



USER MANUAL

VERSION 2.0

NOVEMBER 22, 2010

WRITTEN BY FRANCESCA PALAZZI

WITH THE COLLABORATION OF

ZOÉ PÉRIN-LEVASSEUR, RAFFAELE BOLLIGER, MARTIN GASSNER



Copyright ©2003-2007 by Laboratoire d’Energétique Industrielle (LENI)

All right reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of LENI.

For information about permission for use of material from this document as well as *OSMOSE* program, please contact LENI:

Laboratoire d’Energétique Industrielle
ME A2 434 (Bâtiment ME)
Station 9
CH-1015 Lausanne
Tl.: +41 21 693 35 06
Fax: +41 21 693 73 22
🌐 <http://leni.epfl.ch>
✉ secretariat.leni@epfl.ch

This software has been initiated and developed by:
Francesca Palazzi Francesca.Palazzi@a3.epfl.ch
Raffaele Bolliger Raffaele.Bolliger@epfl.ch

The following people have contributed to *OSMOSE* developement and documentation:

Julien Godat	2001-2002	First code allowing to run Vali and Easy from Matlab
Francesca Palazzi	2003-2007	Developer
Raffaele Bolliger	2003-	Developer
Andrea Fabiano	2003-2004	Developer and beta-tester
Martin Gassner	2004-	Developer and beta-tester
Irene Ricart-Puig	2004-2005	Creator of the <i>OSMOSE</i> report generator
Zoé Perin-Levasseur	2005-	<i>OSMOSE</i> documentation and beta-tester
Damien Muller	2005-2007	<i>OSMOSE</i> documentation and beta-tester
Luc Girardin	2005-	<i>OSMOSE</i> web interface and web service
Nordahl Autissier	2005-	<i>OSMOSE</i> beta-tester
Nicolas Borboën	2006-2007	Subversion maintainer
Isabelle Juchli	2006-2007	Improvement of the report generator
Hubert Thieriot	2007-	Contributions in development of EI structure and related software support

The *OSMOSE* project is made possible thanks to the financial efforts of the *Laboratoire d'Énergétique Industrielle* (LENI) in the *École Polytechnique Fédérale de Lausanne*. Many thanks to:

Professor Daniel Favrat	Director of LENI
François Maréchal	First Assistant at LENI
FIFO	Fond d'Innovation pour la Formation, EPFL
OFEN	Office Fédéral de l'Énergie

Contents

1	Documentation update	1
2	Overview	2
2.1	What does this document contain?	2
2.2	Who should use osmose?	2
2.3	What is osmose?	3
3	Requirements and Installation	4
3.1	Step by step installation	4
3.2	Requirements	5
4	Running osmose step-by-step	7
4.1	Schematic overview	7
4.2	Modeling ABC	8
4.3	Osmose models general structure	10
4.4	The Tags Structure	10
4.5	Establishing a model	12
4.5.1	Pre-computation function	13
4.5.2	External Software Model	13
4.5.3	Intermediate function	15
4.5.4	Energy Integration Model	15
4.5.5	Post-computation function	15
4.6	Choosing the computation to perform	15
4.6.1	Model snapshot	16
4.6.2	Sensitivity analysis	16
4.6.3	Optimization	18
4.6.4	Recomputation	19
4.7	Analysing data	19
4.8	Frontend usage	19
5	Models Definition	21
5.1	General model definition	21
5.2	Vali model definition	22
5.3	Aspen model definition	22
5.4	<i>AMPL</i> model definition	24
5.5	Energy integration model definition	24
5.5.1	Tags definition	25
6	Computation definition	28
6.1	Computation selection	28
6.2	Software location	29
6.3	One run definition	30
6.4	Sensitivity analysis definition	30

6.5	Multi-objective optimisation	31
6.5.1	Restarting a failed optimization run	33
6.6	Recomputing a Pareto curve	33
7	Multi-period problem definition	36
7.1	Constants definition	36
7.2	Variables definition	36
7.3	Post-multiperiod computation	37
8	Results Analysis	38
8.1	Results structure	38
8.2	Reporting	38
8.2.1	Configuring the frontend	38
8.2.2	Reporting results	39
8.3	Pareto plots using OsmosePlots	39
8.3.1	Osmose data storage	39
8.3.2	Calling OsmosePlots	40
8.3.3	Using OsmosePlots	40
.1	Functions examples	45
.1.1	PreComputation example	45
.2	Aspenmodel	45

Chapter 1

Documentation update

Keep the update of the osmose documentation for users is under the responsibility of everyone. As soon as somebody made changes or added new features for the main documentation, this has to be documented in the following way:

- Main documentation is to be written in the LaTeX files that are available on svn at the following address: <https://lenisvn.epfl.ch/svn/osmose/doc>. Please use separate files and input them then in the main files.
- Then, the LaTeX document will be automatically converted in the LeniWiki. Please do not write directly new main documentation on the LeniWiki!

Only authorized people who have an access to svn can make modifications in the documentation. If you do not have the access and that you would like to make a modification, please contact: leda.gerber@epfl.ch.

For the FAQ and the quicktips, which are not part of the main documentation, this will be written on the LeniWiki. The basic idea is that if everyone discovers or creates a new function that can be useful to the others, it has to be documented in the quicktips. Same for the FAQ, even for the very basic questions.

Chapter 2

Overview

2.1 What does this document contain?

The present document is a complete user guide to *OSMOSE*. In addition, elements of software structure and architecture are exposed. This makes therefore the document useful as an introduction for future developers in the project.

Chapter 1 explains the procedure to follow for the update of the documentation.

After the present overview, the installation procedure is given in chapter 3, along with technical specifications.

Chapter ?? introduces the basic mathematical and engineering concepts used in the frame of *OSMOSE*. The reader not familiar with design and optimization methodologies will find a summary of the important aspects and a list of references for further reading.

Chapter 4 introduces osmose usage in a step-by-step approach.

Chapter 5 is an extensive guide to *OSMOSE* model setting.

Osmose is strongly linked to the use of the Energy Technologies database. For more information on Energy Technologies, have a look at the documentation: <https://lenisvn.epfl.ch/svn/EnergyTechnologies/trunk/doc>.

2.2 Who should use osmose?

OSMOSE is designed for researchers and engineers that deal with complex technology models and want to extract the more out of them. *OSMOSE* is also conceived as a way to unify modeling philosophies within a group and to permit storage and documentation of the work.

The *OSMOSE* platform contributes to these goals with the following aspects:

- It allows to establish communication between models built with different software
- It allows to run complex computations, like optimization or sensitivity analysis
- It is able to build automatic reports based on computation results
- It is a collection of analysis tools

The user of *OSMOSE* should be familiar with design problematics and master optimization techniques and purposes. Basic knowledge of Matlab programming is required. *OSMOSE* handles external models written in *Vali* and *AMPL*, languages that shall therefore be familiar to the user.

2.3 What is osmose?

OSMOSE is a platform for the study and design of complex integrated energy systems. The software is developed in the Laboratory for Industrial Energy Systems (LENI) in the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. The project is motivated by the need of a flexible and performant research tool to study and design energy systems.

In this perspective, the general scope of the software is to help the user to develop and compute technology models that combine (a) thermodynamic computations, (b) power and energy integration as well as (c) economic or environomic aspects.

OSMOSE, exploits the models by performing (a) sensitivity analysis, (b) optimization and (c) data analysis.

The seminal publications [3] and [8]¹ present the underlying approach, based on process simulation and modeling, associated with energy integration.

¹see also [11], [7], [12]

Chapter 3

Requirements and Installation

OSMOSE runs both on Windows and Linux platforms.

3.1 Step by step installation

1. Install SVN.

For windows:

Subversion is an open source version control system. Download from <http://tortoisesvn.tigris.org/> for windows. For linux there is no need to install a software. Further documentation can be found at http://leni.epfl.ch/images/procedures/svn/doc_svn_leni.pdf.

For linux:

Use the command:

```
sudo apt-get install subversion
```

2. Get *OSMOSE* package. The software is constituted by a set of matlab functions organized in several sub-folders¹.

For windows :

- (a) Create a folder *osmose* for example on C (notice that the path to *osmose* folders must not contain any space):
- (b) right mouse click, SVN Checkout
- (c) URL of repository : <https://lenisvn.epfl.ch/svn/osmose/trunk> don't forget the "s" at the end of http !

For linux :

Create a folder wherever you want (traditionally in `/usr/local`)

```
sudo mkdir /usr/local/osmose
```

Import *osmose* from SVN:

```
cd /usr/local/osmose
svn co https://lenisvn.epfl.ch/svn/osmose/trunk
```

3. Install required software. The required software is detailed in section 3.2.

¹For users at LENI, the preferred installation is through the svn server. Contact system administrator and request access.

4. **Set your matlab path.** The *OSMOSE* folder with all its sub-folders have to be part of Matlab's search path. To add *OSMOSE* to the search path use either the `addpath` command² or in the Matlab window select **File > Set Path** to open the Set Path dialog box. Save the changes to the path for successive sessions either with the `savepath` command or trough the Set Path dialog box.
5. **Control path definition to external software.** *OSMOSE* can call the software listed in section 3 by locating their main executable file. Control or define your paths as explained in section 6.2

Once these steps are completed, you are ready to begin experiencing *OSMOSE* . Follow the guide in chapter 4!

3.2 Requirements

The modeling tools used with osmose can vary as well as the operating systems, thus the requirements may vary for every situation. You should control that the software specified in the following list is installed on your machine:

- *Matlab*, version 7.6 or above
- *Vali* [2], in case of Vali model usage
Belsim Vali is only supported by Windows
 1. From the server <http://documents.epfl.ch/groups/v/va/vali/private/VALI4500/> Install Vali client (VALI/Vali4Client/Setup.exe)
 2. Replace localhost par 128.178.144.127
 3. Ask somebody from LENI to add an account on the license manager with the same username than for the session windows
- *Aspen* [13], in case of Aspen Plus model usage
Aspen Plus is only supported by Windows.
 1. EPFL licenses for Aspen Plus are hold by the Chemistry and Chemical engineering section
 2. To get the installation CD, ask someone from LENI for the contact of the person in charge at ISIC
- *Easy*, if energy integration is performed in the model with Easy
On windows :
 1. Install Cygwin <http://www.cygwin.com> Choose Base and GnuPlot (also ssh if you want to use pleiades2)
 2. Prepare the environment :
The goals here is to set environment variables that are used by easy. On windows systems be sure you are using a "smart" text editor. Wordpad and Notepad are not! In case of doubts, just install Notepad++ .

Edit C:\cygwin\etc\profile file.

Add following lines at the end of the file:

```
## Easy environment variables
export SCRLIBTERM=tel
export EASY_TDF=/usr/local/zero/dat/tdf
export EASY_MSG=/usr/local/zero/dat/belsim
export EASY_SYNEP=/usr/local/zero/dat/synep
export PATH=$PATH:/usr/local/zero/bin
```

²To add osmose folder and all the subfolders, combine `addpath` with `genpath` in the following way:
`addpath(genpath(YourOsmosePath))`.

3. Open folder "C:/cygwin/usr/local".

On Windows, restart cygwin. If some silly messages appear in the terminal, something went wrong with the text editor used to edit "profile". To solve the problem, type the following command, which will convert windows end of lines to unix end of lines. Then start cygwin again.

```
/usr/bin/dos2unix /etc/profile
```

Enter EASY2 in the cygwin window to test your installation

On linux :

1. Install GnuPlot

```
sudo apt-get install gnuplot
```

2. Depending if you are using a 32 or 64 bit version:

```
svn co https://lenisvn.epfl.ch/svn/easy/linux/x86_32/trunk/ /usr/local/zero
```

Or:

```
svn co https://lenisvn.epfl.ch/svn/easy/linux/x86_64/trunk/ /usr/local/zero
```

3. Testing:

```
source /etc/profile
easy2
```

- *AMPL*, in case of *AMPL* model usage or if energy integration is performed with the software *Eiample*. Control also the availability of the solver you want to use with the *AMPL* model. A student version is available under <http://www.ampl.com/>. LENI purchased recently licenses, but the files are not tested for the moment.
- *GLPK*, for energy integration: GLPK is an open source software for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It can be downloaded under <http://www.gnu.org/software/glpk/>. In the near future eiample (using glpk) will replace easy for energy integration problems.
- *moo*, for multi-objective optimization. *moo* is also a set of matlab functions: SVN Checkout from <http://lenisvn.epfl.ch/svn/moo> ³

For the installation of the above, consult the specific software documentation. Under Microsoft Windows platforms, *Easy* and *gnuplot* can only run with the linux emulator Cygwin ⁴. *Gnuplot* is a Linux packages distributed with cygwin. The installation of *gnuplot* has to be done when installing Cygwin by selecting the *gnuplot* package when running Cygwin setup.

³For LENI users, get it through the svn server.

