SENSITIVITY ANALYSIS WITH GAMS/CPLEX AND GAMS/OSL

Contents:

1.1 Introduction

In general, GAMS offers Sensitivity analysis or post-optimality analysis in Linear Programming allows the user to find out more about an optimal solution of an LP problem. In particular objective coefficient ranging and right-hand side coefficient ranging gives information about how much an objective coefficient cj or a right-hand side coefficient bi can change without changing the optimal basis. In other words it gives information about how sensitive the optimal basis is for changes in the objective function or the right-hand side.

Although not so much used in practical large scale modeling, and not available for mixed-integer models or non-linear models, some users requested the availability of ranging information through GAMS. This document describes how we implemented ranging in GAMS/CLEX and GAMS/OSL.

1.2 The <modelname>.dictfile option (available with GAMS 2.50 and above)

If you want to enable sensitivit analsysis just add thes two statements to you model before the solve statement:

```
modelname.dictfile = 4;
modelname.optfile = 1 (or other values if you are using different option files)
```

1.3 The Dictionary File

(Users of GAMS version 2.5 and above can skip this paragraph)

We did not want to change the format of the solution file, in which primal and dual information about the (optimal) solution is passed back to GAMS. In fact our goal was to incorporate ranging without making any changes in GAMS.

This requires for the solver to have access to the GAMS symbol table, where the names of the equations and variables are stored. This symbol table can be exported by GAMS (in the form of a dictionary file) by changing the gamscomp.txt file. This file is the "solver capability file". It contains information about each solver: its name, what models it can solve (LP's, MIP's etc.), how to run the solver and also whether or not it needs a dictionary file. An entry in gamscomp.txt on a UNIX system, can look like[1]:

```
osl 2 0 LP RMIP MIP
${2}gamsosl.run
cplex 1 0 LP RMIP
${2}gamscplex.run
```

[1] For PC, VMS and VM, the gamscomp entries are slightly different. However, for all these platforms, the second digit in the first line of each entry has the same meaning.

The first digit indicates the file format in which the matrix and solution files are read and written. A zero means coded or ASCII files, a one indicates binary stream I/O and a number 2 results in Fortran style unformatted files with a record structure. For Fortran solvers we use most of the time 2, while for solvers written in C 1 is more appropriate. GAMS itself (gamscmex.out or gamscmex.exe) can read and write either of them.

The second number gives information about the dictionary file. The dictionary file is a file that is optionally written by GAMS and that contains symbol table information. A 0 means "no dictionary file is needed", a 1 writes a standard dictionary file and a 2 writes a dictionary file with quoted strings. The last option is easier to use in a Fortran environment.

For CPLEX we need option 2 and for OSL (Fortran based) we need option 2. So if you want to do ranging with CPLEX or OSL, use the following entries in the (UNIX) gamscomp.txt file:

```
osl 2 2 LP RMIP MIP
${2}gamsosl.run
cplex 1 1 LP RMIP
${2}gamscplex.run
```

1.4 Printing all ranging information in the listing file

To echo all ranging information in the listing file, use the following option file:

```
objrng
rhsrng
```

The option file should be called osl.opt for OSL and cplex.opt for Cplex (cplexmip.opt for CPLEXMIP). Don't forget to use the ".optfile" model suffix inside GAMS, like:

```
MODEL TRANSPORT /ALL/;
TRANSPORT.OPTFILE=1;
SOLVE TRANSPORT USING LP MINIMIZING Z;
```

If this option is not used the option file will not be read by the solver.

