

OSMOSE

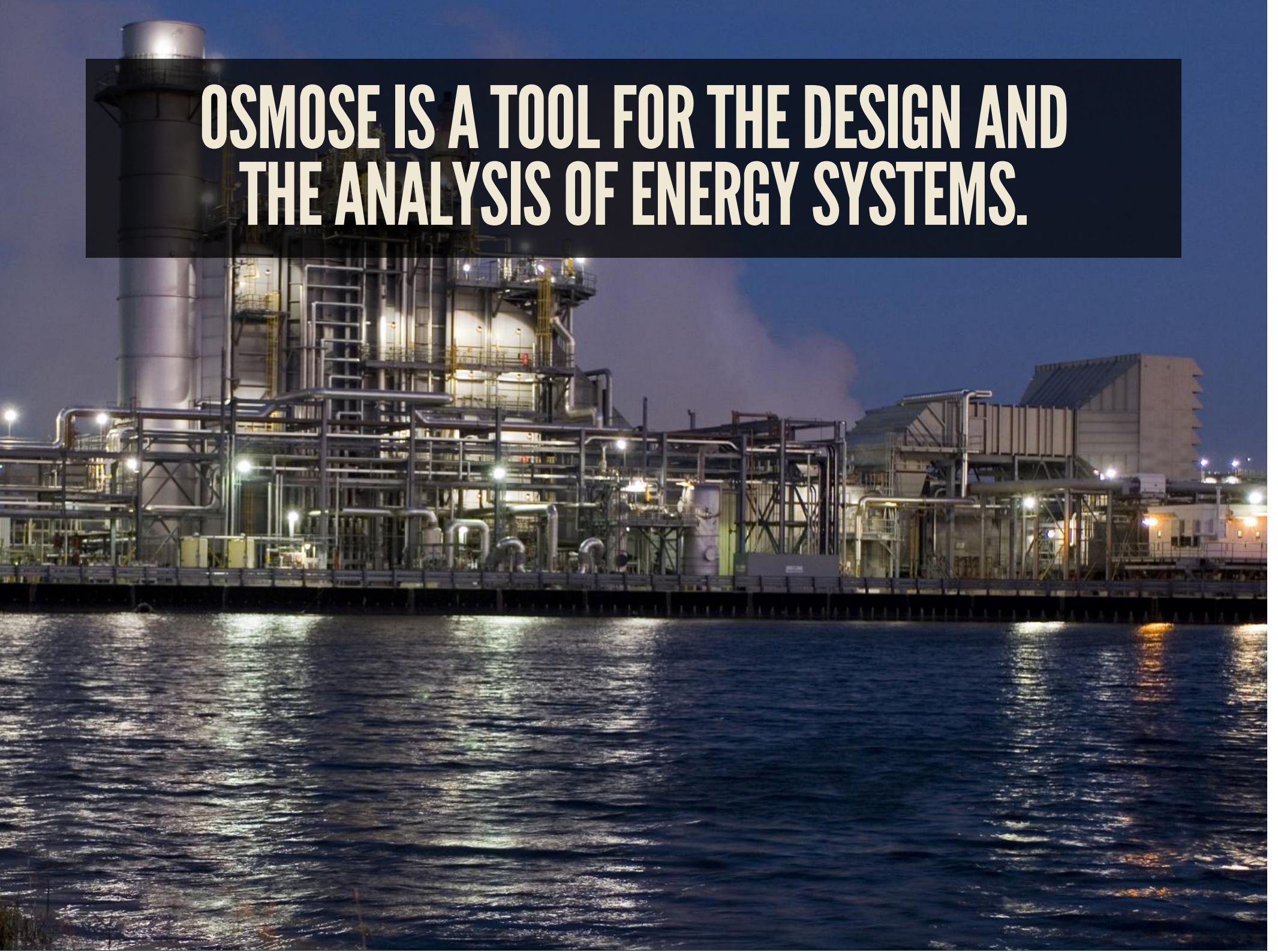
MULTI-OBJECTIVE OPTIMIZATION OF INTEGRATED ENERGY SYSTEMS



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Presented by Renaud Kern (Teti, Lausanne, Switzerland)

