OSeMOSYS Documentation

Release 0.0.1

KTH-dESA

Jun 20, 2023

Contents:

1 The OpTIMUS Community **3**

1.1 Introduction to OSeMOSYS 3

1.2 Structure of OSeMOSYS 5

1.3 Code versions 27

1.4 Create a model in OSeMOSYS 28

1.5 Debugging a model: useful tips 33

1.6 Advanced Functionalities 35

1.7 How to contribute to the development of the tool 39

1.8 References 40

1.9 Contacts 40

1.10 FAQ 41

2 Indices and tables **43**

OSeMOSYS

Open Source Energy Modelling System

Fig. 1: KTH - dESA, www.osemosys.org

OSeMOSYS is an open source modelling system for long-run integrated assessment and energy planning. It has been employed to develop energy systems models from the scale of continents (African Power Pools, South America, EU28+2) down to the scale of countries, regions and villages. Designed to require no upfront financial investment, a fast learning curve and little time commitment to operate, it is fit for use by communities of developers, modellers, academics up to policy makers. Thanks to its transparency, it is broadly employed as a training and dissemination tool.

CHAPTER 1

The OpTIMUS Community

OSeMOSYS is part of the OpTIMUS Community, Practice 3: Open Software, together with other world class, peer reviewed open source tools and data.

OpTIMUS aims at promoting quantitative analysis to inform sustainable development policy, through the coordination of networks to advance open source software, knowledge development and capacity building. It is organized in three practices -modeling and capacity building for policy support, expert review and quality control, and software development. For more information on the OpTIMUS Community, please visit the related website: http://www.optimus.community

Part of OpTIMUS www.optimus.community Practice 3

1.1 Introduction to OSeMOSYS

At present there is a useful, but limited set of open, free and accessible energy system modeling tools. These tools often require significant investment in terms of human resources, training and software purchases in order to apply or further develop them. The Open Source energy MOdelling SYStem (OSeMOSYS) is specifically designed as a tool to inform the development of local, national and multi-regional energy strategies and support them with capacity building activities. Unlike other long established energy systems (partial equilibrium) modeling frameworks (i.e. MARKAL/TIMES[1] , MESSAGE[2] , PRIMES[3] , EFOM[4] and POLES[5]), it potentially requires a less significant learning curve. Additionally, by not using proprietary software or commercial programming languages and solvers, OSeMOSYS requires no upfront financial investment. This feature increases its affordability, particularly in developing countries.

It was developed in collaboration with a range of institutions, including the International Atomic Energy Agency (IAEA), the United Nations Industrial Development Organisation (UNIDO), KTH Royal Institute of Technology, Stanford University, University College London (UCL), University of Cape Town (UCT), Paul Scherrer Institute (PSI), Stockholm Environment Institute (SEI), and North Carolina State University. The first version of OSeMOSYS was made available in 2008, while the first peer-reviewed publication describing its ethos and structure was available in 2011[6] .

OSeMOSYS computes the energy supply mix (in terms of generation capacity and energy delivery) which meets the energy services demands every year and in every time step of the case under study, minimising (in its most common form) the total discounted costs. It can cover all or individual energy sectors, including heat, electricity and transport and has a user-defined spatial and temporal domain and scale. The energy demands can be met through a range of technologies which have certain techno-economic characteristics and draw on a set of resources, defined by certain potentials and costs. On top of this, policy scenarios may impose certain technical constraints, economic realities or environmental targets. As in most long-term optimisation modeling tools, OSeMOSYS in its standard configuration assumes a unique decision-maker, perfect foresight and competitive markets.

In mathematical terms, it is a deterministic, linear optimisation, long-term modeling framework. Mixed-integer linear programming may be applied for certain functions, like the optimisation of discrete power plant capacity expansions.

One of its main characteristics is the wide and flexible definition of technology and energy vector. A technology represents any asset operating energy conversion processes, from resource extraction and processing to electricity supply, transmission and distribution and end-use appliances. It could therefore refer to, for example, an oil refinery, a hydropower plant or a heating system. Each technology is characterised by a transfer function defined by numerous economic, technical and environmental parameters (e.g. investment and operating costs, efficiency, availability, emission factors, capacity factor, minimum load, etc.).

The original OSeMOSYS code is written in GNU MathProg, a high level mathematical programming language, yet straightforward enough to be understandable by all kinds of users, expert or not in linear programming. In its full version, the code consists of 700 text lines, highly resembling algebraic expressions. Further parallel versions of the code have been written in GAMS and Python, for better connection to the respective families of users and coders. The open source solver GLPK may be used for translating the models in matrices, finding the optimal solution and printing the numerical outputs.

OSeMOSYS applications can be created and run without interface. Still, several user interfaces have been developed and are largely employed in teaching and capacity building activities. The Model Management Infrastructure (MoManI) is an open source, free interface for creating models and visualising results, available both in online and desktop version. In addition, OSeMOSYS is integrated into LEAP as module for computing supply capacity expansion planning.

OSeMOSYS is available for download at http://www.osemosys.org/.

[1] Energy Technology Systems Analysis Program (IEA-ETSAP), 2015. IEA-ETSAP Energy Systems Analysis Tools. [Online]. Available at: https://iea-etsap.org/index.php/etsap-tools

[2] International Atomic Energy Agency (IAEA), 2015. Planning & Economic Studies Section (PESS): Capacity Building for Sustainable Energy Development. [Online] Available at: https://www.iaea.org/OurWork/ST/NE/Pess/PESSenergymodels.html

[3] National Technical University of Athens (NTUA), 2015. The PRIMES Energy. [Online] Available at: http://www.e3mlab.ntua.gr/manuals/PRIMES_Brussels_November_2007.pdf

[4] Van der Voort, E., 1982. The EFOM 12C energy supply model within the EC modelling system. *Omega*, 10 (5), pp. 507–523.

[5] Enerdata, 2017. POLES: Prospective Outlook on Long-term Energy Systems. [Online] Available at: https://www.enerdata.net/solutions/poles-model.html

[6] Howells et al., 2011. OSeMOSYS: The Open Source Energy Modeling System: An introduction to its ethos, structure and development. *Energy Policy*, 39 (10), pp. 5850-5870.

1.1.1 The OpTIMUS Community

OSeMOSYS is part of the OpTIMUS Community, Practice 3: Open Software, together with other world class, peer reviewed open source tools and data.

OpTIMUS aims at promoting quantitative analysis to inform sustainable development policy, through the coordination of networks to advance open source software, knowledge development and capacity building. It is organized in three practices -modeling and capacity building for policy support, expert review and quality control, and software development. For more information on the OpTIMUS Community, please visit the related website: http://www.optimus.community/.

1.2 Structure of OSeMOSYS

In OSeMOSYS, like usually in linear programs, sets, parameters and variables are defined. In this section, the sets, parameters and variables existing in OSeMOSYS are listed and briefly described. Illustrative examples on how to assign values for each of these will be given in the following Sections.

1.2.1 Sets

The ‘sets’ define the physical structure of a model, usually independent from the specific scenarios which will be run. They define the time domain and time split, the spatial coverage, the technologies and energy vectors to be considered, etc. For instance, when a variable is defined as a function of the set ‘YEAR’ it will be indicated as **variablename[y]** at it will be computed for every year listed in the set. The sets of OSeMOSYS are presented in the Table below.

Name	Description	Index
YEAR	It represents the time frame of the model, it contains all the years to be considered in the study.	y
TECHNOLOGY	It includes any element of the energy system that changes a commodity from one form to another, uses it or supplies it. All system components are set up as a 'technology' in OSeMOSYS. As the model is an abstraction, the modeller is free to interpret the role of a technology at will, where relevant. It may for example represent a single real technology (such as a power plant) or can represent a heavily aggregated collection of technologies (such as the stock of several million light bulbs), or may even simply be a 'dummy technology', perhaps used for accounting purposes.	t
TIMESLICE	It represents the time split of each modelled year, therefore the time resolution of the model. Common to several energy systems modelling tools (incl. MESSAGE / MARKAL / TIMES), the annual demand is 'sliced' into representative fractions of the year. It is necessary to assess times of the year when demand is high separately from times when demand is low, for fuels that are expensive to store. In order to reduce the computation time, these 'slices' are often grouped. Thus, the annual demand may be split into aggregate seasons where demand levels are similar (such as 'summer, winter and intermediate'). Those seasons may be subdivided into aggregate 'day types' (such as workdays and weekends), and the day further sub divided (such as into day and night) depending on the level of demand.	l
FUEL	It includes any energy vector, energy service or proxies entering or exiting technologies. These can be aggregate groups, individual flows or artificially separated, depending on the requirements of the analysis.	f
EMISSION	It includes any kind of emission potentially deriving from the operation of the defined technologies. Typical examples would include atmospheric emissions of greenhouse gasses, such as CO2.	e
MODE_OF_OPERATION	It defines the number of modes of operation that the technologies can have. If a technology can have various input or output fuels and it can choose the mix (i.e. any linear combination) of these input or output fuels, each mix can be accounted as a separate mode of operation. For example, a CHP plant may produce heat in one mode of operation and electricity in another.	m
REGION	It sets the regions to be modelled, e.g. different countries. For each of them, the supply-demand balances for all the energy vectors are ensured, including trades with other regions. In some occasions it might be computationally more convenient to model different countries within the same region and differentiate them simply by creating ad hoc fuels and technologies for each of them.	r
SEASON	It gives indication (by successive numerical values) of how many seasons (e.g. winter, intermediate, summer) are accounted for and in which order. This set is needed if storage facilities are included in the model.	ls
DAYTYPE	It gives indication (by successive numerical values) of how many day types (e.g. workday, weekend) are accounted for and in which order. This set is needed if storage facilities are included in the model.	ld
DAILYTIMEBRACKET	It gives indication (by successive numerical values) of how many parts the day is split into (e.g. night, morning, afternoon, evening) and in which order these parts are sorted. This set is needed if storage facilities are included in the model.	lh
STORAGE	It includes storage facilities in the model.	s

1.2.2 Parameters

The parameters are the user-defined numerical inputs to the model. While usually the structure of a model, therefore the sets, remains fixed across scenarios, it is common practice to change the values of some parameters when running different scenarios and/or sensitivity analyses. As will be clear in the following, each parameter is a function of the elements in one or more sets. For instance, **CapitalCost[r,t,y]** indicates that the capital cost is a function of the region (r), the technology (t) and the year (y)[1] . A list and brief description of the parameters declared in the master version

[1] Please note that the order of the indexes of the parameters presented here and available in the current version of OSeMOSYS is different from the one in Howells et al. in 2011. For more information refer to the Change Log of the OSeMOSYS versions, available at http://www.osemosys.

of OSeMOSYS is given in the Table below.

Global parameters

Name	Description
YearSplit[l,y]	Duration of a modelled time slice, expressed as a fraction of the year. The sum of each entry over one modelled year should equal 1.
DiscountRate[r]	Region specific value for the discount rate, expressed in decimals (e.g. 0.05)
DaySplit[lh,y]	Length of one DailyTimeBracket in one specific day as a fraction of the year (e.g., when distinguishing between days and night: 12h/(24h*365d)).
Conversionls[l,ls]	Binary parameter linking one TimeSlice to a certain Season. It has value 0 if the TimeSlice does not pertain to the specific season, 1 if it does.
Conversionld[ld,l]	Binary parameter linking one TimeSlice to a certain DayType. It has value 0 if the TimeSlice does not pertain to the specific DayType, 1 if it does.
Conversionlh[lh,l]	Binary parameter linking one TimeSlice to a certain DaylyTimeBracket. It has value 0 if the TimeSlice does not pertain to the specific DaylyTimeBracket, 1 if it does.
DaysInDayType[ls,ld,y]	Number of days for each day type, within one week (natural number, ranging from 1 to 7)
TradeRoute[r,rr,f,y]	Binary parameter defining the links between region r and region rr, to enable or disable trading of a specific commodity. It has value 1 when two regions are linked, 0 otherwise
DepreciationMethod[r]	Binary parameter defining the type of depreciation to be applied. It has value 1 for sinking fund depreciation, value 2 for straight-line depreciation.

Demands

Name	Description
SpecifiedAnnualDemand[r,f,y]	Total specified demand for the year, linked to a specific 'time of use' during the year.
SpecifiedDemandProfile[r,f,l,y]	Annual fraction of energy-service or commodity demand that is required in each time slice. For each year, all the defined SpecifiedDemandProfile input values should sum up to 1.
AccumulatedAnnualDemand[r,f,y]	Accumulated Demand for a certain commodity in one specific year. It cannot be defined for a commodity if its SpecifiedAnnualDemand for the same year is already defined and vice versa.

org/get-started.html.

Performance

Name	Description
CapacityToActivityUnit[r,t]	Conversion factor relating the energy that would be produced when one unit of capacity is fully used in one year.
CapacityFactor[r,t,l,y]	Capacity available per each TimeSlice expressed as a fraction of the total installed capacity, with values ranging from 0 to 1. It gives the possibility to account for forced outages.
AvailabilityFactor[r,t,y]	Maximum time a technology can run in the whole year, as a fraction of the year ranging from 0 to 1. It gives the possibility to account for planned outages.
OperationalLife[r,t]	Useful lifetime of a technology, expressed in years.
ResidualCapacity[r,t,y]	Remained capacity available from before the modelling period.
InputActivityRatio[r,t,f,m,y]	Rate of use of a commodity by a technology, as a ratio of the rate of activity.
OutputActivityRatio[r,t,f,m,y]	Rate of commodity output from a technology, as a ratio of the rate of activity.

