorithm 3 assign the route to a string type variable. We use a virtual disk where the old files are deleted in order to avoid conflicts; nevertheless, you must to use your personals options in drive letters and routes.

Thus, the file of name ' strColType ' ' strMezcla '.bkp' is located in the Folder strMezcla, inside the Folder strColType, inside the Folder 2, which is located in Folder 1.

The information after the sign % are comments, while the rest are the code lines. In this work, we are going to use Aspen Plus® as a local OLE automation server [6]; in other words, we are going to manage Aspen Plus as an external subroutine that allows to generate the values of objectives and constraints required in the optimization strategy.

In this case 'Apwn.Document' is the programmatic identifier of an OLE-compliant COM server, and h is the handle of the server's default interface.

It is worth to mention that in MATLAB® a handle is a reference to an object. Then, the following instruction (Algorithm 4) opens Aspen Plus® from MATLAB® along with the simulation of interest.

In the previous instructions, in Algorithm 4, ihAPsim variable is the handle of the server. Also, the instruction refers to opens strFileAspen, which is the file specified in Algorithm 3.

In this routine, we use a variable from the class ihNode from the automation server; in this object, Aspen Plus® expose the problem input a result data as a tree structure composed of ihnode objects. The used subroutine navigate across the tree structure of variables in Aspen Plus® until find the last node of the variable name, where the "value" method is used to recall the value; when the last node is an array, the subroutine recall the value of the element in the array. Also, the subroutine performs some validation, but for the complexity of the tree of Aspen Plus® the user must to be sure that the names are correct; otherwise the subroutine ends with an error message.

Once the simulation is open, the names of the compounds used in the simulation are retrieved along with the flow of the main stream of the process in the flowsheet (see Algorithm 5).

In Aspen Plus®, every simulation flowsheet has different names thereby we need to program separated functions for each process. In order to make the code more generic we use a handle to change the unit operations of the process.

The handle for each type of process flowsheet is generated with the code showed in Algorithm 6. The instructions to modify the processes variable are contained in this handle, and it will be presented later. At this point, we can make a simulation with the instructions included in Algorithm 6. Each one of the functions for different process has the next structure. The input arguments are: Solution: an array with the values of the manipulated variables in Aspen Plus®, like number of stages, stream flows, among others; ihAPsim: is the handle of the Aspen Plus® server, with the corresponding bkp file opened. And the output arguments are: SolutionFactibility: return true if the solution is feasible (for instance that the total number of stages is greater than the number of the feed stage); SolConstraints: returns the value of the variables defined as constraints (recoveries in this example); Objectives return the value of the variables defined as objectives (number of stages and heat duty in this example). When the simulation is finished and the interested

Algorithm 4. Open Aspen Plus® from MATLAB®

```
% Open connection to Aspen Plus
ihAPsim = actxserver('Apwn.Document');
% Open Aspen Plus file
ihAPsim.InitFromArchive2(strFileAspen);
% Makes visible the interface to the user
ihAPsim.Visible = true;
% Suppress messages to the user
% This is done to avoid popup messages to the user in Aspen Plus during the optimization process.
ihAPsim.SuppressDialogs = true;
```

Algorithm 5. Retrieve names of the compounds and main stream in the simulation

```
% Retrieving the name of the compounds in the simulation used in the stream FEED1 (the main stream in the flowsheet)
ihNode_MIXED_Elements = GetValueAspenVariable( 'Application.Tree.Data.Streams.FEED1.Input. FLOW.MIXED.Elements',ihAPsim);
for
    j= 0:(ihNode_MIXED_Elements.RowCount(0)-1)
Comps{j+1}= ihNode_MIXED_Elements.ItemName(j);
TotFlow(j+1)= ihNode_MIXED_Elements.Item(j).value;
end
% Retrieving the flow of the main stream in the flowsheet (named FEED1)
TotFlowInput= GetValueAspenVariable( 'Application.Tree.Data.Streams.FEED1.Input. TOTFLOW.MIXED',ihAPsim);
TotFlow=TotFlow*TotFlowInput;
```

