
oats Documentation

Release 1.0.2

Waqquas Bukhsh

Jan 28, 2020

Contents

1	Introduction to OATS	3
1.1	Architecture	3
1.2	Citation	4
1.3	License	4
2	Installation	5
2.1	OATS	5
2.2	Solvers	5
3	Quick Start	9
3.1	Solving a problem in OATS	9
3.2	Output	11
4	Models	13
4.1	Load flow problem	13
4.2	Optimal power flow problem	14
4.3	Security constrained optimal power flow problem	16
4.4	Unit Commitment problem	17
5	Data format	19
5.1	OATS data template	19
5.2	Description of sheets in the OATS data format	19
5.3	Filter Matpower2Oats	24
6	Extending models in OATS	25
6.1	Adding variables	25
6.2	Adding parameters	25
6.3	Modifying objective function	26
6.4	Adding/Modifying constraints	26
6.5	Solving user defined models	27
7	Examples	29
7.1	Voltage Dependent Loads	29
7.2	Modelling Storage in OATS	30
7.3	Optimising Reactive Power Margin	30
7.4	Optimising Transformers Tap Settings	30

Python Module Index	31
Index	33



Optimisation and Analysis Toolbox for power Systems (OATS) is a high-level modelling and simulation tool for power system analysis, developed at the University of Strathclyde.

OATS is a collection of optimisation models and Python scripts for analysis and solution of a range of power system analysis problems.

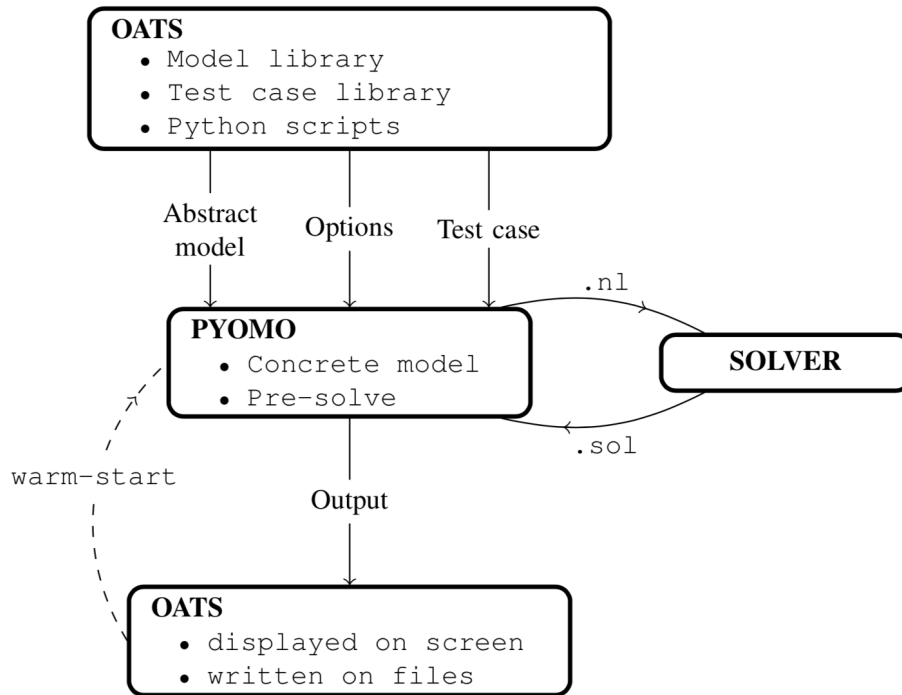
OATS integrates a modelling language PYOMO for describing optimisation models. Using PYOMO's high-level representation of sets, variables and parameters the models in OATS are written in a way that are intuitive and easy to understand and extend.

Introduction to OATS

Optimisation and Analysis Toolbox for power Systems (OATS) is a high-level modelling and simulation tool for power system analysis, developed at the University of Strathclyde. OATS is a collection of optimisation models and Python scripts for solving and analysing a range of power system analysis problems. These problems include: load flow, optimal power flow (AC, DC and security constrained) and unit commitment. OATS can be extended to solve more bespoke problems such as balancing markets, multi-period, and stochastic optimisation.

1.1 Architecture

The optimisation models in OATS are written in an 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.



In OATS the optimisation models are written in PYOMO, the test case data is specified using 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). The concrete model is then passed onto a solver in the 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 optimisation problems can be warm-started meaning that a user-specified (or output of another model) is used as an initial guess for gradient-based solvers. This feature is particularly useful for running batch simulations where the speed of convergence of a solution is important.

1.2 Citation

Waqquas Bukhsh. (2019, September 5). bukhsh/oats: OATS release (Version v1.0.1). Zenodo. <http://doi.org/10.5281/zenodo.3387594>.

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.

See the license file [here](#) for more details.

2.1 OATS

OATS can be installed using pip;

```
pip install oatpower
```

The pip installation is most suited to users wishing to solve standard ‘off-the-shelf’ power flow problems without the need for full access to the OATS scripts. The ipopt solver is included in the oatpower package along with the following key dependencies:

OATS can also be installed directly from the source available [here](#).

Installing OATS from the source offers the advantage of full access to customise the OATS scripts. This is recommended for advanced users comfortable with coding (or learning) in pyomo.

2.2 Solvers

A solver is required to solve a power systems optimisation problem in OATS. The choice of the problem depends on the type of optimisation problem. The optimisation problems can be broadly classified into the following four main categories:

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

The following table presents the classification of the traditional optimisation models implemented in OATS and suitable solvers that can be used to solve these problems.

Model	Type	Solver
DC Optimal Power Flow	LP	cplex, ipopt, glpk
AC Optimal Power Flow	NLP	ipopt
DC Security Constrained OPF	LP	cplex, ipopt, glpk
Unit commitment	MILP	cplex

2.2.1 NEOS

OATS allow a user to submit the problems to NEOS server. A user can specify `neos=True` option while calling the function and also can specify the solver to use on the NEOS server by using the option `'solver'`.