⁴<http://www.cygwin.com>

Chapter 4

Running osmose step-by-step

After a general overview of *OSMOSE* organization (Section 4.1), this chapter introduces the main conceptual steps that compose *OSMOSE* usage. Namely (a) model definition (Section 4.5), (b) possible computations (Section 4.6) and (c) data analysis (Section 4.7).

You will get used to interact with *OSMOSE* through a matlab function called the *front-end*. Section 4.8 introduces front-end usage.

The implementation of your problem following each of the steps above is fully developed in Chapters 5, 6 and ??.

4.1 Schematic overview

The art of modeling refers to the ability of translating phenomena occurring in the real world into mathematical language. Modeling is an essential tool of both scientists and engineers as it allows to understand phenomenas and to make predictions upon the real (see Section 4.2).

OSMOSE users focus on engineering models developed for designing complex energy systems, such as production processes, heating networks and power plants. These models are generally characterized by the fact that they include *phenomenological models*, such as thermodynamical descriptions or chemical reactions models, as well as *engineering models*, such as dimensioning and costing. Moreover, in these models the treatment of heat exchanges has a distinctive role.

OSMOSE is tailored to, in a first step, establish such models rapidly (see Section 4.5) and, in a second step, exploit the model by performing computations such as sensitivity analyses (see Section 4.6.2) or optimization (see Section 4.6.3).

OSMOSE has three levels of functionalities, as shown in fig. 4.1:

Where *OSMOSE*₁ (fig.4.2) 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.

*OSMOSE*₂ (fig.4.3) 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.

*OSMOSE*₃ (fig.4.4) represents the data extraction and treatment from the computation results. The way to extract and interpret results will be discussed.

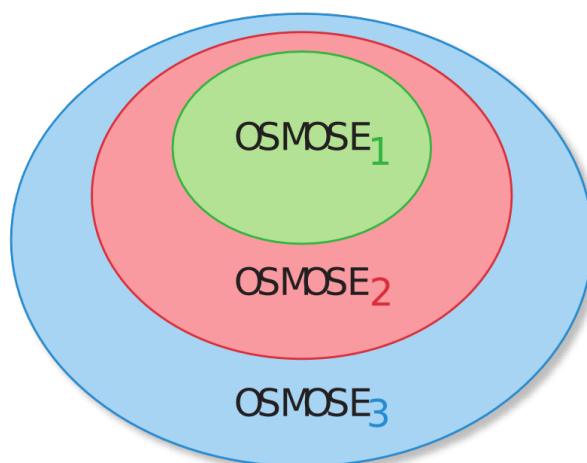
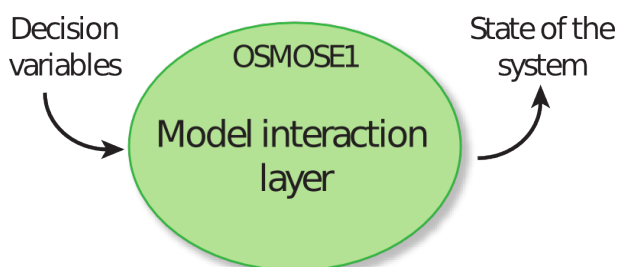
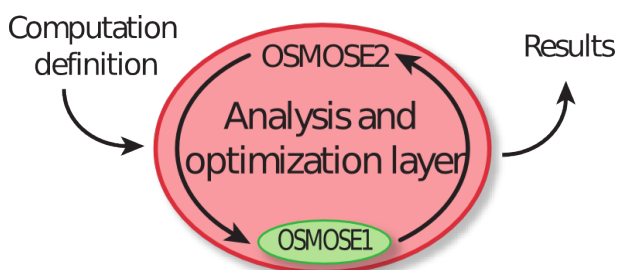


Figure 4.1: OSMOSE onion layer scheme

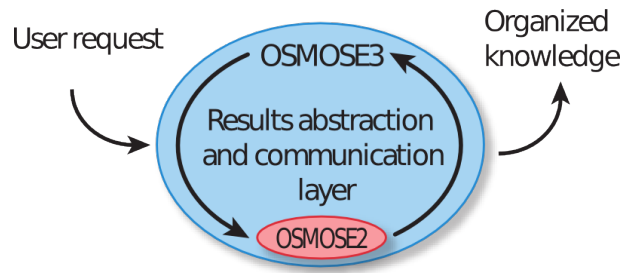
Figure 4.2: OSMOSE₁Figure 4.3: OSMOSE₂

4.2 Modeling ABC

Select the system

As said, a model is the mathematical translation of phenomena of the real world. More precisely, models allow to describe and make predictions on how things happen. Usually the goal is not to establish a precise description of the whole universe. Models restrict therefore modestly the description to what is called a *system*.

The first step to establishing a model is thus to carefully select the system to be described. To fix the ideas, here is a random list of interesting systems: an atom, an engine, a chocolate factory, a parking lot in Upper Manhattan, the solar system, a district heating in Brno, the blood vessels of an trout, a heat

Figure 4.4: OSMOSE₃

exchanger. You will recognize with this list that there are systems that are more easily described than others...

Variables

The second step to the description of a system is to decide what aspects of it are of interest. This is done by selecting the *variables* of the system. Consider the example of the parking lot in Upper Manhattan. To be able to run the parking smoothly, the tenant is interested in knowing the occupation at each moment of the day, the frequency of access and the size of the cars. But he will not be interested in knowing the brand and color of each car nor how many people occupy each car. Therefore the variables used in a model established by the tenant will be: time, occupation as a function of time, frequency as a function of time and size of cars as a function of occupation. The landlord of the same parking lot might be interested in other aspects, for example the heat generated by the cars, the air pollution and the road usury. For the same system, the landlord could start with the tenant model and add complexity using more variables such as: average heat on each floor, air pollution distribution and average road usury in each division of the lot.

Parameters

There are aspects of a system that do not change but that influence the system behavior. These are referred to as *parameters*. Some people prefer to consider parameters as variables taking a constant value. We make however the distinction between parameters and variables as it can be useful, particularly when the parameters take uncertain values, this is however behind the scope of the current manual.

State of a system

Using the variables, one gains the ability to describe what is happening in the system. Each possible situation is referred to as a possible *state of the system*. When the value of all the variables is known one says that the *state of the system* is known. As an example, the state of the tenant's parking lot at 1 p.m on Wednesday could be: occupation 256 places taken over 500, one car arriving every 5 minutes, one car leaving every 10 minutes, 70% berlines 30% SUV.

Model equations

Now the interest of building a model is to try to capture the *relationship* between variables so as to be able to draw predictions. The case of the parking lot model is a little tricky as the relationship between variables is dictated by empirical laws that reflect the behavior of human customers. Luckily, in the coarse world of engineers, the relationship between variables is often given by the laws of physics. As an example, the amount of heat exchanged through a heat exchanger can be computed knowing the temperatures and heat capacities of the exchanging streams.

The formalization of the above is that a model is defined by a set of equations capturing the relationship between variables. One can divide the variables into two categories: the degrees of freedom x ,

or *decision variables*, are determined by the user; the unknowns z , or *dependent variables*, are computed by solving the model equations once the value of x is fixed. Add to these the values of the *parameters* p and you can write the model as a set of equalities $h(x, z, p) = 0$. The solution of which will give access to the state of the system.

Practically, engineers develop models within specifically tailored modeling languages and solve the set of equations using numerical solvers. In the framework of *OSMOSE*, models can be established using the languages *Vali*, *Easy* and *AMPL* as well as matlab. Each software has solvers associated to it so as to compute the state of the system. Please refer to the specific documentation of each software to establish your models.

4.3 Osmose models general structure

We have seen that a model can be defined as a set of equations that allow to compute a system state knowing the value of decision variables. In *OSMOSE*, a model is organized as an input-output entity. The general structure of an *OSMOSE* model is shown on Figure 4.3. As the scope of *OSMOSE* is to allow communication between different software and to include energy integration, a model is composed of several subsections.

A model can contain up to five subsections:

- Pre-computation function
- External software model
- Intermediate computation function
- Energy integration model
- Post-computation function

All the model sub-sections are linked to the input and to the output of the model. Inside the model, the subsections are called sequentially as shown by the arrows in Figure 4.3. In the following paragraph we introduce the *OSMOSE* communication structure by defining the `.Tag` structure. The description of each model sub-section follows.

4.4 The Tags Structure

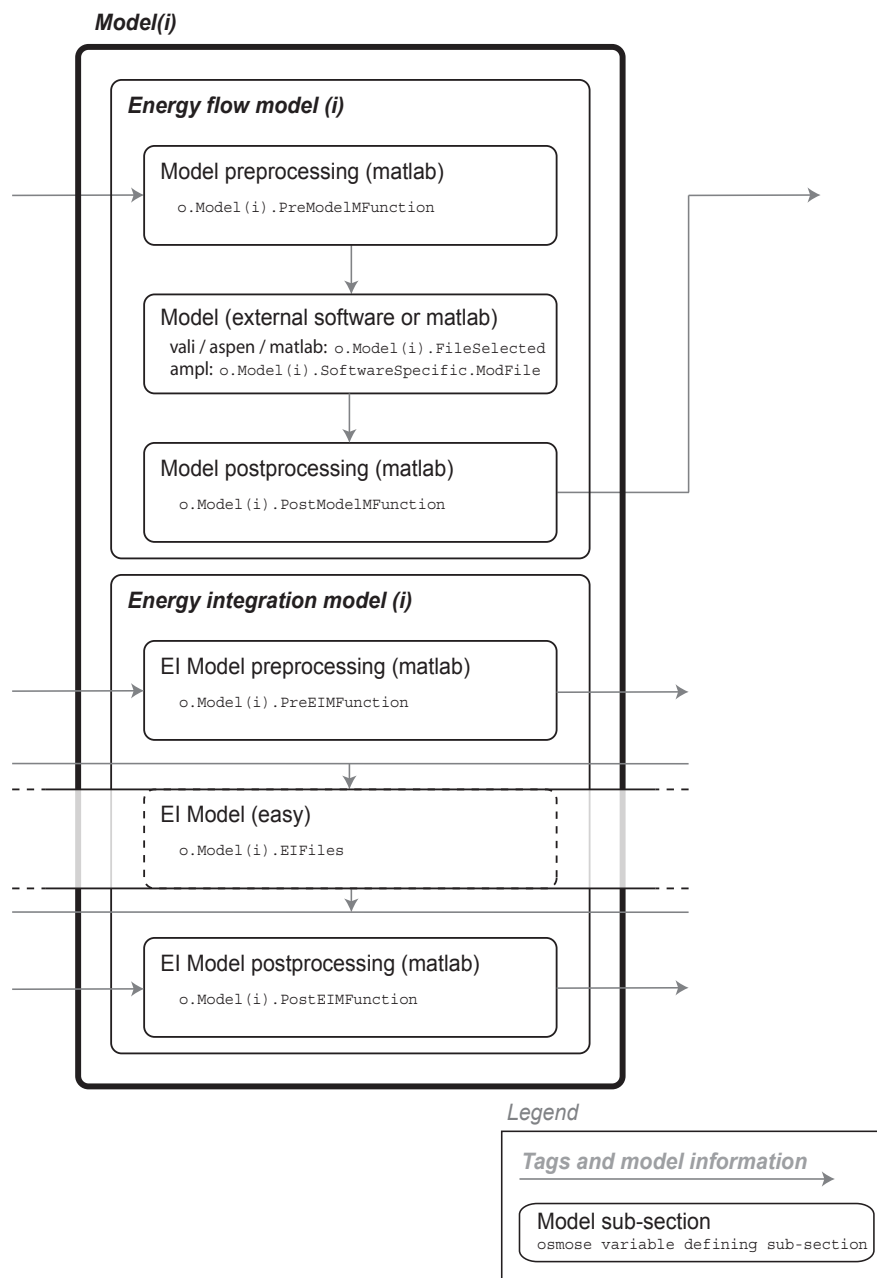
The communication within *OSMOSE* is performed through a matlab structured variable `o`¹. All the information is stored in `o` within different fields. The field `o.Model` contains all information defining models. *OSMOSE* provides the option of defining several models to be computed in the same time. The variable `o.Model` is thus a vector containing as many elements as there are models defined, `o.Model(i)` for $i = 1 : N_{models}$.

The model field is itself a structured variable organized into several subfields. Section 5.1 details the model definition fields. A model is selected for computation through its numeric identifier, i , stored in the field `o.ModelID`.