Algorithm 6. Generates Handle of the function for the process

```
% Generates Handle of the function for the process
fhandleCall= str2func(['CallProyectName' strProcessType ]);

[SolutionFactibility SolConstraints Objectives]=feval(fhandleCall,SolIni(1,:),ihAPsim);
ihAPsim.Close;
ihAPsim.release;
clear ihAPsim;
```

Algorithm 7. Assignation of names of all variables in Petlyuk sequence

```
% Column B1 (Prefractionator)
% Number of stages in column B1
NSTAGE_B1=floor(Variable(1));
% Feed stage of stream flow FEED1
FS_FEED1_B1=floor(Variable(2));
% Column B2 (main column)
% Reflux ratio in column B2
BASIS_RR_B2=Variable(3);
% Number of stages in column B2
NSTAGE_B2=floor(Variable(4));
% Feed stage of FV1 interconnection stream in column B2
FS_FV1_B2=floor(Variable(5));
% Feed stage of FL2 interconnection stream in column B2
FS_FL2_B2=floor(Variable(6));
% Product stage of FL1 interconnection stream in column B2
FS_FL1_B2=floor(Variable(7));
% Product stage of FV2 interconnection stream in column B2
FS_FV2_B2=floor(Variable(8));
% Product stage of SIDE2 product stream in column B2
FS_SIDE2_B2=floor(Variable(9));
% Product and interconnection stream flows
% FL1 interconnection stream flow
PF_FL1=Variable(10);
% FV2 interconnection stream flow
PF_FV2=Variable(11);
% Stream flow where component A (light component) is obtained in column B2
F_A=Variable(12);
% Stream flow where component B (intermediate component) is obtained in column B2
F_B=Variable(13);
% Distillate stream flow in column B2
BASIS_D_B2=F_A;
% Side stream flow in column B2
BASIS_S_B2=F_B;
%% Assigning variables to the number of stage of distillate and bottoms in the columns. These variables are going to be used to retrieving the values of compositions in each product stream.
% String with the number of stage of the distillate in column B1
strNS_B1_D=int2str(1);
% String with the number of stage of the bottoms stream in column B1
strNS_B1_B=int2str(NSTAGE_B1);
% String with the number of stage of the distillate stream in column B2
strNS_B2_D=int2str(1);
% String with the number of stage of the side stream in column B2
strNS_B2_S=int2str(FS_SIDE2_B2);
% String with the number of stage of the bottoms stream in column B2
strNS_B2_B=int2str(NSTAGE_B2);
```

values are retrieved, we can Close Aspen Plus® and it releases the resources in MATLAB®.

On the other hand, the handle subroutine allows modifying the values of the variables required to perform the simulation in Aspen Plus®. Thereby, we start the process of assigning names for all variables in Petlyuk sequence: prefractionator, main column, product and interconnection stream flows, along with some string assignations required to retrieve the values of composition in each product stream (see Algorithm 7).

Before sending these values to Aspen Plus®, we suggest to perform a feasibility verification of the values of the variables; this verification is focused in the physical meaning of the variables. In this way, the variables sent to Aspen Plus® are consistent or physically feasible.

For instance, in the prefractionator the feed stage number must be minor or equal to the total number of stages, but never bigger, and also greater to zero. This can be expressed as is shown in Algorithm 8. The next step is reinitiating the simulation before the values are changed in Aspen Plus®; this is required to have reliable results on the convergence of the simulation (see Algorithm 9).

Once that the values are assigned to the variables, these must be sending to Aspen Plus® to perform the simulation. In order to do this, we have created other subroutines to simplify the connection to Aspen Plus®. The subroutine SetValueAspenVariable is used to write a value to a variable in Aspen Plus®. The function to call this subroutine is shown in Algorithm 10.

Where the input arguments are: strVariable: string with the name of the variable to read, as appear in the Variable Explorer option, inside the Tools Menu of Aspen Plus®; ValAspen: value to be written in the strVariable of Aspen Plus®; ihAPsim: object of type 'Apwn.Document' successfully initialized with a valid Aspen Plus® bkp file. And the output arguments are: Varargout: funOk MsgAspen; funOk: boolean value indicating a succesfully read; MsgAspen: error message displayed from Aspen Plus® when a variable cannot be read.