Technology costs

Name	Description
CapitalCost[r,t,y]	Capital investment cost of a technology, per unit of capacity.
VariableCost[r,t,m,y]	Cost of a technology for a given mode of operation (Variable O&M cost), per unit of activity.
FixedCost[r,t,y]	Fixed O&M cost of a technology, per unit of capacity.

Storage

Name	Description
TechnologyToStorage[r,t,s,m]	Binary parameter linking a technology to the storage facility it charges. It has value 1 if the technology and the storage facility are linked, 0 otherwise.
TechnologyFromStorage[r,t,s,m]	Binary parameter linking a storage facility to the technology it feeds. It has value 1 if the technology and the storage facility are linked, 0 otherwise.
StorageLevelStart[r,s]	Level of storage at the beginning of first modelled year, in units of activity.
StorageMaxChargeRate[r,s]	Maximum charging rate for the storage, in units of activity per year.
StorageMaxDischargeRate[r,s]	Maximum discharging rate for the storage, in units of activity per year.
MinStorageCharge[r,s,y]	It sets a lower bound to the amount of energy stored, as a fraction of the maximum, with a number reanging between 0 and 1. The storage facility cannot be emptied below this level.
OperationalLifeStorage[r,s]	Useful lifetime of the storage facility.
CapitalCostStorage[r,s,y]	Investment costs of storage additions, defined per unit of storage capacity.
ResidualStorageCapacity[r,s,y]	Exogenously defined storage capacities.

Capacity constraints

Name	Description
CapacityOfOneTechnologyUnit[r,t,y]	Capacity of one new unit of a technology. In case the user sets this parameter, the related technology will be installed only in batches of the specified capacity and the problem will turn into a Mixed Integer Linear Problem.
TotalAnnualMaxCapacity[r,t,y]	Total maximum existing (residual plus cumulatively installed) capacity allowed for a technology in a specified year.
TotalAnnualMinCapacity[r,t,y]	Total minimum existing (residual plus cumulatively installed) capacity allowed for a technology in a specified year.
Investment constraints	
TotalAnnualMaxCapacityInvestment[r,t,y]	Maximum capacity of a technology, expressed in power units.
TotalAnnualMinCapacityInvestment[r,t,y]	Minimum capacity of a technology, expressed in power units.

Activity constraints

Name	Description
TotalTechnologyAnnualActivityUpperLimit[r,t,y]	Total maximum level of activity allowed for a technology in one year.
TotalTechnologyAnnualActivityLowerLimit[r,t,y]	Total minimum level of activity allowed for a technology in one year.
TotalTechnologyModelPeriodActivityUpperLimit[r,t]	Total maximum level of activity allowed for a technology in the entire modelled period.
TotalTechnologyModelPeriodActivityLowerLimit[r,t]	Total minimum level of activity allowed for a technology in the entire modelled period.

Reserve margin

Name	Description
ReserveMarginTagTechnology[r,t,y]	Parameter tagging the technologies that are allowed to contribute to the reserve margin. It has value between 0 and 1 for the technologies allowed.
ReserveMarginTagFuel[r,f,y]	Binary parameter tagging the fuels to which the reserve margin applies. It has value 1 if the reserve margin applies to the fuel, 0 otherwise.
ReserveMargin[r,y]	Minimum level of the reserve margin required to be provided for all the tagged commodities, by the tagged technologies. If no reserve margin is required, the parameter will have value 1; if, for instance, 20% reserve margin is required, the parameter will have value 1.2.

RE Generation target

Name	Description
RETagTechnology[r,t,y]	Binary parameter tagging the renewable technologies that must contribute to reaching the indicated minimum renewable production target. It has value 1 for thetechnologies contributing, 0 otherwise.
RETagFuel[r,f,y]	Binary parameter tagging the fuels to which the renewable target applies to. It has value 1 if the target applies, 0 otherwise.
REMinProductionTarget[r,y]	Minimum ratio of all renewable commodities tagged in the RETagCommodity parameter, to be produced by the technologies tagged with the RETechnology parameter.

Emissions

Name	Description
EmissionActivityRatio[r,t,e,m,y]	Emission factor of a technology per unit of activity, per mode of operation.
EmissionsPenalty[r,e,y]	Penalty per unit of emission.
AnnualExogenousEmission[r,e,y]	It allows the user to account for additional annual emissions, on top of those computed endogenously by the model (e.g. emissions generated outside the region).
AnnualEmissionLimit[r,e,y]	Annual upper limit for a specific emission generated in the whole modelled region.
ModelPeriodExogenousEmission[r,e]	It allows the user to account for additional emissions over the entire modelled period, on top of those computed endogenously by the model (e.g. generated outside the region).
ModelPeriodEmissionLimit[r,e]	Annual upper limit for a specific emission generated in the whole modelled region, over the entire modelled period.

1.2.3 Variables

The variables are the outputs computed by the code. As much as the parameters, also the variables are functions of the elements in one or more sets. In the following Table, a list and brief description of all the variables computed by the code of OSeMOSYS (in its full version) is given. As will be explained next in this manual, a shortened version of OSeMOSYS has been created, to improve the computational capability at the expenses of the readability of the code. In such version, only some of the variables here listed are computed. When reasonable, the domain of several variables has been constrained to be positive, in order to decrease the size of the solution space and therefore the computational effort.

Demands

Name	Description	
RateOfDemand[r,l,f,y]>=0	Intermediate variable. It represents the energy that would be demanded in one time slice l if the latter lasted the whole year. It is a function of the parameters SpecifiedAnnualDemand and SpecifiedDemandProfile. | Energy (per year)	
Demand[r,l,f,y]>=0	Demand for one fuel in one time slice.	Energy

Storage

Name	Description	
RateOfStorageCharge[r,s,ls,ld,lh,y]	Intermediate variable. It represents the commodity that would be charged to the storage facility s in one time slice if the latter lasted the whole year. It is a function of the RateOfActivity and the parameter TechnologyToStorage. | Energy (per year)	
RateOfStorageDischarge[r,s,ls,ld,lh,y]	Intermediate variable. It represents the commodity that would be discharged from storage facility s in one time slice if the latter lasted the whole year. It is a function of the RateOfActivity and the parameter TechnologyFromStorage.	Energy (per year)
NetChargeWithinYear[r,s,ls,ld,lh,y]	Net quantity of commodity charged to storage facility s in year y. It is a function of the RateOfStorageCharge and the RateOfStorageDischarge and it can be negative.	Energy
NetChargeWithinDay[r,s,ls,ld,lh,y]	Net quantity of commodity charged to storage facility s in daytype ld. It is a function of the RateOfStorageCharge and the RateOfStorageDischarge and can be negative.	Energy
StorageLevelYearStart[r,s,y]>=0	Level of stored commodity in storage facility s in the first time step of year y.	Energy
StorageLevelYearFinish[r,s,y]>=0	Level of stored commodity in storage facility s in the last time step of year y.	Energy
StorageLevelSeasonStart[r,s,ls,y]>=0	Level of stored commodity in storage facility s in the first time step of season ls.	Energy
StorageLevelDayTypeStart[r,s,ls,ld,y]>=0	Level of stored commodity in storage facility s in the first time step of daytype ld.	Energy
StorageLevelDayTypeFinish[r,s,ls,ld,y]>=0	Level of stored commodity in storage facility s in the last time step of daytype ld.	Energy
StorageLowerLimit[r,s,y]>=0	Minimum allowed level of stored commodity in storage facility s, as a function of the storage capacity and the user-defined MinStorageCharge ratio.	Energy
StorageUpperLimit[r,s,y]>=0	Maximum allowed level of stored commodity in storage facility s. It corresponds to the total existing capacity of storage facility s (summing newly installed and pre-existing capacities).	Energy
AccumulatedNewStorageCapacity[r,s,y]>=0	Cumulative capacity of newly installed storage from the beginning of the time domain to year y.	Energy
NewStorageCapacity[r,s,y]>=0	Capacity of newly installed storage in year y.	Energy
CapitalInvestmentStorage[r,s,y]>=0	Undiscounted investment in new capacity for storage facility s. Derived from the NewStorageCapacity and the parameter CapitalCostStorage.	Monetary units
DiscountedCapitalInvestmentStorage[r,s,y]>=0	Investment in new capacity for storage facility s, discounted through the parameter DiscountRate.	Monetary units
SalvageVal-	Salvage value of storage facility s in year y, as a function of the	Monetary units

Capacity variables

Name	Description	
NumberOfNewTechnologyUnits[r,t,y]>=0, integer	Number of newly installed units of technology t in year y, as a function of the parameter CapacityOfOneTechnologyUnit. | No unit	
NewCapacity[r,t,y]>=0	Newly installed capacity of technology t in year y.	Power
AccumulatedNewCapacity[r,t,y]>=0	Cumulative newly installed capacity of technology t from the beginning of the time domain to year y.	Power
TotalCapacityAnnual[r,t,y]>=0	Total existing capacity of technology t in year y (sum of cumulative newly installed and pre-existing capacity).	Power

Activity variables

Name	Description	
RateOfActivity[r,l,t,m,y] >=0	Intermediate variable. It represents the activity of technology t in one mode of operation and in time slice l, if the latter lasted the whole year. | Energy (per year)	
RateOfTotalActivity[r,t,l,y] >=0	Sum of the RateOfActivity of a technology over the modes of operation.	Energy (per year)
TotalTechnologyAnnualActivity[r,t,y] >=0	Total annual activity of technology t.	Energy
TotalAnnualTechnologyActivityByMode[r,t,m,y] >=0	Annual activity of technology t in mode of operation m.	Energy
TotalTechnologyModelPeriodActivity[r,t]	Sum of the TotalTechnologyAnnualActivity over the years of the modelled period.	Energy
RateOfProductionByTechnologyByMode[r,l,t,m,f,y] >=0	Intermediate variable. It represents the quantity of fuel f that technology t would produce in one mode of operation and in time slice l, if the latter lasted the whole year. It is a function of the variable RateOfActivity and the parameter OutputActivityRatio.	Energy (per year)
RateOfProductionByTechnology[r,l,t,f,y] >=0	Sum of the RateOfProductionByTechnologyByMode over the modes of operation.	Energy (per year)
ProductionByTechnology[r,l,t,f,y] >=0	Production of fuel f by technology t in time slice l.	Energy
ProductionByTechnologyAnnual[r,t,f,y] >=0	Annual production of fuel f by technology t.	Energy
RateOfProduction[r,l,f,y] >=0	Sum of the RateOfProductionByTechnology over all the technologies.	Energy (per year)
Production[r,l,f,y] >=0	Total production of fuel f in time slice l. It is the sum of the ProductionByTechnology over all technologies.	Energy
RateOfUseByTechnologyByMode[r,l,t,m,f,y] >=0	Intermediate variable. It represents the quantity of fuel f that technology t would use in one mode of operation and in time slice l, if the latter lasted the whole year. It is the function of the variable RateOfActivity and the parameter InputActivityRatio.	Energy (per year)
RateOfUseByTechnology[r,l,t,f,y] >=0	Sum of the RateOfUseByTechnologyByMode over the modes of operation.	Energy (per year)
UseByTechnologyAnnual[r,t,f,y] >=0	Annual use of fuel f by technology t.	Energy
UseByTech-	Use of fuel f by technology t in time slice l.	Energy

Costing variables

Name	Description	
CapitalInvestment[r,t,y] >=0	Undiscounted investment in new capacity of technology t. It is a function of the NewCapacity and the parameter CapitalCost. | Monetary units	
DiscountedCapitalInvestment[r,t,y] >=0	Investment in new capacity of technology t, discounted through the parameter DiscountRate.	Monetary units
SalvageValue[r,t,y] >=0	Salvage value of technology t in year y, as a function of the parameters OperationalLife and DepreciationMethod.	Monetary units
DiscountedSalvageValue[r,t,y] >=0	Salvage value of technology t, discounted through the parameter DiscountRate.	Monetary units
OperatingCost[r,t,y] >=0	Undiscounted sum of the annual variable and fixed operating costs of technology t.	Monetary units
DiscountedOperatingCost[r,t,y] >=0	Annual OperatingCost of technology t, discounted through the parameter DiscountRate.	Monetary units
AnnualVariableOperatingCost[r,t,y] >=0	Annual variable operating cost of technology t. Derived from the TotalAnnualTechnologyActivityByMode and the parameter VariableCost.	Monetary units
AnnualFixedOperatingCost[r,t,y] >=0	Annual fixed operating cost of technology t. Derived from the TotalCapacityAnnual and the parameter FixedCost.	Monetary units
TotalDiscountedCostByTechnology[r,t,y] >=0	Difference between the sum of discounted operating cost / capital cost / emission penalties and the salvage value.	Monetary units
TotalDiscountedCost[r,y] >=0	Sum of the TotalDiscountedCostByTechnology over all the technologies.	Monetary units
ModelPeriodCostByRegion[r] >=0	Sum of the TotalDiscountedCost over all modelled years.	Monetary units

Reserve margin

Name	Description	
TotalCapacityInReserveMargin[r,y] >=0	Total available capacity of the technologies required to provide reserve margin. It is derived from the TotalCapacityAnnual and the parameter ReserveMarginTagTechnology. | Energy	
DemandNeedingReserveMargin[r,l,y] >=0	Quantity of fuel produced that is assigned to a target of reserve margin. Derived from the RateOfProduction and the parameter ReserveMarginTagFuel.	Energy (per year)
TotalREProductionAnnual[r,y]	Annual production by all technologies tagged as renewable in the model. Derived from the ProductionByTechnologyAnnual and the parameter RETagTechnology.	Energy
RETotalProductionOfTargetFuelAnnual[r,y]	Annual production of fuels tagged as renewable in the model. Derived from the RateOfProduction and the parameter RETagFuel.	Energy
AnnualTechnologyEmissionByMode[r,t,e,m,y] >=0	Annual emission of agent e by technology t in mode of operation m. Derived from the RateOfActivity and the parameter EmissionActivityRatio.	Quantity of emission
AnnualTechnologyEmission[r,t,e,y] >=0	Sum of the AnnualTechnologyEmissionByMode over the modes of operation.	Quantity of emission
AnnualTechnologyEmissionPenaltyByEmission[r,t,e,y] >=0	Undiscounted annual cost of emission e by technology t. It is a function of the AnnualTechnologyEmission and the parameter EmissionPenalty.	Monetary units
AnnualTechnologyEmissionsPenalty[r,t,y] >=0	Total undiscounted annual cost of all emissions generated by technology t. Sum of the AnnualTechnologyEmissionPenaltyByEmission over all the emitted agents.	Monetary units
DiscountedTechnologyEmissionsPenalty[r,t,y] >=0	Annual cost of emissions by technology t, discounted through the DiscountRate.	Monetary units
AnnualEmissions[r,e,y] >=0	Sum of the AnnualTechnologyEmission over all technologies.	Quantity of emission
ModelPeriodEmissions[r,e] >=0	Total system emissions of agent e in the model period, accounting for both the emissions by technologies and the user defined ModelPeriodExogenousEmission.	Quantity of emission

1.2.4 Equations

The OSeMOSYS code is divided into blocks of equations that comprises one objective function and several constraints. The blocks of code allow for a modular structure in different functionalities can be added or removed based on the user's needs. This flexible, modular structure allows OSeMOSYS to be used for a wide range of applications of different scales, complexities, and objectives.

