Introduction to IPESE software for students carrying out a project at IPESE

Laurence Tock, Matthias Dubuis, Leda Gerber

First version 2009,
update November 11, 2013
Contents

1 Introduction .................................................. 3
   1.1 Goals ................................................ 3
   1.2 IPESE’s research areas ............................... 3

2 OSMOSE concept ........................................... 4
   2.1 OSMOSE essential purpose .......................... 4
   2.2 Structure ............................................ 4

3 OSMOSE description ...................................... 6
   3.1 Illustrative example .................................. 6
   3.2 OSMOSE 1: Model interaction ...................... 7
      3.2.1 Model definition ................................ 7
      3.2.2 Model resolution sequence .................... 7
   3.3 OSMOSE 2: Computation ............................. 10
      3.3.1 OneRun .......................................... 10
      3.3.2 Moc ............................................ 11
      3.3.3 Sensal ......................................... 12
   3.4 OSMOSE 3: Information extraction ................. 12

4 Implementation ............................................ 14
   4.1 Frontend ............................................. 14
   4.2 “Command” part ...................................... 16
   4.3 Problem definition ................................. 16
      4.3.1 Model definition ................................ 17
      4.3.2 OneRun .......................................... 19
      4.3.3 Sensal ......................................... 19
      4.3.4 Moc ............................................ 19
   4.4 Part that must not be edited ....................... 22

5 Softwares used at IPESE ................................. 23
   5.1 Reiasm VALI ......................................... 23
      5.1.1 Model Establishment ............................. 23
      5.1.2 Model Resolution ............................... 25
      5.1.3 Tricks and Tips ................................ 27
   5.2 Energy integration ................................ 28
   5.3 MOX .................................................. 29
   5.4 LCA .................................................. 29
      5.4.1 Output of a LCA ................................. 31
CONTENTS

5.4.2 Implementation of the ecoinvent database .............. 32
5.4.3 Configuration in the frontend .................. 32
5.5 Equipment cost .............................................. 33
5.6 Energy Technologies ........................................ 36
5.7 GUI .......................................................... 36

6 Additional Information ............................................. 37

Bibliography ............................................................ 38

A Results extraction .................................................. 39
A.1.1 Structure ...................................................... 39
A.1.2 Tag Extraction ................................................ 39
A.1.2 Plot Tags ..................................................... 42
A.2 Composite curves ............................................. 42

8 VAD ................................................................. 46
Chapter 1

Introduction

1.1 Goals

This introduction is addressed to students accomplishing a bachelor/master project in the Industrial Process and Energy Systems Engineering (IPESE) group. Its purpose is to expose the softwares that are used or developed in the group and that are common to every project. For those which are specific to one student project, please contact your assistant. The document will consist in three main parts:

- Fundamental concepts through an illustrative example
- Software presentation
- Documentation and information

It’s important to underline that the goal of this introduction is to present the important concepts and main issues, and to know where and how to find information in case of problems.

1.2 IPESE’s research areas

The Industrial Process and Energy Systems Engineering (IPESE) group activities are separated into three parts. The main goal is to develop computer aided energy system design and optimization tools using thermo-economic and environomic modeling and applying exergy analysis, process integration and multi-objective optimization concepts.

- Process design
- Process integration
- Urban systems

The software presented here is mainly used and developed in the IPESE group. It’s important to remind that softwares or associated documentation are constantly under development and may not be up to date.
Chapter 2

OSMOSE concept

2.1  OSMOSE essential purpose

Before making any computation (optimization, sensitivity analysis, ...) on an energy system, it is necessary to model it. However, in many cases, SEVERAL softwares (some professional and some not) have to be used, what induces a lot of data transfer. The main purpose of OSMOSE is to manage:

1. Modelization (Representation of an existing system)
2. Computation (Calculations performed on the model)

in only ONE tool.
For more information, see [3, 4, 5] and the OSMOSE documentation.

2.2  Structure

The structure of the three levels of the OSMOSE functionalities as shown in fig. 2.1.

![Figure 2.1: OSMOSE onion layer scheme](image)

4
Where OSMOSE$_1$ (fig. 2.2) represents the model itself. The different softwares that may be used to build a model and the way to implement it in OSMOSE will be described.

Figure 2.2: OSMOSE$_1$

OSMOSE$_2$ (fig. 2.3) deals with the different computation types that can be performed on the model. Each computation type will be discussed and the way to define it in OSMOSE will be explained.

Figure 2.3: OSMOSE$_2$

OSMOSE$_3$ (fig. 2.4) represents the data extraction and treatment from the computation results. The way to extract and interpret results will be discussed.

Figure 2.4: OSMOSE$_3$

It is to note that the onion layer scheme of OSMOSE is only a way to explain it; the programmed structure is no exactly the same.
Chapter 3

OSMOSE description

In this chapter, the main functionalities of OSMOSE, that were briefly described in section 2.2 will be detailed.

3.1 Illustrative example

The example that will be used to illustrate the main concepts of OSMOSE is the one developed by Autissier and Palazzi [4] represented in fig. 3.1.

![Diagram of SOFC-Gas turbine]

Figure 3.1: SOFC-Gas turbine

The process consists of the following steps:

1. **The fuel processing part:** converting methane, air and water into syngas (constituted of \(H_2, CO, CH_4,\ldots\))

2. **The fuel cell:** a Solid Oxide Fuel Cell (SOFC) similar to those developed in the group of J. Van Herle. One of its key parameters is the fuel utilization. Indeed, the syngas isn’t totally consumed in the SOFC, and part of it remains in the exhaust gas.
3. **The post-combustion**: the fuel remaining in the exhaust gas is burnt in a burner.

4. **The gas turbine**: the exhaust gases leaving the burner are expanded in a turbine to recover work.

In the following paragraphs this problem will be referred to by “Example”.

### 3.2 OSMOSE 1 : Model interaction

#### 3.2.1 Model definition

A model consists in a representation of the reality. This means making assumptions or simplifications that can be taken into account in the mathematical formulation of phenomena. It is defined by several elements:

- Equations and parameters: They define the behavior of the model. One example of an equation is the energy balance for a compressor and one of an associated parameter is the efficiency.

- Decision variables: They define the configuration of the model. It can for example be the number of cells for a SOFC stack.

**Example**: The decision variables that have been chosen in the Example are given in table 3.1.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P\text{turb}$</td>
<td>[1.1 6]</td>
<td>bar</td>
<td>Pressure upstream the turbine</td>
</tr>
<tr>
<td>$T_{\text{ref}}$</td>
<td>[800 1100]</td>
<td>K</td>
<td>Steam reforming temperature</td>
</tr>
<tr>
<td>$T_{\text{sofc}}$</td>
<td>[970 1120]</td>
<td>K</td>
<td>Fuel cell inlet temperature</td>
</tr>
<tr>
<td>$\mu$</td>
<td>[0.5 0.9]</td>
<td>-</td>
<td>Fuel utilization</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>[1.5 5]</td>
<td>-</td>
<td>Air factor</td>
</tr>
<tr>
<td>$\xi_{\text{OC}}$</td>
<td>[0.001 0.5]</td>
<td>-</td>
<td>Oxygen to carbon ratio</td>
</tr>
<tr>
<td>$\xi_{\text{SC}}$</td>
<td>[0.001 4]</td>
<td>-</td>
<td>Steam to carbon ratio</td>
</tr>
<tr>
<td>$n_{\text{cell}}$</td>
<td>[30 1000]</td>
<td>-</td>
<td>Number of cells</td>
</tr>
</tbody>
</table>

Table 3.1: Decision variables

It means that the models have been developed in such a way that the whole system is determined once each of these variables has been defined in the respective validation range.

#### 3.2.2 Model resolution sequence

Several softwares are compatible with OSMOSE as represented in fig 3.2.
CHAPTER 3. OSMOSE DESCRIPTION

![Diagram](image)

Figure 3.2: Model resolution sequence

**Physical model**

The physical model allows to calculate the thermodynamic state of the model. Currently, two kinds of software are used:

- Matlab function.
- Flowsheet model: Belsim VALI described more in detail in section 5.1, or Aspen Plus or gPROMS or etc.

After this step, all the thermodynamic states induced by the decision variables are defined. If for example an engine is completely modeled (including efficiency, nominal power, type of fuel,...), all the desired values (power, exhaust gas temperature,...) can be calculated by knowing the inlet fuel mass flow rate.