**OSMOSE IS A TOOL FOR THE DESIGN AND
THE ANALYSIS OF ENERGY SYSTEMS.**



PROJECT EXAMPLE

In projects/jam.lua file

```
-- osmose is also a luarock
local osmose = require 'osmose'

-- Create a project named 'Jam' of type 'MER' (Minimum Energy Requirement)
local project = osmose.Project('Jam', 'MER')

-- Load some models
project:load({cip="ET.Cip"}, {utilities="ET.generic_utilities"},
{cm1="ET.CookingMixing"}, {cm2="ET.CookingMixing", with='CM2_inputs.csv'})

-- Solve with GLPK (GNU Linear Programming Kit)
project:solve()

-- Do something with the results
project:postCompute('jam_postcompute')
```

Run with

```
lua projects/jam.lua
```

MODEL EXEMPLE 1/2

In ET/cip.lua file

```
-- Create a model
local lib = osmose.Model 'Cip'

-- Let's give some inputs in the model
lib.inputs = {
  -- Percentage of the mass flowrate of the raw water that is recovered from the process. (CW1)
  raw_water_rate = {default = 50, min = 50, max = 50, unit = '%m/m'},
  -- Mass flowrate of cleaning liquid distributed to the process. (CW1)
  distributed_water_flow = {default = 10, min = 5, max = 10, unit = 't/h'},
}

-- Let's define some output of the model
lib.outputs = {
  -- Mass flowrate of the raw water that is recovered from the process.
  raw_water_flow = {unit = 't/h', job = "(raw_water_rate/100) * distributed_water_flow"},
}
```

