MATPOWER Optimal Scheduling Tool
MOST 1.0.1
User's Manual

Ray D. Zimmerman   Carlos E. Murillo-Sánchez

October 30, 2018

Contents

List of Figures

List of Tables

1 Introduction

Beginning with version 6, MATPOWER [1] includes a framework for solving generalized steady-state electric power scheduling problems. This framework is known as MOST, for **MATPOWER Optimal Scheduling Tool** [2, 3].

MOST can be used to solve problems as simple as a deterministic, single period economic dispatch problem with no transmission constraints or as complex as a stochastic, security-constrained, combined unit-commitment and multiperiod optimal power flow problem with locational contingency and load-following reserves, ramping costs and constraints, deferrable demands, lossy storage resources and uncertain renewable generation.

While the problem formulation is general and incorporates a full nonlinear AC network model, the current implementation is limited to DC power flow modeling of the network. Some work has been done on an AC implementation, but it is not yet ready for release.

The primary developers of MOST are Carlos E. Murillo-Sánchez[1] and Ray D. Zimmerman[2] of PSERC[3], with significant contributions from Daniel Muñoz-Álvarez and Alberto J. Lamadrid. It is built on top of MATPOWER[4], a package of MATLAB® M-files for solving power flow and optimal power flow problems [1, 4]. This manual assumes that the user is familiar with using MATPOWER, especially for solving optimal power flow problems, and makes numerous references to version 7.0b1 of the MATPOWER User's Manual [5].

1.1 License and Terms of Use

The code in MOST is distributed along with MATPOWER under the 3-clause BSD license [6]. The full text of the license can be found in the LICENSE file at the top level of the distribution or at http://www.pserc.cornell.edu/matpower/LICENSE.txt and reads as follows.

---

[1]Universidad Nacional de Colombia, Manizales, Colombia

[2]Cornell University, Ithaca, NY, USA

[3]http://pserc.org/

[4]See http://www.pserc.cornell.edu/matpower/ for more information on MATPOWER.

1.2 Citing MOST

While not required by the terms of the license, we do request that publications derived from the use of MOST explicitly acknowledge that fact by citing both the main MATPOWER reference [1]:

R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," *Power Systems, IEEE Transactions on*, vol. 26, no. 1, pp. 12–19, Feb. 2011. DOI: 10.1109/TPWRS.2010.2051168

8

and reference [2]:

C. E. Murillo-Sánchez, R. D. Zimmerman, C. L. Anderson, and R. J. Thomas, "Secure Planning and Operations of Systems with Stochastic Sources, Energy Storage and Active Demand," *Smart Grid, IEEE Transactions on*, vol. 4, no. 4, pp. 2220–2229, Dec. 2013.
DOI: 10.1109/TSG.2013.2281001

1.3   MOST Development

Following the release of MOST 1.0 (with Matpower 6.0), the MOST project moved to an open development paradigm, hosted on the MOST GitHub project page:

https://github.com/MATPOWER/most

The MOST GitHub project hosts the public Git code repository as well as a public issue tracker for handling bug reports, patches, and other issues and contributions. There are separate GitHub hosted repositories and issue trackers for Matpower, MOST, MIPS and the testing framework used by all of them, MP-Test, all available from https://github.com/MATPOWER/.

2 Getting Started

The first step in beginning to use the MOST is to get familiar with MATPOWER. This step is essential and this manual will assume familiarity with MATPOWER.

2.1 System Requirements

To use MOST 1.0.1 you will need:

- MATPOWER version 7.x or later[5]

- *(highly recommended)* a high-performance LP/MILP, QP/MIQP solver such as Gurobi, CPLEX, MOSEK, MATLAB's Optimization Toolbox or GLPK *(included with Octave)*.

MOST requires a working installation of MATPOWER. See the corresponding section in the MATPOWER User's Manual for more information on the system requirements for MATPOWER. MATPOWER 6 and later includes a full version of MOST in the *<MATPOWER>*/most directory.

It is also highly recommended that you install a high-performance solver such as Gurobi, CPLEX, MOSEK, MATLAB's Optimization Toolbox or GLPK, described in Appendix G in the MATPOWER User's Manual. For problems involving unit-commitment, a mixed integer solver is required.[6] But even for continuous LP and QP problems, these solvers will offer much better performance than MATPOWER's built-in solver based on qps_mips. The problems MOST generates can quickly become *very* large, so using the best solver you have available is a priority.[7]

2.2 Installation

The preferred method of installation is simply to install MATPOWER, which is a prerequisite for MOST and also includes its own copy of MOST.

If you have followed the directions for installing MATPOWER found in Section 2.3 of the MATPOWER User's Manual, then MOST should already be installed in the *<MATPOWER>*/most directory and the appropriate paths[8] added to your MATLAB path.

---

[5]MOST 1.0 required MATPOWER 6.

[6]At the time of this writing, MATLAB's Optimization Toolbox and GLPK only address MILP problems. Gurobi, CPLEX and MOSEK all handle MIQP problems as well.

[7]Gurobi and CPLEX are currently our preferred solvers for most MOST problems.

[8]To use MOST your MATLAB path must include *<MATPOWER>*/most/lib and, to run the tests, *<MATPOWER>*/most/lib/t.

To run the test suite and verify that MOST is properly installed and functioning, at the MATLAB prompt, type `test_most` . The result should resemble the following, possibly including extra tests, depending on the availablility of optional packages.

```
>> test_most
t_most_3b_1_1_0........ok
t_most_3b_3_1_0........ok
t_most_3b_1_1_2........ok
t_most_3b_3_1_2........ok
t_most_30b_1_1_0.......ok
t_most_30b_3_1_0.......ok
t_most_30b_1_1_17......ok
t_most_30b_3_1_17......ok
t_most_fixed_res.......ok
t_most_30b_1_1_0_uc....ok
t_most_sp..............ok
t_most_spuc............ok (432 of 720 skipped)
t_most_uc..............ok (156 of 260 skipped)
t_most_suc.............ok (111 of 185 skipped)
t_most_w_ds............ok
All tests successful (995 passed, 699 skipped of 1694)
Elapsed time 84.68 seconds.
```

If, for some reason, you prefer to install your own copy of MOST directly from the MOST GitHub repository[9], simply clone the repository to the location of your choice, where we use *<MOST>* to denote the path the resulting `most` directory. Then add the following directories to your MATLAB or Octave path:

- *<MOST>*/`lib` – core MOST functions
- *<MOST>*/`lib/t` – test scripts for MOST

It is important that they appear before MATPOWER in your path if you want to use this version of MOST, rather than the one included with MATPOWER.

2.3 Running a Simulation

Running a MOST simulation involves (1) preparing the input data defining all of the relevant power system parameters, transition probabilities, additional generator data and offers, storage data, contingencies, and profiles (2) invoking the function to run the simulation and (3) accessing the results in the output data structures. The

---

[9]https://github.com/MATPOWER/most

input data is provided in a MOST Data struct and the results are returned in an updated version of the input data structure.

Since MOST is built upon Matpower, it is assumed that the user is already familiar with running OPF simulations in Matpower (see Section 2.4 in the Matpower User's Manual).

2.3.1 Preparing Input Data

The MOST Data struct, containing all of the data needed for the problem, is sufficiently complex that it is not typically created directly, but rather is assembled from numerous other files or data structures by the loadmd function, as described in Section 5.1. They consist of (1) a Matpower case file or struct describing the system parameters for the base case, (2) transition probability matrices, (3) additional generator parameters, including commitment parameters, offer parameters related to reserves, inc/dec prices, and bounds on energy contract amounts, (4) parameters for storage units, (5) a contingency table defining a credible set of contingencies and their probabilities, and (6) profiles for time varying parameters such as load and renewable availability. In addition to the input data, aspects of the simulation are controlled by a set of Matpower options.

Typically, the Matpower case is defined in a case file and loaded into a struct using the loadcase function. The transition probability matrices can be specified as desired via a user-defined script or function as shown in the example below. The additional generator parameters are typically provided in a file defining an xGenDataTable and loaded by the loadxgendata function. Both the Matpower case and the xGenData can be modified by adding wind generators, or generators that represent energy storage units, using the functions addwind and addstorage, respectively. The latter also loads the additional storage parameters into a StorageData struct. The contingency table can be provided directly or returned by a user-defined function and is a changes table (chgtab) in the form expected by Matpower's apply_changes function.[10] And any time-varying parameters, such as load scaling factors and wind availability, are specified in profile data files and loaded with the getprofiles function.

---

[10]See Section 9.3.5 in the Matpower User's Manual.

```
mpc = loadcase('ex_case3b');
transmat = ex_transmat(12);
xgd = loadxgendata('ex_xgd_uc', mpc);
[iwind, mpc, xgd] = addwind('ex_wind_uc', mpc, xgd);
[iess, mpc, xgd, sd] = addstorage('ex_storage', mpc, xgd);
contab = ex_contab();
profiles = getprofiles('ex_load_profile');
profiles = getprofiles('ex_wind_profile', profiles, iwind);
mdi = loadmd(mpc, transmat, xgd, sd, contab, profiles);
mpopt = mpoption('verbose', 0);
```

2.3.2 Solving the Case

The solver in MOST is implemented in the `most` function. Assuming the input data have been loaded into the input MOST Data struct (`mdi`) and the MATPOWER options set in `mpopt`, the first stage solver can be called as follows.

```
mdo = most(mdi, mpopt);
```

```
================================================================================
            MATPOWER Optimal Scheduling Tool  --  MOST Version 1.0.1
            A multiperiod stochastic secure OPF with unit commitment
                        -----  Built on MATPOWER  -----
   by Carlos E. Murillo-Sanchez, Universidad Nacional de Colombia--Manizales
                    and Ray D. Zimmerman, Cornell University
          (c) 2012-2018 Power Systems Engineering Research Center (PSERC)
================================================================================
- Building indexing structures.
- Building expected storage-tracking mechanism.
- Building constraint submatrices.
  - Building DC flow constraints.
  - Splitting storage injections into charge/discharge.
  - Building CCV constraints for piecewise-linear costs.
  - Building contingency reserve constraints.
  - Building ramping transitions and reserve constraints.
  - Building storage constraints.
  - Building unit commitment constraints.
- Building cost structures.
- Assembling full set of costs.
- Assembling full set of constraints.
- Assembling full set of variable bounds.
- Calling MILP solver.


================================================================================


Gurobi Version 8.1.0 -- automatic MILP solver
--- Integer stage complete, starting price computation stage ---
Gurobi Version 8.1.0 -- automatic LP solver


================================================================================
- MOST: MILP solved successfully.
- Post-processing results.
- MOST: Done.
```

2.3.3 Accessing the Results

By default, the simulation does not output any results to the screen, instead storing the results in the output MOST Data struct (`mdo`). The details of the results in `mdo` can be found in Section 5.3.2.

For example, quantities like the commitment, expected dispatch and expected energy price, upward contingency and ramping reserve amounts for generator i in period t, and the dispatch of generator i in period t, scenario j and contingency k

can be extracted as follows.

```
define_constants;
unit_commitment       = mdo.UC.CommitSched(i, t);
expected_dispatch     = mdo.results.ExpectedDispatch(i, t);
expected_price        = mdo.results.GenPrices(i, t);
cont_reserve_up       = mdo.results.Rpp(i, t);
ramp_reserve_up       = mdo.results.Rrp(i, t);
Pg_tijk               = mdo.flow(t,j,k).mpc.gen(i, PG);
```

There is also a function called `most_summary`, described in Section 6.12, that can be used to print some summary results.

2.3.4 Setting Options

The standard MATPOWER options struct is used to set options such as the amount of progress output to display and algorithm to use to solve the underlying optimization problem. The options struct is set using the `mpoption` function.

```
mpopt = mpoption('verbose', 2, 'most.solver', 'GUROBI');
```

The full set of MOST options are detailed in Section 5.2.

2.4 Documentation

There are two primary sources of documentation for MOST. The first is this manual, which gives an overview of the capabilities and structure and describes the problem formulation. The MOST User's Manual can be found in your MATPOWER distribution at <*MATPOWER*>/most/docs/MOST-manual.pdf and the latest version is always available at: http://www.pserc.cornell.edu/matpower/MOST-manual.pdf.

The second is the built-in `help` command. As with MATLAB's built-in functions and toolbox routines, you can type `help` followed by the name of a command or M-file to get help on that particular function. Many of the MOST related M-files have such documentation and this should be considered the main reference for the calling options for each individual function. See Appendix A for a list of MOST functions.

There is one other important source of related documentation and that is the MATPOWER User's Manual. It is assumed that the user is very familiar with MATPOWER and its OPF capabilities.

3 Background and Overview

MOST grew out of research at Cornell University on the development and testing of new tools for the power industry, funded by the U.S. Department of Energy through the CERTS Reliability and Markets program.[11] Initially the work was focused on extending the AC optimal power flow problem to include co-optimization of locational reserves for security defined in terms of explicitly included contingencies [7,8]. This work was then extended to include multiple base scenarios to represent stochastic load and renewable generation availability, a multiperiod planning horizon, energy storage resources, ramping considerations and binary unit commitment decisions [2].

The formulation implemented in MOST and described in the next section further generalizes the problem by including the facility to define zonal reserve requirements like those from Section 7.5.1 in the MATPOWER User's Manual [5], and to constrain the solutions via a general linear time-varying dynamical system. The initial prototype was implemented by Carlos E. Murillo-Sánchez with subsequent development by Ray D. Zimmerman along with contributions by Daniel Muñoz-Álvarez and Alberto J. Lamadrid.

The description of the problem formulation will begin with a conceptual overview of the approach, beginning with the simplest single period problem that MOST handles and extending and expanding the problem one step at a time to illustrate how each additional aspect of the full problem is handled.

3.1 Continuous Single Period Problems

A single period problem can be as simple as a lossless economic dispatch (ED) problem, where the objective is to find the set of generator dispatch points that minimize the total cost of meeting a specified demand, with no modeling of any network flows at all. Even at this level, dispatchable demands can be introduced, modeled as negative generation as in Section 6.4.2 in the MATPOWER User's Manual, to model either involuntary load shedding or demand-side resources dispatched according to a benefit or bid function. It is assumed throughout that power injection variables may represent either generators or loads.

By introducing DC power flow equations as a function of bus voltage angle variables, along with limits on the branch flows, the problem becomes a DC OPF, taking

---

into account transmission system limitations. Using instead the AC power flow equations and flow constraints and introducing voltage magnitude and reactive injection variables results in an AC OPF problem, which models losses along with voltage and reactive power requirements.

Figure 3-1 illustrates the range of continuous single period problems considered by the formulation, with the varying level of detail of the network modeling represented by the vertical dimension. The horizontal dimension represents different ways to handle (or not) operational security requirements, with a third dimension adding a stochastic option for handling the uncertainty of demand and renewable generation. The green portion denotes the parts of the formulation that are implemented in the current version of MOST.
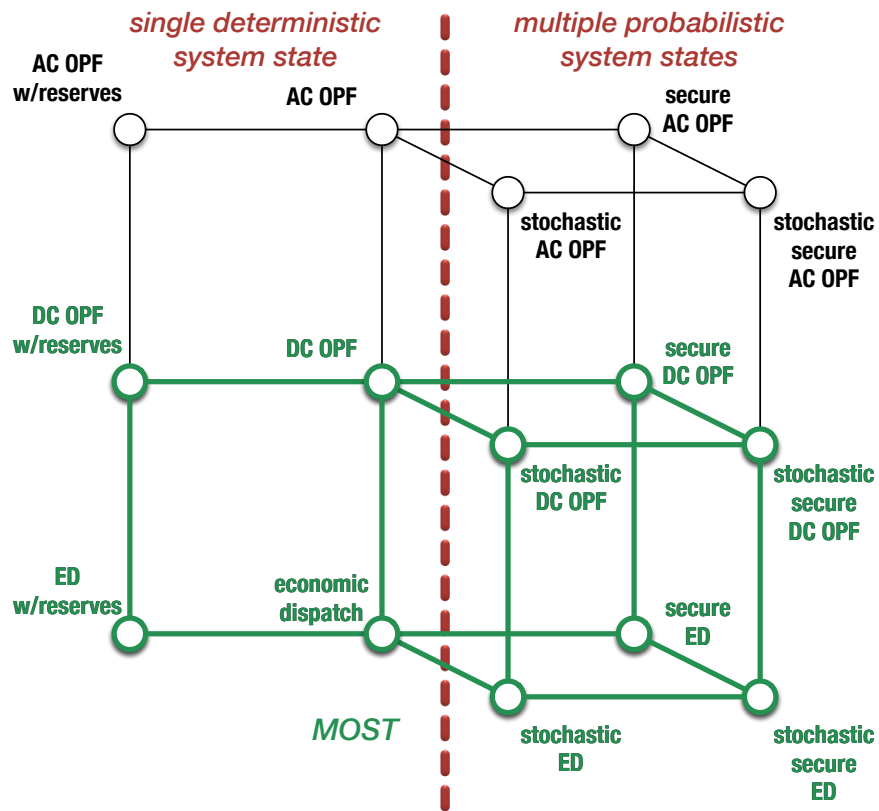
Figure 3-1: MOST Continuous Single-Period Problems

3.2 Security

Two options are included for addressing security in the single period problem, that is the need to find a dispatch that meets some criteria for withstanding disturbances or outages. The first is a deterministic approach that simply adds fixed zonal reserve requirements using the additional variables, constraints and costs described in Section 7.5.1 in the MATPOWER User's Manual.

The second is a stochastic approach, based on explicitly modeling the post-contingency state for each of a set of credible contingencies. In this approach, the base case ED or OPF problem is fully duplicated (all variables, costs and constraints) for each of the contingency states and modified to reflect the outaged equipment.

The base and contingency states are then combined into one large problem, where they are treated as separate islands in a single network, with the cost of each state weighted by its probability of occurence. The base and contingency states are further tied together by ramp limits on the generators, ensuring that the contingency state dispatches can be reached from the base case while respecting ramp rate constraints.

Finally, associated with each generator is a variable representing a reference dispatch value (e.g. optimal contract value) from which dispatch deviations (incremental and decremental redispatches) are defined. The maximum upward and downward deviations from this reference dispatch across all (base and contingency) states are defined as the upward and downward contingency reserves, respectively, and these reserves can have costs associated with them. In addition, the state specific deviations from the reference quantity can have their own probability weighted redispatch costs.

This second approach to security yields security constrained dispatch (ED or OPF) problem with endogenously determined generator-specific locational reserve requirements, derived optimally as a function of the set of included credible contingencies. In this problem formulation, it is assumed that the decisions regarding a reference dispatch and the corresponding dispatch ranges that define the reserves are made before the uncertain outcome of the occurence or non-occurence of a contingency is revealed. The state specific dispatch decisions are recourse decisions contingent on the outcome of that uncertainty.

The structure of this problem is illustrated by the diagram in Figure 3-2, where the circles represent the ED or OPF problems corresponding to the individual states, the dashed box denotes the reference dispatch, redispatches, reserves and associated costs and constraints, and the arrows illustrate the ramp limits constraining the deviations of contingency state dispatches from the base case dispatch.

Figure 3-3 shows the reserve structure for generator i. The green and pink bars show upward and downward redispatches from the reference dispatch p_c^i, where the
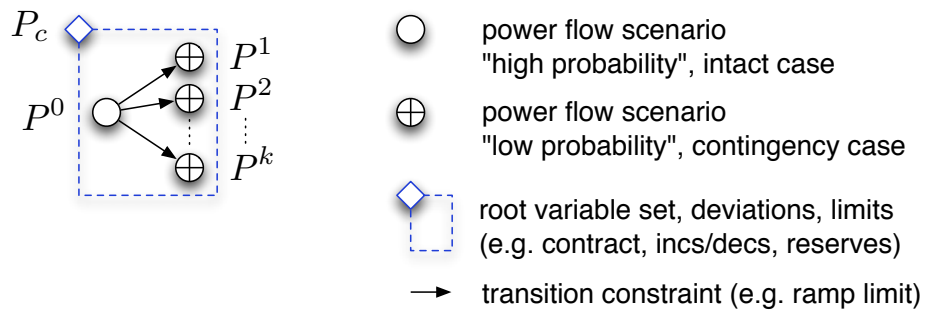
Figure 3-2: Secure Dispatch Problem Structure
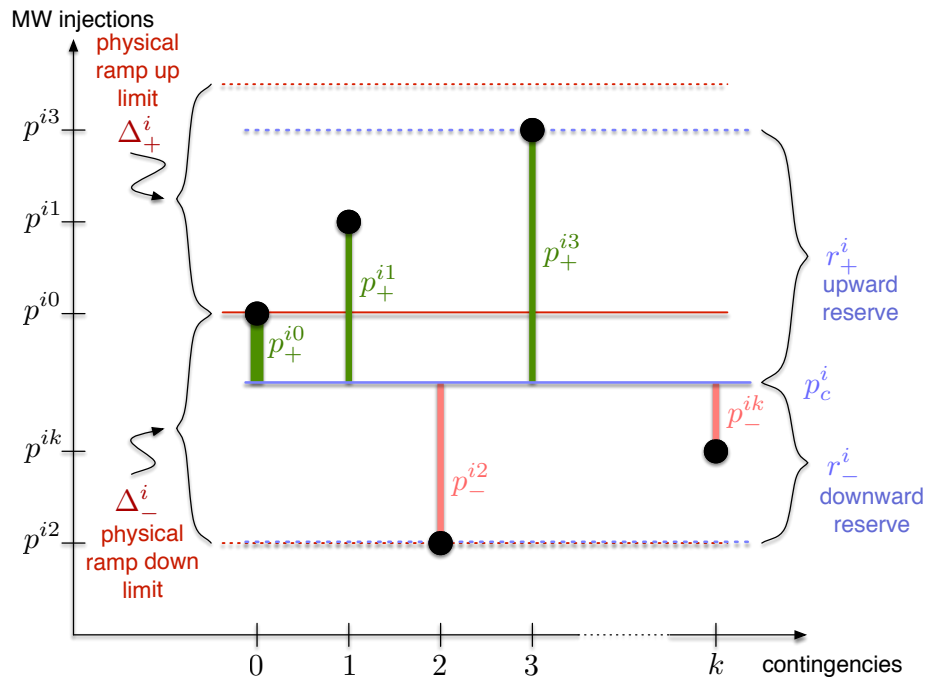
Figure 3-3: Reserve Structure for Generator i

maximums of these deviations define the corresponding reserve quantities. The physical ramp limit restrict deviations of contingency dispatches with respect to the base case dispatch p^{i0}.

3.3   Uncertainty of Demand and Renewable Generation

While contingencies refer to discrete low probability events, there is another kind of uncertainty introduced by errors in forecasting of demand and renewable sources of generation, such as wind and solar production. This type of uncertainty can be characterized by random system parameters with continuous probability distributions. MOST can model this type of uncertainty by drawing scenarios from the joint distribution of uncertain parameters and including these scenarios as multiple probability-weighted base cases in a structure similar to that described above for the continencies. There are two primary differences. First, the probabilities of the scenarios used to represent this type of uncertainty need not be small and, second, the base scenarios are tied to each other by the inc/dec and reserve variables, but not by physical ramp limits. This results in a stochastic dispatch problem with endogenously determined generator-specific locational reserves.

To make this problem secure, outage scenarios can be added for each base case, in the same manner as they were added to the single base case problem described previously. For a case with two contingency states, representing uncertain forecasts with two base scenarios results in the problem structure shown in Figure 3-4. In this case the reserves are defined by the maximum redispatch deviations across all scenarios and contingencies and the physical ramp rates limit the deviations of contingecy cases from the base case within each scenario, as illustrated in Figure 3-5. In this figure the variables include a t index as well, since the same structure is used at each period t in the multiperiod problems discussed below.
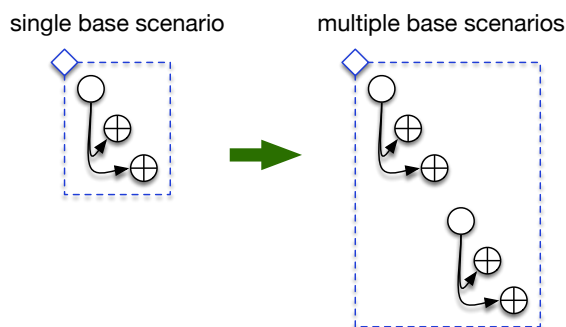
Figure 3-4: Problem Structure with Multiple Base Scenarios
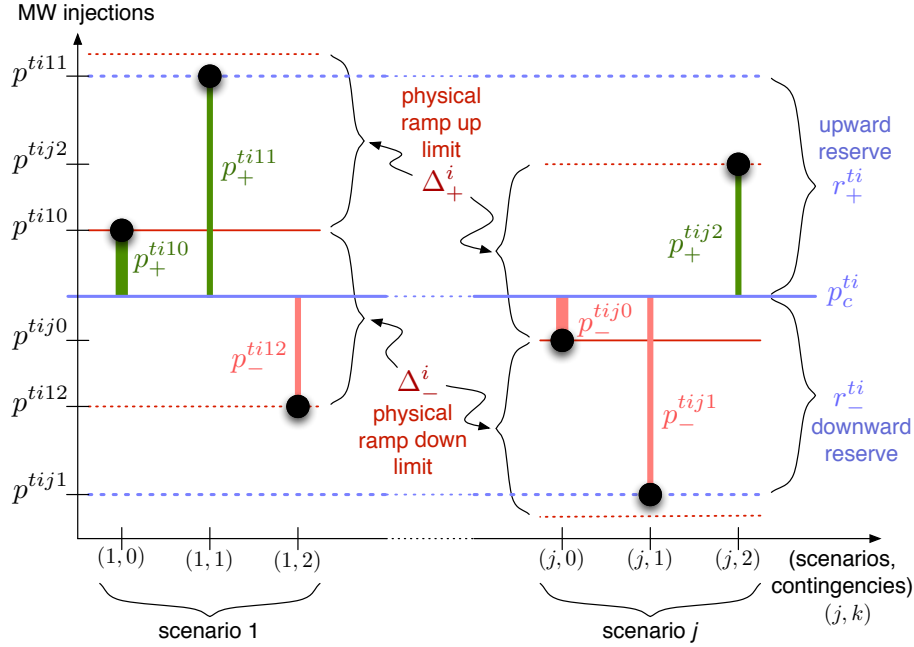
Figure 3-5: Reserve Structure for Generator i in Period t

Though there are now two types of uncertainty, contingencies and parameter uncertainty for demand and renewables, the reference dispatch and reserve decisions are made before the uncertainty is realized, with the state-specific dispatch decisions being recourse decisions contingent on the revealed outcome of both types of uncertainty.

3.4 Multiple Periods

For deterministic scheduling problems, the extension to multiple periods is straightforward. The single period problem is duplicated for each period in the planning horizon and combined into a single large problem which the individual periods appear as islands in a single network. Linking between adjacent periods is in the form of ramping costs and constraints involving the corresponding dispatch variables.