**Example:** Concerning the Example, the VALI layout of the fuel processing is given in fig. 3.3. Material and energy streams are symbolized by arrows. The equations of the pre-defined units (e.g. compressor) are already implemented in VALI. So there’s no need to program them. Specific equations (e.g. phenomena happening in a fuel cell) can be defined “by hand” in a flex code. In the Example each heat exchanger is modeled as expressed in fig. 3.3. This is NOT imposed by VALI. However, the advantage of this representation is that it allows to optimize heat exchanges through the energy integration technique.\(^1\) This means making the assumption that whatever will be the configuration of the system (given by the decision variables), the system will be integrated, i.e. the heat exchange will be optimal.

\(^1\)see course: Advanced energetics, François Marchal, autumn semester, mechanical engineering.
CHAPTER 3. OSMOSE DESCRIPTION

Figure 3.3: Representation of the fuel processing step in Belsim VALI

\[ M_{in} \rightarrow \square \rightarrow M_{out} \]
\[ \downarrow \]
\[ Q_{out} \]

Figure 3.4: Heat exchanger modelization

**Energy Integration**

Energy integration is performed with an MILP solver (ampl/glpk/EASY). See section 5.2 for more information. These techniques allow to qualify an energy system with regard to the minimal energy requirement. It also computes the units that are needed to satisfy the heating and cooling demands of the process (e.g. boiler or cooling unit). In other words, the goal of this step is to provide a rational use of energy. This can be considered as a kind of optimization, but must not be confused with system optimization (see section 6.2). Indeed, the energy integration optimizes the energy system for a given configuration, whereas system optimization is related to a configuration change.

**Example:** The energy integration computes:
• How to recover internally heat through heat exchange in order to minimize the heating needs
• Size a boiler to satisfy the heating demand
• Cooling water facilities characteristics

Performance evaluation

Once the energy integration has been performed, the system can be considered as being defined. However, it may be interesting to define some values reflecting the performances of the system. It is possible to program any function on Matlab that computes the energy efficiency, the costs or the environmental impacts, for example based on values calculated in previous steps. An LCA (Life Cycle Assessment) module is available. It consists in indicators quantifying environmental impact and is programmed on Matlab (more details in section \ref{sec:performance}).

Example: In the present case, two objectives have been taken into account:

• The efficiency.
• The cost. A database of costs functions for different equipments is implemented in Matlab (called “Equipment Cost”, see section \ref{sec:database}).

3.3 OSMOSE 2 : Computation

Once the model is defined, several types of computations can/should be carried out. The list presented here is not exhaustive but consists of the most common ones. They are called:

• OneRun : Run of the model for a given set of decision variables
• Moo : Multi-objectives optimization
• Sensi : Sensitivity analysis

3.3.1 OneRun

**Input:** One set of decision variables

**Output:** Performances corresponding to the set of decision variables

The OneRun computation is used to run the model only once. One should always perform it before making another computation in order to be sure that the model (physical model, energy integration and performances evaluation) works well. However, it’s important to remind that a model can run correctly in OneRun, but can still have problems in other computations or for other decision variables. This may be due to convergence problems at the level of the flowheeting or the energy integration. Indeed, convergence depends on the choice of the initial values, since the solver (based on derivatives calculation) may not find a solution if, the decision variables are too far from their default values.
3.3.2 Moo

Input: A range of validity for each decision variable
Optimization parameters

Output: Pareto curves with associated results
Moo is an acronym for Multi-Objectives Optimization (section 6.3). In engineering, several objectives are often pursued simultaneously. In many cases they are conflicting.

In case of two different objectives, it is possible to represent the results on a Pareto curve as in fig. 3.3 that represents the objectives on the axes. The best solutions are represented by the different types of points. Each type is called a cluster. A cluster is a set of solutions that are alike. The hatched area represents the feasible zone. For each point in this area, it is possible to find a better solution on both objectives.

![Figure 3.5: Pareto curve](image)

Moo is based on an evolutionist algorithm. This term is coming from its analogy with the Darwin law: “survival of the fittest”. An individual is in the system defined by a given set of decision variables. As each of these variables can vary in a certain range, it is possible to generate randomly several individuals, i.e. several sets of decisions variables. The model yields the performances associated to each set, what allows to compare individuals between each other.

Then, the individuals which gave the best performances are kept (i.e. they are the “fittest”) and new individuals are created from their decision variables. And so on until the maximum number of iteration (imposed by the user) is reached. The results are stored in a folder “OSMOSE_results” in the same path than the model folder.

Example: Moo has been used in our case. Indeed, if the purpose is to have an efficient system, it will be necessary to buy efficient equipments what is expensive and vice versa. This two objectives are clearly conflicting, that’s why Moo has been used to optimize this problem.

---

See course: Modeling and Optimization of Energy systems, Prof. Marchal, spring semester, mechanical engineering.
3.3.3 Sensi

**Input:** Upper and lower bound for one particular decision variable
Number of steps

**Output:** Results evolution in the chosen range of the decision variable
The Sensi computation is used to see the influence of the variation of the numerical value of one specific decision variable on the performance. A defined number of runs is performed for the same model and at each run the value of one decision variable is changed until a certain variation range defined by the user is covered. Sensitivity analysis on several decision variables should be performed before the optimization in order to identify the variables that are the most relevant for doing Moo.

3.4 OSMOSE 3: Information extraction

For each computation (one run, sensitivity analysis, optimization) the results are stored in a new folder run_xxxx in the folder OSMOSE_results created in the working folder. Depending on the computation that is made, the run_xxxx folder contains the frontend Matlab file and the following folders storing the information in different files:

**OSMOSE_model:** This folder regroups all the files that define the model which is called in the frontend: ValiFile.blk, Matlab files (precomputation, post-computation (efficiency and cost calculation), EI (defining the energy integration problem)), etc.

**OSMOSE_OneRun:** This folder contains the o.mat file which stores the information about the model and the results in different fields. If Vali is launched in the computation, a folder with the model’s name contains the temp.meo file which stores the values of the constant Vali output Tags. If the energy integration is performed the EnergyIntegration folder is created and all the information about the results and the composite curves are stored there.

**OSMOSE_sensitivity:** This folder regroups the o.mat file and a folder for each point of the sensitivity analysis storing the same information as for the one run.

**OSMOSE_moo:** This folder contains the different evaluations and the information required for the recomputation.

**OSMOSE_pareto:** This folder regroups the o.mat file and for each point that is recomputed from the optimization a folder storing the same information as for the one run.

Once optimization has been performed with Moo, numerical results are stored but no graphs are drawn. The module Recompute has to be run to recompute the points of the pareto one by one. In the end, a graphical user interface described in fig. 3.3 will be automatically opened. The same interface is automatically called for the Sensi computation.
Figure 3.6: Graphical user interface for Pareto plot

1. Selection of the x axis: by default, one of the two first objectives is selected, but it is possible to choose another one if necessary.

2. Selection of the y axis: each tag which has been defined in the model can be selected.

3. Plot the graphs in order to visualize the results. At this step the graphic is not saved.

4. Save the current plot. OSMOSE will create a folder “OSMOSE_plot” in the same path than the folder “OSMOSE_results”. It is important to move the plots in a new folder before making a new Recompute, because there is no warning when a new figure has the same name than a previous one. It will be systematically overwritten.

Detailed information and practical tips for results extraction are found in Appendix A.
Chapter 4

Implementation

In chapter 3, the main principles of OMOSE have been explained. Here the way to implement it as Matlab function in order to run it with OMOSE will be described.

4.1 frontend

There is no graphical user interface for OMOSE. A Matlab function usually called “frontend” is used. It will be only globally described here since a detailed documentation is available for OMOSE. A frontend can be divided into three parts:

- A "Command" part
- A problem definition part
- A part that must not be edited

They are summarized in listing 4.1.