Example:

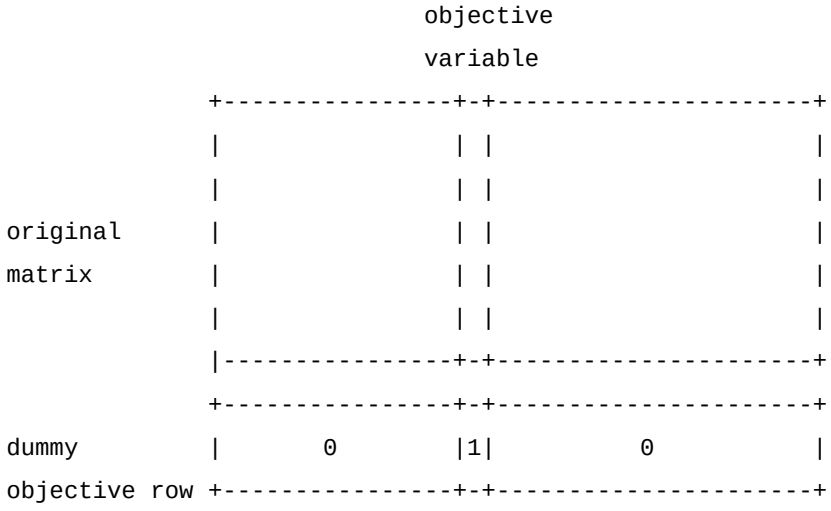
Solving the model `trnsport.gms` (from the model library) with Cplex using the above option file gives the following table in the listing file[2]:

```
EQUATION NAME                    LOWER      CURRENT        UPPER
-------------                    -----      -------        -----
COST                             -INF            0         +INF
SUPPLY(SEATTLE)                   300          350          625
SUPPLY(SAN-DIEGO)                 550          600         +INF
DEMAND(NEW-YORK)                   50          325          375
```

[2] This model is degenerate in its optimal solution. Solving it with different solvers give a different optimal basis, and therefore different ranges.

```
DEMAND(CHICAGO)                    25          300          350
DEMAND(TOPEKA)                      0          275          325
VARIABLE NAME                     LOWER      CURRENT        UPPER
-------------                     -----      -------        -----
X(SEATTLE,NEW-YORK)              -0.009            0            0
X(SEATTLE,CHICAGO)               -0.153            0        0.009
X(SEATTLE,TOPEKA)                -0.036            0         +INF
X(SAN-DIEGO,NEW-YORK)                 0            0        0.009
X(SAN-DIEGO,CHICAGO)             -0.009            0         +INF
X(SAN-DIEGO,TOPEKA)              -0.126            0        0.036
Z                                     0            1         +INF
```

Notice that the current values of the objective function coefficients are all zero, except for the objective variable! GAMS does not have the notion of an objective row, it only knows about an objective variable. GAMS/CPLEX therefore has added a dummy objective function only with a 1.0 in the position of the objective variable:



1.5 Selecting variables and equations

For large scale models the amount of information printed this way can be enormous. Therefore we allow selecting only the blocks of equations and variables for which ranging information is printed.

Consider the following option file:

```
rhsrng supply
objrng x
```

The resulting output from OSL when running the trnsport.gms model is:

```
EQUATION NAME                     LOWER      CURRENT        UPPER
-------------                     -----      -------        -----
SUPPLY(SEATTLE)                300.0000     350.0000     625.0000
```

```
SUPPLY(SAN-DIEGO)                 550.0000     600.0000     900.0000

VARIABLE NAME                        LOWER      CURRENT        UPPER

-------------                        -----      -------        -----

X(SEATTLE,NEW-YORK)                 .0000        .0000         +INF

X(SEATTLE,CHICAGO)                 -.1530        .0000        .0090

X(SEATTLE,TOPEKA)                  -.0360        .0000         +INF

X(SAN-DIEGO,NEW-YORK)               .0000        .0000        .0000

X(SAN-DIEGO,CHICAGO)               -.0090        .0000         +INF

X(SAN-DIEGO,TOPEKA)                -.1260        .0000        .0360
```

You can specify more than one block:

```
rhsrng supply, demand
```

or

```
rhsrng supply
rhsrng demand
```

Non-existent names are (silently) ignored.

1.6 Ranging information available inside GAMS

Printed information in the listing file is not always useful. You can not process this information further (at least not without quite some hand editing). Also in our ranging example there is no choice in formatting.