For more details on the available solvers on the NEOS server, please follow this [link](#).

2.2.2 Local installation of solvers

Installation of a local solver is recommended for using OATS. This is not only computationally efficient but also allow greater control for specifying options to the solver. The following table provides several open-source and free academic license solvers that can be used with OATS.

Solver name	Capability	License	Reference
glpk	LP, MILP	Open source	[1]
cbc	LP, MILP	Open source	[2]
lp_solve	LP, MILP	Open source	[3]
ipopt	LP, NLP	Open source	[4]
bonmin	LP, NLP, MILP, MINLP	Open source	[5]
couen	LP, NLP, MILP	Open source	[6]
gurobi	LP, QP, MILP	Free academic license	[7]
CPLEX	LP, QP, MILP	Free academic license	[8]

2.2.3 Installation instructions for the CPLEX solver

Academics can get free access to the IBM solver CPLEX. The following instructions will help you to download CPLEX solver directly from IBM academic initiative website:

1. Go to the IBM academic initiative page using this [link](#).
2. Register an account with an academic institution-issued email address
3. After registering and logging into your IBM academic initiative account click on Download vXY.Z under ILOG CPLEX Optimization Studio on this [page](#). .
4. A new window will open where you can choose a Download option suitable for your operating system (for example IBM ILOG CPLEX Optimization Studio 12.8 for Windows x86-64 Multilingual).

2.2.4 References

- [1] “GLPK (GNU linear programming kit),” 2006. [Online]. Available: <http://www.gnu.org/software/glpk>
- [2] J. Forrest and R. Lougee-Heimer, CBC User Guide, ch. Chapter 10, pp. 257–277. [Online]. Available: <https://pubsonline.informs.org/doi/abs/10.1287/educ.1053.0020>
- [3] “lp solve: Documentation 5.52.5,” 2016. [Online]. Available: <http://web.mit.edu/lpsolve/doc/>

- [4] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, pp. 25–57, 2006.
- [5] “bonmin (basic open-source nonlinear mixed integer programming),” 2005. [Online]. Available: <https://www.coin-or.org/Bonmin/>
- [6] P. Belotti, J. Lee et al., “Branching and bounds tightening techniques for non-convex MINLP,” *Optimization Methods and Software*, vol. 24, no. 4-5, pp. 597–634, 2009. [Online]. Available: <https://projects.coinor.org/Couenne>
- [7] I. Gurobi Optimization, “Gurobi optimizer reference manual,” 2016. [Online]. Available: <http://www.gurobi.com>
- [8] “IBM ILOG CPLEX Optimizer,” <http://www01.ibm.com/software/integration/optimization/cplex-optimizer/>, Last 2010.

OATS is a powerful power systems optimisation toolbox. OATS include implementation of the following steady-state analysis models:

- DC/AC load flow problem
- DC/AC optimal power flow problem
- Security constrained optimal power flow problem
- Unit commitment problem

3.1 Solving a problem in OATS

```
import oats
oats.dcopf()
```

The above command will solve a DC optimal power flow problem on a default 24-bus IEEE reliability test system. A user provide their own network by using the keyword 'tc', as shown in an example below.

```
>>> import oats
>>> oats.dcopf(solver='cplex')

Problem:
- Lower bound: -inf
- Upper bound: inf
- Number of objectives: 1
- Number of constraints: 272
- Number of variables: 167
- Sense: unknown
Solver:
- Status: ok
- Message: CPLEX 12.9.0.0\3a optimal solution; objective 61001.241036269814; 9 separable QP barrier iterations; No basis.
- Termination condition: optimal
- Id: 0
Solution:
- number of solutions: 0
- number of solutions displayed: 0

>>> █
```

```
import oats
oats.dcopf(tc='mynetwork.xlsx',)
```