In this routine, we use a variable from the class ihNode from the automation server; in this object, Aspen Plus® expose the problem input a result

Algorithm 8. Verification of the feasibility of the initial solution

```
1: FS_FEED1_B1≤NSTAGE_B1
2: FS_FEED1_B1>0
```

Algorithm 9. Reinitiate the simulation in Aspen Plus®

```
1: %% Reinitiate the simulation in Aspen Plus
2: try
3:     ihAPsim.Engine.Reinit(4)
%4=IAP_REINIT_SIMULATION
4: catch
5: end
```

data as a tree structure composed of ihnode objects.

The used subroutine navigate across the tree structure until find the last node of the variable name, where the "value" method is used to recall the value; when the last node is an array, the subroutine recall the value of the element in the array. Also, the subroutine performs some validation, but for the complexity of the tree of Aspen Plus® the user must to be sure that the names are correct; otherwise the subroutine ends with an error message.

Next, we have to set the value of all manipulated and required variables inside Aspen Plus® file to perform the simulation. All route variables can be found in the Variable Explorer option, inside the Tools Menu of Aspen Plus®. Then, we begin changing values of variables in the prefractionator, main column, along with product flows (see Algorithm 11).

Algorithm 11. Change the values of variables in the simulation

```
1: %% Column B1
2: % Changing values of variables of column B1
3: try
4:     % Number of stages in column B1
5:SetValueAspenVariable('Application.Tree.D
ata.Blocks.B1.Input.NSTAGE',NSTAGE_B1,ih
APsim)
6:         % Feed stage FEED1 in column B1
7:SetValueAspenVariable('Application.Tree.D
ata.Blocks.B1.Input.FEED_STAGE.FEED',FS
_FEED1_B1,ihAPsim)
8: % Feed stage FV2 in column B1
9:SetValueAspenVariable('Application.Tree.D
ata.Blocks.B1.Input.FEED_STAGE.FV2',NST
AGE_B1,ihAPsim)
10: % Product stage FL2 in column B1
11:SetValueAspenVariable('Application.Tree.
Data.Blocks.B1.Input.PROD_STAGE.FL2',NS
TAGE_B1,ihAPsim)
12: catch     %% Warning message of error to
set data in column B1
13:    warning('An error occurs during assigning
variables in column B1'); %#ok<WNTAG>
14: end      %% column B1
15: %% Column B2
16: % Changing values of variables of column
B2
17: try
18: % Column B2
19: % Reflux ratio in column B2
20:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.BASIS_RR',BASIS_RR_
B2,ihAPsim)
21: % Number of stages in column B2
22:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.NSTAGE',NSTAGE_B2,i
hAPsim)
23: % Feed stage FV1 in B2
24:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.FEED_STAGE.FV1',FS_
FV1_B2,ihAPsim)
25: % Feed stage FL2 in B2
26:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.FEED_STAGE.FL2',FS_
FL2_B2,ihAPsim)
27: % Product stage FL1 in B2
28:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.PROD_STAGE.FL1',FS
_FL1_B2,ihAPsim)
29: % Product stage FV2 in B2
30:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.PROD_STAGE.FV2',FS
_FV2_B2,ihAPsim)
31: % Product stage SIDE2 in B2
32:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.PROD_STAGE.SIDE2',
FS_SIDE2_B2,ihAPsim)
33: % Product flow FL1 in B2
34: % This instruction sets the value of the
interconnection flow required
35:SetValueAspenVariable('Application.Tree.
Data.Blocks.B2.Input.PROD_FLOW.FL1',PF_
```

```
FL1,ihAPsim)
36: % This instruction updates the initial value of the interconnection flow to improve the convergence
37:SetValueAspenVariable('Application.Tree.Data.Streams.FL1.Input.TOTFLOW.MIXED',PF_FL1,ihAPsim)
38: %Product flow FV2 in B2
39: % This instruction sets the value of the interconnection flow required
40:SetValueAspenVariable('Application.Tree.Data.Blocks.B2.Input.PROD_FLOW.FV2',PF_FV2,ihAPsim)
41: % This instruction updates the initial value of the interconnection flow to improve the convergence
42:SetValueAspenVariable('Application.Tree.Data.Streams.FV2.Input.TOTFLOW.MIXED',PF_FV2,ihAPsim)
43: catch
44: warning('An error occurs during assigning variables in column B2');
45: end
46: %% Product stream flows
47: % Changing values of product stream flows of column B2
48: try
49: % Distillate stream flow in column B2, BASIS_D
50:SetValueAspenVariable('Application.Tree.Data.Blocks.B2.Input.BASIS_D',BASIS_D_B2,ihAPsim)
51: %Side stream product flow in column B2, BASIS_S
52:SetValueAspenVariable('Application.Tree.Data.Blocks.B2.Input.PROD_FLOW.SIDE2',BASIS_S_B2,ihAPsim)
53: catch
54: warning('An error occurs during assigning product stream flows in column B2');
55: end
```

