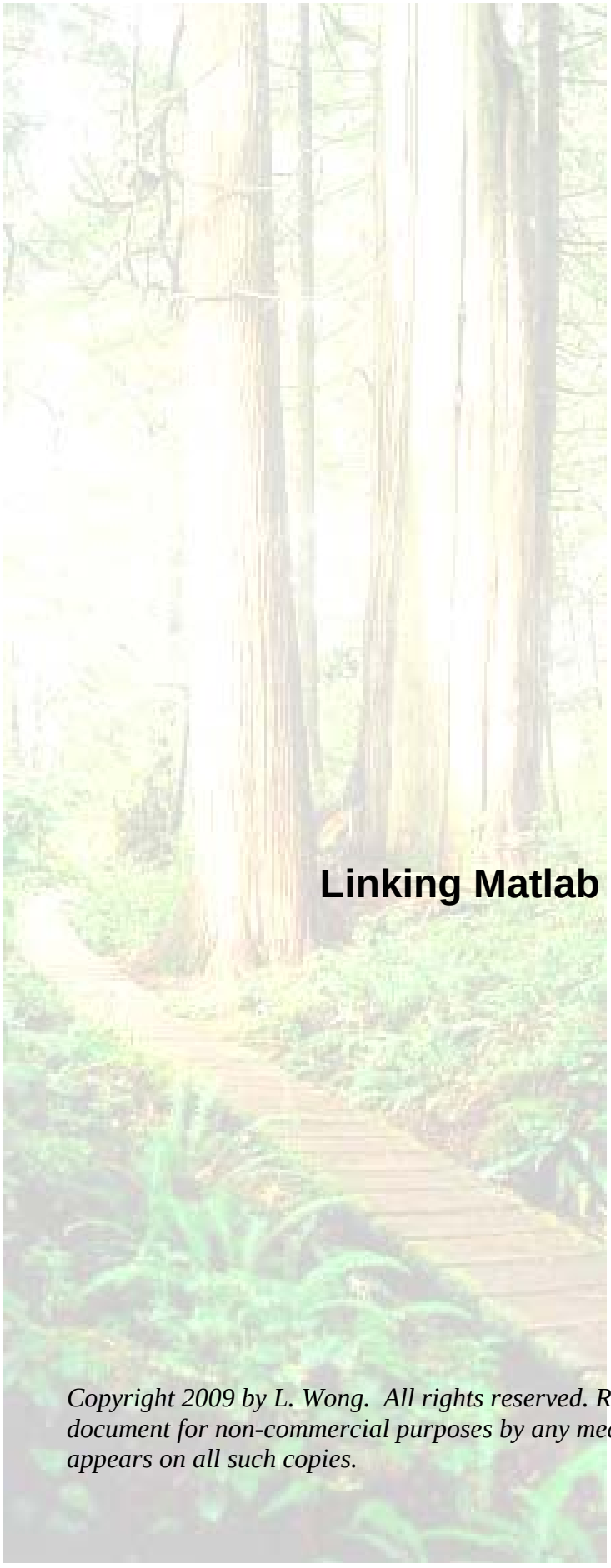

WORKING PAPER
2009-03

REPA

Resource Economics
& Policy Analysis
Research Group

Department of Economics
University of Victoria

Linking Matlab and GAMS: A Supplement

Linda Wong

June 2009

REPA Working Papers:

2003-01 – Compensation for Wildlife Damage: Habitat Conversion, Species Preservation and Local Welfare (Rondeau and Bulte)
2003-02 – Demand for Wildlife Hunting in British Columbia (Sun, van Kooten and Voss)
2003-03 – Does Inclusion of Landowners' Non-Market Values Lower Costs of Creating Carbon Forest Sinks? (Shaikh, Suchánek, Sun and van Kooten)
2003-04 – Smoke and Mirrors: The Kyoto Protocol and Beyond (van Kooten)
2003-05 – Creating Carbon Offsets in Agriculture through No-Till Cultivation: A Meta-Analysis of Costs and Carbon Benefits (Manley, van Kooten, Moeltne, and Johnson)
2003-06 – Climate Change and Forest Ecosystem Sinks: Economic Analysis (van Kooten and Eagle)
2003-07 – Resolving Range Conflict in Nevada? The Potential for Compensation via Monetary Payouts and Grazing Alternatives (Hobby and van Kooten)
2003-08 – Social Dilemmas and Public Range Management: Results from the Nevada Ranch Survey (van Kooten, Thomsen, Hobby and Eagle)
2004-01 – How Costly are Carbon Offsets? A Meta-Analysis of Forest Carbon Sinks (van Kooten, Eagle, Manley and Smolak)
2004-02 – Managing Forests for Multiple Tradeoffs: Compromising on Timber, Carbon and Biodiversity Objectives (Krcmar, van Kooten and Vertinsky)
2004-03 – Tests of the EKC Hypothesis using CO2 Panel Data (Shi)
2004-04 – Are Log Markets Competitive? Empirical Evidence and Implications for Canada-U.S. Trade in Softwood Lumber (Niquidet and van Kooten)
2004-05 – Conservation Payments under Risk: A Stochastic Dominance Approach (Benítez, Kuosmanen, Olschewski and van Kooten)
2004-06 – Modeling Alternative Zoning Strategies in Forest Management (Krcmar, Vertinsky and van Kooten)
2004-07 – Another Look at the Income Elasticity of Non-Point Source Air Pollutants: A Semiparametric Approach (Roy and van Kooten)
2004-08 – Anthropogenic and Natural Determinants of the Population of a Sensitive Species: Sage Grouse in Nevada (van Kooten, Eagle and Eiswerth)
2004-09 – Demand for Wildlife Hunting in British Columbia (Sun, van Kooten and Voss)
2004-10 – Viability of Carbon Offset Generating Projects in Boreal Ontario (Biggs and Laaksonen-Craig)
2004-11 – Economics of Forest and Agricultural Carbon Sinks (van Kooten)
2004-12 – Economic Dynamics of Tree Planting for Carbon Uptake on Marginal Agricultural Lands (van Kooten) (Copy of paper published in the Canadian Journal of Agricultural Economics 48(March): 51-65.)
2004-13 – Decoupling Farm Payments: Experience in the US, Canada, and Europe (Ogg and van Kooten)
2004–14– Afforestation Generated Kyoto Compliant Carbon Offsets: A Case Study in Northeastern Ontario (Biggs)
2005–01– Utility-scale Wind Power: Impacts of Increased Penetration (Pitt, van Kooten, Love and Djilali)
2005–02 –Integrating Wind Power in Electricity Grids: An Economic Analysis (Liu, van Kooten and Pitt)
2005–03 –Resolving Canada-U.S. Trade Disputes in Agriculture and Forestry: Lessons from Lumber (Biggs, Laaksonen-Craig, Niquidet and van Kooten)

2005–04–Can Forest Management Strategies Sustain the Development Needs of the Little Red River Cree First Nation? (Krcmar, Nelson, van Kooten, Vertinsky and Webb)
2005–05–Economics of Forest and Agricultural Carbon Sinks (van Kooten)
2005–06– Divergence Between WTA & WTP Revisited: Livestock Grazing on Public Range (Sun, van Kooten and Voss)
2005–07 –Dynamic Programming and Learning Models for Management of a Nonnative Species (Eiswerth, van Kooten, Lines and Eagle)
2005–08 –Canada-US Softwood Lumber Trade Revisited: Examining the Role of Substitution Bias in the Context of a Spatial Price Equilibrium Framework (Mogus, Stennes and van Kooten)
2005–09 –Are Agricultural Values a Reliable Guide in Determining Landowners' Decisions to Create Carbon Forest Sinks?* (Shaikh, Sun and van Kooten) *Updated version of Working Paper 2003-03
2005–10 –Carbon Sinks and Reservoirs: The Value of Permanence and Role of Discounting (Benitez and van Kooten)
2005–11 –Fuzzy Logic and Preference Uncertainty in Non-Market Valuation (Sun and van Kooten)
2005–12 –Forest Management Zone Design with a Tabu Search Algorithm (Krcmar, Mitrovic-Minic, van Kooten and Vertinsky)
2005–13 –Resolving Range Conflict in Nevada? Buyouts and Other Compensation Alternatives (van Kooten, Thomsen and Hobby) *Updated version of Working Paper 2003-07
2005–14 –Conservation Payments Under Risk: A Stochastic Dominance Approach (Benítez, Kuosmanen, Olschewski and van Kooten) *Updated version of Working Paper 2004-05
2005–15 –The Effect of Uncertainty on Contingent Valuation Estimates: A Comparison (Shaikh, Sun and van Kooten**)**
2005–16 –Land Degradation in Ethiopia: What do Stoves Have to do with it? (Gebreegziabher, van Kooten and.van Soest)
2005–17 –The Optimal Length of an Agricultural Carbon Contract (Gulati and Vercammen)
2006–01 –Economic Impacts of Yellow Starthistle on California (Eagle, Eiswerth, Johnson, Schoenig and van Kooten)
2006–02 -The Economics of Wind Power with Energy Storage (Benitez, Dragulescu and van Kooten)
2006–03 –A Dynamic Bioeconomic Model of Ivory Trade: Details and Extended Results (van Kooten)
2006–04 –The Potential for Wind Energy Meeting Electricity Needs on Vancouver Island (Prescott, van Kooten and Zhu)
2006–05 –Network Constrained Wind Integration: An Optimal Cost Approach (Maddaloni, Rowe and van Kooten)
2006–06 –Deforestation (Folmer and van Kooten)
2007–01 –Linking Forests and Economic Well-being: A Four-Quadrant Approach (Wang, DesRoches, Sun, Stennes, Wilson and van Kooten)
2007–02 –Economics of Forest Ecosystem Forest Sinks: A Review (van Kooten and Sohngen)
2007–03 –Costs of Creating Carbon Offset Credits via Forestry Activities: A Meta-Regression Analysis (van Kooten, Laaksonen-Craig and Wang)
2007–04 –The Economics of Wind Power: Destabilizing an Electricity Grid with Renewable Power (Prescott and van Kooten)
2007–05 –Wind Integration into Various Generation Mixtures (Maddaloni, Rowe and van Kooten)
2007–06 –Farmland Conservation in The Netherlands and British Columbia, Canada: A Comparative Analysis Using GIS-based Hedonic Pricing Models (Cotteleer, Stobbe and van Kooten)