Other options allow a user to specify solver: either on NEOS server or locally on a machine. The following set of lines solves a DC optimal power flow problem using a local installation of the solver ‘cplex’.

```
import oats
oats.dclf(tc='mynetwork.xlsx', solver='cplex', neos=False, out=0):
```

```
>>> oats.dcopf(neos=False, solver='cplex')

Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.3.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2015. All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> Logfile 'cplex.log' closed.
Logfile '/var/folders/d2/hnm6wp8x3hn9zsnnp8w6pj3c0000gn/T/tmp76_e1b2w.cplex.log' open.
CPLEX> Problem '/var/folders/d2/hnm6wp8x3hn9zsnnp8w6pj3c0000gn/T/tmp76_e1b2w.cplex.log' read.
Read time = 0.00 sec. (0.03 ticks)
CPLEX> Problem name : /var/folders/d2/hnm6wp8x3hn9zsnnp8w6pj3c0000gn/T/tmp76_e1b2w.cplex.log
Objective sense : Minimize
Variables : 168 [Nneg: 18, Free: 150, Qobj: 22]
Objective nonzeros : 50
Objective Q nonzeros : 22
Linear constraints : 273 [Less: 88, Greater: 71, Equal: 114]
Nonzeros : 496
RHS nonzeros : 158

Variables : Min LB: 0.000000 Max UB: all infinite
Objective nonzeros : Min : 0.1000000 Max : 1.996071e+07
Objective Q nonzeros : Min : 4.260000 Max : 6568.240
Linear constraints :
Nonzeros : Min : 0.7100000 Max : 71.94245
RHS nonzeros : Min : 0.02400000 Max : 5.000000

CPLEX> Tried aggregator 1 time.
QP Presolve eliminated 170 rows and 20 columns.
Aggregator did 71 substitutions.
Reduced QP has 32 rows, 77 columns, and 162 nonzeros.
Reduced QP objective Q matrix has 22 nonzeros.
Presolve time = 0.00 sec. (0.14 ticks)
Parallel mode: using up to 4 threads for barrier.
Number of nonzeros in lower triangle of A*A' = 92
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.00 sec. (0.01 ticks)
```

```

Summary statistics for Cholesky factor:
Threads          = 4
Rows in Factor   = 32
Integer space required = 37
Total non-zeros in factor = 247
Total FP ops to factor = 2325
Itn    Primal Obj    Dual Obj    Prim Inf    Upper Inf    Dual Inf
0    -4.4599038e+07    -3.1958212e+06    1.01e+02    2.69e+02    1.66e+06
1    -8.0892592e+06    -1.3410664e+06    1.79e+01    4.76e+01    2.95e+05
2    -1.1612724e+06    -4.8231113e+05    2.86e+00    7.66e+00    4.72e+04
3    -1.6613438e+05    -1.6856099e+05    5.83e-01    1.57e+00    9.61e+03
4    3.2515523e+04    2.3045797e+04    8.48e-02    2.30e-01    1.40e+03
5    5.8105532e+04    5.5622820e+04    8.50e-03    2.30e-02    1.40e+02
6    6.0778955e+04    6.0506304e+04    7.80e-04    2.12e-03    1.29e+01
7    6.1003432e+04    6.0987553e+04    3.82e-06    1.04e-05    6.30e-02
8    6.1001279e+04    6.1000940e+04    9.89e-09    2.68e-08    1.63e-04
9    6.1001241e+04    6.1001237e+04    6.00e-11    1.23e-10    7.49e-07
Barrier time = 0.00 sec. (0.40 ticks)

Total time on 4 threads = 0.00 sec. (0.40 ticks)

Barrier - Optimal: Objective = 6.1001241014e+04
Solution time = 0.00 sec. Iterations = 9
Deterministic time = 0.40 ticks (109.81 ticks/sec)

CPLEX> Solution written to file '/var/folders/d2/hnm6wp8x3hn9zsnp8w6pj3c0000gn/T/tmp6oyq4xe.cplex.sol'.
CPLEX> =====

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

- Status: ok
  User time: 0.0
  Termination condition: optimal
  Termination message: Barrier - Optimal\vx3a Objective = 6.1001241014e+04
  Error rc: 0
  Time: 0.03131294250488281

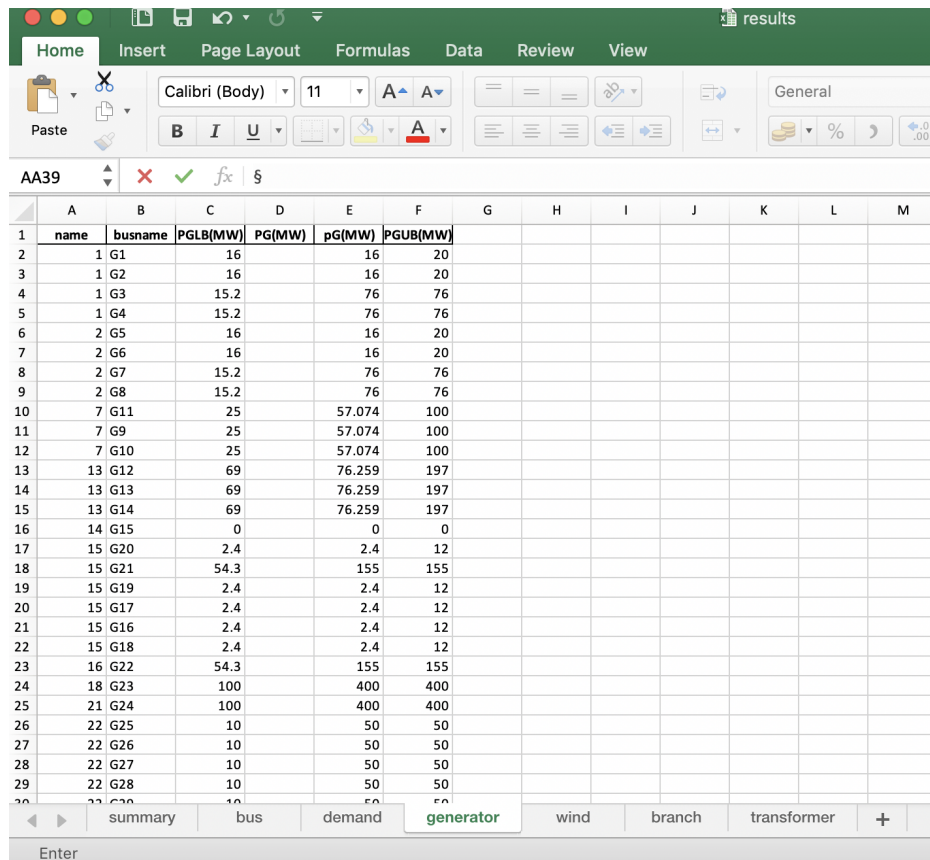
-----
Optimization Converged!
Cost of the objective function: 61001.24101440076
*****

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

```

3.2 Output

A 'results.xlsx' file is produced after an OATS model is solved. This file is created in the directory where the oats is called. The results file has a same data format as the input test case.