The values of variables (decision variables as well as dependent variables) are transmitted and updated throughout the computation using a specific sub-structure of `o.Model` called `.Tag`. Each relevant model variable or parameter is associated to a tag. The tag structure is composed of several fields as shown in Listing 4.1:

¹For an introduction to matlab structured variables, refer to matlab user guide[9]

Figure 4.5: Osmose model general structure



Context	Corresponding Name
Used or produced by Matlab	Name of the Matlab local variable
Used by <i>Vali</i>	Name of <i>Vali</i> tag with 'cst' or precision status
Produced by <i>Vali</i>	Name of <i>Vali</i> tag
Used by <i>Aspen</i>	Name of <i>Aspen</i> tag with 'cst' (see Section 5.3)
Produced by <i>Aspen</i>	Name of <i>Aspen</i> variable
Used by <i>AMPL</i>	Name of an <i>AMPL</i> parameter or set
Produced by <i>AMPL</i>	Name of an <i>AMPL</i> variable
Used by <i>Easy</i>	Name used in the <i>Easy</i> input template file
Produced by <i>Easy</i>	Name of a quantity written into easy output (see Section 5.5)

Table 4.1: Tag names contextual correspondence

- **.TagName** This field is the tag identifier, it must be unique. Communication is established between *OSMOSE* and the model using the **.TagName** field that has to correspond to the name of a variable in one of the sub-models. Table 4.1 gives tag names correspondences so as to perform this communication.
- **.DisplayName** Optional field, the display name is a short description of the tag. It is used for graphic axes labeling as well as in automatic reports.
- **.Unit** The unit field is an informative field that stores the unit of the variable. *OSMOSE* does not handle unit conversion.
- **.Status** The status can be constant ('CST'), variable ('OFF'), a Ampl set ('SET') or a number indicating a precision (to be used with Vali for data reconciliation).
- **.Value** The value field is either filled by *OSMOSE* before entering into the model (see Chapter 6) or it is completed during model resolution. Each model sub-section can fill the value of a previously defined tag as well as define new tags to be sent to output and/or to be used by successive sub-sections.².

```
% Define a tag as input of model i
nt = 0; % Tag index to be incremented at each new tag definition
nt=nt+1;
o.Model(i).Tag(nt).TagName = {'ZFP_T'};
o.Model(i).Tag(nt).DisplayName = {'Temperature of furnace'};
o.Model(i).Tag(nt).Unit = {'K'};
o.Model(i).Tag(nt).Status = {'CST'};
% o.Model(i).Tag(nt).Value : This field is attributed during computation
```

Listing 4.1: Tag definition - An example

This structure for tags allows also to extract easily tags values. For example, if the tag "Hot utility" has been defined in the frontend its value can be recovered as described in the listing 4.2.

4.5 Establishing a model

In this section are grouped the descriptions of the possible sub-models you are going to use.

²Within *OSMOSE* this operation is performed by the `update_output_tags` function. It is also the function to call within a user defined matlab function

```
%List of all tags defined in the model
o.Model.Tags.TagName
%Find the position of a given Tag
find(strcmpi('hot_utility',[o.Model.Tags.TagName]))
%Recover the value of a given tag based on its position (here 5)
o.Model.Tags(5)
```

Listing 4.2: Tag value extraction - An example

4.5.1 Pre-computation function

The pre-computation function is a matlab function performing computations before calling the external model. Its call is defined in the front-end by the variable `o.Model(i).PreComputationFunction`. In this function, tags can be created, and other fields of `o.Model` can be modified. Notably, the pre-computation function can perform the task of selecting the external model, next to be called, among several files listed. The pre-computation function argument as well as its output is the structured variable `o`. An example of pre-computation function for a *Vali* model is given in Appendix .1.

4.5.2 External Software Model

This section of the model calls an sub-model established either in *Vali*, *Aspen* or in *AMPL*. To perform the call, *OSMOSE* uses the values of already defined tags. After the external model resolution, the list of tags is updated and completed to include external model variables values.

As seen in section 4.5.1, several files, thus several models, can be defined as available for the external model step. In this case, the selection of the external model to call is part of the main model and is usually a function of the decision variables. Therefore, the pre-computation function has to be written so as to select the appropriate model according to the value of the input variables of the problem.

Vali model

Vali [2] is a software to assess, validate, monitor and optimize process plant performances. It includes thermodynamic states computations as well as chemical reactions and equilibrium resolutions. Two usages of *Vali* can be distinguished, (a) modeling, (b) data reconciliation. Modeling usage solves the model equations with an equation solver approach, thus computing the state of the system. Data reconciliation is used to compute the most probable state of a system with uncertain measurements by minimizing the sum of the square-residues. A *Vali* model is stored of a file with `.bls` extension.

Vali software internally defines objects called tags for modeling and computation. The correspondence between *Vali* tags and *OSMOSE* tags is performed by giving them the same tag name. *OSMOSE* inputs tag information to *Vali* through a file called *measurement file*³. The *OSMOSE* tag structure is updated after resolution by reading the output file⁴.

For the definition of a *Vali* model in the front-end see section 5.2.

Aspen model

Aspen Plus <http://www.aspentech.com/> is a process modeling tool for conceptual design, optimization and performance monitoring for chemical and power industries. *Aspen* is used to simulate chemical and thermodynamic systems, thus computing the state of the system. It can be run in GUI (Graphical User Interface) mode or in text mode from the command prompt. After having drawn the *Aspen* model and

³'temp_mea.mea' in the *OSMOSE_temp* folder

⁴'r6v' extension in the *OSMOSE_temp* folder

having validated it in the GUI mode, the *Aspen* model has to be exported as an input file (*filename.inp*) for using it in *OSMOSE*.

For the definition of an *Aspen* model in the front-end and the definition of a tag for an import or export *Aspen* variable see section 5.3

Ampl model

AMPL [1] is a language for large-scale optimization and mathematical programming problems. A model written in *AMPL* is generally an optimization problem whose solution is found by connecting the problem with a solver. There are several solvers available to use with *AMPL* [1], ranging from LP to MINLP. In addition to the *AMPL* web site [1], information on *AMPL* can be obtained by reading the user manual [14].

In *AMPL* language, distinction is made between *parameters* and *variables*. Parameters have constant value during optimization, whereas variables are computed by solving the problem. *AMPL* allows the definition of *sets* to perform arrays operations, therefore *AMPL* parameters and variables can be multi-dimensional.

An *AMPL* model is generally composed by a file with the *.mod* extension, called the *model file*, and by one or more files with *.dat* extension, called the *data file(s)*. The model file contains the optimization problem definition, sets, parameters and variables definition, as well as the model equations. The data files contains values of sets and parameters.

The input of an *AMPL* model is composed of the values of the model parameters and of the values of sets. For usage of *AMPL* with *OSMOSE* two types of inputs are distinguished:

- **Dynamic input** : parameters or sets that can be modified during *OSMOSE* computation
- **Static input** : parameters or sets that can not be modified through *OSMOSE*

To organize dynamic input, *OSMOSE* extracts the list of parameters from the *AMPL* model file, completes the values by matching the parameter names with the *OSMOSE* tag names, and writes a data file to be used during *AMPL* model resolution⁵ The data file written by *OSMOSE* is called the *dynamic data file*. It can contain dynamic sets as well, e.g. sets that are defined by an *OSMOSE* tag. An *OSMOSE* tag corresponding to an *AMPL* set has the *.Status* field equal to *SET* *'SET'*. An example of *AMPL* dynamic set definition through *OSMOSE* tag is given in listing 4.3.

Static input can either be defined in the *AMPL* model file, and/or in a *static data file* that has to be specified in the model definition (see example listing 5.5).

```
% Define a set as input of \ampl model i
nt=nt+1;
o.Constants(nt).ModelTagName = {'mymodel'};
o.Constants(nt).TagName = {'tech'};
o.Constants(nt).DisplayName = {'Set of technologies'};
o.Constants(nt).Unit = {'-'};
o.Constants(nt).Status = {'SET'};
o.Constants(nt).Value = [1 2 3 4];
```

Listing 4.3: Tag definition - A dynamic *AMPL* set

It is important to notice that as *AMPL* variables and parameters can have multiple dimensions, the corresponding *OSMOSE* tags will have the same dimensions. Furthermore, if an *AMPL* parameter used

⁵*'data_dyn.dat'* in the *OSMOSE_temp* folder.

in dynamic input is an array, the sets defining the array dimensions have to be numeric sets starting with one and with an increase of one. Example: [1 2 3 4 5] is a valid set, whereas [4 'berkley' 'tomato'] is an invalid set in *OSMOSE* input context.

4.5.3 Intermediate function



The intermediate function is a matlab function performing tags operations and selection of models before entering the energy integration model. It has the same format as the pre-computation and post-computation functions and is called `o.Model(i).EIPreparationFunction` in the frontend.

4.5.4 Energy Integration Model

One important feature of *OSMOSE* is to allow integration of energy systems involving heat exchange. This integration leads to a better efficiency of the considered system by determining the minimum energy requirements (MER) from external utilities. The Energy Integration Model is therefore an optimization problem, solved in a sub-section of the main model.

After computation of the thermal and mechanical energy requirements by the external software model and preparing the information in the intermediate function, it is possible to perform the energy integration of the whole system. For this purpose, the list of hot and cold process streams is defined, which allows to formulate the heat cascade and to determine the minimum energy requirements (MER) of the system as described in [6].

By setting up the appropriate utilities like cooling water or refrigeration cycles and hot streams created through combustion of fuels in a boiler, gas turbine, etc., the system is integrated to meet the MER. Defining further heat recovery technology like Rankine cycles, the optimal heat recovery for combined heat and power production with respect to operating cost, mechanical power production, exergy losses or CO₂ emissions is determined.

In *OSMOSE*, the energy integration is performed with *EASY*, which is an advanced energy integration program developed by LASSC of the University of Liège. *EASY* stands for Energy Analysis and SYnthesis of industrial processes. It implements the Effect Modelling and Optimisation approach using Mixed Integer Linear programming techniques to target the combined energy and environment optimal integration of industrial processes. An overview of the functionalities of the software are available at  http://leni.epfl.ch/exsys/EASY_On_line_manuals/ and  <http://leniwiki.epfl.ch/leniwiki/index.php/Easy>.

The activation of the energy integration model is performed in the front-end by setting the variable `o.ComputeEI` to 1. The definition of the streams is generally done through a template file as explained in Section 5.5.

4.5.5 Post-computation function

The post-computation function closes the sequence of model subsections. It is a Matlab function with accepting `o` as argument and outputting `o`. This function can contain computations as well as new tags definition⁶. Its name is specified in the `o.Model(i).PostComputationFunction` field in the front-end function.

4.6 Choosing the computation to perform

Once the model is defined, it is perceived as a black box input-output structure that can be called under various conditions, thus allowing to perform several kinds of computations.

⁶For completing tag list use the `update_output_tags` function of the *OSMOSE* package

Software	Possible Variable
<i>Vali</i>	Any <i>Vali</i> Tag with 'CST' status or with measured precision status
<i>Aspen</i>	Any <i>Aspen</i> Tag with 'CST' status
<i>Easy</i>	Tag name used in the input template and not computed by the previous sub-models
<i>AMPL</i>	<i>AMPL</i> parameter or set not computed by the previous sub-models and not attributed in the static input
Matlab	matlab variable not computed by the previous sub-models and not attributed in a matlab sub-function

Table 4.2: Available input variables

The list of computations currently handled by *OSMOSE* is:

- Model snapshot, also called one run
- Sensitivity analysis
- Optimization using evolutionary algorithm (mono- or multi- objective)
- Detailed recomputation of optimization results

As we have seen, the communication within a model is performed through a structure called `Tag`. By varying the value associated to a tag, *OSMOSE* commands the resolution of the model under various conditions. For each of the above computations, the user defines which are the tags to be varied using the structure `o.Variables`. Table 4.2, indicates the nature of the variables for each software used by the model.

OSMOSE allows also to define `o.Constants`, a structure defining tags to be kept constant during a computation but which value can differ from the default value of the tag.

Some remarks can be made at this point:

- An input defined by a `o.Variables` field or `o.Constant` field *do not need* to be defined as a `o.Model(i).Tag`. A `o.Variable`, or `o.Constant`, can however correspond to a `o.Model(i).Tag`. In the latter case, the value attributed by *OSMOSE* through the `o.Variables`, or `o.Constant` field is the one considered for computation⁷.
- Any *Vali* 'CST' tag that is neither defined as a `o.Model(i).Tag`, as an `o.Variables` or as a `o.Constant` takes automatically the value defined into the *Vali* or *Aspen* model.

4.6.1 Model snapshot

The model snapshot performs a single computation of the model and stores the obtained tags, files and energy integration graphs in the computation run directory (see Figure 4.6.1). When performing a model snapshot, the values of tags can be made different than the default value by defining the `o.Constant` structure.

4.6.2 Sensitivity analysis

Sensitivity analyses are used to observe the variation of the model dependent variables under the variation of one or more decision variables. *OSMOSE* allows to perform sensitivity analyses in one or two dimensions.

⁷In the software this is performed by the `osmose_assign_variables` function.

