Waqquas Bukhsh

OATS Documentation

Release 0.1

Jan, 2018

published using LaTeXin a design inspired by works of edward tufte.

http://www.wbukhsh.com/p/oats.html

First printing, July 2018

Contents

List of Figures

List of Tables

1 Introduction

Optimisation and analysis tool-kit for power systems (OATS) is a collection of optimisation models and Python scripts for analysis and solution of a range of power system analysis problems. OATS is specifically designed to be a portable and fully accessible simulation toolbox. All ingredients of OATS and its dependencies are open source. This flexibility means that OATS is a useful tool for power system community in academia and industry for analysing, extending and simulating important research questions.

OATS integrates a modelling language PYOMO[1] for describing optimization models. Using PYOMO's high-level representation of sets, variables and parameters the models in OATS are written in a way that are intuitive and easy to understand and extend. Moreover, it is important to note that OATS support the entire modelling life cycle: development, testing, deployment, and maintaining.

[1] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011

This user-manual give a brief introduction to OATS. The best way to understand the potential of this software is to install and run it. A range of test cases are provided in the test case library that could be used along with a range of different standard optimization models.

1.1 Architecture of OATS

The optimisation models in OATS are written in a algebraic modelling language (AML) called PYOMO. An algebraic modelling language provides a convenient interface between an optimisation model and a solver-where the problem is eventually solved. OATS also contains a set of Python scripts that handle the flow of information between OATS, PYOMO and a solver. Figure 1.1 presents the architecture of OATS. Optimisation models are written in PYOMO, test case data is specified in a spreadsheet. OATS passes user-specified optimisation model, a test case and a set of options to PYOMO. PYOMO creates an instance(a concrete model) using the information. The concrete model is then passed onto a solver in a form of a `.nl` binary file. The solver returns a solution in a binary format `.sol` that is processed by PYOMO. OATS reads the output and write it in a spreadsheet. The optimization problems can be warm-started meaning that a user specified (or output of another model) is used as a initial guess for gradient based solvers. This feature is particularly useful for running batch simulations where speed of convergence of a solution if important.