We have provided also a way of storing the ranging information that is produced by the solver in a GAMS restart file that can be used in subsequent GAMS invocations. The ranging information is stored as a normal GAMS parameter. You can operate on this parameter or use it in a PUT statement for customized reports.

The option to be used to instruct CPLEX or OSL to write a restart include file is:

```
rngrestart filename
```

A word of caution for some users of GAMS/OSL: The file with the filename above may be created in the scratch directory in some versions of GAMS/OSL and is therefore deleted at the end of execution. In order to get around this you may want to run `gamskeep <model>` instead of `gams <model>`.

GAMS/CPLEX users shouldn't have this problem. In the example below we used the option file:

```
objrng
rhsrng
rngrestart ranges.inc
```

We executed the command:

```
$ gams trnsport -s sav
```

After completion the file ranges.inc looks like:

```
* INCLUDE FILE WITH RANGING INFORMATION
* The set RNGLIM /LO,UP/ is assumed to
* be declared.
 PARAMETER COSTRNG(RNGLIM) /
```

```
LO  -INF
UP  +INF
 /
 PARAMETER SUPPLYRNG(I,RNGLIM) /
SEATTLE.LO      .3000000000E+03
SEATTLE.UP      .6250000000E+03
SAN-DIEGO.LO      .5500000000E+03
SAN-DIEGO.UP      .9000000000E+03
 /
 PARAMETER DEMANDRNG(J,RNGLIM) /
NEW-YORK.LO      .0000000000E+00
NEW-YORK.UP      .3250000000E+03
CHICAGO.LO      .0000000000E+00
CHICAGO.UP      .3500000000E+03
TOPEKA.LO      .0000000000E+00
TOPEKA.UP      .2750000000E+03
 /
 PARAMETER XRNG(I,J,RNGLIM) /
SEATTLE.NEW-YORK.LO  -INF
SEATTLE.NEW-YORK.UP  +INF
SEATTLE.CHICAGO.LO      .3000000000E+03
SEATTLE.CHICAGO.UP      .6250000000E+03
SEATTLE.TOPEKA.LO      .5500000000E+03
SEATTLE.TOPEKA.UP      .9000000000E+03
SAN-DIEGO.NEW-YORK.LO      .0000000000E+00
SAN-DIEGO.NEW-YORK.UP      .3250000000E+03
SAN-DIEGO.CHICAGO.LO      .0000000000E+00
SAN-DIEGO.CHICAGO.UP      .3500000000E+03
SAN-DIEGO.TOPEKA.LO      .0000000000E+00
SAN-DIEGO.TOPEKA.UP      .2750000000E+03
 /
 PARAMETER ZRNG(RNGLIM) /
LO      .0000000000E+00
UP      .5000000000E+02
 /
```

Notice that the parameters are called XRNG etc. I.e. all names are suffixed by RNG. It is the responsibility of the modeler not to use names longer than 7 characters: otherwise identifiers would results with more than 10 characters which exceed the limit that is used by GAMS.

The following restart job t2.gms will use the display statement to display the parameters:

```
SET RNGLIM /LO,UP/;
```

```
$include ranges.inc
DISPLAY COSTRNG, SUPPLYRNG, DEMANDRNG;
DISPLAY XRNG, ZRNG;
The restart job can be executed as:
         $ gams t2 -r save
The listing file will contain the output of the display statements:
----    110 PARAMETER COSTRNG
LO -INF,    UP +INF
----    110 PARAMETER SUPPLYRNG
                   LO          UP
SEATTLE       300.000     625.000
SAN-DIEGO     550.000     900.000
----    110 PARAMETER DEMANDRNG
                  UP
NEW-YORK     325.000
CHICAGO      350.000
TOPEKA       275.000
----    111 PARAMETER XRNG
                             LO          UP
SEATTLE  .NEW-YORK        -INF        +INF
SEATTLE  .CHICAGO      300.000     625.000
SEATTLE  .TOPEKA       550.000     900.000
SAN-DIEGO.NEW-YORK                 325.000
SAN-DIEGO.CHICAGO                  350.000
SAN-DIEGO.TOPEKA                   275.000
----    111 PARAMETER ZRNG
UP 50.000
```

