GAMS / MPSWRITE Version 1.0

1.Introduction

The GAMS modeling language allows one to represent a mathematical programming model in a compact and easily understood form and protects the user from having to represent the model in the form that is required by any particular solver. This solver pacific form is usually difficult to write and even more difficult to read and understand. However, there may be cases in which one requires the model data in precisely such a form. One may need to use applications or solvers that are not interfaced to GAMS or else one may need it to test out developmental algorithms either in an academic or industrial setting. MPSWRITE is a subsystem of GAMS that addresses this problem. It is not a solver and can be used independently of them. It can generate the following files for a GAMS model:

- an MPS matrix file
- an MPS basis file
- a GAMS to MPS mapping file
- an ANALYZE syntax file
- an ANALYZE LP file

Many solvers can read the input LP or MIP problem in the form of an MPS file. The MPS matrix file stores the problem matrix as well as the various bounds in a specified format. In this file, names with up to eight characters are used to specify the variables and constraints. In GAMS, however, the variable or constraint name can have as many as 110 characters (10 for the row or column name and 10 for each of the maximum of 10 indices). Therefore, in order to create the MPS file, MPSWRITE must generate unique names with a maximum of eight characters for the variable and constraint names.

The MPS basis file describes the status of all the variables and constraints at a point. It is also written in the MPS format. The mapping file relates the names generated for the MPS matrix and basis files to their corresponding GAMS name.

Since the MPS file format can only handle linear models, nonlinear models are linearized around the current point. The matrix coefficients are then the partial derivatives of the nonlinear terms and the right-hand-sides are adjusted accordingly.

ANALYZE is a system developed by Harvey Greenberg and coworkers at the University of Colorado - Denver for the analysis of linear programming models and solutions. The LP file contains the matrix and solution information for the model while the syntax file is similar in role to the mapping file but in a form recognizable by ANALYZE.

2. HOW TO GENERATE MPS FILES WITH MPSWRITE

MPSWRITE is capable of generating MPS files for models of the following types; `lp`, `rmip`, `mip`, `nlp`, `dnlp`, `rminlp`, `minlp`. In case of NLP models, the MPS file is generated at the current point. One can use the following statement in the GAMS model:

```
option lp = mpswrite ;  {or rmip or mip or nlp or ..}
```

3. AN EXAMPLE

In order to illustrate the use of MPSWRITE add the following lines to [TRNSPORT] just before the `solve` statement

```
option lp = mpswrite ;
```

Although we use the `solve` statement to execute MPSWRITE, no optimization will take place. The current solution values will remain unchanged. By default the MPSWRITE outputs the MPS file, the basis file and the mapping file. The generation of any of these files can be turned off using options in the option file as will be explained later.

Apart from the listing file that reports model statistics, the following files are also generated: by running the model - *`gams.mps`*, `gams.bas`, `gams.map`.

```
NAME          GAMSMOD
* OBJECTIVE ROW HAS TO BE MINIMIZED
* ROWS       :        7
* COLUMNS    :        7
* NON-ZEROES :       20
ROWS
 E  R0000001
 L  R0000002
 L  R0000003
 G  R0000004
 G  R0000005
 G  R0000006
 N  R0000007
COLUMNS
    C0000001  R0000001  -0.225000000
    C0000001  R0000002   1.000000000
    C0000001  R0000004   1.000000000
    C0000002  R0000001  -0.153000000
    C0000002  R0000002   1.000000000
    C0000002  R0000005   1.000000000
    C0000003  R0000001  -0.162000000
    C0000003  R0000002   1.000000000
    C0000003  R0000006   1.000000000
    C0000004  R0000001  -0.225000000
    C0000004  R0000003   1.000000000
    C0000004  R0000004   1.000000000
    C0000005  R0000001  -0.162000000
```

```
    C0000005  R0000003   1.000000000
    C0000005  R0000005   1.000000000
    C0000006  R0000001  -0.126000000
    C0000006  R0000003   1.000000000
    C0000006  R0000006   1.000000000
    C0000007  R0000001   1.000000000
    C0000007  R0000007   1.000000000
RHS
    RHS       R0000002   350.0000000
    RHS       R0000003   600.0000000
    RHS       R0000004   325.0000000
    RHS       R0000005   300.0000000
    RHS       R0000006   275.0000000
BOUNDS
 FR BOUND     C0000007
ENDATA
```

Figure 1: MPS matrix file for trnsport.gms

```
NAME          GAMSMOD
ENDATA
```

Figure 2: MPS basis file for trnsport.gms

```
  6  7
R0000001         COST
R0000002         SUPPLY     SEATTLE
R0000003         SUPPLY     SAN-DIEGO
R0000004         DEMAND     NEW-YORK
R0000005         DEMAND     CHICAGO
R0000006         DEMAND     TOPEKA
C0000001         X          SEATTLE     NEW-YORK
C0000002         X          SEATTLE     CHICAGO
C0000003         X          SEATTLE     TOPEKA
C0000004         X          SAN-DIEGO   NEW-YORK
C0000005         X          SAN-DIEGO   CHICAGO
C0000006         X          SAN-DIEGO   TOPEKA
C0000007         Z
```

Figure 3: GAMS to MPS mapping file for trnsport.gms

By default, the row and column names in the generated files correspond to the sequence number of the row or column and are eight characters wide. The MPS matrix and basis files correspond to the standard format. Note that the basis file is empty since the model has not been solved and the default solution values result in a unit basis. A unit basis means that all rows are basic and all variables are at bounds, the default format for the MPS basis file. We could have solved the model first and then use a second `solve` statement to write the MPS files. The MPS basis file will then contain entries for non-basic rows and basic columns. In the mapping file, the first line gives the number of rows and columns in the model. In the other lines the MPSWRITE generated row and column names is followed by the corresponding GAMS name (first the name of row/column followed by the indices). Since the model is being solved only with MPSWRITE, the listing file does not generate any solution and only reports that the solution is unknown.