AA39

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	name	busname	PGLB(MW)	PG(MW)	pG(MW)	PGUB(MW)							
2	1 G1		16		16	20							
3	1 G2		16		16	20							
4	1 G3		15.2		76	76							
5	1 G4		15.2		76	76							
6	2 G5		16		16	20							
7	2 G6		16		16	20							
8	2 G7		15.2		76	76							
9	2 G8		15.2		76	76							
10	7 G11		25		57.074	100							
11	7 G9		25		57.074	100							
12	7 G10		25		57.074	100							
13	13 G12		69		76.259	197							
14	13 G13		69		76.259	197							
15	13 G14		69		76.259	197							
16	14 G15		0		0	0							
17	15 G20		2.4		2.4	12							
18	15 G21		54.3		155	155							
19	15 G19		2.4		2.4	12							
20	15 G17		2.4		2.4	12							
21	15 G16		2.4		2.4	12							
22	15 G18		2.4		2.4	12							
23	16 G22		54.3		155	155							
24	18 G23		100		400	400							
25	21 G24		100		400	400							
26	22 G25		10		50	50							
27	22 G26		10		50	50							
28	22 G27		10		50	50							
29	22 G28		10		50	50							
30	22 G29		10		50	50							

summary bus demand generator wind branch transformer +

Enter

OATS includes implementation of the following steady-state power system optimisation models:

4.1 Load flow problem

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

The following two tables provide information about the generator and transformers types modelled in OATS.

Generator Types in OATS
1= PV bus
2= Distributed slack bus
3= Reference bus

Transformer Types in OATS
1= 2-winding transformer
2= Tap-changing transformer

4.1.1 Mathematical formulation

The load flow problem in OATS is solved as a constrained OPF problem. The fixed parameters of PV, PQ and $V\delta$ buses are modelled using hard constraints. The detailed mathematical formulation is provided in the following technical note.

Bukhsh, W. (2018). On Solving the Load Flow Problem as an Optimization Problem. Glasgow: University of Strathclyde. [Online] Available: <https://pureportal.strath.ac.uk/en/publications/on-solving-the-load-flow-problem-as-an-optimization-problem>

4.1.2 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'.

4.1.3 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.

4.1.4 Solving DC and AC load flow problems in OATS

```
oats.dc1f(tc='default', solver='ipopt', neos=True, out=0)
```

Solves DC load flow problem

ARGUMENTS: **tc** (*.xlsx file) - OATS test case. See OATS data format for details

solver (str) - name of a solver. Default is 'ipopt'

neos (bool) - If True, the problem is solved using NEOS otherwise using a locally install solver.

out (bool) - If True, the output is displayed on screen.

```
oats.ac1f(tc='default', solver='ipopt', neos=True, out=0)
```

Solves AC load flow problem

ARGUMENTS: **tc** (*.xlsx file) - OATS test case. See OATS data format for details

solver (str) - name of a solver. Default is 'ipopt'

neos (bool) - If True, the problem is solved using NEOS otherwise using a locally install solver.

out (bool) - If True, the output is displayed on screen.

4.2 Optimal power flow problem

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.

The DCOPF problem is a linear optimisation problem, whereas the ACOPF problem is nonconvex and nonlinear. The ACOPF problem can be formulated using either polar or rectangular coordinates. The current release of OATS provides an implementation in the polar coordinates. The mathematical model of the ACOPF borrowed from [1] is as follows:

$$\min \sum_{g \in \mathcal{G}} f(p_g^G) \quad (1)$$

subject to

$$\sum_{g \in \mathcal{G}_b} p_g^G = \sum_{d \in \mathcal{D}_b} P_d^D + \sum_{b' \in \mathcal{B}_b} p_{bb'}^L + G_b^B v_b^2 \quad (2)$$

$$\sum_{g \in \mathcal{G}_b} q_g^G = \sum_{d \in \mathcal{D}_b} Q_d^D + \sum_{b' \in \mathcal{B}_b} q_{bb'}^L - B_b^B v_b^2 \quad (3)$$

$$p_{bb'}^L = v_b^2 G_{bb} + v_b v_{b'} (G_{bb'} \cos(\theta_b - \theta_{b'}) + B_{bb'} \sin(\theta_b - \theta_{b'})) \quad (4)$$

$$q_{bb'}^L = -v_b^2 B_{bb} + v_b v_{b'} (G_{bb'} \sin(\theta_b - \theta_{b'}) - B_{bb'} \cos(\theta_b - \theta_{b'})) \quad (5)$$

$$\theta_{b_0} = 0 \quad (6)$$

$$v_b^{\text{LB}} \leq v_b \leq v_b^{\text{UB}} \quad (7)$$

$$P_g^{\text{LB}} \leq p_g \leq P_g^{\text{UB}} \quad (8)$$

$$Q_g^{\text{LB}} \leq q_g \leq Q_g^{\text{UB}} \quad (9)$$

$$p_{bb'}^L{}^2 + q_{bb'}^L{}^2 \leq (S_{bb'}^{\text{max}})^2 \quad (10)$$

`oats.dcopf` (*tc*='default', *solver*='ipopt', *neos*=True, *out*=0)

Solves DC optimal power flow problem

ARGUMENTS: *tc* (*.xlsx file) - OATS test case. See OATS data format for details

solver (str) - name of a solver. Default is 'ipopt'

neos (bool) - If True, the problem is solved using NEOS otherwise using a locally install solver.

out (bool) - If True, the output is displayed on screen.

`oats.acopf` (*tc*='default', *solver*='ipopt', *neos*=True, *out*=0)

Solves AC optimal power flow problem

ARGUMENTS: *tc* (*.xlsx file) - OATS test case. See OATS data format for details

solver (str) - name of a solver. Default is 'ipopt'

neos (bool) - If True, the problem is solved using NEOS otherwise using a locally install solver.

out (bool) - If True, the output is displayed on screen.

[1] W. A. Bukhsh, A. Grothey, K. I. M. McKinnon and P. A. Trodden, "Local Solutions of the Optimal Power Flow Problem," in IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 4780-4788, Nov. 2013.(doi: 10.1109/TPWRS.2013.2274577)

4.3 Security constrained optimal power flow problem

Secure operation of a power system requires that no breach of operating standards take place following a credible contingency. This is achieved by solving 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, transformers or generating units.