2007–07 –Bayesian Model Averaging in the Context of Spatial Hedonic Pricing: An Application to Farmland Values (Cotteleer, Stobbe and van Kooten)
2007–08 –Challenges for Less Developed Countries: Agricultural Policies in the EU and the US (Schure, van Kooten and Wang)
2008–01 –Hobby Farms and Protection of Farmland in British Columbia (Stobbe, Eagle and van Kooten)
2008-01A-Hobby Farm's and British Columbia's Agricultural Land Reserve (Stobbe, Eagle, Cotteleer and van Kooten)
2008–02 –An Economic Analysis of Mountain Pine Beetle Impacts in a Global Context (Abbott, Stennes and van Kooten)
2008–03 –Regional Log Market Integration in New Zealand (Niquidet and Manley)
2008–04 –Biological Carbon Sequestration and Carbon Trading Re-Visited (van Kooten)
2008–05 –On Optimal British Columbia Log Export Policy: An Application of Trade theory (Abbott)
2008–06 –Expert Opinion versus Transaction Evidence: Using the Reilly Index to Measure Open Space premiums in the Urban-Rural Fringe (Cotteleer, Stobbe and van Kooten)
2008–07 –Forest-mill Integration: a Transaction Costs Perspective (Niquidet and O'Kelly)
2008–08 –The Economics of Endangered Species Poaching (Abbott)
2008–09 –The Ghost of Extinction: Preservation Values and Minimum Viable Population in Wildlife Models (van Kooten and Eiswerth)
2008–10 –Corruption, Development and the Curse of Natural Resources (Pendergast, Clarke and van Kooten)
2008–11 –Bio-energy from Mountain Pine Beetle Timber and Forest Residuals: The Economics Story (Niquidet, Stennes and van Kooten)
2008-12 –Biological Carbon Sinks: Transaction Costs and Governance (van Kooten)
2008-13 –Wind Power Development: Opportunities and Challenges (van Kooten and Timilsina)
2009-01 –Can Domestication of Wildlife Lead to Conservation? The Economics of Tiger Farming in China (Abbott and van Kooten)
2009-02 – Implications of Expanding Bioenergy Production from Wood in British Columbia: An Application of a Regional Wood Fibre Allocation Model (Stennes, Niquidet and van Kooten)
2009-03 – Linking Matlab and GAMS: A Supplement (Wong)

For copies of this or other REPA working papers contact:
REPA Research Group
Department of Economics
University of Victoria PO Box 1700 STN CSC Victoria, BC V8W 2Y2 CANADA
Ph: 250.472.4415
Fax: 250.721.6214
www.vkooten.net/repa

Linking Matlab and GAMS: A Supplement

Linda Wong

May 25, 2009

Contents

1. Introduction

1.1 Motivation

GAMS and Matlab are both valuable programs for solving optimization problems. The former has powerful nonlinear optimization capability, but no visualization tools, whereas the latter has imaging capabilities, but can sometimes be unsuitable for large-scale models (Ferris 2005). Hence, linking the two can allow each program to compensate for the deficiencies in the other. Currently, there are mechanisms for linking Matlab and GAMS, and the two methods developed by Michael C. Ferris, GDXMRW and MATGAMS, serve as the main references for this document and the corresponding interface.[1] Thus, what is supplied here can be viewed as supplementary to these existing tools.