In the case of the stochastic problems with multiple probabilistic system states in each period, the extension to multiple periods is more complicated. One approach is to define each "scenario" as a particular realization of all uncertainties, defining a full trajectory through the entire planning horizon. The challenge here is in enforcing the non-anticipativity of the recourse decisions, since in reality the uncertainty is revealed period by period. On the other extreme, the option is to assume that from

21

each scenario in period t the system might transition to any of a number of states in period $t + 1$. Clearly, the explosion of the number of states with the length of the planning horizon is the barrier to adopting this approach.

MOST takes this multi-stage decision approach, but adds scenario recombination and scenario trimming to avoid the exploding number of scenarios. This is accomplished by assuming a Markovian structure for a high-probability central path where a transition probability matrix is used to describe the transitions from a limited set of base scenarios in one period to a limited set of base scenarios in the next period. States subsequent to contingency states are trimmed, on the assumption that they occur with low probability and likely require re-optimizing for the future following their occurence.

3.5   Ramping and Load Following Ramp Reserves

Ramping feasibility is only enforced on this high probability central path, in which all possible transitions are constrained to be feasible with respect to physical ramping capabilities as well as any load following reserve capacity offers. MOST includes the facility to define a simple quadratic "wear and tear" cost on the difference in dispatch from one period to another, applied in as a probability-weighted cost to each possible transition, as well as up and down load-following ramping reserve costs. These reserve costs apply to maximum upward and downward transitions included in the central path scenarios, as illustrated in Figure 3-6.

3.6   Storage and Deferrable Demand

Including the time dimension in the problem also opens the door for centrally dispatching resources such as energy storage and deferrable demand technologies which inherently couple operations across time periods. Battery terminology, such as charging and discharging, will be used to describe the storage model implemented by MOST, though the concepts and model are applicable to a range of storage and energy limited technologies, including batteries, pumped-storage units, combined space conditioning and thermal storage (e.g. ice batteries), and even dispatchable demands or generation with an energy quota over a given horizon.

A storage unit is implemented as a generator with an associated stored energy. The unit has upper and lower bounds on both power and energy, all of which can be either positive or negative. The power bounds define the maximum and minimum power injections, corresponding to the discharging and charging power limits for a traditional battery. The bounds on stored energy, which can vary over time, are
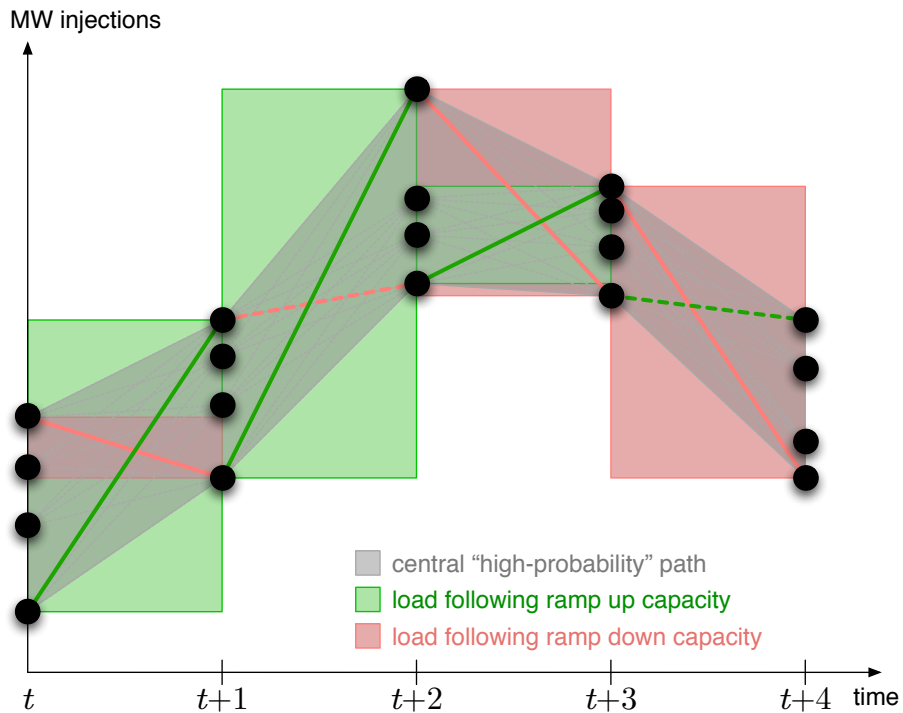
Figure 3-6: Ramping and Load Following Ramp Reserves

used to specify the energy capacity (e.g. for a battery) or quota (e.g. time-flexible demand). The storage unit can also have non-unity charging and discharging efficiencies as well as a loss coefficient defining energy losses in each period as a linear function of the amount of stored energy.

For deterministic problems, as with ramping, enforcing the intertemporal constraints imposed by such an energy storage resource is straightforward. In this case, it is a simple set of conservation of energy constraints.

For the stochastic problems, on the other hand, it becomes more complicated. Since the quantity of stored energy available in any given scenario at time t is dependent on the dispatches in preceding periods, there is no uniquely defined quantity of stored energy for each scenario, so it is not possible to track actual stored energy values, only expected values and minimum and maximum values.

For this reason MOST defines decision variables representing the upper and lower bounds on stored energy in each period and enforces feasibility with respect to these limits. Figure 3-7 shows how these limits spread out from period to period when the dispatch of the storage unit varies across scenarios. The only instance in which
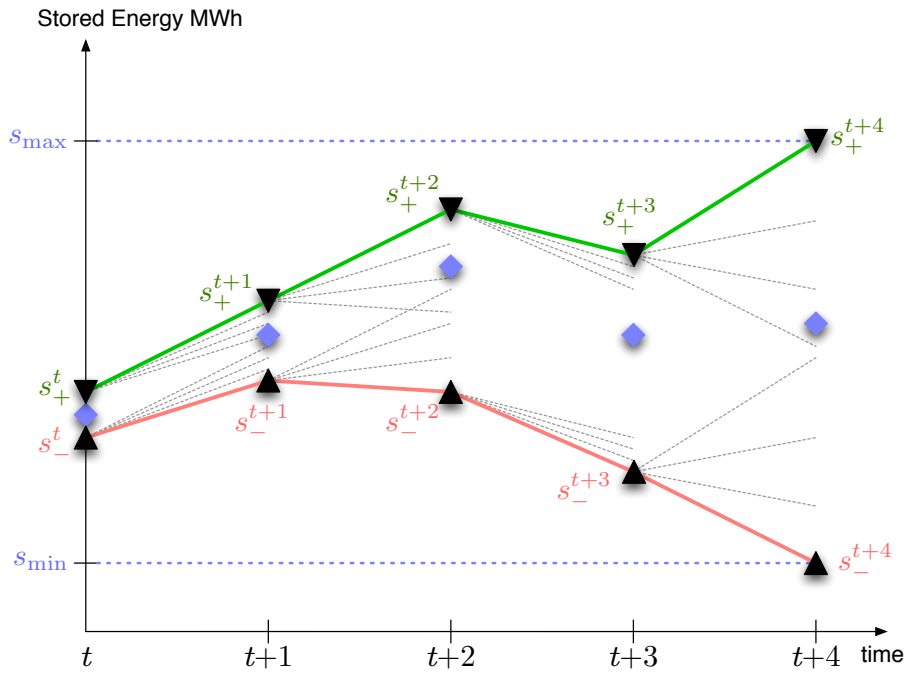
Figure 3-7: Storage

the limits do not spread out is if the unit adheres to the same dispatch schedule across all scenarios in a period. This formulation allows the optimization problem to make an optimal tradeoff between using the storage to arbitrage energy across time (same dispatch schedule across scenarios to avoid spreading of limits) and using it to mitigate uncertainty (vary the dispatch in response to the different scenarios, but at the expense of spreading limits).

The formulation also includes the option to relax the restrictions on the stored energy bounds in some or all periods to base them on the range of *expected* stored energy quantities in the previous period, rather than on the worst case ranges.

An important aspect of modeling storage is to include some way of either valuing or constraining both the initial amount of stored energy in each unit and the amount of leftover storage in terminal states, both end-of-horizon states and contingency states. MOST includes several options. First, for each unit there is a cost assigned to any initial stored energy at the beginning of the horizon. Similarly, there are a number of parameters used to specify the value of leftover stored energy in terminal states. Furthermore, the initial stored energy amount can be specified and the final expected stored energy amount can be constrained to equal a target quantity.

Finally, there is also the option to simply constrain the initial amount and the final expected amount to be equal while letting that level be determined as an output of the optimization.

3.7   Linear Time-Varying Dynamical System

The main decision variables in the problem are the dispatches, and they in turn can affect subsequent systems of different kinds. Specifically, the dispatch schedule can be restricted by putting constraints on the states of a linear time-varying system whose inputs are the expected dispatches at each time period.

One can think, for example, of a very simplified atmospheric diffusion model in which the amount of a greenhouse gas or pollutant released by a specific generation unit is proportional its dispatch. Or, perhaps the water level at different locations of a river downstream from a hydro unit can be modeled using a linear time-varying system driven by the unit's dispatch history. Flood levels or navigability might impose constraints on those levels. While this modeling capability is not, by any means, for the casual user of MOST, advanced users may find that it provides a flexible mechanism for customization that precludes the need for modifying the MOST code directly to achieve purpose-specific ends.

3.8   Unit Commitment

So far, the discussion has been restricted to problems with continuous optimization variables, with an overall structure for the general case as illustrated in Figure 3-8.[12] The class of problems addressed by MOST also includes those with discrete unit commitment decisions, with optional startup and shutdown costs associated with changes in online status from a prior commitment state. In multiperiod problems, these states are coupled through time, not only by the startup and shutdown costs, but also by minimum up and down time constraints.

For stochastic problems modeling multiple scenarios and/or contingency states there is a single commitment schedule shared by all states. That is, in the current formulation, a single binary variable is used to model the commitment for a given unit across all scenarios and contingencies in a given period.[13]

---

[12]The linear dynamical system constraints and the unit commitment aspects of the problem are not included in this illustration.

[13]To correctly model the commitment of fast start units with non-zero minimum generation constraints requires individual commitment variables for each scenario.
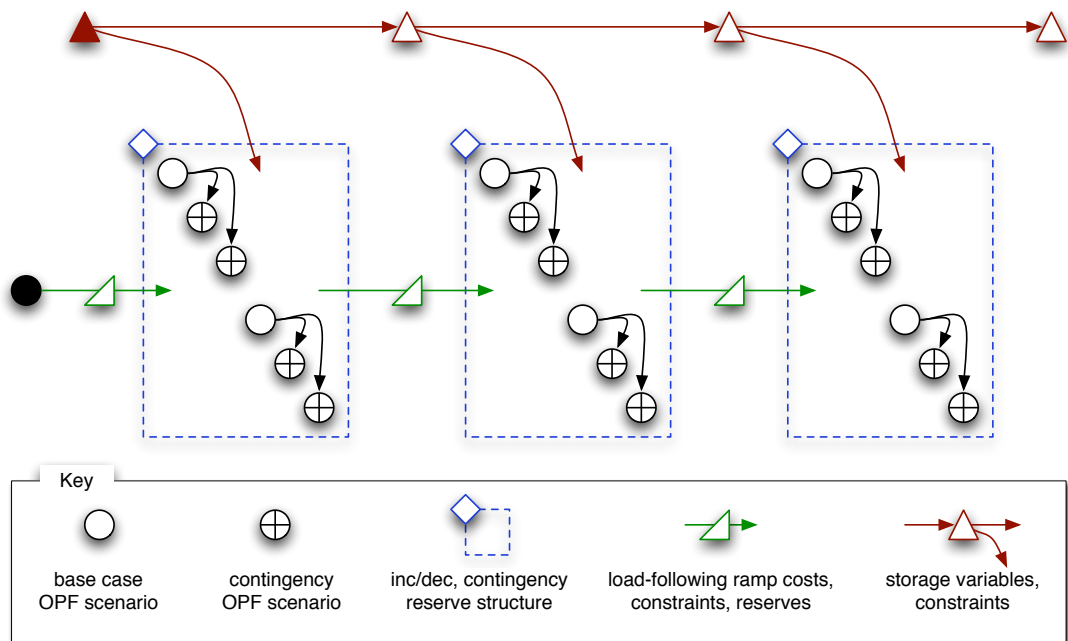
Figure 3-8: Overall Problem Structure

Figure 3-9 illustrates the ways in which any one of the single period continuous variable problems of Figure 3-1 can be extended to include combinations of multiple periods, ramping, storage, integer commitment, startup and shutdown costs and minimum up and down times.
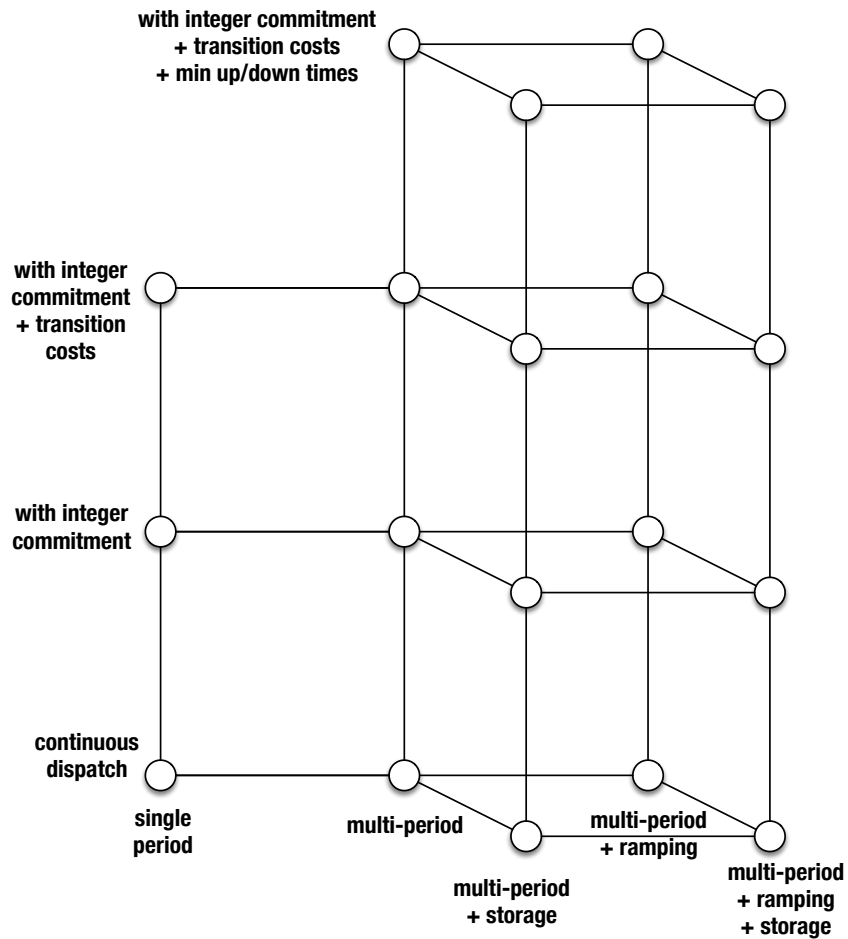
Figure 3-9: MOST Mixed Integer and Multi-Period Problems

4 Problem Formulation

4.1 Nomenclature

This section summarizes the nomenclature for the full multi-period mixed-integer nonlinear problem. In order to simplify the indexing notation, the index literals and their order are maintained wherever possible. Furthermore, no commas are used, so the superindex $tijk$ refers to time period t, generator i (or dispatchable load or storage unit i), base scenario j and contingency k. A dispatchable or curtailable load is modeled as negative generation, as in Section 6.4.2 in the MATPOWER User's Manual, each wind farm as a generator whose maximum output varies by scenario according to the forecast distribution, and a storage unit as a device with a given loss factor, energy capacity, and "charging" and "discharging" power capacities and efficiencies.

Variable and Parameter Indexing

Symbol	Meaning
t	Index over time periods.
T	Set of indices of time periods in planning horizon, typically $\{1 \ldots n_t\}$.
j	Index over scenarios.
J^t	Set of indices of all scenarios considered at time t.
k	Index over post-contingency cases ($k = 0$ for base case, i.e. no contingency occured).
K^{tj}	Set of indices of contingencies considered in scenario j at time t, including base case ($k = 0$).
i	Index over injections (generation units, storage units and dispatchable or curtailable loads).
I^t	Indices of all units (generators, storage and dispatchable or curtailable loads) available for dispatch in *any* contingency at time t.
I^{tjk}	Indices of all units available for dispatch in post-contingency state k of scenario j at time t.

l	Index over reserve zones.
L^{tjk}	Indices of all reserve zones defined in post-contingency state k of scenario j at time t.
Z_l^{tjk}	Set of generators providing reserves in zone l in post-contingency state k of scenario j at time t.
n_t^{ds}	Number of time periods in the horizon of the dynamical system model.

Optimization Variables

Symbol	Meaning
p^{tijk}, q^{tijk}	Active/reactive injection for unit i in post-contingency state k of scenario j at time t.
p_c^{ti}	Active power contract quantity for unit i at time t.
p_+^{tijk}, p_-^{tijk}	Upward/downward deviation from active power contract quantity for unit i in post-contingency state k of scenario j at time t.
r_z^{tijk}	Zonal reserve quantity provided by unit i in post-contingency state k of scenario j at time t.
r_+^{ti}, r_-^{ti}	Upward/downward active contingency reserve quantity provided by unit i at time t.
$\delta_+^{ti}, \delta_-^{ti}$	Upward/downward load-following ramping reserves needed from unit i at time t for transition to time $t+1$.
$\theta^{tjk}, V^{tjk}, p^{tjk}, q^{tjk}$	Voltage angles and magnitudes, active and reactive injections for power flow in post-contingency state k of scenario j at time t.
$p_{\mathrm{sc}}^{tijk}, p_{\mathrm{sd}}^{tijk}$	Charge/discharge power injections of storage unit i in post-contingency state k of scenario j at time t.
s_+^{ti}, s_-^{ti}	Endogenously computed upper/lower bounds on the energy stored in storage unit i at the end of period t. For $t=0$ this is a fixed input parameter representing the bounds at the beginning of the first period.

s_0^i	Initial stored energy (expected) in storage unit i.
u^{ti}	Binary commitment state for unit i in period t, 1 if unit is on-line, 0 otherwise.
v^{ti}, w^{ti}	Binary startup and shutdown states for unit i in period t, 1 if unit has a startup/shutdown event in period t, 0 otherwise.
z^t	Partition ($n_z^{\mathrm{ds}} \times 1$ vector) corresponding to period t of state variable z of the dynamical system model.

Constraint Functions and Parameters

Symbol	Meaning
$g^{tjk}(\cdot)$	Nonlinear AC power flow equations in post-contingency state k of scenario j at time t.
$h^{tjk}(\cdot)$	Transmission, voltage and other limits in post-contingency state k of scenario j at time t.
$P_{\min}^{tijk}, P_{\max}^{tijk}$	Limits on active injection for unit i in post-contingency state k of scenario j at time t.
$Q_{\min}^{tijk}, Q_{\max}^{tijk}$	Limits on reactive injection for unit i in post-contingency state k of scenario j at time t.
$R_{\max+}^{ti}, R_{\max-}^{ti}$	Upward/downward contingency (or zonal) reserve capacity limits for unit i at time t.
R_l^{tjk}	MW reserve requirement for zone l in post-contingency state k of scenario j at time t.
$\delta_{\max+}^{ti}, \delta_{\max-}^{ti}$	Upward/downward load-following ramping reserve limits for unit i at time t for transition to time $t+1$.
Δ_+^i, Δ_-^i	Upward/downward physical ramping limits for unit i for transitions from base ($k=0$) to contingency cases.

ρ^{ti} Parameter for storage unit i at period t that determines the weighting in storage constraints, where the storage dispatch bounds are computed relative to a weighted average of previous period endogenous bounds $s_+^{(t-1)i}$, $s_-^{(t-1)i}$ ($\rho^{ti} = 1$) and period- and scenario-specific initial expected stored energy ($\rho^{ti} = 0$).

$S_{\max}^{ti}, S_{\min}^{ti}$ Stored energy (in MWh) max/min limits for storage unit i at time t.

$S_{\min}^{0i}, S_{\max}^{0i}$ Lower/upper bounds on initial stored energy (expected) in storage unit i.

$S_{\min}^{n_t i}, S_{\max}^{n_t i}$ Lower/upper bounds on target stored energy (expected) in storage unit i at end of final period n_t.

τ_i^+, τ_i^- Minimum up and down times for unit i in number of periods.

$A_{\mathrm{ds}}^t, B_{\mathrm{ds}}^t, C_{\mathrm{ds}}^t, D_{\mathrm{ds}}^t$ Matrices used to define the state transitions and output constraints for the dynamical system model at time t.

$z_{\min}^t, z_{\max}^t$ Lower and upper bounds on dynamical system model state z^t at time t.

$y_{\min}^t, y_{\max}^t$ Lower and upper bounds on dynamical system model output at time t.

Cost Functions and Parameters

Symbol	Meaning
$C_P^{ti}(\cdot)$	Cost function for active injection i at time t.
$C_{P+}^{ti}(\cdot), C_{P-}^{ti}(\cdot)$	Cost for upward/downward deviation from active power contract quantity for unit i at time t.
$C_z^{ti}(\cdot)$	Cost function for zonal reserve purchased from unit i at time t.
$C_{R+}^{ti}(\cdot), C_{R-}^{ti}(\cdot)$	Cost function for upward/downward contingency reserve purchased from unit i at time t.
$C_{\delta+}^{ti}(\cdot), C_{\delta-}^{ti}(\cdot)$	Cost of upward/downward load-following ramp reserve for unit i at time t for transition to time $t+1$.

$C_\delta^i(\cdot)$	Quadratic, symmetric ramping cost on the difference between the dispatches for unit i in adjacent periods.
C_{s0}	Vector of costs by storage unit associated with starting out with a given level of stored energy s_0 in the storage units at time $t = 0$.
C_s	Vector of prices by storage unit for contributions to terminal storage[14] from charging or discharging in non-terminal states.
C_{sc0}, C_{sd0}	Vector of prices by storage unit for contributions to terminal storage[14] from charging/discharging in terminal end-of-horizon base states.
C_{sck}, C_{sdk}	Vector of prices by storage unit for contributions to terminal storage[14] from charging/discharging in terminal contingency states.
C_v^{ti}, C_w^{ti}	Startup and shutdown costs for unit i at time t in \$ per startup/shutdown.

Other Parameters

Symbol	Meaning
Δ	Length of scheduling time slice in hours, typically 1 hour.
$\eta_{\text{in}}^{ti}, \eta_{\text{out}}^{ti}$	Charging/discharging (or pumping/generating) efficiencies for storage unit i at time t.
η_{loss}^{ti}	Fraction of stored energy lost per hour by storage unit i at time t.
α	For contingency cases, the fraction of the time slice that is spent in the base case before the contingency occurs ($\alpha = 0$ means the entire period is spent in the contingency).
ψ_0^{tjk}	Conditional probability of contingency k in scenario j at time t, conditioned on making it to period t without branching off the central path in a contingency in periods $1 \dots t-1$ and on scenario j being realized in some form (base or contingency). ψ_0^{tj0} is the conditional probability of no contingency, i.e. the base case.

---

[14]That is, expected leftover stored energy in terminal states.

Symbol	Meaning

$\phi^{tj_2j_1}$ — Probability of transitioning to scenario j_2 in period t given that scenario j_1 was realized in period $t-1$.

$\zeta^{tj_2j_1}$ — Binary valued mask indicating whether transition to scenario j_2 in period t from scenario j_1 in period $t-1$ should be included in load-following ramp requirements.

Derived Parameters

Symbol	Meaning

$\widetilde{C}_P^{ti}(\cdot)$ — Modified cost function for active injection i at time t with the no load cost subtracted, $\widetilde{C}_P^{ti}(p) \equiv C_P^{ti}(p) - C_P^{ti}(0)$.

$C_{\text{ts0}}, C_{\text{tsc}}, C_{\text{tsd}}$ — Weighted price vectors summarizing contributions to the value of terminal storage[14] from initial storage/charging/discharging, derived from C_s, C_{sc0}, C_{sd0}, $C_{\text{sc}k}$, $C_{\text{sd}k}$.[15]

ψ^{tjk} — Probability of contingency k in scenario j at time t, derived from transition probabilities $\phi^{tj_2j_1}$ and conditional probabilities of contingencies ψ_0^{tjk}. ψ^{tj0} is the probability of no contingency, i.e. the base case.

ψ_α^{tjk} — Probability ψ^{tjk} of contingency k in scenario j at time t adjusted for α.

$$\psi_\alpha^{tjk} = \begin{cases} \psi^{tj0} + \alpha \displaystyle\sum_{\kappa \in K^{tj} \neq 0} \psi^{tj\kappa}, & k = 0 \\[2mm] (1-\alpha)\psi^{tjk}, & \forall k \in K^{tj} \neq 0 \end{cases} \tag{4.1}$$

γ^t — Probability of making it to period t without branching off the central path in a contingency in periods $1 \ldots t-1$.

$$\gamma^t \equiv \sum_{j \in J^{t-1}} \psi^{(t-1)j0} = \sum_{j \in J^t, k \in K^{tj}} \psi^{tjk} \tag{4.2}$$

$\beta_1^{ti} \ldots \beta_5^{ti}$ — Storage parameters defined in (4.61) and (4.69)–(4.71) in terms of Δ, α and η_{loss}^{ti} to simplify storage constraint expression.

---

[15]See Section 4.4, especially (4.104)–(4.107), for details.

Derived Variables [16]

Symbol	Meaning
s_Δ^{tijk}	Net increase in stored energy due to charging or discharging for unit i in post-contingency state k of scenario j at time t.

$$s_\Delta^{tijk} \equiv -\Delta(\eta_{\text{in}}^{ti} p_{\text{sc}}^{tijk} + \frac{1}{\eta_{\text{out}}^{ti}} p_{\text{sd}}^{tijk}) \tag{4.3}$$

Symbol	Meaning
$\bar{S}_I^{tj}$	Vector of expected stored energy for all storage units in base scenario j at the beginning of period t, defined as a linear function of s_0, p_{sc} and p_{sd}.[17]
$\bar{s}_I^{tij0}$	Expected stored energy in storage unit i in base scenario j at the beginning of period t (i-th element of $\bar{S}_I^{tj}$).
$s_F^{n_t i}$	Expected stored energy in storage unit i at the end of period n_t, the final period.[17]
$\bar{p}^t$	Vector of expected values of p^{tijk} across j and k at time t, conditional on making it to time t.

$$\bar{p}^t = \frac{1}{\gamma^t} \sum_{j \in J^t, k \in K^{tj}} \psi_\alpha^{tjk} p^{tjk} \tag{4.4}$$

Individual variables can be grouped into vectors such as p^t for all active injections considered across all scenarios and contingencies at hour t and it will be consistent with the context. The subset referring to scenario j would be p^{tj}.

---

[16]All are linear functions of the optimization variables, used only to simplify the presentation.
[17]See Section 4.4 for details.