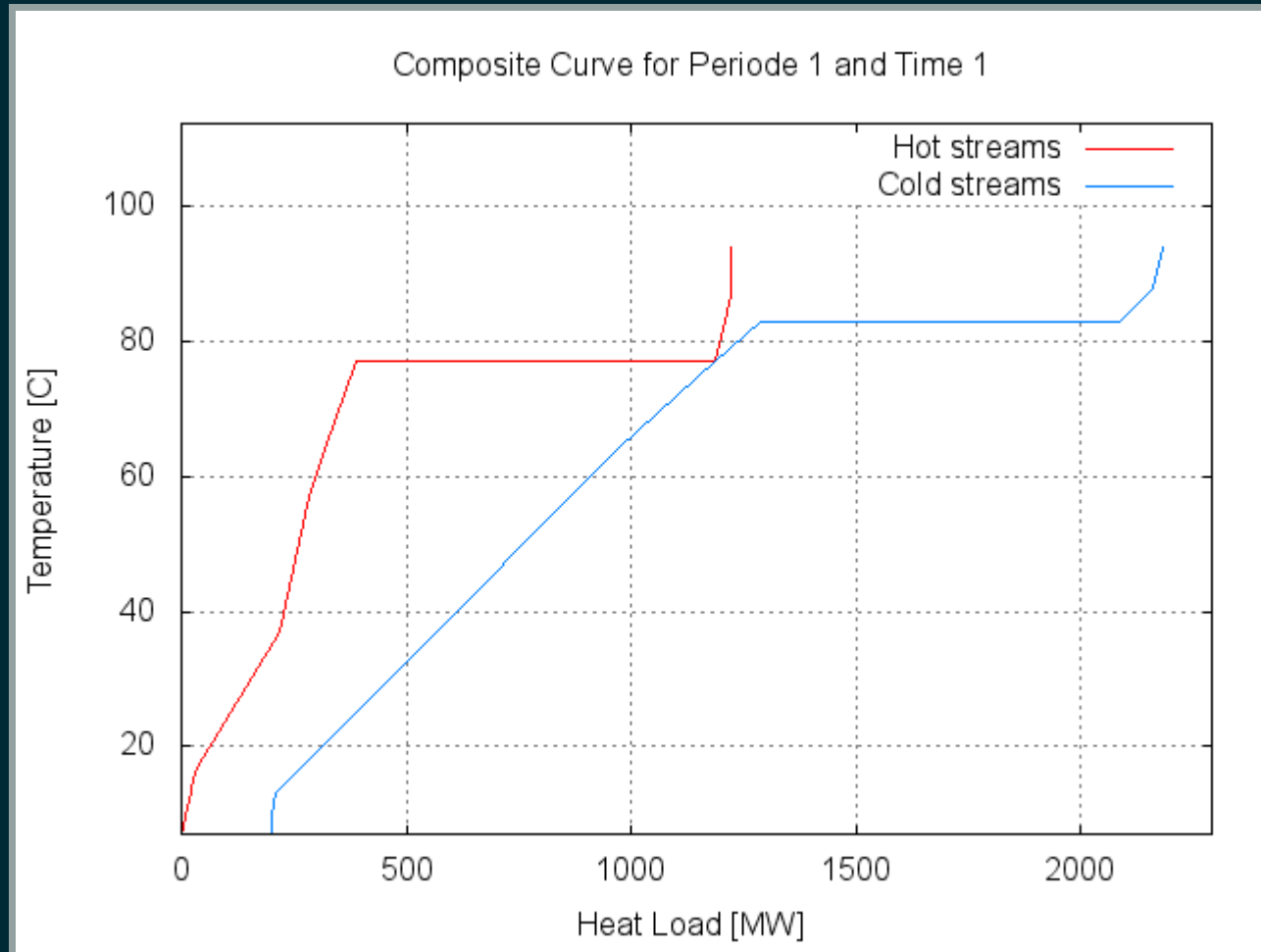
MODEL EXAMPLE 2/2

```
-- Add a unit named 'CipUnit' of type 'Process'.
-- Units define the smallest entity possible in the model.
lib:addUnit("CipUnit", {type = 'Process', addToProblem='j1'})

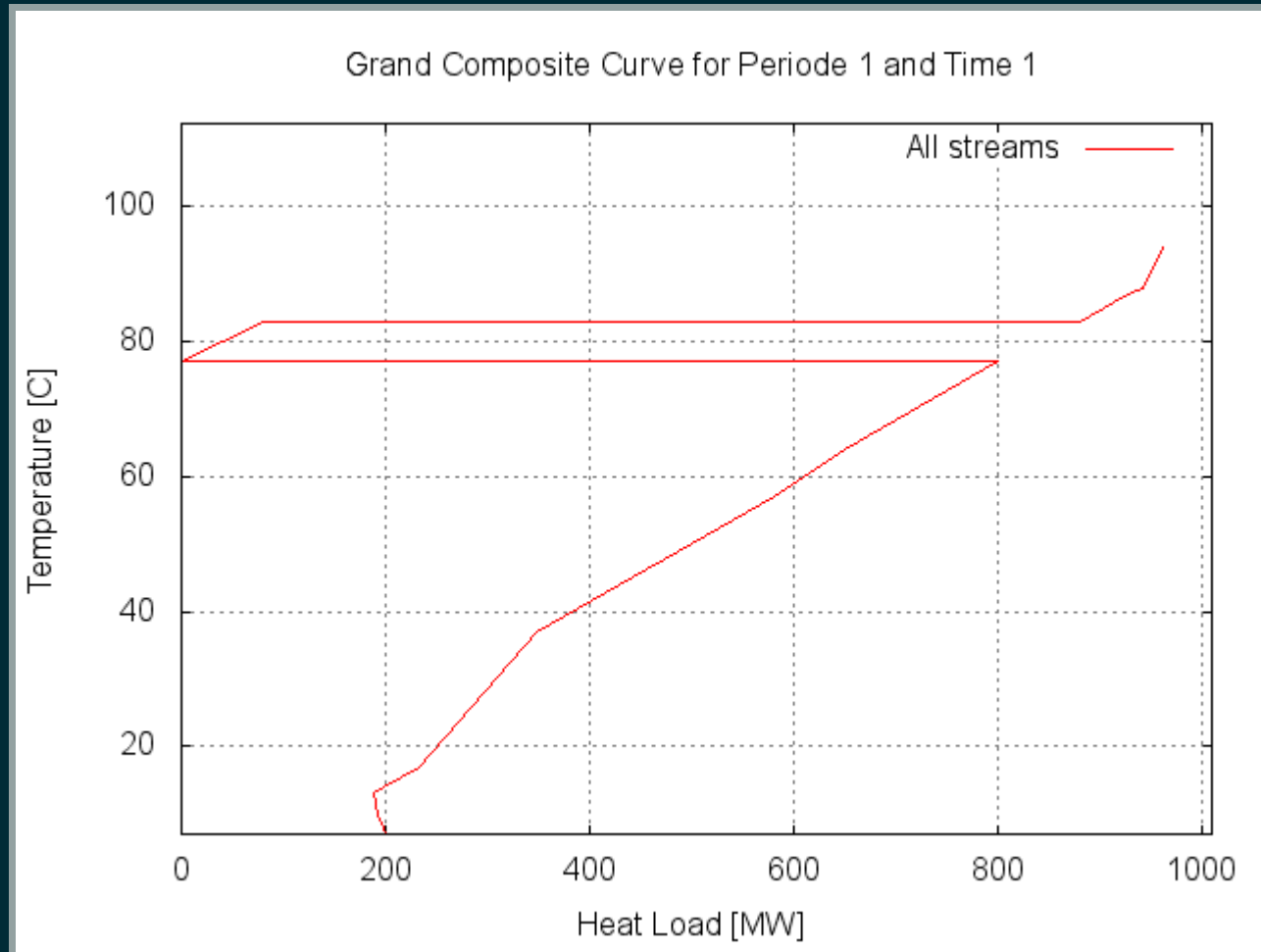
-- Add 3 streams to the unit.
-- Streams are supplied by heat transfert and are defined by a enthalpy-t
cip["CipUnit"]:addStreams({
cleaning_agent = qt{'cleaning_agent_temp', 0, 'tank_temp', 'cleaning_agent_
fresh_water = qt{'source_temp', 0, 'tank_temp', 'fresh_water_load', 3, 'wate
discharge = ht{{'return_temp'}, {'discharge_load'}, {'max_temp'}, {0}, { 3},
})

return lib
```