As a brief overview, MATGAMS involves the use of either a dynamically linked library (DLL) or a Matlab executable (MEX) procedure to allow Matlab users to call GAMS as if it were a built-in function. Unfortunately, DLLs are no longer supported in new releases of Matlab, and there are some compatibility issues with the supplied MEX-files. GDXMRW provides a set of MEX procedures that read and write GAMS Data Exchange (GDX) files, which can be used to pass results of a GAMS model to different programs, and vice versa. Oddly enough, there were no compatibility issues with these MEX-files.

Here an additional MEX procedure has been created, henceforth referred to as *gams2*, which can invoke GAMS in a manner similar to MATGAMS, but uses GDXMRW to transfer data between the programs. As a result, there are many similarities between *gams2* and MATGAMS; therefore, credit is owed to M.C. Ferris (2005) for the concepts used in the design of this procedure. Any users interested in linking Matlab and GAMS are encouraged to use the tools available at

http://pages.cs.wisc.edu/~ferris/matlab.html

as a first resource. In terms of program execution time, *gams2* offers no substantial advantage over GDXMRW. As for program development, the use of *gams2* may offer greater efficiency relative to GDXMRW, but not to MATGAMS. The remainder of this document describes what is contained in *gams2* and provides examples of how it can be used. Also included are explanations for how certain functionality can be replicated without using *gams2*. Although care was taken to ensure accuracy, users are advised to refer to the original documentation by M.C. Ferris for official information on the procedures described here.

[1] Two methods are supplied by Michael Ferris at http://pages.cs.wisc.edu/~ferris/matlab.html

1.2 Basic Features and Limitations

Features
gams2 is capable of allowing users to run a GAMS program using the values of variables passed from the Matlab environment. Following GAMS notation, these variables include scalars, sets and parameters.[2] The ability to retrieve set elements prior to running the optimization is also available. Like MATGAMS, certain aspects of the procedure's default behavior may be modified.

Limitations
Because *gams2* relies on the GDXMRW procedure, *writegdx*, to convert Matlab variables into a format that can be read by GAMS, a very specific representation is required of parameters in particular. So, it is necessary to format input data matrices prior to invoking *gams2* or using *writegdx*. Additionally, there is no support for writing multi-dimensional sets and only arguments of type string are accepted.

In light of these limitations, the source code is provided for interested parties to modify as they wish. Further details on the features and limitations appear in the following sections. Section I is relatively technical as it provides background material and explains what goes into linking Matlab and GAMS. Section II is a demonstration with a wind energy model.[3]

Section I: Linking GAMS and Matlab

2. Installation (PC)

It is assumed that users are already familiar with the installation instructions for MATGAMS and GDXMRW.[4] The procedure for installing *gams2* is virtually identical to MATGAMS. That is, *gams2* and the GDXMRW utilities should be copied into a directory on your Matlab path, and the GAMS system directory should be added to your normal Windows path. The .m file that invokes *gams2* will need to be in the same directory as the .gms file. The accompanying MEX-file is platform dependent, so users may need to compile the MEX-function source code instead. To do this, the source code should be saved as gams2.c in a directory on your Matlab path and compiled with a command similar to

```
>> mex gams2.c
```

Please refer to the Matlab documentation for further information on compiling MEX-functions.

[2] For the remainder of this document, these terms will be used in the GAMS sense. That is, a one-dimensional parameter is dependent on one set in GAMS, a two-dimensional parameter depends on two sets, etc.
[3] By G. Cornelis van Kooten, University of Victoria
[4] See Ferris (2005) and http://www.gams.com/~steve/gdxmrw.html.

3. GDX, MEX, and GDXMRW: Basic Steps

3.1 GAMS Data Exchange (GDX)

This is the means by which input and output are transferred between the programs. With *gams2*, there is only one GDX procedure that the user needs to know. This procedure unloads GAMS output.

execute_unload 'filename.gdx' [symbol1, symbol2, …];

In the GAMS program, this statement should be placed after the solve statement. It writes the specified variables to filename.gdx. It should be noted that any existing content will be overwritten. By default, all variable attributes will be written to the GDX file, if they have been assigned values. It is straightforward to write a specific attribute only. For example, the statements

```
execute_unload 'file.gdx' var1.l, var2.m;
execute_unload 'file.gdx' var1.l=symb1, var2.m=symb2;
```

will write *var1*'s primal values and *var2*'s dual values.[5] However, domains may not be specified. To write a subset, a new parameter consisting of elements over the desired domain will need to be created. Alternatively, users may deal with this issue in Matlab once the results have been retrieved, but this is an inefficient use of memory and may lead to "out of memory" problems, especially when working with large data sets (MathWorks 2009).To write all identifiers, variables, and equations:

```
execute_unload 'file.gdx';
```

The above statements are execution time commands. It is not recommended to write data at compile time; however, this may done using $Gdxout and $Unload (see McCarl 2007). Users should note that the Matlab procedure used to read gdx files will only read sets and parameters. Thus, variable or equation attributes will need to be converted into parameters or scalars before they can be retrieved by Matlab.