Note also that the number of rows in the MPS file is one more than in the GAMS file. This is because GAMS creates an extra row containing only the objective variable. This transforms a problem in GAMS format, using an objective variable, into the MPS format, using an objective row. Finally, it is important to note that MPSWRITE always converts a problem into a minimization form when presenting the model in the MPS file format.

4. NAME GENERATION: A SHORT NOTE

The process involves compressing a name that can potentially be 110 characters long into a unique one of up to sixteen characters for the ANALYZE files and the mapping file and up to eight characters for the MPS and basis files. MPSWRITE allows for two ways of generating these names. The default names correspond to the sequence number of the particular row or column. However, these names, like the ones shown in Fig.1 are not always easy to identify with the corresponding GAMS entity without the aid of the mapping file. MPSWRITE, therefore, also allows for the more intelligent generation of names that correspond more closely with the original GAMS name and are therefore more easily identifiable. The procedure used for generating the names in this form is explained in detail in Section 9.

5. The Option File

The option file is called ***mpswrite.opt***. This file is always read if it exists to avoid certain situations that occur due to adding a solver to the definition of MPSWRITE (as explained later in the chapter). In such a case, if say, BDMLP and MPSWRITE are called within the same solver step, the system does not know which option file to pick up. Therefore, the system always looks for ***mpswrite.opt*** when using MPSWRITE even if the name of the solver has been changed. The option file has the following format: each line contains either a comment (comments have a '*' as the first non-blank character) or an option. For example,

```
* write MPS names with a maximum width of 8 characters
namewid 8
* design name syntax based on GAMS name
nametyp 2
```

The contents of the option file are echoed to the screen and copied to the listing file.

6. ILLUSTRATIVE EXAMPLE REVISITED

Create the option file 'mpswrite.opt' and in it add the following line

```
nametyp 2
```

The option file is explained in greater detail in a following section. The MPS file generated on running the above problem is shown in Fig. 4.

```
NAME          GAMSMOD
* OBJECTIVE ROW HAS TO BE MINIMIZED
* ROWS       :        7
* COLUMNS    :        7
* NON-ZEROES :       20
ROWS
 E  C
 L  SSE
 L  SSA
 G  DNE
 G  DCH
 G  DTO
 N  +OBJ
COLUMNS
    XSENE     C          -0.225000000
    XSENE     SSE         1.000000000
    XSENE     DNE         1.000000000
    XSECH     C          -0.153000000
    XSECH     SSE         1.000000000
    XSECH     DCH         1.000000000
    XSETO     C          -0.162000000
    XSETO     SSE         1.000000000
    XSETO     DTO         1.000000000
    XSANE     C          -0.225000000
    XSANE     SSA         1.000000000
    XSANE     DNE         1.000000000
    XSACH     C          -0.162000000
    XSACH     SSA         1.000000000
    XSACH     DCH         1.000000000
    XSATO     C          -0.126000000
    XSATO     SSA         1.000000000
    XSATO     DTO         1.000000000
    Z         C           1.000000000
    Z         +OBJ        1.000000000
RHS
    RHS       SSE         350.0000000
    RHS       SSA         600.0000000
    RHS       DNE         325.0000000
    RHS       DCH         300.0000000
    RHS       DTO         275.0000000
BOUNDS
 FR BOUND     Z
ENDATA
```

Figure 4. MPS file for trnsport.gms with nametyp=2

Note that the names of the rows and columns are now more easily identifiable with the corresponding GAMS name. For example, the MPS variable `XSENE` corresponds to `x('Seattle','New-York')` and is much more easily identifiable than `C0000001` as in Fig.1. Note that in the newly generated name, the first character `X` represents the variable name `x`, the next two characters `SE` represent the first index label `Seattle` and the last two characters, `NE`, represent the second index label `New-York`.

7. USING ANALYZE WITH MPSWRITE

MPSWRITE uses two files to communicate with ANALYZE: The LP file and the system file. The LP file contains matrix information and the values of the rows and columns. ANALYZE then studies the linear program and its solution and interactively assists in diagnostic analysis. The syntax file is similar to the mapping file and serves to provide additional information for ANALYZE. Here, names of up to 16 characters are permitted to represent the GAMS variable or constraint.

In order to illustrate the generation of the ANALYZE LP and syntax files consider [TRNSPORT].The options `domps` and `domap` are used to switch to ANALYZE format and are set to the value of 2. The new option file is shown below:

```
nametyp      2
domps  2
domap  2
```

The explanation of these options is given in greater detail in a following section. On solving [TRNSPORT] in the same form as in the previous section, the LP and syntax files are generated as *`gams.lp`* and *`gams.syn`* respectively and are shown in Figures 5 and 6.

```
* file created by gams
* BASE
OPTIMAL  = MINIMIZE
STATUS   = UNKNOWN
NAME     = GAMS
ROWS     =        7
COLUMNS  =        7
NONZEROS =       20
LENGTH   =        8
OBJECTIV = +OBJ
ENDBASE
*** ROWS
C 7 0. 0.
B 0. 0.
SSE 3 -1.E20 350.
B 0. 0.
SSA 3 -1.E20 600.
B 0. 0.
DNE 2 325. 1.E20
B 0. 0.
DCH 2 300. 1.E20
B 0. 0.
DTO 2 275. 1.E20
B 0. 0.
+OBJ 1 -1.E20 1.E20
B 0. 0.
*** COLUMNS
XSENE 3 0. 1.E20
L 0. 0.
XSECH 3 0. 1.E20
L 0. 0.
XSETO 3 0. 1.E20
```

```
L 0. 0.
XSANE 3 0. 1.E20
L 0. 0.
XSACH 3 0. 1.E20
L 0. 0.
XSATO 3 0. 1.E20
L 0. 0.
Z 2 -1.E20 1.E20
B 0. 0.
*** NONZEROS
C XSENE -0.22500000E+00
SSE XSENE 1.
DNE XSENE 1.
C XSECH -0.15300000E+00
SSE XSECH 1.
DCH XSECH 1.
C XSETO -0.16200000E+00
SSE XSETO 1.
DTO XSETO 1.
C XSANE -0.22500000E+00
SSA XSANE 1.
DNE XSANE 1.
C XSACH -0.16200000E+00
SSA XSACH 1.
DCH XSACH 1.
C XSATO -0.12600000E+00
SSA XSATO 1.
DTO XSATO 1.
C Z 1.
+OBJ Z 1.
```