This chapter is just an overview explaining frontend content. Detailed specifications can be found in OMOSE documentation.

```
function o = frontend_example
warning off
o.Silent = 0;
o.DoOneRun = 1;
o.DoSensi = 0;
o.DoMoo = 0;
o.DoRestartMoo = 0;
o.DoPushPareto = 0;
o.DoRecompute = 0;
o.DoParetoAnalysis = 0;
o.DoReport = 0;
o = launch_osmose(o);

% -------------------------------------------------------------
function o = DefineModel(o)
...% -------------------------------------------------------------
```

"Command" part: BEGINING

"Command" part: END

Problem definition part: BEGINING
function o = DefineOneRun(o)
...
% ----------------------------------------------------------------------
function o = DefineSensi(o)
...
% ----------------------------------------------------------------------
function o = DefineMooOptim(o)
...
% ----------------------------------------------------------------------
function o = DefineRestartMoo(o)
...
% ----------------------------------------------------------------------
function o = DefinePushPareto(o)
...
% ----------------------------------------------------------------------
function o = DefineRecompute(o)
...
% ----------------------------------------------------------------------
function o = DefineParetoAnalysis(o)
...
% ----------------------------------------------------------------------
function o = launch_osmose(o)
% ----------------------------------------------------------------------
o = DefineModel(o);
if o.DoOneRun == 1
    o = DefineOneRun(o);
end
if o.DoSensi == 1
    o = DefineSensi(o);
end
if o.DoMoo == 1
    o = DefineMooOptim(o);
end
if o.DoRestartMoo == 1
    o = DefineMooOptim(o);
    o = DefineRestartMoo(o);
end
if o.DoPushPareto == 1
    o = DefineMooOptim(o);
    o = DefinePushPareto(o);
end
if o.DoRecompute == 1
    o = DefineMooOptim(o);
    o = DefineRecompute(o);
end
if o.DoParetoAnalysis == 1
    o = DefineMooOptim(o);
    o = DefineParetoAnalysis(o);
end
o = run_frontend(o);
% ----------------------------------------------------------------------

Listing 4.1: Frontend example - General Model Definition
The following sections describe the different functions of the part that must be edited.

4.2 “Command” part

This part is mainly used to select which type of calculation (OneRun, Moo, Sensi, Recompute) OSMOSE must do. Only one of them can be selected.
As a reminder, a Recompute can be performed only once an optimization has been run.

4.3 Problem definition

As shown in fig. [Fig] in this section the matlab file defining the model (for example ModelXXX.m) is called with the osmose_appendModelMFileNames function. Details about how to define your model are found in the EnergyTechnologies documentation. Several information are given in Section [XXX].
In addition, you can activate or deactivate the energy integration and LCA calculation here and fill the additional fields. For details for the LCA see section [XXX].

```matlab
function o = DefineModel(o)
% Definition of the model(s)
% Fill first the general model specifications, then each model can be
% completed with specific properties according to the software used.
% Available softwares : vali, easy, AMPL/glpk
% Several models can be called with osmose_appendModelMFileNames(o,'ModelXXX.m','ModelZZZ.m'))

o = osmose_appendModelMFileNames(o,'ModelXXX.m');

% Energy Integration
% Specify if energy integration is performed
% and which software is used [required].
o.EnergyIntegration = 1; \ % 1: yes, 0: no if yes, define the fields below.
% Specify the objective used for the resolution of the heat cascade [optional].
% Ex.: o.Eiamp1.Objective = {'Operating cost'}; or {'Exergy'}; or {'Mechanical power'}); {'MER'});
% GLPK (if using GLPK instead of AMPL)
o.Eiamp1.MILPParserName = {'glpk'};
o.Eiamp1.Params.Software = {'glpk'};
o.Eiamp1.Objective = {'OperatingCost'}; {'MER'}; \%
o.glpk.Executable = {'C:\glpk-4.43\bin\glpsol.exe'}; \% ie c:\......\C:\Program Files\GnuWin32\bin\glpsol.exe
o.glpk.OutMsgFile = {'GlpkOutMsg.txt'};

% Life cycle assessment
% Specify if LCA is performed
o.ComputeLCIA = 1; \ % set 1 if you want LCIA analysis, 0 otherwise \% if
% yes, define the fields below.
```

16
%o.LCIA.ImpactMethod = {'impact02_end'}; % 'cm101_short', 'cm101_ext',
   'e1999a', 'e199ee', 'e199ii', 'impact02_end', 'ced', 'ubp', 'footprint',
   'ipcd01'
 o.LCIA.ImpactMethod = {'e1999a'};
 o.LCIA.ResultStage = {'end'};
 o.LCIA.IncludeProcessEquipment = 1;

Listing 4.2: Model definition

4.3.1 Model definition

The main information of the model definition are the model software (i.e. Matlab, Vali, ...), the model name the files for pre- and post-computation, the tags definition and the definition of the energy integration problem (i.e. units, hot/cold streams, ...). The beginning of a typical Model definition file is reported in List 4.3

technology=ModelXXX

%% General
%% Classification by grouping
technology.Group = {'Model example'}; % First level
technology.Type = {'example 1'}; % Second level
technology.SubType = {''}; % Third level of classification

%% OSMOSE parameters
% Specify the version of OSMOSE you use [optional].
technology.OMOSEVersion = {'2.7.8'};
technology.ETVersion = {'2.0.0'};
technology.OMOSEVersion = {'0.8.2'};
% User tagging
% Define the name of the model [required].
technology.TagName = {'SOFc'}; % User-defined tag name
technology.DisplayName = {'SOFc system'}; % User-defined pretty short name
technology.Description = {'illustrative SOFC example'}; % Short paragraph containing the technology description
% Files
technology.FileName = {'SOFc.bls', 'ModelXXX.m', 'premod.m', 'postEl.m'};
% Define the software the model is developed in [required].
technology.Software = {'vali'};
% Define the storage location of the model [required].
technology.Location =pwd;
% Vali model definition
technology.FileSelected = {'SOFc.bls'};
technology.SoftwareSpecific.PPD = {'Main'};
technology.SoftwareSpecific.AdditionalMEA = {'ToVali.mea'};

Listing 4.3: Matlab code for model definition

Pre- and post-computation files can either be called before/after the Model or before/after the energy integration Fig. 4.2. The are defined as following by
It’s important to underline that objective function are considered as PostEIMFunction.

Listing 4.4: Pre- and post-computation files

The details about how to implement the energy integration are found in the Energy Technologies documentation. The streams, for example “QT streams”, which are implemented as \([T_{in}, H_{in}, T_{out}, H_{out}, DT \text{min}/2]\) (List 4.3)

Listing 4.5: Streams definition for energy integration

**Tags definition**

What is a tag? As discussed in chapter 4, OSMOSE is a platform dealing with several softwares (VALI, ampl, glpk, Matlab,...). It implies information exchanges between each of these softwares. The way OSMOSE proceeds is that each Value that has to be stored is saved in a structure called “tag”. A typical structure is expressed in list 4.4

% Define a tag as input of model i
nt = 0; % Tag index to be incremented at each new tag definition
nt = nt + 1;
technology.Tag(nt).TagName = {'ZFP_T'};
technology.Tag(nt).DisplayName = {'Temperature of furnace'};
technology.Tag(nt).Unit = {'K'};
technology.Tag(nt).Status = {'CST'};
technology.Tag(nt).DefaultValue = {'900'};
% Define a tag as output of model i
% If you want to recover the value of the tag from the calculation
% the status is 'OFF' and there is no DefaultValue field i
nt=nt+1;
technology.Tag(nt).TagName = {'X_T'};
technology.Tag(nt).DisplayName = {'Temperature of stream X'};
technology.Tag(nt).Unit = {'K'};
technology.Tag(nt).Status = {'OFF'};

Listing 4.6: Tag structure

However, not all these fields are necessary. See OSMOSE and EnergyTechnologies documentation for more information. If the model is a VALI flowsheet, and the tag is specified as an input tag ('CST' in Vali also), the same name has to be used in the model definition function and in the VALI flowsheet. Concretely, it means that if the inlet massflow of fuel is called on VALI "MFLFUELIN" for example, the same name should be used in the field Tags(i).TagName.

Once the model definition function is written, the different computation have to be defined as expressed in section [3.2.1] and [3.2.2].

4.3.2 OneRun

In a OneRun computation, the only thing that have to be defined are the different values imposed to the model.