4.2 Formulation

The problem formulation can be expressed as a mixed-integer nonlinear optimization problem, where the optimization variable x is comprised of all the θ, V, p, q, p_c, p_+, p_-, r_z, r_+, r_-, δ_+, δ_-, p_{sc}, p_{sd}, s_0, s_+, s_-, u, v, w, and z variables, for all t, j, k, i and l. The u commitment variables are binary and the rest continuous. For simplicity, the formulation restricts the treatment of costs, deviations, ramping and reserves to consider only active power, but an extension to include reactive counterparts is straightforward.

4.2.1 Objective Function

The objective then is to

$$\min_{x} f(x) \tag{4.5}$$

where $f(x)$ is comprised of seven components.[18]

$$
\begin{aligned}
f(x) = {}& f_p(p, p_+, p_-) + f_z(r_z) + f_r(r_+, r_-) + f_\delta(p) \\
& + f_{lf}(\delta_+, \delta_-) + f_s(s_0, p_{sc}, p_{sd}) + f_{uc}(u, v, w)
\end{aligned} \tag{4.6}
$$

Each part is expressed in terms of the individual optimization variables as follows.

   – expected cost of active power dispatch and redispatch

$$f_p(p, p_+, p_-) = \sum_{t\in T}\sum_{j\in J^t}\sum_{k\in K^{tj}} \psi_\alpha^{tjk} \sum_{i\in I^{tjk}} \left[\widetilde{C}_P^{ti}(p^{tijk}) + C_{P+}^{ti}(p_+^{tijk}) + C_{P-}^{ti}(p_-^{tijk}) \right] \tag{4.7}$$

   – cost of zonal reserves[18]

$$f_z(r_z) = \sum_{t\in T}\sum_{j\in J^t}\sum_{k\in K^{tj}} \psi_\alpha^{tjk} \sum_{i\in I^{tjk}} C_z^{ti}(r_z^{tijk}) \tag{4.8}$$

   – cost of endogenous contingency reserves[18]

$$f_r(r_+, r_-) = \sum_{t\in T} \gamma^t \sum_{i\in I^t} \left[C_{R+}^{ti}(r_+^{ti}) + C_{R-}^{ti}(r_-^{ti}) \right] \tag{4.9}$$

---

[18]A typical secure problem uses either zonal reserves or endogenous contingency reserves, but not both. Zonal reserves are typically used only for cases with a single base scenario with no contingencies, that is, when $J^t = \{1\}$ and $K^{tj} = \{0\}$.

– expected cost of load-following ramping (wear and tear)

$$f_\delta(p) = \sum_{t\in T}\gamma^t\sum_{\substack{j_1\in J^{t-1}\\ j_2\in J^t}}\phi^{tj_2j_1}\sum_{i\in I^{tj_20}}C_\delta^i(p^{tij_20} - p^{(t-1)ij_10}) \tag{4.10}$$

– cost of load-following ramp reserves

$$f_{\mathrm{lf}}(\delta_+,\delta_-) = \sum_{t\in T}\gamma^t\sum_{i\in I^t}\left[C_{\delta+}^{ti}(\delta_+^{ti}) + C_{\delta-}^{ti}(\delta_-^{ti})\right] \tag{4.11}$$

– cost of initial stored energy and value (since it is negative) of expected leftover stored energy in terminal states

$$f_s(s_0, p_{\mathrm{sc}}, p_{\mathrm{sd}}) = C_{s0}^\mathsf{T}s_0 - (C_{\mathrm{ts0}}^\mathsf{T}s_0 + C_{\mathrm{tsc}}^\mathsf{T}p_{\mathrm{sc}} + C_{\mathrm{tsd}}^\mathsf{T}p_{\mathrm{sd}}) \tag{4.12}$$

– no load, startup and shutdown costs

$$f_{\mathrm{uc}}(u, v, w) = \sum_{t\in T}\gamma^t\sum_{i\in I^t}\left(C_P^{ti}(0)u^{ti} + C_v^{ti}v^{ti} + C_w^{ti}w^{ti}\right) \tag{4.13}$$

4.2.2   Constraints

This minimization is subject to the following constraints, for all $t \in T$, all $j \in J^t$, all $k \in K^{tj}$, all $i \in I^{tjk}$, and all $l \in L^{tjk}$, beginning with the constraints that are separable by period.

Standard OPF Constraints

– power balance equations[19]

$$g^{tjk}(\theta^{tjk}, V^{tjk}, p^{tjk}, q^{tjk}) = 0 \tag{4.14}$$

– transmission flow limits, voltage limits, any other OPF inequality constraints[20]

$$h^{tjk}(\theta^{tjk}, V^{tjk}, p^{tjk}, q^{tjk}) \leq 0 \tag{4.15}$$

---

[19]These can take the form of nonlinear AC power balance equations *(not yet implemented)* (4.2) and (4.3), linear DC power balance equations (6.16), or a simple equating of total demand and total supply, where the equation numbers referenced are from the MATPOWER User's Manual.

[20]These can also take the form of inequality constraints from a nonlinear AC OPF *(not yet implemented)* including (6.7), (6.8) and (6.11), a linear DC OPF including (6.17) and (6.18), or a simple economic dispatch, where the equation numbers referenced are from the MATPOWER User's Manual.

Security Constraints (Option 1): Zonal Reserve Requirements[21]

– fixed zonal reserve requirements

$$0 \le r_z^{tijk} \le \min(R_{\max+}^{ti}, \Delta_+^i) \tag{4.16}$$

$$p^{tijk} + r_z^{tijk} \le u^{ti} P_{\max}^{tijk} \tag{4.17}$$

$$\sum_{i \in Z_l^{tjk}} r_z^{tijk} \ge R_l^{tjk} \tag{4.18}$$

Security Constraints (Option 2): Contingency Constraints[21]

– reserve, redispatch and contract variables

$$0 \le p_+^{tijk} \le r_+^{ti} \le R_{\max+}^{ti} \tag{4.19}$$

$$0 \le p_-^{tijk} \le r_-^{ti} \le R_{\max-}^{ti} \tag{4.20}$$

$$p^{tijk} - p_c^{ti} = p_+^{tijk} - p_-^{tijk} \tag{4.21}$$

– ramping limits on transitions from base to contingency cases

$$-\Delta_-^i \le p^{tijk} - p^{tij0} \le \Delta_+^i, \quad k \ne 0 \tag{4.22}$$

The remaining sets of constraints include intertemporal restrictions and typically include constraints from a known starting point at $t = 0$ into the first period.

Load-following Ramping Limits and Reserves

– variable limits

$$0 \le \delta_+^{ti} \le \delta_{\max+}^{ti} \tag{4.23}$$

$$0 \le \delta_-^{ti} \le \delta_{\max-}^{ti} \tag{4.24}$$

– load-following ramp reserve definition

$$\forall \{t \in T, i \in I^{tjk}, j_1 \in J^{t-1}, j_2 \in J^t \mid \zeta^{tj_2j_1} = 1\} : \tag{4.25}$$

$$p^{tij_20} - p^{(t-1)ij_10} \le \delta_+^{(t-1)i} \tag{4.26}$$

$$p^{(t-1)ij_10} - p^{tij_20} \le \delta_-^{(t-1)i} \tag{4.27}$$

---

[21]A typical secure problem uses either zonal reserves or endogenous contingency reserves, but not both. Zonal reserves are typically used only for cases with a single base scenario with no contingencies, that is, when $J^t = \{1\}$ and $K^{tj} = \{0\}$.

Storage Constraints

– storage dispatch definition and limits

$$p^{tijk} = p_{\text{sc}}^{tijk} + p_{\text{sd}}^{tijk} \tag{4.28}$$

$$p_{\text{sc}}^{tijk} \leq 0 \tag{4.29}$$

$$p_{\text{sd}}^{tijk} \geq 0 \tag{4.30}$$

– energy bound limits

$$s_-^{ti} \geq S_{\min}^{ti} \tag{4.31}$$

$$s_+^{ti} \leq S_{\max}^{ti} \tag{4.32}$$

– storage dispatch vs. base scenario energy bounds

$$s_-^{ti} \leq \beta_1^{ti} \left[\rho^{ti} s_-^{(t-1)i} + \left(1 - \rho^{ti}\right) \bar{s}_I^{tij0} \right] + \beta_2^{ti} s_\Delta^{tij0} \tag{4.33}$$

$$s_+^{ti} \geq \beta_1^{ti} \left[\rho^{ti} s_+^{(t-1)i} + \left(1 - \rho^{ti}\right) \bar{s}_I^{tij0} \right] + \beta_2^{ti} s_\Delta^{tij0} \tag{4.34}$$

– storage dispatch vs. contingency scenario energy limits

$$S_{\min}^{ti} \leq \beta_5^{ti} \left[\rho^{ti} s_-^{(t-1)i} + \left(1 - \rho^{ti}\right) \bar{s}_I^{tij0} \right] + \beta_4^{ti} s_\Delta^{tij0} + \beta_3^{ti} s_\Delta^{tijk}, \quad k \neq 0 \tag{4.35}$$

$$S_{\max}^{ti} \geq \beta_5^{ti} \left[\rho^{ti} s_+^{(t-1)i} + \left(1 - \rho^{ti}\right) \bar{s}_I^{tij0} \right] + \beta_4^{ti} s_\Delta^{tij0} + \beta_3^{ti} s_\Delta^{tijk}, \quad k \neq 0 \tag{4.36}$$

– optional storage constraints

  – Option 1 : Constrain the expected final stored energy in each unit at the end of the horizon[22] to equal some target value or lie in some target range.

$$S_{\min}^{n_t i} \leq s_F^{n_t i} \leq S_{\max}^{n_t i} \tag{4.37}$$

  – Option 2 : Constrain the expected final stored energy at the end of the horizon[22] to equal the initial stored energy.

$$s_F^{n_t i} = s_0^i \tag{4.38}$$

$$S_{\min}^{0i} \leq s_0^i \leq S_{\max}^{0i} \tag{4.39}$$

  When using this option s_0^i is an optimization variable that can take on any value between its bounds. When not using this option, it is simply a fixed parameter.

---

[22]See (4.93) in the Section 4.4 for details on how $s_F^{n_t i}$ is computed as a linear function of x.

Unit Commitment

   – injection limits and commitments

$$u^{ti} P_{\min}^{tijk} \leq p^{tijk} \leq u^{ti} P_{\max}^{tijk} \tag{4.40}$$

$$u^{ti} Q_{\min}^{tijk} \leq q^{tijk} \leq u^{ti} Q_{\max}^{tijk} \tag{4.41}$$

   – startup and shutdown events

$$u^{ti} - u^{(t-1)i} = v^{ti} - w^{ti} \tag{4.42}$$

$$0 \leq v^{ti} \leq 1 \tag{4.43}$$

$$0 \leq w^{ti} \leq 1 \tag{4.44}$$

   – minimum up and down times

$$\sum_{y=t-\tau_i^+ +1}^{t} v^{yi} \leq u^{ti} \tag{4.45}$$

$$\sum_{y=t-\tau_i^- +1}^{t} w^{yi} \leq 1 - u^{ti} \tag{4.46}$$

Note: These summations can be made to "wrap around" to implement a cyclic commitment schedule where commitment transitions from the last period of the horizon back to the first are also constrained to be feasible.

   – integrality constraints

$$u^{ti} \in \{0, 1\} \tag{4.47}$$

Linear Time-Varying Dynamical System

The partition of the state variable corresponding to time t is denoted z^t, the corresponding output by y^t (not an explicit optimization variable), and $\bar{p}^t$ represents the vector of expected values of p^{tijk} across j and k at time t, conditional on making it to time t, as defined in (4.4). That is

$$\bar{p}^t = \frac{1}{\gamma^t} \sum_{j \in J^t, k \in K^{tj}} \psi_\alpha^{tjk} p^{tjk} \tag{4.48}$$

– state bounds

$$z^t_{\min} \le z^t \le z^t_{\max}, \qquad\qquad t = 1 \dots n^{ds}_t \qquad\qquad (4.49)$$

– state update equations

$$z^{t+1} = A^t_{ds} z^t + B^t_{ds} \bar{p}^t, \qquad t = 1 \dots n_t \qquad\qquad (4.50)$$
$$z^{t+1} = A^t_{ds} z^t, \qquad\qquad t = (n_t + 1) \dots (n^{ds}_t - 1) \qquad (4.51)$$

– output equations

$$y^t_{\min} \le C^t_{ds} z^t + D^t_{ds} \bar{p}^t \le y^t_{\max}, \qquad t = 1 \dots n_t \qquad\qquad (4.52)$$
$$y^t_{\min} \le C^t_{ds} z^t \qquad\qquad \le y^t_{\max}, \qquad t = (n_t + 1) \dots n^{ds}_t \qquad (4.53)$$

The initial state z^1 is not a variable, but is assumed to be a given initial condition.

4.3   Probability Weighting of Base and Contingency States

This section describes the assumptions regarding the probabilistic weighting of each term in the cost function. A given initial state at time $t = 0$, with known dispatches and energy storage states, is assumed to have probability 1. From this initial state, transitions, each with some known probability, are possible to any of the scenarios considered for period $t = 1$. Hence the probabilities assigned to each of the states in period $t = 1$ sum to 1.

Now, consider the transition into $t = 2$. This transition is only possible provided the system did not branch off into any contingency at time $t = 1$, that is, provided the realized state at $t = 1$ is one of the base scenarios. However, given that at least one contingency has a non-zero probability of occurrence, the sum of the probabilities of these base cases for $t = 1$ is less than 1. So the probability of actually making it to $t = 2$ in the considered graph is equal to the sum of the probabilities of the base cases at $t = 1$, which is less than 1. Generalizing for period t, the probability of making it to period t is equal to the sum of the probabilities of the base cases at $t - 1$, a value less than 1 except when $t = 1$:

$$\gamma^t = \sum_{j \in J^t} \psi^{(t-1)j0} = \sum_{j \in J^t, k \in K^{tj}} \psi^{tjk} < 1, \;\; t > 1 \qquad\qquad (4.54)$$

While the fact that the probabilities in future periods do not sum to 1 may seem odd, this results from choosing, in an $N - 1$ contingency fashion, to ignore the

cost implications of resuming normal operations after branching off in a contingency, since that would involve exploration of a geometric number of possible paths. This implies, for the branches that have been trimmed, the existence of an unknown cost with respect to the decision variables. Since we do not have any information about the relationship of this unknown cost to our decisions, we explicitly ignore its impact by excluding it from the optimization.

The probability of transitioning to scenario j_2 in period t given that scenario j_1 was realized in period $t-1$ is assumed to be a known value $\phi^{tj_2j_1}$. These transition probabilities for each time step t can be arranged in a transition probability matrix.

$$
\Phi^t = \begin{bmatrix}
\phi^{t11} & \phi^{t12} & \cdots & \phi^{t1n_{Jt-1}} \\
\phi^{t21} & \phi^{t22} & \cdots & \phi^{t2n_{Jt-1}} \\
\vdots & \vdots & \ddots & \vdots \\
\phi^{tn_{Jt}1} & \phi^{tn_{Jt}2} & \cdots & \phi^{tn_{Jt}n_{Jt-1}}
\end{bmatrix}
\tag{4.55}
$$

The columns of Φ^t sum to 1, and its coefficients are used to weight the wear and tear costs of ramping.

The individual state specific probabilities ψ^{tjk} for period t can be derived from those in period $t-1$ in two steps. First, the probability γ^{tj} that scenario j or any of its associated contingencies will occur at time t is given by

$$
\begin{bmatrix}
\gamma^{t1} \\
\gamma^{t2} \\
\vdots \\
\gamma^{tn_{Jt}}
\end{bmatrix}
= \Phi^t
\begin{bmatrix}
\psi^{(t-1)10} \\
\psi^{(t-1)20} \\
\vdots \\
\psi^{(t-1)n_{Jt-1}0}
\end{bmatrix}
\tag{4.56}
$$

where

$$
\gamma^{tj} = \sum_{k \in K^{tj}} \psi^{tjk}.
\tag{4.57}
$$

Since the sum across k of the conditional probabilities of contingencies ψ_0^{tjk} is 1, we simply scale each by the corresponding γ^{tj} to get the correct state specific probabilities

$$
\psi^{tjk} = \gamma^{tj}\psi_0^{tjk}.
\tag{4.58}
$$

Caveat Regarding Unit Commitment

The derivation of scenario probabilities presented above is based on the assumption that the conditional probability of a contingency occuring in any given state

j is fixed and independent of any optimization variables. However, in the context of unit commitment, this is not a valid assumption for a generator outage contingency. In this case the probability of that contingency goes to zero if the generator is not committed. The current MOST implementation uses the formulation described above and does not take into account this dependency of contingency probabilities on commitment status.

4.4 Value of Residual Storage

Given the complexity of the storage model, numerous derived parameters and variables were used in Section 4.2 to simplify the presentation of the problem formulation. The specifics of these derivations are presented here. This includes details of the sixth term $f_s(\cdot)$ of the objective function (4.6), specifically the last three terms of (4.12) related to the expected residual value of stored energy in terminal states. It also includes details of the storage constraints (4.33)–(4.38).

First, for each storage resource i, an efficient method is needed to compute the expected amount of stored energy at the beginning and end of each period t for each scenario j. We will denote these by the $n_{J^t} \times 1$ vectors S_I^{ti} and S_F^{ti}, respectively, where n_{J^t} is the number of scenarios in period t.

The stored energy s_F^{tij0} in unit i at the end of period t in base state j can be computed deterministically from the stored energy at the beginning s_I^{tij0} and the injections in that state, where the losses are assumed to be proportional to the average stored energy during the period. Using the definition of s_Δ^{tijk} from (4.3), this relationship can be expressed as follows

$$s_F^{tij0} = s_I^{tij0} + s_\Delta^{tij0} - \Delta\eta_{\text{loss}}^{ti} \frac{s_I^{tij0} + s_F^{tij0}}{2} \tag{4.59}$$

$$= \beta_1^{ti} s_I^{tij0} + \beta_2^{ti} s_\Delta^{tij0}, \tag{4.60}$$

where

$$\beta_1^{ti} \equiv \frac{1 - \Delta\frac{\eta_{\text{loss}}^{ti}}{2}}{1 + \Delta\frac{\eta_{\text{loss}}^{ti}}{2}}, \quad \beta_2^{ti} \equiv \frac{1}{1 + \Delta\frac{\eta_{\text{loss}}^{ti}}{2}}. \tag{4.61}$$

For a period where a contingency occurs at a fraction α of the way through the period, the losses are more tricky to compute. Let us call the expected stored energy at the moment the contingency occurs s_α^{tijk}, expressed as

$$s_\alpha^{tijk} = s_I^{tijk} + \alpha(s_F^{tij0} - s_I^{tijk}). \tag{4.62}$$

Then the losses are equal to

$$s_{\text{loss}}^{tijk} = \Delta\eta_{\text{loss}}^{ti} \left[\alpha \frac{s_I^{tijk} + s_\alpha^{tijk}}{2} + (1 - \alpha) \frac{s_\alpha^{tijk} + s_F^{tijk}}{2} \right] \tag{4.63}$$

$$= \Delta\eta_{\text{loss}}^{ti} \left[\alpha \frac{s_I^{tij0} + s_F^{tij0}}{2} + (1 - \alpha) \frac{s_I^{tijk} + s_F^{tijk}}{2} \right] \tag{4.64}$$

43

where (4.64) follows directly from (4.62) and (4.63), keeping in mind that $s_I^{tijk} = s_I^{tij0}$.

In this case, the stored energy in unit i at the end of period t in state jk can be computed deterministically from the stored energy at the beginning and the injections in states $j0$ and jk as follows.

$$s_F^{tijk} = s_I^{tijk} + \alpha s_\Delta^{tij0} + (1-\alpha)s_\Delta^{tijk} - s_{\text{loss}}^{tijk} \tag{4.65}$$

$$= \alpha \left[s_I^{tij0} + s_\Delta^{tij0} - \Delta\eta_{\text{loss}}^{ti} \frac{s_I^{tij0} + s_F^{tij0}}{2} \right]$$

$$+ (1-\alpha) \left[s_I^{tijk} + s_\Delta^{tijk} - \Delta\eta_{\text{loss}}^{ti} \frac{s_I^{tijk} + s_F^{tijk}}{2} \right] \tag{4.66}$$

$$= \alpha \left[\beta_1^{ti} s_I^{tij0} + \beta_2^{ti} s_\Delta^{tij0} \right] + (1-\alpha) \left[s_I^{tijk} + s_\Delta^{tijk} - \Delta\eta_{\text{loss}}^{ti} \frac{s_I^{tijk} + s_F^{tijk}}{2} \right] \tag{4.67}$$

$$= \beta_5^{ti} s_I^{tijk} + \beta_4^{ti} s_\Delta^{tij0} + \beta_3^{ti} s_\Delta^{tijk} \tag{4.68}$$

where

$$\beta_3^{ti} \equiv \left(\frac{1}{1-\alpha} + \Delta\frac{\eta_{\text{loss}}^{ti}}{2} \right)^{-1}$$

$$= \frac{1-\alpha}{1 + (1-\alpha)\Delta\frac{\eta_{\text{loss}}^{ti}}{2}} \tag{4.69}$$

$$\beta_4^{ti} \equiv \frac{\alpha}{1-\alpha} \beta_2^{ti} \beta_3^{ti}$$

$$= \frac{\alpha}{(1 + \Delta\frac{\eta_{\text{loss}}^{ti}}{2})(1 + (1-\alpha)\Delta\frac{\eta_{\text{loss}}^{ti}}{2})} \tag{4.70}$$

$$\beta_5^{ti} \equiv \frac{\beta_1^{ti}}{\beta_2^{ti}} (\beta_3^{ti} + \beta_4^{ti})$$

$$= \left(1 - \Delta\frac{\eta_{\text{loss}}^{ti}}{2} \right) \frac{\alpha + (1-\alpha)(1 + \Delta\frac{\eta_{\text{loss}}^{ti}}{2})}{(1 + \Delta\frac{\eta_{\text{loss}}^{ti}}{2})(1 + (1-\alpha)\Delta\frac{\eta_{\text{loss}}^{ti}}{2})} \tag{4.71}$$

Let G_k^{ti} and H_k^{ti} be matrices containing appropriately placed efficiencies relating the charging and discharging injections, respectively, in state jk of storage unit i in period t to the corresponding change in stored energy from the beginning to the end of the period. Specifically, the elements g_{jl}^{ti} and h_{jl}^{ti} in row j and column l of G_k^{ti} and

H_k^{ti} are set as follows

$$g_{jl}^{ti} = \begin{cases} -\Delta\eta_{\text{in}}^{ti}, & \text{where column } l \text{ corresponds to } p_{\text{sc}}^{tijk} \\ 0, & \text{otherwise} \end{cases} \tag{4.72}$$

$$h_{jl}^{ti} = \begin{cases} -\Delta\dfrac{1}{\eta_{\text{out}}^{ti}}, & \text{where column } l \text{ corresponds to } p_{\text{sd}}^{tijk} \\ 0, & \text{otherwise} \end{cases} \tag{4.73}$$

The reason for keeping G_k^{ti} and H_k^{ti} separate is to make it possible to use different prices to represent the gain in value from increasing the amount of residual storage and the loss in value from reducing the amount of residual storage. The need to use different prices to value charging and discharging is supported by the intuition that stored energy should not be used in a given terminal state if there is a better time to use it (expect a higher price on the horizon), neither should we be storing additional energy in a given terminal state if there is a better time to store it (expect a lower price on the horizon).

Using these matrices, (4.60) can be expressed for the vector S_F^{ti} as a deterministic function of S_I^{ti} and the injections as

$$S_F^{ti} = \beta_1^{ti} S_I^{ti} + \beta_2^{ti}(G_0^{ti} + H_0^{ti})x \tag{4.74}$$

On the other hand, the expected stored energy in each scenario at the beginning of period t depends on the corresponding values at the end of period $t-1$ and the transition probabilities. Let σ^t equal the vector of probabilities of each of the base scenarios at the end of period $t-1$, conditional on arriving at the end of that period without the occurence of a continency.

$$\sigma^t \equiv \frac{1}{\gamma^t}\psi^{(t-1)J0} = \frac{1}{\gamma^t}\begin{bmatrix} \psi^{(t-1)10} \\ \psi^{(t-1)20} \\ \vdots \\ \psi^{(t-1)n_{J_{t-1}}0} \end{bmatrix}. \tag{4.75}$$

If we also let $[a]$ denote a diagonal matrix with the vector a on the main diagonal, then the relationship between S_I^{ti} and $S_F^{(t-1)i}$ can be expressed as

$$\left[\Phi^t\sigma^t\right]S_I^{ti} = \Phi^t\left[\sigma^t\right]S_F^{(t-1)i}. \tag{4.76}$$

In other words,

$$S_I^{ti} = D^{ti}S_F^{(t-1)i} \tag{4.77}$$

45

where

$$D^{ti} \equiv \begin{cases} \mathbf{1}_{n_{Jt} \times 1}, & t = 1 \\ \left[\Phi^t \sigma^t\right]^{-1} \Phi^t \left[\sigma^t\right], & t \neq 1. \end{cases} \tag{4.78}$$

Stacking the vectors S_I^{ti} and S_F^{ti} for all storage units (i from 1 to n_s) allows the relationships above to be expressed in terms of matrices formed by stacking the D^{ti} along the diagonals and the G_k^{ti} and H_k^{ti} vertically.

$$D^t = \begin{bmatrix} D^{t1} & 0 & \cdots & 0 \\ 0 & D^{t2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D^{tn_s} \end{bmatrix}, G_k^t = \begin{bmatrix} G_k^{t1} \\ G_k^{t2} \\ \vdots \\ G_k^{tn_s} \end{bmatrix}, H_k^t = \begin{bmatrix} H_k^{t1} \\ H_k^{t2} \\ \vdots \\ H_k^{tn_s} \end{bmatrix} \tag{4.79}$$

Similarly, scalars β_n^{ti} are converted to diagonal matrices $B_n^{ti} \equiv \beta_n^{ti} \cdot I_{n_{Jt} \times n_{Jt}}$ and stacked to form

$$B_n^t = \begin{bmatrix} B_n^{t1} & 0 & \cdots & 0 \\ 0 & B_n^{t2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_n^{tn_s} \end{bmatrix}. \tag{4.80}$$

The full expression for all storage units in all scenarios in period t can then be expressed as follows.

$$S_I^t = D^t S_F^{(t-1)} \tag{4.81}$$
$$S_F^t = B_1^t S_I^t + B_2^t (G_0^t + H_0^t) x \tag{4.82}$$

The relationships in (4.81) and (4.82) imply that the expected stored energy at any point in the planning horizon can be expressed in the following form as a linear function of the expected initial stored energy s_0 and the active power injections in x, specifically the injections of the storage units.

$$S_I^t = L_I^t s_0 + (M_g^t + M_h^t) x \tag{4.83}$$
$$S_F^t = L_F^t s_0 + (N_g^t + N_h^t) x \tag{4.84}$$

The following recursive expressions can be used for computing L_I^t, L_F^t, M_g^t, M_h^t, N_g^t

and N_h^t

$$
\begin{align}
L_I^t &= D^t L_F^{(t-1)} = D^t B_1^{(t-1)} L_I^{(t-1)} \tag{4.85}\\
L_F^t &= B_1^t L_I^t = B_1^t D^t L_F^{(t-1)} \tag{4.86}\\
M_g^t &= D^t N_g^{(t-1)} \tag{4.87}\\
M_h^t &= D^t N_h^{(t-1)} \tag{4.88}\\
N_g^t &= B_1^t M_g^t + B_2^t G_0^t \tag{4.89}\\
N_h^t &= B_1^t M_h^t + B_2^t H_0^t, \tag{4.90}
\end{align}
$$

where $L_I^1 = D^1$ and $M_g^1 = M_h^1 = \mathbf{0}$.