4.3.1 Mathematical formulation

The mathematical formulation of the SCOPF implemented in OATS takes the following form:

$$\begin{aligned} & \min_{u_c, x_c: \forall c \in \{0\} \cup C} f(u_0, x_0) \\ & \text{subject to} \\ & \quad h_c(u_c, x_c) = 0, \quad \forall c \in \{0\} \cup C \\ & \quad g_c(u_c, x_c) \leq 0, \quad \forall c \in \{0\} \cup C \\ & \quad |u_c - u_0| \leq R_c, \quad \forall c \in C \end{aligned}$$

where C is the set of contingencies. OATS allow a user to build the set of contingencies by selecting generators, branches and transformers to be included in the contingency list. The user is referred to the data format section for information regarding selecting a contingency. The equality constraints (h_c) and inequality constraints (g_c) are imposed on the set of contingencies C and on the pre-fault operating state $\{0\}$. The last constraint is a coupling constraint that couples pre-fault and post-fault state of operation.

4.3.2 SCOPF problem with pre-fault AC and post-fault DC

The traditional implementations of the SCOPF problem model the pre-fault and post-fault operation of a system using DC-model of power flow. OATS allow a user to model the pre-fault operation of a system using AC power flow (hence giving information regarding voltage and reactive power) and post-fault operation using DC equations. The mathematical formulation of such problem is given as follows.

$$\begin{aligned} & \min_{u_c, x_c: \forall c \in \{0\} \cup C} f(u_0, x_0) \\ & \text{subject to} \\ & \quad h_0^{\text{AC}}(u_c, x_c) = 0, \quad c \in \{0\} \\ & \quad g_0^{\text{AC}}(u_c, x_c) \leq 0, \quad c \in \{0\} \\ & \quad h_c^{\text{DC}}(u_c, x_c) = 0, \quad \forall c \in C \\ & \quad g_c^{\text{DC}}(u_c, x_c) \leq 0, \quad \forall c \in C \\ & \quad |u_c - u_0| \leq R_c, \quad \forall c \in C \end{aligned}$$

As noted above, the AC power flow equations are only used to model the pre-fault operation of a system. The post-fault operation of a system is modelled using DC power flow equations.

4.3.3 Solving SCOPF problems in OATS

`oats.scopf` (*tc*='default', *solver*='ipopt', *neos*=True, *out*=0)

Solves security constrained optimal power flow problem

ARGUMENTS: *tc* (*.xlsx file) - OATS test case. See OATS data format for details

solver (str) - name of a solver. Default is 'ipopt'

neos (bool) - If True, the problem is solved using NEOS otherwise using a locally install solver.

out (bool) - If True, the output is displayed on screen.

4.4 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 minimise 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.

4.4.1 A mathematical formulation of the UC problem

Several mathematical formulations of the unit commitment problem exists in literature. The current release of oats implement a formulation from the following paper:

G. Morales-España, J. M. Latorre and A. Ramos, “Tight and Compact MILP Formulation for the Thermal Unit Commitment Problem,” in IEEE Transactions on Power Systems, vol. 28, no. 4, pp. 4897-4908, Nov. 2013. doi: 10.1109/TPWRS.2013.2251373

Note that the above formulation provides several tight relaxations around minimum start-up (shut-down) times of the thermal generators. These relaxations can easily be implemented in OATS by adopting the current ‘UC.mod’ model file. In case of any issues, the user is encouraged to raise an issue via GitHub page for support.

4.4.2 Solving UC problems in OATS

```
oats .uc (tc='default', solver='cplex', neos=True, out=0)
```

Solves unit commitment problem

ARGUMENTS: **tc** (*.xlsx file) - OATS test case. See OATS data format for details

solver (str) - name of a solver. Default is ‘ipopt’

neos (bool) - If True, the problem is solved using NEOS otherwise using a locally install solver.

out (bool) - If True, the output is displayed on screen.

The following table provides information about the implementation of the steady-state optimisation models in OATS. Note that the selected solver(s) column is not an exhaustive list of solvers. Also, the references column provide links to a selected set of publications where the users can find the mathematical formulation of the models implemented in OATS.

OATS ID	Model	Classification	Selected solver(s)	References
DCLF	DC load flow	LP	cplex, glpk	[1,2]
DCOPF	DC optimal power flow	LP	cplex, gurobi	[3]
SCOPF	Security constrained OPF	LP	cplex, gurobi	[4]
ACLF	AC load flow	NLP	ipopt	[2,3]
ACOPF	AC optimal power flow	NLP	ipopt	[5,6]
UC	Unit commitment problem	MILP	cplex, bonmin	[7]

[1] W. Bukhsh, On Solving the Load Flow Problem as an Optimization Problem. Tech. Report, University of Strathclyde, May 2018. [Online]. Available: <https://strathprints.strath.ac.uk/64156/>

- [2] S. Frank and S. Rebennack, “An introduction to optimal power flow: Theory, formulation, and examples,” *IEEE Transactions*, vol. 48, no. 12, pp. 1172–1197, 2016. [Online]. Available: <https://doi.org/10.1080/0740817X.2016.1189626>
- [3] A. J. Wood, *Power generation, operation, and control*, Third edition ed., 2014.
- [4] D. Phan and J. Kalagnanam, “Some efficient optimization methods for solving the security-constrained optimal power flow problem,” *IEEE Transactions on Power Systems*, vol. 29, no. 2, pp. 863–872, March 2014.
- [5] W. Bukhsh, A. Grothey et al., “Local solutions of the optimal power flow problem,” *Power Systems, IEEE Transactions on*, vol. 28, no. 4, pp. 4780–4788, Nov 2013.
- [6] M. B. Cain, R. P. O’Neil, and A. Castillo, “History of optimal power flow and formulations, optimal power flow paper 1,” 2012.
- [7] G. Morales-Espana, J. Latorre, and A. Ramos, “Tight and compact MILP formulation for the thermal unit commitment problem,” *Power Systems, IEEE Transactions on*, vol. 28, no. 4, pp. 4897–4908, Nov 2013.