1.7 Interpreting "basis change" in OSL

There are some subtle differences in the ranging information supplied by the GAMS LP solvers. For example, when solving the LP

```
scalar b1 / 12 /;
positive variables x, y, u1;
free variable z;
equations obj, f1, f2;

obj .. z =e= x + y;
f1 ..  2 * x  +      y =l= b1;
f2 ..      x  +  2 * y =l= 4;
```

```
model foo / obj, f1, f2 /;
solve foo maximizing z using lp;
```

the CPLEX LP solver would give the range [8, infinity) for the parameter b1. This range treats the point where equation f1 becomes binding, the variable y becomes basic, and the shadow price (dual value) associated with f1 becomes positive as a breakpoint in the solution space for the optimal primal and dual values, expressed as a function of the parameter b1. This is what one would normally expect.

The OSL solver indicates that the range of values for the rhs of equation f1 is [2, infinity). The lower bound of two is the parameter value at which the variable x (basic at the solution when b1 is in (2, infinity) becomes non-basic. What this range information fails to catch is the parameter value b1 = 8, where the variable y becomes basic (it is nonbasic at the solution when b1 > 8). OSL chooses to ignore this basis change as "uninteresting", since the leaving column is one that is not included in the original problem formulation.

In order to get the same behavior with GAMS/OSL and GAMS/CPLEX, it is necessary to add an explicit slack variable to this equation when using OSL, as follows:

```
f1 ..  2 * x + y + u1 =e= b1;
```

When this slack column is included, OSL respects the change in basis when equation f1 becomes binding, and these solvers give the same results. A similar slack could be added to equation f2 as well.

In the example above, both solvers give an upper bound of infinity for the parameter associated with the slack constraint. This will not always be the case with OSL, however. Consider the following model.

```
sets
 I / 1 * 3 /,
 J / 1 * 2 /;

table A(I,J)
      1     2
1           1
2     1     1
3     1           ;

parameters
c(J) /
1     2,
2     1
/,
b(I) /
1     8,
2     3,
3     2
/;

positive variables  x(J);
```

```
free variable       z;

equations f(I), obj;

obj .. z =e= sum(J, c(J)*x(J) );
f(I) ..      sum(J, A(I,J)*x(J) ) =l= b(I);

model bar / f, obj  /;
option lp=cplex; bar.optfile=1;
solve bar using lp maximizing z;
```

The optimal solution to model bar occurs at x = (2, 1), so that f(2) and f(3) are the active constraints. It is easy to see that b(1) can range between 1 and infinity without causing f(1) to become binding. However, as observed above, OSL pushes the value of b(1) down to zero before it sees an "interesting" basis change (i.e., x(2) leaves the basis), so OSL will give a lower bound of zero for the parameter b(1). In addition, OSL gives an upper bound of 3 for the parameter b(1), *a value lower than the current parameter value*. To explain this, we note that instead of determining a range of **parameter** values for slack constraints, OSL tries to determine how far the **activity level** (i.e. level values for the constraint body) can be moved before an "interesting" change in the basis occurs. The definition of "interesting" is somewhat involved.

If the level value can be decreased somewhat without violating any constraints or variable bounds, then the lower bound for the range is the value at which another constraint or variable bound first becomes active. If the level value cannot be decreased, then the constraints and bounds blocking this decrease are ignored during the computation of the lower bound. The upper bound on the range is computed in a similar fashion. Those having programming experience with OSL will perhaps recognize that this type of behavior is achieved by passing the value **`respect=1`** to the OSL routine **`ekksbnd`**.

Although the explanation of how OSL gives rhs range information for slack constraints is less then clear, one should remember that the ranges for the *parameter* values (as opposed to the *activity levels*) can be gotten from the marginal and level values for the rows in question, both of which are available inside the GAMS model.