If the rows of each of these vectors and matrices are sorted and partitioned by scenario (as opposed to by storage unit), we can denote the resulting j-th components, whose i-th row corresponds to storage unit i, with a bar, for example $\bar{S}_F^{tj}$, $\bar{S}_I^{tj}$, $\bar{G}_k^{tj}$, $\bar{H}_k^{tj}$, $\bar{L}_I^{tj}$, $\bar{L}_F^{tj}$, $\bar{M}_g^{tj}$, $\bar{M}_h^{tj}$, $\bar{N}_g^{tj}$ and $\bar{N}_h^{tj}$. It should be noted that for the B matrices, the corresponding $\bar{B}_n^{tj}$ is just the diagonal matrix $[\beta_n^t]$, with the individual β_n^{ti} on the diagonal. Using this notation, the expected residual stored energy for all units in a base scenario j at the end of the last period n_t of the horizon, the vector $\bar{S}_F^{n_t j}$ can be written as a function of these matrices

$$
\begin{align}
\bar{S}_F^{n_t j} &= [\beta_1^t]\bar{S}_I^{n_t j} + [\beta_2^t](\bar{G}_0^{n_t j} + \bar{H}_0^{n_t j})x \notag\\
&= [\beta_1^t]\left(\bar{L}_I^{n_t j} s_0 + (\bar{M}_g^{n_t j} + \bar{M}_h^{n_t j})x\right) + [\beta_2^t](\bar{G}_0^{n_t j} + \bar{H}_0^{n_t j})x \notag\\
&= [\beta_1^t]\bar{L}_I^{n_t j} s_0 + \left([\beta_1^t]\bar{M}_g^{n_t j} + [\beta_2^t]\bar{G}_0^{n_t j} + [\beta_1^t]\bar{M}_h^{n_t j} + [\beta_2^t]\bar{H}_0^{n_t j}\right) x. \tag{4.91}
\end{align}
$$

Likewise, the expected residual stored energy at the end of period t for any scenario j and contingency k is expressed as follows,

$$
\begin{align}
\bar{S}_F^{tjk} &= [\beta_5^t]\bar{S}_I^{tj} + [\beta_4^t]\bar{S}_\Delta^{tj0} + [\beta_3^t]\bar{S}_\Delta^{tjk} \notag\\
&= [\beta_5^t]\bar{S}_I^{tj} + \left([\beta_4^t](\bar{G}_0^{tj} + \bar{H}_0^{tj}) + [\beta_3^t](\bar{G}_k^{tj} + \bar{H}_k^{tj})\right) x \notag\\
&= [\beta_5^t]\left(\bar{L}_I^{tj} s_0 + (\bar{M}_g^{tj} + \bar{M}_h^{tj})x\right) + \left([\beta_4^t](\bar{G}_0^{tj} + \bar{H}_0^{tj}) + [\beta_3^t](\bar{G}_k^{tj} + \bar{H}_k^{tj})\right) x \notag\\
&= [\beta_5^t]\bar{L}_I^{tj} s_0 + \big([\beta_5^t]\bar{M}_g^{tj} + [\beta_4^t]\bar{G}_0^{tj} + [\beta_3^t]\bar{G}_k^{tj} \notag\\
&\qquad + [\beta_5^t]\bar{M}_h^{tj} + [\beta_4^t]\bar{H}_0^{tj} + [\beta_3^t]\bar{H}_k^{tj}\big) x. \tag{4.92}
\end{align}
$$

The overall expected quantity of stored energy across all non-contingency states at the end of the horizon is given by

$$
s_F^{n_t} = \frac{1}{\gamma^{(n_t+1)}} \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{S}_F^{n_t j} \tag{4.93}
$$

47

where $\gamma^{(n_t+1)} = \sum_j \psi^{n_t j 0}$. This expression can be used in contraints, such as (4.37) or (4.38) or in constructing terms of the objective function.

Finally, we return to the value, call it $v_S(x)$, of the expected stored energy leftover in terminal states, expressed in the last three terms of f_s in (4.12).

$$v_S(x) = C_{\text{ts0}}^{\mathsf{T}} s_0 + C_{\text{tsc}}^{\mathsf{T}} p_{\text{sc}} + C_{\text{tsd}}^{\mathsf{T}} p_{\text{sd}} \tag{4.94}$$

If we were to use a single price for each storage unit i to value all contributions to that expected leftover energy, regardless of the state in which they occur, then the value $v_S(x)$ would be that price times a simple probability-weighted sum of the energy in each state, modified by the output efficiency. To be more precise, the price relates to the value of each MW of *recoverable* energy[23] as opposed to *stored* energy.

$$v_S(x) = C_s^{\mathsf{T}} \left([\eta_{\text{out}}^{n_t}] \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{S}_F^{n_t j} + \sum_{t \in T} [\eta_{\text{out}}^t] \sum_{j \in J^t} \sum_{k \in K^{tj} \neq 0} \psi^{tjk} \bar{S}_F^{tjk} \right) \tag{4.95}$$

However, it may be useful to classify the system states into three categories: terminal contingency states, terminal end-of-horizon base states, and non-terminal states (base states preceding the last period). This allows for the possibility of valuing differently the contributions made to the expected terminal stored energy in each of these categories of states. It may also be useful to differentiate between the value gained by increasing the expected terminal stored energy and the value lost by decreasing it.

Table 4-1: Five Price Model

price	applies to contributions from ...
C_s	charging and discharging in non-terminal states
C_{sc0}	charging in terminal end-of-horizon base states
C_{sd0}	discharging in terminal end-of-horizon base states
$C_{\text{sc}k}$	charging in terminal contingency states
$C_{\text{sd}k}$	discharging in terminal contingency states

This leads to the current design based on the five price model summarized in Table 4-1. Expressing (4.95) in terms of (4.91) and (4.92), splitting up the terms and applying different prices to the five different types of contributions to the expected terminal storage quantities, yields the following.

---

[23]It is not the amount of energy stored that is of interest, but rather the amount which can be recovered after output efficiency losses η_{out}^{ti}.

$$v_S(x) = C_s^{\mathsf{T}}(A_1 s_0 + A_2 x + A_3 x) + C_{\mathrm{sc}0}^{\mathsf{T}} A_4 x + C_{\mathrm{sd}0}^{\mathsf{T}} A_5 x$$
$$+ C_{\mathrm{sc}k}^{\mathsf{T}} A_6 x + C_{\mathrm{sd}k}^{\mathsf{T}} A_7 x \qquad (4.96)$$

where

$$A_1 = [\eta_{\mathrm{out}}^{n_t}][\beta_1^{n_t}] \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{L}_I^{n_t j} + \sum_{t \in T} [\eta_{\mathrm{out}}^t][\beta_5^t] \sum_{j \in J^t} \left(\sum_{k \in K^{tj} \neq 0} \psi^{tjk} \right) \bar{L}_I^{tj} \qquad (4.97)$$

$$A_2 = [\eta_{\mathrm{out}}^{n_t}][\beta_1^{n_t}] \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{M}_g^{n_t j} + \sum_{t \in T} [\eta_{\mathrm{out}}^t] \sum_{j \in J^t} \left(\sum_{k \in K^{tj} \neq 0} \psi^{tjk} \right) \left([\beta_5^t] \bar{M}_g^{tj} + [\beta_4^t] \bar{G}_0^{tj} \right)$$
$$(4.98)$$

$$A_3 = [\eta_{\mathrm{out}}^{n_t}][\beta_1^{n_t}] \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{M}_h^{n_t j} + \sum_{t \in T} [\eta_{\mathrm{out}}^t] \sum_{j \in J^t} \left(\sum_{k \in K^{tj} \neq 0} \psi^{tjk} \right) \left([\beta_5^t] \bar{M}_h^{tj} + [\beta_4^t] \bar{H}_0^{tj} \right)$$
$$(4.99)$$

$$A_4 = [\eta_{\mathrm{out}}^{n_t}][\beta_2^{n_t}] \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{G}_0^{n_t j} \qquad (4.100)$$

$$A_5 = [\eta_{\mathrm{out}}^{n_t}][\beta_2^{n_t}] \sum_{j \in J^{n_t}} \psi^{n_t j 0} \bar{H}_0^{n_t j} \qquad (4.101)$$

$$A_6 = \sum_{t \in T} [\eta_{\mathrm{out}}^t][\beta_3^t] \sum_{j \in J^t} \sum_{k \in K^{tj} \neq 0} \psi^{tjk} \bar{G}_k^{tj} \qquad (4.102)$$

$$A_7 = \sum_{t \in T} [\eta_{\mathrm{out}}^t][\beta_3^t] \sum_{j \in J^t} \sum_{k \in K^{tj} \neq 0} \psi^{tjk} \bar{H}_k^{tj} \qquad (4.103)$$

If we use $\bar{A}_n$ to represent the version of A_n with all columns removed except for those corresponding to the relevant charging and discharging injections (p_{sc} for $n = 2, 4, 6$ and p_{sd} for $n = 3, 5, 7$), then we can express the cost of initial and terminal stored energy f_s from (4.12) as

$$f_s(s_0, p_{\mathrm{sc}}, p_{\mathrm{sd}}) = C_{s0}^{\mathsf{T}} s_0 - v_S(x)$$
$$= C_{s0}^{\mathsf{T}} s_0 - (C_{\mathrm{ts}0}^{\mathsf{T}} s_0 + C_{\mathrm{tsc}}^{\mathsf{T}} p_{\mathrm{sc}} + C_{\mathrm{tsd}}^{\mathsf{T}} p_{\mathrm{sd}}) \qquad (4.104)$$

where

$$C_{\text{ts0}} = A_1^\mathsf{T} C_s \tag{4.105}$$

$$C_{\text{tsc}} = \bar{A}_2^\mathsf{T} C_s + \bar{A}_4^\mathsf{T} C_{\text{sc0}} + \bar{A}_6^\mathsf{T} C_{\text{sc}k} \tag{4.106}$$

$$C_{\text{tsd}} = \bar{A}_3^\mathsf{T} C_s + \bar{A}_5^\mathsf{T} C_{\text{sd0}} + \bar{A}_7^\mathsf{T} C_{\text{sd}k}. \tag{4.107}$$

5 `most`

In MATPOWER, a MOST optimization problem is executed by calling `most` with a *MOST Data struct* as the first argument (`mdi`). The results are returned in an updated MOST Data struct (`mdo`). An additional optional input argument can be used to set options (`mpopt`).

The following sections describe the input data, the MOST options, the MOST Data struct itself, including the results, and some additional considerations.

5.1 Input Data

The MOST Data struct, containing all of the data needed for the problem, is sufficiently complex that it is not typically created directly, but rather is assembled from numerous other files or data structures by the `loadmd` function as illustrated in Figure 5-1.

```
mdi = loadmd(mpc, transmat, xgd, sd, contab, profiles);
```

The following sections describe the input arguments to `loadmd` and the way they are normally constructed. Except for the transition probability matrices, all parameters which vary from period to period are specified via `profiles`, as per-period changes applied to a set of base values provided in the other input arguments.

Since the input arguments to `loadmd` are handled by `loadgenericdata` (see Section 6.8 for details), they can either take the form of the data structure described in each section below, or a string containing the name of an M-file or MAT-file that returns the required data structure.

5.1.1 `mpc` – MATPOWER **Case**

The `mpc` argument is a MATPOWER case,[24] specified either as a file name or a struct, to be passed by `loadmd` as an input to `loadcase` from Section 9.1.1 in the MATPOWER User's Manual. This case corresponds to the base case from which all other cases, corresponding to different time periods, scenarios and contingency states, are built using change tables and the `apply_changes` function described in Section 9.3.5 in the MATPOWER User's Manual.

---

[24]It is important that this case have consecutively numbered buses starting at 1 (i.e. internal ordering). See MATPOWER's `ext2int` function in Section 9.4.1 of the MATPOWER User's Manual for converting a case to internal ordering.
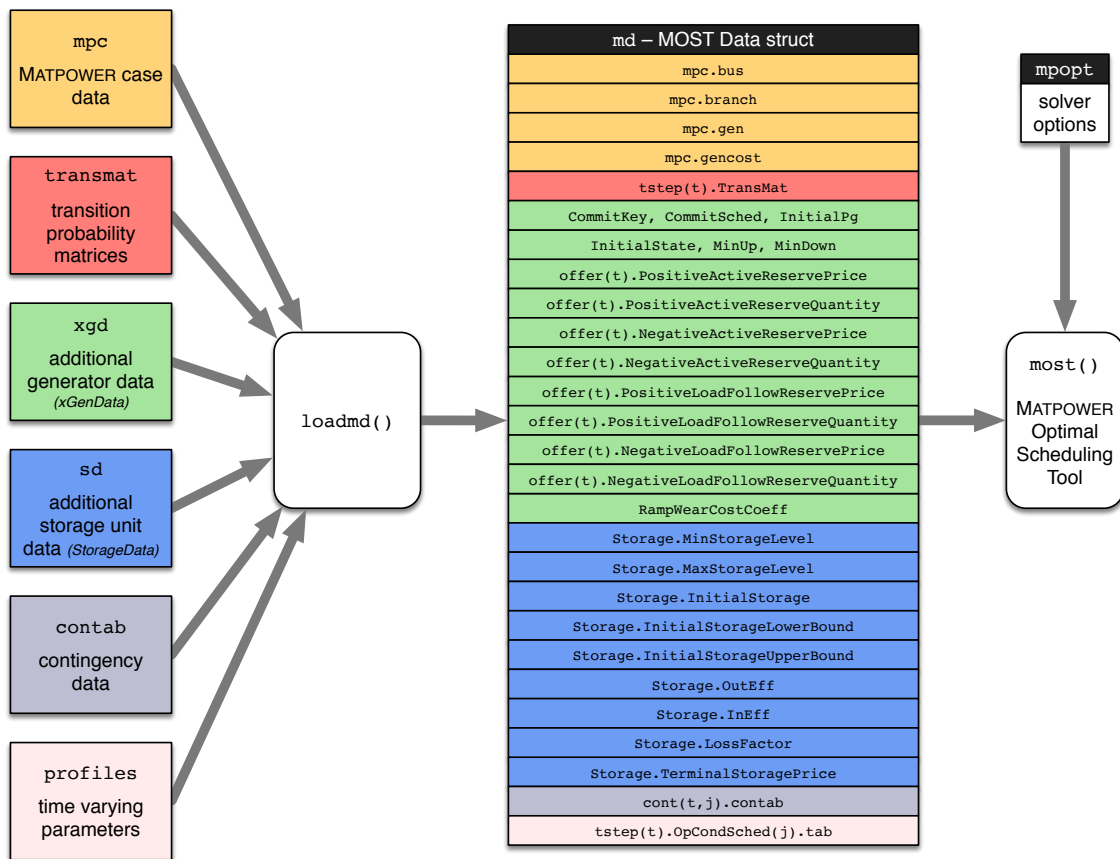
Figure 5-1: Assembling the MOST Data struct

If the problem you are setting up includes storage resources or wind generators, it may be simpler to exclude these from the `gen` matrix in `mpc` and add them later using the `addstorage` and `addwind` functions described below in Sections 6.2 and 6.3.

5.1.2 `transmat` – Transition Probability Matrices

In the general case of a stochastic model with multiple base scenarios per period, `transmat` is a cell array of length n_t containing the transition probability matrices Φ^t of (4.55). That is, `transmat{t}` contains the $n_{J^t} \times n_{J(t-1)}$ matrix of transition probabilities from period $t-1$ to period t. The first element `transmat{1}` is a column vector of transition probabilities from period 0 ($n_{J^t} = 1$) to period 1.

For deterministic models or secure models with contingencies but only a single base scenario per period, `transmat` can simply be an integer n_t specifying the number of periods in the planning horizon and it will be expanded internally to a cell array of 1's with the appropriate length. If the problem is also single period, `transmat` is optional, with a default value of 1.

5.1.3 `xgd` – Extra Generator Data (`xGenData`)

The optional `xgd` argument is an `xGenData` struct containing base values for all of the per-generator data required for the problem that are not included in `mpc`, that is, in the standard MATPOWER case data. This includes unit commitment (UC) data, reserve offer data, ramping costs, and more. Table 5-1 presents the details of the `xGenData` struct, whose fields are all $n_g \times 1$ vectors of per generator values.

An `xGenData` struct is typically created from a data file or struct in the `xGenDataTable` format via the `loadxgendata` function described in Section 6.11. In this context, all fields are optional and `loadxgendata` will provide defaults for anything not explicitly specified. The `addstorage` and `addwind` functions described in Sections 6.2 and 6.3 also return `xGenData` by calling the `loadxgendata` function internally.

Table 5-1: Fields[*] of `xGenData` struct (`xgd`)

name	default[†]	description
`CommitSched`	C	0 or 1, UC status to use for non-UC runs
`InitialPg`	P	active power dispatch at time $t = 0$
`TerminalPg`		active power dispatch at time $t = n_t + 1$ *(only included if explicitly specified)*
`RampWearCostCoeff`	0	quadratic coefficient for $C_\delta^i(\cdot)^\ddagger$ in (4.10)
`PositiveActiveReservePrice`	0	linear coefficient for $C_{R+}^{ti}(\cdot)^\ddagger$ in (4.9)
`PositiveActiveReserveQuantity`	R	max upward reserve quantity $R_{\max+}^{ti}$ in (4.19)
`NegativeActiveReservePrice`	0	linear coefficient for $C_{R-}^{ti}(\cdot)^\ddagger$ in (4.9)
`NegativeActiveReserveQuantity`	R	max downward reserve quantity $R_{\max-}^{ti}$ in (4.20)
`PositiveActiveDeltaPrice`	0	linear coefficient for $C_{P+}^{ti}(\cdot)^\ddagger$ in (4.7)
`NegativeActiveDeltaPrice`	0	linear coefficient for $C_{P-}^{ti}(\cdot)^\ddagger$ in (4.7)
`PositiveLoadFollowReservePrice`	0	linear coefficient for $C_{\delta+}^{ti}(\cdot)^\ddagger$ in (4.11)
`PositiveLoadFollowReserveQuantity`	R	max upward ramp reserve $\delta_{\max+}^{ti}$ in (4.23)
`NegativeLoadFollowReservePrice`	0	linear coefficient for $C_{\delta-}^{ti}(\cdot)^\ddagger$ in (4.11)
`NegativeLoadFollowReserveQuantity`	R	max downward ramp reserve $\delta_{\max-}^{ti}$ in (4.24)
`CommitKey`		required for problems with UC -1 – offline, unit forced *off* 0 or 1 – available for UC decisions 2 – must run, unit forced *on*
`InitialState`[§]	$\pm\infty$[¶]	if positive (negative), number of uptime (downtime) periods at time $t = 0$
`MinUp`[§]	1	minimum up time in number of periods
`MinDown`[§]	1	minimum down time in number of periods

[*] All fields are $n_g \times 1$ vectors of per-generator values.

[†] These are defaults provided by `loadxgendata`. If `gen` is provided, either directly or as the `gen` field of `mpc`, then P = gen(:, PG), C = gen(:, GEN_STATUS) and R = 2*(gen(:, PMAX) - MIN(0, gen(:, PMIN))), otherwise C = 1, R = 0 and no default is provided for P (corresponding field is not optional).

[‡] Each of these costs $C(\cdot)$ is presented in the formulation as a general function, but is implmented as a simple linear function of the form $C(x) = ax$, where the linear coefficient being supplied is a. The only exception is the ramping cost, which has the quadratic form $C_\delta^i(x) = ax^2$.

[§] Requires that `CommitKey` be present and non-empty.

[¶] Sign is based on C[†], i.e. $+\infty$ for C = 1, $-\infty$ for C = 0.

5.1.4   `sd` – **Storage Data (`StorageData`)**

The optional `sd` argument is a `StorageData` struct containing base values for all of the storage unit data required for the problem that are not included in the standard MATPOWER case `mpc` or in the `xGenData`.

    This includes bounds on stored energy (capacities), efficiencies, loss factors, initial and terminal values, prices used to value leftover storage, and more. Table 5-2 presents the details of the `StorageData` struct, whose fields are all $n_s \times 1$ vectors of per storage unit values, except where indicated otherwise.

    A `StorageData` struct is typically created from data files or structs in `StorageDataTable` format via the `loadstoragedata` function described in Section 6.10. The `addstorage` function from Section 6.2 also returns `StorageData` by calling the `loadstoragedata` function internally.

5.1.5   `contab` – **Contingency Table**

The optional `contab` argument is a contingency table with a master set of contingencies used for security throughout entire horizon. It is a matrix in the form of a *change table* recognized by `apply_changes`, described in Section 9.3.5 in the MATPOWER User's Manual. The probabilities defined in this contingency table correspond to the conditional probabilities ψ_0^{tjk} of contingency k occuring conditioned on being in base scenario j. While the MOST Data struct (`md`) itself allows for contingencies to be defined independently for all scenarios and time periods, `loadmd` applies a single set of contingencies and conditional probabilities (single `contab`) to all.

Table 5-2: Fields[*] of `StorageData` struct (`sd`)

name	default	description
`UnitIdx`	*none*	corresponding row index into `gen` matrix
`ExpectedTerminalStorageAim`	*none*[†]	target value for expected final stored energy at end of last period, overrides any values provided for both `ExpectedTerminalStorageMin` and `ExpectedTerminalStorageMax`
`ExpectedTerminalStorageMax`	*none*[†]	upper bound $S_{\max}^{n_t i}$ on expected final stored energy at end of last period in (4.37)
`ExpectedTerminalStorageMin`	*none*[†]	lower bound $S_{\min}^{n_t i}$ on expected final stored energy at end of last period in (4.37)
`InitialStorage`	*none*	value for initial (expected) stored energy s_0
`InitialStorageCost`	*none*[‡]	cost C_{s0} associated with starting with amount s_0 at time $t = 0$
`InitialStorageLowerBound`	*none*[‡]	lower bound $S_{\min}^{0i}$ on inital (expected) stored energy s_0 in (4.39)
`InitialStorageUpperBound`	*none*[‡]	upper bound $S_{\max}^{0i}$ on inital (expected) stored energy s_0 in (4.39)
`InEff`[¶]	1	input efficiency η_{in}^{ti}
`OutEff`[¶]	1	output efficiency η_{out}^{ti}
`LossFactor`[¶]	0	fraction of stored energy lost per hour η_{loss}^{ti}
`MaxStorageLevel`[¶]	*none*	stored energy maximum limit $S_{\max}^{ti}$
`MinStorageLevel`[¶]	*none*	stored energy minimum limit $S_{\min}^{ti}$
`rho`[¶]	1	ρ^{ti} parameter controlling weighting of worst case ($\rho^{ti} = 1$) and expected values ($\rho^{ti} = 0$) for defining storage constraints in (4.33)–(4.36)
`TerminalStoragePrice`		prices C_s for contributions to terminal storage from charging/discharging in non-terminal states
`TerminalChargingPrice0`	C_s[§]	prices $C_{\text{sc}0}$ for contributions to terminal storage from charging in terminal end-of-horizon base states
`TerminalDischargingPrice0`	C_s[§]	prices $C_{\text{sd}0}$ for contributions to terminal storage from discharging in terminal end-of-horizon base states
`TerminalChargingPriceK`	C_s[§]	prices $C_{\text{sc}k}$ for contributions to terminal storage from charging in terminal contingency states
`TerminalDischargingPriceK`	C_s[§]	prices $C_{\text{sd}k}$ for contributions to terminal storage from discharging in terminal contingency states

[*] All fields are $n_s \times 1$ vectors of per storage unit values, except where indicated otherwise. If no default is specified, it means the field is required.

[†] If the `most.storage.terminal_target` option is set to 0, the `ExpectedTerminalStorage*` parameters are optional and ignored; if set to 1, at least one of them is required; if set to -1, the presence of any of the optional `ExpectedTerminalStorage*` parameters will turn the `most.storage.terminal_target` option on.

[‡] If the `most.storage.cyclic` option is set to 1, `InitialStorageCost` is required and the default values of the initial storage bounds `InitialStorageLowerBound` and `InitialStorageUpperBound` are taken from `MinStorageLevel(:, 1)` and `MaxStorageLevel(:, 1)`, respectively, otherwise `InitialStorageCost` is optional and both initial storage bounds default to `InitialStorage`.

[¶] Can also be a scalar, in which case the value will be used for all storage units.

[§] That is, the default is taken from `TerminalStoragePrice`.

5.1.6 `profiles` – Profiles for Time-Varying Parameters

Profiles are used to specify how parameters vary from period to period and are defined in terms of changes to a base value. There are currently three different types of data that can be changed by a profile, corresponding to the base values provided by the `mpc`, `xgd` and `sd` arguments to `loadmd`.

A profile is a struct that specifies, for a given set of changes to be applied across time periods and scenarios, the type of data, the table or field, the elements in that table or field, the method of modification and the values to be applied. Table 5-3 summarizes the structure of the `profile` struct.

Table 5-3: Fields of Profile Struct (`profile`)

name	description
type	string defining type of target data structure modified by the profile
	`'mpcData'` – parameters in fields of `mpc`, e.g. `bus`, `gen`, `gencost`, etc.[*]
	`'xGenData'` – parameters in fields of `xGenData`, e.g. reserve offers, commitment parameters, etc.
	`'StorageData'` – parameters in fields of `StorageData`, e.g. storage capacities, efficiencies, etc.
table	numeric scalar or string identifying the table or field; valid values depend on `type`
	`type = 'mpcData'` – numeric values defined in Table 9-2 in the MATPOWER User's Manual
	`type = 'xGenData'` – string-valued field names defined in Table 5-1
	`type = 'StorageData'` – string-valued field names defined in Table 5-2
rows	vector of indices of rows (first dimension) of array to be modified, or area indices for area-wide modifications; 0 means "apply to all"
col	column index or other ID of parameter to modify as defined in Table 9-4 in the MATPOWER User's Manual; ignored unless `type = 'mpcData'`
chgtype	method of modification, see Table 9-3 in the MATPOWER User's Manual
values	$n_1 \times n_2 \times n_3$ numeric array containing the new values or scale/shift factors
	n_1 – corresponds to number of time periods n_t
	n_2 – corresponds to number of scenarios $n_j^{\max}$
	n_3 – corresponds to number of elements indicated by `rows`
	A singleton dimension in `values` not matching with $n_t = 1$ or $n_j^{\max} = 1$ or `length(profile.rows) = 1` is interpreted as "apply to all" whenever the parameter being modified allows such an expansion.