Objective function

This equation represents the overall objective of the model. The default in OSeMOSYS is to minimise the total system cost, over the entire model period.

```
minimize cost: sum{r in REGION, y in YEAR} TotalDiscountedCost[r,y];
```

Rate of demand

The equation below is used to generate the term *RateOfDemand*, from the user-provided data for *SpecifiedAnnualDemand* and *SpecifiedDemandProfile*. The *RateOfDemand* is defined for each combination of commodity, TimeSlice and Year.

```
s.t. EQ_SpecifiedDemand{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:
        SpecifiedAnnualDemand[r,f,y]*SpecifiedDemandProfile[r,f,l,y] / YearSplit[l,
↪y]=RateOfDemand[r,l,f,y];
```

Capacity Adequacy A

Used to first calculate total capacity of each technology for each year based on existing capacity from before the model period (*ResidualCapacity*), *AccumulatedNewCapacity* during the modelling period, and *NewCapacity* installed in each year. It is then ensured that this Capacity is sufficient to meet the *RateOfTotalActivity* in each TimeSlice and Year. An additional constraint based on the size, or capacity, of each unit of a Technology is included (*CapacityOfOneTechnologyUnit*). This stipulates that the capacity of certain Technology can only be a multiple of the user-defined *CapacityOfOneTechnologyUnit*.

```
s.t. CAa1_TotalNewCapacity{r in REGION, t in TECHNOLOGY, y in YEAR}:
        AccumulatedNewCapacity[r,t,y] = sum{yy in YEAR: y-yy < OperationalLife[r,t] &&
↪ y-yy>=0} NewCapacity[r,t,yy];

s.t. CAa2_TotalAnnualCapacity{r in REGION, t in TECHNOLOGY, y in YEAR}:
        AccumulatedNewCapacity[r,t,y]+ ResidualCapacity[r,t,y] =␣
↪TotalCapacityAnnual[r,t,y];

s.t. CAa3_TotalActivityOfEachTechnology{r in REGION, t in TECHNOLOGY, l in TIMESLICE,␣
↪y in YEAR}:
        sum{m in MODE_OF_OPERATION} RateOfActivity[r,l,t,m,y] = RateOfTotalActivity[r,
↪t,l,y];

s.t. CAa4_Constraint_Capacity{r in REGION, l in TIMESLICE, t in TECHNOLOGY, y in YEAR}
↪:
        RateOfTotalActivity[r,t,l,y] <= TotalCapacityAnnual[r,t,y] * CapacityFactor[r,
↪t,l,y]*CapacityToActivityUnit[r,t];

s.t. CAa5_TotalNewCapacity{r in REGION, t in TECHNOLOGY, y in YEAR:
        CapacityOfOneTechnologyUnit[r,t,y]<>0}: CapacityOfOneTechnologyUnit[r,t,
↪y]*NumberOfNewTechnologyUnits[r,t,y] = NewCapacity[r,t,y];
```

NOTE: OSeMOSYS uses Mixed Integer Programming to solve models that define *CapacityOfTechnologyUnit*. Using this parameter is likely to increase the model computation time.

Capacity Adequacy B

Ensures that adequate capacity of technologies is present to at least meet the average annual demand.

```
s.t. CAb1_PlannedMaintenance{r in REGION, t in TECHNOLOGY, y in YEAR}:
        sum{l in TIMESLICE} RateOfTotalActivity[r,t,l,y]*YearSplit[l,y] <= sum{l in␣
↪TIMESLICE} (TotalCapacityAnnual[r,t,y]*CapacityFactor[r,t,l,y]*YearSplit[l,y])*␣
↪AvailabilityFactor[r,t,y]*CapacityToActivityUnit[r,t];
```

(continues on next page)

(continued from previous page)

```
```

Energy Balance A

Ensures that demand for each commodity is met in each TimeSlice.

```
s.t. EBa1_RateOfFuelProduction1{r in REGION, l in TIMESLICE, f in FUEL, t in␣
↪TECHNOLOGY, m in MODE_OF_OPERATION, y in YEAR: OutputActivityRatio[r,t,f,m,y] <>0}:␣
↪
        RateOfActivity[r,l,t,m,y]*OutputActivityRatio[r,t,f,m,y]  =␣
↪RateOfProductionByTechnologyByMode[r,l,t,m,f,y];

s.t. EBa2_RateOfFuelProduction2{r in REGION, l in TIMESLICE, f in FUEL, t in␣
↪TECHNOLOGY, y in YEAR}:
        sum{m in MODE_OF_OPERATION: OutputActivityRatio[r,t,f,m,y] <>0}␣
↪RateOfProductionByTechnologyByMode[r,l,t,m,f,y] = RateOfProductionByTechnology[r,l,
↪t,f,y];

s.t. EBa3_RateOfFuelProduction3{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:
        sum{t in TECHNOLOGY} RateOfProductionByTechnology[r,l,t,f,y]  = ␣
↪RateOfProduction[r,l,f,y];

s.t. EBa4_RateOfFuelUse1{r in REGION, l in TIMESLICE, f in FUEL, t in TECHNOLOGY, m␣
↪in MODE_OF_OPERATION, y in YEAR: InputActivityRatio[r,t,f,m,y]<>0}:
        RateOfActivity[r,l,t,m,y]*InputActivityRatio[r,t,f,m,y]  =␣
↪RateOfUseByTechnologyByMode[r,l,t,m,f,y];

s.t. EBa5_RateOfFuelUse2{r in REGION, l in TIMESLICE, f in FUEL, t in TECHNOLOGY, y␣
↪in YEAR}:
        sum{m in MODE_OF_OPERATION: InputActivityRatio[r,t,f,m,y]<>0}␣
↪RateOfUseByTechnologyByMode[r,l,t,m,f,y] = RateOfUseByTechnology[r,l,t,f,y];

s.t. EBa6_RateOfFuelUse3{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:
        sum{t in TECHNOLOGY} RateOfUseByTechnology[r,l,t,f,y]  = RateOfUse[r,l,f,y];

s.t. EBa7_EnergyBalanceEachTS1{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:
        RateOfProduction[r,l,f,y]*YearSplit[l,y] = Production[r,l,f,y];

s.t. EBa8_EnergyBalanceEachTS2{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:␣
↪
        RateOfUse[r,l,f,y]*YearSplit[l,y] = Use[r,l,f,y];

s.t. EBa9_EnergyBalanceEachTS3{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:
        RateOfDemand[r,l,f,y]*YearSplit[l,y] = Demand[r,l,f,y];

s.t. EBa10_EnergyBalanceEachTS4{r in REGION, rr in REGION, l in TIMESLICE, f in FUEL,␣
↪y in YEAR}:
        Trade[r,rr,l,f,y] = -Trade[rr,r,l,f,y];

s.t. EBa11_EnergyBalanceEachTS5{r in REGION, l in TIMESLICE, f in FUEL, y in YEAR}:
        Production[r,l,f,y] >= Demand[r,l,f,y] + Use[r,l,f,y] + sum{rr in REGION}␣
↪Trade[r,rr,l,f,y]*TradeRoute[r,rr,f,y];
```

Energy Balance B

Ensures that demand for each commodity is met in each Year.

```
s.t. EBb1_EnergyBalanceEachYear1{r in REGION, f in FUEL, y in YEAR}:
        sum{l in TIMESLICE} Production[r,l,f,y] = ProductionAnnual[r,f,y];

s.t. EBb2_EnergyBalanceEachYear2{r in REGION, f in FUEL, y in YEAR}:
        sum{l in TIMESLICE} Use[r,l,f,y] = UseAnnual[r,f,y];

s.t. EBb3_EnergyBalanceEachYear3{r in REGION, rr in REGION, f in FUEL, y in YEAR}:
        sum{l in TIMESLICE} Trade[r,rr,l,f,y] = TradeAnnual[r,rr,f,y];

s.t. EBb4_EnergyBalanceEachYear4{r in REGION, f in FUEL, y in YEAR}:
        ProductionAnnual[r,f,y] >= UseAnnual[r,f,y] + sum{rr in REGION} TradeAnnual[r,
↪rr,f,y]*TradeRoute[r,rr,f,y] + AccumulatedAnnualDemand[r,f,y];
```

Accounting Technology Production/Use

Accounting equations used to generate specific intermediate variables: *ProductionByTechnology*, *UseBytechnology*, *TotalAnnualTechnologyActivityByMode*, and *ModelPeriodCostByRegion*.

```
s.t. Acc1_FuelProductionByTechnology{r in REGION, l in TIMESLICE, t in TECHNOLOGY, f␣
↪in FUEL, y in YEAR}:
        RateOfProductionByTechnology[r,l,t,f,y] * YearSplit[l,y] =␣
↪ProductionByTechnology[r,l,t,f,y];

s.t. Acc2_FuelUseByTechnology{r in REGION, l in TIMESLICE, t in TECHNOLOGY, f in FUEL,
↪ y in YEAR}:
        RateOfUseByTechnology[r,l,t,f,y] * YearSplit[l,y] = UseByTechnology[r,l,t,f,
↪y];

s.t. Acc3_AverageAnnualRateOfActivity{r in REGION, t in TECHNOLOGY, m in MODE_OF_
↪OPERATION, y in YEAR}:
        sum{l in TIMESLICE} RateOfActivity[r,l,t,m,y]*YearSplit[l,y] =␣
↪TotalAnnualTechnologyActivityByMode[r,t,m,y];

s.t. Acc4_ModelPeriodCostByRegion{r in REGION}:
        sum{y in YEAR}TotalDiscountedCost[r,y] = ModelPeriodCostByRegion[r];
```

Storage Equations

```
s.t. S1_RateOfStorageCharge{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE,␣
↪lh in DAILYTIMEBRACKET, y in YEAR}:
        sum{t in TECHNOLOGY, m in MODE_OF_OPERATION, l in␣
↪TIMESLICE:TechnologyToStorage[r,t,s,m]>0} RateOfActivity[r,l,t,m,y] *␣
↪TechnologyToStorage[r,t,s,m] * Conversionls[l,ls] * Conversionld[l,ld] *␣
↪Conversionlh[l,lh] = RateOfStorageCharge[r,s,ls,ld,lh,y];

s.t. S2_RateOfStorageDischarge{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE,
↪ lh in DAILYTIMEBRACKET, y in YEAR}:
        sum{t in TECHNOLOGY, m in MODE_OF_OPERATION, l in␣
↪TIMESLICE:TechnologyFromStorage[r,t,s,m]>0} RateOfActivity[r,l,t,m,y] *␣
↪TechnologyFromStorage[r,t,s,m] * Conversionls[l,ls] * Conversionld[l,ld] *␣
↪Conversionlh[l,lh] = RateOfStorageDischarge[r,s,ls,ld,lh,y];
```

(continues on next page)