Figure 5: ANALYZE LP file for 'trnsport.gms'

```
* SYNTAX FILE GENERATED BY GAMS/MPSWRITE
00   Universal Set
 SE SEATTLE
 SA SAN-DIEGO
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
I. canning plants
 SE SEATTLE
 SA SAN-DIEGO
J. markets
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
ROW SYNTAX
 C define objective function
 S observe supply limit at plant i &I.:2
 D satisfy demand at market j &J.:2
COLUMN SYNTAX
 X shipment quantities in cases &I.:2 &J.:4
 Z total transportation costs in thousands of dollars
ENDATA
```

Figure 6: ANALYZE Syntax file for 'trnsport.gms'

A further explanation on how to use these two files can be found in Section 10 and in greater detail in the ANALYZE manual.

8. The GAMS/MPSWRITE Options

There are 15 options that can be used to alter the structure and content of the various output files. They are classified by purpose and presented below.

8.1. Name Generation

`namewid`	integer		Maximum width of the names generated. NAMEWID can have a maximum width of 16. If any of the standard MPS files is being generated the maximum width is reduced to 8.(`domps=1, dobas=1`) *(default = 8)*
`nametyp`	integer		Type of names created for variables and equations. *(default = 1)*
		1	The generated names correspond to the ordinality number of the corresponding row or column
		2	More intelligent name generation base on GAMS names

8.2. Name Design

Used if `nametyp=2`. A more detailed exposition on name generation can be found in the next section.

`labminwid`	integer	Minimum width for labels (set members) in new name *(default = 1)*
`labmaxwid`	integer	Maximum width for labels (set members) in new name. The upper bound for this is 4 *(default = 4)*
`labperc`	real	100x is the percentage of clashes that are permitted in label names before renaming. In the name generation algorithm, if the percentage of clashes is more than `labperc`, then the width of the newly generated code for the label is increased. *(default = 0.10)*

`stemminwid`	integer		Minimum width of generated row or column. *(default = 1)*
`stemmaxwid`	integer		Maximum width of generated row or column name. The upper bound is 4. *(default = 4)*
`stemperc`	real		Percentage of clashes permitted in row or column names before renaming. Used similarly to `labperc`. *(default = 0.10)*
`textopt`	integer		Option to process special @ symbol used by ANALYZE *(default = 0)*
		0	Ignore @ in the GAMS symbol text.
		1	ANALYZE specific option related to the *`gams.syn`* file. The macro symbol @ in the GAMS symbol text is recognized and processed accordingly.
`fixed`	integer		Option for fields in LP file. This is again an ANALYZE specific option. *(default = 1)*
		0	Fixed fields for writing LP file.
		1	The LP files is squeezed to remove extra spaces

8.3. Using Files

`usesol`	integer		Option to control the use of primal and dual information. *(default= 0)*
		0	Use matrix file for obtaining dual information. This is then used for the generation of the basis file and the ANALYZE LP file. MPSWRITE will also generate an empty solution file on completion to pass return information required by GAMS.
		1	Use the GAMS solution file to obtain primal and dual solution values. If the model contains nonlinear terms, the partial derivatives (matrix coefficients) and the right-hand-side are also updated to reflect the linearization around the new point. If the matrix is written in ANALYZE LP format, the solution status will be the one from the solution file.

8.4. Writing Files

`domps`	integer		Option to output of the problem matrix. *(default= 1)*
		0	No matrix file is written
		1	MPS file written as *`gams.mps`*
		2	Matrix file in ANALYZE LP format is written as *`gams.lp`*
`btol`		real	Basis tolerance when generating the ANALYZE file *`gams.lp`*. Optimization involves calculations with floating point numbers and are inherently noisy. MPSWRITE re-computes the row activity levels for linear and nonlinear models and the basis tolerance `btol` is used to determine the final basis status. In order to avoid spurious warnings in subsequent steps like ANALYZE, activity levels are reset to the value of the appropriate bound when within the tolerance level. *(default = 1.0d-8)*
`dobas`		integer	controls the output of a file containing basis information *(default = 1)*
		0	No basis file is written
		1	Basis file is written as *`gams.bas`*. As explained earlier, the basis file can be written based on primal and dual information from the solution file if `usesol=1` or else the information is obtained from the matrix file. Potential complications arise with superbasic variables in nonlinear problems. Although most modern LP codes allow work with a similar concept, the standard format of the MPS basis file does not allow for this situation. The OSL specific 'BS' format is used to indicate superbasic activity.
`domap`		integer	controls the output of a file containing name mapping information. *(default = 1)*
		0	no mapping file is written
		1	mapping file containing generated name followed by GAMS name is written in *`gams.map`*.

2	syntax file with ANALYZE format is written in *gams.syn*. Superbasic activities cannot be represented with the ANALYZE LP format. When such activities are encountered, one of the bounds is reset to the activity level, the activity is made non-basic, and a comment is inserted with an appropriate message and the value of the original bound. The lower or upper bound is selected to make the modified problem optimal (according to the sign of the marginal).

Therefore, by default, the MPS, basis and mapping files are generated as *gams.mps*, *gams.bas*, and *gams.map* respectively. The dual information for the generation of the basis file is read from the GAMS matrix file. The generated names have a default width of 8. The effect of various options on the output files is demonstrated through [TRNSPORT] in Section 9.

9. Name Generation Examples

In GAMS, the row and column names have an identifier followed by up to 10 indices which are also identifiers. An identifier consists of up to 10 characters. An example of a name from [TRNSPORT] is `x(Seattle,New-York)`. As a result of this, the total length of a single row or column name can be as long as 110 positions.