[*] Possible modifications are as implemented by MATPOWER's `apply_changes` function. See Section 9.3.5 in the MATPOWER User's Manual.

Section 6.7 describes `idx_profile` which defines a number of constants that are useful for specifying profiles. The `apply_profile` function from Section 6.4 is used internally by `loadmd` to apply the profiles. And `getprofiles` from Section 6.6 is used to load a profile or set of profiles from a struct, MAT-file or M-file function.

5.2 MOST Options

In addition to making use of the `verbose` option and the solver-specific options, such as those under the fields `cplex`, `glpk`, `gurobi`, etc. as documented in Appendix C in the MATPOWER User's Manual, there are also a number of options specific to MOST that appear under a `most` field in the standard MATPOWER options struct. These can be classified into two main categories and are described in Tables 5-4 and 5-5. The first consists of options related to how `most` is run, such as the solver to use and the phases of the problem building and solving to be included. The second is all of the options controlling various details of the model to be built and solved.

Table 5-4: MOST Run Options

name	default	description
`most.solver`	`'DEFAULT'`	solver option passed in `opt.alg` to `miqps_matpower` or `qps_matpower`, see `miqps_matpower` or `qps_matpower` for details
`most.skip_prices`	0	skip price computation stage for mixed integer problems, see `miqps_matpower` for details
`most.price_stage_warn_tol`	10^{-7}	tolerance on the objective function value and primal variable relative match required to avoid mis-match warning message, see `miqps_matpower` for details
`most.build_model`	1	toggle building of MIQP model 0 – do *not* build MIQP, assume it has already been built 1 – build MIQP, both constraints and standard costs (not coordination cost[†]) and store in `mdo.QP`
`most.solve_model`	1	toggle solving of MIQP model 0 – do *not* solve MIQP, assume it is just being built; requires `'most.build_model'` set to 1 1 – solve MIQP; if coordination cost[†] exists, update it; requires either `'most.build_model'` set to 1 or `mdi.QP` must contain previously built model
`most.resolve_new_cost`	0	toggle solving pre-built MIQP with updated coordination cost[‡] 0 – build full MIQP from scratch, then solve 1 – solve pre-built MIQP with updated coordination cost[†]

[†] Coordination costs are related to a price coordination scheme employed in a decomposition used for solving the non-linear AC formulation *(not yet implemented)*.

[‡] Requires `'most.solve_model'` set to 1.

Table 5-5: MOST Model Options

name	default	description
most.dc_model	1	power balance model 0 – use simple total power balance constraint $\sum \text{generation} = \sum \text{demand}$ 1 – use DC power flow network model
most.q_coordination	0	create q^{tijk} variables for reactive power coordination (0 or 1)
most.fixed_res	-1	include fixed zonal reserve constraints (4.8), (4.16)–(4.18) for security -1 – if fixed zonal reserves specified 0 – never 1 – always
most.security_constraints	-1	include contingency constraints for security -1 – if contingencies specified 0 – never 1 – always
most.storage.terminal_target	-1	constrain expected terminal storage to a target range[†] -1 – if target range specified 0 – never 1 – always
most.storage.cyclic	0	use cyclic storage constraints (4.38)[†] 0 – *off*, initial storage s_0^i is a fixed parameter, no constraint on final expected storage $s_F^{n_t i}$ 1 – *on*, initial storage s_0^i is an optimization variable constrained to equal final expected storage $s_F^{n_t i}$
most.uc.run	-1	flag to indicate whether to perform unit commitment -1 – perform unit commitment if and only if mdi.UC.CommitKey is present and non-empty 0 – do *not* perform unit commitment 1 – *do* perform unit commitment
most.uc.cyclic	0	commitment restrictions (e.g. min up and down times) roll over from end of horizon back to beginning (0 or 1)
most.alpha	0	α, for contingency states, fraction of period spent in base state before contingency occurs (0–1)

[†] The most.storage.terminal_target and most.storage.cyclic options cannot be used simultaneously (i.e. at least one of them must be set to 0).

5.3 MOST Data struct

5.3.1 Input Data

The input to the `most` function takes the form of a MOST Data struct, a single MAT-LAB struct `md`, with the primary input fields described in Tables 5-6 through 5-9. As described previously in Section 5.1, a MOST Data struct is typically not constructed directly, but rather assembled from various other inputs by the `loadmd` function. However, some of the features, such as fixed zonal reserve requirements, binary transition masks for load-following ramp, or linear dynamical system constraints, are only available by modifying portions of the MOST Data struct directly.

Table 5-6: Input Data Fields of `md`

name	type[*]	default	description
`cont(t,j).contab`	I	*empty*	changes table[†] defining contingencies for period t, scenario j
`Delta_T`	I	1	length of time step in hours
`idx.nt`	I		number of periods in scheduling horizon
`InitialPg(i)`	I		$n_g \times 1$, injection of generator i at $t = 0$
`mpc`	I		base system data, standard MATPOWER case struct[‡] with `baseMVA`, `bus`, `gen`, `branch` and `gencost` fields
`offer(t)`	I		struct with offer data for period t, see Table 5-8 for details of sub-fields
`OpenEnded`	I	1	ignore terminal dispatch ramp constraints, *deprecated*
`RampWearCostCoeff(i,t)`	I	0	$n_g \times n_t$, cost of ramping of generator i from period $t-1$ to t, coefficient C_δ^i for square of dispatch difference in (4.10)
`Storage`	B		struct with parameters for storage units, see Table 5-9 for the input fields
`TerminalPg(i)`	I		$n_g \times 1$, injection of generator i at $t = n_t$, *deprecated, untested*
`tstep(t)`	B		$n_t \times 1$ struct of parameters related to period t
`.OpCondSched(j).tab`	I		changes table defining modifications from `mpc` for each base scenario j in period t
`.TransMask`	I		an $n_{j(t)} \times n_{j(t-1)}$ matrix of binary transition masks $\zeta^{tj_2j_1}$ from scenario j_1 in period $t-1$ to j_2 in period t, see Section 6.5
`.TransMat`	I		Φ^t, an $n_{j(t)} \times n_{j(t-1)}$ matrix of transition probabilities $\phi^{tj_2j_1}$ from scenario j_1 in period $t-1$ to j_2 in period t, see Section 5.1.2

[*] I = input, O = output, B = both, opt = taken from MATPOWER options.
[†] See Section 5.1.5 for details. Note that, while `loadmd` assigns the same `contab` to all t and j, it is possible to set different `contab` values manually and they will be respected by `most`.
[‡] See Appendix B in the MATPOWER User's Manual for details.

61

Table 5-7: Additional Input Data Fields of `md`

name	type[*]	default	description
CoordCost	I	*empty*	user supplied coordination costs for AC version
.Huser	I		sparse matrix of quadratic coefficients
.Cuser	I		vector of linear coefficients
.cuser	I		scalar constant term
dstep(t)	I	*empty*	$n_t^{\mathrm{ds}} \times 1$ struct with parameters for optional dynamical system model, (4.49)–(4.53)
.A	I		A_{ds}^t, from (4.50)–(4.51)
.B	I		B_{ds}^t, from (4.50)
.C	I		C_{ds}^t, from (4.52)–(4.53)
.D	I		D_{ds}^t, from (4.52)
.ymin	I		$y_{\min}^t$, lower bound on output from (4.52)–(4.53)
.ymax	I		$y_{\max}^t$, upper bound on output from (4.52)–(4.53)
.zmin	I		$z_{\min}^t$, lower bound on z^t from (4.49)
.zmax	I		$z_{\max}^t$, upper bound on z^t from (4.49)
FixedReserves(t,j,k)	I	*empty*	zonal reserve input parameters for period t, scenario j, contingency k, in form of `reserves` field of `mpc` from Table 7-1 in the Matpower User's Manual, with `cost`, `qty`, `zones`, `req` sub-fields.
UC	B		struct with unit commitment parameters
.CommitKey(i,t)	I	*empty*	optional $n_g \times n_t$ vector specifying availability of unit i for commitment at time t -1 – offline, unit forced *off* 0 or 1 – available for UC decisions 2 – must run, unit forced *on*
.CommitSched(i,t)	B		$n_g \times n_t$ matrix UC status (0 or 1) of unit i at time t, input for non-UC runs, result for UC runs
.InitialState(i)†	I	*empty*	$n_g \times 1$ vector of initial states, if positive (negative), number of uptime (downtime) periods at time t = 0
.MinUp(i)†	I	*empty*	$n_g \times 1$ vector, minimum up time in number of periods
.MinDown(i)†	I	*empty*	$n_g \times 1$ vector, minimum down time in number of periods
z1	I	*empty*	initial state z^1 of optional dynamical system model

[*] I = input, O = output, B = both, opt = taken from Matpower options.
† Requires that `CommitKey` be present and non-empty.

62

Table 5-8: Fields of Offer struct `md.offer(t)`

name	description
`PositiveActiveReservePrice`[†]	linear coefficient of $C_{RP+}^{ti}(\cdot)$ in (4.6)
`PositiveActiveReserveQuantity`[†]	max upward reserve quantity $R_{P\max+}^{ti}$ in (4.19)
`NegativeActiveReservePrice`[†]	linear coefficient of $C_{RP-}^{ti}(\cdot)$ in (4.6)
`NegativeActiveReserveQuantity`[†]	max downward reserve quantity $R_{P\max-}^{ti}$ in (4.20)
`PositiveActiveDeltaPrice`[†]	linear coefficient of $C_{P+}^{ti}(\cdot)$ in (4.6)
`NegativeActiveDeltaPrice`[†]	linear coefficient of $C_{P-}^{ti}(\cdot)$ in (4.6)
`PositiveLoadFollowReservePrice`[†]	linear coefficient of $C_{\delta+}^{ti}(\cdot)$ in (4.6)
`PositiveLoadFollowReserveQuantity`[†]	max upward ramp reserve $\delta_{\max+}^{ti}$ in (4.23)
`NegativeLoadFollowReservePrice`[†]	linear coefficient of $C_{\delta-}^{ti}(\cdot)$ in (4.6)
`NegativeLoadFollowReserveQuantity`[†]	max downward ramp reserve $\delta_{\max-}^{ti}$ in (4.24)
`gencost`[‡]	energy offers in the form of generator cost functions

[†] $n_g \times 1$ vector of values for each generator at time t.
[‡] Deprecated. Use profiles instead.

Table 5-9: Input Fields of `md.Storage`

name	type[*]	default	description[†]
`UnitIdx(i)`	I		corresponding `gen` matrix row index
`ExpectedTerminalStorageAim(i)`	I		target value for expected final stored energy at end of last period for storage unit i, overrides any values provided for both `ExpectedTerminalStorageMin` and `ExpectedTerminalStorageMax`
`ExpectedTerminalStorageMax(i)`	I		upper bound $S_{\min}^{n_t i}$ on expected final stored energy in (4.37)
`ExpectedTerminalStorageMin(i)`	I		lower bound $S_{\max}^{n_t i}$ on expected final stored energy in (4.37)
`InitialStorage(i)`	B		initial (expected) stored energy s_0 in MWh
`InitialStorageCost(i)`	I		cost C_{s0} associated with starting with amount s_0 at time $t = 0$
`InitialStorageLowerBound(i)`	I		lower bound $S_{\min}^{0i}$ on inital (expected) stored energy s_0 in (4.39)
`InitialStorageUpperBound(i)`	I		upper bound $S_{\max}^{0i}$ on inital (expected) stored energy s_0 in (4.39)
`InEff(i,t)`[‡]	I	1	input efficiency η_{in}^{ti}
`OutEff(i,t)`[‡]	I	1	output efficiency η_{out}^{ti}
`LossFactor(i,t)`[‡]	I	0	fraction of stored energy lost per hour η_{loss}^{ti}
`MaxStorageLevel(i,t)`[‡]	I		stored energy maximum limit $S_{\max}^{ti}$
`MinStorageLevel(i,t)`[‡]	I		stored energy minimum limit $S_{\min}^{ti}$
`rho(i,t)`	I		ρ^{ti} parameter controlling weighting of worst case ($\rho^{ti} = 1$) and expected values ($\rho^{ti} = 0$) for defining storage constraints in (4.33)–(4.36)
`TerminalStoragePrice(i)`	I		prices C_s for contributions to terminal storage from charging/discharging in non-terminal states
`TerminalChargingPrice0(i)`	I	C_s[§]	prices C_{sc0} for . . . charging in terminal end-of-horizon base states
`TerminalChargingPriceK(i)`	I	C_s[§]	prices $C_{\text{sc}k}$ for . . . charging in terminal contingency states
`TerminalDischargingPrice0(i)`	I	C_s[§]	prices C_{sd0} for . . . discharging in terminal end-of-horizon base states
`TerminalDischargingPriceK(i)`	I	C_s[§]	prices $C_{\text{sd}k}$ for . . . discharging in terminal contingency states

[*] I = input, O = output, B = both, opt = taken from MATPOWER options.
[†] All fields have n_s rows, where row i refers to storage unit i. See also Table 5-2.
[‡] Automatically expanded from scalar, $n_s \times 1$ or $1 \times n_t$ vector to a full $n_s \times n_t$ matrix.
[§] That is, the default is taken from `TerminalStoragePrice`.

64

5.3.2 Output Data

Additional fields are initialized or added to the MOST Data struct by `most` and returned in the updated output struct. Some simply record the values of corresponding options found in the MATPOWER options struct passed in, while others contain computed results. The output fields added or updated by `most` are summarized in Tables 5-10 through 5-13.

Table 5-10: Output Data Fields of `md`

name	type[*]	description
`alpha`	opt	α, copy of `most.alpha` option
`CostWeights(k,j,t)`[†]	O	ψ^{tjk}, probability of contingency k in scenario j at time t
`CostWeightsAdj(k,j,t)`[†]	O	ψ_α^{tjk}, same as ψ^{tjk}, but adjusted for α as in (4.1)
`DCMODEL`	opt	copy of `most.dc_model` option
`flow(t,j,k)`	O	case data for period t, scenario j, contingency k
`.mpc`		MATPOWER case struct,[‡] prices and gen costs are probability-weighted
`.PLsh`		vector needed to compute branch flow results
`idx`	B	various problem dimensions, see Table 5-11
`IncludeFixedReserves`	opt	copy of `most.fixed_res` option
`QCoordination`	opt	copy of `most.q_coordination` option
`QP`	B[¶]	(MI)QP/LP problem setup and results, see Table 5-12
`results`	O	results, see Table 5-13
`SecurityConstrained`	opt	copy of `most.security_constraints` option
`StepProb(t)`	O	γ^t, probability of making it to period t
`Storage`	B	γ^t, probability of making it to period t
`.ExpectedStorageDispatch(i,t)`	O	$n_s \times n_t$, expected dispatch of storage unit i
`.ExpectedStorageState(i,t)`	O	$n_s \times n_t$, expected stored energy in storage unit i at end period t
`.ForceCyclicStorage`	opt	copy of `most.storage.cyclic` option
`.ForceExpectedTerminalStorage`	opt	copy of `most.storage.terminal_target` option
`.InitialStorage(i)`	B	$n_s \times 1$, initial (expected) stored energy s_0 in MWh, computed as output when `most.storage.cyclic` option is on
`tstep(t)`	B	$n_t \times 1$ struct of parameters related to period t
`.E`[§]	O	E^t, used to compute expected injections in period t
`.G`[§]	O	G^t
`.H`[§]	O	H^t
`.Li`[§]	O	L_I^t
`.Lf`[§]	O	L_F^t
`.Mg`[§]	O	M_g^t
`.Mh`[§]	O	M_h^t
`.Ng`[§]	O	N_g^t
`.Nh`[§]	O	N_h^t
`UC.CommitSched(i,t)`	B	$n_g \times n_t$ matrix UC status (0 or 1) of unit i at time t, input for non-UC runs, result for UC runs

[*] I = input, O = output, B = both, opt = taken from MATPOWER options.
[†] Note index order – `(:,:,t)` refers to period t.
[‡] See Appendix B in the MATPOWER User's Manual for details.
[¶] The `QP` field is either contructed by `most` or taken as an input, based on the value of the `most.build_model` option described in Table 5-4.
[§] Used to compute expected inital and final storage amounts for period t. See (4.83)–(4.90) for details.

Table 5-11: Fields of Index struct `md.idx`

name	type[*]	description
`nt`	I	number of periods in scheduling horizon
`nj(t)`	O	number of base scenarios for period t, computed from length of `tstep(t).OpCondSched(j)`
`nc(t,j)`	O	number of contingencies in period t, scenario j
`nb(t,j,k)`	O	number of buses in period t, scenario j, contingency k
`nb_total`	O	total number of buses summed over all flows
`ng`	O	number of gens in `mpc.gen`
`ny(t,j,k)`	O	number of gens with piecewise linear costs in period t, scenario j, contingency k
`nf_total`	O	total number of flows (periods $t \times$ scenarios $j \times$ contingencies k)
`ns`	O	number of storage units
`ns_total`	O	`ns` $\times$ `nf_total`
`ntramp`	O	number of periods of load-following reserves, always equal to `nt - 1` since `OpenEnded` has been deprecated
`ntds`	O	n_t^{ds}, number of time periods in the horizon of the dynamical system model
`nzds`	O	n_z^{ds}, size of state vector for dynamical system model (4.50)-(4.51)
`nyds`	O	n_y^{ds}, number of outputs of dynamical system model (4.52)-(4.53)
`nvars`	O	total number of variables

[*] I = input, O = output, B = both, opt = taken from MATPOWER options.
[†] Used to compute expected inital and final storage amounts for period t. See (4.83)–(4.90) for details.

67

Table 5-12: Fields of QP struct `md.QP`

name	type[*]	description
A[§]	B	linear constraint matrix
l[§]	B	linear constraint lower bound
u[§]	B	linear constraint upper bound
x[§]	O	full optimization variable x
f[§]	O	value of objective function at solution (same as `md.results.f`)
vtype[§]	B	string containing variable types
x0[§]	B	variable initial value
xmin[§]	B	variable lower bound
xmax[§]	B	variable upper bound
H[§]	B	quadratic cost coefficient matrix[†]
C[§]	B	linear cost coefficient vector[†]
c	B	constant cost term[†]
H1	B	quadratic cost coefficient matrix[‡]
C1	B	linear cost coefficient vector[‡]
c1	B	constant cost term[‡]
Cfstor	B	linear cost coefficients of full x to reflect expected value of storage in terminal states
opt[§]	B	options struct for `qps_matpower` or `miqps_matpower`, set by MOST run options and solver-specific MATPOWER options via `mpopt2qpopt`
exitflag[§]	O	1 = converged successfully, 0 or negative value = solver specific failure code
output[§]	O	struct with solver-specific fields and `alg` field specifying solver that was used
lambda[§]	O	Lagrange and Kuhn-Tucker multipliers on constraints
.mu_l[§]	O	lower (left-hand) limit on linear constraints
.mu_u[§]	O	upper (right-hand) limit on linear constraints
.lower[§]	O	lower bound on optimization variables
.upper[§]	O	upper bound on optimization variables

[*] I = input, O = output, B = both, opt = taken from MATPOWER options. The `QP` struct and its fields are either contructed by `most` or taken as an input, based on the value of the `most.build_model` option described in Table 5-4.
[†] Including user defined coordination costs from `md.CoordCost`.
[‡] Excluding user defined coordination costs from `md.CoordCost`.
[§] See input and output arguments for `qps_matpower` or `miqps_matpower` for details.

Table 5-13: Fields of Results struct `md.results`

name	type[*]	description
`f`	O	value of objective function $f(x)$ in (4.6) at solution (same as `md.QP.f`)
`success`	O	optimization success flag, 1 = succeeded, 0 = failed
`Pc(i,t)`	O	$n_g \times n_t$, active power contract quantity, p_c^{ti}
`Rpp(i,t)`	O	$n_g \times n_t$, upward active contingency reserve quantity r_+^{ti}
`Rpm(i,t)`	O	$n_g \times n_t$, downward active contingency reserve quantity r_-^{ti}
`Rrp(i,t)`	O	$n_g \times n_t$, upward load-following ramping reserve quantity δ_+^{ti}
`Rrm(i,t)`	O	$n_g \times n_t$, downward load-following ramping reserve quantity δ_-^{ti}
`Sp`	O	$n_s \times n_t$, endogenously computed upper stored energy bounds s_+^{ti}
`Sm`	O	$n_s \times n_t$, endogenously computed lower stored energy bounds s_-^{ti}
`GenPrices(i,t)`	O	$n_g \times n_t$, expected energy price
`CondGenPrices(i,t)`	O	$n_g \times n_t$, expected energy price, conditional on making it to time t
`RppPrices(i,t)`	O	$n_g \times n_t$, price on upward active contingency reserve
`RpmPrices(i,t)`	O	$n_g \times n_t$, price on downward active contingency reserve
`RrpPrices(i,t)`	O	$n_g \times n_t$, price on upward load-following ramping reserve
`RrmPrices(i,t)`	O	$n_g \times n_t$, price on downward load-following ramping reserve
`ExpectedRampCost(i,t)`	O	$n_g \times n_t$, expected ramping cost (wear and tear)
`ExpectedDispatch(i,t)`	O	$n_g \times n_t$, expected generator dispatch across base cases
`Z`	O	$n_z^{\mathrm{ds}} \times n_t^{\mathrm{ds}}$, dynamical system model state z
`Y`	O	$n_y^{\mathrm{ds}} \times n_t^{\mathrm{ds}}$, dynamical system model output
`SetupTime`	O	time to construct model in seconds
`SolveTime`	O	time to solve model in seconds

[*] I = input, O = output, B = both, opt = taken from MATPOWER options.

5.4 Additional Considerations

The current version of MOST has a number of modeling limitations relative to Matpower. The following is a list of Matpower features that are not yet supported by MOST:

- branch angle difference limits

- branch flow soft limits

- DC transmission lines

- full non-linear AC network modeling

- interface flow limits

6 Additional Functions

6.1 `addgen2mpc`

```
[new_mpc, idx] = addgen2mpc(mpc, gen, gencost, gen_type)
```

Appends a set of generators to those contained in an existing MATPOWER case struct. The existing case in `mpc` is a standard MATPOWER case with additional fields, `genfuel` containing a cell array of string-valued fuel types (one for each generator), and `i<type>` containing (for each fuel type) a vector of indices of the generators of type `<type>`. The `gen` and `gencost` inputs are the standard MATPOWER generator and generator cost matrices and `gen_type` is a string or cell array of strings of fuel types for the generators to be appended.

It returns the updated case struct in `new_mpc`, with the new generators appended to the `gen`, `gencost` and `genfuel` fields and updated `i<type>` fields, along with a vector `idx` of indices of the newly added generators.

While there are no canonical definitions for generator types, Table 6-1 contains some strings that have been used by convention, where `'ess'` and `'wind'` are used explicitly by some functions in MOST.

Table 6-1: Typical Generator Types

type string	description
`'biomass'`	biomass
`'coal'`	coal
`'dl'`	dispatchable load
`'ess'`	energy storage system
`'hydro'`	hydro
`'ng'`	natural gas, combustion turbine
`'ngcc'`	natural gas, combined cycle
`'nuclear'`	nuclear
`'oil'`	oil
`'refuse'`	refuse
`'solar'`	solar PV
`'syncgen'`	synchronous condensor
`'wind'`	wind
`'na'`	*none*
`'unknown'`	*unknown*

6.2 `addstorage`

```
[idx, new_mpc] = addstorage(storage, mpc)
[idx, new_mpc, new_xgd, new_sd] = addstorage(storage, mpc)
[idx, new_mpc, new_xgd, new_sd] = addstorage(storage, mpc, xgd)
[idx, new_mpc, new_xgd, new_sd] = addstorage(storage, mpc, xgd, sd)
```

Given a `StorageUnitData` struct (`storage`), or the name of a function or MAT-file containing such a struct, this function adds the specified storage units, modeled as additional special generators, to the existing `gen` and `gencost` matrices of the MATPOWER case (`mpc`) and to the existing `xGen` and `StorageData` structs, providing a convenient way to specify in one place all of the parameters for a set of storage units.

The parameters for the storage units to be added are specified in a `StorageUnitData` struct, which is a single struct with the four fields described in Table 6-2. Return values include a vector `idx` of generator indices for the newly added storage units, along with updated versions of the `mpc`, `xgd` and `sd` structs specified by the inputs..

Table 6-2: Fields of `StorageUnitData` struct (`storage`)

name	default	description
gen	*none*	rows to be appended to the `gen` matrix[*] in `mpc`
gencost	*zero cost*	rows to be appended to the `gencost` matrix[*] in `mpc`
xgd_table	*none*	`xGenDataTable` struct[†] corresponding to units to be added
sd_table	*none*	`StorageDataTable` struct[‡] corresponding to units to be added

[*] See Tables B-2 and B-4 in Appendix B of the MATPOWER User's Manual for details on the format.
[†] See `loadxgendata` in Section 6.11 and Table 6-6 for details of the `xGenDataTable` struct.
[‡] See `loadstoragedata` in Section 6.10 and Table 6-5 for details of the `StorageDataTable` struct.

6.3 `addwind`

```
[idx, new_mpc] = addwind(wind, mpc)
[idx, new_mpc, new_xgd] = addwind(wind, mpc)
[idx, new_mpc, new_xgd] = addwind(wind, mpc, xgd)
```

Given a `WindUnitData` struct (`wind`), or the name of a function or MAT-file containing such a struct, this function adds the specified wind generators to the existing `gen` and `gencost` matrices of the MATPOWER case (`mpc`) and to the existing `xGen` struct, providing a convenient way to specify in one place all of the parameters for a set of wind generators.

The parameters for the wind generators to be added are specified in a `WindUnitData` struct, which is a single struct with the four fields described in Table 6-3. Return values include a vector `idx` of generator indices for the newly added wind generators, along with updated versions of the `mpc` and `xgd` structs specified by the inputs.

Table 6-3: Fields of `WindUnitData` struct (`wind`)

name	default	description
gen	*none*	rows to be appended to the `gen` matrix[*] in `mpc`
gencost	*zero cost*	rows to be appended to the `gencost` matrix[*] in `mpc`
xgd_table	*none*	`xGenDataTable` struct[†] corresponding to units to be added

[*] See Tables B-2 and B-4 in Appendix B of the MATPOWER User's Manual for details on the format.
[†] See `loadxgendata` in Section 6.11 and Table 6-6 for details of the `xGenDataTable` struct.

6.4   `apply_profile`

```
chgtabs = apply_profile( profile, chgtabsi )
xgd     = apply_profile( profile, xgdi, dim )
sd      = apply_profile( profile, sdi, dim )
```

The `apply_profile` function applies a single profile of a given type to the provided input. See Section 5.1.6 and Table 5-3 for details on the profile struct.