Once all required and manipulated variables are sent to Aspen Plus®, then the simulation is performed with the following instructions (Algorithm 12).

Another subroutine developed to simplify the code is VerifyRunStatusError, which is executed as follows (Algorithm 13).

Algorithm 12. Perform a simulation in Aspen Plus®

```
1: %% Performs the simulation
2: % Starts the simulation
3: try
4: ihAPsim.Run2;
5: catch
6: end
```

Algorithm 13. Verify the status error in the simulation

```
1: function RunStatusError= VerifyRunStatusError(ihAPsim)
```

In this function, we read the error matrix of Aspen Plus® in case that it exists, which is located in the following path: 'Application.Tree.Data. ResultsSummary.Run-Status.Output. PER_ERROR.Elements'. If this matrix does not exist, then the simulation has a successfully end, and we accept to the results of the performed simulation.

As was mentioned before, we have created several subroutines in order to simplify the connection to Aspen Plus®. For instance, GetValueAspenVariable is used to read a variable from Aspen Plus® simulation file, and it is used as follows (Algorithm 14):

Algorithm 14. Verify the status error in the simulation

```
1: function [valAspen,varargout]= GetValueAspenVariable(strVariable,ihAPsim)
```

Where the input arguments are: strVariable: string with the name of the variable to read, as appear in the Variable explorer menu from Aspen Plus®; ihAPsim: object of type 'Apwn.Document' successfully initialized with a valid Aspen Plus® bkp file. The output arguments are: ValAspen: value of the requested variable; Varargout: funOk MsgAspen; funOk: boolean value indicating a succesfully read; MsgAspen: error message displayed from Aspen Plus® when a variable cannot be read. This subroutine is very useful to take back the information from Aspen Plus® to MATLAB®.

At this point, Aspen Plus® has simulated the process, and we have three possible scenarios with respect to the convergence: convergence

without warnings, convergence with warnings and no convergence due to errors. If the simulation converges and there are available results, with or without warnings, then the values of objectives and constraints of interest are taken back to MATLAB®, using these commands (Algorithm 15).

Algorithm 15. Retrieve values for objectives and constraints if the simulation converges

```
1: %% Retrieving output data of interest
2: try
3:     if ~VerifyRunStatusError(ihAPsim)
4:     % Taken back the heat duty of the column
B2
5:     DUTY = GetValueAspenVariable( [
'Application.Tree.Data.Blocks.B2.Output.DUT
Y.' strNS_B2_B ] ,ihAPsim);
6:     % Assigning values for the objectives of
interest: number of stages and heat duty
7:     Objectives(1)=DUTY;              % Heat duty
of the column B1
8:     Objectives(2)=NSTAGE_B1;     % Number
of stages in column B1
9:     Objectives(3)=NSTAGE_B2;     % Number
of stages in column B2
10: % Recovering the value of the composition
of the component of interest in its
corresponding product stream
11: % Composition of component 1 in the
distillate stream of column B2
12: Constraints(1)= GetValueAspenVariable( [
'Application.Tree.Data.Blocks.B2.Output.X.'
strNS_B2_D '.' Comps{1} ] ,ihAPsim);
13:  % Composition of component 2 in the side
stream of column B2
14:  Constraints(2)= GetValueAspenVariable( [
'Application.Tree.Data.Blocks.B2.Output.X.'
strNS_B2_S '.' Comps{2} ] ,ihAPsim);
15: % Composition of component 3 in the
bottoms stream of column B2
16: Constraints(3)= GetValueAspenVariable( [
'Application.Tree.Data.Blocks.B2.Output.X.'
strNS_B2_B '.' Comps{3} ] ,ihAPsim);
17: %Calculation of the recoveries of each
component
18: % Recovery of component 1 in the distillate
stream of column B2.
19:
Constraints(4)=Constraints(1)*F_A/TotFlow(1);
20: % Recovery of component 2 in the side
stream of column B2
21:
Constraints(5)=Constraints(2)*F_B/TotFlow(2);
22: % Recovery of component 3 in the bottoms
stream of column B2
23:
Constraints(6)=Constraints(3)*(TotFlowInput -
F_A - F_B)/TotFlow(3);
```

