

OSL Stochastic Extensions

Contents

1	Introduction	1
2	How to Run a Model with GAMS/OSLSE	1
3	Overview of OSLSE	1
4	Model Formulation	2
4.1	Intended Use	2
4.2	Model Requirements	4
5	Future Work / Wish List	4
6	GAMS Options	5
7	Summary of GAMS/OSLSE Options	5
8	The GAMS/OSLSE Log File	5
9	Detailed Description of GAMS/OSLSE Options	5

1 Introduction

This document describes the GAMS 2.50 link to the IBM OSL Stochastic Extensions solvers.

The GAMS/OSLSE link combines the high level modeling capabilities of GAMS with a powerful decomposition solver for stochastic linear programs. It allows the user to formulate the deterministic equivalent stochastic program using general GAMS syntax and to pass this on to the solver without having to be concerned with the programming details of such a task. The default solver options used, while suitable for most solves, can be changed via a simple options file.

Those seeking an introduction to the ideas and techniques of stochastic programming should consult the [IBM Stochastic Extensions](#) web page or the recently published texts *Introduction to Stochastic Programming* (J.R.Birge and F.V.Louveaux, Springer-Verlag, New York, 1997) or *Stochastic Programming* (P.Kall and S.W.Wallace, John Wiley and Sons, Chichester, England, 1994).

2 How to Run a Model with GAMS/OSLSE

The following statement can be used inside your GAMS model:

```
option lp = oslse;
```

This statement should appear before the solve statement you wish to affect.

3 Overview of OSLSE

OSL Stochastic Extensions (OSLSE) allows the input of a stochastic program and its solution via a nested decomposition solver that implements the L-shaped method of Van-Slyke & Wets, a.k.a. Benders decomposition.

The stochastic program is assumed to have an event tree structure that is crucial to the efficient input and solution of the problem. At this time, only discrete probability distributions are accommodated.

4 Model Formulation

To properly understand and use the GAMS/OSLSE link, one must understand the *event tree* representation of a stochastic program. The OSLSE subroutine library assumes that the SP has this form, and depends on this fact to efficiently store and solve the problem.

To construct an event tree, the variables in the SP are first partitioned, where the variables in each subset represent decisions made at a particular time period and based on one realization of the stochastic parameters known at that time period. Each of these subsets form a *node*. Thus, for a two-stage SP with 2 possible realizations of a stochastic parameter, there will be one node at time period 1, and two nodes at time period 2, where the variables in the last two nodes represent the same decision made in the face of different data.

Given the set of nodes and the variables belonging to them, we partition the set of constraints or equations in the problem according to this rule: a constraint belongs to the node of latest time stage containing a variable used in the constraint. Put more simply, the constraint goes in the latest node possible. Note that for valid stochastic programs, this rule is not ambiguous: a constraint cannot involve variables in different nodes of the same time period. Thus, a two-stage SP may have constraints involving only those variables from the node at time period 1 (e.g. initial investment limitations, etc.) that are also assigned to this node, and constraints involving first- and second-stage variables (e.g. recourse constraints) that are assigned to the nodes in time period 2.

Once all variables and equations are partitioned among the nodes, we can define an *ancestor relationship* among the nodes as follows: node i is an ancestor of node j if an equation in node j uses a variable in node i . This ancestor relationship does not explicitly define a tree, but in a valid stochastic program, it does so implicitly. The *root* of the tree is the node (at time period 1) that has no ancestors, while the nodes at time period 2 are descendants of this root node and of no others, so they can be made children of the root.

4.1 Intended Use

The GAMS/OSLSE link is intended to be used for stochastic models with 2 or more stages, where decisions (variables) from one time stage determine the feasibility or optimality of decisions made in later time stages. The key requirement is that the modeler group the variables and constraints by node. If this is done correctly, the link takes care of all the other details. The quickest and easiest way to begin using the GAMS/OSLSE link is to meet this key requirement and follow the examples below. Those wishing to push the limits of the link or to understand the error messages for bad formulations, etc., will want to read the next subsection on Model Requirements more carefully.