Data format

OATS uses a spreadsheet format for specifying network, demand and generation data.

OATS also has a test case library where a number of standard IEEE and some real-world test cases are provided. The test case library is available on the [OATS GitHub page](#).

5.1 OATS data template

A blank template of OATS data format can be downloaded from this [link](#).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	name	baseKV	type	zone	VM	VA	VNLB	VNULB	VELB	VEUB										
2	1	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
3	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
4	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
5	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
6	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
7	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
8	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
9	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
10	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
11	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
12	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
13	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
14	2	138	3	1	1.05	0	0.95	1.05	0.95	1.05										
15	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
16	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
17	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
18	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
19	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
20	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
21	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
22	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
23	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
24	2	138	2	1	1.05	0	0.95	1.05	0.95	1.05										
25	2	138	1	1	1.05	0	0.95	1.05	0.95	1.05										
26																				

bus	demand	branch	transformer	wind	shunt	zonalNTC	generator	baseMVA	timeseries	zone	+
-----	--------	--------	-------------	------	-------	----------	-----------	---------	------------	------	---

5.2 Description of sheets in the OATS data format

Following tables provide information about each sheet of the OATS dataformat.

The parameters required to describe a network in OATS are outlined below. Optional parameters are highlighted by (^);

5.2.1 bus

name	bus name. string (can contain letters and/or numbers)
baseKV	base voltage (kV)
zone [^]	zone (positive integer) ¹
VM	Voltage Magnitude (p.u.)
VA	Voltage angle
VNLB	Normal minimum voltage magnitude (p.u.)
VNUB	Normal maximum voltage magnitude (p.u.)
VELB	Extreme minimum voltage magnitude (p.u.) ²
VEUB	Extreme minimum voltage magnitude (p.u.)

Notes

5.2.2 demand

name	Demand name (string)
busname	Bus name ³
real	real power demand (MW)
reactive	reactive power demand (MVar)
stat	Status (1- connected, 2-disconnected)
VOLL	Value of Lost Load (£/MW)

Notes

5.2.3 branch

name	branch name (string)
from_busname	from bus name ⁴
to_busname	to bus name ⁴
stat	Status (1-connected, 0-disconnected)
r	resistance (p.u.)
x	reactance (p.u.)
b	total line charging susceptance (p.u.)
ShortTermRating	MVA rating (short term rating), set to 0 for unlimited ⁵
ContinuousRating	MVA rating (continuous rating), set to 0 for unlimited
angLB	minimum angle difference (degrees) ⁶
angUB	maximum angle difference (degrees) ⁶
contingency [^]	1-include in SCOPF contingencies, 0- don't include
failure_rate [^]	failure rate over user specified time horizon

¹ Zone is used in unit commitment problem to define inter zonal transfer constraints

² Extreme values columns are provided as an option for security constrained optimal power flow when relaxed post-fault voltage bounds are desired

³ Must match a bus name from the bus sheet

Notes

5.2.4 transformer

name	transformer name (string)
from_busname	from bus name ⁷
to_busname	to bus name ⁷
stat	Status (1-connected, 0-disconnected)
type^	1- 2-winding transformer with fixed tap ratios 2- tap-changing transformer
r	resistance (p.u.)
x	reactance (p.u.)
ShortTermRating	MVA rating (short term rating), set to 0 for unlimited ⁸
ContinuousRating	MVA rating (continuous rating), set to 0 for unlimited
angLB	minimum angle difference (degrees) ⁹
angUB	minimum angle difference (degrees) ⁹
PhaseShift^	transformer phase shift angle (degrees), positive => delay
TapRatio^	Transformer turns ratio
TapLB	Transformer minimum turns ratio
TapUB	Transformer maximum turns ratio
contingency^	1-include in SCOPF contingencies, 2- don't include
failure_rate^	failure rate over user specified time horizon

Notes

5.2.5 wind

The wind sheet is included to separate variable generation from fixed capacity

busname	Bus name ¹⁰
name	Wind farm name
stat	Status (1-connected, 0-disconnected)
PG	Real power output (MW)
QG	Reactive power output (MVar)
PGLB	Minimum real power output (MW)
PGUB	Maximum power output (MW)
QGLB	Minimum Reactive power output (MW)
QGUB	Maximum reactive power output (MVar)
VS	Voltage magnitude setpoint (p.u.)
contingency^	1-include in SCOPF contingencies, 0- don't include
failure_rate^	failure rate over user specified time horizon

⁴ Must match a bus name from the bus sheet

⁵ The short term rating is used in post fault calculation in SCOPF

⁶ The voltage angle difference is taken to be unbounded below if angLB < -360 and unbounded above if angUB > 360. If both parameters are zero, it is unconstrained.

⁷ Must match a bus name from the bus sheet

⁸ The short term rating is used in post fault calculation in SCOPF

⁹ The voltage angle difference is taken to be unbounded below if angLB < -360 and unbounded above if angUB > 360. If both parameters are zero, it is unconstrained.

¹⁰ Must match a bus name from the bus sheet

Notes

5.2.6 shunt

busname	Bus name ¹¹
name	Shunt name (string)
GL	Shunt conductance (MW demanded at $V = 1.0$ p.u.)
BL	Shunt susceptance (MVar injected at $V = 1.0$ p.u.)
stat	Status (1- connected, 0-disconnected)

Notes

5.2.7 zone

interconnection_ID	ID for interconnector between zones
from_zone	from zone ¹²
to_zone	to zone ¹²
TransferCapacity(MW)	Transfer capacity between 'from_zone' and 'to_zone'