If the simulation does not converge, then the algorithm allocates infinite heat duty and purities and recoveries of zero. Once that the objectives and/or constraints values are taken back to MATLAB®, the optimization process can continue. We suggest using a data base to record all the values of manipulated variables, objectives and/or constraints; so, all the information of the performance of the optimization strategy and final results are available any time.

The main advantage of using a database if there is not storing limit, in counterpart with the limitation in the number of cells found in Excel®. Also, the direct relation between MATLAB® and Aspen Plus® avoid losing information; for instance, when the connection is through Excel®, this software can crash and all the information will be missing. Also, the use of database allows having data processing tools, or even this database can be used for data mining. It is worth to mention that this software has been used to the study of different types of chemical processes with interesting results [29, 4, 13, 21, 11, 14, 10].

4 Generating bkp Files of Optimal Solutions

Once than the optimization procedure has been performed, we have as results a single or a set of optimal designs; depending if we are using a mono or multi objective strategy.

Thus, it would be desirable to generate automatically the bkp files for all optimal solutions, in order to analyze composition, temperature or pressure profiles, or even use them to generate dynamic files and performing control studies. Next, the procedure to generate bkp files, given a set of design variables for n optimal designs, is presented.

First, we need to create a matrix in MATLAB® where each row represents a complete design of a given process, and the columns represent a variable of each design:

$$\begin{matrix} Var_{1,1} & Var_{2,1} & Var_{3,1} & \cdots & Var_{n\,\mathrm{var},1} \\ Var_{1,2} & Var_{2,2} & Var_{3,2} & \cdots & Var_{n\,\mathrm{var},2} \\ Var_{1,3} & Var_{2,3} & Var_{3,3} & \cdots & Var_{n\,\mathrm{var},1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ Var_{1,ndesigns} & Var_{2,ndesigns} & Var_{3,ndesigns} & \cdots & Var_{n\,\mathrm{var},ndesigns} \end{matrix} \quad (2)$$

Once that the matrix is created the following code is executed. First, the size of the matrix is determined (30) (see Algorithm 16).

Algorithm 16. Calculate the size of the matrix

```
1: %Determine the size of the solutions family
2: % Row number= number of optimal designs
3: % Column number= number of variables of
each optimal design
4: [numSolutions,numVariables]=size(Sol);
```

Then, we initialize a counter, and the handle function is generated according with the type of process. This handle denotes the name of the process or unit operations whose optimal solutions bkp files are going to be generated (see Algorithm 17).

Algorithm 17. Generates the handle function according to the process type

```
1: % Updates the value of the actual solution
2: actualSolution=actualSolution+1;
3: SolNorm=Sol;
4: % Generates the handle function according
to the process type
5: fhandleCall= str2func(['CallProjectName'
strProcessType ]
```

Next, Aspen Plus® is open with the following instructions (Algorithm 18).

Algorithm 18. Open Aspen Plus®

```
1: % Open the connection to Aspen Plus
2: ihAPsim = actxserver('Apwn.Document');
3: % Open the Aspen Plus file
4: ihAPsim.InitFromArchive2(strFileAspen);
5: % Makes visible the application to the user
6: ihAPsim.Visible = true;
7: % Suppress the messages to the user
8: ihAPsim.SuppressDialogs = true;
```

Once Aspen Plus® is opened, then we use a cycle to simulate and save each optimal design, with the following code (Algorithm 19):

