Bruce McCarl's GAMS Newsletter Number 34

This newsletter covers

Updates to Expanded GAMS User Guide by McCarl et al.

I updated the Expanded User's Guide to reflect release 24.2 with changes added here and there. The latest can be found at http://www.gams.com/dd/docs/bigdocs/gams2002/mccarlgamsuserguide.pdf and will be in upcoming GAMS releases.

New GAMS features in release 24.2

Including probability distributions

GAMS has introduced the ability to include probability functions and inverse probabilities in models and calculations. In particular one can use the extrinsic libraries cppcclib, stodclib and lsadclib which provide random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for a number of distributions. The included distributions across all of these are are Beta, Cauchy, ChiSquare, F, Exponential, Gamma, Gumbel, Inverse Gaussian, Laplace, Logistic , Log Normal, Univariate and Bivariate Normal, Pareto, Rayleigh, Student's t, Triangular, Uniform and Weibull plus the discrete distributions Binomial, Geometric, Hypergeometric, Logarithmic, Negative Binomial, Poisson and Uniform Integer. More details on the preprogrammed functions appear in the GAMS User's Guide (see appendix J).

Other language additions

- A feature was introduced that does domain checking when things are loaded from a GDX file at execution time. This is called EXECUTE_LOADDC and generates execution errors when items from the loaded GDX file contain items not in the domain of the items being loaded.

- A feature was introduced that unloads not only requested items but also all of the sets used in this domain. This is called EXECUTE_UNLOADDI.
  - One can use load to define a set based on the elements with non zero entries in the data. thus if one has a set that is to be defined and we know the parameter a is defined over that set then one can use something like the following syntax

```
set i
parameter a(i)
$gdxin trannoset
$load i<a
```

  - One can dump all options and their current settings using the option DmpOpt.
  - Model status 7 which used to be Intermediate Nonoptimal to was renamed to Feasible Solution.
  - The Alias statement mow works on multidimensional sets as well

Solvers

- New libraries are included for ALPHAECP, Baron, Bonmin, CBC, Couenne, Cplex/CplexD, CONOPT, DICOPT, EMPSP, GloMIQO, Gurobi, Ipopt, IpoptH, JAMS, KNITRO, Lindo, LindoGlobal, MOSEK, MSNLP, OQNLP, Osi, SCIP, SULUM and XPRESS

Using the GUSS "solver"

GUSS is a GAMS facility that permits solution of a set of scenarios for a GAMS model modifying data to run each scenario. GUSS allows the collection of models to be solved in a single pass without needing repeated solves or a LOOP over multiple solves. GUSS is not really a solver but rather organizes and passes data to the other gams solvers for most model types. This is all done in a faster fashion than say when using multiple solves through the GAMS Loop command and is much faster for small models.

In particular GUSS runs the model repeatedly over user specified data for model parameters that collectively define alternative scenarios to be run. In doing this it repeatedly updates the base model with the altered scenario data, then solves the updated model for that scenario and saves user chosen results for each scenario.

GUSS was developed by Michael R. Bussieck, Michael C. Ferris, and Timo Lohmann. It is documented in http://www.gams.com/modlib/adddocs/gusspaper.pdf and in the GUSS section of the solver manual.

GUSS is available in all versions of GAMS starting with release 23.7.

Use of GUSS for an existing model requires six steps

1. Definition of scenarios to run
2. Definition of parameters holding scenario specific data for the items in the model that to be changed
3. Definition of parameters that will hold scenario specific model results for the items that the user wished to save
4. Definition of a set that tells GUSS the scenarios to run, data to change and results to save
5. Modification of the solve statement to identify that scenarios will be run
6. Development of code to report the scenario results

Each will be covered below.in the context of the model in risk.gms which originally solved a model repeatedly for different risk aversion parameters using the code below

```
loop (raps,rap=riskaver(raps);
        solve evportfol using nlp maximizing obj ;
        var = sum(stock, sum(stocks,
           invest.l(stock)*covar(stock,stocks)*invest.l(stocks))) ;
        output("rap",raps)=rap;
        output(stocks,raps)=invest.l(stocks);
        output("obj",raps)=obj.l;);
```

We will now discuss the 6 steps of set up for an example based on the risk model which we callt GUSSRISK.gms.