¹¹ Must match a bus name from the bus sheet

¹² Must match a zone name from the bus sheet

Notes

5.2.8 generators

busname	Bus name ¹³
name	Generator name (string)
stat	Status (1-connected, 0-disconnected)
PG	Real power output (MW)
QG	Reactive power output (MVar)
PGLB	Minimum real power output (MW)
PGUB	Maximum power output (MW)
QGLB	Minimum Reactive power output (MW)
QGUB	Maximum reactive power output (MVar)
VS	Voltage magnitude setpoint (p.u.)
RampDown (MW/hr)^	Ramp down rate (MW/hr) ¹⁴
RampUp (MW/hr)^	Ramp up rate (MW/hr) ¹⁴
MinDownTime(hr)^	Minimum down time (hr) ¹⁵
MinupTime(hr)^	Minimum up time (hr) ¹⁵
FuelType^	Coal, Nuke - nuclear, CCGT, OCGT, Unknown
contingency	1-include in SCOPF contingencies, 0- don't include
startup^	Start up cost (£) ¹⁵
shutdown^	Shut down cost (£) ¹⁵
costc2	Quadratic cost coefficient
costc1	Linear cost coefficient
costc0	Constant cost coefficient
bid^	Bid in balancing mechanism to reduce generation ¹⁶
offer^	Offer in balancing mechanism to increase generation ¹⁶

Notes

5.2.9 storage

name	Name for the storage device
zone	Name of the zone
stat	Status
Minoperatingcapacity(MW)	Min operating capacity
capacity(MW)	Total capacity of the storage
chargingrate(MW/hr)	charging rate
dischargingrate(MW/hr)	discharging rate
ChargingEfficiency(%)	charging efficiency
DischargingEfficiency(%)	discharging efficiency
InitialStoredPower(MW)	Initial stored energy
FinalStoredPower(MW)	Final stored energy at the end of the planning horizon

¹³ Must match a bus name from the bus sheet¹⁴ Ramp rates required for security constrained OPF or unit commitment problems¹⁵ Minimum up/down times, startup and shutdown costs are required in the unit commitment models¹⁶ These parameters are part of the balancing market extension model that is available as an extension to OATS

5.3 Filter Matpower2Oats

A Python script is provided that can be used to convert Matpower test-cases into equivalent OATS test-cases. This script is available on the [OATS GitHub page](#).

Extending models in OATS

A salient feature of OATS is its ease of extending a model to define a new class of problem.

No modelling tool is capable of capturing all the extensions in constraint handling, or extension in objective function so it is important to give the user freedom to define their problems.

Knowledge of PYOMO modelling language is required to defined OATS models. The [GitHub](#) page of OATS includes a range of models that are extended for specific applications.

6.1 Adding variables

Variables in OATS are defined using Pyomo's 'Var' function. For example, real power generation variable in DCOPF model of OATS is defined as follows:

```
model.pG = Var(model.G, domain= Reals)
```

The above definition of the variable name 'pG' states that it is variable defined on a set of generators 'G' and is with the domain of real numbers.

6.2 Adding parameters

Parameters in OATS are defined using Pyomo's 'Param' function. The following example shows definition of a parameter:

```
model.PGmax = Param(model.G, within=NonNegativeReals)
```

The above definition of a parameter 'PGmax' defines a new parameter that belongs to the set of generators 'G' and since it is modelling the capacity of a generator it's domain is defined as a set of non-negative real numbers. OATS will raise an error if a user tries to input a negative value for a generator capacity.

6.3 Modifying objective function

The objective function describes the main aim of the model which is either to minimise or maximise. In power systems optimisation problems often the objective function is to minimise the total cost of generation that is required to meet demand. The objective function of DCOPF and ACOPF models in OATS is written as follows:

```
def objective(model):
    obj = sum(model.c2[g]*(model.baseMVA*model.pG[g])**2+model.c1[g]*model.
    ↪baseMVA*model.pG[g]+ model.c0[g] for g in model.G)+\
    sum(model.VOLL[d]*(1-model.alpha[d])*model.baseMVA*model.PD[d] for d in model.D)
    return obj
```

The first part of the objective function is to minimise the cost of generation and the second part of the objective function is to minimise the cost of load shedding. 'c2', 'c1' and 'c0' are the coefficients of the quadratic cost function and 'VOLL' represents the value of lost load. 'g' represents a generator in 'model.G' set of generators. 'd' represents a demand in 'model.D' set of demands.

The objective function in OATS can be changed by modifying the 'obj' variable. For example, if it is desired in the ACOPF model that the voltages deviation from 1 p.u. is penalised then that could be achieved by modifying the objective function in the following way:

```
def objective(model):
    obj = sum(model.c2[g]*(model.baseMVA*model.pG[g])**2+model.c1[g]*model.
    ↪baseMVA*model.pG[g]+ model.c0[g] for g in model.G)+\
    sum(model.VOLL[d]*(1-model.alpha[d])*model.baseMVA*model.PD[d] for d in model.D)+\
    model.WV*sum((1-model.v[b])**2 for b in model.B)
    return obj
```

Note that we have added a penalty in the objective function that penalises the violation of the voltages from 1 p.u. 'WV' is a weighting on the voltage deviation part of the objective function. 'b' represents a demand in 'model.B' set of demands. 'model.v[b]' represents the voltage at bus 'b'.

6.4 Adding/Modifying constraints

Constraints in OATS are implemented as function calls for each member of a set. For example, line limits in DCOPF are implemented as follows:

```
def line_lim1_def(model,l):
    return model.pL[l] <= model.SLmax[l]
def line_lim2_def(model,l):
    return model.pL[l] >= -model.SLmax[l]
```

The line limit constraints are applied for each member of the set of lines 'L'. The following code snippet presents an example where the line limits are relaxed by 10%. 'pL[l]' represents the active power flow in line 'l' and 'SLmax[l]' represents the continuous line rating of line 'l'.

```
def line_lim1_def(model,l):
    return model.pL[l] <= 1.10*model.SLmax[l]
def line_lim2_def(model,l):
    return model.pL[l] >= -1.10*model.SLmax[l]
```

Consider a case when the relaxation of 10% is required to be penalised in the objective function. This could be achieved by defining new variables (a variable for each line) that captures line violations up to 10% and then penalises it in the objective function.