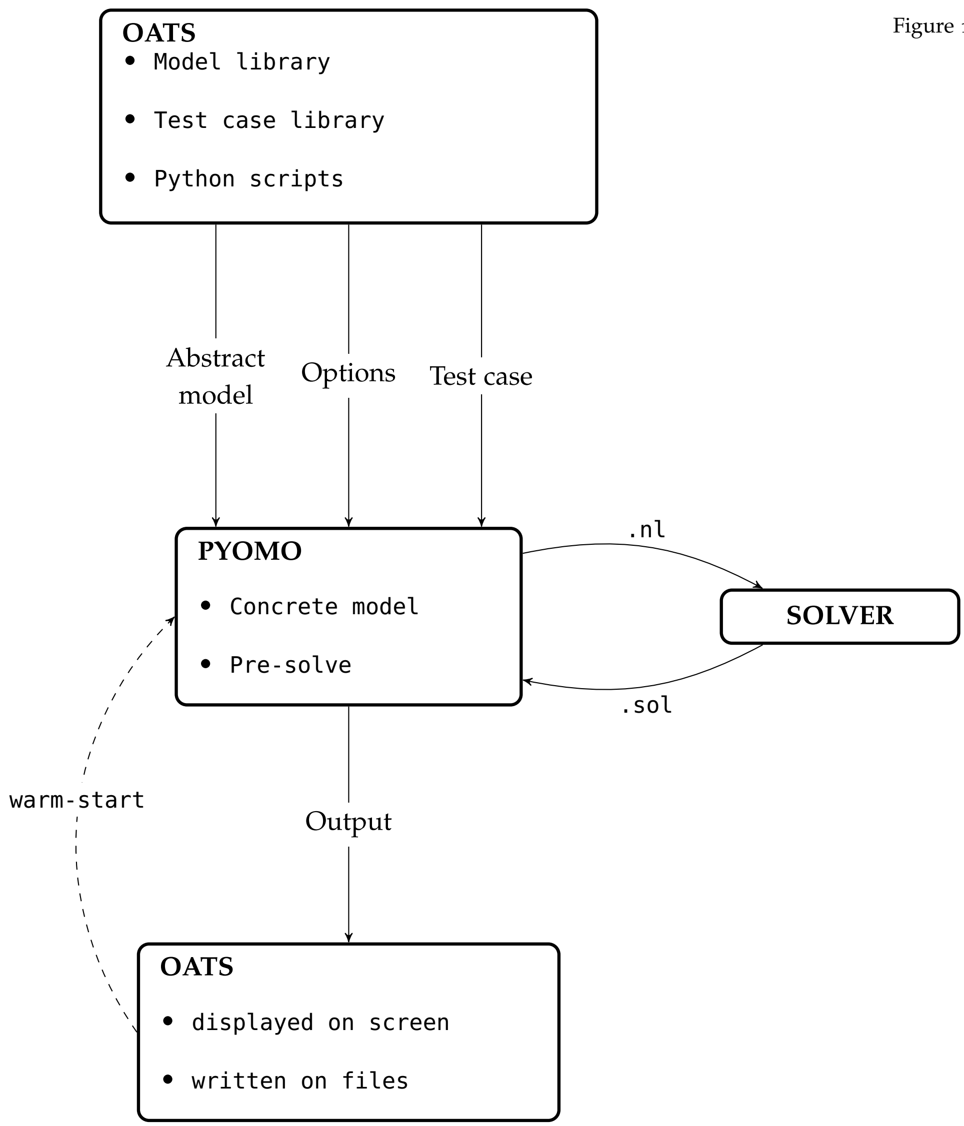

Figure 1.1: Architecture of OATS.

1.2 *Package Dependencies*

OATS has following dependencies on third part software;

- Python 2.7[2] (or higher) and following Python packages;
  - pandas

[2] Note that OATS currently does not support Python 3.

  - xlreader
  - numpy
- PYOMO
- A suitable linear and/or nonlinear solver[3].

[3] the models can be solved online on NEOS server
J. Czyzyk, M. P. Mesnier, and J. J. More. The neos server. *IEEE Computational Science and Engineering*, 5(3): 68–75, Jul 1998. ISSN 1070-9924. DOI: 10.1109/99.714603

1.3 *License*

OATS is distributed under the open source license GNU GPLv3. In simple terms, this is a copyleft license that requires anyone who distributes this code or a derivative work to make the source available under the same terms.

The more sophisticated extensions of OATS models and data are not distributed under this license and are reserved for consultancy work. Please contact us for details if you have a specific application in mind.

1.4 *Citation*

Citation of OATS is not required by the terms of the license, however, this is a only academic credit we receive from this work. We request that the publications derived from OATS acknowledge that by citing this user-manual and accompanying publication.

2 Installation of OATS and solvers

Installation of OATS is not a straight forward task. Be prepared for spending frustrating moments and getting stuck-really stuck. However, if things go according to the plan the effort will be worth it at the end. In this chapter, a step-by-step instructions for installation are provided for the three mostly used operating systems.

2.1 Installation of OATS and dependencies

The source code of OATS is distributed on the following github page:

`https://github.com/bukhsh/oats`

The first step is to install Python programming language. Currently OATS is only supported on Python 2. Please download[1] latest version of Python 2 for your operating system. OATS also require installation of following Python packages to function:

- `pandas`
- `pyomo`
- `xlsxwriter`
- `xlrd`
- `tabulate`

[1] Link to download Python: https://www.python.org/downloads/

2.1.1 Linux

Following instructions are for the users that are using Linux, or an OS that is derived from Linux *e.g.* Ubuntu, Anaconda. *etc.*

The first step is to install `pip` and `virtualenv` on your machine. Open a terminal window[2] and issue following commands.[3]

```
sudo apt-get install python-pip python-dev build-essential
sudo pip install --upgrade pip
sudo pip install --upgrade virtualenv
```

[2] `Ctrl+T` normally opens a terminal window.

[3] `sudo` command will prompt a user to enter the system's password.