The modeler must identify one set in the GAMS model as the *node set*. This is done by using the descriptive text `nodes` (not case sensitive) for the set in question. The set name and set elements can be any legal GAMS values. Each equation and variable in the model (with the exception of the objective) can then be indexed by this node set, so that the link can partition the set of rows and columns by node and construct an event tree for the model. Note that model formulation that results is completely general, so that it can be passed to any LP solver. Only the OSLSE solver is making use of this special partitioning information.

As an example, consider the simple two-stage model (the stochastic nature is suppressed for simplicity):

$$\begin{aligned} \max \quad & x_i \quad i = 1, 2 \\ \text{s.t.} \quad & x_{root} \leq 1 \\ & x_i = x_{root} + 1, \quad i = 1, 2 \end{aligned}$$

An equivalent GAMS model is the following:

```
set
  N 'nodes' / root,one,two /,
  LEAF(N) / one, two /,
```

```

    ROOT(N) / root /;

variables
  x(N),
  obj;

equations
  objdef,
  g(N),
  f(N);

objdef..    sum {LEAF, x(LEAF)} =e= obj;
f(ROOT)..  x(ROOT) =l= 1;
g(LEAF)..  x(LEAF) =e= x('root') + 1;

model eg1 / f, g, objdef /;
solve eg1 using lp maximizing obj;

```

Note that in the model above, every variable except the objective is indexed by the set N , and similarly for the equations. The objective is treated specially, so that it can involve variables from the same time period and different nodes. It is not used in determining the ancestor relationship between the nodes, and in fact, should never be indexed.

It is not strictly necessary to index all variables and equations by node. Any *nodeless* variables or equations (i.e. those not indexed by node) are placed into the root node of the event tree. So, a completely equivalent formulation for the above model would be:

```

set
  N 'nodes' / root, one,two /,
  LEAF(N) / one, two /;

variables
  xr          'root variable'
  x(N),
  obj;

equations
  objdef,
  f,
  g(N);

objdef..    sum {LEAF, x(LEAF)} =e= obj;
g(LEAF)..  x(LEAF) =e= 1 + xr;
f..        xr =l= 1;

model eg1 / f, g, objdef /;

option lp = oslse;
solve eg1 using lp maximizing obj;

```

There are a number of stochastic programming examples on the OSLSE Web page. Formulations of these and other models in GAMS and suitable for use with GAMS/OSLSE can be downloaded [here](#).

4.2 Model Requirements

While there is an implicit event tree in all stochastic models of the type we are considering, it is not necessary for the modeler to construct this tree explicitly. In some cases, it is good modeling practice to do so, but this is not a requirement made by GAMS/OSLSE. All that is necessary is that (some) variables and constraints be indexed by node. The link will extract the ancestor relationships between the nodes by looking at the constraints and the variables they use. From these ancestor relationships, a tree is formed, and the resulting model is passed to and solved by the routines in OSLSE.

The above process can fail for a number of reasons. For example, the ancestor relationships between the nodes could be circular (e.g. $A \rightarrow B \rightarrow A$), in which case a tree cannot exist. Such a model will be rejected by GAMS/OSLSE. At the opposite extreme, the model could consist of many independent nodes, so that no ancestor relationships exist. In this case, each node is the root of a one-node tree. The link will recognize these multiple roots and create an artificial super-root that adopts each of the root nodes as its children, so that we have a proper tree.

The OSLSE routines make some other assumptions about the correct form of the model that are checked by the link. Each scenario must be complete (i.e. must end in the final time stage), so that all leaf nodes in the tree exist at the same level. Furthermore, each node in a given time stage must have the same nonzero structure. This implies that the number of rows, columns, and nonzeros in these nodes is the same as well. If any of these counts differ, some diagnostic messages are printed out and the solve is terminated.

There are very few requirements put on the form of the objective function. It must be a scalar (i.e. unindexed) row with an equality relationship in which the objective variable appears linearly. This objective variable must not appear in any other rows of the model. Variables from any node of the model may appear in this row, since the objective row will be removed from the model before the model is passed on for solution.

5 Future Work / Wish List

In this section we list features and functionality that are either incomplete, unfinished, or not yet implemented. They are not listed in any particular order, nor are they necessarily going to be part of any future release. We invite your comments on the importance of these items to you.