4.3.3 Sensi

The name of the function used to define sensitivity analysis is DefineSensi. The purpose here is to define the range of the variable, what is done through the fields LowerBound and UpperBound, as well as to define the number of points to be computed in this range (NumberOfSteps) [list. [1.7]]

\[
\text{function } o = \text{DefineSensi}(o) \\
i=0; \\
% define the decision variable for the sensitivity analysis \\
i=i+1; \\
o.Variables(i).ModelTagName = {'SOFC'}; \\
o.Variables(i).TagName = {'X_T'}; \\
o.Variables(i).DisplayName = {'Temperature of stream X'}; \\
o.Variables(i).Unit = {'K'}; \\
o.Variables(i).LowerBound = 800; \\
o.Variables(i).UpperBound = 1100; \\
o.Variables(i).NumberOfSteps = 10; \\
\]

Listing 4.7: Matlab code for sensitivity analysis

4.3.4 Moo

The name of the function used to define an multi-objective optimization is DefineMooOptim [list. [1.8]]. The purpose here is to define the number of objectives (nobjectives), the objectives that have to be maximized or minimized,
as well as the decision variables with their associated range of validity. Other parameters to be fixed for the evolutionist algorithm are the number of clusters (nclusters), the maximal number of evaluations (max_evaluations) and the size of the initial population (InitialPopulationSize). Moreover, the parameters for parameters for the recomputation (DefineRecompute) and the pareto analysis (DefineParetoAnalysis) have to be defined.

```matlab
function o = DefineMooOptim(o)

o.Moo.mobjectives = 2; % number of objectives
o.Moo.nclusters = 1;
o.Moo.max_evaluations = 800;
o.Moo.InitialPopulationSize = 100;
% --- OBJECTIVES --- %
i=0;
i=i+1;
o.ObjectiveFunction(i).Model = {'SGFC'};
o.ObjectiveFunction(i).DisplayName = {'Total cost'};
o.ObjectiveFunction(i).TagName = {'total_cost'};
o.ObjectiveFunction(i).Unit = {'euro'};
o.ObjectiveFunction(i).MinOrMax = {'min'}
i=i+1;
o.ObjectiveFunction(i).Model = {'SGFC'};
o.ObjectiveFunction(i).DisplayName = {'Energetic efficiency'};
o.ObjectiveFunction(i).TagName = {'Eff_energy'};
o.ObjectiveFunction(i).Unit = {'%'};
o.ObjectiveFunction(i).MinOrMax = {'max'}

% --- VARIABLES --- %
i=0;
i=i+1;
o.Variables(i).ModelTagName = {'SGFC'};
o.Variables(i).TagName = {'Tref'};
o.Variables(i).DisplayName = {'Reforming Temperature'};
o.Variables(i).Unit = {'K'};
o.Variables(i).Limits = [800 1100];
o.Variables(i).Is_integer = 0;
i=i+1;
o.Variables(i).ModelTagName = {'SGFC'};
o.Variables(i).TagName = {'SGFC Temperature'};
o.Variables(i).DisplayName = {'SGFC temperature'};
o.Variables(i).Unit = {'K'};
o.Variables(i).Limits = [970 1100];
o.Variables(i).Is_integer = 0;

% --- DISPLAY PARAMETERS --- %
o.Moo.monitor = ... %
\{o.Moo.InitialPopulationSize = 'moo_restart_monitor'
o.Moo.InitialPopulationSize = 'moo_count_monitor' \} % population display in the prompt [number of eval.]
```
o.Moo.InitialPopulationSize = 'moo_speed_monitor' % speed display in the prompt [number of eval.]
o.Moo.InitialPopulationSize = 'moo_objective_monitor' % objective display in the prompt [number of eval.]
o.Moo.InitialPopulationSize = 'moo_draw' % graph displaying
o.Moo.InitialPopulationSize = 'moo_stop_monitor' % number of evaluations
;

o.Moo.drawing.invert = 1;
o.Moo.test_name = mfilename;
%

function o = DefineRestartMoo(o)
% Definition of the results folder for restarting an optimisation
% Specify the name of the results directory
o.Paths.MooRunDirectory = {'\home\CMSSSE_results\run_0003'};
%

function o = DefineRecompute(o)
% Definition of files and folders for recomputing the points on the Pareto front
%
function o = DefineParetoAnalysis(o)

o.Paths.MooRunNumber = {'run_0003'};
% Choose if the Pareto Curve should be plotted or not
oGraphicOptions.PlotPareto = 1;
oGraphicOptions.PlotDecVar = 0;
oGraphicOptions.PlotScatterplots = 0;
oGraphicOptions.PlotCorrelations = 0;
oGraphicOptions.PlotCorrelationCoefficients = 1;
oGraphicOptions.PlotVariableDistribution = 1;
% Choose the objective which should be visualized in the RANKED VARIABLE DISTRIBUTION
oGraphicOptions.VariableDistributionObjectiveNumber = 2;
o.paretoAnalysis.PlotParetoHistory = 0;
o.paretoAnalysis.PlotCompositeCurvesEvolution = 0;
oGraphicOptions.ColorStyle = {'summer'};
oGraphicOptions.SavePlots = 0;
Listing 4.8: Matlab code for optimization

4.4 Part that must not be edited

That part that must not be edited [list. 24] is required to initialize the run and should not be modified.
Chapter 5

Softwares used at IPESE

5.1 Belsim VALI

Vali is a process simulation software edited by Belsim that applies a simultaneous resolution method to solve the models. The main features of this software are:

- Flowsheeting
- Process modeling and resolution
- Data validation and reconciliation

This chapter summarizes some important basics for the establishment and resolution of process models and gives tips to overcome some common problems that might occur. Detailed information and explanations on the software, its applications and handling, and the different objects are found in the software's Help Manual which can be accessed through the menu bar.

5.1.1 Model Establishment

The process modeling is based on the development of a flowsheet. The model has to be coherent and easily comprehensible by another person without supplementary information, therefore the following rules have to be respected:

- **Organization:** For complex process models it is recommended to split the process up into its main process steps and to represent them in separate PFD’s. Each PFD has to be clearly structured and arranged.

- **Nomenclature:** The nomenclature of the PFD, the units and the streams has to be coherent and self-explanatory. Each name must be unique. If OSMOSE is used afterwards it is recommended not to begin names with numbers and to avoid signs such as - and + (for example, \(F_Z\) _X_: material stream, \(q_X\): thermal stream, \(W_X\): mechanical stream, \(E_Y\): heat exchanger)

The description of a process in Vali is based on the following objects:
CHAPTER 5. SOFTWARES USED AT IPESE

PFD  A process flow diagram is a sketch of the whole process or of one step of it, that graphically represents the process UNITS and the interconnecting STREAMS.

UNIT  A unit represents one equipment of the installation. For each equipment category such as heat exchangers, reactors, etc. several types with different modeling capabilities are available. For example, a BATVAL unit models a reactor based on the atomic balances, while in a RKPVAL unit the chemical reactions have to be defined exactly.

STREAM A stream describes the connections between the units. Three types of streams are distinguished: material, thermal (heat flow) and mechanical (mechanical power transfer) streams. A material stream represents a stream of chemical compounds that is characterized by the pressure, temperature, enthalpy, flowrate, composition and phase (L, V, L/V, etc.). Each material stream has an associated THERMOD.

THERMOD  A thermodynamic method (Soave, Peng Robinson, etc.) that is used to compute its thermodynamic properties. For the chosen thermodynamic model the binary interaction parameters can be specified.

COMPOUND A compound is a chemical component, like $H_2$ or $CH_4$. A database is integrated in the software which contains the thermodynamic properties for the common compounds.

REACTION A chemical reaction is defined by the compounds involved in the reaction and their stoichiometric coefficients.

TAG  A tag defines the properties of measurements and is defined by a name, a value and an accuracy of a model variable (i.e. temperature T, pressure P, pressure drop DP, heat load, etc.). Each Tag is characterized by a default value, a measured value, a used value and a reconciled value. If the accuracy is CST: the value will remain constant (this constant is "Measured") and if it is OFF: the value will be computed by the software (Fig. 3.3).

Procedure to define a flowsheet

1. Open Belsim → Vali Modeller

2. **File** - **New** - insert process name - **Save**

3. Add Compound: select compound - **Load** - select 2nd compound - **Load** - etc. - **OK** (Fig. 3.3)

4. Define Thermol: enter name (for ex. Air) - select compounds (i.e. $N_2$ and $O_2$) - select method - **OK** (to add a compound, a thermol or a reaction: **Edit** - **Add...** or right mouse klick on the corresponding element in the legend at the left (Fig. 3.3))
5. Add Units: select unit - give name - choose mode of equations - select 2nd unit - etc.