Figure 4.6: Model snapshot organization

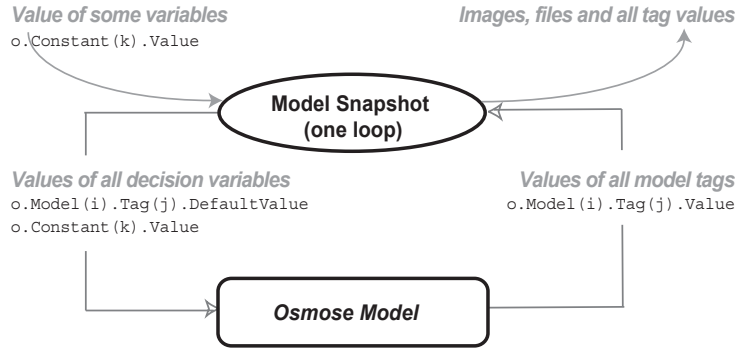
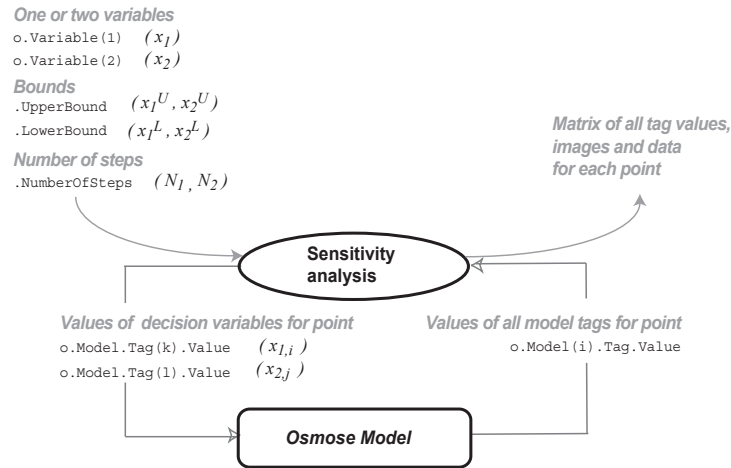


Figure 4.7: Sensitivity analysis organization



The decision variables are defined through the `o.Variables` structure (see Paragraph 6.4) by giving the lower and upper bounds of the variations as well as the number of steps. *OSMOSE* will then call the model by looping the variables as follows.

Let x_1 and x_2 be the variables on which the analysis is performed. Define x_1^U, x_2^U the upper bounds, x_1^L, x_2^L the lower bounds, and N_1, N_2 the number of steps. The loop on the first variable is defined by equation 4.1. The loop on the second variable is defined by equation 4.2

$$x_1^i = x_1^L + i * \frac{(x_1^U - x_1^L)}{N_1} \quad i = 1, \dots, N_1 \quad (4.1)$$

$$x_2^j = x_2^L + j * \frac{(x_2^U - x_2^L)}{N_2} \quad j = 1, \dots, N_2 \quad (4.2)$$

These loops build a grid. The model is computed at each node of this grid. For each computed point, *OSMOSE* retrieves tag information and organizes it into a matrix. Energy integration graphics and model files can also be saved to be analyzed in a second time (see Figure 4.6.2).

4.6.3 Optimization

Another main feature of *OSMOSE* is the opportunity to perform optimization on the considered model. For this purpose, *OSMOSE* is coupled with an optimizer that allows multi-objective optimization of black-box models. The algorithm used is evolutionary based and is implemented in the software *moo*.

moo stands for Multi-Objective Optimizer. It is a Matlab based application, which has been developed at LENI by G. Leyland [5] and A. Molyneaux [10]. Its search for optima, inspired from genetics, is perfectly adapted for solving optimization problems on energy systems, which are frequently non-linear and non-continuous.

In this context, an optimization problem is generally defined as follows:

$$\begin{aligned}
 \min \quad & f(x, z) \\
 \text{s.t.} \quad & h(x, z) = 0 \\
 & g(x, z) \leq 0 \\
 & x_i^L \leq x_i \leq x_i^U \quad i = 1, \dots, N
 \end{aligned} \tag{4.3}$$

Otherwise stated, the problem is to minimize function $f(x, z)$ of the decision variables $x = [x_1, \dots, x_N]$ and dependent variables $z = [z_1, \dots, z_M]$. The search is submitted to the model equality constraints, $h(x, z) = 0$, and inequality constraints, $g(x, z) \leq 0$, and is limited to the space determined by lower bounds, x_i^L for $i = 1, \dots, N$, and upper bounds, x_i^U for $i = 1, \dots, N$ on the decision variables.

The nature of the objective function and of the model equations influences the way to solve an optimization problem. The following four situations can occur (in ascending order of difficulty). When $f(x, z)$ and the model equations are linear, the problem is a linear programming (LP) model. When some of the variables are integer (representing for example yes or no decisions) the problem is a mixed integer linear programming (MILP) problem. If there are non-linear equations the problem is known as being a non linear programming (NLP) problem. A NLP problem is a mixed integer non linear programming (MINLP) problem if integer variables are involved.

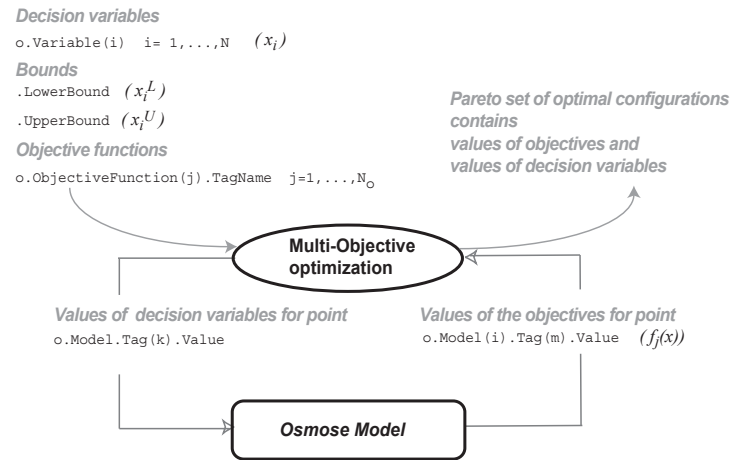
When $f(x, z)$ has more than one dimension, the problem is known as being a multi-objective problem. This is the kind of problem that often arise in the design of industrial systems. A simple example is the problem of maximizing the production of a plant while minimizing its installed cost. These two objectives are competing, when the installed cost is low, the production will also be low, when the production is high the cost will be high. However some solutions will offer a better compromise between the two objectives as others, this are the solutions we are going to find.

The solution of a multi-objective problem is a set of solutions that express the possible compromise between the objectives. In the domain of the objective functions, this compromise is represented by the Pareto frontier. This curve represents the set of non dominated solutions, it delimits the unfeasible domain from the feasible but sub-optimal one. The book of Kalyanmoy Deb [4] contains exhaustive information about multi-objective problems and the methods to solve these using evolutionary algorithms as it is the case in *OSMOSE*.

Evolutionary algorithm search for the optimal trade-offs by comparing the fitness of solutions. An initial population of random points is generated in the space of the decision variables x , each point is evaluated by computing the values of objectives. The algorithm performs then a breeding of the points that are more satisfactory with respect to the objective; this breeding generates a new population of points that is closer to the optimal solution. These points become the new genitors of the next generation and the process starts all over again. There is no termination criterion of the search, the user normally set an upper limit of iterations at which he expects to be close enough to the real optimum.

To perform an optimization, *OSMOSE* is coupled with *moo* and evaluates the objective function by solving the model at the points generated by *moo* (Figure 4.6.3). The user specifies the search space by

Figure 4.8: Optimization organization



defining the decision variables and their bounds. So as to be retrieved by the optimizer, the objective has to be a tag of the model.

The result of such an optimization is a Pareto set. The points of the Pareto set are defined and stored by their coordinates in the decision variables space, *OSMOSE* stores also the Pareto curve with the values of the objectives. However, the complete state of the optimal systems is not retrieved by an optimization computation. To have a complete insight on the state of the optimal systems the Pareto sets points have to be recomputed and the value of all model tags retrieved as explained in the following section.

4.6.4 Recomputation

The recomputation tool is used after an optimization. As we have seen, the results of an optimization in *OSMOSE* do not furnish the complete state of the system at each optimal point, but merely the coordinates of these points in the space of the decision variables.

During a recomputation, the optimal points are run once again in the model in order to retrieve complete information on the state of each optimal point (see Figure 4.6.4). This information consists in the value of all the model tags. Other properties, such as energy integration curves or model input files can also be stored for each of the optimal points.

OSMOSE organizes the outputs of a recomputation by storing the information in specific folders. For complete recomputation description see section 6.6.

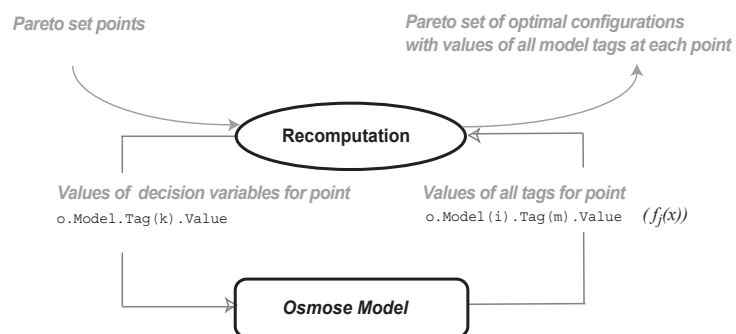
4.7 Analysing data

The previous computations generally generate a large amount of data. The organisation of the data for further analysis is an essential point. *OSMOSE* provides the opportunity to generate graphical outputs of the computation using an interactive window.

4.8 Frontend usage

OSMOSE is composed of several matlab functions. The user interacts with the software by defining the problem in a matlab function called the *front-end* (or *front-end function*).

Figure 4.9: Recomputation organization



Chapter 5

Models Definition

As seen in chapter 4, problem setting in *OSMOSE* can be performed through a matlab function called *front-end function* (or *front-end*). This function has to contain model definition as well as computation instructions. The present chapter details model definition aspects of the front-end.

In a front-end, models are defined within the `define_model` sub-function. Each model definition consists in a general part (section 5.1) that has to be completed with specific parts according to the model type (see sections 5.2, 5.4 and 5.5). These parts are described below. For clarity the matlab code is displayed in several pieces with listings. A complete front-end file template is given in annex (**faire l'annexe!!**)

5.1 General model definition

In this part of the front-end, the model is defined by its name, location and type. For identification purposes as well as for folders naming, each model must have a name, defined by the `o.Model(i).TagName` field. The name is defined by the user and can be any string of characters, without spaces.

As seen in 4.3, an *OSMOSE* model is generally composed by several sections, each section consisting in one or more files. All the files composing a model have to be in the same folder, the *model repository*. The path to the model repository is defined in the `o.Model(i).Location` field.

The `.Software` field refers to the software of the external model. Available options are listed in Table 5.1.

Value	Action
{'vali'}	The external model is a <i>Vali</i> model
{'aspen'}	The external model is a <i>Aspen</i> model
{'ampl'}	The external model is an <i>AMPL</i> model
{'easy'}	The external model is described according to Energy Technologies syntax
{'eiamp1'}	The external model is described according to Energy Technologies syntax
{'matlab'}	The external model is a model composed of <i>Matlab</i> functions

Table 5.1: Available external softwares

The `.FileName` field contains the names of the files composing the model. The list should contain all the files that are necessary for model computation. Files specified in another `o.Model` field do not need to be mentioned as the list is internally completed by *OSMOSE*. For example the matlab functions used for pre-computation, intermediate computation and post-computations do not need to be in the `.FileName` list as they are defined by `.PreComputationFunction`, `.EIPreparationFunction` and `.PostComputationFunction`. Table 5.2 gives the list of fields containing file names.

Field	Description
.PreComputationFunction	Matlab function performing pre-computation
.EIPreparationFunction	Matlab function preparing energy integration
.PostComputationFunction	Matlab function performing post-computation
.EIFiles	List of files used for energy integration
.SoftwareSpecific.ModFile	AMPL model : name of the model file
.SoftwareSpecific.FixedDataFile	AMPL model : eventual fixed data files list
.FileName	All other model files not defined in above fields

Table 5.2: o.Model fields defining model files

If the model contains an energy integration section, the `o.ComputeEI` field has to be set to 1. `o.Easy.Objective` contains the name of the objective function used for the resolution of the heat cascade. The energy integration properties of the model are further detailed in section 5.5

The *OSMOSE* version can also be specified here. By default, this option is set to 2.0. .

5.2 Vali model definition

A *Vali* model is defined by the location of the executable of *Vali* and by specifying the process flow diagram (PFD) of the *Vali* model to be executed as shown in listing 5.2. *OSMOSE* recognises the *Vali* model file to be used by extracting from the `o.Model(i).FileName` list the files with `'.bls'` extension. If the `FileName` list contains more than one *Vali* file, the selection of the *Vali* file has to be performed in the pre-computation function.

The measurement file, input of the vali model, is generated by *OSMOSE* . The user can specify additional lines to be written in the measurement file by writing them in a file with `'.mea'` extension. The `.SoftwareSpecific.AdditionalMEA` field specifies the name of this file.