Currently, it is not possible to "warm start" the stochastic solvers in OSLSE (i.e. they do not make use of any current information about a basis or level or marginal values).

Currently, it is not possible to return complete basis information to GAMS. The basis status is not made available by the OSLSE link. This is evident on degenerate models, for example: when x is at bound and has a zero reduced cost, is x in the basis or not?

The current link does not allow the user to specify an iterations limit or a resource limit, or to cause a user interrupt. The solution subroutines do not support this. Also, the iteration count is not made available.

When the OSLSE solution routines detect infeasibility or unboundedness, no solution information is passed back (i.e. levels and marginals). This is reported properly to GAMS via the model status indicator.

It is not possible to solve a "trivial" stochastic program, i.e. one with only one stage. The solution subroutines do not allow this. This should be allowed in a future release of OSLSE. Currently, the link rejects single-stage models with an explanatory message.

The number of rows in any stage must be positive. The link checks this and rejects offending models with an explanatory message.

The nonzero structure of each node in a stage must be identical. This is a requirement imposed by OSLSE. It would be possible to construct, in the link, a nonzero structure consisting of the union of all nodes in a stage, but this has not been done in this version. A workaround for this is to use the EPS value in the GAMS model to include zeros into nodes where they did not exist (see the example models for more detail).

6 GAMS Options

There are a number of options one can set in a GAMS model that influence how a solver runs (e.g. iteration limit, resource limit). The relevant GAMS options for the OSLSE link are:

```
option iterlim = N;      is used to set the iteration limit. Not clear yet whether this is a limit on Bender's
                        iterations, total pivots, or what.
option reslim = X;      is used to set a resource limit (in seconds) for the solver. Currently not imple-
                        mented.
```

7 Summary of GAMS/OSLSE Options

The GAMS/OSLSE options are summarized here. The section of detailed descriptions can be found later in this document.

Option	Description
<code>benders_major_iterations</code>	limits # of master problems solved
<code>cut_stage</code>	earliest stage not in master
<code>ekk_crsh_method</code>	OSL crash method
<code>ekk_nwmt_type</code>	OSL matrix compression
<code>ekk_prsl</code>	perform presolve on submodels
<code>ekk_prsl_type</code>	controls reductions attempted in presolve
<code>ekk_scale</code>	perform scaling on submodels
<code>ekk_sslv_algorithm</code>	simplex variant for initial submodel solution
<code>ekk_sslv_init</code>	simplex option for initial submodel solution
<code>large_master_bounds</code>	avoid unboundedness, large swings in master solution
<code>maximum_submodels</code>	limits amount of decomposition
<code>nested_decomp</code>	use nested Benders
<code>optimality_gap</code>	termination criterion
<code>print_tree_all</code>	print nodes, rows, and columns
<code>print_tree_cols</code>	print list of columns in model, grouped by node
<code>print_tree_nodes</code>	print list of nodes in model
<code>print_tree_rows</code>	print list of rows in model, grouped by node
<code>submodel_row_minimum</code>	minimum # of rows in any submodel
<code>write_spl_file</code>	name of SPL file to write

8 The GAMS/OSLSE Log File

This section will not be implemented until the log file format is "finalized".

9 Detailed Description of GAMS/OSLSE Options

The options below are valid options for GAMS/OSLSE. To use them, enter the option name and its value, one per line, in a file called *oslse.opt* and indicate you would like to use this options file by including the line `<modname>.optfile = 1;` in your GAMS input file or using the `optfile=1` argument on the command line. Is the name of the options file case sensitive? It is only necessary to enter the first three letters of each token of an option name, so the option `benders_major_iterations` can be entered as `ben_maj_iter`. Comment lines (those beginning with a `#`) and blank lines are ignored.