EXAMPLE OUTPUT 1



EXEMPLE OUTPUT 2



EXTERNAL TOOL 1

GNU LINEAR PROGRAMMING KIT (GLPK)

GLPK (GNU Linear Programming Kit)



[Introduction](#) | [Downloading](#) | [Documentation](#) | [Mailing Lists/Newsgroups](#) | [Request an Enhancement](#) | [Report a Bug](#) | [Maintainer](#)

[Introduction to GLPK](#)

The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

GLPK supports the *GNU MathProg modeling language*, which is a subset of the AMPL language.

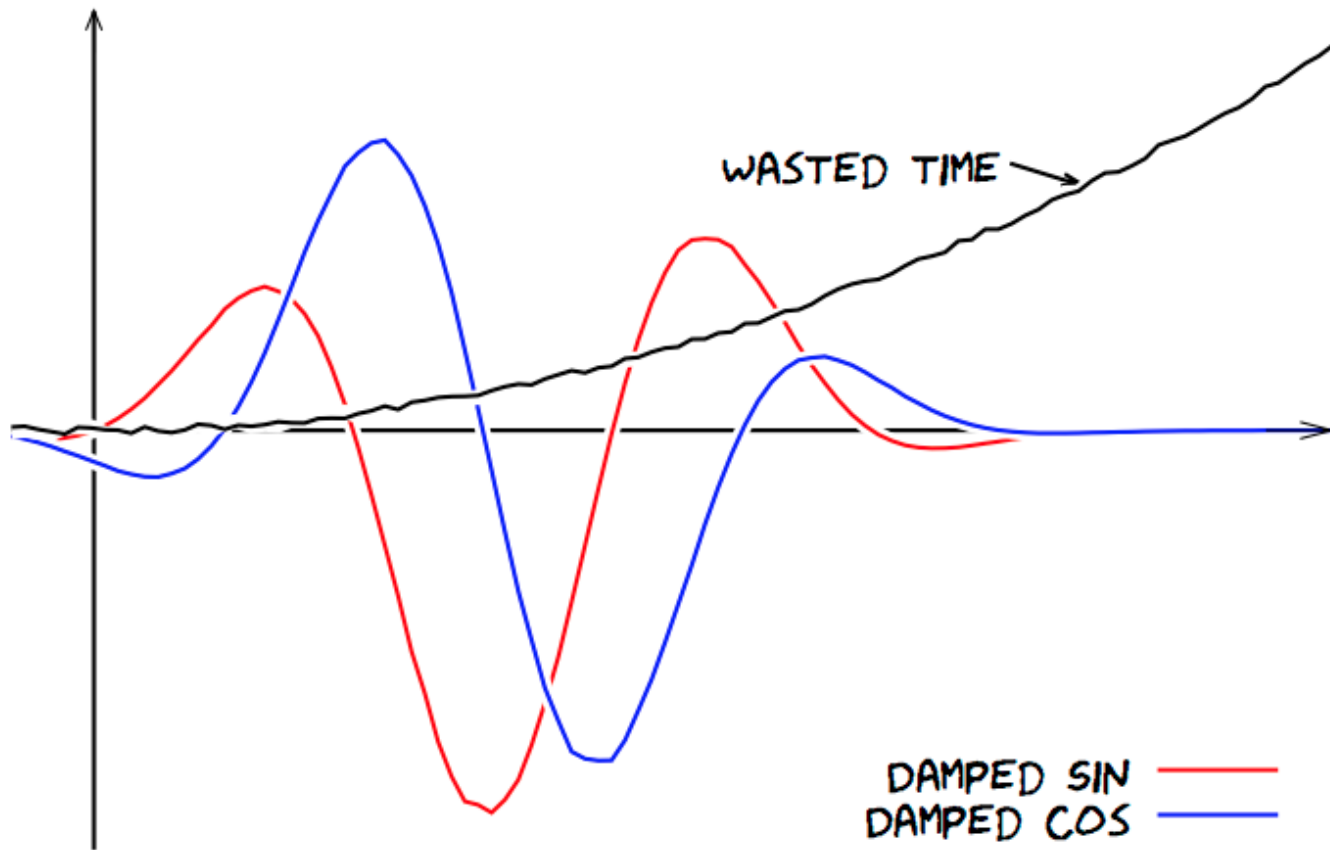
The GLPK package includes the following main components:

- primal and dual simplex methods
- primal-dual interior-point method
- branch-and-cut method
- translator for GNU MathProg
- application program interface (API)
- stand-alone LP/MIP solver

EXTERNAL TOOL 2

GNU PLOT

CHECK THIS OUT! XKCD IN GNU PLOT



EXTERNAL TOOL 3

DAKOTA

The DAKOTA Project

Large-Scale Engineering Optimization and Uncertainty Analysis



Sandia National Laboratories

Home

About

Search

DAKOTA@SNL

ABOUT DAKOTA

Applications

Research

Software Design

Team

Contributors

SEARCH DAKOTA

SITE:

Google Search

About - Software Design

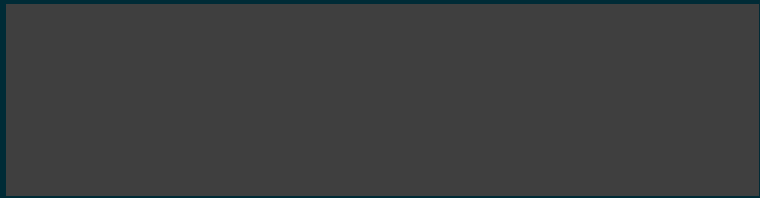
Detailed information on software components is available from the [Developers Manual](#). The following discussion provides a high level overview of these components.

Overview of DAKOTA: In the DAKOTA system, the *strategy* creates and manages *iterators* and *models*. A model contains a set of *variables*, an *interface*, and a set of *responses*, and an iterator operates on the model to map the variables into responses using the interface. In a DAKOTA input file, the user specifies these components through strategy, method, model, variables, interface, and responses keyword specifications.