Algorithm 19. Save each optimal design

```
1: %% Evaluate each optimal design
2: for j=1:numSolutions
3:     %Calculates the percentage progress
4:     Avance= [ j/numGeneraciones  * 100 ]
5:     [SolutionFactibility     SolutionConstraints
SolutionsObjective]=feval(fhandleCall,SolNor
m(j,:),ihAPsim );
6:    if nout >= 1
7:       if j==1
8:          [n,m] = size (SolutionConstraints);
9:          [n1,m1] = size (SolutionsObjective);
10:           res= zeros(numSolutions,m+1);
11:      end
12:            res(j,:)=[  SolutionConstraints(1,:),
SolutionsObjective(1,1)]       ;
13:   else
14: strFileAspen2   = [ regexprep(strFileAspen,
'.bkp', '') ' S' int2str(j) '.bkp' ];
15: ihAPsim.SaveAs(strFileAspen2 );
16:   end
17: end
18: if nout >= 1
19:    varargout(1) = res;
20: end
```

Finally, Aspen Plus® is closed and the resources are released (see Algorithm 20).

Algorithm 20. Close Aspen Plus®

```
1: %Close Aspen Plus, releasing the resources:
2: ihAPsim.Close;
3: ihAPsim.release;
4: clear ihAPsim;
```

The files generated are going to be located in the same folder where the original file for the optimization is located. The name of the file will be the same of the process type, plus the identification code S1, S2, …, Sn designs, where Sn refers to the n^{th} solution.

5 Conclusion

A procedure to link Aspen Plus® with MATLAB® has been presented, including the generation of bkp files of optimal designs. The link between these two powerful tools has great value, since it allows using the computational capabilities of MATLAB® along with the complete models for chemical process of Aspen Plus®. The main advantage of this link is the reduction in computational resources, since just two softwares are running, and the storage capacity of all generated solutions is not limited, as in the case where Excel® is used as linking software. The availability of this code allows increasing the use of complete models in the optimization of chemical processes.

Acknowledgements

Financial support provided by Universidad Autónoma de Querétaro, Exxerpro Solutions (grant CAWS-1) and CONACyT (grants 239765 and 279753) for the development of this project is gratefully acknowledged.

References

1. **Agarwal, A. & Grossmann, I. E. (2009).** Linear coupled component automata for MILP modeling of hybrid systems. *Computers and Chemical Engineering,* Vol. 33, No. 1, pp.162–175. DOI: 10.1016/j.compchemeng.2008.07.014.
2. **Alabdulkarem, A., Mortazavi, A., Hwang, Y., Radermacher, R., & Rogers, P. (2011).** Optimization of propane pre-cooled mixed refrigerant LNG plant. *Applied Thermal Engineering*, Vol. 31, No. 6-7, pp. 1091–1098. DOI: 10.1016/j.applthermaleng.2010.12.003.
3. **Bhattacharyya, D., Turton, R., & Zitney, S. (2012).** Dynamic simulation and load-following control of an integrated gasification combined cycle (IGCC) power plant with CO2 capture. *AIChE Annual Meeting*, Report Number NETL-PUB-390.
4. **Bravo-Bravo, C., Segovia-Hernández, J. G., Gutiérrez-Antonio, C., Durán, A. L., Bonilla-Petriciolet, A., & Briones-Ramírez, A. (2010).** Extractive Dividing Wall Column: Design and Optimization. *Industrial and Engineering Chemistry Research*, Vol. 49, No. 8, pp. 3672–3688. DOI: 10.1021/ie9006936.
5. **Brunet, R., Guillén-Gosálbez, G., Pérez-Correa, J. R., Caballero, J. A., & Jiménez, L. (2012).** Hybrid simulation-optimization based approach for the optimal design of single-product biotechnological processes. *Computers and Chemical Engineering*, Vol. 37, No. 10, pp. 125–135. DOI: 10.1016/j.compchemeng.2011.07.013.
6. **Cowell, J. (1996).** *Object Linking and Embedding (OLE).* Essential Visual Basic 4.0 Fast. Essential Series (Everything you need to know to develop applications in VB4). Springer, London. pp. 136–144. DOI: 10.1007/978-1-4471-3093-2_18.
7. **Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002).** A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, pp. 182–197. DOI: 10.1109/4235.996017.
8. **Eslick, J. C. & Miller, D. C. (2011**). A multi-objective analysis for the retrofit of a pulverized coal power plant with a CO_2 capture and compression process. *Computers and Chemical Engineering*, Vol. 35, No. 8, pp. 1488–1500. DOI: 10.1016/j.compchemeng.2011.03.020.
9. **Getu, M., Mahadzir, S., & Lee, M. (2013).** Profit optimization for chemical process plant based on a probabilistic approach by incorporating material flow uncertainties. *Computers and Chemical Engineering*, Vol. 59, pp. 186–196. DOI: 10.1016/j.compchemeng.2013.05.026.
10. **Gómez-Castro, F. I., Segovia-Hernández, J. G., Hernández, S., Gutiérrez-Antonio, C., Briones-Ramírez, A., & Garmiño-Arroyo, Z. (2015).** Design of non-equilibrium stage separation systems by a stochastic optimization approach for a class of mixtures. *Chemical Engineering and Processing: Process Intensification*, Vol. 88, pp. 58–69. DOI: 10.1016/j.cep.2014.11.001.
11. **Gómez-Castro, F. I., Rodríguez-Ángeles, M. A., Segovia-Hernández, J. G., Gutiérrez-Antonio, C., & Briones-Ramírez, A. (2011).** Optimal designs of multiple dividing wall columns. *Chemical Engineering and Technology*, Vol. 34, No. 12, pp. 2051–2058. DOI: 10.1002/ceat.201100176.
12. **Gutiérrez-Antonio, C. & Briones-Ramírez, A. (2009).** Pareto front of ideal Petlyuk sequences using a multiobjective genetic algorithm with constraints. *Computers and Chemical Engineering*, Vol. 33, No. 2, pp. 454–464. DOI: 10.1016/j.compchemeng.2008.11.004.
13. **Gutiérrez-Antonio, C., Briones-Ramírez, A., & Jiménez-Gutiérrez, A. (2011).** Optimization of Petlyuk sequences using a multi objective genetic