Option	Description	Default
<code>benders_major_iterations</code>	At most this many iterations of Bender's will be done. Another stopping criterion for Bender's is the optimality gap; usually, the gap is reduced to this tolerance before the iteration limit is reached.	30
<code>cut_stage</code>	All nodes prior to this time stage are placed in the master problem, while the nodes in this time stage or later are placed into the sub-problems. The root node is assumed to be at time stage one. Note that this option keeps nodes in the master, but cannot necessarily keep them out. For example, the master problem will always include one entire scenario, and other scenarios can be added to the master as a result of other options settings, e.g. <code>submodel_row_min</code> or <code>maximum_submodels</code> .	2
<code>ekk_crsh_method</code>	Sets type of OSL crash to perform before first solution of submodels. 0: No crash is performed. 1: Dual feasibility may not be maintained. 2: Dual feasibility is maintained, if possible, by not pivoting in variables that are in the objective function. 3: Dual feasibility may not be maintained, but the sum of the infeasibilities will never increase. 4: Dual feasibility is maintained, if possible, by not pivoting in variables that are in the objective function. In addition, the sum of the infeasibilities will never increase.	3
<code>ekk_nwmt_type</code>	Controls whether a new, compressed version of the submodel matrices will be created prior to calling the LP submodel solver. 0: Do not compress. 1: A copy of the matrix is stored by indices. 2: A copy of the matrix is stored by columns. 3: A copy of the matrix is stored in vector block format. 4: A copy of the matrix is stored in vector-block-of-1's format (if the matrix is composed of all 0's and 1's). 5: A copy of the matrix is stored in network format (multiple blocks are compressed into a single network).	1
<code>ekk_prsl</code>	Perform presolve for each submodel of the decomposed problem. Only significant for when using the non-nested decomposition solver (i.e. when <code>nested_decomp</code> is set to no, its default value).	no
<code>ekk_prsl_type</code>	Controls type of presolve reduction to perform. 0: Redundant rows are eliminated, positive variables summing to zero are fixed. 1: Type 0 reductions, and the doubleton rows are eliminated. 2: Type 0 reductions, and variable substitutions performed. 3: Type 0, 1, & 2 reductions are done.	3
<code>ekk_scale</code>	Perform scaling for each submodel of the decomposed problem.	no

Option	Description	Default
<code>ekk_sslv_algorithm</code>	Specifies variant of simplex algorithm to be used for first solution of submodels. 0: Variant chosen automatically. 1: Primal Simplex. 2: Dual Simplex.	1
<code>ekk_sslv_init</code>	Simplex option. 0: Use an existing basis (not recommended). 1: Devex pricing is used. 2: Random pricing is used for the first "fast" iterations, then a switch is made to Devex pricing. 3: Use previous values to reconstruct a basis. Note: The dual simplex method can only have options 0 or 1.	2
<code>large_master_bounds</code>	If set to 1, large bounds are placed on the variables in the master problem. This can help to keep the solutions of the master problem from varying too wildly from one iteration to the next.	0
<code>maximum_submodels</code>	The problem will be decomposed into at most N submodels. Once this limit is reached, the decomposition code assigns subproblems or nodes to existing submodels. Note that for the (default) nested decomposition solver, the settings of <code>submodel_row_minimum</code> and <code>cut_stage</code> take precedence.	100
<code>nested_decomp</code>	If yes, use the nested Benders decomposition solution subroutine. Otherwise, use the straight Benders routine.	yes
<code>optimality_gap</code>	The decomposition solvers will termination when the optimality gap is less than this tolerance.	1e-8
<code>print_tree_*</code>	Prints information about the event tree on the listing file. The node listing indicates what nodes are in the tree, their parents and children, and what GAMS label is used to index a given node. The row and column listings, printed separately from the node listing, print a list of the rows and columns of the GAMS model, grouped by node.	no
<code>submodel_row_minimum</code>	If the argument $N > 0$, nodes are added to a submodel until the number of rows in the submodel exceeds N. This option applies only to the nested decomposition solver. If $N > 0$, it overrides the setting of <code>cut_stage</code> and <code>maximum_submodels</code> .	0
<code>write_spl_file</code>	Once the problem has been passed successfully to the OSLSE routines, the problem will be written to a file of the name given as the argument to this option keyword. The filename is not given an SPL extension, but is taken as-is. The SPL file format is a format internal to OSLSE used for compact storage of a particular model.	SPL file not written by default