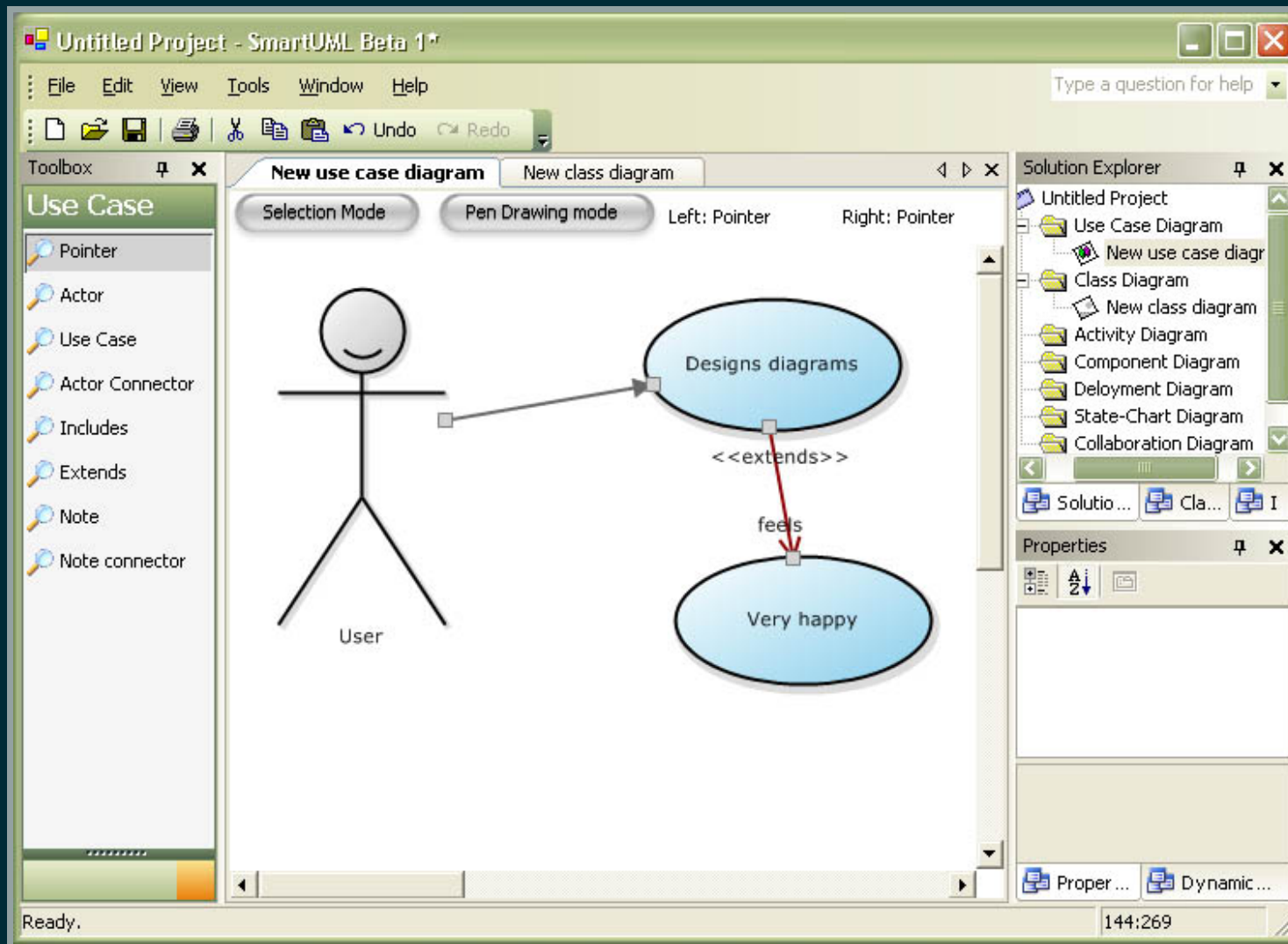
The Iterator class hierarchy of iteration methods provides a broad spectrum of capabilities, including optimization, parameter estimation, uncertainty quantification, design of computer experiments, and parametric analysis.