5.3 Aspen model definition

An *Aspen* model is defined by the location of the *Aspen* executable and by specifying the model as shown in listing 5.3. During execution *OSMOSE* will create a temporary copy of the `filename.inp`, append some lines to it, execute the *Aspen* engine in text mode, extract the variables and save them in `o.Model.Tags`. From Aspen's view, the interface appears as a *Calculator block* automatically inserted by *OSMOSE* .

The new function `DefineAspenTags` must be created to define a new tag for each import and export variable. Each tag must be defined in a separate line by calling the function `define_aspentag` as outlined in listing 5.4. The variables `unit`, `type`, `param_1`, `param_2` and `param_3` must match the corresponding ASPEN variable definition. It can be found in ASPEN by creating a fake calculator. From the ASPEN GUI, simply go to: Data Browser / Flowsheeting options / Calculator / Variable definition. Do not save any file in ASPEN with a calculator in it! An example screenshot can be found in appendix 2. Table 5.3 shows the supported variable types and keywords and Table 5.4 gives a few examples of common variables.

Type	Keyword 1	Keyword 2	Keyword 3
STREAM-VAR	STREAM	SUBSTREAM	VARIABLE
BLOCK-VAR	BLOCK	VARIABLE	SENTENCE
INFO-VAR	INFO	VARIABLE	STREAM
STREAM-PROP	STREAM	PROPERTY	-

Table 5.3: Supported Aspen variable types and keywords

```

function o = DefineModel(o)
i = 0; % the index i can be incremented for any new model
i = i+1;
% Define a name of the model [required].
o.Model(i).TagName = {'namemodel'};
% Define the storage location of the model [required].
o.Model(i).Location = {'C:\oc\SNG_model_2.5'};
% Define the software the model is developed in [required].
o.Model(i).Software = {'vali'};
% Define the files that are needed for the computation [required].
o.Model(i).FileName = {'valimodel.bls','easymodel.txt','matlabmodel.m',...
,...,'add_mea.mea'};
% Define a matlab function performing computations before entering in the software
% [optional]
o.Model(i).PreComputationFunction = {'vali_tag_assignment.m'};
% Define a matlab function performing computations after software and energy integration
% [optional]
o.Model(i).PostComputationFunction = {'PostcomputeModelSimple.m'};

% -----
% Specific properties for energy integration
% Specify if EASY is used to perform the energy integration [required].
o.ComputeEI = 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.EasyObjective = {'MER'}; or {'Operating cost'}; or {'Exergy'}; or {'Mechanical power'};
% (Default is {'Operating cost'})
o.Easy.Objective = {'MER'};
% If you need to perform computations between external software model and energy integration
% specify the name of the matlab file to run [optional]
o.Model(nm).EIPreparationFunction = {'easy_tag_assignment.m'}
% -----

% Specify the version of OSMOSE you use [optional]. Default is {'2.0'}.
o.Model(i).OSMOSEVersion = {'2.5'};

```

Listing 5.1: Front-end example - General Model Definition

Note: the syntax displayed in the fake calculator definition is not always correct, especially for block variables. In case of doubt, export the ASPEN file, including the fake calculator, in a temporary `.inp` file by clicking on File/Export. The correct syntax can be found by checking the calculator definition in the source code in the temporary `.inp` file. Again, do not save the `.bkp` file with the fake calculator in it!

After Aspen tags are defined, import variables must be assigned a value according to the *OSMOSE* structure `o.Constants` or `o.Variables`. Import variables must have the 'CST' status and export variables must have the 'OFF' status. If the status is not defined, ASPEN will automatically assign the 'CST' status to all tags present in `o.Variables` and `o.Constants`, and the 'OFF' status to all other tags.

To make sure that execution errors result in a failed convergence in OSMOSE, set the maximum number of errors in *Aspen* to 1. Go to Data_Browser/Simulations_options/Limits. In that case, ASPEN will quit after the first error encountered, even if it could have recovered from it. A higher number of maximum errors in *Aspen* could lead to model convergence however the errors would not be seen in *OSMOSE*.

```
% Define the location of the executable vali file [optional].
o.Vali.Path = {'C:\BelSim\bin\valiauto.exe'};
% Define the PFD of your vali file you want to execute [optional].
% (Default is {'MAIN'};)
o.Model(i).SoftwareSpecific.PFD = {'MAIN'};
% Specify additional lines to use in the vali MEA file [optional].
o.Model(i).SoftwareSpecific.AdditionalMEA = {'add_mea.mea'};
```

Listing 5.2: Code matlab - Vali model definition

```
% Define the software the model is developed in [required].
o.Model(i).Software = {'aspen'};
% Define the location of the executable aspen file [optional].
% Default is {'C:\Program Files\AspenTech\Aspen Plus 2004.1\Engine\Xeq\aspen'};
o.Aspen.Path = {'C:\Program Files\AspenTech\Aspen Plus V7.0\Engine\Xeq\aspen.exe'};
% Define the files that are needed for the computation [required].
o.Model(i).FileName = {'filename.inp'};
% Define the model file to be run [required]
o.Model(i).FileSelected = {'filename.inp'};
% Define tags for import /export variables
o = DefineAspenTags(o);
```

Listing 5.3: Code matlab - Aspen model definition

5.4 *AMPL* model definition

Listing 5.5 shows the front-end section used to define an *AMPL* model. The properties of the model are defined through the `o.Model.SoftwareSpecific` substructure. The mandatory field for an *AMPL* model definition is `.ModFile` that contains the name of the *model file* (complete with extension). Optional static data files can be specified through the field `.FixedDataFile`.

The solver to be used is specified in the `.AmplSolver` field. This field is optional as the '*minos*' solver is set by default. Consult the *AMPL* manual [14] for a list of solver options, these can be specified through the `SolverOption` field.

Additional solving and reporting options can be specified through the `.AmplOption` field, consult the manual [14] for a list of options.

5.5 Energy integration model definition

In this section of the frontend template, the *EASY* model is defined. Several specifications are required to achieve the definition of the *EASY* model as shown in listing 5.6. Don't forget that the selection of *EASY* to perform the energy integration and the specification of the objective function to be minimized for the resolution of the heat cascade have been done earlier in the model definition section with the fields `.Easy.Objective` and `.ComputeEI`. A template of the text files specified as `.Model(i).EIFiles` and containing the list of hot and cold streams and the definition of the utility and heat recovery systems

type	param_1	param_2	param_3	description
STREAM-VAR	<i>stream name</i>	MIXED	TEMP	temperature
STREAM-VAR	<i>stream name</i>	MIXED	MOLE-FLOW	mole flow

Table 5.4: Example of common *Aspen* import/export variables


```

function o = DefineAspenTags(o)
% -----
% Definition the tags for import/export from the aspen model
% To get a list of possibilities, type 'help define_aspentag' in the
% matlab command window
% -----

% o=define_aspentag(o, name, unit, type, param1, param2, param3)

% TO ASPEN
% H2O partial massflow in stream GAS-HOT in kg/hr named in_H2O
o = define_aspentag(o,'in_H2O','kg/hr',...
    'Mass-Flow Stream=GAS-HOT Substream=MIXED','Component=H2O');

% Compressor 'COMPR' pressure named Pressure
o = define_aspentag(o,'Pressure','bar',...
    'Block-Var Block=COMPR Variable=PRES','Sentence=PARAM');

% From ASPEN
% Temperature of stream RICH-HOT in °C, named HPin_T
o = define_aspentag(o,'HPin_T','C',...
    'Stream-Var Stream=RICH-HOT Substream=MIXED','Variable=TEMP');
% Heat duty of stripper named HP-STRIP
o = define_aspentag(o,'HP_DUTY','Watt',...
    'Block-Var Block=HP-STRIP Variable=COND-DUTY','Sentence=RESULTS');

```

Listing 5.4: Code matlab - Aspen Tags definition

necessary to compute the energy integration with *EASY* can be found [put chapter or section].

Optionally, other tags than the multiplication factors of the streams can be extracted from the easy report file¹. The strings of these variables are defined in the field `.SoftwareSpecific.ToOutputTags`. The according column number in which they occur in the report file and their unit is specified in the fields `.SoftwareSpecific.Tags_position` `.SoftwareSpecific.ToOutputTagsUnit` respectively.

5.5.1 Tags definition

listing 5.7

¹'synep.html' in the OSMOSE_temp/EnergyIntegration folder

```

% Main ampl model file containing parameters and variable definition
% model equations and constraints [required]
o.Model(i).SoftwareSpecific.ModFile = {'Probleme_simple_M00.mod'};
% Data file of static parameters [optional]
o.Model(i).SoftwareSpecific.FixedDataFile = {'Probleme_simple_M00.dat'};
% Definition of \ampl options [optional]
nbo = 0 ;
nbo=nbo+1;
o.Model(i).SoftwareSpecific.AmplOption(nbo).Name = {'presolve_fixeps'};
o.Model(i).SoftwareSpecific.AmplOption(nbo).Value= {'2.73e-12'};
% Definition of ampl solver [optional] (default is minos)
o.Model(i).SoftwareSpecific.AmplSolver = {'cplexamp'};
% Definition of solver options [optional]
nbo = 0 ;
nbo=nbo+1;
o.Model(i).SoftwareSpecific.SolverOption(nbo).Name = {'prestats'};
o.Model(i).SoftwareSpecific.SolverOption(nbo).Value = {'1'};

```

Listing 5.5: Code matlab - *AMPL* model definition

```

% Define the template files containing the easy problem definition (i.e. the list of hot and
% cold process streams and the definition of the utility and heat recovery systems) [required]
].
% Ex.: o.Model(i).EIFiles = {'h2o_psa1.txt','h2o_psa2.txt'};
o.Model(i).EIFiles = {'h2o_psa.txt'};

% Give the tags (other than flowrates) you want to obtain from easy [optional].
% Ex.: o.Model(i).SoftwareSpecific.ToOutputTags = {'BALANCE','ADV TOTALCO2','ADV RAMBIANT'};
o.Model(i).SoftwareSpecific.ToOutputTags={'BALANCE','ADV TOTALCO2','ADV RAMBIANT',...
    'ADV O2_ELSYS','ADV F105','ADV F113','ADV F118','ADV F227','ADV F238'};

% Give the position (nth column) of the ToOutputTags in the easy report file [required if
% ToOutputTags are defined].
% Ex.: o.Model(i).SoftwareSpecific.Tags_position = [6,7,7];
o.Model(i).SoftwareSpecific.Tags_position=[6,7,7,7,7,7,7,7,7];

% Give the units of the ToOutputTags [optional].
% Ex.: o.Model(i).SoftwareSpecific.ToOutputTagsUnit = {'kW','kg/s','kg/s'};
o.Model(i).SoftwareSpecific.ToOutputTagsUnit = {'kW','kg/s','kg/s','kg/s','kg/s','kg/s','kg/s','kg/s','kg/s'};

% Define the Matlab function that determines which easy template file to use [optional if one
% file is specified in o.Model(i).EIFiles], [required] otherwise. In this function, you are
% also free to perform operations on the tags before running easy.
% Ex.: o.Model(i).EIPreparationFunction = {'easy_tag_assignment'};
o.Model(i).EIPreparationFunction = {'easy_tag_assignment'};

% -----
% Call the tags definition defined at the end of this template
% [required] for osmose version 2.5 (see o.Model(i).OSMOSEVersion)
% -----
o = DefineTags(o);

```

Listing 5.6: Code matlab - Energy integration model definition

```
function o = DefineTags(o)
% -----
% Definition the tags of the model (osmose version 2.5 only)
% This tags might be used in vali, easy or other computations
% -----
i=0;
% i = i+1;
% Specify the name of the model tag that is displayed [required].
% Ex.: o.Model(nm).InputTags(i).DisplayName = {'Nominal thermal capacity'};
%       o.Model(nm).InputTags(i).DisplayName = {'Reactor pressure'};
% Specify the name of the model tag [required].
% Ex.: o.Model(nm).InputTags(i).TagName = {'pth'};
%       o.Model(nm).InputTags(i).TagName = {'F_101_P'};
% Specify the unit of the model tag [required].
% Ex.: o.Model(nm).InputTags(i).Unit = {'kW'};
%       o.Model(nm).InputTags(i).Unit = {'bar'};
% Specify the default value of the model tag [required].
% Ex.: o.Model(nm).InputTags(i).DefaultValue = 20000;
%       o.Model(nm).InputTags(i).DefaultValue = 1.15;
% Specify the status of the model tag [required].
% Set it {'CST'} if it is fix or give an accuracy.
% Ex.: o.Model(nm).InputTags(i).Status = {'CST'};
%       o.Model(nm).InputTags(i).Status = 0.5;
% Continue the list...
% i = i+1;
% o.Model(nm).InputTags(i).TagName = {''};
% o.Model(nm).InputTags(i).DisplayName = {''};
% o.Model(nm).InputTags(i).Unit = {''};
% o.Model(nm).InputTags(i).DefaultValue = ;
% o.Model(nm).InputTags(i).Status = {'CST'};
```

Listing 5.7: Code matlab - Tags definition

Chapter 6

Computation definition

The present chapter discusses computation definition within *OSMOSE* through a front-end function. Once a stable model is defined¹, computations to be performed can be determined (see Section ??).

