Bruce McCarl's GAMS Newsletter Number 26

Here I cover a number of items including new developments in GAMS 22.9 and mention future course offerings.

Expanded GAMS User Guide by McCarl et al.

I updated the User's Guide. Major sections were added on Utilities, ASK, Macros, Invert, Cholesky, EIGENVALUE, EIGENVECTOR, POSIX utilities, SOLVEOPT=Clear and errors with ORD. A number of other new features from 22.9 were added here and there. This is accessed through Help in the IDE or at http://www.gams.com/dd/docs/bigdocs/gams2002/mccarlgamsuserguide.pdf .

Matrix utilities

GAMS has recently introduced utilities for matrix inversion (INVERT), Cholesky factorization (CHOLESKY), eigenvalues (EIGENVALUE) and eigenvectors (EIGENVECTOR). All are discussed in the newest document on GDX utilities plus in the updated Expanded GAMS User Guide that can be accessed through the IDE help function or through the web.

Unfortunately, I feel that this software has shortcomings. All the utilities require that the matrices plus results be passed through GDX files and that the input matrix be square with the same set defining the rows and columns. These characteristics make the software unnecessarily difficult to use or for inversion of limited usefulness as:

- The utilities require one to write the target matrix to a GDX file, then invoke the utility and finally read the results from a GDX file. I do not think this is the best way to do this and rather the GDX operations should be done automatically by the program in a fashion transparent to the user.
- The commonality of row and column labeling assumption is an unrealistic requirement for a matrix to be inverted. One typically expects the rows and columns to be defined over different sets (like basic variable names for the columns and rows for the rows in a linear programming inverse). Furthermore, on return the inverse would have the column dimension for the rows and the row dimension for the columns.

To overcome the shortcomings I wrote a piece of code for the matrix inversion which allows differing row and column names and does not require the user to write the matrix to be inverted in a GDX file and read the inverse from another one. Rather it embeds the GDX functions within the code much as in the fashion of Tom Rutherford's XLIMPORT.

This code can be accessed through the expanded Expanded GAMS user guide (McCarl) in the form of the utility arealinverter.gms_ along with an example (inverseexample.gms). To find this lookup invert in the index. The arguments for arealinverter.gms are

- it must be called with a $batinclude or a $libinclude
- The call involves four arguments
    - Name of the GAMS parameter that holds the matrix to be inverted
    - Name of the set that defines the rows of the matrix to be inverted
    - Name of the set that defines the columns of the matrix to be inverted
    - Name of the GAMS parameter that will hold the inverse that if the matrix to be inverted is dimensioned A(I,J) is dimensioned AI(J,I);

Note this will call out an error if the matrix is not square but does not do anything on a failure of INVERT. Extensions are welcome.

Macros

GAMS now includes the ability to define macros. The definition takes the form

```
$macro name  macro body
$macro name(arg1,arg3,arg2,..) macro body with arguments arg1,..
```

For example, let us define a simple macro that forms 1 over an item

```
$macro oneoverit(y) 1/y
```

which can be used as follows

```
z = oneoverit(x1)+oneoverit(x2);
```

and will translate (hereafter called expand) into the following within the GAMS code:

```
z = 1/x1 +1/x2;
```

as GAMS recognizes `oneoverit(x1)` as a macro and substitutes x1 for y in expanding it and does the same for x2.

Macros can also have multiple arguments separated by commas

```
$macro ratio(a,b) a/b
```

which when called with

```
z = ratio(x1,x2);
```

will expand into the following within the GAMS code:

```
z = x1/x2;
```

More details on macros are covered in the expanded GAMS guide by going to the index section and making the choice macro. In particular

- Macros can be nested
- Exactly what characters are replaced can be more carefully controlled using ampersands & or && in the macro body.
- Macros can be used in report writing and when a variable is identified then if used in conjunction with the new dollar command $ondotl a .L suffix is automatically added allowing one to use code in the model and the same code without alteration in report writing.
- $show now shows the active macros
- Macro definitions are preserved in a save/restart file and are available again when performing a continued compilation.

Save, replace and at last Clear

I am pleased to report GAMS has introduced a feature that fixes an item I called a bug in a 2001 newsletter. Namely solveopt has a new choice Clear. For years I have experienced a problem that when a model is solved with a variable eliminated that cases can arise where the solution values for the eliminated variables are maintained in GAMS and bias reports generating using variablename.L. This happened whether setting Solveopt=replace or merge but now one can use Solveopt=clear. I am grateful to GAMS for resolving this issue as it has been a frequent irritant for me.