To load input from Matlab, the statement

```
$if exist matdata.gms $include matdata.gms
```

should be inserted after all set and parameter declarations.[6] By default, the data is loaded at compile time, so any execution time commands may overwrite the Matlab input (Ferris 2005).

[5] These are automatically classified as "parameters." The second statement specifies the symbols by which the variables are to be referenced with.
[6] This is the technique employed by MATGAMS, and it is reproduced by *gams2*.

To load items at execution time, users may define *gams_input='exec'* in Matlab.[7] Sets must be loaded at compile time, however (McCarl 2007).

Supplemental Information

As mentioned in the introduction, it is not necessary to use *gams2* to link GAMS and Matlab. The same result can be achieved with a few more GDX procedures and essentially requires loading the desired variables before they are used in the GAMS program. For example:

```
$gdxin matdata.gdx      open the file to be read
$kill param1            reset identifier
$load param1            read in identifier
[$kill param2; …]
$gdxin                  close file
```

By default, the above code is more or less what appears in the *matdata.gms* file created by *gams2*.[8] Note that items are loaded at compile time. The statement

```
execute_load 'matdata.gdx' param1 [, param2, …];
```

will load items at execution time. It is straightforward to reproduce *matdata.gms* using a Matlab function/script each time the parameter to be loaded is changed.[9] Section 3.3 provides instructions on how to write a GDX file from Matlab.

3.2 Matlab Executable (MEX)

These utilities allow "custom C or Fortran subroutines to be called directly from Matlab as if they were built-in functions" (MathWorks 2009). The following describes three types of uses for *gams2*.

(1) Retrieving Sets: [index1, …, indexN, UEL]=gams2('filename', 'getsets', 'set1', …, 'setN');

It is often useful to retrieve set elements from GAMS prior to formatting the input data matrices. This can be done by calling *gams2* with the string 'getsets' as the second argument. The .gms extension for the filename is not required. The remaining string arguments need to match the set names assigned during the GAMS declarations.

[7] Same as MATGAMS.
[8] This is also what appears in the matdata.gms file created by MATGAMS.
[9] See source code for how this is programmed in C. It can easily be rewritten in Matlab code.

There are two types of output arguments when retrieving sets. *UEL* is the unique element list that GAMS creates when sets are declared (and UEL is a GAMS term). It is a vector containing the names of all set elements and is returned as the last output argument in the form of an (m×1) cell array of strings. All other output arguments are numeric matrices containing the indices of *UEL* that correspond to the elements of the specified sets. Note that the order of the output arguments must match the order of the input arguments. If no left-hand output arguments are specified, *gams2* will assign default names to the retrieved entries, which are then put into the Matlab environment. The default name for the UEL is *UEL*. For the indices, the default names will have *Indices* appended to the set name. Thus, any existing variables in the Matlab environment with these names will be overwritten if no left-hand output arguments are specified.

For *gams2* to successfully retrieve sets, the statement

```
$if exist matdata.gms $include matdata.gms
```

will need to be inserted into the GAMS program per the instructions in section 3.1. It will then unload all identifiers declared up to that point in the program. Note that right-hand arguments do not necessarily have to be set names; parameters can be retrieved as well.

(2) Modifying Values: gams2('filename' [, 'input1', …, 'inputN', 'UEL']);

This subroutine can be called after all the input data matrices have been assigned in Matlab. Again, the names given to the input arguments must match the declarations in the GAMS program. With the exception of scalars, *gams2* treats an (m×1) Matlab variable as a set. Scalars and all other (m×n) variables are treated as parameters. The reason is that *writegdx* requires parameters to be at least (m×2) and GAMS requires that one-dimensional sets are (m×1). According to the *writegdx* documentation, a parameter's values should be listed in column n with columns 1 through n-1 containing the indices of the n-1 sets associated with each of the data points. For example, if we want to provide input in a given hour (h) for generating plant (p), then the input is found in column 3 and h and p in columns 1 and 2. Note that GAMS has a limit of twenty dependent sets (McCarl 2007), so the input data matrices should not exceed (m×21).