In ANALYZE, the names can be up to 16 characters. Character positions determine the meaning, for instance `XSANE` may mean shipping (`X`) from San-Diego (`SA`) to New-York (`NE`). The ANALYZE command 'explain' shows one the meaning of a name in this way based on the information in the syntax file. In MPS files, the names are shorter and have to be less than 8 characters.

The way these names are generated from the GAMS name is as follows: Stems are generated for the equation/variable names. The width of these stems are within the limits provided by the users through the option file. `stemminwid` and `stemmaxwid` provide the minimum and maximum width for the rows and column names while `labminwid` and `labmaxwid` provide similar limits for the label names. The procedure for the generation of these stems is given below.

Initially, the first few characters of the name are chosen to form the name stem or Label name, the width corresponding to the minimum width permitted through the option file. If all the stems that are created in this manner are unique, then the procedure is complete and the stem names created are used. If there are repetitions or non-uniqueness in the stems created then the fraction of these repetitions (the number of repetitions divided by the number of names) is checked against the maximum permissible fraction allowed through the option file through the setting of `labperc` and `stemperc`. If the fraction is above the permissible limit, then the size of the stem or label is increased by one until the maximum width is reached or the fraction is less than the permissible limit. When this is achieved, the remaining clashes are resolved by leaving the first occurrence of

the stem or label name as it is and altering the other occurrences of the stem or label name. These alterations are done by first changing the last character of the stem to a unique one. If this is not possible, the procedure attempts to change the second to the last character to make the stem or label unique. If one cannot still find a unique name, progressively the more significant positions of the stem or label are changed until a unique name is found. Only alphanumeric characters are considered during the generation process.

In order to illustrate the procedure explained above, consider the following set of label names

```
NEW-YORK, BOSTON, PITTSBURGH, NEWARK, WASHINGTON, BUFFALO
```

Let the values of `labminwid`, `labmaxwid` and `labperc` be 1,3 and 0.1 respectively. Initially stems of width 1 corresponding to `labminwid` are created : `N`, `B`, `P`, `N`, `W`, `B`. The result is that 0.33 of the number of names are repeated (2 repetition of a total of 6 names). This is greater than the permitted maximum of 0.1. The width of the label is therefore increased to two, creating the labels, `NE`, `BO`, `PI`, `NE`, `WA`, `BU`. This now increases the number of unique label names but still leaves the fraction of repetitions at 0.17 and so the label width has to be further increased by one to 3. The resulting label are `NEW`, `BOS`, `PIT`, `NEW`, `WAS`, `BUF`. The fraction of repetitions is unchanged, but a further increase in the label width is not possible because the maximum allowable width (`labmaxwid`) has been reached. The remaining non-unique label names have to be altered. The first occurrence of `NEW` is left unchanged while the second occurrence is altered to `NEA` which is a unique name. This leads to the following set of unique label names that are finally obtained:

```
NEW, BOS, PIT, NEA, WAS, BUF
```

In this manner, unique label names are generated for the label names and stem names for the equation/variable names separately. Once this is done, the name associated with any particular row or column is created by putting together the stem name of the equation/variable and the label of all the indices associated with it in order. For example in the variable `x(Seattle, New-York)` in [TRNSPORT], if the stem name of the variable is `X` and the stem names of the associated labels are `SE` and `NE`, the name of the corresponding column is `XSENE`.

However, there is a limit of `namewid` characters that the row/column name can have. If the width of the labels or equation/variable stems is too large or if the row/column has too many indices, the size of the row or column created in the manner described above may lead to a width greater than `namewid`. For example, consider that `namewid` is 16 and that the width of the stems of the labels and equation/variable names created by the procedure described above is 3. Now consider the variable `y(i,j,k,l,m)` which has 5 index positions. The width of any instance of the variable would be 3 + 5(3) = 18. This is clearly above the limit set by the option file. In such a situation, the latter indices are collapsed to form new index sets and labels so as to limit the name width to a maximum of `namewid`. In the example above, the index sets `l` and `m` are collapsed to create a new set and new unique labels for each occurring combination of labels

of `l` and `m` in `y(i,j,k,l,m)`. The new label names are generated by simply using the ordinality of the row or column.

The effect of these procedures are shown in the examples below.

Example 3: Consider solving trnsport.gms with the following entries in the option file

```
nametyp      2
namewid      8
labminwid    1
labmaxwid    3
labperc      0.25
stemminwid   2
stemmaxwid   3
stemperc     0.25
```

The resulting GAMS to MPS mapping file that explains the new names is shown in Figure 7. Notice here that due to the increase in the allowable percentage of clashes in the stems of the label names and the equation/variable names, stems with just one character were sufficient for the label names. A `stemminwid` of 2 forced the stems of the equation/variable names to have two characters. It must be pointed out that all extra positions are padded with the '.' character. So the variables X and Z being smaller than `stemminwid` were changed to `X.` and `Z.` . `X.SN` therefore refers to the GAMS column `X(Seattle, New-York)`.

```
  6  7
CO              COST
SUS             SUPPLY     SEATTLE
SUA             SUPPLY     SAN-DIEGO
DEN             DEMAND     NEW-YORK
DEC             DEMAND     CHICAGO
DET             DEMAND     TOPEKA
X.SN            X          SEATTLE    NEW-YORK
X.SC            X          SEATTLE    CHICAGO
X.ST            X          SEATTLE    TOPEKA
X.AN            X          SAN-DIEGO  NEW-YORK
X.AC            X          SAN-DIEGO  CHICAGO
X.AT            X          SAN-DIEGO  TOPEKA
Z.              Z
```