algorithm with constraints. *Computers and Chemical Engineering*, Vol. 35, No. 2, pp. 236–244. DOI: 10.1016/j.compchemeng.2010.10.007.

14. **Gutiérrez-Antonio, C., Ojeda-Gasca, A., Bonilla-Petriciolet, A., Segovia-Hernández, J. G., & Briones-Ramírez, A. (2014).** Effect of Using Adjusted Parameters, Local and Global Optimums, for Phase Equilibrium Prediction on the Synthesis of Azeotropic Distillation Columns. *Industrial and Engineering Chemistry Research*, Vol. 53, No. 4, pp. 1489–1502. DOI: 10.1021/ie4019885.
15. **Khodadoost, M. & Sadeghi, J. (2011).** Dynamic Simulation of distillation Sequences in Dew Pointing unit of South Pars Gas Refinery. *Journal of Chemical and Petroleum Engineering*, Vol. 45, No. 2, pp. 109–116. DOI: 10.22059/JCHPE.2011.1512.
16. **Kiss, A. A., Segovia-Hernández, J. G., Sorin Bildea, C., Miranda-Galindo, E. Y., & Hernández, S. (2012).** Reactive DWC leading the way to FAME and fortune. *Fuel*, Vol. 95, pp. 352–359. DOI: 10.1016/j.fuel.2011.12.064.
17. **Lababidi, H. M. S., Alatiqi, I. M., & Bañares Alcántara, R. (1996).** Application of controllability analysis tools during the conceptual design stage. *Computers and Chemical Engineering*, Vol. 20, No. 1, pp. S207–S212. DOI: 10.1016/0098-1354(96)00045-2.
18. **Leipold, M., Gruetzmann, S., & Fieg, G. (2009).** An evolutionary approach for multi-objective dynamic optimization applied to middle vessel batch distillation. *Computers and Chemical Engineering*, Vol. 33, No. 4, pp. 857–870. DOI: 10.1016/j.compchemeng.2008.12.010.
19. **Logist, F., Vallerio, M., Houska, B., Diehl, M., & Van Impe, J. (2012).** Multi-objective optimal control of chemical processes using ACADO toolkit. *Computers and Chemical Engineering*, Vol. 37, pp. 191–199. DOI: 10.1016/j.compchemeng.2011.11.002.
20. **Martins, F., & Costa, C. A. V. (2010).** Economic, environmental and mixed objective functions in non-linear process optimization using simulated annealing and tabu search. *Computers and Chemical Engineering*, Vol. 34, No. 3, pp. 306–317. DOI: 10.1016/j.compchemeng.2009.10.015.
21. **Miranda-Galindo, E. Y., Segovia-Hernández, J. G., Hernández, S., Gutiérrez-Antonio, C., & Briones-Ramírez, A. (2011).** Reactive Thermally coupled Distillation Sequences: Pareto Front. *Industrial and Engineering Chemistry Research*, Vol. 50, No. 2, pp. 926–938. DOI: 10.1021/ie101290t.
22. **Navarro-Amorós, M. A., Caballero, J. A., Ruiz-Femenia, R., & Grossmann, I. E. (2013).** An alternative disjunctive optimization model for heat integration with variable temperatures. *Computers and Chemical Engineering*, Vol. 56, No. 3, pp. 12–26. DOI: 10.1016/j.compchemeng.2013.05.002.