To format Matlab variables into multi-dimensional parameters, GDXMRW includes the function *full2sp*. Examples will be provided in subsequent sections. When no UEL is specified, *writegdx* uses {'1'; '2'; '3'; … }. If this default UEL is not applicable in the user's GAMS program, it is recommended that the UEL and indices be retrieved using 'getsets' or *readgdx* rather than generating them in Matlab. Otherwise, the numerical indices used in the *writegdx* input matrices may be inconsistent with the GAMS-generated UEL, and, hence, inconsistent with the UEL and indices of the output matrices returned by *readgdx*. This is something to be aware of, but will not be much of an issue as long as users take care to pair the appropriate UEL with the indices. If the default UEL is applicable, users have the option of letting *gams2* format the input matrices by

defining *f2sp='yes'* in Matlab. This causes the procedure to call *full2sp* on the arguments prior to passing them to *writegdx*.

gams2 does not return any values when used in this manner. Thus, *readgdx* or the next use for *gams2* is required in Matlab to retrieve GAMS results.

(3) Retrieving Results: [symbol1, …, symbolN]=gams2('file.gdx', 'symbol1', …, 'symbolN');

When the first input argument has a .gdx extension, *gams2* will read the specified entries from the GDX file and apply *sp2full* to those results prior to returning them to Matlab. As explained in a later section, it will, in most cases, be more efficient to use *readgdx* to retrieve results from GAMS. Nevertheless, this feature is included for programs in which the default UEL is applicable. However, because the minimum matrix size is not specified when calling *sp2full*, this option will not function correctly if the last entry in any dimension is zero. In this case, users may opt to prevent *gams2* from calling *sp2full* by defining *sp2f='no'* in Matlab. The remaining input arguments are the GDX symbols.

Output arguments do not need to be specified when using *gams2* to retrieve results. The GDX symbols will be used as the names for the retrieved entries in this case. When output arguments are specified, the order of the output arguments must match the order of the input arguments.

Supplemental Information

Without *gams2*, users will need to create the GDX input file prior to executing a system call to GAMS. The procedure *writegdx* is explained in section 3.3. Then GAMS can be invoked using a statement like *system('gams filename lo=0')*. *lo=0* causes GAMS to suppress the output log, and this is the *gams2* default. Defining *logoption='file'* or *'stdout'* will change this option during *gams2* execution. These options correspond to *lo=2* and *lo=3* for the system call. The default is *lo=1*, which outputs the log to the Windows console. It is important to note that only sets and parameters can be written with *writegdx*. For scalar reassignments, users will need to classify the scalar as a parameter. One way to retrieve the UEL and set indices prior to creating the GDX input file is by placing the statements

```
execute_unload 'getsets.gdx';
$stop
```

after the set declarations. The GAMS program can then be run once to create the GDX file, and *readgdx* can be used to retrieve the data. This is how 'getsets' functions in *gams2*.

3.3 GDX Matlab Read/Write (GDXMRW)

GDXMRW is a set of four Matlab files: readgdx.mex_, writegdx.mex_, sp2full.m and full2sp.m. The last two files are utilities for manipulating the data so that they have the appropriate Matlab matrix structure.[10] These two utilities have only been partially incorporated into *gams2*, so, in most situations, users will need to know how to use *full2sp* if multi-dimensional data matrices are passed to GAMS and *sp2full* if results are anticipated to contain any zero entries. The syntax for calling *readgdx* to retrieve GAMS output is also needed.

[x, UEL] = readgdx('filename.gdx', 'symbol')

Only one GDX entry may be retrieved at a time. The quickest way to retrieve multiple entries is to create a cell array containing the list of symbols and then place the *readgdx* call within a *for* loop. Demonstrations are provided in Section II.

fullMatrix = sp2full(x, option, s)

x is a sparse matrix. *option* can either be 'set' or 'parameter'. *s* is the minimum matrix size, in every dimension, of *fullMatrix*. If the default UEL mentioned in the previous section is not applicable to the user's GAMS program, the indices in the sparse matrix returned by *readgdx* may need modification prior to calling *sp2full* in order to improve efficiency. For example, suppose these two GAMS programs are executed:

Ordering 1:

```
SETS
a /one, two/
b /1*1000/
;

PARAMETERS
paramAB(a,b) /two.set.b 1/
;

execute_unload 'output.gdx';
```

Ordering 2:

```
SETS
b /1*1000/
a /one, two/
;

PARAMETERS
paramAB(a,b) /two.set.b 1/
;

execute_unload 'output.gdx';
```

For both orderings, *readgdx* returns a (1000×3) sparse matrix for *paramAB*. The UEL will be (1002×1), but the numerical indices for set *a* in Ordering 1 will run from 1 to 2 whereas the indices will run from 1001 to 1002 in Ordering 2. If *sp2full* is called at this point, the resulting full matrix will be (2×1002) for Ordering 1 and (1002×1000) for Ordering 2. Then a further Matlab statement like

```
>> paramAB = fullMatrix(aIndices, bIndices);
```