6. Add Stream: select stream type - if material stream: select thermal - connect to units (out/in) - define the state: V, L, L/V - etc. (to add a unit or a stream: Edit - Add... or right mouse klick on the background in a PFD)

7. Specify Tags for units and streams (Fig. 5.3). The number of Tags to be specified is defined by the degree of freedom (section 11).

Practical Example: In the Help Manual it is explained step by step how to set up a model of a gas turbine (Chapter: How to build your data validation model, Section 13: A practical example)

5.1.2 Model Resolution

It is strongly recommended to construct and solve the model step by step in order to avoid initialization problems and non-convergence. Since Vali is based on an equation solver method resolving the system by the calculation of derivatives, a good initialization of the variables is required to ensure model convergence. An efficient way to initialize the model is to validate each PFD separately and to validate progressively the full model by activating the various PFD’s. In each PFD, the first unit of this process step is calculated first and the subsequent units are added progressively. A reasonable starting point must therefore be provided for the inlet process stream (i.e. composition, T, P).

To solve the model the state variables have to be identified and the equations have to be defined in order to define the degree of freedom DOF. The DOF is the difference between the number of variables \( n_v \) and the number of equations \( n_e \) and defines the number of variables that have to be specified to solve the problem.

\[
\text{DOF} = n_v - n_e
\]

The different equations that can be considered are: the mass balance for each chemical compound, the energy balance of the unit, phase equilibrium equations, performance equations and pressure drop equations. For each unit the MODE parameter defines the type of balances that is considered. The DOF is computed by the software on the basis of the analysis of the incidence matrix. The system can only be solved if the incidence matrix (lines=equations ; columns=variables) is invertible: square matrix and independent equations (see Error messages below). The choice of the state variables that are specified by a constant value (Tag value CST) is based on the constraints of the problem.

To make a run press the icon with the green triangle in the menu bar (Fig. 6.4), specify the runtime options (Fig. 6.3) by selecting the model: MAIN (or PFD or unit you want to calculate) and ticking generate HTML Report.

To resolve the model the ValiEngine goes through different steps: Tag reading, verification, analysis, resolution, report and output. If everything is well
Figure 5.1: To run the simulation: specification of runtime options

Set up the model is resolved successfully (Fig. 5.3) and the results are saved in the model file (Menu: Application - Results). The results of the last run can be loaded either by pressing directly **Reload** when the message window opens or through the icon (Fig. 5.4) on the menu bar or through Menu: Application - Load Results.

In some cases the validation does not succeed and no output can be produced because a fatal error occurred in one resolution phase and Vali stopped before starting the next step. The insight in the generated HTML Report allows in most of the cases to identify the cause of the error and to solve it. Here some common error messages and hints to solve these problems are given.

*ERF* 2529 Measurement ... cannot be equal to zero with a zero standard deviation. This error occurs when you request that a total flowrate should be zero. To avoid this you can set the stream OFF or fixe the flowrate at a small value (i.e. 10^-6).

*ERF* 2553 Undetermined recycle. This error can occur if there are closed loops. The Jacobian matrix is singular and it is impossible to calculate the recycle flowrate of one compound or the pressure. This error can be overcome by introducing an artificial blackbox unit CUTVAL which is a tear unit for loops which allows to break material or pressure loops, or by adding a purge and an inlet stream to the loop and to fix one of these flowrates to a small value.

*ERF* 2424 Not enough measurements to solve whole problem. If this error occurs, there are not enough measurements that are specified to satisfy the DOF. The analysis of the DOF gives the number of redundancy and hence the number of measurements that are missing (Fig. 5.3). To overcome this situation the easiest way is to add a new measurement from the list of Tags proposed in the HTML Report, or alternatively to add new equations.
*ERF* 2489 **Overspecifications.** This occurs if the number of constants is too high (i.e. higher than DOF) so that the number of equations in which they occur is bigger than the number of involved variables. To overcome this, constants have to be removed by setting Tags OFF in accordance with the propositions given in the HTML Report.

*ERF* 2540 **The Jacobian matrix is singular.** This error is the most difficult to be overcome. It is advised to run different parts of the model separately in order to identify the process section which causes the errors.

### 5.1.3 Tricks and Tips

**Menu Bar** The menu bar (Fig. 3.2) shows different items and icons for the most common actions. Most of them are intuitive but the following two are special.

**Edit Mode** The edit mode can be ON or OFF. To enter any graphical change the Edit mode must be activated (ON). In the OFF mode you can review the model and update the units and streams data but all operations on the structure of the model are disabled.

**Display Status** The Display Status field displays units and streams status and selects which tags values are displayed in the notebooks:

- **None:** Units and streams status is not displayed
- **Default:** Display Units and Streams status as set for the next run
- **Measured:** Display Units and Streams status as set after met files reading
- **Used:** Display Units and Streams status finally used by ValiEngine

Tags Data Set option corresponds to the set of tags values shown when you open a notebook (Default, Measured or Used). If you want to change a Tag the status Default has to be displayed.

**Model Coherence** After the model resolution it is recommended to check the coherence of the model by reviewing the inlets and outlets of streams of the models in the global balance (Report file) and by verifying that the constants are really considered as constants.

**Closed Loops** To initialize closed loops it is recommended to open the loop and to validate the units progressively. Once the model is initialized a CUTVAL unit (or a purge with an inlet stream with very small flowrates) has to be introduced before closing the loop to avoid convergence problems.

**PFD/Unit/Stream Status** The status of the stream can be set ON or OFF and the one of the PFD and Unit ON, OFF or OUT (right click: DATA - General). This option is very useful to initialize and validate the model progressively.

- **ON:** this is the default status and the unit is operated normally
- **OFF:** the unit does exist but is shut down so that nothing flows to or from it and consequently the connected streams are shut down.
• OUT: the PFD is considered as not existing at all. All the units and streams of a PFD that is OUT will not be part of the model during the next run.

Special Units In some cases the introduction of artificial units that does not represent real process equipments can be very useful for the modeling. Controller units allow for example to introduce user-equations.

$EQUVAL$: To set equality of composition between two streams. In the unit’s parameter Others choose the two streams whose composition will be identical and specify T, P and flowrate.

$OPXVAL$: To introduce simple user defined equations involving only 2 variables. In the unit’s parameters choose the control Tags, define the number of equations to be considered and define them with the actions ( +, -, /, etc.). This can for example be used to set a temperature difference between two streams.

More complex functions and equations can be defined through a Flex code in $FLEXLIB$ and $FLXVAL$ respectively.

$VARVAL$: To define additional variables. For example, to associate a variable (Tag) to a yield that is defined in the Flex code.

Detailed information about how to write Flex code functions are found in the Belsim Vali Help Manual.

Pressure Drop If a pressure drop is defined for a unit with several inlet streams, the reference stream for the DP has to be defined in the unit’s parameters. In the case where no pressure drop is considered for most of the units, the mathematical unit $OPTVAL$ can be introduced and in the properties one chooses pressure variables; 0:DP constant if not measured. For each unit one chooses then in the parameters: $MODEDP$: DP variable and the pressure drop is consequently zero, if not stated otherwise. This avoids to specify the pressure drop of each unit by a Tag.

$TAGS$ When working with $OMOSE$ and referring to a value of a variable calculated in Vali this one has to be defined by a Tag. Attention has to be paid to the units of the values (Fig. 13). For the energy integration for example, the temperatures have to be given in Kelvin [K] and the enthalpy ($HFLOWTOT$) in kiloWatt [kW].

5.2 Energy integration

IPESE has developed in ampl/glpk the code for solving the MILP problem of the Energy Integration. It is a computer aided process integration software which allows to optimize the utilities integration and the combined heat and power production based on the pinch analysis. General information about the different features and the syntax are found in the EnergyTechnologies documentation and in the IPESE wiki.
To run an energy integration of the modeled process the corresponding commands have to be called in the frontend (see: o.ComputeEI=1). Information about how to set up a matlab file (defining the hot and cold streams and the utilities, etc.) for the energy integration are found in the Energy Technologies documentation.