Next step is to install PYOMO. Issue the following command in terminal window to install PYOMO.[4]

```
sudo pip install pyomo
```

[4] Installation instructions for PYOMO can also be found on it's own website: http://www.pyomo.org/installation/

Install conditional dependencies of OATS by issuing following command;

```
sudo pip install pyomo.extras
```

Install additional packages for OATS by issuing following commands:

```
sudo pip install xlrd
sudo pip install xlsxwriter
sudo pip install pandas
sudo pip install tabulate
sudo pip install graphviz
```

2.1.2 *Windows*

After installation of Python, it is recommended that Python is added to the windows environment variables. The exact steps of doing this depends on the version of Windows.

Open a command prompt.[5] You would need to change directory into location of Python. Most probably the location would be `C:\Python27`

[5] Win+R is short cut for opening 'run' function in Windows. Type 'cmd' to launch command prompt.

Change directory into the folder `Scripts`. We want to be in the folder: `C:\Python27`

This folder contains an executable `pip.exe` and will allows us to download all the dependencies of OATS. Now rest of the steps are similar to installation to Linux. User should issue the following commands one by one to install all the dependencies of OATS.

```
pip install pyomo
pip install pyomo.extras
pip install xlrd
pip install xlsxwriter
pip install pandas
pip install tabulate
pip install graphviz
```

2.1.3 *MAC OSx*

Python comes installed along with OS X, although please make sure that you have latest version of Python 2.

Open a terminal window.[6] `pip` is a preferred package management system that we shall use to install all the dependencies of OATS. Issue following commands in the terminal to install all the required packages.

```
pip install pyomo
pip install pyomo.extras
pip install xlrd
pip install xlsxwriter
pip install pandas
pip install tabulate
pip install graphviz
```

[6] Cmd+T is a short cut for opening terminal window in Mac

2.2 *Installing a local solver*

Installation of a local solver is required to use OATS. The choice of the solver depends on the type of problem that a user wants to solve using OATS. For example, DC-OPF problem is a linear programming problem and can be solved using IBM's solver `CPLEX`[7]. AC-OPF is a nonlinear optimization problem and can be solved by a NLP solver `ipopt`.[8]

[7] `CPLEX` can be downloaded from IBM academic initiative webpage

[8] The source code for `ipopt` can be downloaded from http://www.coin-or.org/download/source/Ipopt

The optimization problems in power systems can be broadly classified into following four main categories:

- Linear programming (LP)
- Nonlinear programming (NLP)
- Mixed integer programming problem (MILP)
- Mixed integer nonlinear programming problem (MINLP)

The above classification is important because the solvers for solving optimization problems are tailored to a particular type of problem. Table 2.1 gives the classification of the optimization problems implemented in OATS.

Model	Type
DC-Load flow	LP
AC-Load flow	NLP
DC-optimal power flow	LP
AC-optimal power flow	NLP
Unit commitment problem	MILP
Security constrained optimal power flow	LP

Table 2.1: Classification of optimization problems implemented in OATS.

2.3 *Testing OATS*

`run_file.py` is the top-level file in OATS. All the settings related to model and test case are specified in this file. The default model is `DC-OPF` and default test case is a 9-bus network. After installation of OATS and all its dependencies, open a terminal window and change directory into where OATS is installed. Issue the following command:

```
python runfile.py
```

```
Number of Iterations....: 25

                                   (scaled)                 (unscaled)
Objective...............:   1.8317835013542672e-01    6.0998390595097095e+04
Dual infeasibility......:   2.8444423032196249e-14    9.4719928697213512e-09
Constraint violation....:   1.7763568394002505e-15    1.7763568394002505e-15
Complementarity.........:   3.2000572592662928e-11    1.0656190673356755e-05
Overall NLP error.......:   3.2000572592662928e-11    1.0656190673356755e-05


Number of objective function evaluations             = 30
Number of objective gradient evaluations             = 26
Number of equality constraint evaluations            = 30
Number of inequality constraint evaluations          = 30
Number of equality constraint Jacobian evaluations   = 26
Number of inequality constraint Jacobian evaluations = 26
Number of Lagrangian Hessian evaluations             = 25
Total CPU secs in IPOPT (w/o function evaluations)   =      0.031
Total CPU secs in NLP function evaluations           =      0.001

EXIT: Optimal Solution Found.
========================

 Output from the OATS
========================
------Solver Message------

- Status: ok
  Message: Ipopt 3.12.4\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.0425460338593

--------------------------
Optimization Converged!
Cost of the objective function: 60998.3905953
***********

 Summary
***********
+--------------------------------+-------------------------+---------------+
|   Conventional generation (MW) |   Wind generation (MW)  |   Demand (MW) |
+================================+=========================+===============+
|                           2850 |                       0 |          2850 |
+--------------------------------+-------------------------+---------------+
```