1. Definition of scenarios to run

The first step in the procedure is to establish a set that covers the scenarios that will be run. For the risk.gms example (from the Expanded user guide) the alternative runs are ones for different risk aversion parameters controlled by the set RAPS. So in GUSSRISK.gms (accessible in the Expanded user guide) we will define a set of risk aversion parameters and give it the name **RAPSCENARIOS** in a set statement as follows

```
SET RAPSCENARIOS   RISK AVERSION PARAMETERS /R0*R25/
```

2. Definition of parameters holding scenario specific data

The second step in the procedure is to establish scenario dependent values for the model data items that will be changed across the scenarios run that will be run. For the GUSSRISK.gms example we wish to use scenario dependent risk aversion parameters defined over the set **RAPSCENARIOS**. We do this in a parameter statement as follows

```
PARAMETER RISKAVER(RAPSCENARIOS) RISK AVERSION COEFICIENT
BY RISK AVERSION PARAMETER
        /R0   0.0000000001, R1   0.00025, R2   0.00050, R3   0.00075,
         R4   0.00100, R5   0.00150, R6   0.00200, R7   0.00300,
         R8   0.00500, R9   0.01000, R10  0.01100, R11  0.01250,
         R12  0.01500, R13  0.02500, R14  0.05000, R15  0.10000,
         R16  0.30000, R17  0.50000, R18  1.00000, R19  2.50000,
```

R20 5.00000, R21 10.0000, R22 15. , R23 20.
R24 40. , R25 80./ ;

In general, the parameter storing the data has to have the scenario set in its first index position. Thus if one is altering a scalar like RAP in the GUSSRISK.gms example, we define a one dimensional parameter over the scenario set in this case **RISKAVER(RAPSCENARIOS).**

When the scenario analysis involves modifying a parameter in a model named **modelparam(i,j,k)** one would define a new parameter whith a structe like **newmodelparam(scenarioset,i,j,k)** where **scenarioset** is the set of scenarios that will be handled by GUSS and the i,j,k are the original set definitions in the parameter to be changed..

Note we could have changed more than one parameter but here this is all we will modify. For a more complex example see gussexample1.gms..

3. Definition of parameters to hold scenario specific model results

The third step in the procedure is to establish scenario dependent repositories where the scenario dependent model solution related items will be stored. For the GUSSRISK.gms example we will store the levels of investment, the objective function value and the available funds shadow price. Each of these items has to have the named **scenario set in the first index position** plus the full dimension of the associated solution information. In the GUSSRISK.gms example this is **RAPSCENARIOS** and we use a parameter statement as follows

```
PARAMETER
STOCKOUTPUT(RAPSCENARIOS,STOCKS) RESULTS FOR INVEST with
VARYING RAP
OBJLEVEL(RAPSCENARIOS) OBJECTIVE FUNCTION WITH VARYING RAP
INVESTAVshadow(RAPSCENARIOS) FUNDS SHADOW PRICE WITH
VARYING RAP
;
```

One can also specify a parameter to hold solution status information relative to each model. In that case we specify the nature of the information we want and the array name to hold the information again with the scenario set name in the first index position. In the GUSSRISK.gms example this involves the statements

```
Set       modelattrib model solution information to collect     / modelstat, solvestat,
objval /;
PARAMETER   solutionstatus(RAPSCENARIOS, modelattrib) Place to store
Solution status reporting
*           assign initial values
          / #RAPSCENARIOS.(ModelStat na, SolveStat na, ObjVal na) /;
```

Where

- the first line defines a set that contains the names of the model attributes to store using in this case the attributes for
    - model solution status (modelstat with an explanation of the possible numerical values given here)
    - solver solution status (solvestat with an explanation of the possible numerical values given here)
    - the optimal value of the objective function (objval)
    - Note more items can be stored and are domusd, iterusd, objest, nodusd, numnopt, numinfes, robj, and suminfes as mostly defined in the list of model attributes here
- the second line defines the parameter in which the values are to be stored
- The fourth line initializes all values to na and if the solves fail then those values will remain.