The possible computations are recalled here:

- Model snapshot, also called one run
- Sensitivity analysis
- Optimization using evolutionary algorithm (mono- or multi- objective)
- Detailed recomputation of optimization results

Each of the precedent choices is selected into the main function of the front-end file as shown in the following section. The definition of the computation parameters is then performed in specific sub-functions of the front-end. This process is detailed in the subsequent sections. For clarity, the matlab code is displayed in several pieces within listings. A complete front-end file template is given in annex (**faire l'annexe!!**)

6.1 Computation selection

The main function of the front-end contains computation definition. Listing 6.1 presents an example. As seen previously, the communication within *OSMOSE* is performed through a structured variable `o`. All the information is stored within `o` in different fields.

The first two fields concern display and output environment settings. The environment field `o.Silent` indicates if routine messages have to be silenced, `o.Silent=1`, or not, `o.Silent=0`. As a rule of thumb, deactivate silencing when developing a model or debugging a computation, *OSMOSE* will provide you with many useful messages about computation progression. Displaying messages is however extremely time consuming, therefore enable silencing for long computations.

The computation field `o.ComputationMode` selects the quantity of output data to be generated and stored.

The computation selection fields are `o.DoOneRun`, `o.DoSensi`, `o.DoMoo`, and `o.DoRecompute`. Their names are self-explanatory. The attribution of the value 1 to the field commands the computation execution; the value 0 deactivates the calculation. The field `o.DoRestartMoo` is used to restart a moo optimization in case of premature termination. Each of the computation fields is naturally activated independently, as it makes for example little sense to perform an optimization right after a sensibility analysis. One exception is the advisable recomputation of optimization results right after an optimization.

¹For *OSMOSE* model structure see Section 4.5. For model definition through the front-end refer to Chapter 5

Customizing options for reporting are also defined in the main part of the front-end, their description is given in chapter ??.

```
function o = My_Front_End_Function_Name

%%Environnement and output settings
% Reduce the routine messages to the minimum [required].
o.Silent = 0;           % 1: yes, 0: no
% Decide about the output of results [optional].
o.ComputationMode = {'simple'};
% {'simple'}: no details
% {'details'}: detailed results, which include composite curves

%% Computation setting
% Perform a single evaluation of your model [required].
o.DoOneRun = 1;         % 1: yes, 0: no
% Perform a sensibility analysis [required].
o.DoSensi = 0;         % 1: yes, 0: no
% Perform an optimisation with moo [required].
o.DoMoo = 0;           % 1: yes, 0: no
% Restart an optimisation with moo [required].
o.DoRestartMoo = 0;     % 1: yes, 0: no
% Recompute the points on the Pareto front to get the details [required].
o.DoRecompute = 0;     % 1: yes, 0: no

%% Reporting setting
% Generate a report of the computation [required].
o.DoReport = 0;        % 1: yes, 0: no
% Generate a custom report of the computation [required].
o.DoCustomReport = 0;  % 1: yes, 0: no

%% Call of the subfunction handling computations,
o = launch_osmose(o);
```

Listing 6.1: Front-end main function: computation definition

Each of the computation selection fields has a sub-function counterpart containing computation parameters definition. The definitions sub-functions are named `DefineOneRun`, `DefineSensi`, `DefineMoo`, `o.DefineRecompute` and `DefineRestartMoo`. Table 6.1 gives computation variables and sub-functions connections.

To call the required sub-function and thereafter launch computation, front-end execution passes through the `launch_osmose(o)` sub-function. This sub-function is normally placed at the end of the front-end and does not have to be edited. Listing 6.2 shows the `launch_osmose` sub-function. The last line, `o = run_frontend(o);`, calls the *OSMOSE* program.

6.2 Software location

Before running a model, the path and installation of the software used has to be checked. *OSMOSE* can communicate with the software listed in section 3 by locating their main executable file. *OSMOSE* locates automatically *Easy* and *AMPL* whereas paths to *Vali*, *Aspen* and *cygwin* executables have to be inserted. For *moo*, the path has to be added to the Matlab path following the same procedure as for the *OSMOSE* package (see Paragraph 3.1).

Table 6.1: Computations in *OSMOSE*

Operation	Corresponding variable	Corresponding sub-function
Single run	<code>o.DoOneRun</code>	<code>DefineOneRun</code>
Sensitivity analysis	<code>o.DoSensi</code>	<code>DefineSensi</code>
Optimisation	<code>o.DoMoo</code>	<code>DefineMooOptim</code>
Pareto points recomputation	<code>o.DoRecompute</code>	<code>DefineRecompute</code>
Restart stopped optimization	<code>o.DoRestartMoo</code>	<code>DefineRestartMoo</code>

Default paths are set in the `osmose_defaults` function. However, you can redefine the paths in the front-end function to match your local installation. This definition can be added in the main function of the front-end described above.

The default values of the paths are resumed in table 6.2 for Windows environment. Under Linux, the paths are already set in the system environment. This table gives also the variable name associated to each path.

Name	Default path	Variable Name
Vali	<code>v:\bin\valiauto.exe</code>	<code>o.Vali.Path</code>
Cygwin	<code>c:\cygwin\bin</code>	<code>o.Cygwin.Path</code>

Table 6.2: Default paths to external software under Microsoft Windows

Listing 6.3 gives an example of path redefinition into the front-end.

6.3 One run definition

Parameters for the execution of a single evaluation are defined in the `DefineOneRun` front-end sub-function (see Listing 6.4).

The definition of parameters for a single run computation is optional. If nothing is specified the model is run once using default values for the tags.

Two optional indications can be given for a one run computation: (1) additional files to be stored and (2) values of tags differing from the default values.

The model field `o.Model(nm).FilesToCopy` contains the names of files to be copied in the run folder. Using the `o.Variables` structure, the value of some tags can be modified. For a single run, the required fields of the `o.Variables` structure are:

- `.ModelTagName` : reference to the model containing the variable.
- `.TagName` : name of the concerned tag.
- `.Value` : value attributed to the defined tag for this computation.

6.4 Sensitivity analysis definition

The parameters for the execution of a sensitivity analysis are defined within the `DefineSensi` front-end sub-function. Sensitivity analyses can be performed up to two dimensions. Each of the dimension of the analysis is defined by a `o.Variables(i)` structure².

For a sensitivity analysis, the required fields of the `o.Variables` structure are:

²More precisely, `o.Variables` is a vector of one or two rows

```

function o = launch_osmose(o)
%do not edit this function
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,p] = DefineMooOptim(o);
    o = DefineRestartMoo(o);
end
if o.DoRecompute == 1
    [o,p] = DefineMooOptim(o);
    o = DefineRecompute(o);
end
if o.DoAutoReport == 1
    o = DefineAutoReport(o);
end
if o.DoCustomReport == 1
    o = DefineCustomReport(o);
end

o = run_frontend(o);

```

Listing 6.2: Code matlab - Launch osmose

- `.ModelTagName` : reference to the model containing the variable.
- `.TagName` : name of the concerned tag.
- `.DisplayName` : short description used for displaying in results analysis.
- `.Unit` : unit of the concerned tag, for display purposes.
- `.LowerBound` : lower bound of the concerned sensitivity axis.
- `.UpperBound` : upper bound of the concerned sensitivity axis.
- `.NumberOfSteps` : number of points computed between upper and lower bound.

6.5 Multi-objective optimisation

The parameters for the execution of a optimization using the evolutionary algorithm *moo* are defined within the `DefineMooOptim` front-end sub-function. An example is given in Listing 6.7.

The information required to perform an optimization is of three kinds:

- Definition of the optimization problem
- Definition of algorithm properties
- Definition of output format

```
% Local path to Vali
o.Vali.Path = {'C:\Belsim\textbackslash bin\textbackslash valiauto.exe'};
% Local path to Aspen
o.Aspen.Path = {'C:\Program Files\AspenTech\Aspen Plus V7.0\Engine\Xeq\aspen.exe'};
% Local Cygwin install
o.Cygwin.Path = {'D:\textbackslash Programs\textbackslash cygwin \textbackslash bin'};
```

Listing 6.3: Paths definition in the front-end

```
function o = DefineOneRun(o)
o.Model(nm).FilesToCopy = {'report_h2o_psa.html'};
i=0;
i = i+1;
% Define the name of the model to which the tag belongs [required].
o.Variables(i).ModelTagName = {'SNG_h2o_psa'};
% Define the name of the tag [required].
% This might be a general model parameter you use as tag (i.e. an index
% for economic calculation), or a variable used in vali or easy.
% Ex.: o.Variables(i).TagName = {'MS_index'}; for a parameter
%       o.Variables(i).TagName = {'F_101_P'}; for a vali variable
o.Variables(i).TagName = {'gp'};
% Specify the value of the variable for the run [required].
o.Variables(i).Value = 1.15;
% Continue the list...
% i = i+1;
% o.Variables(i).ModelTagName = {''};
% o.Variables(i).TagName = {''};
% o.Variables(i).Value = ;
```

Listing 6.4: One run definition in the front-end function

The optimization problem is defined by the fields:

- `o.Moo.nobjectives` : number of objectives
- `o.ObjectiveFunction` : structured variable indicating the tags that define the objectives
- `o.Variables` : structured variable defining the decision variables of the optimization problem with their bounds

The algorithm properties are defined in the fields:

- `o.Moo.InitialPopulationSize` : number of points evaluated in the initial population
- `o.Moo.max_evaluations` : Termination criterion for the optimization, maximal number of points evaluated iterations
- `o.Moo.nclusters` : number of families of points distinguished by the algorithm. Negatives values can be used to allows Moo to automatically handle clusters.

The display and output formats are defined in the fields:

The `o.GraphicOptions` subfields offer several options for displaying graphical results. The following fields take the value 1 if the option is activated, 0 else.

Tableau A COMPLETER ci dessous!!


```

function o = DefineSensi(o)
% Define the parameter to vary [required].
% Define the name of the model to which the tag belongs [required].
o.Variables(1).ModelTagName = {'SNG'};
% Specify the name that is displayed in results analysis [required].
o.Variables(1).DisplayName = {'Gasification pressure'};
% Specify the name of the model tag [required].
o.Variables(1).TagName = {'gp'};
% Specify the unit of the model tag [required].
o.Variables(1).Unit = {'bar'};
% Specify the lowest value of the variable [required].
o.Variables(1).LowerBound = 1.15;
% Specify the highest value of the variable [required].
o.Variables(1).UpperBound = 5.15;
% Specify the number of steps in the interval [required].
o.Variables(1).NumberOfSteps = 3;
% Define a second parameter to vary for 2D sensitivity [optional].
% o.Variables(2).ModelTagName = {''};
% o.Variables(2).DisplayName = {''};
% o.Variables(2).Unit = {''};
% o.Variables(2).TagName = {''};
% o.Variables(2).LowerBound = ;
% o.Variables(2).UpperBound = ;
% o.Variables(2).NumberOfSteps = ;

```

Listing 6.5: Sensitivity analysis definition in the front end function

```

o.Moo.monitor
\item \mtl{o.Moo.drawing.invert}
\item \mtl{o.GraphicOptions}

```

Listing 6.6: ???a completer???

- .DoParetoAnalysis
- .PlotPareto
- .PlotDecVar
- .PlotCorrelations
- .PlotMatrix

6.5.1 Restarting a failed optimization run

6.6 Recomputing a Pareto curve

(il n'y a rien dans la fonction 6.9???)

```

function o = DefineMooOptim(o)
% -----
% Definition of the optimisation problem
% Number of objectives [required].
o.Moo.nobjectives = 2 ;
% Number of maximal iterations [required].
o.Moo.max_evaluations = 30000;
% Number of clusters [required].
% A cluster is a subset of the Pareto population with similar values of the
% variables
o.Moo.nclusters = 4;
% Size of the initial population, i.e. number of initial points [required].
o.Moo.InitialPopulationSize = 500;
% -----
% Objectives definition
i=0;
% Definition of objectives [required]
i=i+1;
o.ObjectiveFunction(i).Model = {'AZEP'};
o.ObjectiveFunction(i).TagName = {'TotalPower'};
o.ObjectiveFunction(i).MinOrMax = {'max'};
o.ObjectiveFunction(i).DisplayName = {'System efficiency'};
i=i+1;
o.ObjectiveFunction(i).Model = {'AZEP'};
o.ObjectiveFunction(i).TagName = {'TotalCost'};
o.ObjectiveFunction(i).MinOrMax = {'min'};
o.ObjectiveFunction(i).DisplayName = {'System efficiency'};
% Define the decision variables [required].
i = 0;
i = i+1;
% -----
% Define the name of the model to whom the tag belongs [required].
% Ex.: o.Variables(i).ModelTagName = {'SNG'};
% o.Variables(i).ModelTagName = {''};
% Specify the name of the model tag [required].
% Ex.: o.Variables(i).TagName = {'F_101_P'};
% o.Variables(i).TagName = {''};
% Specify the unit of the model tag [required].
% Ex.: o.Variables(i).Unit = {'bar'};
% o.Variables(i).Unit = {''};
% Specify the name that is displayed in results analysis [required].
% Ex.: o.Variables(i).DisplayName = {'Reactor pressure'};
% o.Variables(i).DisplayName = {''};
% Specify the domain of the variable [required].
% Ex.: o.Variables(i).Limits = [1 20];
% o.Variables(i).Limits = [];
% Specify the type of variable [required].
% 1: integer, 0: continuous
% Ex.: % o.Variables(i).Is_integer = 0;
% o.Variables(i).Is_integer = ;
% Continue the list...
% i = i+1;
% o.Variables(i).ModelTagName = {''};
% o.Variables(i).DisplayName = {''};
% o.Variables(i).TagName = {''};
% o.Variables(i).Unit = {''};
% o.Variables(i).Limits = [];
% o.Variables(i).Is_integer = ;
% -----
% Definition the parameters for displaying
o.Moo.monitor = ...
34 {
    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_minimization_iteration' % minimization iteration display in the prompt [number
        of eval.]