5.3 MOO

MOO is an advanced evolutionary algorithm implemented under Matlab and used to perform multi-objective optimizations. Some information are given in the paper [O] and on the IPESE wiki.

How to implement a multi-objective optimization in Osmose and how to interpret the results was already explained in previous chapters of this documentation: [MOO] and [MOO-1].

5.4 LCA

Life cycle assessment (LCA) is a methodology used to assess the environmental impacts of a product, a service or a system, taking into account its overall life cycle in relation with its function. These environmental impacts are expressed as a set of quantitative indicators, called impact categories, and address different types of environmental issues.

The LCA module, using a Matlab interface, is used in Osmose when it is required to have an indicator of the environmental performance of the system that is modeled. LCA is one of the several methods used for the estimation of environmental impacts. It is more complex than a simple evaluation of the on-site emissions, such as CO₂, NOx or SOx, but has several advantages over it, which makes it more appropriate for an environmental evaluation of an energy system:

**Off-site emissions:** One of the main advantages of using LCA in Osmose rather than an evaluation of the on-site emissions is that it can take into account all the off-site emissions that come from the raw material production, from the different transports and from the possible waste disposal. These off-site emissions may have a particularly high contribution to the environmental impact in the case of systems using renewable energy sources, such as biofuels for example.

**Global indicators:** Moreover, LCA is not only restricted to an estimation of single emissions, but allows for aggregating these different emissions in more global indicators that have an environmental significance. For example, the CO₂, CO, CH₄ and hydrocarbons emissions can be weighed and combined together into a more general indicator called GWP (Global Warming Potential), which expresses the potential of these cumulated emissions to have an adverse environmental effect on the global warming issue.
**Wide range of environmental issues:** A last advantage of using the LCA approach in Osmose over the simple evaluation of on-site emissions is that it does not only address problems concerning the emissions of pollutants in air or water, but also other important environmental issues such as the depletion of fossil and mineral resources or the land occupation.

LCA is submitted to ISO norms 14040 & 14044, and everyone conducting an LCA study should read and follow these norms throughout his/her work. LCA is performed in four main stages: the goal and scope definition, the life cycle inventory, the impact assessment and the interpretation. More information can be found on the LCA in general in [3]. In addition to these norms, some specific guidelines apply to the implementation of a LCA model in Osmose. Fig. 5.2 summarizes the four stages of the LCA and shows also how they have to be conducted to build a LCA model that can be used in Osmose. In dashed are the steps that do not change compared to a conventional LCA for building a LCA model in Osmose.

![Figure 5.2: Representation of the major steps to be conducted in an LCA](image)

The goal of this chapter is however not to explain the detailed methodology of a LCA. More information about this methodology can be found in [3]. More information about the syntax used in the Osmose framework, is available in the documentation of Energy Technologies.

The following paragraphs list the useful output coming from a LCA in the Osmose framework, and explain how to implement the LCA module and how to launch and specify a LCA computation.
5.4.1 Output of a LCA

LCIA (Life Cycle Impact Assessment) step yields a series of quantitative indicators that are a measure of different types of environmental impacts. When LCA is coupled to a model constructed in Osmose to evaluate the environmental performance, these indicators can be used for different purposes:

- as characteristics of the environmental performance of the process that is modeled
- as objectives in a MOO framework to calculate the trade-offs existing between the environmental impact and other objectives such as the costs

The results of an LCA are the following fields contained in the o.LCIA structure:

- .TotalImpact, which is a vector containing the values for the LCIA calculation of the different environmental performance indicators, or impact categories. Their number and the environmental issues that are covered depend on the chosen impact method.

- .SubstanceVector, which is a vector containing the cumulated values for the single substances that are involved as emissions or extractions in the LCA of the studied system.

- .ImpactContribution, which is a vector containing the contribution of all the flows taken into account in the LCA of the studied system to the different impacts categories.

The only results that are saved in new tags are the .TotalImpact that can be used afterwards in a MOO. Therefore, the elements of .SubstanceVector and of .ImpactContribution are just indicative values and can not be used in a MOO. Results of a LCIA calculation can be extracted and analyzed like any other result given by Osmose, as explained in section [X] and [Y]

If it is required in the frontend, some graphics showing the contribution of the LCI flows to the different impact categories are automatically generated for a OneRun. This is useful for results interpretation. Fig. [X] shows an example of such a graphic.
Figure 5.3: Example of an impact contribution graphic automatically generated

If it is required that the graphics are saved, they can be found in the folder run\xxxx/Osmose_Plots/LCIA.

5.4.2 Implementation of the ecoinvent database

To construct and run a LCA model within Osmose, it is necessary to use the ecoinvent database. It is a database containing life cycle inventory data for around 4'000 different unit processes, and also most of the recognized impact methods necessary to conduct the LCIA phase. The ecoinvent database can be downloaded from EPFL GIT (Contact your assistant to get access).

More information about ecoinvent can be found at [http://www.ecoinvent.org](http://www.ecoinvent.org).

5.4.3 Configuration in the frontend

List (1) shows how to call and configure a LCA computation in the frontend, for one example.

```matlab
% Specify if life cycle impact assessment calculation is done [required]
obj.LCIA.LCIA = 1; % 1:yes, 0:no
    % if yes, define the fields below.

obj.LCIA.EcoinventDirectory = fullfile('C:\models','ecoinvent');

obj.LCIA.ImpactMethod = {'impact02_end'};

obj.LCIA.IncludeEquipment = 1;

obj.LCIA.FunctionalUnit.Quantity = 10000; % default value, may be updated
obj.LCIA.FunctionalUnit.Unit = {'kWh'}; % unit in which is given the FU
obj.LCIA.FunctionalUnit.Description = {'Total electricity produced during lifecycle'};
```

32
The `.EcoinventDirectory` field is used to specify the path to where the ecoinvent database is stored.

The `.ImpactMethod` field is used to specify which impact method has to be used for the LCIA, i.e., the calculation of the environmental performance indicators. Several methods are implemented in Osmose, and they are listed in the Energy Technology documentation.

The `.IncludeProcessEquipment` field is used to specify if the process equipment has to be included in the LCIA calculation, 1 meaning yes and 0 meaning no. More details about the inclusion of process equipment in a LCA is available in the Energy Technologies documentation.

The `.FunctionalUnit` field and its subfields are used to specify and describe the functional unit that is used to bring back and compare every flow taken into account in the LCA.

The `.Graphics` field and its subfields are used to specify if graphics of contribution of the LCI flow to the different impact categories have to be plotted, what is the cut-off to apply and if the plots have to be saved.

### 5.5 Equipment cost

The equipment _cost_ database developed in the IPESE group regroups for different equipment types (pumps, compressors, heat exchangers, vessels, etc.) correlations to calculate the capital costs. The database is available via the EPFL GIT (Contact your assistant to get access).

The estimations are based on the general methodology outlined in [3, 4]. For each equipment the purchase costs $C_P$ are given, assuming atmospheric pressure and carbon steel construction, by a correlation of the type:

$$\log_{10} C_P = K_1 + K_2 \log_{10} A + K_3 (\log_{10} A)^2$$  \hspace{1cm} (5.1)

where $K_i$ are constants and $A$ is the characteristic size parameter (power for compressor, length/diameter for the reactors and heat transfer area for the heat exchangers).

The bare module costs ($C_{BM}$) representing the purchase costs adjusted by material ($F_M$) and pressure ($F_P$) factors taking into account the specific process pressures and materials, is given by:

$$C_{BM}^o = (B_1 + B_2 F_M F_P) IC_P$$  \hspace{1cm} (5.2)

where $I$ is the actualization factor expressed by the ratio of the Marshall and Swift Equipment Cost Index at actual time to the cost data’s reference time.

The total grassroots costs defining the total investment costs for a new production site are then calculated from the bare module cost by using further
multiplication factors to take into account indirect expenses like labor, transportation, fees, contingencies and auxiliary facilities.

\[ C_{GR} = (1 + \alpha_1) \sum_{i=1}^{n} C_{BM,i} + \alpha_2 \sum_{i=1}^{n} C_{BM,n,i} \] (5.3)

where \( C_{BM,n,i} \) represents the bare module costs of the \( i^{th} \) equipment for the base case conditions (i.e. atmospheric pressure and carbon steel material) and \( C_{BM,i} \) the costs at the operating conditions. The two factors represent additional costs related to the construction of the plant being dependent (\( \alpha_1 \)) or independent (\( \alpha_2 \)) of the process conditions. The numeric values that are used are: \( \alpha_1 = 0.18 \) (contingencies 0.15 and fees 0.03) and \( \alpha_2 = 0.35 \) (auxiliary facilities, site development and buildings).