For profiles of type `'mpcData'`, the output is an $n_t \times n_j^{\max}$ cell array of *change tables* in the format expected by MATPOWER's `apply_changes` function.[25] The second input is also $n_t \times n_j^{\max}$ cell array. Each element can be either empty or contain a change table to which the new changes are appended.

For profiles of type `'xGenData'` the second argument is the `xGenData` struct to be modified (`xgdi`) and the output `xgd` is a modified version of the same struct. The the third argument `dim` is a positive integer indicating the number of elements corresponding to the third dimension of `profile.values`. This allows this dimension to be expanded to the appropriate size if it is specified as a singleton dimension in `profile.values`.

Profiles of type `'StorageData'` are completely analogous, taking a `StorageData` struct (`sdi`) as the second input and returning a modified version of it in `sd`.

6.5   `filter_ramp_transitions`

```
md = filter_ramp_transitions(md0, threshold)
```

---

[25]See Section 9.3.5 in the MATPOWER User's Manual.

The `filter_ramp_transitions` function creates a binary valued transition mask $\zeta^{tj_2j_1}$ for ramping reserves based on a given probability `threshold`. Only transitions with probabilities greater than or equal to a given `threshold` value are included, where the probability of the transition from state j_1 to j_2 is taken to be the conditional probability Φ^t from (4.55), specified in the `transmat` argument to `loadmd`, multiplied by the conditional probability of being in state j_1, given that you've made it to period t.

6.6 `getprofiles`

```
profiles = getprofiles(profilesi);
profiles = getprofiles(profilesi, profiles0);
profiles = getprofiles(profilesi, idx);
profiles = getprofiles(profilesi, profiles0, idx);
```

Loads a profile or set of profiles from a struct, MAT-file or M-file (`profilei`), optionally using the n-dimensional index vector `idx` to modify any non-zero values in the `rows` fields so that the corresponding `rows` field in the returned `profiles` is equal to `idx(rows)`. This makes it easy to use profiles defined for a particular set of generators, such as those added to a case as a group by `addwind` or `addstorage`.

The optional `profiles0` argument allows the user to provide an existing profile struct to which the new profiles are appended.

6.7 `idx_profile`

```
[PR_REP, PR_REL, PR_ADD, PR_TCONT, PR_TYPES, PR_TMPCD,...
    PR_TXGD, PR_TCTD, PR_TSTGD, PR_CHGTYPES] = idx_profile;
```

This function defines constants that are useful in defining profiles.

6.8 `loadgenericdata`

```
var = loadgenericdata(varfile, vartype)
var = loadgenericdata(varfile, vartype, fields)
var = loadgenericdata(varfile, vartype, fields, varname)
var = loadgenericdata(varfile, vartype, fields, varname, args)
```

The `loadgenericdata` function loads data from a variable, M-file or MAT-file and checks that it matches a specified type. The first argument, `varfile`, is a variable

Table 6-4: Constants Defined by `idx_profile`

name	value	description
PR_REP	1	replace old values with new ones
PR_REL	2	multiply old values by scale factors
PR_ADD	3	add shift factor to old values
PR_TCONT*	1	
PR_TYPES	*list*	list of profile types
PR_TMPCD	*list*	vector of valid table types for `'mpcData'`
PR_TXGD	*list*	list of valid table types for `'xGenData'`
PR_TCTD*	*list*	list of valid table types for `'ContingencyData'`
PR_TSTGD	*list*	list of valid table types for `'StorageData'`
PR_CHGTYPES	*list*	list of valid change types

* Related to functionality not yet implemented.

containing the data structure or a string containing the name of a function M-file or a MAT-file on the MATLAB path. If no file extension is provided, it will attempt to load a MAT-file with the specified name and, if not found, will call a function by that name to get the data. The function M-file should return a single argument containing the data. A MAT-file should either contain a single variable with the desired data or provide the variable name in `varname`.

The second argument, `vartype`, is a string or cell array of strings with, in order of priority, the data structure type to be returned. Valid values are `'struct'`, `'cell'` and `'array'`.

The third argument, `fields`, is optional and contains a string or cell array of strings containing a list of required fields in case the `vartype` is `'struct'`. If a required field is missing it will throw an error.

The `varname` and `args` arguments are also optional. `varname` is a string containing the name of the variable to extract when loading a MAT-file. If not provided, the default is to extract the first variable, regardless of name. And `args` is a scalar or cell array of values that are passed as input arguments to the function, in the case where `varfile` is a function name.

6.9   `loadmd`

```
md = loadmd(mpc, transmat, xgd, sd, contab, profiles)
```

The `loadmd` function provides the canonical way of loading a MOST Data struct. For details please see Sections 5.1.1–5.1.6.

6.10 `loadstoragedata`

```
sd = loadstoragedata(sd_table)
sd = loadstoragedata(sd_table, gen)
sd = loadstoragedata(sd_table, mpc)
```

The `loadstoragedata` function provides the canonical way of loading additional parameters for storage resources into a `StorageData` struct, described in Section 5.1.4 and summarized in Table 5-2. It takes a `StorageDataTable` struct as input, either directly or as the name of a function or MAT-file that returns such a struct. If the optional second argument is provided, either a MATPOWER `gen` matrix or a MATPOWER case file `mpc`, the number of storage units is checked for consistency.

The `StorageDataTable` struct is used as a convenient way to define the `StorageData` struct using a table format for the data and is summarized in Table 6-5. It has two mandatory fields, `colnames` and `data`. The `data` field is a $n_s \times N$ matrix, where n_s is the number of storage units and N is the number of fields in the `StorageData` being defined. The `colnames` field is an N dimensional cell array of strings with field names corresponding to the columns in `data`. The number of columns in the table and their order are determined by the user, depending on the fields for which they want to specify non-default values.

Table 6-5: Fields of `StorageDataTable` struct (`sd_table`)

name	default	description
`colnames`	*none*	N element cell array of `StorageData` field names[*] corresponding to the columns of the `data` field[†]
`data`	*none*	$n_s \times N$ matrix of storage parameters[†]
`MinStorageLevel`[‡]	*none*	stored energy minimum limit $S_{\min}^{ti}$
`MaxStorageLevel`[‡]	*none*	stored energy maximum limit $S_{\max}^{ti}$
`OutEff`[‡]	1	output efficiency η_{out}^{ti}
`InEff`[‡]	1	input efficiency η_{in}^{ti}
`LossFactor`[‡]	0	fraction of stored energy lost per hour η_{loss}^{ti}
`rho`[‡]	1	ρ^{ti} parameter controlling weighting of worst case ($\rho^{ti} = 1$) and expected values ($\rho^{ti} = 0$) for defining storage constraints in (4.33)–(4.36)

[*] See Table 5-2 for a list of valid field names.
[†] n_s is the number of storage units and N is the number of fields in the `StorageData` being defined by the given `StorageDataTable`.
[‡] Values in these scalar fields are overridden by any corresponding values in the `data` table.

There are six additional optional scalar fields that can be used instead of the `data` table if a single value is to be assigned uniformly to all of the storage units. An example `StorageDataTable` struct is created by the following code.

76

```
storage.sd_table.OutEff    = 0.9;
storage.sd_table.InEff     = 0.9;
storage.sd_table.LossFactor = 0.02;
storage.sd_table.rho       = 0;
storage.sd_table.colnames = {  % indented to align with data cols
    'InitialStorage', ...
        'InitialStorageLowerBound', ...
            'InitialStorageUpperBound', ...
                'InitialStorageCost', ...
                    'TerminalStoragePrice', ...
                        'MinStorageLevel', ...
                            'MaxStorageLevel', ...
};
storage.sd_table.data = [
    40  0   80  45  43  0   40;
    30  0   60  47  45  0   30;
    50  0   100 46  44  0   50;
];
```

See also the `addstorage` function in Section 6.2 for a potentially more convenient way to specify all of the parameters for your storage resources in a single file or struct.

6.11   loadxgendata

```
xgd = loadxgendata(xgd_table)
xgd = loadxgendata(xgd_table, gen)
xgd = loadxgendata(xgd_table, mpc)
```

The `loadxgendata` function provides the canonical way of loading extra generator data into an `xGenData` struct described in Section 5.1.3 and summarized Table 5-1. It takes an `xGenDataTable` struct as input, either directly or as the name of a function or MAT-file that returns such a struct. If the optional second argument is provided, either a MATPOWER `gen` matrix or a MATPOWER case file `mpc`, the generator status and limits are used to set certain default values as indicated in Table 5-1.

The `xGenDataTable` struct is used as a convenient way to define the `xGenData` struct using a table format for the data and is summarized in Table 6-6. It has two fields, `colnames` and `data`. The `data` field is a $n_g \times N$ matrix, where n_g is the number of generators and N is the number of fields in the `xGenData` being defined. Those that are not defined in the `xGenDataTable` struct are assigned default values

77

by `loadxgendata`. The `colnames` field is an N dimensional cell array of strings with field names corresponding to the columns in `data`. The number of columns in the table and their order are determined by the user, depending on the fields for which they want to specify non-default values.

Table 6-6: Fields of `xGenDataTable` struct (`xgd_table`)

name	default	description
colnames	*none*	N element cell array of `xGenData` field names[*] corresponding to the columns of the `data` field[†]
data	*none*	$n_g \times N$ matrix of extra generator parameters[†]

[*] See Table 5-1 for a list of valid field names.
[†] n_g is the number of generators and N is the number of fields in the `xGenData` being defined by the given `xGenDataTable`.

An example `xGenDataTable` struct is created by the following code.

```
xgd_table.colnames = {  % indented to align with data cols
    'CommitSched', ...
        'PositiveActiveReservePrice', ...
            'PositiveActiveReserveQuantity' };
xgd_table.data = [
    1   5    25;
    1   8    200;
    1   20   60;
    1   2    100;
];
```

6.12 `most_summary`

```
most_summary(mdo)
ms = most_summary(mdo)
```

This function should be considered experimental. It is included because it is often better than nothing, though it is very incomplete.

Given a MOST Data struct returned by `most`, this function returns a struct with the fields listed in Table 6-7. Printing to the console is currently controlled by the `mdo.QP.verbose` flag.

Table 6-7: Fields of `most_summary` struct (`ms`)

name	description
f	objective function value
nb	n_b, number of buses
ng	n_g, number of generators (incl. storage, disp. load, etc.)
nl	n_l, number of branches
nt	n_t, number of periods in planning horizon
nj_max	$n_j^{\max}$, max number of scenarios per period
nc_max	$n_c^{\max}$, max number of contingencies per scenario in any period
Pg	$n_g \times n_t \times n_j^{\max} \times (n_c^{\max} + 1)$, real power generation
Rup	$n_g \times n_t$, upward ramping reserve quantities
Rdn	$n_g \times n_t$, downward ramping reserve quantities
Pf	$n_g \times n_t \times n_j^{\max} \times (n_c^{\max} + 1)$, real power generation
u	$n_g \times n_t \times n_j^{\max} \times (n_c^{\max} + 1)$, generator commitment status
lamP	$n_b \times n_t \times n_j^{\max} \times (n_c^{\max} + 1)$, shadow price on power balance
muF	$n_l \times n_t \times n_j^{\max} \times (n_c^{\max} + 1)$, shadow price on flow limits

6.13 `mostver`

```
mostver
vnum = mostver
v = mostver('all')
```

Called with no output arguments, `mostver` prints the version number and release date of the current MOST installation. Otherwise, if called with no input arguments, it returns the current version as a string and with any true input argument, such as the string `'all'`, it returns a struct with the fields `Name`, `Version`, `Release` and `Date` (all strings).

7 Tutorial Examples

The examples in this section are based on the simple three bus model summarized in Table 7-1 and illustrated in Figure 7-1. The case data can be found in the `ex_case3a` and `ex_case3b` files. Not all examples include every part of the model. For example, the single-period deterministic examples do not have the wind generator at bus 2, none of the deterministic cases include the contingencies and the stochastic cases do not include the fixed reserve requirement. The storage unit is only included where specifically mentioned.

Table 7-1: Summary of Tutorial Example System Data

topology	3-bus triangle network
generators	2 identical 200 MW gens at bus 1, different reserve cost 500 MW gen at bus 2 all 3 have identical quadratic generation costs[*]
load	450 MW at bus 3 curtailable at \$1000/MWh
branches	300 MW limit, line 1–2 240 MW limit, line 1–3 300 MW limit, line 2–3
adequacy requirement	option 1: 150 MW system requirement option 2: contingencies: - generator 2 at bus 1 - line 1–3
wind	unit at bus 2 with 100 MW output in nominal case stochastic cases use 3 samples of normal distribution
storage	200 MWh unit at bus 3 80 MW max charge/discharge rate

[*] Linear costs of \$25, \$30, and \$40/WMh are used for some examples.

The code for the following examples can be found in *<MATPOWER>*/`most/lib/t` in the files whose names begin with `most_ex`, such as `most_ex1_ed.m`. For all of the following examples, assume that `mpopt` is a MATPOWER options struct and that `define_constants` has already been executed.

```
define_constants
mpopt = mpoption('verbose', 0);
```
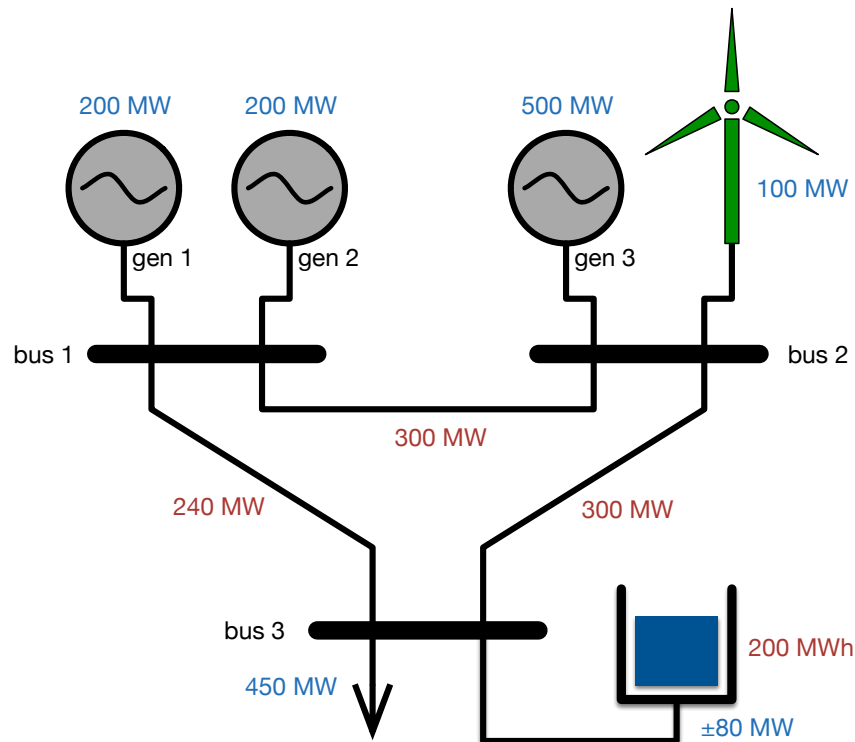
Figure 7-1: Tutorial Example System

7.1 Single Period Problems

Even without MOST, MATPOWER can solve the deterministic single period problems since they are just special cases of the optimal power flow problem. Where applicable, both methods will be shown for comparison.

7.1.1 Example 1 – Deterministic Economic Dispatch

The single period deterministic economic dispatch problem minimizes the cost of generation subject to generator limits, so the solution is identical to that given by a DC OPF problem when the branch flow limits are eliminated.[26] Of course, this can be solved without MOST by using rundcopf, as shown in the example below, where the results are returned in the variable r1, a standard MATPOWER OPF results struct with bus, branch, gen fields, etc.

---

[26]The deterministic economic dispatch examples can be found in most_ex1_ed.m.

```
mpc = loadcase('ex_case3a');
mpc.branch(:, RATE_A) = 0;  % disable line flow limits (mimic no network case)
r1 = rundcopf(mpc, mpopt);
Pg1 = r1.gen(:, PG);        % active generation
lam1 = r1.bus(:, LAM_P);    % nodal energy price
```

The equivalent "no network" economic dispatch problem can be solved by MOST as follows. In this case, the variable r2 with many of the results can be extracted from the mpc field of the first (and, in this case, only) element of mdo.flow. As described in Table 5-10, this is also a standard MATPOWER case struct containing the expected results.

```
mpc = loadcase('ex_case3a');
mpopt = mpoption(mpopt, 'most.dc_model', 0);    % use model with no network
mdi = loadmd(mpc);
mdo = most(mdi, mpopt);
r2 = mdo.flow.mpc;
Pg2 = r2.gen(:, PG);        % active generation
lam2 = r2.bus(:, LAM_P);    % nodal energy price
```

Zonal reserve requirements can be added and the problem solved by runopf_w_res as described in Section 7.5.1 in the MATPOWER User's Manual. The solved reserve quantities and prices are returned in r1.reserves, as summarized in Table 7-5 in the same section.

```
mpc = loadcase('ex_case3a');
mpc.branch(:, RATE_A) = 0;  % disable line flow limits (mimic no network case)
mpopt = mpoption(mpopt, 'model', 'DC');
r1 = runopf_w_res(mpc, mpopt);
Pg1 = r1.gen(:, PG);        % active generation
lam1 = r1.bus(:, LAM_P);    % nodal energy price
R1 = r1.reserves.R;         % reserve quantity
prc1 = r1.reserves.prc;     % reserve price
```

The equivalent problem, solved by MOST is the following, where the inputs must be specified in mdi.FixedReserves and the solved reserve quantities and prices are found in r2.reserves.

```
mpc = loadcase('ex_case3a');
mpopt = mpoption(mpopt, 'most.dc_model', 0);    % use model with no network
mdi = loadmd(mpc);
mdi.FixedReserves = mpc.reserves;   % include fixed zonal reserves
mdo = most(mdi, mpopt);
r2 = mdo.flow.mpc;
Pg2 = r2.gen(:, PG);          % active generation
lam2 = r2.bus(:, LAM_P);      % nodal energy price
R2 = r2.reserves.R;           % reserve quantity
prc2 = r2.reserves.prc;       % reserve price
```

7.1.2 Example 2 – Deterministic DC OPF

The deterministic optimal power flow problem simply adds a DC power flow network model, including branch flow limits.[27] Once again, this problem can be solved with `rundcopf`, now without disabling branch flow limits.

```
mpc = loadcase('ex_case3a');
r1 = rundcopf(mpc, mpopt);
Pg1 = r1.gen(:, PG);          % active generation
lam1 = r1.bus(:, LAM_P);      % nodal energy price
```

And it can be solved with MOST, by turning the DC network model back on (the default).

```
mpc = loadcase('ex_case3a');
mpopt = mpoption(mpopt, 'most.dc_model', 1);    % use DC network model (default)
mdi = loadmd(mpc);
mdo = most(mdi, mpopt);
r2 = mdo.flow.mpc;
Pg2 = r2.gen(:, PG);          % active generation
lam2 = r2.bus(:, LAM_P);      % nodal energy price
```

Similarly, zonal reserve requirements can be included as above in the economic dispatch problem, and solved via **runopf_w_res**.

---

[27]The deterministic DC OPF examples can be found in `most_ex2_dcopf.m`.

```
mpc = loadcase('ex_case3a');
mpopt = mpoption(mpopt, 'model', 'DC');
r1 = runopf_w_res(mpc, mpopt);
Pg1 = r1.gen(:, PG);          % active generation
lam1 = r1.bus(:, LAM_P);      % nodal energy price
R1 = r1.reserves.R;           % reserve quantity
prc1 = r1.reserves.prc;       % reserve price
```

And the MOST equivalent, in this case, looks like the following.

```
mpc = loadcase('ex_case3a');
mpopt = mpoption(mpopt, 'most.dc_model', 1);    % use DC network model (default)
mdi = loadmd(mpc);
mdi.FixedReserves = mpc.reserves;   % include fixed zonal reserves
mdo = most(mdi, mpopt);
r2 = mdo.flow.mpc;
Pg2 = r2.gen(:, PG);          % active generation
lam2 = r2.bus(:, LAM_P);      % nodal energy price
R2 = r2.reserves.R;           % reserve quantity
prc2 = r2.reserves.prc;       % reserve price
```

7.1.3 Example 3 – Deterministic DC OPF with Binary Commitment

The option of binary commitment decisions can be added to the deterministic economic dispatch and DC OPF problems[28] above by specifying a `'CommitKey'` value for each generator in the corresponding `xGenData`.[29]

In this example, ex_case3b is modified by adding a wind generator at bus 2, with available generation capacity of 100 MW, and scaling the load to 499 MW. Startup and shutdown costs are also ignored. The `xGenData` in this example indicates that the three conventional generators are available for commitment (`CommitKey = 1`) and the dispatchable load is always on (`CommitKey = 2`).[30]

---

[28]The deterministic DC OPF examples with binary commitment can be found in most_ex3_dcopf_w_uc.m.

[29]See Section 6.11.

[30]The `xGenData` specified in ex_wind_uc.m for the wind unit also indicates that it is always on (`CommitKey = 2`).

```
casefile = 'ex_case3b';
mpc = loadcase(casefile);
xgd_table.colnames = { 'CommitKey' };
xgd_table.data = [ 1; 1; 1; 2];
xgd = loadxgendata(xgd_table, mpc);
[iwind, mpc, xgd] = addwind('ex_wind_uc', mpc, xgd);
mpc = scale_load(499, mpc, [], struct('scale', 'QUANTITY'));
mpc.gencost(:, STARTUP) = 0;    % ignore STARTUP and SHUTDOWN
mpc.gencost(:, SHUTDOWN) = 0;   % costs for this example
```

MATPOWER's `runduopf` function uses a heuristic to solve this problem as described in Chapter 8 of the MATPOWER User's Manual. This heuristic however can be quite slow on large systems and there is no measure of the quality of the resulting solution.

```
r1 = runduopf(mpc, mpopt);
u1 = r1.gen(:, GEN_STATUS); % commitment status
Pg1 = r1.gen(:, PG);        % active generation
lam1 = r1.bus(:, LAM_P);    % nodal energy price
```

On the other hand, MOST takes advantage of an explicit MIP solver to solve this problem rather more efficiently and with solution quality guarantees.

```
mdi = loadmd(mpc, [], xgd);
mdo = most(mdi, mpopt);
r2 = mdo.flow.mpc;
u2 = mdo.UC.CommitSched;    % commitment status
Pg2 = r2.gen(:, PG);        % active generation
lam2 = r2.bus(:, LAM_P);    % nodal energy price
```

7.1.4   Example 4 – Secure and Stochastic DC OPF

In contrast to a deterministic optimal power flow, which solves for dispatches, voltages, flows and prices for a single scenario, the examples in this section in corporate multiple probability-weighted scenarios.[31]

Secure DC OPF - with contingencies

Instead of using pre-determined fixed zonal reserve requirements to ensure a secure dispatch, as in the example in Section 7.1.2, a set of credible contingencies can be

_____

[31]The secure and stochastic DC OPF examples can be found in `most_ex4_dcopf_ss.m`.

included explicitly via a contingency table, described in Section 5.1.5. The function ex_contab defines a contingency table with two outages. Generator 2 at bus 1 trips off-line with a 6% probability, and the transmission line from bus 1 to bus 3 fails with a 4% probability.

```
function contab = ex_contab
define_constants;
% label probty  type          row column       chgtype newvalue
contab = [
    1   0.06    CT_TGEN     2   GEN_STATUS  CT_REP  0;      %% gen 2 at bus 1
    2   0.04    CT_TBRCH    2   BR_STATUS   CT_REP  0;      %% line 1-3
];
```

The xGenData is loaded from a file (ex_xgd_res) that defines prices and capacities for contingency reserves.

```
xgd = loadxgendata('ex_xgd_res', mpc);
mdi = loadmd('ex_case3a', [], xgd, [], 'ex_contab');
mdo = most(mdi, mpopt);
EPg = mdo.results.ExpectedDispatch;     % expected active generation
Elam = mdo.results.GenPrices;           % nodal energy price
most_summary(mdo);      % print results, depending on 'verbose' option
```

The results can be printed by most_summary or extracted directly from the output MOST Data struct, mdo.

Stochastic DC OPF - with renewable uncertainty

In contrast to the discrete event uncertainty related to contingencies, forecasting of uncertain system parameters, such as demand or renewable generation from wind or solar, results in a different sort of uncertainty that can be approximated by a set of probability-weighted scenarios.

In the example below, a wind generator is added to the system and the maximum output of the unit is set to 0, 50 and 100 MW, respectively, for three scenarios defined by a profile. The probabilities of the scenarios are specified in transmat.

```
mpc = loadcase('ex_case3a');
xgd = loadxgendata('ex_xgd_res', mpc);
[iwind, mpc, xgd] = addwind('ex_wind', mpc, xgd);
transmat = {[0.16; 0.68; 0.16]};
nt = 1;     % number of periods
nj = 3;     % number of scenarios
profiles = getprofiles(uniformwindprofile(nt, nj), iwind);
```

These parameters are then used by `loadmd` to create the MOST Data struct to pass to the solver.

```
mdi = loadmd(mpc, transmat, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
EPg = mdo.results.ExpectedDispatch;     % active generation
Elam = mdo.results.GenPrices;           % nodal energy price
most_summary(mdo);      % print results, depending on 'verbose' option
```

As with the previous example, the results can be printed by `most_summary` or extracted directly from the output MOST Data struct, `mdo`.

Secure Stochastic DC OPF

Both types of uncertainty can be included in the same problem, with uncertain wind and contingencies, by passing both the contingency table and the wind profile with transition probabilities to `loadmd`.

```
mdi = loadmd(mpc, transmat, xgd, [], 'ex_contab', profiles);
mdo = most(mdi, mpopt);
EPg = mdo.results.ExpectedDispatch;     % active generation
Elam = mdo.results.GenPrices;           % nodal energy price
most_summary(mdo);      % print results, depending on 'verbose' option
```

Secure Stochastic DC OPF with Binary Commitment

Binary commitment decisions are added to the previous example by including a `'CommitKey'` value for each generator in the corresponding `xGenData`, in this case specified in `ex_xgd_uc.m`. In this example the generator startup and shutdown costs are ignored and the load is decreased to 350 MW.

```
casefile = 'ex_case3b';
mpc = loadcase(casefile);
xgd = loadxgendata('ex_xgd_uc', mpc);
[iwind, mpc, xgd] = addwind('ex_wind_uc', mpc, xgd);
mpc = scale_load(350, mpc, [], struct('scale', 'QUANTITY'));
mpc.gencost(:, STARTUP) = 0;
mpc.gencost(:, SHUTDOWN) = 0;
```

Using the wind profile and transition probabilities defined in the previous example, we can load and run this case as follows.

```
mdi = loadmd(mpc, transmat, xgd, [], 'ex_contab', profiles);
mdo = most(mdi, mpopt);
u = mdo.UC.CommitSched;               % commitment status
EPg = mdo.results.ExpectedDispatch;   % active generation
Elam = mdo.results.GenPrices;         % nodal energy price
most_summary(mdo);      % print results, depending on 'verbose' option
```

Looking at the value of u, we see that in this example generator 2 is shut down.

```
>> u

u =

    1
    0
    1
    1
    1
```

7.2 Multiperiod Problems

For the multiperiod examples, a 12 hour planning horizon is used. The load profile is defined in ex_load_profile.m that varies from a high of 540 MW in period 3 to a low of 300 MW in period 9, as illustrated in Figure 7-2. The available output of the wind farm is represented in various ways. For the stochastic cases the output is represented by 3 samples from a normal distribution around a mean forecast value, where the distribution widens the further as we forecast futher into the future. In the deterministic cases the available wind is simply set to this mean, as represented by the solid red line in Figure 7-2. These profiles are defined in ex_wind_profile.m and ex_wind_profile_d.m, respectively.