(continued from previous page)

```
s.t. S3_NetChargeWithinYear{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE,␣
↪lh in DAILYTIMEBRACKET, y in YEAR}:
        sum{l in TIMESLICE:Conversionls[l,ls]>0&&Conversionld[l,ld]>0&&Conversionlh[l,
↪lh]>0}  (RateOfStorageCharge[r,s,ls,ld,lh,y] - RateOfStorageDischarge[r,s,ls,ld,lh,
↪y]) * YearSplit[l,y] * Conversionls[l,ls] * Conversionld[l,ld] * Conversionlh[l,lh]␣
↪= NetChargeWithinYear[r,s,ls,ld,lh,y];

s.t. S4_NetChargeWithinDay{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh␣
↪in DAILYTIMEBRACKET, y in YEAR}:
        (RateOfStorageCharge[r,s,ls,ld,lh,y] - RateOfStorageDischarge[r,s,ls,ld,lh,
↪y]) * DaySplit[lh,y] = NetChargeWithinDay[r,s,ls,ld,lh,y];

s.t. S5_and_S6_StorageLevelYearStart{r in REGION, s in STORAGE, y in YEAR}: if y = min
↪{yy in YEAR} min(yy) then StorageLevelStart[r,s]

↪StorageLevelYearStart[r,s,y-1] + sum{ls in SEASON, ld in DAYTYPE, lh in␣
↪DAILYTIMEBRACKET} NetChargeWithinYear[r,s,ls,ld,lh,y-1]

↪StorageLevelYearStart[r,s,y];

s.t. S7_and_S8_StorageLevelYearFinish{r in REGION, s in STORAGE, y in YEAR}: if y <␣
↪max{yy in YEAR} max(yy) then StorageLevelYearStart[r,s,y+1]

↪StorageLevelYearStart[r,s,y] + sum{ls in SEASON, ld in DAYTYPE, lh in␣
↪DAILYTIMEBRACKET} NetChargeWithinYear[r,s,ls,ld,lh,y]

↪StorageLevelYearFinish[r,s,y];

s.t. S9_and_S10_StorageLevelSeasonStart{r in REGION, s in STORAGE, ls in SEASON, y in␣
↪YEAR}: if ls = min{lsls in SEASON} min(lsls) then StorageLevelYearStart[r,s,y]

↪StorageLevelSeasonStart[r,s,ls-1,y] + sum{ld in DAYTYPE, lh in DAILYTIMEBRACKET}␣
↪NetChargeWithinYear[r,s,ls-1,ld,lh,y]

↪StorageLevelSeasonStart[r,s,ls,y];

s.t. S11_and_S12_StorageLevelDayTypeStart{r in REGION, s in STORAGE, ls in SEASON, ld␣
↪in DAYTYPE, y in YEAR}: if ld = min{ldld in DAYTYPE} min(ldld) then␣
↪StorageLevelSeasonStart[r,s,ls,y]

↪StorageLevelDayTypeStart[r,s,ls,ld-1,y] + sum{lh in DAILYTIMEBRACKET}␣
↪NetChargeWithinDay[r,s,ls,ld-1,lh,y] * DaysInDayType[ls,ld-1,y]

↪StorageLevelDayTypeStart[r,s,ls,ld,y];

s.t. S13_and_S14_and_S15_StorageLevelDayTypeFinish{r in REGION, s in STORAGE, ls in␣
↪SEASON, ld in DAYTYPE, y in YEAR}:        if ls = max{lsls in SEASON} max(lsls) &&␣
↪ld = max{ldld in DAYTYPE} max(ldld) then StorageLevelYearFinish[r,s,y]

↪if ld = max{ldld in DAYTYPE} max(ldld) then StorageLevelSeasonStart[r,s,ls+1,y]

↪StorageLevelDayTypeFinish[r,s,ls,ld+1,y] - sum{lh in DAILYTIMEBRACKET}␣
↪NetChargeWithinDay[r,s,ls,ld+1,lh,y] * DaysInDayType[ls,ld+1,y]

↪StorageLevelDayTypeFinish[r,s,ls,ld,y];
```

Storage Constraints

```
s.t. SC1_LowerLimit_
↪BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInFirstWeekConstraint{r in␣
↪REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in␣
↪YEAR}:
        0 <= (StorageLevelDayTypeStart[r,s,ls,ld,y]+sum{lhlh in DAILYTIMEBRACKET:lh-
↪lhlh>0} NetChargeWithinDay[r,s,ls,ld,lhlh,y])-StorageLowerLimit[r,s,y];
↪

s.t. SC1_UpperLimit_
↪BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInFirstWeekConstraint{r in␣
↪REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in␣
↪YEAR}:
        (StorageLevelDayTypeStart[r,s,ls,ld,y]+sum{lhlh in DAILYTIMEBRACKET:lh-lhlh>0}
↪ NetChargeWithinDay[r,s,ls,ld,lhlh,y])-StorageUpperLimit[r,s,y] <= 0;
↪

s.t. SC2_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInFirstWeekConstraint
↪{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y␣
↪in YEAR}:
        0 <= if ld > min{ldld in DAYTYPE} min(ldld) then (StorageLevelDayTypeStart[r,
↪s,ls,ld,y]-sum{lhlh in DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[r,s,ls,ld-1,
↪lhlh,y])-StorageLowerLimit[r,s,y];
↪

s.t. SC2_UpperLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInFirstWeekConstraint
↪{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y␣
↪in YEAR}:
        if ld > min{ldld in DAYTYPE} min(ldld) then (StorageLevelDayTypeStart[r,s,ls,
↪ld,y]-sum{lhlh in DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[r,s,ls,ld-1,lhlh,
↪y])-StorageUpperLimit[r,s,y] <= 0;

s.t. SC3_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInLastWeekConstraint
↪{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y␣
↪in YEAR}:
        0 <= (StorageLevelDayTypeFinish[r,s,ls,ld,y] - sum{lhlh in␣
↪DAILYTIMEBRACKET:lh-lhlh<0} NetChargeWithinDay[r,s,ls,ld,lhlh,y])-
↪StorageLowerLimit[r,s,y];
↪

s.t. SC3_UpperLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInLastWeekConstraint
↪{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y␣
↪in YEAR}:
        (StorageLevelDayTypeFinish[r,s,ls,ld,y] - sum{lhlh in DAILYTIMEBRACKET:lh-lhlh
↪<0} NetChargeWithinDay[r,s,ls,ld,lhlh,y])-StorageUpperLimit[r,s,y] <= 0;

s.t. SC4_LowerLimit_
↪BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInLastWeekConstraint{r in REGION,
↪ s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}:␣
↪
        0 <= if ld > min{ldld in DAYTYPE} min(ldld) then (StorageLevelDayTypeFinish[r,
↪s,ls,ld-1,y]+sum{lhlh in DAILYTIMEBRACKET:lh-lhlh>0} NetChargeWithinDay[r,s,ls,ld,
↪lhlh,y])-StorageLowerLimit[r,s,y];

s.t. SC4_UpperLimit_
↪BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInLastWeekConstraint{r in REGION,
↪ s in STORAGE, ls in SEASON, ld in DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}:
```

(continues on next page)

(continued from previous page)

```
        if ld > min{ldld in DAYTYPE} min(ldld) then (StorageLevelDayTypeFinish[r,s,ls,
↪ld-1,y]+sum{lhlh in DAILYTIMEBRACKET:lh-lhlh>0} NetChargeWithinDay[r,s,ls,ld,lhlh,
↪y])-StorageUpperLimit[r,s,y] <= 0;
↪

s.t. SC5_MaxChargeConstraint{r in REGION, s in STORAGE, ls in SEASON, ld in DAYTYPE,␣
↪lh in DAILYTIMEBRACKET, y in YEAR}: RateOfStorageCharge[r,s,ls,ld,lh,y] <=␣
↪StorageMaxChargeRate[r,s];

s.t. SC6_MaxDischargeConstraint{r in REGION, s in STORAGE, ls in SEASON, ld in␣
↪DAYTYPE, lh in DAILYTIMEBRACKET, y in YEAR}: RateOfStorageDischarge[r,s,ls,ld,lh,y]
↪<= StorageMaxDischargeRate[r,s];
```

Storage Investments

Calculates the total discounted capital costs expenditure for each storage technology in each year.

```
s.t. SI1_StorageUpperLimit{r in REGION, s in STORAGE, y in YEAR}:
        AccumulatedNewStorageCapacity[r,s,y]+ResidualStorageCapacity[r,s,y] =␣
↪StorageUpperLimit[r,s,y];

s.t. SI2_StorageLowerLimit{r in REGION, s in STORAGE, y in YEAR}:
        MinStorageCharge[r,s,y]*StorageUpperLimit[r,s,y] = StorageLowerLimit[r,s,y];

s.t. SI3_TotalNewStorage{r in REGION, s in STORAGE, y in YEAR}:
        sum{yy in YEAR: y-yy < OperationalLifeStorage[r,s] && y-yy>=0}␣
↪NewStorageCapacity[r,s,yy]=AccumulatedNewStorageCapacity[r,s,y];

s.t. SI4_UndiscountedCapitalInvestmentStorage{r in REGION, s in STORAGE, y in YEAR}:
        CapitalCostStorage[r,s,y] * NewStorageCapacity[r,s,y] =␣
↪CapitalInvestmentStorage[r,s,y];

s.t. SI5_DiscountingCapitalInvestmentStorage{r in REGION, s in STORAGE, y in YEAR}:
        CapitalInvestmentStorage[r,s,y]/((1+DiscountRate[r])^(y-min{yy in YEAR}␣
↪min(yy))) = DiscountedCapitalInvestmentStorage[r,s,y];

s.t. SI6_SalvageValueStorageAtEndOfPeriod1{r in REGION, s in STORAGE, y in YEAR:
        (y+OperationalLifeStorage[r,s]-1) <= (max{yy in YEAR} max(yy))}: 0 =␣
↪SalvageValueStorage[r,s,y];

s.t. SI7_SalvageValueStorageAtEndOfPeriod2{r in REGION, s in STORAGE, y in YEAR:
        (DepreciationMethod[r]=1 && (y+OperationalLifeStorage[r,s]-1) > (max{yy in␣
↪YEAR} max(yy)) && DiscountRate[r]=0) || (DepreciationMethod[r]=2 &&␣
↪(y+OperationalLifeStorage[r,s]-1) > (max{yy in YEAR} max(yy)))}:␣
↪CapitalInvestmentStorage[r,s,y]*(1-(max{yy in YEAR} max(yy) - y+1)/
↪OperationalLifeStorage[r,s]) = SalvageValueStorage[r,s,y];

s.t. SI8_SalvageValueStorageAtEndOfPeriod3{r in REGION, s in STORAGE, y in YEAR:
        DepreciationMethod[r]=1 && (y+OperationalLifeStorage[r,s]-1) > (max{yy in␣
↪YEAR} max(yy)) && DiscountRate[r]>0}: CapitalInvestmentStorage[r,s,y]*(1-
↪(((1+DiscountRate[r])^(max{yy in YEAR} max(yy) - y+1)-1)/((1+DiscountRate[r])^
↪OperationalLifeStorage[r,s]-1))) = SalvageValueStorage[r,s,y];

s.t. SI9_SalvageValueStorageDiscountedToStartYear{r in REGION, s in STORAGE, y in␣
↪YEAR}:
```

(continues on next page)

(continued from previous page)

```
        SalvageValueStorage[r,s,y]/((1+DiscountRate[r])^(max{yy in YEAR} max(yy)-min
↪{yy in YEAR} min(yy)+1)) = DiscountedSalvageValueStorage[r,s,y];

s.t. SI10_TotalDiscountedCostByStorage{r in REGION, s in STORAGE, y in YEAR}:
        DiscountedCapitalInvestmentStorage[r,s,y]-DiscountedSalvageValueStorage[r,s,
↪y] = TotalDiscountedStorageCost[r,s,y];
```

Capital Costs

Calculates the total discounted capital cost expenditure for each technology in each year.

```
s.t. CC1_UndiscountedCapitalInvestment{r in REGION, t in TECHNOLOGY, y in YEAR}:
        CapitalCost[r,t,y] * NewCapacity[r,t,y] = CapitalInvestment[r,t,y];

s.t. CC2_DiscountingCapitalInvestment{r in REGION, t in TECHNOLOGY, y in YEAR}:
        CapitalInvestment[r,t,y]/((1+DiscountRate[r])^(y-min{yy in YEAR} min(yy))) =␣
↪DiscountedCapitalInvestment[r,t,y];
```

Salvage Value

Calculates the fraction of the initial capital cost that can be recouped at the end of a technologies operational life. The salvage value can be calculated using one of two depreciation methods: straight line and sinking fund.

```
s.t. SV1_SalvageValueAtEndOfPeriod1{r in REGION, t in TECHNOLOGY, y in YEAR:
        DepreciationMethod[r]=1 && (y + OperationalLife[r,t]-1) > (max{yy in YEAR}␣
↪max(yy)) && DiscountRate[r]>0}: SalvageValue[r,t,y] = CapitalCost[r,t,
↪y]*NewCapacity[r,t,y]*(1-(((1+DiscountRate[r])^(max{yy in YEAR} max(yy) - y+1)-1)/
↪((1+DiscountRate[r])^OperationalLife[r,t]-1)));

s.t. SV2_SalvageValueAtEndOfPeriod2{r in REGION, t in TECHNOLOGY, y in YEAR:
        (DepreciationMethod[r]=1 && (y + OperationalLife[r,t]-1) > (max{yy in YEAR}␣
↪max(yy)) && DiscountRate[r]=0) || (DepreciationMethod[r]=2 && (y +␣
↪OperationalLife[r,t]-1) > (max{yy in YEAR} max(yy)))}: SalvageValue[r,t,y] =␣
↪CapitalCost[r,t,y]*NewCapacity[r,t,y]*(1-(max{yy in YEAR} max(yy) - y+1)/
↪OperationalLife[r,t]);

s.t. SV3_SalvageValueAtEndOfPeriod3{r in REGION, t in TECHNOLOGY, y in YEAR:
        (y + OperationalLife[r,t]-1) <= (max{yy in YEAR} max(yy))}: SalvageValue[r,t,
↪y] = 0;

s.t. SV4_SalvageValueDiscountedToStartYear{r in REGION, t in TECHNOLOGY, y in YEAR}:
        DiscountedSalvageValue[r,t,y] = SalvageValue[r,t,y]/((1+DiscountRate[r])^
↪(1+max{yy in YEAR} max(yy)-min{yy in YEAR} min(yy)));
```

Operating Costs

Calculates the total variable and fixed operating costs for each technology, in each year.

```
s.t. OC1_OperatingCostsVariable{r in REGION, t in TECHNOLOGY, l in TIMESLICE, y in␣
↪YEAR}:
        sum{m in MODE_OF_OPERATION} TotalAnnualTechnologyActivityByMode[r,t,m,
↪y]*VariableCost[r,t,m,y] = AnnualVariableOperatingCost[r,t,y];
```

