Bruce McCarl's GAMS Newsletter Number 39

This newsletter covers

1 Updates to GAMS User Guide by McCarl et al.

I updated the Expanded User's Guide to reflect version 24.7, some miscellaneous items and the items. This included covering =b= types of constraints, the distributed processing from below, the GDX to GMS capability through GDXDUMP, and GMSZIP/GMSUNZIP capabilities.

The updated version will be available after the next maintenance or full release on line at http://www.gams.com/help/index.jsp?topic=%2Fgams.doc%2Fuserguides%2Fmccarl%2Findex.html or in the next GAMS release.

2 Updates to GAMSCHK

After years with nothing being done to GAMSCHK a few issues were fixed by Steve Dirkse and me. These involved fixing

1) The way scaling and descaling are handled was changed. At some point in the past GAMS changes rendered the output on models that were being scaled to not be handled as stated in the GAMSCHK user guide. The code was fixed to do this properly (giving the user the choice of seeing scaled or original data plus showing POSTOPT in descaled fashion by default).
2) Some output characteristics of the GAMSCHK ANALYSIS procedure were modified. When ANALYSIS executes and it finds model formulation errors it then uses DISPLAYCR to print them out. In order to avoid large LST files the code was programmed to limit the cases shown for a column or row block to the first three cases of particular error but under certain circumstances hundreds could appear. We repaired this so at most three cases are output.
3) Updating of the Jacobian to reflect any solutions done. GAMSCHK can be used to do post optimality displays of nonlinear models. To do this it uses the matrix of first derivatives (the Jacobian). When a model is set up the Jacobian sent to a solver is the one that has been evaluated at the starting point and GAMSCHK was not programmed to update this to a

current solution. GAMSCHK POSTOPT did not update to reflect the solution point and the derivatives only reflected the starting point. Now the Jacobian is updated to the results of any solves that are done.

3 SCIP solver commercial license

A commercial license is now available for the SCIP solver, which solves Constraint Integer Programming. According to the SCIP web page, it is a standalone solver for linear programming (LP), mixed integer programming (MIP), and mixed integer nonlinear programming (MINLP). More can be found on the SCIP at http://scip.zib.de/#about .

4 Graphical Interface Generator

Wolfgang Britz has developed a GAMS Graphical Interface Generator (GGIG) for use with GAMS programs. GGIG generates a basic graphical user interface (GUI) which allows one to operate GAMS models models through a Java based user interface. The documentation indicates GGIG supports 5 main functions:

1. Generation of user operable graphical controls from XML based definitions where the user can then interact with the GUI to change the state of the controls and can map the controls to GAMS code.
2. Generation of GAMS compatible data from the state of the control which can be stored in GDX format.
3. Execution of a GAMS program while passing the state of the control to GAMS as an include file.
4. Exploitation of results from GAMS runs by providing an interface to define the necessary interfacing definitions to load results from a GAMS into the CAPRI exploitation tools.
5. Access to GAMS related utilities for model analysis. These include a viewer for GDX files, a utility to build a HTML based documentation of the GAMS code and a batch execution utility.

GGIG is currently used in a number of economic and agricultural modeling projects. GGIG also supports applications using R and Java. Details, code and documentation on GGIG is present at http://www.ilr.uni-bonn.de/agpo/staff/britz/ggig_e.htm.

5 Distributed Processing

GAMS model instances can take a long time to solve and one may need to solve the model for a number of scenarios with the result taking weeks or months. GAMS has the GRID computing facility but on PCs use over a network of machines can require substantial IT involvement often rendering such an implementation to be impractical. Recently I had a many scenarios to run for a time consuming models and I wrote a semi-automated procedure to share the work across different machines then merge the results. I then wrote a simple example of the procedure and improved it with some help from GAMS personnel. Basically, the code (merge_control_example.gms) writes GAMS script files to run a distributed set of jobs on

multiple processors/machines involving creating GDX files to pass scenario data and return scenario results plus writing a GAMS file that runs the scenarios. The procedure also compresses the files for the remote machines into a zip file for transfer. A run of merge_control_example.gms also writes a file merge_mergethem.gms that merges all into a unified file and a script file that runs it.
The example extends the AGRESTE.GMS model library file and is available at http://www.gams.com/mccarl/newsletter/examples39.zip .

The core code - merge_control_example.gms creates all files used in the whole process. It is AGRESTE context specific with AGRESTE related data handling, reporting and scenario development but also many generic features. Within that code the user tells: a) how to allocate scenarios to processors, and b) what data to save and load. Then it writes script files for each processor plus a processor specific GDX file containing the scenario definition data. It also writes a file that runs the scenarios. In turn it zips the all the relevant files for transfer to the remote machine.

On the distributed processors/machines, the user needs to move in the zip files and extract then then initiate the scrip file whereupon the scenarios are run and as they finish the results are automatically unloaded into GDX files. Then the user needs to transfer these GDX files back to a location on which the data will be merged by running merge_mergethem.gms.