```

```
% =====  
function o = DefineRestartMoo(o)  
% -----  
%   Definition of the results folder for restarting an optimisation  
% -----  
% Specify the name of the results directory  
o.Paths.MooResultsDirectoryName = {''};
```

Listing 6.8: Code matlab - Restart moo

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

Listing 6.9: Code matlab - Recompute a Pareto curve

Chapter 7

Multi-period problem definition

The present chapter explains how to define multi-period problems in *OSMOSE*. For the moment, it is only possible to define periods that have no links between them, and storage problem solving is not yet implemented. Therefore, multi-period problems are currently more equivalent to scenarios, and each period is simulated sequentially by *OSMOSE* independently of the others.

7.1 Constants definition

For any computation type (*OneRun*, *Sensi*, *Moo*, *Recompute*) constants can be defined for the required number of periods. These constants are tags of the models, taking a different value for each period defined. There are no limitations in the number of constants that can be defined in multi-period, but the number of periods has to be the same for every constant defined with the multi-period syntax. The syntax to be used is described by listing 7.1.

```
i=0;
i=i+1;
o.Constants(i).ModelTagName = {'cold_source'};
o.Constants(i).TagName = {'cold_t_in'};
p=0;
p=p+1;
o.Constants(i).Period(p).Value = 8;
p=p+1;
o.Constants(i).Period(p).Value = 10;
```

Listing 7.1: Constants definition in multi-period problems

7.2 Variables definition

It is possible as well possible to use multi-period variables to perform a multi-objective optimization (*Moo*), and to generate randomly a different value per period for each variable, with different limits for all the periods if necessary. The syntax, explained by listing 7.2, is quite similar to the one used for the constants.

It has to be noticed that one new variable is generated for each period, which multiplies the number of decision variables by the number of period for the optimization problem. For example, if 7 decision variables are defined for 3 periods, this is equivalent to 21 variables in a mono-period problem. The advantage of this approach is that it is more robust and easier to treat the results afterwards than generating randomly a vector of values. However, it requires a higher initial population and a higher number of total iterations to converge to a stable Pareto, and therefore more computing time.

```
i=i+1;
o.Variables(i).ModelTagName = {'orc_simple'};
o.Variables(i).TagName = {'orc_wf3_t'};
o.Variables(i).DisplayName = {'Evaporation temperature of working fluid'};
o.Variables(i).Unit = {'C'};
o.Variables(i).Is_integer = 0;
p=0;
p=p+1;
o.Variables(i).Period(p).Limits = [90 120];
p=p+1;
o.Variables(i).Period(p).Limits = [100 110];
```

Listing 7.2: Variables definition in multi-period problems

7.3 Post-multiperiod computation

If further calculations have to be made after all the periods have been simulated, it is possible to call a post-multiperiod matlab function, which has to be declared as illustrated by the example of the listing 7.3.

```
o.Model(i).PostMultiperiodMFunction = {'geotherm_Postcompute_Total'};
```

Listing 7.3: Definition of a post-multiperiod computation matlab function

Chapter 8

Results Analysis

8.1 Results structure

Each computation stores results in the `o.Results` structures. This structure holds all the results concerning the models, the variables, the constants, the objective functions and of all the interfaces called during the computation (ex: EI). The results structure is displayed in listing 8.1

```
%% Results structure o.Results
%
% indexes:
%   i : defines actual result
%   m : defines actual model
%   p : defines actual period

o.Results(i).Model(m).Variables
    .Tags      % tags computed at the end of period-dependent computation
    .TagsList  % list of tag names contained in Tags field. Used to accelerate
                tag recovering
    .Period(p).Tags      % period-dependent tags values
                        .EI
                        .Constants
                        .TagsList
    .Objectives % objective functions
    .Clusters  % appears only for some computation types
    .ModelsList % Contains the TagName of the models. For faster recovering
    .Period.EI % EI results referring to the whole problem
```

Listing 8.1: Results structure

8.2 Reporting

OSMOSE is able to create an automatic report of the computation. The report is a pdf file containing the most important details of the computation. Next paragraphs explain how to setup reporting and indicate where to find the report.

8.2.1 Configuring the frontend

To allow reporting, you must activate this option in the first part of the frontend, as shown in listing 8.2.

```
o.DoReport = 1; % activate reporting
```

Listing 8.2: Frontend activation of reporting

Several options are available to personalize the report. They can be set in the `defineReport` sub-function, as shown in listing 8.3.

```
% =====
function o = DefineReport(o)
% -----
% Definition of the options to produce a report of the computation
% -----

o.Report.Format = {'PDF'}; % PDF or HTML are available
o.Report.Name = {'Automatic_Report'}; % Name of the report file
o.Report.Title = {'AUTOGENERATED REPORT'}; % Title of the report
o.Report.Authors = {'Me, you and someone else'};
o.Report.Date = {datestr(now,'dd. mmmm, yyyy')};
```

Listing 8.3: Frontend options for reporting

Other options can be found in the `osmose_report_defaults.m` function.

8.2.2 Reporting results

The results of reporting are stored in your results folder. You will find two folders: `pictures` and `SRC`. The latter contains all the \LaTeX files used to build the report. You can modify them and recompile the report.

8.3 Pareto plots using OsmosePlots

Pareto analysis can be performed in the frame of *OSMOSE*. When using *OSMOSE* to perform two-objectives optimization, an interactive tool, *OsmosePlots*, is available to generate visualisations of the results. Different kind of plots can be generated using *OsmosePlots*:

- Plot of the Pareto curve
- Projection of variable values on an objective axis
- Correlation between two variables

8.3.1 Osmose data storage

Within *OSMOSE*, the values of model variables (decision variables as well as dependent variables) are transmitted and updated throughout the computation using a specific sub-structure called `Tag`. Each relevant model variable or parameter is associated to a tag.

During an optimization, the values of tags are used to perform computations and to transmit the objectives values to the optimizer. The tags are not stored during the evolution of the optimization. After optimization termination, the points composing the Pareto front can be recomputed. For each recomputed point, *OSMOSE* creates a folder where the user can decide to store specific files generated

by the model (for example composite curves plots). Moreover the values of all the tags are stored in a specific matlab structure that is used by *osmose_gui_plot* for Pareto analysis.

OSMOSE creates a results folder, *OSMOSE_results*, at the location of the front-end function. For each run of the problem, a numbered run folder is created *run_xxxx*. Computation results are stored into two matlab tables *set_o.mat* and *set_s.mat*. Figures and files are stored in a subfolder *OSMOSE_recompute* containing as many folders as there are recomputed points.

8.3.2 Calling OsmosePlots

OsmosePlots is a Graphical User Interface (gui) written in matlab. The corresponding functions within *OSMOSE* are stored into *osmose\trunk\gui*.

Generally, after a recomputation the plotter is automatically called by *OSMOSE*. You can however also access results by entering with Matlab into the run folder of your choice (folder containing *set_o.mat* and *set_s.mat*) and using the command *osmose_gui_plot* in the command line.

8.3.3 Using OsmosePlots

OsmosePlots uses information retrieved from the model tags to generate the required graphics. It also separated the points according to the cluster classification of *moo*.

Figure 8.3.3 shows the plotter interactive window, the areas outlined on the figure are:

- **a. Selection of the x axis** Here you can select which Tag is going to be used for the x axis. For commodity the two-objectives are separated outside the list. If the third button, "Another tag, select below" is checked, a scrolling list containing all the tags is displayed (as seen in Figure 8.3.3).
- **b. Selection of the y axis** One can select the y axis between all the tags using a scrolling menu.
- **c. Plot command button** After having selected x and y axis, push the plot button to obtain the graphic.
- **d. Plot area** Area containing the resulting plot.
- **e. Figure saving commands** Once a plot is generated, there is the possibility of saving the matlab figure. You can enter the name you want for your file and then press save. The plot is saved in a subfolder of the run folder called *OSMOSE_plots*.
- **f. Data cursor command** Allows enabling data cursor mode. If data cursor mode is enabled, clicking on a point in the plot area will display the point coordinates (see Figure 8.3.3)

How do I obtain the Pareto curve?

To obtain the pareto curve, select one of the two objectives in the x axis box. In the y axis scrolling menu select the other objective. Click Plot.

An example of Pareto plot is shown on Figure 8.3.3. The clusters identified by *moo* (in this case four) are displayed with different colors.

How do I analyse variables evolution along the Pareto curve?

The evolution of the values of one variable along the Pareto can be visualized by projecting the points on the plane formed by the axis of one of the objectives and the axis of the variable.

Select one objective in the x axis box. In the y axis scrolling menu select the tag corresponding to the variable of interest.

An example is shown on Figure 8.3.3.