4. Definition of a set that tells GUSS what to do

The fourth step in the procedure is to establish a set statement in the form of a three dimensional tuple that tells GUSS what you wish to do plus possibly defining a parameter holding GUSS option settings.

The tuple contains

- the name of the set defining the scenarios
- the names of model parameters to be changed and the name of the parameters where the scenario dependent data are stored
- the names of model solution parameters to be saved along with identification of the type of the solution information to save and the name of a the place where to save it.
- the names of a parameter with options to pass to GUSS along with the name of the place to store solution attributes

For the GUSSRISK.gms example the statement is as follows.

```
set GUSSdict   / RAPSCENARIOS.scenario   . ''
                 rap        .param      .RISKAVER
                 INVEST     .level      .STOCKOUTPUT
                 OBJ        .level      .OBJLEVEL
                 INVESTAV   .marginal   .INVESTAVshadow
               /;
```

Here the named set is GUSSDICT and

- the first line identifies the name of the set defining the scenarios (**RAPSCENARIOS**)and associates it with the word **SCENARIO** and a third entry of **' '**
- the second line identifies the name of the data element in the model (**rap**) to be changed in running the scenarios and associates it with the word **param** and an entry telling where the alternative values are held (**RISKAVER**)

- the third line identifies the name of a solution output to store (**INVEST**), its nature (a **level** or **INVEST.L** in this case) and the place to store it (**STOCKOUTPUT**)
- the fourth line identifies the name of a solution output to store (**OBJ**), its nature (a **level** or **OBJ.L** in this case) and the place to store it (**OBJLEVEL**)
- the fifth line identifies the name of a solution output to store (**INVESTAV**), its nature (a **marginal** or **INVESTAV.M** in this case) and the place to store it (**INVESTAVshadow**)

Note the key words that can be used in the second tuple position are

param	Indicating this is an item that provides scenario data for a model parameter that will be altered
lower	Indicating this is an item that provides alternative lower bounds for model variables or equation RHS's that will be changed
upper	Indicating this is an item that provides alternative upper bounds for model variables or equation RHS's that will be changed
fixed	Indicating this is an item that provides alternative fixed bounds for model variables or RHS's that will be changed
level	Indicating this is an item that will be used to store solution levels for model variables or equations
Marginal	Indicating this is an item that will be used to store solution marginals for model variables or equations
opt	Indicating the parameter holding GUSS options to use and where to store model solution attributes

One may also modify multiple input parameters as in the tuple specified in gussexample1.gms where parameters a and b take on multiple values.

```
set dict   / scenariostorun.scenario  .''
             gussoptions   .opt       .solutionstatus
             a             .param     .newsupply
             b             .param     .newdemand
             x             .level     .resultantx
           /
```

If one uses the opt command one also needs to specify a parameter that holds options for GUSS using syntax like

```
parameter gussoptions  options to use in running GUSS
                     / UpdateType 1, Optfile 1 /
```

The available options are discussed in the solver manual in the GUSS section with the most important ones involving option files to use controlling the option files to use for the first and subsequent solves, the amount of output in the LOG file, the way the data update is performed and the type of solution point to restart from.

5. Modification of the solve statement to identify that scenarios will be run

The fifth step involves altering the solve statement so it both knows that GUSS is to be used plus an identification of the name of the tuple that passes instructions on what GUSS needs to do.

The format of this in the GUSSRISK.gms example is

SOLVE EVPORTFOL USING NLP MAXIMIZING OBJ SCENARIO GUSSDICT ;

where the solve statement is of the conventional form with the addition of the key word **SCENARIO** and name of the tuple from step 4 that tells GUSS what to do. In this case the name of that tuple is **GUSSDICT.**

In the gussexample1.gms case the solve statement is

Solve transport using lp minimizing z scenario dict;

where again we have the addition of the key word **SCENARIO** and **DICT** is the name of the tuple that tells GUSS what to do**.**

6. Development of code to report the scenario results

The sixth step involves implementing post solution instructions to report the scenario dependent family of solutions to the user. This is done either directly through a display or through calculation of tables and inclusion in output through display, put files or passing to other programs as discussed elsewhere in this guide.