Operationally in the main merge_control_example.gms file we

1. Define the scenarios and in this agreste example link them to some alternative manipulations of the agreste price, land and risk aversion data (see section 1 in the code).
2. Define the processors to be used and assign scenarios to be run by each processor (see section 2)
3. Give a name to the procedure that will run the scenarios on the assigned processors (in this case agreste_loop - see section 3)
4. Identify which report parameter that reflect the solution for each scenario are to be merged (see section 4).
5. Set up parameters for the data items and sets that define the scenarios (see section 5)
6. Create processor specific GDX files that contain scenario data to be run and processor specific script files plus create a zip file to pass to the processor (Section 6).
   There are a number of subparts to this step. In particular, for each processor we
   a. Set up scenario data and put it in an appropriately named GDX file (section 6a)
   b. Create a script file for that processor (section 6b)
7. Create GAMS code in the file merge_mergethem.gms that will merge the results (section 7).
   Here there are several steps involved
   a. We define a control variable that contains the names of the report items to merge - section 7a.
   b. We declare the parameters and sets that are involved with holding the scenario dependent reports to be merged (not needed if restarting) - section 7b.

   c. We include a statement to load the GDX file data from each processor using $loadm so the data will be merged - section 7c.
   d. We form a set that tells what scenarios have been found - section 7e.
   e. We display merged results and write a GDX file of them - section 7f
8. Get agreste.gms from the library and run it to create a restart file - section 8.
9. Create a GMS file that runs the scenarios called agreste_loop - section 9. In doing this we
   a. Set up scenario particulars - section 9a
   b. Define the names of the cross scenario reports that are computed and subsequently passed on for use in the merge operations - section 9b.
   c. Load in data that tells what scenarios to use and accompanying data - section 9c.
   d. Loop through the scenarios assigned to a processor setting it up, solving it and computing reports on the results - section 9d.
   e. Unload data into the GDX file for subsequent use in the merge operation - section 9e

B. In doing this merge control_example.gms writes several types of files

1. Files for use on each of the remote processors
   a. A script file for that processor with a name like script_for_secondprocessor.gms that when run executes all the GAMS tasks needed to be done on that processor. One will be generated for each processor identified in the runstodo table.
   b. A GDX file for each processor with the scenario data named send_to_processorname.GDX or in the example send_to_secondprocessor.GDX
   c. A gms file that is used on all remote processors to runs the scenarios called agreste_loop.gms.
   d. A restart file that is created after agreste is run (a1.g00).
   e. A zip file that contains the 4 files just above for each processor called zip_to_send_to_processorname.zip or in the example zip_to_send_to_Secondprocessor.zip.
2. A file for the merge operation named merge_mergethem.gms that merges the data from the distributed runs assuming the GDX files of results have been moved to the project directory.

C. On the remote processor one needs to

1. Copy in and unzip the file zip_to_send_to_processorname.zip. In this example one file is named zip_to_send_to_processorname.zip. This unpacks the script file (script_for_second_processor.gms), a1.g00, the agreste_loop.gms file and the send_to_processorname.GDX (e,g, send_to_secondprocessor.GDX)
2. Run the script file which causes a load of the GDX file of data, a run of agreste_loop.gms executing all assigned scenarios, creation of scenario dependent reports and an unloading of the results in a GDX file for use in the merge exercise. (results_processorname.GDX or in the example results_secondprocessor.GDX)
3. Move the results GDX file to the location where the merge is to be run

D. Then back on the main machine

1. One makes sure the results_procssorname.GDX files have been moved.
2. Run the merge_mergethem.gms to obtain the merged results. These will appear in merge_mergethem.lst or in results_agreste.GDX.
3. Note this procedure can be run at any time even if all the GDX files are not present.

A few notes

- The GAMS command **execute_unload** is used to place the results into GDX files since they only exist at execution time.
- The code for loading from the GDX files uses the GAMS command **$loadm** to merge the results as that command augments the earlier information merging in the data.
- merge_mergethem when executed does the merger and it can be followed by something else to create user reports. For this you may want to restart it from an agreste , agreste_loop or merge_control_example restart file.
- Much more could be done with this. For example, one could insert commands that copy the zip file to the remote processors on a network, then unzip and run it. Additionally one could write code to copy back the GDX files of results. One can also write a procedure that waits for the GDX files to be present before running the merge_mergethem.gms file.

6 Transforming GDX data into a GMS file

One can take the data in a GDX file and put it into a GMS source file using GDXDUMP including domains and the parameter values. Consider the following example

```
*load in the transport model
$  call gamslib trnsport

*execute the model and create a GDX file
$  call gams trnsport lo=%gams.lo% GDX=trnsport

* Create GAMS source that declares symbols and data
$call GDXdump trnsport.GDX NoData > newtransport_data_only.gms
```

Here the line `$call GDXdump trnsport.GDX NoData > newtransport_data_only.gms` takes the contents of the GDX file and writes them into the file newtransport_data_only.gms. The resultant file is just sets and parameters. No model declaration statements or calculations will be present.

7 Courses offered

I will be teaching

- Basic to Advanced GAMS class Aug 8, 2016- Aug 12, 2016 (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/mccarl_combined.pdf .

- Basic GAMS class Aug 8, 2015- Aug 10, 2015 (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/mccarl_basic.pdf.
- Advanced GAMS class Aug 10, 2015- Aug 12, 2015 (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/mccarl_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.

8 Unsubscribe or subscribe to future issues of this newsletter

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

Those who wish to subscribe to future issues can do this trough the newsletter section of http://www.gams.com/maillist/index.htm.

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

June 10, 2016