The function \( c = \text{cost}_\text{defaults}(c) \) fills the cost constants structure \( c \) with defaults or appropriate values of the Marshall Swift Year Index, the factor for the Total Module and the factor for the Grass Roots cost calculation (related to \( \alpha_1 \) and \( \alpha_2 \)) (list \( \text{BM}_3 \)).

All the equipments functions are developed based on the same principle:

\[ \text{Cost}=\text{cost}_\text{equipment}(c, \text{pressure}, \text{material},...) \]

The function’s input includes the cost constants structure \( c \) and all the parameters involved in the purchase cost estimation (i.e. pressure, construction material, flowrate, etc.) (list \( \text{BM}_3 \)). The output is a structure formed of the Bare Module cost of the equipment (\( \text{Cost.BM} \)), the Total Module cost of the equipment(\( \text{Cost.TM} \)), the specific Grass Root cost associated to the equipment(\( \text{Cost.GR} \)) and the currency of the costs (‘USD’, ‘EUR’, ‘CHF’) (\( \text{Cost.Unit} \)) (list \( \text{BM}_3 \)).

```matlab
function Cost = cost_Cyclones(c, volumeflow, material)
% UNIT: Cyclones
%
%
% INPUT: c : cost constants structure
% c.Year_Index_MS Marshall and Swift index for the current year
% c.f_TotalModule Total module factor, 'alpha1'
% c.f_GrassRoot Grass root factor, 'alpha2'
% volume flow rate [m3/s] 0.0005:0.08
%
% OUTPUT: Cost.BM : Bare Module cost of equipment
% Cost.TM : Total Module cost of equipment
% Cost.GR : specific Grass Root cost associated to the equipment
% Cost.Unit : Currency of costs {'USD'}, {'EUR'}, {'CHF'}
%
% CURRENCY: USD
```
Listing 5.2: Matlab function to calculate the costs of a cyclone

```
% % YEAR_INDEX: reference year: mid-1982
% Chemical Engineering Plant Cost Index: 315
% Marshall and Swift Equipment Cost Index: 774
%
% % AUTHOR: M. Gassner
% Last modification: F. Guinet 19.02.08
%-----------------------------------------------
% % Constants
Reference_Year_Index_MS = 774; % Marshall and Swift index for the
  reference year 1982
k1=4.2419;
k2=0.5056;
k3=0;
if strcmp(material,'CS')
  f_BM = 3; % Carbon steel
else if strcmp(material,'SS')
  f_BM = 4; % Stainless steel
else if strcmp(material,'Ni')
  f_BM = 6; % Nickel alloy
end
f_actualisation = (c.Year_Index_MS/Reference_Year_Index_MS);
Currency = {'USD'};
% % Range validation
if volumeflow < 0.0005
  warning('Volumetric flowrate into Cyclone is too low!');
end
if volumeflow > 0.08
  warning('Volumetric flowrate into Cyclone is too high!');
end
% % Cost calculation
Purchase_Cost_1982 =10^((k1 + k2*log10(volumeflow) + k3*(log10(volumeflow))
  )^2);
Purchase_Cost = Purchase_Cost_1982*f_actualisation;
BareModule_Cost = f_BM*Purchase_Cost;
% % Bare Module cost
Cost.BM = BareModule_Cost;
% % Total module cost
Cost.TM = Cost.BM * c.f_TotalModule;
% % Grass root cost
Cost.GR = Cost.TM + c.f_GrassRoot * Cost.BM;
% % Currency
Cost.Unit = Currency;
```
>> Cost = cost_Cyclones(c, 0.05, 'CS')
Cost =

BM: 2.0811e+004
TM: 2.4557e+004
GR: 3.1841e+004
Unit: {'USD'}

Listing 5.3: Matlab commands and results for the cost estimation of a cyclone

5.6 Energy Technologies

Energy Technologies is a markup language intended to describe models developed in the domain of energy conversion systems. OSMOSE uses Energy Technologies to connect models together and in general to query models by sending input parameters and recovering results. The previous chapters giving information about Osmose have shown how to query a model and how to perform complex computations, like optimization. For detailed information see the documentation on Energy Technologies and the IPESE wiki.

5.7 git

Git is a version control system allowing you to:

- Keep track of every change in every file of your project
- Go back to a previous version of a given file
- Work with other people on the same set of files

For more information about git features and installation see [http://www.git-scm.com/](http://www.git-scm.com/)

EPFL offers a git hosting service [https://git.epfl.ch/polyrepo/](https://git.epfl.ch/polyrepo/) IPESE uses git repositories to share the latest version of the different codes (osmose, Energy Technology, Equipment Cost and LCIA). Git should be used by students **exclusively** to download and update with the latest version of the files required to launch Osmose. To get access to the repositories on git you have to contact your assistant. For the lectures given by Prof. Maréchal no access to git is required all the required data is available on the course moodle site.
Chapter 6

Additional Information

A list of the places where additional information can be found and of the different documents that were referred to in this documentation is given here.


- IPESE website - Resources: http://ipese.epfl.ch
  - Osmose: Main_OsmoseDoc.pdf
  - EnergyTechnologies: EnergyTechnologies_doc.pdf
Bibliography


Appendix A

Results extraction

In this chapter a brief explanation about the OSMOSE results (for more details see Main_OsmoseDoc.pdf) is given, as well as some useful tips to extract the results for results analysis.

A.1 o. Structure

The Matlab structured variable o is used to perform the communication within OSMOSE and all the information are stored in o within different fields.

The o variable can be loaded from a file saved on the disk by calling the function: o=osmose_loadVar('o.mat'). Once it is loaded the different information can be assessed.

o.Model: This field contains all the information defining the models.

o.EI: This field contains all the information of the energy integration.

A.1.1 Tag Extraction

All the information about a Tag of a one run computation can be extracted by the following commands (List.

```
o.Model.Tags: gives a structured array (1x number of Tags) containing the following fields: ModelTagName, TagName, DisplayName, Unit, Value, Status, isGlobal, ExpectedDeviation, Measurement, PosterioriAccuracy.

o.Model.Tags.TagName: gives a list of all the Tags defined in the model.
```

```matlab
>> o.Model.Tags(5) % extracts all the information of a Tag based on its position (here: 5)
ans =
```
APPENDIX A. RESULTS EXTRACTION

ModelTagName: {'HYDR0'}
TagName: {'PG_1_2_T'}
DisplayName: {'WGS reaction inlet temperature'}
Unit: {'K'}
Value: 479
Status: {'OFF'}
isGlobal: 0
ExpectedDeviation: 0
Measurement: 479
PosteriorAccuracy: 0

>> o.Model.Tags(5).Value  % extracts the numerical value of this Tag
ans =
479

>> find(strcmp('natgas_spe2',o.Model.Tags.TagName))  %To identify the
       position of a Tag having a given name (here: natgas\_spe2)
ans =
682

Listing A.1: Commands for tag extraction

In the case where several points are calculated during the computation (sensitivity analysis, optimization) the o.Results structured field contains for each point all the information about the model, the constants, the variables or the objectives. The tags, objectives and variables can be extracted in a similar way as for one point (List. A.2).