7.2.1 Example 5 – Deterministic Multiperiod OPF

This example illustrates a simple deterministic multiperiod DC OPF problem, where the dispatches in adjacent periods are linked by ramping constraints and costs.[32] As described above, the variations to load and wind through the planning horizon are defined using profiles.

The xGenData is loaded from ex_xgd_ramp.m, which includes a $10/MW cost on upward and downward ramping of generator 3 from one period to the next.

---

[32]The deterministic multiperiod DC OPF examples can be found in most_ex5_mpopf.m.
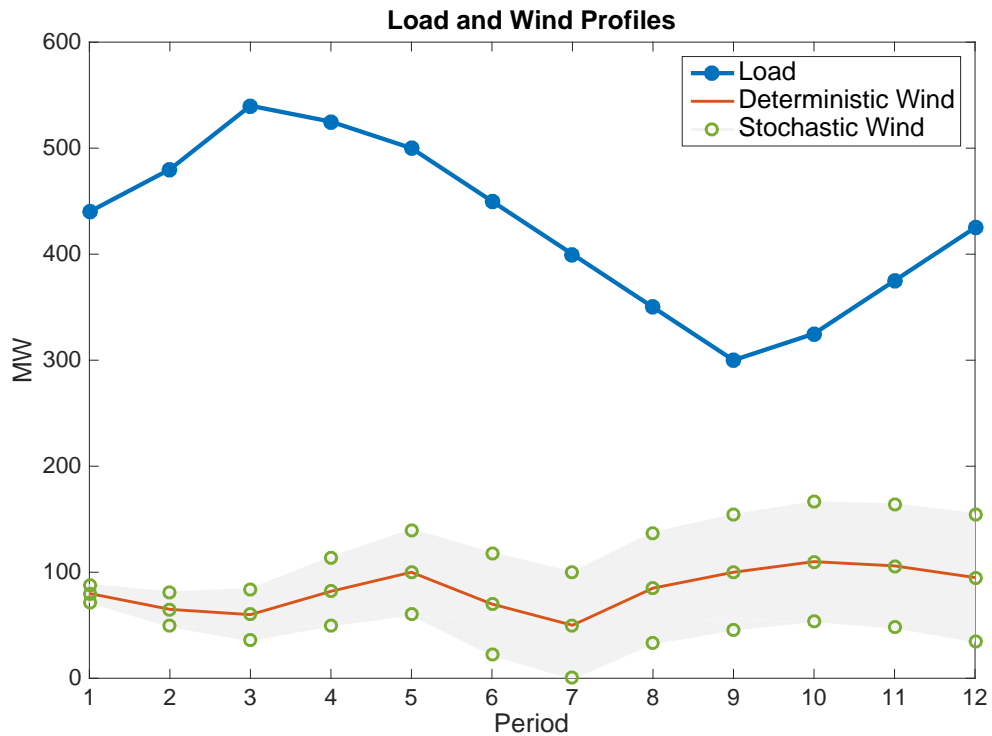
Figure 7-2: Example Load and Wind Profiles

```
casefile = 'ex_case3b';
mpc = loadcase(casefile);
xgd = loadxgendata('ex_xgd_ramp', mpc);
[iwind, mpc, xgd] = addwind('ex_wind', mpc, xgd);
profiles = getprofiles('ex_wind_profile_d', iwind);
profiles = getprofiles('ex_load_profile', profiles);
nt = size(profiles(1).values, 1);        % number of periods
```

The generator dispatches can be found in `mdo.flow(t).mpc.gen(:, PG)` for period t, or for all periods in the $n_g \times n_t$ matrix `mdo.results.ExpectedDispatch`.

```
mdi = loadmd(mpc, nt, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
EPg = mdo.results.ExpectedDispatch;     % active generation
Elam = mdo.results.GenPrices;            % nodal energy price
most_summary(mdo);      % print results, depending on 'verbose' option
```

The `RampWearCostCoeff` field of `xgd` is modified to add the wear and tear ramping costs from (4.10). This can be done directly for an existing `xgd`, as shown below, or by adding a `RampWearCostCoeff` column in the `xGenData` file and defining the parameters there.

```
xgd.RampWearCostCoeff(1:3) = 1;
mdi = loadmd(mpc, nt, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
EPg = mdo.results.ExpectedDispatch;    % active generation
Elam = mdo.results.GenPrices;          % nodal energy price
most_summary(mdo);      % print results, depending on 'verbose' option
```

7.2.2   Example 6 – Deterministic Unit Commitment

This example illustrates a deterministic unit commitment problem and how the commitment changes as more features are added.[33] These examples require a mixed-integer solver as described in Section 2.1 on System Requirements.

The MATPOWER case and `xGenData` for the full-featured example are loaded first and saved for later. Note that the `xGenData` here comes from `ex_xgd_uc.m` which includes `CommitKey` and activates the unit commitment formulation.

```
casefile = 'ex_case3b';
mpc = loadcase(casefile);
xgd = loadxgendata('ex_xgd_uc', mpc);
[iwind, mpc, xgd] = addwind('ex_wind_uc', mpc, xgd);
profiles = getprofiles('ex_wind_profile_d', iwind);
profiles = getprofiles('ex_load_profile', profiles);
nt = size(profiles(1).values, 1);       % number of periods
mpc_full = mpc;                          % save for later
xgd_full = xgd;                          % save for later
```

Base : No Network

This example begins with a simple sequence of economic dispatch problems, with no network constraints, no startup and shutdown costs, no minimum up or down time constraints, no ramp constraints or ramp reserve costs, and no storage. These features, except for storage, are already in included in the full model data loaded, so

---

[33]The deterministic unit commitment examples can be found in `most_ex6_uc.m`. These example cases and the code used to produce the plots can also be found in the test file `t_most_uc.m`. Both files contain additional solver-specific options that you may find useful for these unit commitment examples.

the first step is to remove these features to prepare for the first example. They will be added back in one at a time in the subsequent examples.

```
mpc.gencost(:, [STARTUP SHUTDOWN]) = 0; % remove startup/shutdown costs
xgd.MinUp(2) = 1;                        % remove min up-time constraint
xgd.PositiveLoadFollowReserveQuantity(3) = 250; % remove ramp reserve
xgd.PositiveLoadFollowReservePrice(3) = 1e-6;   % constraint and costs
xgd.NegativeLoadFollowReservePrice(3) = 1e-6;
```

This model can then be run after turning off the DC network modeling.

```
mpopt = mpoption(mpopt, 'most.dc_model', 0);    % use model with no network
mdi = loadmd(mpc, nt, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
ms = most_summary(mdo);      % print results, depending on 'verbose' option
```

The resulting commitment, dispatch and price schedules are shown in Figure 7-3. Notice that generator 3 only operates during hours 2, 3 and 4 and, as expected, the prices at all three buses are identical, since there is no network model to introduce transmission congestion.

Add DC Network Model

If a DC network model is added, by simply toggling the 'most.dc_model' option and re-running the same model, the congestion in the line from bus 1 to bus 3 results in the nodal prices separating from one another and generator 3 carrying more of the load during the periods of higher net load, as seen in Figure 7-4.

```
mpopt = mpoption(mpopt, 'most.dc_model', 1);    % use DC network model (default)
mdo = most(mdi, mpopt);
```

Add Startup and Shutdown Costs

To add startup and shutdown costs, restore the values from the original MATPOWER case and re-run.

```
mpc.gencost(2, [STARTUP SHUTDOWN]) = [ 200 200];
mpc.gencost(3, [STARTUP SHUTDOWN]) = [3000 600];
% equivalent to doing: mpc = mpc_full;
mdi = loadmd(mpc, nt, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
```

Notice in Figure 7-5 that this results in generator 3 remaining on through the lower load hours, allowing generator 2 to stay off for hours 8–11.
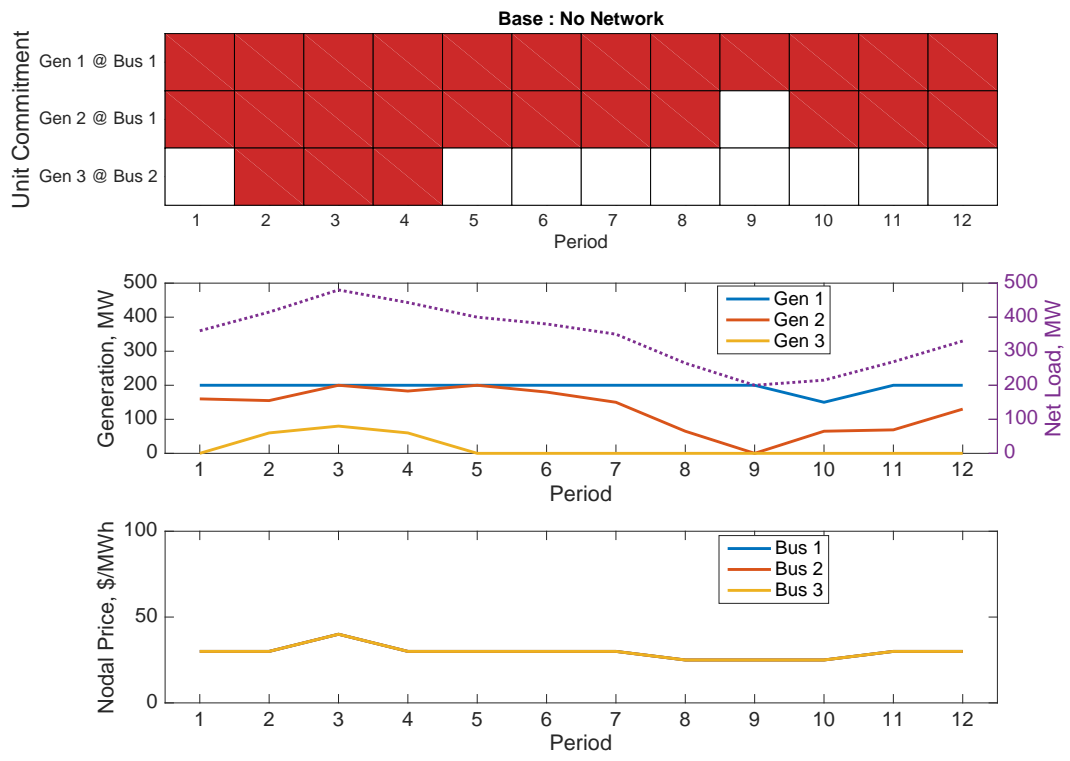
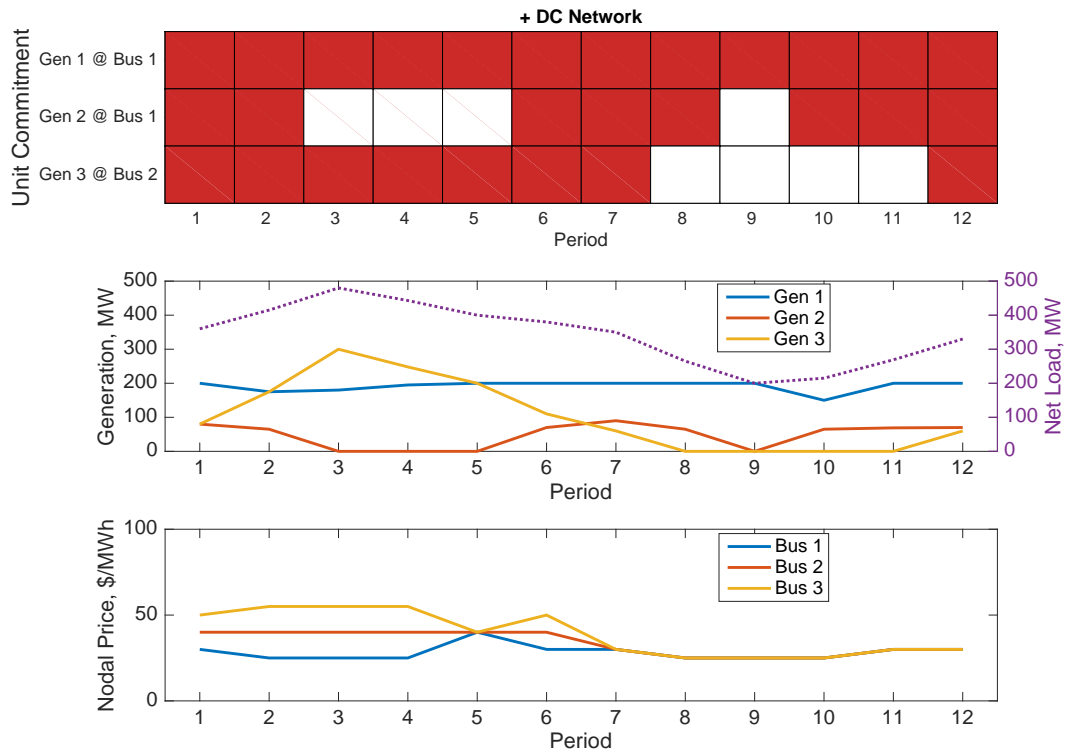Figure 7-3: Deterministic UC : Base Case with No Network

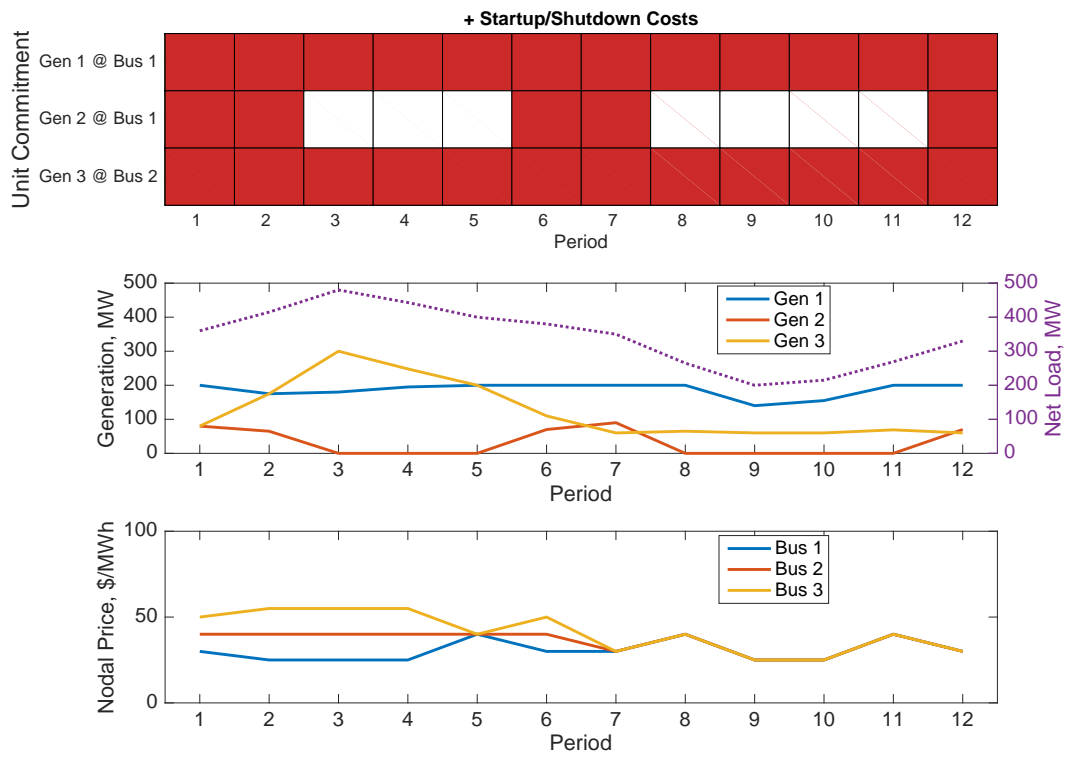Figure 7-4: Deterministic UC : Add DC Network Model

Figure 7-5: Deterministic UC : Add Startup and Shutdown Costs

Add Minimum Up and Down Time Constraints

Notice in the previous example that the generator 2 is only running for two hours (6 and 7) in the middle of the planning horizon. Adding back the 3 hour minimum up-time constraint eliminates that solution.

```
xgd.MinUp(2) = 3;
mdi = loadmd(mpc, nt, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
```

Figure 7-6 shows how this constraint results in starting up generator 2 an hour earlier (in period 5). Notice also that previously there was no network congestion in period 5, but starting generator 2 introduces congestion, causing the nodal prices in that period to separate from one another.

Add Ramping Constraints and Ramp Reserve Costs

To add back the ramping constraints and ramp reserve costs, restore the values from the original xGenData and re-run.

```
xgd.PositiveLoadFollowReserveQuantity(3) = 100; % restore ramp reserve
xgd.PositiveLoadFollowReservePrice(3) = 10;     % constraint and costs
xgd.NegativeLoadFollowReservePrice(3) = 10;
% equivalent to doing: xgd = xgd_full;
mdi = loadmd(mpc, nt, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
```

Previously, generator 3 was ramping more than 200 MW from hour 1 to hour 3, which the newly added ramp constraint of 100 MW per hour precludes. Figure 7-7 shows that this fast ramp is reduced by shutting down generator 2 during the first two hours and starting generator 3 at a higher output level.

Add Storage

Finally, a 200 MWh storage unit is added at bus 3, as shown in the diagram in Figure 7-1. The magnitude of the power injection for this storage unit is limited to 80 MW, both for "charging" and "discharging". The cyclic storage constraint option is used to ensure that the stored energy at the end of the planning horizon is equal to the stored energy at the beginning.
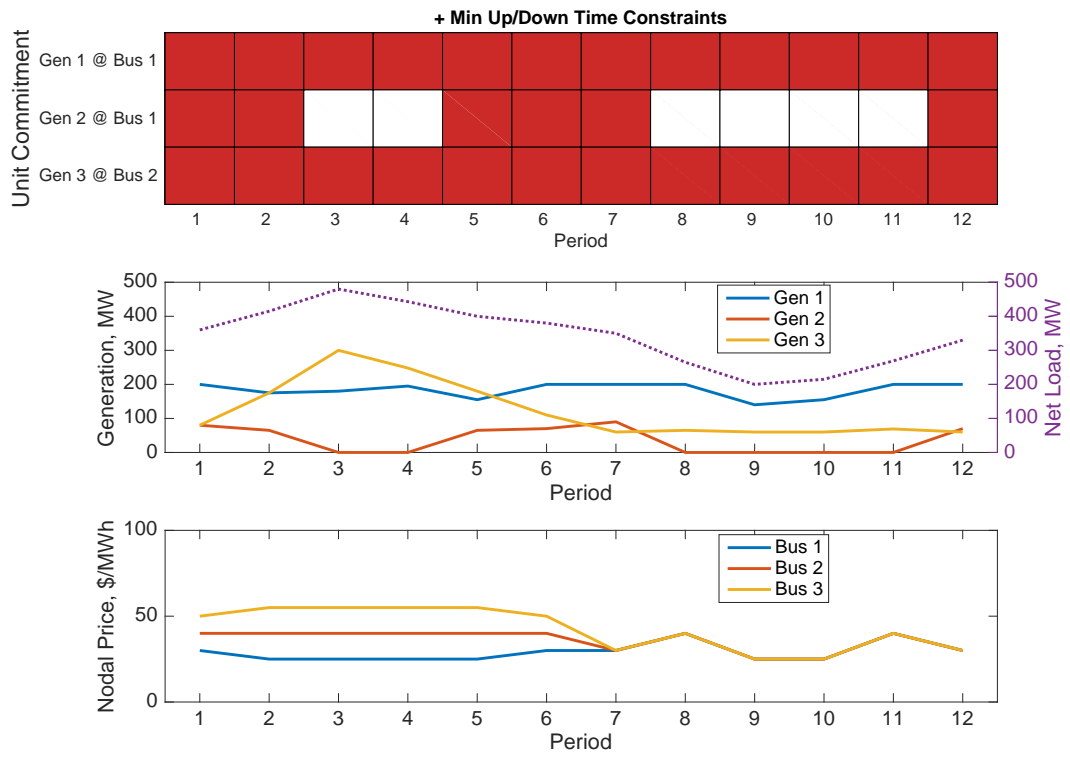
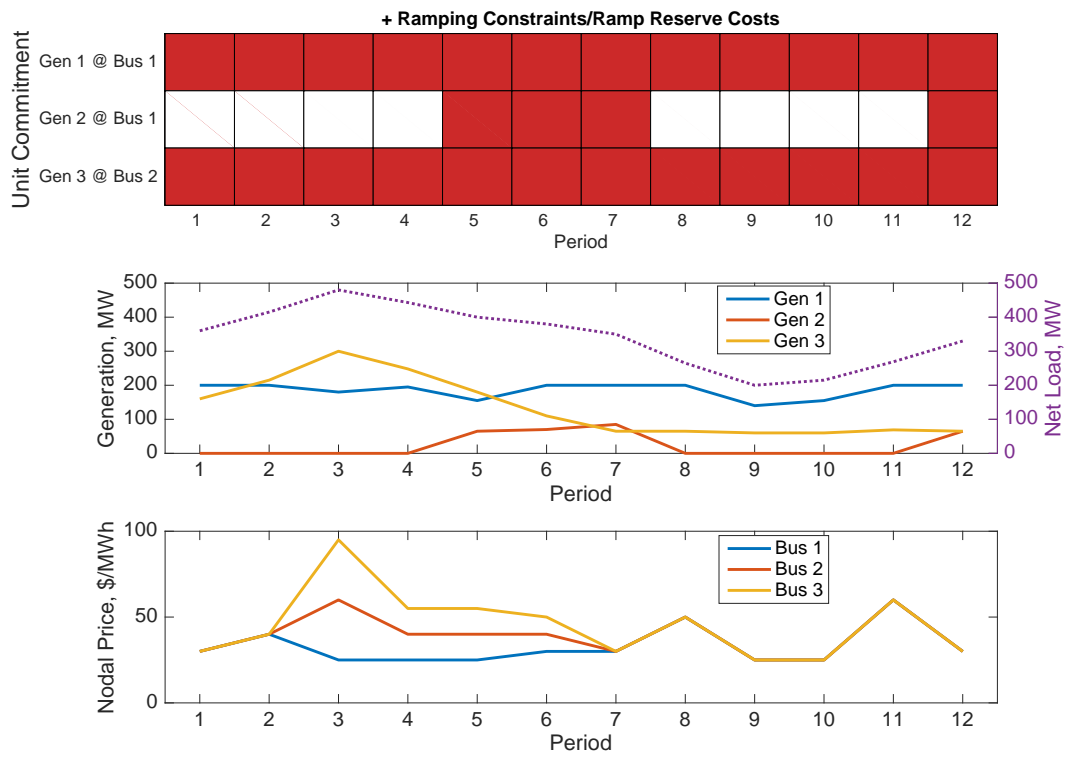Figure 7-6: Deterministic UC : Add Min Up/Down Time Constraints

Figure 7-7: Deterministic UC : Add Ramp Constraints and Ramp Reserve Costs

```
mpopt = mpoption(mpopt, 'most.storage.cyclic', 1);
[iess, mpc, xgd, sd] = addstorage('ex_storage', mpc, xgd);
mdi = loadmd(mpc, nt, xgd, sd, [], profiles);
mdo = most(mdi, mpopt);
```

Figure 7-8 illustrates the effect of adding the storage unit. Since the unit is located at bus 3 with the load, it can reduce the congestion enough in peak hours to allow generator 2 to stay on for the full 12 hours. It also reduces the ramping capability required from the more expensive generator 3. As expected, the storage unit is charged during the periods of lower demand and lower price and discharged during the periods of higher demand and higher price.
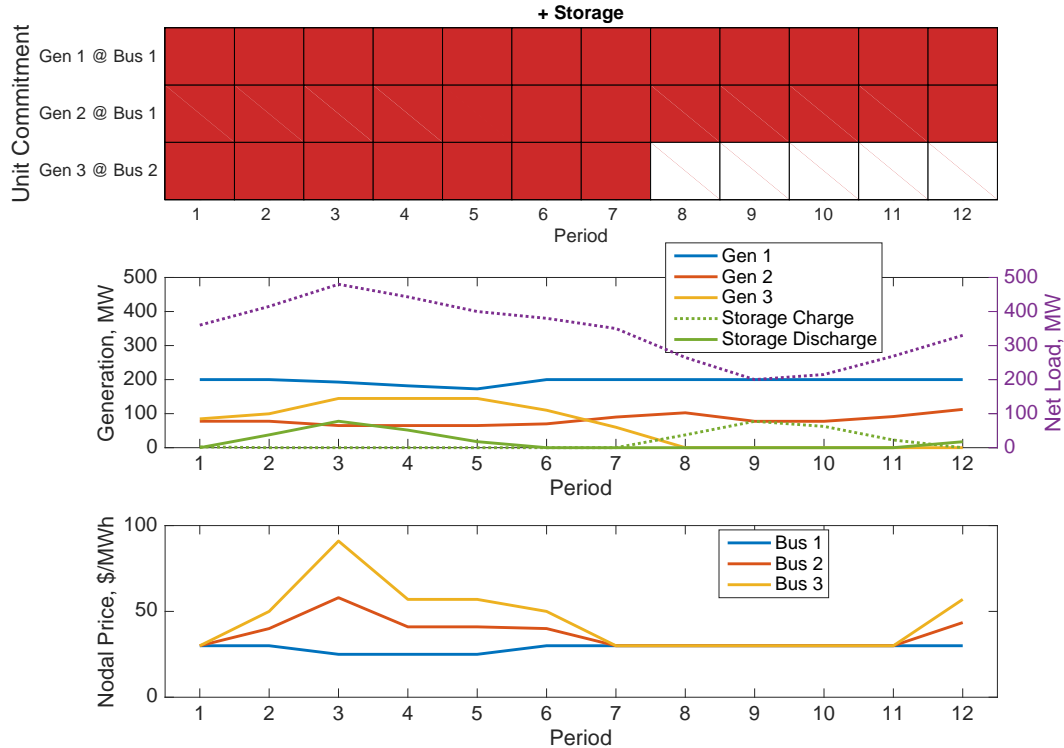
Figure 7-8: Deterministic UC : Add Storage

98

7.2.3 Example 7 – Secure Stochastic Unit Commitment

The following examples are based on example 6 above, with all of the features, except storage, included. Instead of deterministic wind, however, a stochastic model of wind is assumed.[34] In these examples, three samples of wind availability serve as the base scenarios, representing low, average and high wind realizations. These wind scenarios are defined in `ex_wind_profile.m`. In this case, since there is a single load profile defined in `ex_load_profile.m`, it is automatically expanded to apply to all three wind scenarios as well. The examples in this section all use the following setup.

```
mpc = loadcase('ex_case3b');
xgd = loadxgendata('ex_xgd_uc', mpc);
[iwind, mpc, xgd] = addwind('ex_wind_uc', mpc, xgd);
profiles = getprofiles('ex_wind_profile', iwind);
profiles = getprofiles('ex_load_profile', profiles);
nt = size(profiles(1).values, 1);      % number of periods
nj = size(profiles(1).values, 2);      % number of scenarios
```

Stochastic Unit Commitment – Individual Trajectories

The `transmat` argument to `loadmd` defines the probabilities of transitions from the scenarios in period $t - 1$ to the scenarios in period t. If identity matrices are used for these transition probabilities, this results in the special case in which there are 3 full trajectories through the horizon, each of which can be viewed as a different "scenario". That is, if the system is in the high wind state in the first period, it will stay in the high wind state in every subsequent period, and the same with the average and low wind states. Figure 7-9 illustrates this special case.