The first step is to define new variables as follows:

```
model.relaxL = Var(model.L, domain= NonNegativeReals)
```

The line limit constraints are modified as follows:

```
def line_lim1_def(model,l):
    return model.pL[l] <= model.SLmax[l]+model.relaxL[l]
def line_lim2_def(model,l):
    return model.pL[l] >= model.SLmax[l]-model.relaxL[l]
```

The line_lim1_def constraint ensures that the active power flow ‘model.pL[l]’ through line ‘l’ is less than or equal to the continuous line rating ‘model.SLmax’ plus the relaxation variable ‘model.relaxL[l]’. The line_lim2_def constraint ensures that the active power flow ‘model.pL[l]’ through line ‘l’ is more than or equal to the continuous line rating ‘model.SLmax’ minus the relaxation variable ‘model.relaxL[l]’.

The variable ‘relaxL’ needs to be bounded so that the line violations are limited to 10%. This can be achieved using the following constraint:

```
def relaxL_bound(model,l):
    return model.relaxL[l] <= 0.1*model.SLmax[l]
```

The final step is to penalise variable ‘relaxL’ in the objective function:

The objective function now puts a cost on relaxation of the line power flow constraint, ‘WR’ is the weighting of the cost.

6.5 Solving user defined models

A user can solve a new model by called oats function ‘model’. Here is an example of solving DCOPF_BM model:

```
import oats
oats.model(model='DCOPF_BM')
```

The DCOPF_BM model is a balancing optimisation model where the objective function is to minimise the cost of redispatching generation from their set points; to balance supply and demand, and/or due to thermal and voltage constraints.

This section provides a number of examples on extending models in OATS.

7.1 Voltage Dependent Loads

The steady-state analysis models implemented in OATS have a constant power model of electricity load. The ACOPF model in OATS can be easily extended to model voltage dependent loads. In this demonstrative examples, a ZIP model of electricity load consists of constant impedance (Z), constant current (I) and constant power (P) load components and are represented by a second-order polynomial in bus voltage magnitude as follows:

$$\begin{aligned} p_d^D(v_d) &= P_d^D (a_d^P v_d^2 + b_d^P v_d + c_d^P) \\ q_d^D(v_d) &= Q_d^D (a_d^Q v_d^2 + b_d^Q v_d + c_d^Q) \end{aligned} \quad (7.1)$$

where a_d^P, b_d^P, c_d^P are the active and reactive power coefficients of the quadratic polynomial, respectively. Parameters a_d^P represent the relative participation of constant impedance load, b_d^P the relative participation of constant load, and c_d^P the relative participation of constant power load.

The real and reactive power demand are modelled as parameters in the ACOPF model of OATS (written in PYOMO syntax) on the set of demands D, as follows:

```
model.PD = Param(model.D, within=Reals)
model.QD = Param(model.D, within=Reals)
```

In order to model the dependence of electricity demand on voltages, the real and reactive power parameters are modelled as variables pD and qD, as follows.

```
model.pD = Var(model.D, within=Reals)
model.qD = Var(model.D, within=Reals)
model.aPD = Param(model.D, within=NonNegativeReals)
model.bPD = Param(model.D, within=NonNegativeReals)
model.cPD = Param(model.D, within=NonNegativeReals)
model.aQD = Param(model.D, within=NonNegativeReals)
```

(continues on next page)

(continued from previous page)

```
model.bQD = Param(model.D, within=NonNegativeReals)
model.cQD = Param(model.D, within=NonNegativeReals)
```

The coefficients of the quadratic function are defined as parameters given by the user. Equations (2) are implemented in the ACOPF model of OATS to model the ZIP load as follows:

```
def real_power_demand(model, d):
    return model.pD[d] == model.PD(model.aPD[d]*model.v[b]**2+model.bPD[d]*model.
    ↪v[b]+model.cPD[d])

def reactive_power_demand(model, d):
    return model.qD[d] == model.QD(model.aQD[d]*model.v[b]**2+model.bQD[d]*model.
    ↪v[b]+model.cQD[d])
```

The implementation of the above model is provided in the [GitHub folder](#) containing OATS models extensions.

7.2 Modelling Storage in OATS

In the current version of OATS, storage is modelled as part of the unit commitment (UC) problem.

The UC problem is a time-linked problem where the time-periods are coupled via ramp rate constraints. The following power balance equation is imposed for each time period t and for each zone z :

$$\sum_{g \in G} p_{g,t}^G + \sum_{s \in S} (p_{s,t}^{\text{Out}} - p_{s,t}^{\text{In}}) = \sum_{d \in D} P_{d,t}^D + \sum_{l \in L} p_{l,t}^L$$

where p^{In} and p^{Out} represents the charging and discharging of energy storage, respectively. The energy storage is modelled using the following equation:

$$p_{s,t}^S = \eta_s^D p_{s,t}^{\text{Out}} - \frac{1}{\eta_s^C} p_{s,t}^{\text{In}}$$

where η_s^C, η_s^D are the charging and discharging efficiencies of the storage asset s , respectively.

For details about specifying storage data type, see the explanation of fields [here](#). The following command will run the unit commitment model and display results for storage that has been modelled in the network data test.xlsx.

```
oats.uc(neos=False, solver='cplex', tc = 'test.xlsx')
```

7.3 Optimising Reactive Power Margin

7.4 Optimising Transformers Tap Settings

O

oats, [17](#)

A

`ac1f()` (*in module oats*), 14
`acopf()` (*in module oats*), 15

D

`dc1f()` (*in module oats*), 14
`dcopf()` (*in module oats*), 15

O

`oats` (*module*), 14–17

S

`scopf()` (*in module oats*), 16

U

`uc()` (*in module oats*), 17