Figure 7: GAMS to MPS Mapping file with LABPERC, STEMPERC = 0.25

Example 4: In the option file described above, the width of the permissible name is set to 3 by `namewid 3`. Also, to create ANALYZE files, add `domap 2` and `domps 2`. Since the equation/variable name stems still had to have at least two characters because of `stemminwid`, this example is used to show the effect of the collapsing process on the names generated. The variable `x(i,j)` requires the collapsing of its indices since two indices would not be able to fit into the last remaining position. The resulting ANALYZE syntax file (*`gams.syn`*) that explains the new labels and set that were created due to the collapsing is shown in Figure 8. The new set (called 3) consists of the 6 members of `x(i,j)` and has labels whose names depend on the ordinality of the occurrence of label combinations in (I,J). The new row/column names that are

generated can be seen in the ANALYZE LP file (*gams.lp*) shown in Figure 9. Notice now that a 3 character name `X.1` represent `x(Seattle, New-York).` This example may seem rather forced due to the relatively small number of indices in the equations and variables, however it should be realized that when the number of indices becomes larger, this process is inevitable.

```
* SYNTAX FILE GENERATED BY GAMS/MPSWRITE
0    Universal Set
 S SEATTLE
 A SAN-DIEGO
 N NEW-YORK
 C CHICAGO
 T TOPEKA
I canning plants
 S SEATTLE
 A SAN-DIEGO
J markets
 N NEW-YORK
 C CHICAGO
 T TOPEKA
3 NEW SET FOR X
 1 SEATTLE   .NEW-YORK
 2 SEATTLE   .CHICAGO
 3 SEATTLE   .TOPEKA
 4 SAN-DIEGO .NEW-YORK
 5 SAN-DIEGO .CHICAGO
 6 SAN-DIEGO .TOPEKA
ROW SYNTAX
 CO define objective function
 SU observe supply limit at plant i &I:3
 DE satisfy demand at market j &J:3
COLUMN SYNTAX
 X. shipment quantities in cases &3:3
 Z. total transportation costs in thousands of dollars
ENDATA
```

Figure 8: ANALYZE Syntax file with NAMEWID=3

```
* file created by gams
* BASE
OPTIMAL  = MINIMIZE
STATUS   = UNKNOWN
NAME     = GAMS
ROWS     =        7
COLUMNS  =        7
NONZEROS =       20
LENGTH   =        3
OBJECTIV = +OBJ
ENDBASE
*** ROWS
CO 7 0. 0.
B 0. 0.
SUS 3 -1.E20 350.
B 0. 0.
SUA 3 -1.E20 600.
B 0. 0.
DEN 2 325. 1.E20
B 0. 0.
DEC 2 300. 1.E20
```

```
B 0. 0.
DET 2 275. 1.E20
B 0. 0.
+OBJ 1 -1.E20 1.E20
B 0. 0.
*** COLUMNS
X.1 3 0. 1.E20
L 0. 0.
X.2 3 0. 1.E20
L 0. 0.
X.3 3 0. 1.E20
L 0. 0.
X.4 3 0. 1.E20
L 0. 0.
X.5 3 0. 1.E20
L 0. 0.
X.6 3 0. 1.E20
L 0. 0.
Z. 2 -1.E20 1.E20
B 0. 0.
*** NONZEROS
CO X.1 -0.22500000E+00
SUS X.1 1.
DEN X.1 1.
CO X.2 -0.15300000E+00
SUS X.2 1.
DEC X.2 1.
CO X.3 -0.16200000E+00
SUS X.3 1.
DET X.3 1.
CO X.4 -0.22500000E+00
SUA X.4 1.
DEN X.4 1.
CO X.5 -0.16200000E+00
SUA X.5 1.
DEC X.5 1.
CO X.6 -0.12600000E+00
SUA X.6 1.
DET X.6 1.
CO Z. 1.
+OBJ Z. 1.
```

Figure 9: ANALYZE LP file with NAMEWID=3

```
$TITLE  A TRANSPORTATION PROBLEM (TRNSPORT,SEQ=1)
$OFFUPPER

  SETS
       I   canning plants   / SEATTLE, SAN-DIEGO /
       J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;

  PARAMETERS

       A(I)  capacity of plant i in cases
         /    SEATTLE     350
              SAN-DIEGO   600  /

       B(J)  demand at market j in cases
         /    NEW-YORK    325
```

```
             CHICAGO     300
             TOPEKA      275  / ;

  TABLE D(I,J)  distance in thousands of miles
                    NEW-YORK       CHICAGO      TOPEKA
      SEATTLE          2.5           1.7          1.8
      SAN-DIEGO        2.5           1.8          1.4  ;

  SCALAR F  freight in dollars per case per thousand miles  /90/ ;

  PARAMETER C(I,J)  transport cost in thousands of dollars per
case ;

           C(I,J) = F * D(I,J) / 1000 ;

  VARIABLES
       X(I,J)  shipment quantities in cases from @i to @j
       Z       total transportation costs in thousands of dollars
;

  POSITIVE VARIABLE X ;

  EQUATIONS
       COST        define objective function
       SUPPLY(I)   observe supply limit at plant @i
       DEMAND(J)   satisfy demand at market @j ;

  COST ..        Z  =E=  SUM((I,J), C(I,J)*X(I,J)) ;

  SUPPLY(I) ..   SUM(J, X(I,J))  =L=  A(I) ;

  DEMAND(J) ..   SUM(I, X(I,J))  =G=  B(J) ;

  MODEL TRANSPORT /ALL/ ;

  OPTION LP = MPSWRITE ;

  SOLVE TRANSPORT USING LP MINIMIZING Z ;

  DISPLAY X.L, X.M ;
```

Figure 10: Modified 'trnsport.gms' for Example (A.3)

```
* SYNTAX FILE GENERATED BY GAMS/MPSWRITE
00   Universal Set
 SE SEATTLE
 SA SAN-DIEGO
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
I. canning plants
 SE SEATTLE
 SA SAN-DIEGO
J. markets
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
ROW SYNTAX
 C define objective function
```

```
 S observe supply limit at plant &I.:2
 D satisfy demand at market &J.:2
COLUMN SYNTAX
 X shipment quantities in cases from &I.:2 to &J.:4
 Z total transportation costs in thousands of dollars
ENDATA
```