>> o.Results(2);  %allows to assess all the information of the point
numbter 2
>> o.Results(2).Model  %gives for point 2 all the information about the
Tags
ans =
TagName: {'HYDR0'}
Period: [1x1 struct]
Tags: [1x682 struct]
TagsList: {1x682 cell}
Variables: [1x1 struct]

>> o.Results(2).Model.Variables  %gives for point 2 information about the
variable used in Sensi
ans =

ModelTagName: {'HYDR0'}
TagName: {'ST_EXP_P'}
Display Name: {'Saturation pressure of evaporation'}  
Unit: {'bar'}  
Lower Bound: 30  
Upper Bound: 50  
Number of Steps: 10  
Value: 32.2222  
is Global: 0

```matlab
>> find(strcmp('natgas_spe2', [o.Results(2).Model.Tags.TagName]))
an = 680
```

```matlab
>> o.Results(2).Model.Tags(680)
an =
```
```matlab
    Model Tag Name: []
    Tag Name: {'natgas_spe2'}
    Display Name: []
    Unit: []
    Value: 0.5876
    Status: {'OFF'}
```

Listing A.2: Commands to extract a tag of the results structure of a sensitivity analysis

An array containing the value of one Tag, Objective or Variable for each point of the sensitivity analysis or optimization can be obtained by the following functions:
```matlab
    ommose_results_getTag(o, 'Tag Name', 'Model Tag Name'),
    ommose_results_getObjective(o, 'Tag Name', 'Model Tag Name'),
    ommose_results_getVariable(o, 'Tag Name', 'Model Tag Name').
```

For example:
```matlab
To get the Tag information:
    yTag = ommose_results_getTag(o, 'natgas_spe2', 'Hydro')
```
```matlab
To extract the numerical values of the Tag:
    y.Value = yTag.Value
```

Other useful functions to access the results data are:
```matlab
    ommose_results_getAllHeaders(o): loads all headers (Objectives, Variables, Tags, Clusters, CCs, Models) from results at once and stores them in "headers" structure.
    ommose_results_getAllTags(o): retrieving all Tags from results. Similar functions allow to retrieve all the objectives or variables.
```

The extraction of several Tags at a time can be simplified by writing a loop in a Matlab script.
**Tags in postcomputation** To use tags in the postcomputation, there are several possibilities.

All the model tags can be retrieved with the function `ET_generateVariablesFromTags` and be evaluated. technology = o.Model(o.ModelID); s = ET_generateVariablesFromTags(technology.Tags);
eval([s]);

Alternatively values of model tags, energy integration or LCA tags can also be retrieved by the `osmose_getTag` function:

`osmose_getTag(o,'@Model:xModelName.xyTagName','Value');` where `xy-TagName` is a tag of the model `xModelName`

`osmose_getTag(o,'@EI:zkTagName','Value');` where `zkTagName` is a tag of the energy integration

Example: `tref = osmose_getTag(o,'@Model:wood_gas.t_ref','Value');`

`p_setl = osmose_getTag(o,'@EI:power_MULTI','Value');`

**Energy integration tags** Extract tags of heat exchange area and minimum number of heat exchangers:

`osmose_getTag(o,'@EI:DefaultHeatCascade_HENArea_Bath')`

`osmose_getTag(o,'@EI:DefaultHeatCascade_NMinMER')`

### A.1.2 Plot Tags

OsmosePlots is a Graphical User Interface (gui) written in Matlab that uses the information retrieved from the model tags to generate graphics. After a sensitivity analysis or a recombination it is automatically called by OSMOSE, however it can also be called in Matlab (you have to be in the folder containing the o.mat file !) by the command:

`osmose_gui_plot(o)`

Through this tool each Tag of the model can be plotted against another (Fig. 3.10). This allows hence to show for example, how one objective (or Tag) evolves in the range of one variable.

Alternatively the plots can also be constructed through the conventional Matlab functions by extracting first the values of the Tags used for the x and y axis:

`xTag=osmose_results_getTag(o, 'TagName', 'ModelTagName')`
`x.Value = xTag.Value`

`yTag=osmose_results_getTag(o, 'TagName', 'ModelTagName')`
`y.Value = yTag.Value`

### A.2 Composite curves

The information of the Energy Integration are stored in the field `o.EI`.

For the cases where several points are calculated during the computation, the composite curves can be retrieved from the results with the command:

`osmose_results_getCC(o,ccName)`
where ccName refers to the different types of curves:

- cc : Composite Curve representing the hot and cold streams (temperature - enthalpy plot)
- cfc : Carnot Composite Curve representing the hot and cold streams \((1 - \frac{T_u}{T_g})\) - enthalpy plot
- ecc.com : Hot and Cold Composite Curves with real and corrected temperature
  - egcc.com : Grand Composite Curve with real and corrected temperature
  - gcc : Grand Composite Curve
  - power.uco : Integrated Composite Curve of Unit power
  - ucc : Utility integrated Composite Curve
  - utility.uco : Integrated Composite Curve of Unit utility
- xcc.com : Hot and Cold exergy Composite Curves with real and corrected temperatures \((1 - \frac{T_u}{T_g})\) - enthalpy plot
- xgcc : Carnot Grand Composite Curve \((1 - \frac{T_u}{T_g})\) - enthalpy plot
- xgcc.com : Exergy Grand Composite Curve with real and corrected temperatures \((1 - \frac{T_u}{T_g})\) - enthalpy plot
- xucc : Carnot utility integrated Composite Curve \((1 - \frac{T_u}{T_g})\) - enthalpy plot

This command generates a structured array (1x number of points) with the following fields: `TagName`, `graph_title`, `legend_labels`, `source`, `x_label`, `y_label`, `x`, `y` (Fig. [W]). To extract the information of the selected composite curve (here: GCC) for one specific point the number of the respective point has to be indicated (here: 1).

The composite curve can then be plotted by calling the command `cemose_plotCompositeCurve(o, curve)`. The plot’s aspect can be changed (scale axis, etc.) by editing the figure properties.

The composite curves of two different points can be superposed by constructing a new composite curve whose x and y data array is formed by the corresponding values of the two points (List. [W]). Alternatively, the x and y data of both points can be extracted and the plot can be drawn with the `plot` and `hold on` Matlab commands. Once the values used for the x and y axis are extracted it is also possible to modify them (as seen hereafter) in order to shift the curves.

```matlab
>> cc=cemose_results_getCC(o,'gcc')

cc =

1x10 struct array with fields:
  TagName
graph_title
legend_labels
source
x_label
y_label
x
```

43
y

>> cci=cc(1) %GCC information of point 1
cci =

    TagName: {'gcc'}
    graph_title: {'Grand Composite Curve'}
    legend_labels: {'All streams'}
    source: [1x1 struct]
    x_label: {'Heat Load [kW]'}
    y_label: {'Temperature [K]'}
    x: [1x1 struct]
    y: [1x1 struct]

>> cc2=cc(5); %GCC information of point 2
>> cc3=cc2;
>> cc3.x=[cci.x,cc2.x]; %Superposition of two curves: x values
>> cc3.y=[cci.y,cc2.y]; %Superposition of two curves: y values
>> osmose_plotCompositeCurve(0,cc3)

Listing A.3: Matlab commands to extract information and plot composite curves

For the one run computation, the information of the different composite curves is stored in the field o.EI.CC. One particular composite curve can be assessed by giving the corresponding number (here: 8). The list and fig. show an example of how the curves can be modified by extracting the x and y data and performing some operations on them. This can for example be used to shift the curves so that the pinch point is found at x=0, by adding or subtracting a certain number from the x values.

>> cc=o.EI.CC(8)
cc =

    TagName: {'power.uco'}
    graph_title: {'Integrated composite Curve of Unit power'}
    legend_labels: {'Others' 'power' 'Mech. Power'}
    source: [1x1 struct]
    x_label: {'Heat Load [kW]'}
    y_label: {'Temperature [K]'}
    x: [1x4 struct]
    y: [1x4 struct]

>> cc.x
ans =

  1x4 struct array with fields:
    data

>> cc.x(1)
%APPENDIX A. RESULTS EXTRACTION%

\texttt{ans =}
\begin{verbatim}
data: [672x1 double]
\end{verbatim}

\begin{verbatim}
>> osmose_plotCompositeCurve(o,cc);
>> hold on
>> plot(cc.x(1).data+16155.6, cc.y(1).data);
>> fh=figure(13);
>> saveas(fh,'plotmodified','epsc')
\end{verbatim}

Listing A.4: Matlab commands to modify composite curves

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figureA1.png}
\caption{Modification of original composite curve}
\end{figure}

The Pictures Format is stored in \texttt{o.PicturesFormat} and can be set in the frontend file:

\begin{verbatim}
o.PicturesFormat={‘png’, ‘fig’,’ai’,…}
\end{verbatim}

The aspect of the pictures (front size, line color etc.) can be modified in Matlab through the figure properties or in Adobe Illustrator.
Appendix B

VALI

Figure B.1: To add a compound
Figure B.2: Vali modeller interface

Figure B.3: Tags definition
Figure B.4: Display if model is resolved successfully

Figure B.5: Error occurred during the resolution of the model