[10] http://pages.cs.wisc.edu/~ferris/matlab.html

is needed for *paramAB* to be (2×1000). However, a statement similar to

```
>> spMatrix(:,2) = spMatrix(:,2) - max(aIndices);
```

prior to calling *sp2full* will result a (2×1000) full matrix for Ordering 1. For Ordering 2, the appropriate statement would be

```
>> spMatrix(:,1) = spMatrix(:,1) - max(bIndices);
```

Adjustments to higher dimensions are conducted in a similar manner and depend on the order of the set declarations. There are many ways to adjust the indices, but the object is to have each dimension begin at index 1 prior to calling *sp2full*. Similar steps are required for *full2sp*.

sparseMatrix = full2sp(fullMatrix, option, mask)

Again, *option* is either 'set' or 'parameter' with 'set' being the default. *mask* is on optional matrix of indices to extract. When option is 'parameter,' *full2sp* will, at minimum, return a (m×3) matrix (2D parameter). When writing one-dimensional parameters, it is not necessary to use *full2sp*. If the user opted to use the 'getsets' feature, the data matrix can be constructed as

```
>> fullMatrix = [aIndices data];
```

where *aIndices* is the set indices of *a* retrieved using 'getsets' and *data* is a column vector. *full2sp* can always be used to format input matrices, but if the default UEL is not applicable to the user's GAMS program, the indices will need to be modified so that they correspond to the UEL that is passed to *gams2* or *writegdx*. Using the same GAMS programs from the previous section, suppose *dataAB* is a (2×1000) full matrix, with no zero elements, that needs to be formatted before being passed to GAMS as *paramAB*. The statement

```
>> paramAB = full2sp(dataAB,'param');
```

will return a (2000×3) matrix in which each dimension begins at index 1. That is, *paramAB* will have the form [1, 1, dataAB(1,1); 2, 1, dataAB(2,1); 1, 2, dataAB(1,2); 2, 2, dataAB(2,2); …]. Again, suppose the identifier definitions in the GAMS programs are:

Ordering 1:

```
SETS
a /one, two/
b /1*1000/
;

PARAMETERS
paramAB(a,b) /two.set.b 1/
;
```

Ordering 2:

```
SETS
b /1*1000/
a /one, two/
;

PARAMETERS
paramAB(a,b) /two.set.b 1/
;
```

Assuming that the UEL has been retrieved from GAMS, the modification needed for *paramAB* in Ordering 1 is

```
>> paramAB(:,2) = paramAB(:,2) + max(aIndices);
```

For Ordering 2, the modification is

```
>> paramAB(:,1) = paramAB(:,1) + max(bIndices);
```

These modifications to the indices can be avoided if the GAMS program is created so that all sets begin at 1 and only contain elements with numerical counterparts. In this case, users have the option having *gams2* apply *full2sp* to the input matrices by defining *f2sp='yes'*. However, if this option is enabled, *gams2* will only be able to differentiate between scalars and parameters; therefore, sets cannot be passed to GAMS.

Supplemental Information

The syntax for calling *writegdx* is:

```
writegdx('file.gdx', 'datatype', 'symbol', dataMatrix
          [,'datatype2', 'symbol2', dataMatrix2, …, UEL]);
```

This statement writes each *dataMatrix* to *file.gdx*. With the exception of *dataMatrix*, which must be a numeric matrix, all arguments must be of type string. *datatype* may either be 'set' or 'parameter'. *symbol* is the name *dataMatrix* is to be identified with. According to the *writegdx* documentation, the procedure assumes that parameters have n-1 dimensions and cardinality m. Thus, the value is in column n, and columns 1 to n-1 contain its indices. Because *dataMatrix* must be numeric, optional labels can be passed to GAMS by specifying a unique element list, in the form of a cell array, for the last argument. If *UEL* is specified, then those element names will apply to each *dataMatrix* in the *writegdx* statement. Thus, if different sets of element names are desired for the data matrices, the user must ensure that the different groups of parameters have no overlapping numerical indices. To illustrate, consider a GAMS program which loads two parameters from one file and outputs the same parameters to a second file:

```
SETS
a /one, two, three/
b /four, five/
;

PARAMETERS
paramA(a)
/one   1
 two   2
 three 3/

paramB(b)
/four 4
 five 5/
;

execute_load 'test.gdx' paramA, paramB;

execute_unload 'output.gdx' paramA, paramB;
```