Figure 11: ANALYZE Syntax file with TEXTOPT=1

```
* file created by gams
* BASE
OPTIMAL  = MINIMIZE
STATUS   = UNKNOWN
NAME     = GAMS
ROWS     =        7
COLUMNS  =        7
NONZEROS =       20
LENGTH   =        8
OBJECTIV = +OBJ
ENDBASE
*** ROWS
C                       7    0.00000000E+00    0.00000000E+00
B                            0.00000000E+00    0.00000000E+00
SSE                     3   -0.10000000E+21    0.35000000E+03
B                            0.00000000E+00    0.00000000E+00
SSA                     3   -0.10000000E+21    0.60000000E+03
B                            0.00000000E+00    0.00000000E+00
DNE                     2    0.32500000E+03    0.10000000E+21
B                            0.00000000E+00    0.00000000E+00
DCH                     2    0.30000000E+03    0.10000000E+21
B                            0.00000000E+00    0.00000000E+00
DTO                     2    0.27500000E+03    0.10000000E+21
B                            0.00000000E+00    0.00000000E+00
+OBJ                    1   -0.10000000E+21    0.10000000E+21
B                            0.00000000E+00    0.00000000E+00
*** COLUMNS
XSENE                   3    0.00000000E+00    0.10000000E+21
L                            0.00000000E+00    0.00000000E+00
XSECH                   3    0.00000000E+00    0.10000000E+21
L                            0.00000000E+00    0.00000000E+00
XSETO                   3    0.00000000E+00    0.10000000E+21
L                            0.00000000E+00    0.00000000E+00
XSANE                   3    0.00000000E+00    0.10000000E+21
L                            0.00000000E+00    0.00000000E+00
XSACH                   3    0.00000000E+00    0.10000000E+21
L                            0.00000000E+00    0.00000000E+00
XSATO                   3    0.00000000E+00    0.10000000E+21
L                            0.00000000E+00    0.00000000E+00
Z                       2   -0.10000000E+21    0.10000000E+21
B                            0.00000000E+00    0.00000000E+00
*** NONZEROS
C               XSENE              -0.22500000E+00
SSE             XSENE               0.10000000E+01
DNE             XSENE               0.10000000E+01
C               XSECH              -0.15300000E+00
SSE             XSECH               0.10000000E+01
DCH             XSECH               0.10000000E+01
C               XSETO              -0.16200000E+00
SSE             XSETO               0.10000000E+01
```

```
DTO             XSETO            0.10000000E+01
C               XSANE           -0.22500000E+00
SSA             XSANE            0.10000000E+01
DNE             XSANE            0.10000000E+01
C               XSACH           -0.16200000E+00
SSA             XSACH            0.10000000E+01
DCH             XSACH            0.10000000E+01
C               XSATO           -0.12600000E+00
SSA             XSATO            0.10000000E+01
DTO             XSATO            0.10000000E+01
C               Z                0.10000000E+01
+OBJ            Z                0.10000000E+01
```

Figure 12: ANALYZE LP file with FIXED=1

Example 5: This example is used to demonstrate the effect of the options `fixed` and `textopt`. Consider the same gams input file *`trnsport.gms`* with the following modification: replace the text following the definition of the variable `x(i,j)` by '`shipment quantities in cases from @i to @j`' and that following the definitions of equations `supply` and `demand` by '`observe supply at plant @i`' and 'satisfy demand at market @j' respectively. The resulting gams file is shown in Figure 10. The macro character `@` is used as a flag for GAMS/MPSWRITE to recognize the following characters as a set name and replace them by a string recognizable by ANALYZE. Now include the option file shown below.

```
nametyp      2
dobas        0
domap        2
domps        2
textopt      1
fixed        1
```

The options, `textopt 1` and `fixed 1` have been added relative to the previous examples. The resulting ANALYZE syntax file is shown in Figure 11. Notice the string that replaces the macros `@i` and `@j`. In the string, ANALYZE recognizes the characters between '`&`' and '`:`' as representing the stem of the index name corresponding to the string following the '`@`' macro in the gams file, and the number immediately following the ':' as the number of characters from the left of the row/column name that the particular set label stem begins. As an illustration, in the row syntax for `X`, we see the strings `&I.:2` and `&J.:4`. This means that in any row of `X`, say `XSENE`, the label corresponding to `I` begins at the second position (`SE`) while the label corresponding to `J` begins at the fourth position (`NE`). This is used by the ANALYZE command EXPLAIN. For further information on this topic, consult the ANALYZE manual.

The ANALYZE LP file shown in Figure 11 demonstrates the effect of the option `fixed` which uses a fixed field format for the LP file. This makes the file size larger but at the same time is easier to read.

10. INTEGRATING GAMS AND ANALYZE

In this section a number of examples are given to demonstrate the use of ANALYZE with GAMS. This description is intended for users already familiar with ANALYZE. For more information on ANALYZE you should consult the User's Guide for ANALYZE or contact Prof. Harvey Greenberg at the University of Denver by email at the following address - hgreenberg@cudnvr.denver.colardo.edu.

GAMS takes advantage of a special ANALYZE feature which allows 16 character long names instead of the usual 8 character MPS type names. In order to use this feature, the problem has to presented in an ANALYZE specific format, the ANALYZE LP file, which needs further conversion into an ANALYZE PCK file format before it can be read by ANALYZE. The ANALYZE Utility LPRENAME is needed to rewrite the GAMS generated LP file into an ANALYZE PCK file. The LPRENAME utility is part of the ANALYZE system but is usually not installed during standard installation under DOS. The name LPRENAME.EXE can be easily copied from the distribution disketted by using the following command,

```
d:pkxarc d:LPRNEXE
```

The general steps to translate an LP file are:

```
>LPRENAME
 < LPRENAME credits etc >
 LPRENAME... LOAD GAMS
 LPRENAME... WRITE GAMS
 LPRENAME... QUIT
```

The command 'LOAD GAMS' instructs LPRENAME to read the input file GAMS.LP. The command `write` will generate a file called *`gams.pck`* which is a binary (packed) version of *`gams.lp`*. For the purposes of this document, this is all one needs to know about the LPRENAME function, further details can be found in Chapter 8 of the ANALYZE User's Guide.