Figure 2.1: An example of OATS output on a Python command terminal.

3 Models and test cases

This chapter provides the reader with an introduction to the optimization models that are implemented in OATS. Note that the mathematical details of the models are omitted from this user manual for the sake of brevity. The reader is, where possible, referred to suitable textbooks and publications for the technical material.

ID in OATS	Model name
`DCLF`	DC load flow
`ACLF`	AC load flow
`DCOPF`	DC optimal power flow
`ACOPF`	AC optimal power flow
`UC`	Unit commitment problem
`SCOPF`	Security constrained optimal power flow

Table 3.1: Models available in OATS.

3.1 Unit commitment problem

Unit commitment is the problem of determining the least cost schedule of generating units subject to power balance and network constraints. In OATS, the unit commitment problem is modelled as a mixed integer linear programming problem. The objective function is to minimize the total cost of generation over a given time horizon. The constraints in each step are of power balance, restrictions on ramp rates, zonal net transfer limits and generation limits.

3.2 Optimal power flow

Optimal power flow problem (OPF) is a well studied optimization problem in power systems. The objective of OPF is to find a steady state operating point that minimizes the cost of electric power generation while satisfying operating constraints and meeting demand. The problem can be formulated in various ways [1].

[1] Anya Castillo Mary B Cain, Richard P. O'Neill. History of optimal power flow and formulations. Technical report, Federal Regulatory Energy Commission, 2013

3.3 Security constrained optimal power flow

Secure operation of a power system requires that no breach of operating standards take place following a credible contingency. This is achieved by solved a security constrained optimal power flow. A DC version of SCOPF is implemented in OATS. A set of credible contingencies can be specified in the test case. The credible contingencies in OATS are outages of single circuits[2], transformers or generating units.

[2] double circuit outages will be included in next release of OATS

3.4 Load flow analysis

The load flow problem can be stated as: for a given power network, with known complex power loads, and a set of specifications on power generation and voltages, determine bus voltages, any unspecified generation set point and finally the complex power flow in the network components. Load flow (or power flow) problem forms the core of power system analysis. This problem is at the heart of system planning, operation, contingency analysis and the implementation of real-time monitoring systems.

Load flow analysis is commonly used for following applications:

- Identify real and reactive power flow
- Identify proper transformer tap settings
- Identify transformer and circuit loadings
- Contingency analysis

Generator type	
1=	PV bus
2=	Distributed slack bus
3=	Reference bus

Table 3.2: Generator types in OATS.

Transformer type	
1=	2-winding transformer
2=	Tap-changing transformer

Table 3.3: Transformer types in OATS.

3.4.1 OATS implementation of the load flow problem

The load flow problem in OATS is solved as a constrained OPF problem. The fixed parameters of PV, PQ and Vδ buses are modelled using hard constraints.

3.4.2 *Distributed slack*

The load flow problem in OATS allow a user to model a distributed slack. The user can specify the number of slack buses in a system by changing the generator type from '1' to '2'.

3.4.3 *Tap-changing transformer*

OATS allow a user to determine tap setting of the transformers connecting a high voltage bus to a lower voltage bus. The tap-changing transformers can be specified using '2' in the type field of the transformers. The target voltage at the lower-voltage side is specified in column `VM` in the `bus` sheet. The turn ratios are determined at the high-voltage side of the transformer.