In this case, it is also necessary to define the binary valued mask $\zeta^{tj_2j_1}$ in (4.25) so that the ramp reserve constraints (4.26)–(4.27) only include the transitions with non-zero probability. This is done using the `filter_ramp_transitions` function.[35]

---

[34]The secure and stochastic unit commitment examples can be found in `most_ex7_suc.m`. These example cases and the code used to produce the plots can also be found in the test file `t_most_suc.m`.
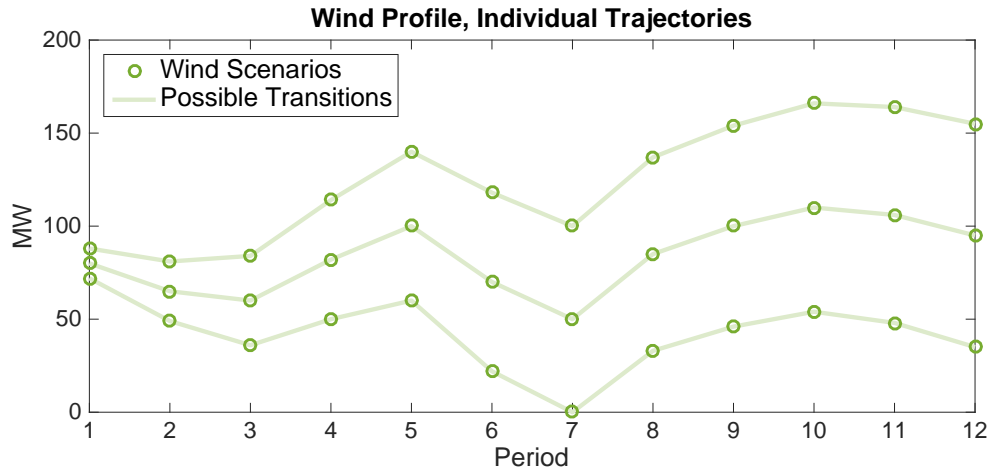[35]See Section 6.5.

Figure 7-9: Example Wind Profiles, Individual Trajectories

```
transmat = cell(1, nt);
I = speye(nj);
[transmat{:}] = deal(I);
transmat{1} = [0.16; 0.68; 0.16];   % period 1 probabilities
mdi = loadmd(mpc, transmat, xgd, [], [], profiles);
mdi = filter_ramp_transitions(mdi, 0.1);
mdo = most(mdi, mpopt);
ms = most_summary(mdo);
```

The resulting unit commitment, expected dispatch and price schedules are shown in Figure 7-10.

Stochastic Unit Commitment – Full Transition Probabilities

The more general case of stochastic unit commitment implemented by MOST uses full transition probability matrices, where transitions between low, average and high wind scenarios are allowed from period to period as illustrated in Figure 7-11. In this case the binary valued mask $\zeta^{tj_2j_1}$ is left at its default value of all ones, resulting in ramp reserve constraints that encompass the largest period to period ramps.

```
transmat = ex_transmat(nt);
mdi = loadmd(mpc, transmat, xgd, [], [], profiles);
mdo = most(mdi, mpopt);
ms = most_summary(mdo);
```
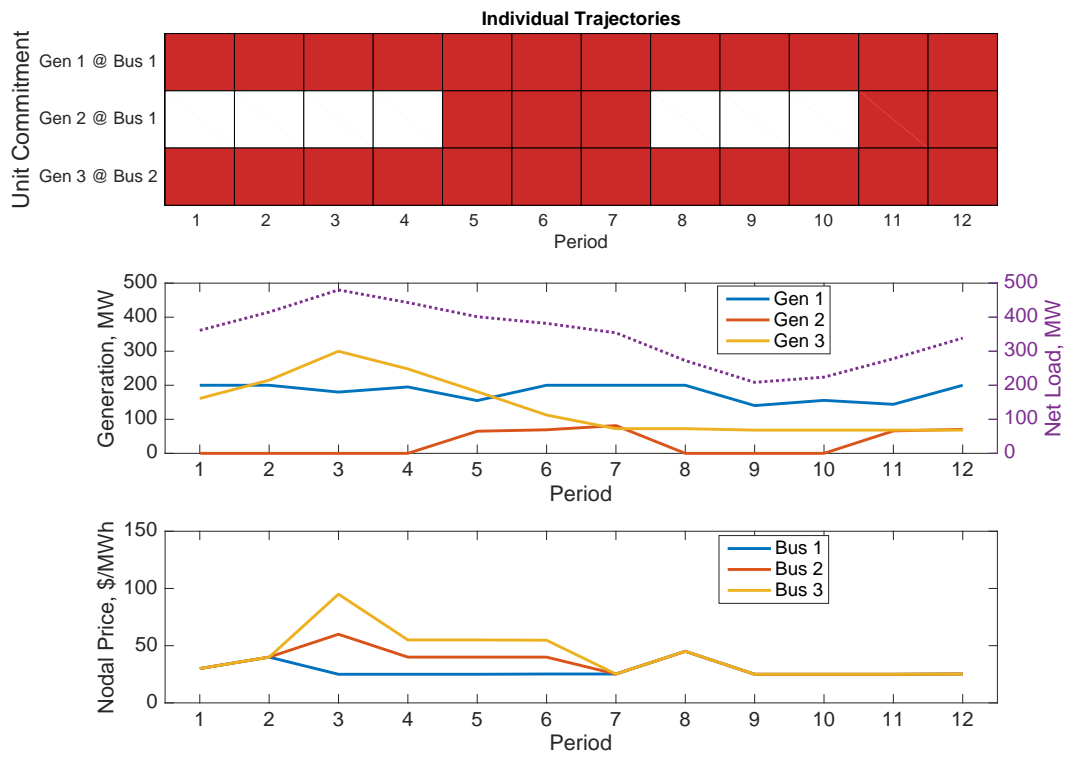
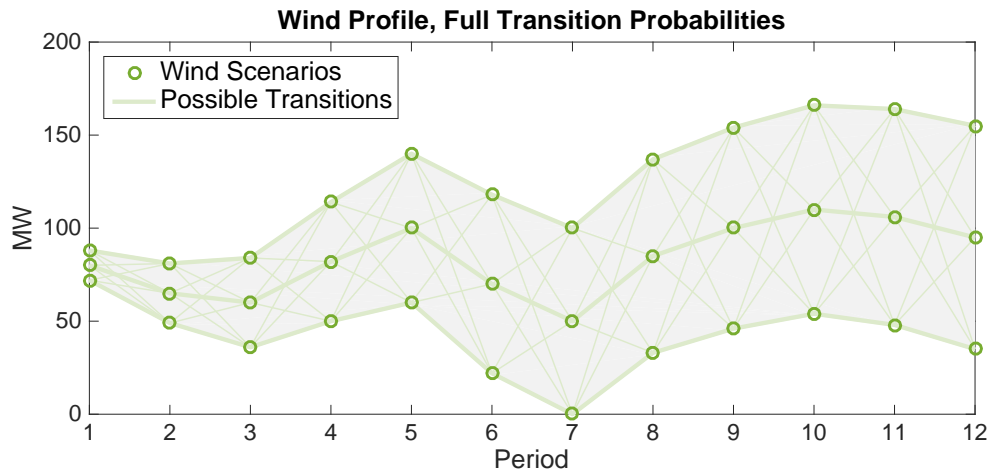Figure 7-10: Stochastic UC : Individual Trajectories

Figure 7-11: Example Wind Profiles, Full Transition Probabilities

This case results in a different unit commitment as seen in Figure 7-12, where generator 2 remains on throughout the later hours.

Secure Stochastic Unit Commitment

This example uses the general full transition probabilities and also includes contingencies for security in each period.

```
transmat = ex_transmat(nt);
mdi = loadmd(mpc, transmat, xgd, [], 'ex_contab', profiles);
mdo = most(mdi, mpopt);
ms = most_summary(mdo);
```

Figure 7-13 shows the resulting unit commitment, expected dispatch and pricing scheules.

Secure Stochastic Unit Commitment with Storage

Finally, a storage unit is added to the system, yielding a case that utilizes the majority of the features of the MOST formulation.

```
transmat = ex_transmat(nt);
[iess, mpc, xgd, sd] = addstorage('ex_storage', mpc, xgd);
mdi = loadmd(mpc, transmat, xgd, sd, 'ex_contab', profiles);
mdo = most(mdi, mpopt);
ms = most_summary(mdo);
```
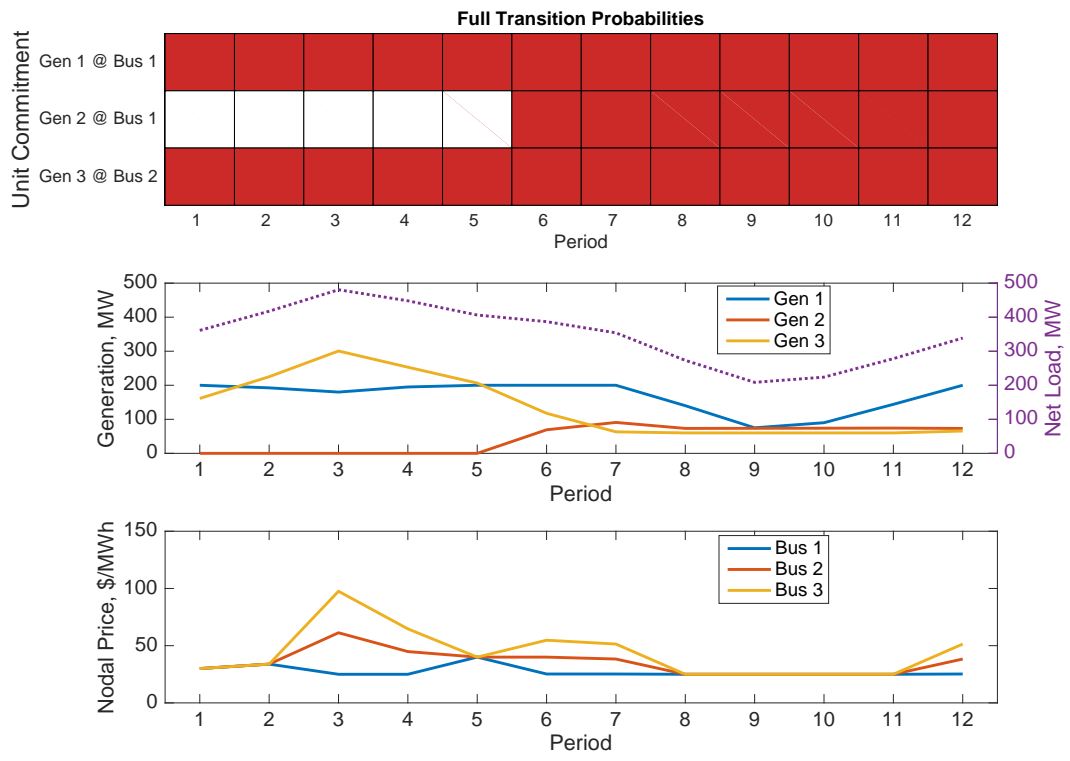
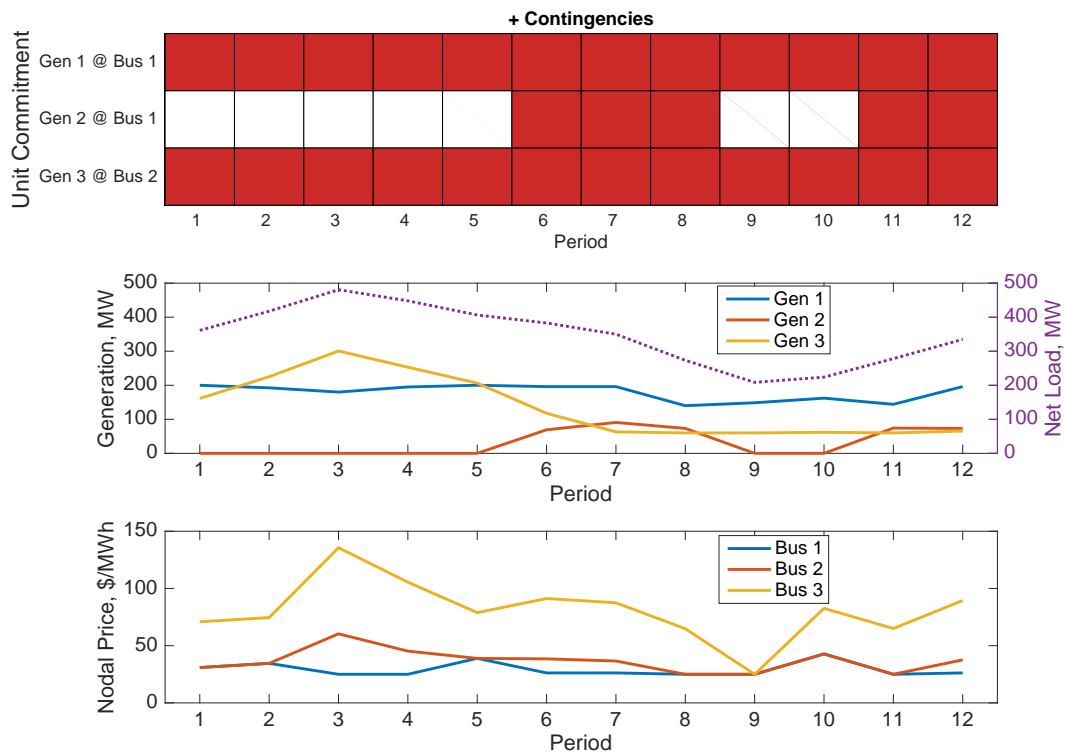Figure 7-12: Stochastic UC : Full Transition Probabilities

Figure 7-13: Secure Stochastic UC : Full Transition Probabilities + Contingencies

As seen in Figure 7-14, the storage allows all units to remain on for the entire horizon, avoiding the startup and shutdown costs.
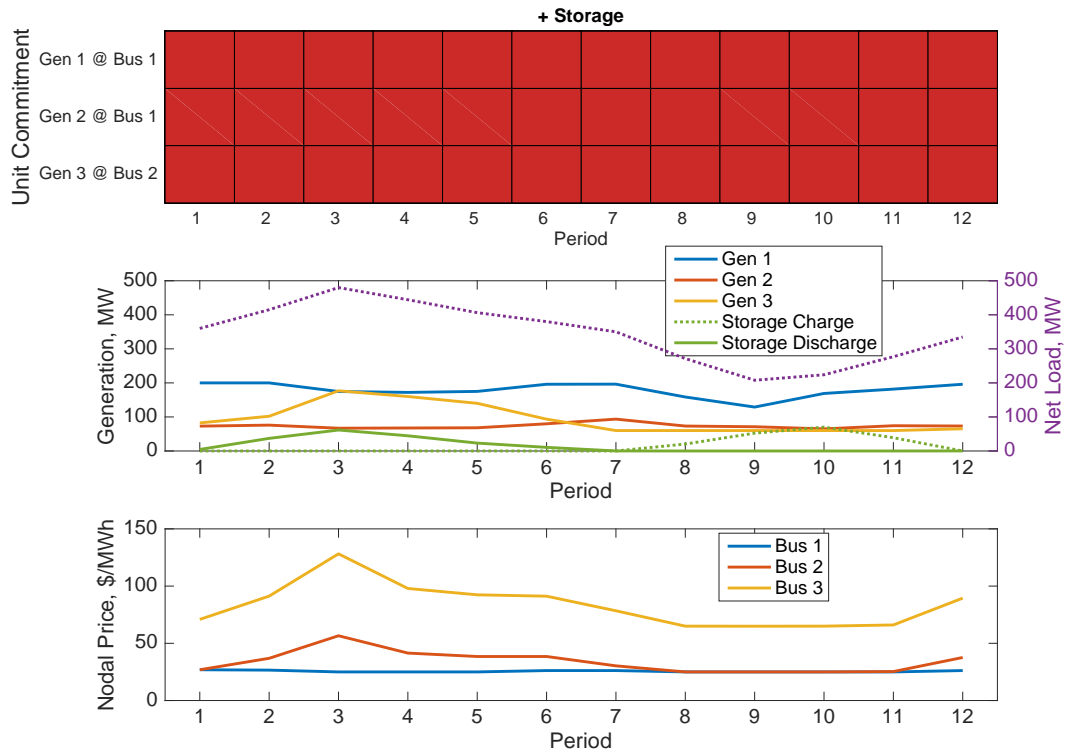

Figure 7-14: Secure Stochastic UC with Storage

7.2.4 Dynamical System Constraint Example

An example of using the linear time-varying dynamical system constraints can be found in `t_most_w_ds.m`.

8 Acknowledgments

Appendix A   MOST Files and Functions

This appendix lists all of the files and functions that MOST provides. In most cases, the function is found in a MATLAB M-file of the same name in the `lib` directory of the MOST distribution[36], where the `.m` extension is omitted from this listing. For more information on each, at the MATLAB prompt, simply type `help` followed by the name of the function. For documentation and data files, the filename extensions are included.

A.1   MOST Documentation Files

Table A-1: MOST Documentation Files

name	description
`docs/`	MOST documentation
`MOST-manual.pdf`	MOST User's Manual
`src/`	LaTeX source for MOST User's Manual
`lib/`	MOST software (see Table A-2)
`t/`	MOST tests and examples (see Tables A-3 and A-4)
`AUTHORS`	list of MOST authors
`CHANGES.md`	MOST change history
`CONTRIBUTING.md`	MOST Contributors Guide
`LICENSE`	license file
`README.md`	basic introduction to MOST

---

[36]That is, in the *<MATPOWER>*`/most/lib` directory.

A.2   MOST Functions

Table A-2: MOST Functions

name	description
lib/	
addgen2mpc	appends generators to existing case, see Section 6.1
addstorage	appends storage units to existing case, see Section 6.2
addwind	appends wind generators to existing case, see Section 6.3
apply_profile	applies a single profile to the specified data, see Section 6.4
filter_ramp_transitions	creates binary valued transition mask $\zeta^{tj_2 j_1}$ for ramping reserves based on a probability threshold, see Section 6.5
getprofiles	loads profiles from a struct or file, see Section 6.6
idx_profile	defines constants for use with profiles, see Section 6.7
loadgenericdata	loads data from a variable, M-file or MAT-file and checks that it matches a specified type, see Section 6.8
loadmd	loads a MOST Data struct, see Section 6.9
loadstoragedata	loads parameters for storage units, see Section 6.10
loadxgendata	loads extra generator data, see Section 6.11
md_init	data structure initialization
most	top-level solver, see Chapter 5
mostver	prints/returns version info for MOST, see Section 6.13
mpoption_info_most	option information for MOST
plot_gen	create plots of generator results
plot_storage	create plots of storage unit results
plot_uc_data	plot generator commitment summary from raw data
plot_uc	plot generator commitment summary from `md`

A.3 Automated Test Suite

Table A-3: MOST Test and Example Data

name	description
lib/t/	
c118swf	118-bus test case, used by t_most_w_ds
ex_case3a	sample 3-bus MATPOWER case (version a)
ex_case3b	sample 3-bus MATPOWER case (version b)
ex_contab	sample contingency table
ex_load_profile	sample deterministic load profile
ex_storage	sample StorageUnitData
ex_transmat	sample transition probability data
ex_wind_profile	sample stochastic wind profile
ex_wind_profile_d	sample deterministic wind profile
ex_wind	sample WindUnitData
ex_wind_uc	sample WindUnitData for UC problem
ex_xgd	sample xGenData
ex_xgd_ramp	sample xGenData with reserve and ramping costs
ex_xgd_res	sample xGenData with reserve costs
ex_xgd_uc	sample xGenData for UC problem
most_ex1_ed	Tutorial Example 1, see Section 7.1.1
most_ex2_dcopf	Tutorial Example 2, see Section 7.1.2
most_ex3_dcopf_w_uc	Tutorial Example 3, see Section 7.1.3
most_ex4_dcopf_ss	Tutorial Example 4, see Section 7.1.4
most_ex5_mpopf	Tutorial Example 5, see Section 7.2.1
most_ex6_uc	Tutorial Example 6, see Section 7.2.2
most_ex7_suc	Tutorial Example 7, see Section 7.2.3
t_case3_most	3-bus MATPOWER test case
t_case30_most	30-bus MATPOWER test case
t_most_suc_soln.mat	solution data for t_most_suc
t_most_uc_soln.mat	solution data for t_most_uc
t_most_w_ds_z.mat	solution data for t_most_w_ds

Table A-4: MOST Tests

name	description
lib/t/	
test_most	runs full MOST test suite
t_most_3b_1_1_0	3-bus, single period, no contingencies
t_most_3b_1_1_2	3-bus, single period, 2 contingencies
t_most_3b_3_1_0	3-bus, 3 periods, no contingencies
t_most_3b_3_1_2	3-bus, 3 periods, 2 contingencies
t_most_30b_1_1_0_uc	30-bus, single period, no contingencies, w/unit commitment
t_most_30b_1_1_0	30-bus, single period, no contingencies
t_most_30b_1_1_17	30-bus, single period, 17 contingencies
t_most_30b_3_1_0	30-bus, 3 periods, no contingencies
t_most_30b_3_1_17	30-bus, 3 periods, 17 contingencies
t_most_fixed_res	with fixed zonal reserve requirements
t_most_sp	single period continuous problems
t_most_spuc	single period mixed-integer problems, i.e. w/UC
t_most_suc	multiperiod with stochastic unit commitment
t_most_uc	multiperiod with deterministic unit commitment
t_most_w_ds	with linear dynamical system constraints
uniformwindprofile	creates a wind profile with evenly spaced capacity values

Appendix B Release History

The full release history can be found in *<MATPOWER>*/most/docs/CHANGES.

B.1 Version 1.0 – released Dec 16, 2016

The MOST 1.0 User's Manual is available online.[37]

New Open Development Model

- MOST development has moved to GitHub! The code repository is now publicly available to clone and submit pull requests.[38]

- Public issue tracker for reporting bugs, submitting patches, etc.[39]

- Separate repositories for MATPOWER, MOST, MIPS, MP-Test, all available from https://github.com/MATPOWER/.

- New developer e-mail list (MATPOWER-DEV-L) to facilitate communication between those collaborating on MATPOWER-related development. Sign up at: http://www.pserc.cornell.edu/matpower/mailinglists.html#devlist.

Other Changes

- No significant changes since first public beta release.[40]

B.2 Version 1.0.1 – released Oct 30, 2018

The MOST 1.0.1 User's Manual is available online.[41]

Bugs Fixed

- Fix bugs in plot_uc_data() resulting in incorrect legends.

- Fix dimension of RampWear cost indexing if mdi.OpenEnded is true.

- Add missing constant term to objective function value reported by most_summary.

---

[37]http://www.pserc.cornell.edu/matpower/docs/MOST-manual-1.0.pdf
[38]https://github.com/MATPOWER/most
[39]https://github.com/MATPOWER/most/issues
[40]Version 1.0b1 was released on Jun 1, 2016 and 1.0b2 on Nov 1, 2016
[41]http://www.pserc.cornell.edu/matpower/docs/MOST-manual-1.0.1.pdf

Other Changes

- LaTeX source code for MOST User's Manual included in `docs/src`.

- Updated to use OOP notation for `opt_model` object, and avoid calls to deprecated methods, using `init_indexed_name()` and `add_lin_constraint()` instead.

- Updated to use MATPOWER's new quadratic costs in `opt_model` in place of the legacy cost model.

- Add `success` flag to `md.results` output MOST Data struct to indicate success or failure of optimization.

Incompatible Changes

- Failure of the optimization no longer halts execution and jumps to the debugger.

- Requires MATPOWER 7.x or later.

References

[1] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," *Power Systems, IEEE Transactions on*, vol. 26, no. 1, pp. 12–19, Feb. 2011. DOI: 10.1109/TPWRS.2010.2051168 1, 1.2

[2] C. E. Murillo-Sánchez, R. D. Zimmerman, C. L. Anderson, and R. J. Thomas, "Secure Planning and Operations of Systems with Stochastic Sources, Energy Storage and Active Demand," *Smart Grid, IEEE Transactions on*, vol. 4, no. 4, pp. 2220–2229, Dec. 2013. DOI: 10.1109/TSG.2013.2281001 1, 1.2, 3

[3] A. J. Lamadrid, D. Muñoz-Álvarez, C. E. Murillo-Sánchez, R. D. Zimmerman, H. D. Shin and R. J. Thomas, "Using the Matpower Optimal Scheduling Tool to Test Power System Operation Methodologies Under Uncertainty," *Sustainable Energy, IEEE Transactions on*, 2018. DOI: 10.1109/TSTE.2018.2865454 1

[4] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower's Extensible Optimal Power Flow Architecture," *Power and Energy Society General Meeting, 2009 IEEE*, pp. 1–7, July 26–30 2009. DOI: 10.1109/PES.2009.5275967 1

[5] R. D. Zimmerman and C. E. Murillo-Sánchez, Matpower User's Manual, [Online]. Available: http://www.pserc.cornell.edu/matpower/ 1, 3

[6] The BSD 3-Clause License. [Online]. Available: http://opensource.org/licenses/BSD-3-Clause. 1.1

[7] A. J. Lamadrid, S. Maneevitjit, T. D. Mount, C. E. Murillo-Sánchez, R. J. Thomas, R. D. Zimmerman, "A 'SuperOPF' Framework", *CERTS Report*, December 2008. [Online]. Available: http://certs.lbl.gov/pdf/superopf-framework.pdf 3

[8] C. E. Murillo-Sánchez, R. D. Zimmerman, C. L. Anderson, and R. J. Thomas, "A Stochastic, Contingency-Based Security-Constrained Optimal Power Flow for the Procurement of Energy and Distributed Reserve," *Decision Support Systems*, Vol. 56, pp. 1-10, Dec. 2013, ISSN 0167-9236. DOI: 10.1016/j.dss.2013.04.006 3