23. **Ochoa-Estopier, L. L. M, Jobson, M., & Smith, R. (2013).** Operational optimization of crude oil distillation systems using artificial neural networks. *Computers and Chemical Engineering*, Vol. 59, pp. 178–185. DOI: 10.1016/j.compchemeng.2013.05.030.
24. **Pham, Q. T. (2012).** Using fuzzy logic to tune an evolutionary algorithm for dynamic optimization of chemical processes. *Computers and Chemical Engineering*, Vol. 37, pp. 136–142. DOI: 10.1016/j.compchemeng.2011.08.003.
25. **Ramzam, N. & Witt, W. (2006)**. Methodology for decision support among conflicting objectives using process simulators. *Computer Aided Chemical Engineering*, Vol. 21, pp. 415–420.
26. **Rebennack, S., Kallrath, J., & Pardalos P. M. (2011).** Optimal storage design for a multi-product plant: A non-convex MINLP formulation. *Computers and Chemical Engineering*, Vol. 35, No. 2, pp. 255–271. DOI: 10.1016/j.compchemeng.2010.04.002.
27. **Tona-Vásquez, R. V., Jiménez Esteller, L., & Bojarski, A. D. (2008).** Multiscale Modeling Approach for production of Perfume Microcapsules. *Chemical Engineering and Technology*, Vol. 31, No. 8, pp. 1216–1222. DOI: 10.1002/ceat.200800174.
28. **Urselmann, M., Barkmann, S., Sand, G., & Engell, S. (2011).** Optimization-based design of reactive distillation columns using a memetic algorithm. *Computers and Chemical Engineering*, Vol. 35, No. 5, pp. 787–805. DOI: 10.1016/j.compchemeng.2011.01.038.
29. **Vázquez-Castillo, J. A., Venegas-Sánchez, J. A., Segovia-Hernández, J. G., Hernández-Escoto, H., Hernández, S., Gutiérrez-Antonio, C., & Briones-Ramírez, A. (2009).** Design and Optimization, using Genetic Algorithms, of Intensified Distillation Systems for a Class of Quaternary Mixtures. *Computers and Chemical Engineering*, Vol. 33, No. 11, pp. 1841–1850. DOI: 10.1016/j.compchemeng.2009.04.011.
30. **Yue, D., Kim, M. A., & You, F. (2013).** Design of Sustainable Product Systems and Supply Chains with Life Cycle Optimization Based on Functional Unit: General Modeling Framework, Mixed-Integer Nonlinear Programming Algorithms and Case Study on Hydrocarbon Biofuels. *Sustainable Chemistry and Engineering,* Vol. 1, No. 8, pp. 1003–1014. DOI: 10.1021/sc400080x.
31. **Yue, D. & You, F. (2013).** Sustainable scheduling of batch processes under economic and

environmental criteria with MINLP models and algorithms. *Computers and Chemical Engineering*, Vol. 54, pp. 44–59. DOI: 10.1016/j.compchemeng.2013.03.013.

32. **Zhang, H. & Rangaiah, G. P. (2012).** An efficient constraint handling method with integrated differential evolution for numerical and engineering optimization. *Computers and Chemical Engineering*, Vol. 37, pp. 74–88. DOI: 10.1016/j.compchemeng.2011.09.018.

Article received on 14/05/2018; accepted on 15/07/2018.
Corresponding author is Abel Briones-Ramírez.