Figure 8.1: Osmose plotter interactive window

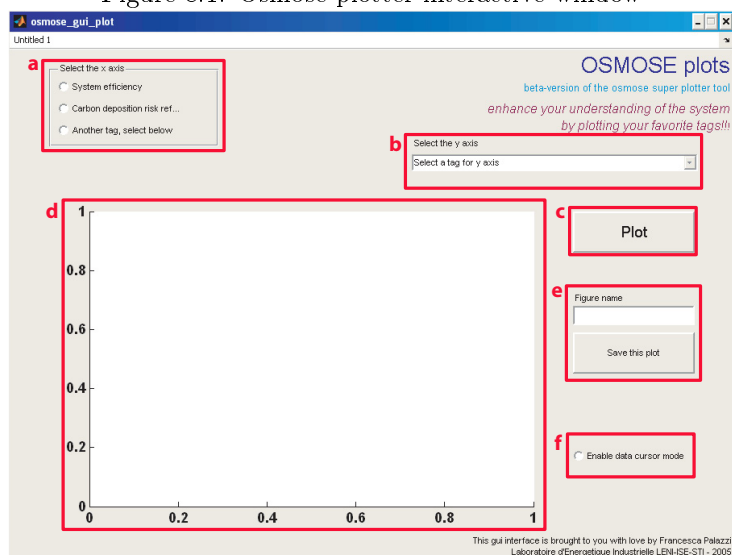


Figure 8.2: Menu for x axis selection among all model tags

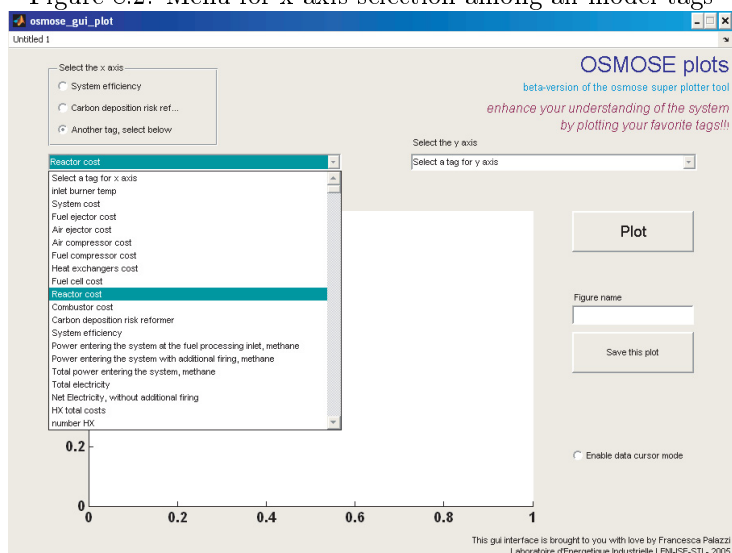


Figure 8.3: Obtain coordinates of a point using the cursor mode

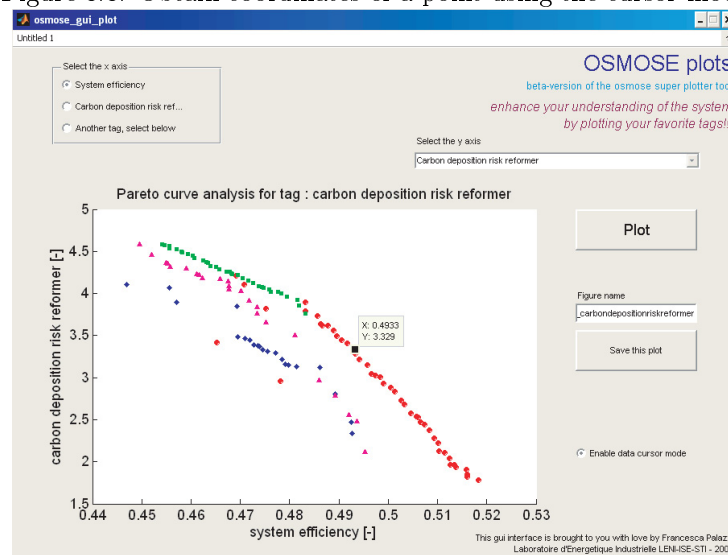


Figure 8.4: Pareto plot within OsmosePlots

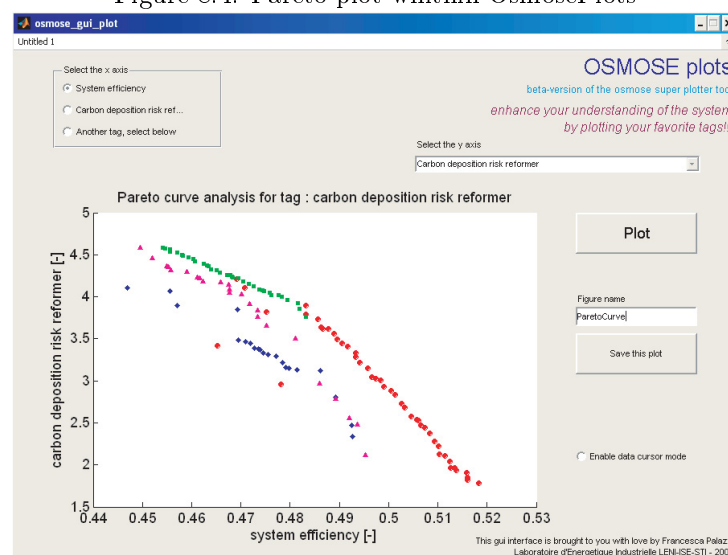
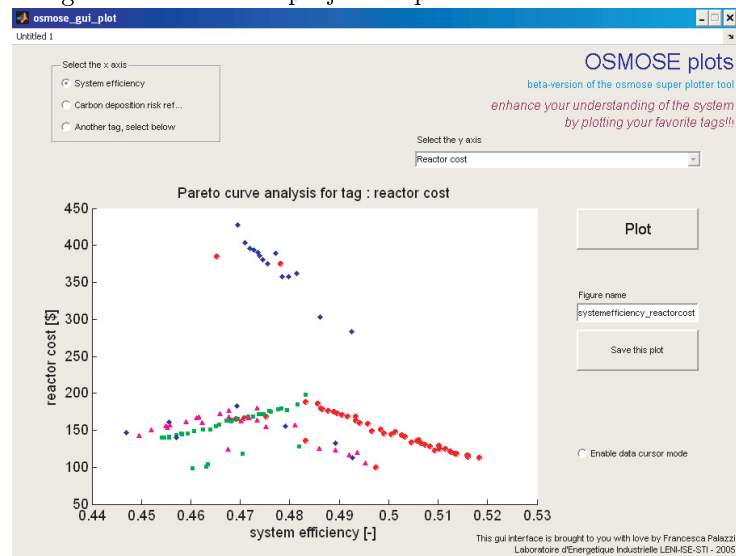


Figure 8.5: Variables projection plot within OsmosePlots



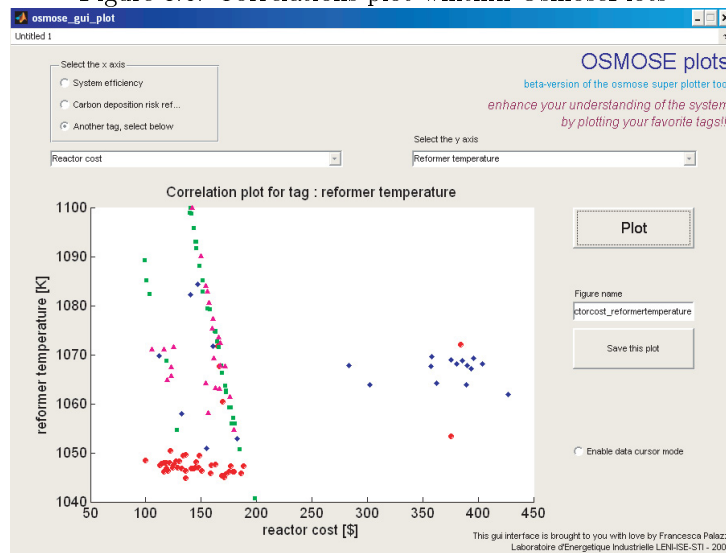
How can I have an idea of existing correlations between variables?

Existing correlation between tags can be visualized by plotting the points on a plane formed by the two concerned tags.

Select the first tag in the x scrolling menu, and the second tag in the y scrolling menu.

An example is shown on Figure 8.3.3. In this example, the green cluster of points shows a strong linear correlation.

Figure 8.6: Correlations plot within OsmosePlots

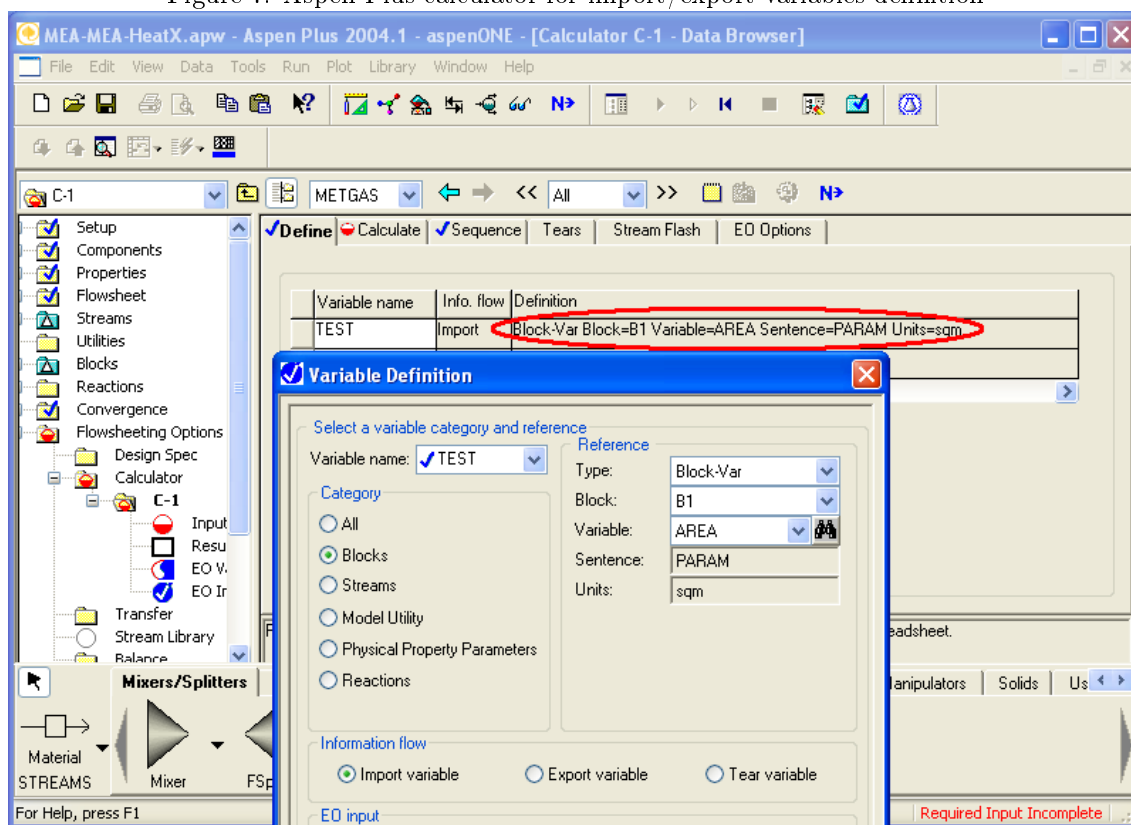


.1 Functions examples

.1.1 PreComputation example

.2 Aspenmodel

Figure 7: Aspen Plus calculator for import/export variables definition



```

function o = PreComputationExample(o)
% function o = PreComputationExample(o)
%
% OSMOSE models: This is an example of pre-computation function
%
% In a pre-computation function, all operations are possible on the structured variable o.
% Generally two actions need to be performed:
%     - definition of new tags
%     - selection of the external model to run
%
% The code below is an example on how you can perform these two operations,
% other ways are of course possible and allowed!
%
% AUTHOR: Francesca Palazzi novembre 2006
% -----

%% Local parameters: parameters used only in this function,
% their value is not transmitted to the tag structure
Pi = 3.14;
BigProd_LowerBound = 500;
LocalBakery_UpperBound = 10;

%% Use existing tags definition to create local matlab variables
% All the existing tags are loaded in the local workspace as variables
% by evaluating the expression ".TagName = .TagValue"
for i = 1:size(o.Model(o.ModelID).Tags,2)
    eval(sprintf('%s = %s;', char(o.Model(o.ModelID).Tags(i).TagName), num2str(o.Model(o.ModelID).
        Tags(i).Value)));
end

% In this example we assume that the Tags 'CakeRadius' and 'NbOfCakes' have been defined in the
    front-end
% with the piece of code above, the value of these tags
% have been allocated into local variables called 'CakeRadius' and 'NbOfCakes'

%% The variables retrieved from tags is used for computation
CakeSurface = Pi * CakeRadius^2;
TotalCakeSurface = NbOfCakes * CakeSurface;

%% Selection of the external model
% remark : All the models have to be defined in the 'o.Model(i).FileName' field. This is done
    in the front-end.

if TotalCakeSurface > BigProd_LowerBound
    o.Model(o.ModelID).FileSelected={'BigProductionChain.bls'};
elseif TotalCakeSurface < LocalBakery_UpperBound
    o.Model(o.ModelID).FileSelected={'LocalBakery.bls'};
else
    o.Model(o.ModelID).FileSelected={'CakeProductionChain.bls'};
end

%% New tags definition
% the new tags can be local variables, in that case one can organize information storage as
    follows:

nt = 0; %New tag counter, incremented for each new tag definition

% Define the names and units of the new tags
nt = nt+1;
Tag(nt).TagName = {'CakeSurface'};
Tag(nt).DisplayName = {'Surface of one cake'};
Tag(nt).Unit = {'m^2'};

nt = nt+1;
Tag(nt).TagName = {'TotalCakeSurface'};
Tag(nt).DisplayName = {'Total surface of cakes'};
Tag(nt).Unit = {'m^2'};

```

Bibliography

- [1] AMPL. <http://www.ampl.com/>.
- [2] Belsim. <http://www.belsim.com>.
- [3] R. Bolliger, D. Favrat, and F. Maréchal. Advanced Power Plant Design Methodology using Process Integration and Multi-Objective Thermo-Economic Optimisation. In *ECOS 2005, 18th International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems*, volume 2, pages 777–784, Trondheim, Norway, 2005.
- [4] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley & Sons, 2001.
- [5] G. Leyland. *Multi-objective optimisation applied to industrial energy problems*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2002.
- [6] F. Maréchal. Modelisation et optimisation des systèmes industriels, 2006. Lecture notes.
- [7] F. Maréchal, D. Favrat, F. Palazzi, and J. Godat. Thermo-economic modelling and optimization of fuel cell systems. In *Fuel Cell Research Symposium*, ETH Zürich, 2004.
- [8] F. Maréchal, D. Favrat, F. Palazzi, and J. Godat. Thermo-economic modelling and optimisation of fuel cell systems. *Fuel Cells- From Fundamentals to Systems*, 5(1):5–24, 2005.
- [9] MathWorks. <http://www.mathworks.com>.
- [10] A. Molyneaux. *A practical evolutionary method for the multi-objective optimisation of complex integrated energy systems including vehicle drivetrains*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2002.
- [11] F. Palazzi, N. Autissier, F. Maréchal, and J. Van herle. A Methodology for Thermo-Economic Modeling and Optimization of SOFC Systems. *Chemical Engineering Transactions*, 7:13–18, 2005.
- [12] F. Palazzi, D. Favrat, F. Maréchal, and J. Van herle. Energy Integration and System Modelling of Fuel Cell Systems. Technical report, CH-3003 Berne, Switzerland, 2004.
- [13] A. Plus. <http://www.aspentech.com/>.
- [14] B. K. R. Fourer, D. M. Gay. *Ampl, a modeling language for mathematical programming*, 2003.