There are various ways how one can generate the appropriate input files for ANALYZE and different methods of invoking the analysis software. The choice will depend on the kind of analysis is required and how frequent the GAMS-ANALYZE link will be used. Several examples will be used to illustrate different GAMS/ANALYZE set-ups for DOS and UNIX (AIX RS/6000) environments.

Example 6: This example shows how to generate the ANALYZE LP and SYN files and save them for further processing. An MPSWRITE option file will be used to specify the types of files required and name design option selected. The 'mpswrite.opt' file will contain:

```
* design meaningful names
nametyp      2
* analyze can use wide names
namewid      16
* write an analyze syntax file
domap  2
```

```
* write an analyze matrix file
domps  2
* don't write basis file
dobas  0
```

In the GAMS file a solve statement is used to call the standard MPSWRITE subsystem. Note that MPSWRITE is invoked after a `solve`e statement in order to pass correct dual information. Once again [TRNSPORT] is used for illustration. The following slice of code is added at the end of the file:

```
solve transport minimizing z using lp;
display x.l, x.m;
* additions follow below
option lp=mpswrite;
solve transport minimizing z using lp
```

After running the modified TRNSPORT model one may want to save the file *`gams.lp`* and *`gams.syn`* under a different name. For example, one could just copy and translate the files as follows:

```
>GAMS TRNSPORT
    < GAMS log >
>COPY GAMS.LP  TRNSPORT.LP
>COPY GAMS.SYN TRNSPORT.SYN
>LPRENAME
  < translate .lp file into a .pck as shown above>
>ANALYZE
  < various credits etc.>
  ANALYZE... READIN PAC TRNSPORT
  ANALYZE... READIN SYN TRNSPORT
  < now we are ready to use ANALYZE commands >
  ANALYZE... QUIT
```

Example 7: This example shows how one can modify the MPSWRITE script or batch files to run LPRENAME automatically and have a new default option file. On some systems one may not have access to the GAMS systems directory and need the systems administrator to help with changes. Also see the next example which will introduce a new subsystem.

The standard MPSWRITE script is located in the GAMS systems directory and usually looks as follows:

```
: GAMSMPS3.BAT  Batch file for MPSWRITE
: %3 parameter file
: %4 control file
@echo off
gams3mps %4
gamscmex %3
gamsnext %3
```

For easier management the echo command is used to generate the intermediate files needed in this example. The modified script/batch file is as follows:

```
: GAMSAYZ3.BAT  Batch file for MPSWRITE
: modified to generate ANALYZE files
: %3 parameter file
: %4 control file
@echo off
: do MPSWRITE option file
echo NAMETYP 2            >  mpswrite.opt
echo NAMEWID 16           >> mpswrite.opt
echo DOMAP 2        >> mpswrite.opt
echo DOMPS 2        >> mpswrite.opt
echo DOBAS 0        >> mpswrite.opt
echo TEXTOPT 1            >> mpswrite.opt
: do LPRENAME input script
echo Y              >  lprename.in
echo Y              >> lprename.in
echo LOAD GAMS            >> lprename.in
echo WRITE GAMS           >> lprename.in
echo QUIT                 >> lprename.in
: do ANALYZE profile file
: need to move CREDITS.DOC
echo * GAMS/ANALYZE >  profile.exc
: following lines depend on ANALYZE installation
echo READIN PAC GAMS      >> profile.exc
echo READIN SYN GAMS      >> profile.exc
gams3mps %4
LPRENAME < lprename.in > lprename.out
ANALYZE
gamscmex %3
gamsnext %3
```

Note that LPRENAME and ANALYZE are assumed to be accessible through the path or can be found in the current directory. If this is not the case, one has to enter the appropriate path to the script file above. Also note the use of the ANALYZE profile feature. The ***profile.exc*** file is executed only after the credits have been processed. The only way to avoid having to respond with Y/N to ANALYZE questions interactively is the renaming of the ***credits.doc*** file. If no ***credits.doc*** file is found by ANALYZE, it will immediately look for a ***profile.exc*** file and execute the commands included in this file. After having made these changes we can redo the previous example as follows:

```
>GAMS TRNSPORT
 .
 ... GAMS log
 .
 < ANALYZE credits, file reading, etc. >
 ANALYZE...
 ...
 ANALYZE... QUIT
 < GAMS termination messages >
> next command
```

The experienced ANALYZE user may want to use further ANALYZE macros to automate the analysis and reporting.

Example 8: The previous example has two main drawbacks: (1) the standard MPSWRITE set-up has been modified and other users may be rather upset that MPSWRITE has changed its default behavior. (2) The model status is UNDEFINED although one may have called ANALYZE for a model for which the status is known. The first problem can be overcome by introducing a new solver name and/or making ones own copy of the *gamscomp.txt* file which tells GAMS the availability and location of the subsystems. In this example a new solver name will be introduced for the modified MPSWRITE script.

Without going into details of the use of the *gamscomp.txt* file, only the changes needed for this example are shown. The *gamscomp.txt* file is located in the GAMS system directory. The relevant portions of this file are shown below:

```
MPSWRITE 12 3 LP MIP RMIP NLP MINLP RMINLP MCP
gamsmps3
```

A new solver is introduced by making a new entry similar to the MPSWRITE entry with one difference, the modified script is given the new name *gamsayz3.bat* and call the new solver MPSAYZ. After making the changes the file will look like:

```
MPSWRITE 12 3 LP MIP RMIP NLP MINLP RMINLP MCP
gamsmps3
MPSAYZ 12 3 LP MIP RMIP NLP MINLP RMINLP MCP
gamsayz3
```

Now one can either make the changes global by replacing the *gamscomp.txt* file in the GAMS system directory, or we keep a local (personal) copy. When making changes to the *gamscomp.txt* file in the GAMS system director, it is important to remember that the GAMS installation program rewrites this file and may erase the changes we just made when reinstalling the system for some reason. Similar copies of the MPSWRITE script file will then be needed in the GAMS system directory. The original one called *gams3mps.bat* and the new one called *gamsayz.bat*.