(continues on next page)

(continued from previous page)

```
s.t. OC2_OperatingCostsFixedAnnual{r in REGION, t in TECHNOLOGY, y in YEAR}:
        TotalCapacityAnnual[r,t,y]*FixedCost[r,t,y] = AnnualFixedOperatingCost[r,t,y];

s.t. OC3_OperatingCostsTotalAnnual{r in REGION, t in TECHNOLOGY, y in YEAR}:
        AnnualFixedOperatingCost[r,t,y]+AnnualVariableOperatingCost[r,t,y] =␣
↪OperatingCost[r,t,y];

s.t. OC4_DiscountedOperatingCostsTotalAnnual{r in REGION, t in TECHNOLOGY, y in YEAR}
↪:
        OperatingCost[r,t,y]/((1+DiscountRate[r])^(y-min{yy in YEAR} min(yy)+0.5)) =␣
↪DiscountedOperatingCost[r,t,y];
```

Total Discounted Costs

Calculates the total discounted system cost over the entire model period to give the *TotalDiscountedCost*. This is the variable that is minimized in the model's objective function.

```
s.t. TDC1_TotalDiscountedCostByTechnology{r in REGION, t in TECHNOLOGY, y in YEAR}:
        DiscountedOperatingCost[r,t,y]+DiscountedCapitalInvestment[r,t,
↪y]+DiscountedTechnologyEmissionsPenalty[r,t,y]-DiscountedSalvageValue[r,t,y] =␣
↪TotalDiscountedCostByTechnology[r,t,y];

s.t. TDC2_TotalDiscountedCost{r in REGION, y in YEAR}:
        sum{t in TECHNOLOGY} TotalDiscountedCostByTechnology[r,t,y]+sum{s in STORAGE}␣
↪TotalDiscountedStorageCost[r,s,y] = TotalDiscountedCost[r,y];
```

Total Capacity Constraints

Ensures that the total capacity of each technology in each year is greater than and less than the user-defined parameters *TotalAnnualMinCapacityInvestment* and *TotalAnnualMaxCapacityInvestment* respectively.

```
s.t. TCC1_TotalAnnualMaxCapacityConstraint{r in REGION, t in TECHNOLOGY, y in YEAR}:
        TotalCapacityAnnual[r,t,y] <= TotalAnnualMaxCapacity[r,t,y];

s.t. TCC2_TotalAnnualMinCapacityConstraint{r in REGION, t in TECHNOLOGY, y in YEAR:
        TotalAnnualMinCapacity[r,t,y]>0}: TotalCapacityAnnual[r,t,y] >=␣
↪TotalAnnualMinCapacity[r,t,y];
```

New Capacity Constraints

Ensures that the new capacity of each technology installed in each year is greater than and less than the user-defined parameters *TotalAnnualMinCapacityInvestment* and *TotalAnnualMaxCapacityInvestment* respectively.

```
s.t. NCC1_TotalAnnualMaxNewCapacityConstraint{r in REGION, t in TECHNOLOGY, y in YEAR}
↪:
        NewCapacity[r,t,y] <= TotalAnnualMaxCapacityInvestment[r,t,y];

s.t. NCC2_TotalAnnualMinNewCapacityConstraint{r in REGION, t in TECHNOLOGY, y in␣
↪YEAR:
        TotalAnnualMinCapacityInvestment[r,t,y]>0}: NewCapacity[r,t,y] >=␣
↪TotalAnnualMinCapacityInvestment[r,t,y];
```

Annual Activity Constraints

Ensures that the total activity of each technology over each year is greater than and less than the user-defined parameters *TotalTechnologyAnnualActivityLowerLimit* and *TotalTechnologyAnnualActivityUpperLimit* respectively.

```
s.t. AAC1_TotalAnnualTechnologyActivity{r in REGION, t in TECHNOLOGY, y in YEAR}:
        sum{l in TIMESLICE} RateOfTotalActivity[r,t,l,y]*YearSplit[l,y] =␣
↪TotalTechnologyAnnualActivity[r,t,y];

s.t. AAC2_TotalAnnualTechnologyActivityUpperLimit{r in REGION, t in TECHNOLOGY, y in␣
↪YEAR}:
        TotalTechnologyAnnualActivity[r,t,y] <=␣
↪TotalTechnologyAnnualActivityUpperLimit[r,t,y] ;

s.t. AAC3_TotalAnnualTechnologyActivityLowerLimit{r in REGION, t in TECHNOLOGY, y in␣
↪YEAR:
        TotalTechnologyAnnualActivityLowerLimit[r,t,y]>0}:␣
↪TotalTechnologyAnnualActivity[r,t,y] >= TotalTechnologyAnnualActivityLowerLimit[r,t,
↪y] ;
```

Total Activity Constraints

Ensures that the total activity of each technology over the entire model period is greater than and less than the user-defined parameters *TotalTechnologyModelPeriodActivityLowerLimit* and *TotalTechnologyModelPeriodActivityUpperLimit* respectively.

```
s.t. TAC1_TotalModelHorizonTechnologyActivity{r in REGION, t in TECHNOLOGY}:
        sum{y in YEAR} TotalTechnologyAnnualActivity[r,t,y] =␣
↪TotalTechnologyModelPeriodActivity[r,t];

s.t. TAC2_TotalModelHorizonTechnologyActivityUpperLimit{r in REGION, t in TECHNOLOGY:
        TotalTechnologyModelPeriodActivityUpperLimit[r,t]>0}:␣
↪TotalTechnologyModelPeriodActivity[r,t] <=␣
↪TotalTechnologyModelPeriodActivityUpperLimit[r,t] ;

s.t. TAC3_TotalModelHorizenTechnologyActivityLowerLimit{r in REGION, t in TECHNOLOGY:
        TotalTechnologyModelPeriodActivityLowerLimit[r,t]>0}:␣
↪TotalTechnologyModelPeriodActivity[r,t] >=␣
↪TotalTechnologyModelPeriodActivityLowerLimit[r,t] ;
```

Reserve Margin Constraints

Ensures that sufficient reserve capacity of specific technologies (*ReserveMarginTagTechnology* = 1) is installed such that the user-defined *ReserveMargin* is maintained.

```
s.t. RM1_ReserveMargin_TechnologiesIncluded_In_Activity_Units{r in REGION, l in␣
↪TIMESLICE, y in YEAR}:
        sum {t in TECHNOLOGY} TotalCapacityAnnual[r,t,y] *␣
↪ReserveMarginTagTechnology[r,t,y] * CapacityToActivityUnit[r,t]          =␣
↪         TotalCapacityInReserveMargin[r,y];

s.t. RM2_ReserveMargin_FuelsIncluded{r in REGION, l in TIMESLICE, y in YEAR}:
        sum {f in FUEL} RateOfProduction[r,l,f,y] * ReserveMarginTagFuel[r,f,y] =␣
↪DemandNeedingReserveMargin[r,l,y];
```

(continues on next page)

(continued from previous page)

```
s.t. RM3_ReserveMargin_Constraint{r in REGION, l in TIMESLICE, y in YEAR}:
        DemandNeedingReserveMargin[r,l,y] * ReserveMargin[r,y]<=␣
↪TotalCapacityInReserveMargin[r,y];
```

RE Production Target

Ensures that production from technologies tagged as renewable energy technologies (*RETagTechnology* = 1) is at least equal to the user-defined renewable energy (RE) target.

```
s.t. RE1_FuelProductionByTechnologyAnnual{r in REGION, t in TECHNOLOGY, f in FUEL, y␣
↪in YEAR}:
        sum{l in TIMESLICE} ProductionByTechnology[r,l,t,f,y] =␣
↪ProductionByTechnologyAnnual[r,t,f,y];

s.t. RE2_TechIncluded{r in REGION, y in YEAR}:
        sum{t in TECHNOLOGY, f in FUEL} ProductionByTechnologyAnnual[r,t,f,
↪y]*RETagTechnology[r,t,y] = TotalREProductionAnnual[r,y];

s.t. RE3_FuelIncluded{r in REGION, y in YEAR}:
        sum{l in TIMESLICE, f in FUEL} RateOfProduction[r,l,f,y]*YearSplit[l,
↪y]*RETagFuel[r,f,y] = RETotalProductionOfTargetFuelAnnual[r,y];

s.t. RE4_EnergyConstraint{r in REGION, y in YEAR}:
        REMinProductionTarget[r,y]*RETotalProductionOfTargetFuelAnnual[r,y] <=␣
↪TotalREProductionAnnual[r,y];

s.t. RE5_FuelUseByTechnologyAnnual{r in REGION, t in TECHNOLOGY, f in FUEL, y in YEAR}
↪:
        sum{l in TIMESLICE} RateOfUseByTechnology[r,l,t,f,y]*YearSplit[l,y] =␣
↪UseByTechnologyAnnual[r,t,f,y];
```

Emissions Accounting

Calculates the annual and model period emissions from each technology, for each type of emission. It also calculates the total associated emission penalties, if any. Finally, it ensures that emissions are maintained before stipulated limits that may be defined for each year and/or the entire model period.

```
s.t. E1_AnnualEmissionProductionByMode{r in REGION, t in TECHNOLOGY, e in EMISSION, m␣
↪in MODE_OF_OPERATION, y in YEAR}:
        EmissionActivityRatio[r,t,e,m,y]*TotalAnnualTechnologyActivityByMode[r,t,m,
↪y]=AnnualTechnologyEmissionByMode[r,t,e,m,y];

s.t. E2_AnnualEmissionProduction{r in REGION, t in TECHNOLOGY, e in EMISSION, y in␣
↪YEAR}:
        sum{m in MODE_OF_OPERATION} AnnualTechnologyEmissionByMode[r,t,e,m,y] =␣
↪AnnualTechnologyEmission[r,t,e,y];

s.t. E3_EmissionsPenaltyByTechAndEmission{r in REGION, t in TECHNOLOGY, e in EMISSION,
↪ y in YEAR}:
        AnnualTechnologyEmission[r,t,e,y]*EmissionsPenalty[r,e,y] =␣
↪AnnualTechnologyEmissionPenaltyByEmission[r,t,e,y];
```

(continues on next page)

(continued from previous page)

```
s.t. E4_EmissionsPenaltyByTechnology{r in REGION, t in TECHNOLOGY, y in YEAR}:
        sum{e in EMISSION} AnnualTechnologyEmissionPenaltyByEmission[r,t,e,y] =␣
↪AnnualTechnologyEmissionsPenalty[r,t,y];

s.t. E5_DiscountedEmissionsPenaltyByTechnology{r in REGION, t in TECHNOLOGY, y in␣
↪YEAR}:
        AnnualTechnologyEmissionsPenalty[r,t,y]/((1+DiscountRate[r])^(y-min{yy in␣
↪YEAR} min(yy)+0.5)) = DiscountedTechnologyEmissionsPenalty[r,t,y];

s.t. E6_EmissionsAccounting1{r in REGION, e in EMISSION, y in YEAR}:
        sum{t in TECHNOLOGY} AnnualTechnologyEmission[r,t,e,y] = AnnualEmissions[r,e,
↪y];

s.t. E7_EmissionsAccounting2{r in REGION, e in EMISSION}:
        sum{y in YEAR} AnnualEmissions[r,e,y] = ModelPeriodEmissions[r,e]-␣
↪ModelPeriodExogenousEmission[r,e];

s.t. E8_AnnualEmissionsLimit{r in REGION, e in EMISSION, y in YEAR}:
        AnnualEmissions[r,e,y]+AnnualExogenousEmission[r,e,y] <=␣
↪AnnualEmissionLimit[r,e,y];

s.t. E9_ModelPeriodEmissionsLimit{r in REGION, e in EMISSION}:
        ModelPeriodEmissions[r,e] <= ModelPeriodEmissionLimit[r,e];
```

1.3 Code versions

Currently the OSeMOSYS code is available in two versions:

- The *"long code"*, which includes a full version of equations, and
- The *"short code"*, which was developed by merging equations from the long version, in order to reduce the computational time.

The long version is easier to read and understand, as it includes more simple, user-friendly equations. The way the shorter version was developed instead, allows for eliminating the need to calculate and store intermediate values and results in faster calculation times. Key outputs remains the same.

The long version computes more variables, which help the user in understanding what the model really does. The long version is therefore particularly helpful also to test modifications and enhancements to the main code. Therefore, the use of the long version is recommended unless the calculation times get prohibitively long.

Problems which cannot be solved with the long code, due to time or RAM constraints, can be solved with the short code. To check if the RAM is a limiting constraint, during the computation it is sufficient to open the system task manager (for Windows-based machines, you can use the command Ctrl+Alt+Delete) and check if the RAM use is close to saturation. The RAM and time constraints may become binding when the model gets too complex. This may happen, e.g., when a multi-regional model is set up, with many technologies and time slices. In such cases, the shorter version of the code is recommended.

The reduction in the number of equations in the short code translates into the generation of a smaller matrix of results to be solved. This significantly reduces the memory usage and the processing time, by 10 times (10x) and 5 times (5x) respectively.

The short version of the OSeMOSYS code contains only the essential equations required for running the model. However, all the previous equations have been left as before, and "commented out" to better understand the methodology followed to shorten the code. It is important to note that the shortening of the code does not change any aspect of the functionality of OSeMOSYS. Furthermore, there are no special formatting requirements for the data file necessary to run the short code instead of the long one. Both the long and the short versions of OSeMOSYS code are developed and released as parallel versions simultaneously, every time a change is made or a functionality is added.

The latest versions of the OSeMOSYS code were released on the 8th of November 2017, both for the long and the short code, and are available for download on the related website.