Ordered sets and Compile Errors

Use of the GAMS command ORD sets requires the subject set to be ordered. Ordered sets are typically ones with explicit set elements. Unordered sets generally have elements that are calculated. They also occur when elements in explicit sets appear in different order across more than one set. The last case is illustrated in the following

```
set a        a couple of the elements        /r2,r3/;
set b      more elements                      /r1*r4/;
set a1a    all the elements                   /s1*s10/;
set a1b    some elements in different order   /s3,s1/;
```

where an error would occur if one used ORD on set b or set a1b.
To avoid this all elements would need to appear in the same order with the largest set defined first.

Demise of GAMSBAS

The next release will spell the demise of GAMSBAS, a "solver" that I wrote to save an advanced basis. Today the use of the option savepoint and the Execute_loadpoint options provide superior capabilities (See the sections on a basis or these commands in the expanded GAMS User's Guide for explanation) so I have abandoned use of GAMSBAS and do not oppose dropping it. For those using GAMSBAS if you really want GAMS to retain it you should email support@gams.com with your opinion.

Executing GAMS Batch jobs in the IDE

I frequently find it desirable to run a multistage GAMS job involving saves and restarts. This started when I was largely using Linux but today I am largely on a PC employing the IDE. There are several approaches to run multiple step jobs within the IDE including use of

- An IDE script
- $CALL
- EXECUTE.

The latter two would be implemented operating over either a DOS batch file or a series of commands each individually starting up a GAMS job.
Initially I was quite interested in using IDE script files but unfortunately have discovered that scripts have a number of issues including limiting my ability to save the results of any editing on doing while the job is running. Nor do scripts work very well if I happen to change projects again while a job is running. This is a major issue to me when running a long job in effect in the background on a multiple core PC and caused me to abandon the IDE script files.

I then began to write GAMS jobs like the following

```
    execute "GAMS trandata        s=s1"
    execute "GAMS tranmodl  r=s1   s=s2"
    execute "GAMS tranrept  r=s2"
```

but these runs did not put anything in the log file so I could not tell what their progress was (note I could manually insert log messages as discussed in the expanded user guide under the comparative runs chapter but this did not always get done).

Recently, the people in GAMS helped me solve this using code like the following

```
    $set gamsparm "ide=%gams.ide% lo=%gams.lo% errorlog=%gams.errorlog% errmsg=1"
    execute "GAMS trandata  %gamsparm%        s=s1"
    execute "GAMS tranmodl  %gamsparm%  r=s1   s=s2"
    execute "GAMS tranrept  %gamsparm%  r=s2"
```

The gamsparm item here is what I call a control variable in the expanded GAMS User's Guide and in this particular case it is being set up to tell GAMS that the

- IDE is being used (ide=1 on a PC)
- To write the log to a file which in turn is intercepted by the IDE (lo=3 on a pc)
- To redirect a given number of error messages to that file (errorlog=99 on a pc)

- To place error messages next to the GAMS lines causing them (errmsg=1)

The syntax %gamsparm% is placing the contents of gamsparm in the calling parameters for GAMS. The syntax %gams.ide% picks up the setting of the GAMS command line parameter ide on the computer being used while %gams.lo% and %gams.errorlog% pick up the values of the lo and errorlog GAMS command line parameters.

The same syntax works with $Call.

Features in GAMS 22.9

In the last few days GAMS 22.9 was been released. It has a number of other minor features I did not elaborate on in this newsletter although I have included them in the expanded GAMS User's Guide. Those wishing information on them should examine the release notes.

Courses offered

I teach
Basic GAMS May 19-22, 2009 (3 1/2 days) in the Colorado mountains at Frisco (near Breckenridge). The course is designed for those without GAMS usage experience but has also proved useful for those with years of experience.
Advanced GAMS class Aug 11-14, 2009 (3 1/2 days) in the Colorado mountains at Frisco (near Breckenridge). The course covers such diverse topics as links to other programs like macros, spreadsheets, speeding up GAMS, scaling, debugging, improving output and advanced basis use along with many other topics. Further information and other courses are listed on http://www.gams.com/courses.htm.

Unsubscribe to future issues of this newsletter

To remove your name, please send an email to mccarl-news-request@gams.com containing unsubscribe on the subject line.

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

December 8, 2008