3.5 *Test cases*

OATS test case library includes a number of test cases. The test cases are written in a spread sheet format (`.xlsx`). Open source software are available to edit this type of files. All Matpower test cases can be converted into OATS equivalent test cases using a filter provided with OATS. Table 3.4 list a number of test cases that are included with the test case library of OATS.

Test case	Comment
GBReduced29	29-bus reduced network of Great Britain
SWNetwork	35-bus model of the South-West Peninsula of Great Britain
Matpower Cases	A Python filter is provided that can convert a Matpower test case into an equivalent OATS test case

Table 3.4: Test networks included with the release of OATS.

4 Contributing and supporting

OATS is an open source project maintained by a team of researchers at the University of Strathclyde. Over the last three years[1], we have spent more than 500 developer-hours building OATS. This software has been extensively used by students for doing their projects and researchers working on EU and UK projects.

[1] 2015-2018

A great way to support OATS is to cite the software if it has been useful in your work. Your citations will help us to build a case for obtained funding from research councils. Another great way to support OATS is to contribute to its development. In the following, some possible ways to contribute to the software are provided.

4.1 Software developers

If you would like to use OATS for a specific purpose then fork the OATS repo on GitHub page. This will help us to track the development of OATS and possibly extend our help in any extensions that you are working on.

4.2 Data users

We are keen to include more realistic data set in OATS test case library. All submission to OATS will be credited. There is no restriction on data format. We shall do data processing to make your data coherent with OATS format. Please make sure that your submission does not include copyrighted or propriety material.

Create an issue on OATS GitHub page if you wish to submit a data set. We will arrange a way for you to send a submission offline.

4.3 Project managers

Get in touch if you are thinking of using OATS in your group. We can arrange a training workshop to facilitate this. We are also interested in engaging with industry for consultancy work that may include more sophisticated extensions of OATS.

Test case format

Test cases in OATS are written in the spreadsheet (`xlsx`) file format which is a file extension of a Microsoft Excel Open XML format spreadsheet file. Many open source software are available to manipulate this file format. OATS uses Python package `pandas` to read and write the `xlsx` file formats. It is important the test case file follows a particular format, including all the headers are in correct sheets.

An OATS test case contains following 10 sheets:

Sheet name	Description
`bus`	Bus (node) data
`demand`	Demand data
`branch`	Transmission line data
`transformer`	Transformer data
`baseMVA`	baseMVA for conversion into p.u. system
`wind`	Wind generators
`generator`	Generation data
`shunt`	Shunt data
`zone`	Zonal data (only for running unit commitment problem)
`zonalNTC`	Transmission constraints between zones (only for running unit commitment problem)
`timeseries`	Time-series data (only for running unit commitment problem)

Table 1: Details of sheets in a OATS test case.

Table 2: Busbar data in OATS.

Name	Data Type	Description
`name`	integer or string	unique name of a busbar
`baseKV`	integer	bus base voltage
`zone`	integer or string	zone number
`VM`	float	voltage magnitude (p.u.)
`VA`	float	voltage angle (degrees)
`VNLB`	float	normal voltage magnitude lower bound (p.u.)
`VNUB`	float	normal voltage magnitude upper bound (p.u.)
`VELB`[1]	float	emergency voltage magnitude lower bound (p.u.)
`VEUB`[1]	float	emergency voltage magnitude lower bound (p.u.)

[1] Emergency voltage bounds are not used in this release of OATS.

Table 3: Demand data in OATS.

Name	Data Type	Description
`name`	integer or string	unique name of the demand
`busname`	integer or string	unique name of the busbar where demand is connected
`PD`	float	real power demand (MW)
`QD`	float	reactive power demand (Mvar)
`stat`	integer	status of the demand (0=out of service, otherwise=in service)
`VOLL`	float	value of lost load (£/MWh)

Name	Data Type	Description
`name`	integer or string	unique name of the transmission line
`from_bus`	integer	from bus name
`to_bus`	integer	to bus name
`stat`	integer	status of a branch (0=out of service, otherwise=in service)
`r`	float	resistance of a transmission line (p.u.)
`x`	float	reactance of a transmission line (p.u.)
`b`	float	susceptance of a transmission line (p.u.)
`rateC`	float	continuous line rating (MVA)
`rateS`	float	short term line rating (MVA)
`angLB`	float	lower bound of angle difference (degrees)
`angUB`	float	upper bound of angle difference (degrees)
`contingency`	integer	contingency flag (0=not considered, otherwise=considered)

Table 4: Transmission line data in OATS.