1.4 Create a model in OSeMOSYS

This section introduces the user to the basic components of any application of OSeMOSYS and describes the steps for the creation of a model. To this end, a sample case study, Atlantis, is used and examples from it are shown throughout the section. The concepts described here apply to any application, with due adjustments, and they are valid both in case interfaces are used or not.

In the following subsections, *The Atlantis Case Study* describes the Atlantis sample case study for OSeMOSYS, representing a country sharing features of developed and developing ones. *How to build a model in OSeMOSYS* shows the step-by-step creation of a model without using interfaces.

1.4.1 The Atlantis Case Study

Atlantis is a country with a total population of 10 million people. 40% of the population lives in urban areas, distribute in 1.25 million households. The rest lives in rural areas, within 923 thousand households. The total population is expected to reach 15.9 million people by 2040 with an average annual growth rate of 1.8%.

Currently, Atlantis imports 100% of its fuels. It relies on 5 power plant types to meet its electricity demand, each running on a single type of fuel: one large hydropower plant, natural gas-fired Single Cycle Steam Turbine, heavy fuel oil-fired Single Cycle Steam Turbine, diesel-fed Open Cycle Gas Turbine and coal-based Integrated Gasification Combined Cycle facility. Distributed diesel generators are the main source of electricity in many rural areas.

Over the modelling period, this system will be expanded to explore the feasibility of including the following new technologies:

- wind turbines (25% load factor);
- mini hydro power plants (less than 1 MW);
- concentrated solar power plants (CSP);
- grid connected PV systems (Commercial);
- rooftop PV systems (in residential areas);
- a nuclear power plant (light water reactor);
- a new Combined Cycle Power Plant running on natural gas.

The Atlantis input data file can be accessed here

Mapping the RES of Atlantis

When developing a model using an optimization tool like OSeMOSYS, the energy/electricity system needs to be mapped to identify all the relevant technologies and fuels that will be involved in the analysis. The schematic representation of the system for such purposes is called Reference Energy System (RES).

Figure 1 illustrates the RES of Atlantis, where the lines represent energy carriers (e.g., crude oil, coal etc.), while the blocks represent transformation technologies (e.g., power plants, transmission and distribution stations etc.). The RES can be read from the left to the right. On the left hand side, the primary energy resources are represented. They can be extracted domestically, imported or both. Extraction and import processes are represented as technologies in the RES (i.e. black boxes with outgoing lines representing the fuels they make available). Importantly, each chain must always start with a technology. Moving from the left to the right, the energy carriers are transformed by different technologies, each with a user-defined transfer function, to ultimately meet the final demand for energy or services, presented by the lines on the far right hand side.

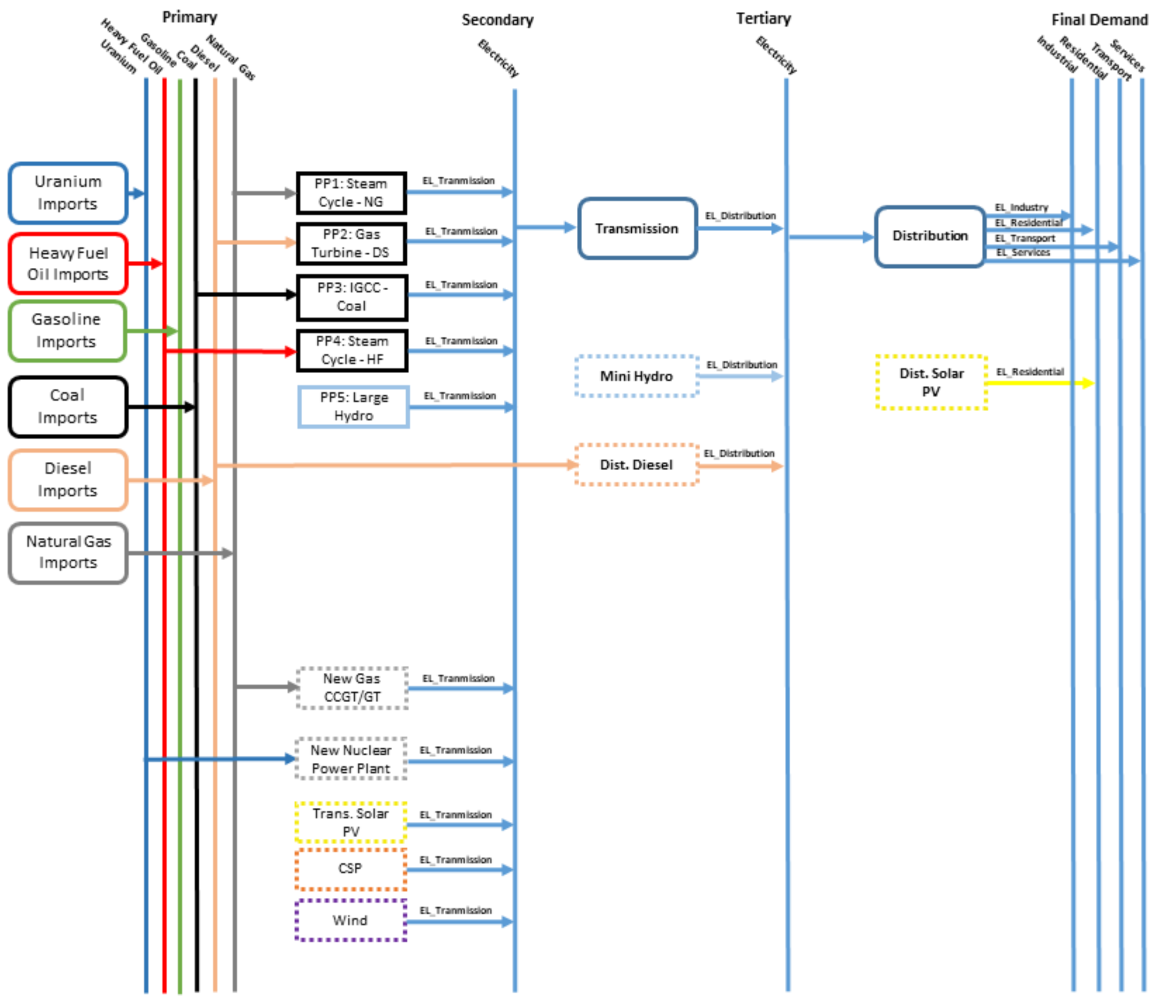

Fig. 1: The Reference Energy System (RES) of the Atlantis case study.

1.4.2 How to build a model in OSeMOSYS

Downloading OSeMOSYS and GLPK

To develop an energy systems model using OSeMOSYS and optimize it using GLPSOL (GNU- Linear programming solver), we will need two different files: one OSeMOSYS code file and a data/model file representing the data corresponding to the model developed by the user.

The OSeMOSYS code (both the long and short versions) can be downloaded from the OSeMOSYS GitHub repository or the OSeMOSYS website; the latter also contains older versions of the code. A data file corresponding to the Atlantis model described in the previous section *The Atlantis Case Study* can also be downloaded from the above-mentioned sources.

The latest version of the GNU-Linear Programming Kit (GLPK) which contains the solver GLPSOL can be downloaded here. The Setup file on extraction will create a folder containing the necessary library files and GLPSOL. The folder can be extracted to any location (referred by ”. . . ” in the sections below).

The main steps to perform a model run are outlined in the following:

1. ***The OSeMOSYS model file and the Atlantis data file*** Cut and paste the OSeMOSYS_201x_xx_x.txt and Atlantis.txt file into the . . . /GnuWin32/bin directory. (Remember that you set which directory this is after executing the glpk-xx.xx-setup.exe file.)

2. ***Open the command prompt*** The command prompt can be opened by clicking in Windows on Start, Run, and then typing ‘cmd’. (There are other ways of doing this but the above mentioned procedure is one of the easiest.) The command prompt will have some text to indicate the current directory to which you can send ‘prompts’. If you are not in the same directory where GLPSOL is stored, you need to change the directory to the address that contains ‘glpsol.exe’ (. . . /GnuWin32/bin). If GLPK has been extracted to (the default) ‘program files’ folder, you will need to go to C:/Program Files/GnuWin32/bin. In case you are not sure about the current directory, return to the root directory by typing ”cd/”[1] . Then, navigate to the folder in which the files are stored by typing ”cd C:/. . . /GnuWin32/bin” then hit the return key (if you need help on command prompt, type ‘help’ and possible command options will be listed).

3. ***Running the Atlantis example in OSeMOSYS using “glpsol“*** Now that you are in the . . . /GnuWin32/bin directory in command prompt, and you have OSeMOSYS_201x_xx_x.txt and Atlantis.txt pasted in the same directory you can run the Atlantis example in OSeMOSYS. To do this type:

   ```
   glpsol -m OSeMOSYS_201x_xx_xx.txt -d Atlantis_201x_xx_.dat -o results.txt**
   ```

   All characters are important including space. This invokes a command (glpsol) to take the model file (OSeMOSYS_201x_xx_x.txt) and associated data input file (Atlantis.txt) to produce an output file with a full set of results (result.txt). The results file is particularly large, even for a simple problem. To easily comprehend the results, the OSeMOSYS code includes a few extra lines of code (not part of the optimization routine and available at the end of the code file) to produce a summary of results called SelectedResults.csv. This, as well as the full results file will appear in the . . . /GnuWin32/bin directory after the model run. A list of other possible command options can be found in the command prompt by typing glpsol –help.

4. ***Errors*** Glpsol will display an error message if it does not understand the input files, and prompt the line number where there is a conflict.

5. ***Output*** When glpsol is running successfully, it prints a status line. Each line will look similar to the following:

   *4: objval = 6.254971664e+03 infeas = 0.000000000e+000

   ‘*’ means that a basic feasible solution has been found, ‘4:’ means that there have been 4 iterations to find a solution so far, ‘objval’ shows the current objective value, and ‘infeas’ shows the amount of infeasibility. When a feasible solution has been found, its value will be either 0 or a very small number. For more information on this please read the documentation on GNU Linear Programming by Rodrigo Ceron Ferreira.

6. ***Solution*** To see the full solution, use a text editor to open results.txt. (For example, Notepad or Notepad++, see Section 4.2.5 on Supportive Programmer and Documentation). Recall that the solution file will be found in the directory . . . /GnuWin32/bin. The solution summary file is a comma separated file called SelectedResults.csv. A csv file can be conveniently opened in a spreadsheet and the ‘text to column’ tab can be used to delineate the data by using the ‘comma’ option as the delimiter. The selected results file produces tables of the following outputs. (The units indicated are specific to the Atlantis example. Other units may be defined by the user when setting up a new data file):

   - Total emissions, by type and region (emissions units, Mton)
   - Total costs, by region (currency units, m$)

[1] cd stands for *change directory*.

- The (time independent) demand for each: energy carrier (this is zero if no demand was entered), region and year (energy units, PJ)
- The (time dependent) demand for each: energy carrier (this is zero if no demand was entered), time slice, region and year (energy units, PJ)
- The (time dependent) production for each energy carrier timeslice, region and year (energy units, PJ)
- The total annual capacity of each technology by region (capacity units, GW)
- The new investment in capacity for each technology for each year by region (capacity units, GW)
- The annual production by each technology of each energy source by region (energy units,GW)
- The annual use by each technology of each energy source by region (energy units, PJ)
- Annual emissions, by species and region (emissions units, Mton)
- Annual emissions by technology, species and region (emissions units, Mton).

If you have problems in running the files from the …/GnuWin32/bin directory (due to lack of administrative rights), redirect the OSeMOSYS code and data files to a different directory out of the C drive.

Note that if the directory on the command prompt window is not same as …/GnuWin32/bin, then you need to write the complete path (e.g., C:/Users/user001/Documents/OSeMOSYS_201x_xx_x.txt) in the command Prompt or change to the new folder before running the model.

Creating an input file

To create the input data for an optimization run, you can set up the model directly in a text editor. It is advisable to start with a small model and build it up step wise. This will simplify the debugging process (see following chapter in this manual on *Debugging a model*). It is further advisable to back up working versions of model data files by saving them in a folder of your choice. The Atlantis input file provided with the downloaded model code might serve as a useful starting point to see how data needs to be correctly formatted. Alternatively, LEAP has also proven useful to write an OSeMOSYS data file.

Data and choices of units

The cornerstone of a legitimate model is input data. Using accurate data, relevant model designs and a consistent choice of assumptions will ultimately offer better and more representative insights into the system. Typical data requirements include:

- Energy demand for the activities that are considered in the model and an annual (hourly) load curve for the relevant demands;
- Technology specific efficiencies, Electricity generation capacity, technology specific factors (capacity/availability), construction time, lifetime;
- Technology costs (capital, fixed and variable O&M), Fuel costs (both local and imported costs);
- Resource potential (fossil fuel reserves, renewable energy potential), water availability for hydro power plants;
- Emissions accounting and corresponding fuel specific emission factors.

Useful technology briefs containing such data have been developed by ETSAP. The World Energy Outlook from IEA, IEA Cost of Generating Electricity (look for latest publication at www.iea.org) and IRENA's Renewable energy publications can further be used to obtain the required data for modelling a country's energy sector. The fossil fuel reserves in every country can be obtained from EIA (U.S. Energy Information Administration) (5). The World Bank database is another useful source of data for energy demand. Note that these publications provide generic values, and data from national studies and strategy documents are usually preferred.

For OSeMOSYS there are 4 units that needs to be chosen in a consistent manner. Bear in mind that certain default constraint levels, e.g., the total max capacity are set to values such as 999999. These limits can be violated if the choice of unit is too small (like kW for a large system, for which GW are recommendable). If a smaller unit is chosen, then the default values for certain parameters need to be increased in the model/data file.