Instead of using the global configuration file one can tell GAMS to use any other file to find out what capabilities are installed. Here we will keep the modified 'gamscomp.txt' and the new gamsayz3.bat in the current directory. The GAMS input file will use the new solver as shown below:

```
option lp=mpsayz;
solve ....;
```

When calling GAMS we need to indicate that we want to use a nonstandard *gamscomp.txt* file by using the command line parameter `subsys` as follows:

```
>gams ..... subsys=gamscomp.txt
```

Example 9: This example shows how one can synchronize the values of the primal and dual solution and pass the appropriate information to ANALYZE. To accomplish this one needs to

combine a standard optimize script with an MPSWRITE script. The model is first solved, then the MPSWRITE system is called to produce the correct files, the LP file is translated to a PCK file and finally ANALYZE is called directly out of the solver script as in the previous example. The GAMS job will automatically be suspended until one exits ANALYZE. Once ANALYZE has terminated, GAMS will continue as before.

The BDMLP script is combined with a modified MPSWRITE script by calling the MPSWRITE step immediately after the optimization step is completed. The new name for this combined solver will be ANALYZE. Both the DOS and Unix versions of the scripts will be given.

From the BDMLP script *gamsbdm3.bat* one can see that is needed is to insert a call to the BDMLP solver which will automatically produce a solution file which can then be read by MPSWRITE. The modified script looks then as follows:

```
: GAMSANA3.BAT  Batch file for MPSWRITE
: modified to generate ANALYZE files
: and call the BDMLP solver to get correct
: status and duals
:
: %3 parameter file
: %4 control file
@echo off
: do MPSWRITE option file
echo NAMETYP 2            >  mpswrite.opt
echo NAMEWID 16           >> mpswrite.opt
echo DOMAP 2        >> mpswrite.opt
echo DOMPS 2        >> mpswrite.opt
echo DOBAS 0        >> mpswrite.opt
echo TEXTOPT 1            >> mpswrite.opt
echo USESOL 1             >> mpswrite.opt  ---- additional option
: do LPRENAME input script
echo Y              >  lprename.in
echo Y              >> lprename.in
echo LOAD GAMS            >> lprename.in
echo WRITE GAMS           >> lprename.in
echo QUIT                 >> lprename.in
: do ANALYZE profile file
: need to move CREDITS.DOC file
echo * GAMS/ANALYZE >  profile.exc
echo READIN PAC GAMS      >> profile.exc
echo READIN SYN GAMS      >> profile.exc
gams3bdm %4     ---- additional call to BDMLP
gams3mps %4
LPRENAME < lprename.in > lprename.out
ANALYZE
gamscmex %3
gamsnext %3
```

The GAMS configuration file *gamscomp.txt* will now have one more solver, ANALYZE which will first call an LP code and the ANALYZE to do further analysis. The updated *gamscomp.txt* file will look as follows:

```
MPSWRITE 12 3 LP MIP RMIP NLP DNLP MINLP RMINLP MCP
gamsmps3
```

```
ANALYZE 2 3 LP MIP RMIP NLP DNLP MINLP RMINLP MCP
gamsana3
```

The careful reader will notice a difference in the second item following the solver names MPSWRITE and ANALYZE. This second item specifies the file format to be used when communicating between GAMS and the solvers. The code 12 means to add dual information to the matrix file and use the packed format number 2. The code 2 means to use the old matrix file format and use packed format 2. First, we do not need dual information in the matrix file because we will use the solver solution file. Second, the '1x' format is not compatible with older subsystems and may create difficulties with installations that run a 'mixed generation' system. Before running the model again, the new solver name ANALYZE has to be used in the GAMS program:

```
option lp=analyze; solve transport using ... ;
```

Assuming that the global *gamscomp.txt* file has been updated, one can now simply call GAMS and have all information about the model automatically passed on the ANALYZE system.

The interested ANALYZE user will find a copy of the scripts in the GAMS system directory.

Example 10: This example is identical to the previous one, except that it is for an IBM AIX RS/6000 system, a powerful unix workstation. The script file look very similar and are listed without further explanation. The modified 'gamsana' script is shown below:

```
# gamsana.run script file for MPSWRITE/ANALYZE
# This is modified mpswrite.run script
# $3 parameter file
# $4 control file
# $5 system directory
# do MPSWRITE option file
echo NAMETYP 2    >  mpswrite.opt
echo NAMEWID 16   >> mpswrite.opt
echo DOMAP 2      >> mpswrite.opt
echo DOMPS 2      >> mpswrite.opt
echo DOBAS 0      >> mpswrite.opt
echo TEXTOPT 1    >> mpswrite.opt
echo USESOL 1     >> mpswrite.opt
# do LPRENAME input script
echo Y              >  lprename.in
echo Y              >> lprename.in
echo LOAD gams.lp   >> lprename.in
echo WRITE gams.pck >> lprename.in
echo QUIT           >> lprename.in
# do ANALYZE profile file
# need to move CREDITS.DOC
echo \* GAMS/ANALYZE     >  PROFILE.EXC
echo READIN PAC gams.pck >> PROFILE.EXC
echo READIN SYN gams.syn >> PROFILE.EXC
${5}gamsbdmlp.out  $4
${5}gams3mps.out  $4
LPRENAME < lprename.in > lprename.out
```

```
ANALYZE
exit 11
```

Selected portions of the 'gamscomp.txt' file are listed below:

```
MPSWRITE 12 3 1 LP MIP RMIP NLP DNLP MINLP RMINLP MCP
${2}gamsmps3.run
${2}gams3mps.out

ANALYZE 2 3 LP MIP RMIP NLP DNLP MINLP RMINLP MCP
gamsana.run
```

Using the flexibility of both systems, GAMS and ANALYZE, one can further enhance the above examples and provide application tailored analytic capabilities for standard GAMS applications.