Table 5: Transformer data in OATS.

Name	Data Type	Description
name	integer or string	unique name of the transformer
from_bus	integer	from bus name
to_bus	integer	to bus name
stat	integer	status of a branch (0=out of service, otherwise=in service)
type	integer	transformer type (1=2-winding transformer, 2=tap-changing transformer)
r	float	resistance of a transmission line (p.u.)
x	float	reactance of a transmission line (p.u.)
rateC	float	continuous line rating (MVA)
rateS	float	short term line rating (MVA)
angLB	float	lower bound of angle difference (degrees)
angUB	float	upper bound of angle difference (degrees)
phaseshift	float	phase shift angle (degrees)
tap	float	tap ratio
contingency	integer	contingency flag (0=not considered, otherwise=considered)

Name	Data Type	Description
`name`	integer or string	unique name of the generator
`busname`	integer or string	name of the bus where the generator is connected
`stat`	integer	status of a generator (0=out of service, otherwise=in service)
`PG`	float	initial set point of real power output (MW)
`QG`	float	initial set point of reactive power output (Mvar)
`PGLB`	float	lower bound on real power generation (MW)
`PGUB`	float	upper bound on real power generation (MW)
`QGLB`	float	lower bound on reactive power generation (Mvar)
`QGUB`	float	upper bound on reactive power generation (Mvar)
`VS`	float	Voltage set point at the node (p.u.)
`contingency`	integer	contingency flag (0=not considered, otherwise=considered)

Table 6: Wind generation data in OATS.

Table 7: Generation data in OATS.

Name	Data Type	Description
`name`	integer or string	unique name of the generator
`busname`	integer or string	name of the bus where the generator is connected
`stat`	integer	status of a generator (0=out of service, otherwise=in service)
`type`	integer	generator type (1=PV, 2=Distributed slack, 3=reference bus)
`PG`	float	initial set point of real power output (MW)
`QG`	float	initial set point of reactive power output (Mvar)
`PGLB`	float	lower bound on real power generation (MW)
`PGUB`	float	upper bound on real power generation (MW)
`QGLB`	float	lower bound on reactive power generation (Mvar)
`QGUB`	float	upper bound on reactive power generation (Mvar)
`VS`	float	Voltage set point at the node (p.u.)
`rampdown`	float	ramp down rate (MW/hr)
`rampup`	float	ramp up rate (MW/hr)
`mindowntime`	positive integer	minimum down time of a generator (hr)
`minuptime`	positive integer	minimum up time of a generator (hr)
`contingency`	integer	contingency flag (0=not considered, otherwise=considered)
`costc2`	float	coefficient of quadratic component of objective function
`costc1`	float	coefficient of linear component of objective function
`costc0`	float	constant padding to the objective function
`shutdown`	float	shut down cost of the generator
`startup`	float	start up of the generator

Name	Data Type	Description
`name`	integer or string	unique name of the shunt
`busname`	integer or string	unique name of the busbar where demand is connected
`GL`	float	shunt conductance (MW demanded at V =1.0 p.u.)
`BL`	float	shunt susceptance (Mvar injected at V=1.0 p.u.)
`stat`	integer	status of the shunt (0=out of service, otherwise=in service)

Table 8: Shunt data in OATS.

Bibliography

J. Czyzyk, M. P. Mesnier, and J. J. More. The neos server. *IEEE Computational Science and Engineering*, 5(3):68–75, Jul 1998. ISSN 1070-9924. DOI: 10.1109/99.714603.

William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

Anya Castillo Mary B Cain, Richard P. O'Neill. History of optimal power flow and formulations. Technical report, Federal Regulatory Energy Commission, 2013.

Index