Input variables	Possible choice of unit
Energy	GWh, MWh, PJ, GJ, etc.
Power	GW, MW, etc.
Cost	Million $, Million £, Million Euro, etc.
Emission	Mton

There is no unit conversion in OSeMOSYS: the modelling system assumes that all units are consistent. For example, the unit for capital costs needs to be coherent with the choice of units from the above table and is applied for all parameters relating to the costs. For example, when choosing GW and $ as power (capacity) and monetary units respectively, the capital cost has to be defined in Million $/GW. Similarly, if the energy unit is PJ, then the activity ratios and variable costs need to calculated with care to avoid errors. This is particularly important for the parameter **CapacityToActivityUnit** (1) (2), which depends on the final energy unit and the unit of capacity.

Supporting programmes and documentation

The installation of a text editor such as Notepad++ or Visual Studio Code is recommended to work with and edit the model and data files.

The following three files are recommended for more background documentation on the basics of GNU MathProg and the linear optimization logic applied in OSeMOSYS:

1. The GNU Linear Programming Kit, Part 1: Introduction to linear optimization;
2. The GNU Linear Programming Kit, Part 2: Intermediate problems in linear programming;
3. The GNU Linear Programming Kit, Part 3: Advanced problems and elegant solutions.

Further, as mentioned before, the most comprehensive description of how OSeMOSYS works is provided in :

1. "OSeMOSYS: The Open Source Energy Modeling System, An introduction to its ethos, structure and development" by Howells et al. in 2011
2. "Modelling elements of Smart Grids – Enhancing the OSeMOSYS (Open Source Energy Modelling System) code" by Welsch et al. in 2012.

It should be noted that the salvage value as described in Howells et al. in 2011 is not applicable anymore: please see the Change Log provided at www.osemosys.org for latest changes. Further, the description of storage in Howells et al. in 2011 is not applicable any longer. Instead, refer to the current way of modelling storage or variability as described in Welsch et al. in 2012.

The Python package *otoole* has been developed to provide a set of useful scripts and data management functions accessible from the command line.

1.4.3 How to run OSeMOSYS using a macOS

You have two options to run OSeMOSYS on OSX.

Use Homebrew to install GLPK:

(Thanks to @EmiFey for the instructions)

1. Open the terminal

2. Enter the following: `xcode-select --install`
3. After the command line developer tools are installed, go to: `https://brew.sh`
4. On the home page, copy the ”Install Homebrew” link to the Terminal prompt on your Mac.
5. Follow the instructions in the Terminal prompt, it will require your computer password.
6. After the installation of HomeBrew is completed, you should see this line in the Terminal prompt: `==> Installation successful!`. Check that HomeBrew is correctly installed by entering `brew help` in the TerminalPrompt. This should give you examples for usage or troubleshooting.
7. Enter the following command into the Terminal prompt: `brew install glpk`. Homebrew will now install the GLPK package on your computer. You can check if

Or, you can use miniconda to create an environment:

1. Follow these instructions to install miniconda
2. Create a new conda environment `conda create -n osemosys glpk`
3. Activate the new conda environment (you need to do this everytime you open a new terminal window or tab) `conda activate osemosys`
4. The command `glpsol` should now be available in the terminal

Downloading model files and running the model

Download the model and data *.txt* files provided on the OSeMOSYS website, under Get Started.

This tutorial uses the OSEMOSYS_2011_11_08.txt model and the UTOPIA_2011_11_08.txt data. Save these *.txt* files in your working folder.

Notice: When using Textedit.app to save your model and data, your file might be saved with the *.rtf* extension. To change it to *.txt*, locate the file, click on the name (so it turns blue, do not double click it, just select) and change the extension to *.txt* manually.

To run the model go back to the Terminal, locate the OSeMOSYS files in the working folder and then run the model by typing the following line:

```
glpsol -m OSEMOSYS_2011_11_08.txt -d UTOPIA_2011_11_08.txt -o Results.txt
```

The model will run in the Terminal and when finished you should see the following message:

```
OPTIMAL LP SOLUTION FOUND
Time used:  2.8
```

Running the model will create a results file in the working folder.

1.5 Debugging a model: useful tips

1.5.1 Adding a dummy technology

There can be several reasons why a model has “no feasible solution”. One way of finding where the errors might be located is to add a dummy technology (which is not originally part of the model) that has high capacity and/or a very high variable cost and, to make things simple, no input fuel. This ensures that the model only uses the dummy technology when no other option remains. To debug a model, therefore define a single dummy technology and run the model to see if there is now a solution and start as close to the demand as possible, referred to as “1” in the below figure. If the model now runs successfully, check the results file and check when the dummy technology is used (e.g. in which time slices, in which years) and also check which other technologies are not used to the extent one would expect. This may give some clues where the error in the model is situated. After trying to find and fix the error, rerun the model and remove the dummy technology if it is not used any longer (alternatively, check every single time you run the model that the dummy technology is not used again, potentially for another reason if the model was modified in the meantime). In more complex models, the model might solve after adding a dummy technology, yet it may still be unclear where the mistake in the model file might be. In these cases, revert back to the original model and add a dummy technology in "2" in the RES (as seen in the Figure below) (and subsequently, if necessary, in "3"). This method will help to identify more clearly in which part of the Reference Energy System to mistake might be.

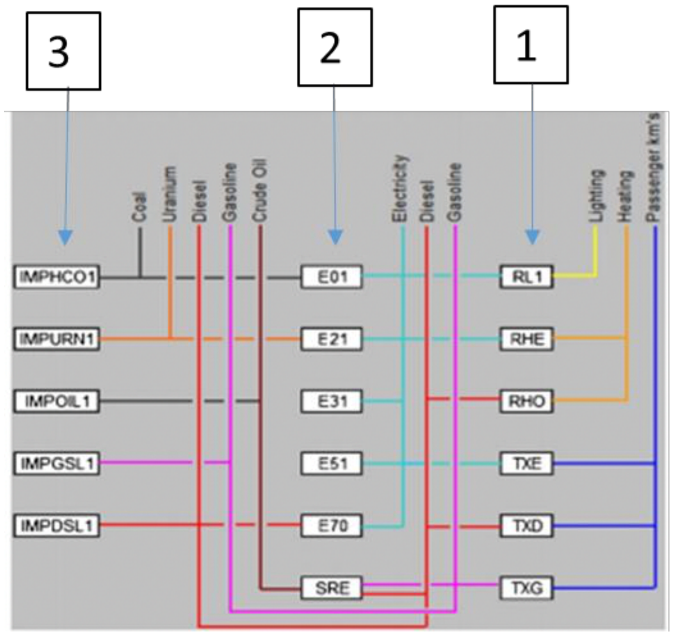

Fig. 2: Strategy to add a dummy technology in the RES.

1.5.2 Upperlimit and maxlimit increase

If the optimization has no feasible solution, then one issue might be that certain upper limits or maximum limits have become binding and prevent the solver from finding a feasible solution. There are upper limits defined by the user but also default upper limits on parameters that are embedded in the code for OSeMOSYS (either in terms of Capacity or Activity). To find these "upperlimit" and "maxlimit" search in the code (Ctrl+F) and take a step-by-step approach both in the input file and in the OSeMOSYS code to identify if the issue is in one of these constraints. One option is for example to write a hash "#" before one or several of these constraints, thus commenting them out. If the model then solves successfully, it is clear that these constraints (potentially in combination with others) where the reason for the previous infeasibility and adjustments in the data file are necessary. These could involve correcting incorrectly entered input data or raising the maximum limits defined in the data file. Of course, the commented out constraints would have to be uncommented in the corrected model by removing the hash "#" sign.

1.5.3 Starting the chain with a technology

In OSeMOSYS the chain always needs to start with a technology. This means that in order to have a fuel in the system, a technology needs to be defined as the producer. For instance, in an oil power plant we define oil as an input (parameter "InputActivityRatio") and electricity as an output (parameter "OutputActivityRatio"). However, the oil has to be produced by another technology, such as "oil import". In this case the oil import technology has NO input (unless we are defining the entire oil supply chain as well), but only oil as an output. Another example where an error is common is when defining a renewable energy technology, such as solar photovoltaics (PV). For example, in the case of a PV, the output is electricity. You can choose to define sunlight or have no input at all. In the former case though, a technology providing this sunlight needs to be defined also (e.g. “Sun”).

It is advisable to construct a Reference Energy System (similar to the Figure above, under *Adding a dummy technology*), before starting to construct the model. This will help visualize the overall system with its interconnections and it will facilitate in the quick identification of potential sources of error.

1.5.4 Incorrect demand split definitions

A common mistake is misrepresenting the seasonal/daily variation of demand. To avoid this, make sure that the sum of all timeslices in the parameter “SpecifiedDemandProfile” adds up to 1. An indication that this may be a problem is if the total production of the respective fuel is lower (in case sum of “SpecifiedDemandProfile” specified is lower than 1) or higher (in case the sum of “SpecifiedDemandProfile” specified exceeds 1) than what defined in the parameter “SpecifiedAnnualDemand”.

1.5.5 Setting the correct default values

Each parameter is typically defined with a default value. This in turn is assumed to be the value for all the parameters of technologies, fuels etc. that the modeller chooses not to define explicitly. Make sure to go through all default values and verify that these are not the source of a problem. For instance, one common mistake is setting a default value for the parameter “DiscountRate” at 5 (indicating 500%) instead of 0.05 (indicating 5%). As a result, since OSeMOSYS optimizes the discounted cost of the system, the difference between the various alternative pathways is assumed to be minimal due to the incredibly high discount rate and the resulting optimal solution is simply a random choice from the vast solution space. As a measure of good practice, it is advised that appropriate default values are chosen for all parameters at the very beginning of model development.

1.5.6 Capacity factors

The parameter Capacity Factor in OSeMOSYS represents an upper limit on the capacity of a specific technology that can operate in each time slice. For fossil fuel fired power plants, e.g. steam cycles fuelled by coal, it may be around 85% of the installed capacity, meaning that in a given timeslice at most 85% of the capacity can be used, due to technological constraints. For renewables like PV panels or wind turbines it can be used in a slightly different way (and it usually is): it still represents an upper limit on the ratio of the capacity that can be used, but due to the availability of resources, rather than to technological constraints. For instance, for PV panels at night it is set to 0, so that the user makes sure the model won’t choose to use PV panels when the sun is not there.

1.6 Advanced Functionalities

1.6.1 Using OSeMOSYS with the solver CPLEX

CPLEX is a commercial solver, more performing than the freely available GLPK solver for large problems. It is freely available for use by universities and in non-commercial projects. To run OSeMOSYS using CPLEX, you need to have CPLEX and Python installed on your PC.

1. In order to use CPLEX, the OSeMOSYS model and data files first need to be combined into a single ‘.lp’ file. To do this, open the command prompt, select the GLPK folder containing the OSeMOSYS model and data files (see . . . for basic command prompt functions and how to select folders) and use the following command: **glpsol -m [OSeMOSYS model file] -d [Data file] - -wlp [Input_Filename.lp]**

2. After the .lp file is generated, close the command prompt window. Open CPLEX. Type *'read C:Input_FilepathInput_Filename.lp'* and press Enter. The command will require from few second, up to minutes to be executed, depending on the size of the file to be read from the chosen filepath.
3. After the file is read, type *'optimize'* and press enter. CPLEX will run the problem and find an optimal solution (or give a message of non-feasibility if no solution can be found).
4. After the optimization is over, type *'write C:Output_FilepathOutput_Filename.sol'* and press Enter. The command will require from few second, up to minutes to be executed, depending on the size of the file to be written to the chosen filepath. When the solution file is written, close the CPLEX window.
5. The solution file needs now to be sorted and reordered. For this, download from the OSeMOSYS website the Python sorting script (bottom page) that was developed for this function and copy it in to the Python installation folder. Usually, when Python is installed, the folder *'C:PythonXX'* is chosen by default. For this step and the next two, the instructions will assume this as the folder. Please check what the path of the installation folder is in your case and use that in steps 5, 6 and 7 instead, if different. Copy in the same folder also the *Output_Filename.sol* generated in the steps before.
6. Open the command prompt again. Select the directory *'C:PythonXX'* and type: *python transform_31072013.py Output_Filename.sol Output_Filename.txt*. The execution of the command may take from few seconds to minutes, depending on the size of the file.
7. After the command is executed, type: *'sort/+1<C:PythonXXOutput_FilepathOutput_Filename.txt>C:PythonXXOutput_Filename rted.txt'*.

The file produced through the above process is available in the *'C:PythonXX'* installation folder. The user may cut and paste it where she/he finds it convenient. It contains the results of the model run in a format that is easy to analyze, either directly or after copying into another platform such as MS Excel.

1.6.2 Modifications, enhancements and extensions of the main code

A number of extensions of the published version of the OSeMOSYS code were developed over time by different users from the community. They are briefly described in the following sub-sections. Information is provided on where to find the code, test case-studies (if available) and other relevant documentation.

Reduction of the translation time

One of the highest barriers to the creation of large continental models with detail to the country level, or highly detailed national applications, is the time and memory requirement to translate a model into a solvable matrix. In order to increase the applicability of the tool, a revision of the OSeMOSYS code was completed in late 2016, as the result of a public call launched and led by UNite Ideas.

Much of the flexibility of OSeMOSYS arises from the broad definition of all energy components as technologies and all energy carriers as fuels. The connections between them are specified in the data file by the user in the form of region- and time-specific tensors named Input- and OutputActivityRatio, that define the rate at which each fuel goes in and comes out of an active technology. Since in the majority of cases each technology converts one fuel into another – e.g. a gas turbine turning gas into electricity – the expected density of these tensors is very low, on the order of 1 over |regions| × |technologies| × |modes| × |fuels|. While previous versions of OSeMOSYS already filtered out technology-fuel relations with a zero rate and presented a straightforward linear problem to the solver, the amount of time used by having to apply this filter was repeatedly underestimated.