BUT WHERE ARE MY EGGS?





BUT WHERE IS MY UML?

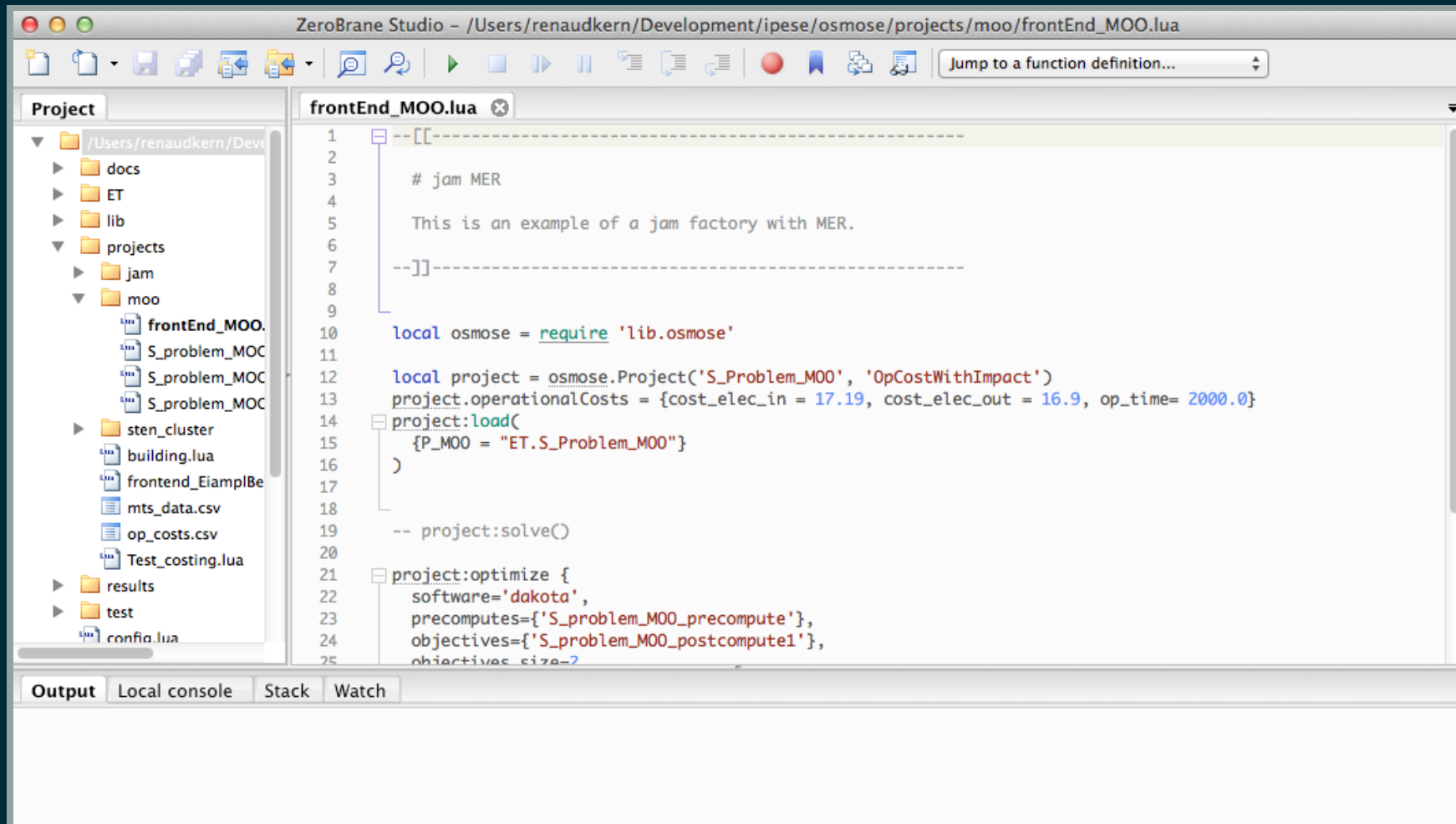


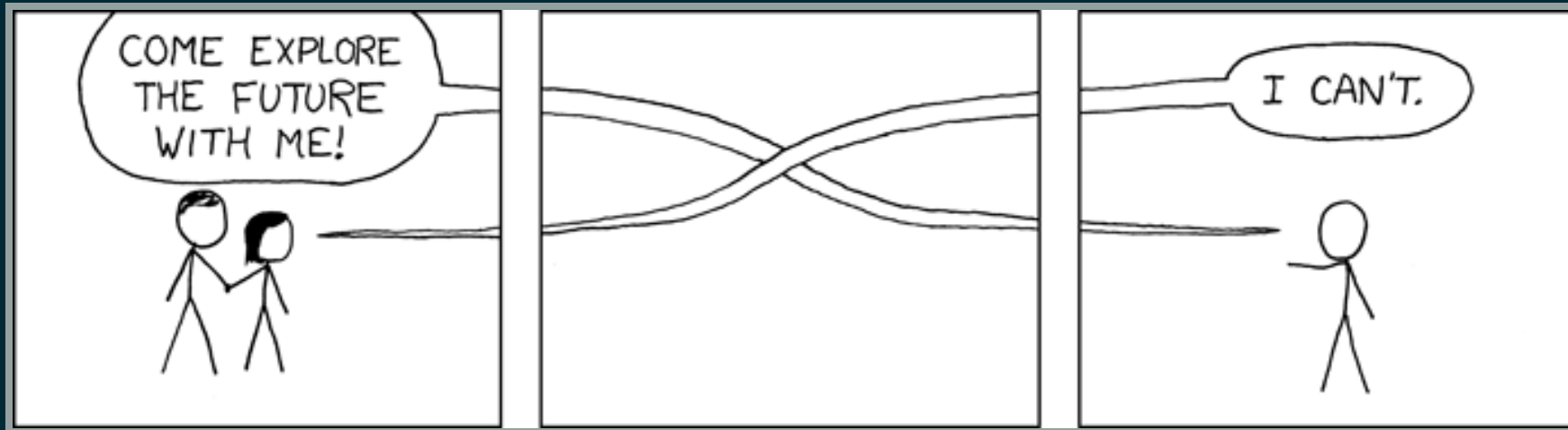
LUA FOR WINDOWS...

Libraries included

Library	Version	Description
Alien	0.5.0	Provides access to functions in an unknown or new .dll.
IUP	3.5.0	Light Portable Graphical User Interface library.
CD	5.4.1	Canvas Draw: A platform-independent graphic library.
IM	3.6.3	A toolkit for Digital Imaging.
Ex	Jan 07	Adds environment, file system, I/O (Locking and pipes), and process control.
LPeg	0.9	Pattern-matching library based on Parsing Expression Grammars (PEGs).
Lua-GD	2.9.33r2	Image manipulation library based on Thomas Boutell's GD library.
LuaCOM	1.4	Enable use & implementation of Microsoft's Component Object Model.
LuaCURL	1.0	Interface to Internet browsing capabilities based on the cURL library.
Date	2	Date and Time library for Lua.
LuaDoc	3.01	Documentation tool for Lua source code.
LuaExpat	1.1.0	Lua interface to XML Expat parsing library.
LuaFileSystem	1.4.2	Access the directory structure and file attributes.
LuaLogging	1.2.0	Logging features in Lua, based on log4j.
LuaProfiler	2.0.1	Time profiler designed to find bottlenecks in Lua programs.
LuaSocket	2.0.2	Lua interface to support HTTP,FTP,SMTP, MIME, URL & LTN12.
LuaSQL	2.1.1	Lua interface for PostgreSQL, ODBC, MySQL, SQLite, Oracle, and ADO dbms.
LuaUnit	2.0	Testing framework for Lua.
LuaZip	1.2.3	Read files from zip files.

WE NEED AN IDE





- Graphical programming
- HTTP Connection
- Parallel

THANK YOU

<http://leni.epfl.ch/en>

<https://github.com/ipesse/osmose>