In the example in gussexample1.gms we simply display the array

```
option resultantx:0:1:2;
display resultantx,solutionstatus;
```

In GUSSRISK.gms we run through a report writing loop placing the scenario dependent solution information into the model variable levels and shadow prices and then build a report table

```
 PARAMETER OUTPUT(*,rapscenarios);
 LOOP (RAPSCENARIOS,RAP=RISKAVER(RAPSCENARIOS);
*        LOAD IN SOLUTION INFORMATION
         INVEST.L(STOCKS)=STOCKOUTPUT(RAPSCENARIOS,STOCKS);
         OBJ.L=OBJLEVEL(RAPSCENARIOS);
         INVESTAV.m=INVESTAVshadow(RAPSCENARIOS);
*        COMPUTE SOME ITEMS
         INVESTAV.L=SUM(STOCKS,INVEST.L(STOCKS));
         VAR = SUM(STOCK, SUM(STOCKS,
```

```
                INVEST.L(STOCK)*COVAR(STOCK,STOCKS)*INVEST.L(STOCKS)))
;
          OUTPUT("RAP",rapscenarios)=RAP;
          OUTPUT(STOCKS,rapscenarios)=INVEST.L(STOCKS);
          OUTPUT("OBJ",rapscenarios)=OBJ.L;
          OUTPUT("MEAN",rapscenarios)
             =SUM(STOCKS, MEAN(STOCKS) * INVEST.L(STOCKS));
          OUTPUT("VAR",rapscenarios) = VAR;
          OUTPUT("STD",rapscenarios)=SQRT(VAR);
          OUTPUT("SHADPRICE",rapscenarios)=INVESTAV.M;
          OUTPUT("IDLE",rapscenarios)=FUNDS-INVESTAV.L
           );
DISPLAY OUTPUT,solutionstatus;
```

Here we loop over the scenarios run (**RAPSCENARIOS)**and during that loop we load the GUSS saved investment levels into the original model investment variables using the statement

```
INVEST.L(STOCKS)=STOCKOUTPUT(RAPSCENARIOS,STOCKS);
```

along with the saved scenario dependent values of the objective function and the funds shadow prices.

```
OBJ.L=OBJLEVEL(RAPSCENARIOS);
INVESTAV.m = INVESTAVshadow(RAPSCENARIOS);
```

Finally in the loop a number of calculations are done placing results into a parameter named OUTPUT and after the loop the result is displayed as is the array holding the solution and model termination status.

Much more complex setups could be run.

Notes

- GUSS is not a solver and is not activated using normal solver choice methods such as option LP=GUSS or any such variants. Rather one uses a modification to the solve statement as discussed above.
- GUSS will cause the problems to be solved with the solver that is currently active in the GAMS instance. This may be specified using multiple ways as discussed here employing for example OPTION NLP=CONOPT or LP=CPLEX or the like as well as through choice in the IDE or on the computer.
- GUSS has a number of additional options as discussed in http://www.gams.com/modlib/adddocs/gusspaper.pdf and in the GUSS section of the solver manual.

Courses offered

I will be teaching

- Basic to Advanced GAMS class Aug 4, 2014- Aug 8, 2014 (5 days) in the Colorado mountains at Frisco (near Breckenridge). The course spans from Basic topics to an Advanced GAMS class. Details are found at http://www.gams.com/courses/basic_and_advanced.pdf .
- Basic GAMS class Aug 4, 2014- Aug 6, 2014 (3 days) in the Colorado mountains at Frisco (near Breckenridge). The course starts assuming no GAMS background. Details are given at http://www.gams.com/courses/basic.pdf .
- Advanced GAMS class Aug 6, 2014- Aug 8, 2014 (3 days) in the Colorado mountains at Frisco (near Breckenridge). The course is for users, who have a GAMS background. Details are found at http://www.gams.com/courses/advanced.pdf .

Further information and other courses are listed on http://www.gams.com/courses.htm . Note I also give custom courses for individual groups a couple of times a year.

Unsubscribe to future issues of this newsletter

Please unsubscribe through the web form available at:
http://app.streamsend.com/public/XLmY/5eq/subscribe

This newsletter is not a product of GAMS Corporation although it is distributed with their cooperation.

April 7, 2014