The updated version speeds this process up by generating intermediate parametric sets from a single scan over the sparse connection tensors after reading in the data file. These sets hold all the combinations of technology and modes of operation which consume or produce a specific fuel, and the rest of the model definition refers to them exclusively when considering technologies and fuels without any loss of generality. The updated version of OSeMOSYS including these changes is available on GitHub[1] .

All measures combined have been shown to reduce the time for translating the MathProg language of a reference country case-study from 526 seconds to 38 seconds on an Intel Core i5-2520M processor, a factor of 13; other cases show a similar speed-up. The subsequent time for solving the model is not affected. Further, when the code adjustments were tested on one of the largest OSeMOSYS models, the TEMBA model, considering the electricity supply system of 47 African countries, the sum of the translation and the calculation time was reduced from 18 hours to under 2 hours.

Short-term operational constraints of power plants

The global push towards increasing the share of Renewable Energy Sources (RES) in the energy supply poses challenges in terms of security and adequacy of electricity networks. Intermittent generation from e.g. wind and solar power may require back up. Two of the widely discussed options include controllable fossil fuel-fired generation (such as Open Cycle or Combined Cycle Gas Turbines) and storage. In order to assess the costs and benefits of using the first as a back-up option for peaking generation, new blocks of functionality computing short-term costs and operational constraints of dispatchable generation were designed. These include:

- *Computing the reserve capacity dispatch to meet an exogenously given demand, under constraints on ramp rates and minimum duty [#welsch1]_* : the enhanced version of OSeMOSYS was compared to a modelling framework coupling TIMES and PLEXOS, through a case-study analysing optimal energy infrastructure investments in Ireland in 2020[3] . While avoiding the high computational burden of the TIMES-PLEXOS model (the time resolution of the latter is 700 times higher), the OSeMOSYS model provides similar results. For instance, the investments diverge by 5%. The new block of functionality was further modified to make the reserve capacity demand an endogenous variable, namely a function of the penetration of intermittent renewables[4] .
- *Costs related to the flexible operation of power plants, specifically*: increased specific fuel consumption at lower load, wear and tear costs associated to the number of ramp-up and ramp-down cycles and costs for refurbishing existing units[5] .

Demand-flexibility

The expansions of the OSeMOSYS code introduced in Section 6.2.2. allow for the modelling of flexible supply options to back up the increasing intermittent renewable generation in energy systems. However, the transition to low-carbon and highly renewable energy systems can be facilitated also by demand-side options. Welsch et al. proposed an expansion to the code of OSeMOSYS, to allow for the modelling of elements of Smart Grids. The description of the enhancements is provided in[6] and the code formulation in the attachments to the same publication. The enhancements are compatible with the version of the OSeMOSYS code dated 8 November 2011 and available at www.osemosys.org. They include:

- *Variability in generation*. The CapacityFactor parameter is made timeslice-specific. This modification is embedded also in the currently published version of OSeMOSYS.
- *Improved storage representation*. Block of equations adding detail to the previous formulation for the storage. This modification is embedded also in the currently published of OSeMOSYS.
- *Prioritizing demand types*. Block of equations allowing the cost-optimal amount of load shedding and its overall cost to be computed, for certain flexible demand types defined by the user.

[1] Optimus.community, OSeMOSYS GitHub, (2017). https://github.com/KTH-dESA/OSeMOSYS (accessed October 3, 2017).

[3] Welsch, M., Deane, P., Howells, M., O Gallachoir, B., Rogan, F., Bazilian, M., Rogner, H., 2014. Incorporating flexibility requirements into long-term energy system models–A case study on high levels of renewable electricity penetration in Ireland. *Applied Energy*, 135, pp. 600–615. doi:10.1016/j.apenergy.2014.08.072.

[4] Maggi, C., 2016. Accounting for the long term impact of high renewable shares through energy system models: a novel formulation and case study. *Politecnico di Milano* [Online]. Available at: https://www.politesi.polimi.it/handle/10589/125684.

[5] Gardumi, F., 2016. A multi-dimensional approach to the modelling of power plant flexibility. *Politecnico di Milano*.

[6] Welsch, M., Howells, M., Bazilian, M., DeCarolis, J., Hermann, S., Rogner, H., 2012. Modelling elements of smart grids–enhancing the OSeMOSYS (open source energy modelling system) code. *Energy*, 46, pp. 337–350. doi:10.1016/j.energy.2012.08.017.

- *Demand shifting*. Block of equations allowing part of the demand to be shifted earlier or later in a day. The demand shift has a cost defined by the user and it can be constrained to occur within a certain time frame and up to a certain quantity.

Short-term planning

This version of OSeMOSYS was developed to further evaluate the short-term performance characteristics of systems with a high penetration of variable RES. It stems from the original code, enhanced by both the short-term operational constraints and the storage block of functionality described above. A number of additional modifications were introduced in order to improve the applicability of OSeMOSYS to finer time resolutions. Their focus was to preserve the temporal sequence of renewable energy availability and to evaluate the reaction of storage and other system management techniques to these dynamics. Specific changes include:

- *Revised storage equations that are more computationally efficient for short-term modelling*. Specifically, the intra-time slice storage equations in the base OSeMOSYS code were replaced with inter-time slice equations. This allows for much faster computation of the storage levels and allows for a larger number of scenarios to be computed in a shorter amount of time.
- *Equations that model the ramping constraints of conventional generators*. With large penetrations of variable renewables, the ramping demand in the system is significantly increased. The ability to constrain the ramping capabilities of generators in the system allows for a more accurate representation of the system dynamics and associated costs.
- *Equations that incorporate the cost of curtailment into the model*. This is not usually accounted for in a long-term model due to the averaging imposed by the time slice definitions.

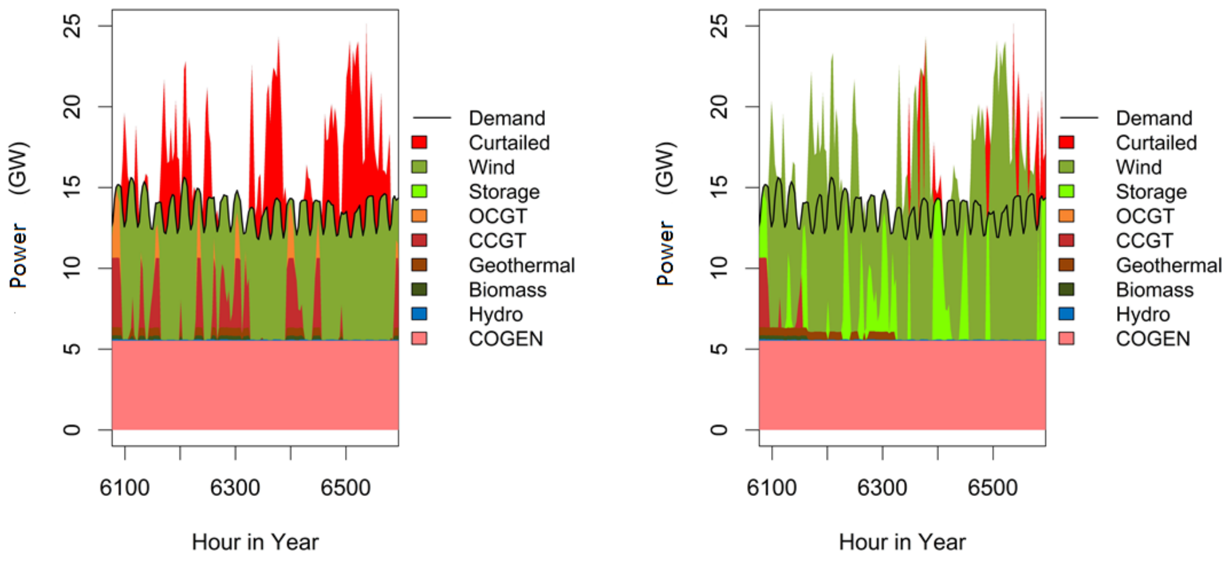

Fig. 3: Comparison between the power generation profile without and with storage.

The Figure above shows results obtained when using OSeMOSYS for short-term planning. The curtailed energy is marked in red above the demand line. Energy stored for future use is shown in light green.

Stochastic modelling of energy security assessment

This extension was developed by Linas Martišauskas[7] . It aims to assess the amount of unsupplied energy and its costs, in case disturbances to the energy system occur. The extension consists of two parts:

- *Stochastic model of disturbances*. A probability distribution is created for several potential disturbances and scenarios are generated by randomly picking realisations of the disturbances. This module is external to OSeMOSYS and written in Matlab.
- *Energy system optimization in the presence of disturbances*. Block of equations to compute the system-wide unsupplied energy and its cost, for each of the disturbance scenarios generated through the stochastic model. This part constitutes an extension of OSeMOSYS and it can be directly embedded in the code.

The two modules are described in detail in the related Doctoral dissertation[7] .

Cascaded water storage

This addition to the basic OSeMOSYS storage equations allows for cascaded facilities to be included with an upper reservoir and generation station feeding water into a lower reservoir for a second generation station. Further, it is designed to track the storage levels and water flows, in and between each reservoir. Constraints representing minimum and maximum output flows from each dam are included to model both flood control and fish habitat management. Specific focus was put on maintaining a similar storage structure as the base OSeMOSYS code while providing a more user friendly formulation for modelling hydroelectric generation.

This application has proven important in the study of integrated approaches for the management of water and energy resources[8] .

The cascaded hydro storage equations have been uploaded to GitHub where anyone interested can download, use and modify them for their own purposes[9] .

1.7 How to contribute to the development of the tool

All the community is encouraged to collaborate to the maintenance and development of OSeMOSYS by proposing code modifications, enhancements and extensions. The proposers may want to have the modifications reviewed by experts and included in a bi-annual peer-reviewed publication on the status of development of OSeMOSYS. In such case, they shall submit a related OSeMOSYS Enhancement Proposal (OEP) on the OSeMOSYS GitHub Repository following the instructions below:

- Sign in or sign up to GitHub and create a new OEP using the GitHub Issue tracker in the OSeMOSYS GitHub repository.
- Fill in the OEP following the the template provided here
  - Give a short (250 words) description of what you are proposing for the OSeMOSYS formulation;
  - Describe the rational for the proposed change in the OSeMOSYS code, keeping in mind that changes to the main OSeMOSYS formulation should provide value to the wider OSeMOSYS community;
  - Provide a detailed description of proposed changes, to inform on how the OSeMOSYS formulation will be affected; this should include details on:
    1) addition, removal or changes of sets, parameters;

[7] Martišauskas, L., 2014. Investigations of Energy Systems Disturbances Impact on Energy Security. *Kaunas University of Technology and Lithuanian Energy Institute* [Online]. Available at: http://www.lei.lt/_img/_up/File/atvir/2014/disertacijos/Santrauka_Martisauskas.pdf (accessed February 8, 2018).

[8] de Strasser, L., Lipponen, A., Howells, M., Stec, S., Bréthaut, C., 2016. A Methodology to Assess theWater Energy Food Ecosystems Nexus in Transboundary River Basins. *Water*, 8, 59. doi:10.3390/w8020059.

[9] Niet, T., 2017. GitHub - tniet/OSeMOSYS. Available at: https://github.com/tniet/OSeMOSYS (accessed October 7, 2017).

2) addition, removal or changes of equations;
3) addition, remove or changes of constraints.

OEPs can then be discussed with the administrators of the OSeMOSYS organisation, and representatives of the OSeMOSYS Steering Committee in order to be either approved, rejected, or changes requested, following a peer-review process.

If an OEP is accepted, then the developers can submit a pull request in the repository of their chosen implementation linking back to the original OEP. The pull request will then be reviewed by the admins. The submission will also require additions to the main documentation.

For further information, please refer to the contacts provided on the OSeMOSYS website.

1.8 References

Howells, M., Rogner, H., Strachan, N., Heaps, C., Huntington, H., Kypreos, S., Hughes, A., Silveira, S., DeCarolis, J., Bazilian, M., Roehrl, A., 2011. OSeMOSYS: The Open Source Energy Modeling System: An introduction to its ethos, structure and development. *Energy Policy*, 39 (10), pp. 5850-5870.

A complete list of the main publications using and applying OSeMOSYS as main tool for their analysis is available on the OSeMOSYS website at the following PUBLICATIONS webpage.

1.9 Contacts

1.9.1 Contacts

For any questions, please go the dedicated Q&A Forum.

You will need just a few clicks to sign up to the forum.

Contact us:

E-mail: osemosys@gmail.com (tbu) or meet the team here.

1.9.2 Contribute to the OSeMOSYS Community

If you are willing to contribute to the enhancement of the OSeMOSYS tool, or you are interested in connecting to the Community of experts actively working with OSeMOSYS, have a look at the OSeMOSYS GitHub repository.

1.9.3 Feedback

Please send all comments and feedback you may have regarding this documentation to osemosys@gmail.com.

1.9.4 Funding

OSeMOSYS is a non-profit project. We rely on external funding and welcome partnerships. In this regard please contact us at osemosys@gmail.com.

1.9.5 Administration

KTH Royal Institute of Technology, division of Energy Systems Analysis (dESA)

Postal Address:

KTH - School of Industrial Engineering and Management division of Energy Systems Analysis

Brinellvägen 68, SE-100 44 STOCKHOLM (Sweden)

https://www.desa.kth.se

1.10 FAQ

CHAPTER 2

Indices and tables

- genindex
- modindex
- search