From Matlab, *writegdx* is called as follows:

```
Command Window
>> a=[1 2; 2 3; 3 4];
>> b=[4 5; 5 6];
>> writegdx('test.gdx', 'param', 'paramA', a, ...
            'param', 'paramB', b, {'one'; 'two'; 'three'; 'four'; 'five'});
>> system('gams t1.gms');
>> readgdx('output.gdx', 'paramA')
ans =
          1.00          2.00
          2.00          3.00
          3.00          4.00
>> readgdx('output.gdx', 'paramB')
ans =
          4.00          5.00
          5.00          6.00
```

It is important to note that the *UEL* must contain at least as many elements as the value of the largest numerical index. Suppose that the indices used for *b* were 99 and 100 instead of 4 and 5. If a *UEL* is specified, *writegdx* will require one hundred element names.

3.4 Summary

On the GAMS side, the interface involves:

1. Inserting *$if exist matdata.gms $include matdata.gms* after the identifier declarations.
2. Inserting the desired *execute_unload* command(s) after the solve statements.

On the Matlab side, the interface involves:

1. Formatting the data matrices and invoking *gams2*.
2. Using *readgdx* and *sp2full* to retrieve and format GAMS output.

Modifications to the default behavior include:

1. Defining gams_input = ‘exec’ to load Matlab input at execution time rather than at compile time.
2. Defining logoption = ‘file’ or ‘stdout’ to write the GAMS output log to a file or have it displayed, respectively.
3. Defining gams_show = ‘normal’ to show the Windows console upon invoking GAMS.
4. Defining f2sp = ’yes’ to apply *full2sp* to the input data matrices. This option should only be used if the default UEL is applicable to the user’s GAMS program.
5. Defining sp2f = ’no’ to prevent the application of *sp2full* to the output data matrices when using *gams2* to retrieve output.

Section II: Demonstrations – WindMix Model

4. Passing Values with Labels: WindMixDemo

GAMS

So that the GAMS program will load input passed from Matlab, the line

```
$if exist matdata.gms $include matdata.gms
```

should be added after the items being modified have been initialized and assigned, but before their first use. Once this line is inserted, it does not need to be removed even if subsequent uses of the *gams2* interface do not modify any values. This part of the GAMS program is as follows:

```
SETS
  h        hours             /1*744/
  p        Power plants      /hydro, nuke, coal, gas, peak/
;

SETS
  hfirst(h)  first hour
  hlast(h)   final hour
;

  hfirst(h)=yes$(ord(h) eq 1);
  hlast(h)=yes$(ord(h) eq card(h));

PARAMETERS
maxeff(p) Max power plant efficiencies
/hydro 1.0
 nuke  0.92
 coal  0.88
 gas   0.85
 peak  0.85/
;
*Variables and scalars related to Matlab
SCALAR fval;
SCALAR exitflag;
SCALAR carbtax;
PARAMETER demand(h);
PARAMETER fuel(p);
PARAMETER OMcost(p);
PARAMETER emission(p);
PARAMETER plantcap(p);
PARAMETER fixedcost(p);
PARAMETER ramup(p);
PARAMETER ramdown(p);
PARAMETER returnTotal(p,h);

$if exist matdata.gms $include matdata.gms
```

Again, GAMS variables will need to be formatted into scalars or parameters before they can be retrieved by *readgdx*. For example, to write the model status and solver status, statements like

```
Parameter returnStat;
returnStat('modelstat') = grid.modelstat;
returnStat('solvestat') = grid.solvestat;
```

should be placed directly after the solve statement for the model. In order to write the model results to a gdx file, the *execute_unload* command can be placed after the values to be returned to Matlab have been formatted. With the exception of the filename, the ordering of the items in the *execute_unload* statement is unimportant. For the WindMix model, the model status, objective value and total energy generation are returned to Matlab as follows:

```
Model grid /all/;

option lp=CPLEX;

Solve grid using lp minimizing z;

* Variables returned to Matlab
exitflag=grid.modelstat;
fval=z.l;
returnTotal(p,h)=total.l(p,h);

execute_unload 'output.gdx' exitflag, fval, returnTotal=total;
```

When there is an assignment, as in *returnTotal=total*, in the *execute_unload* statement, the item on the left-hand side of the equality is written to the GDX file with the item on the right-hand side as its symbol. Thus, in Matlab, *returnTotal* would be referenced by the symbol *total* when using *readgdx*.

Matlab

Before values are passed to GAMS, the input data matrices must be formatted so that the appropriate indices are included. For this process, it useful to retrieve the set indices and UEL from GAMS using

```
gams2('WindMixDemo', 'getsets', 'h', 'p')
```

which assigns those items the default names *hIndices*, *pIndices* and *UEL* when no left-hand output arguments are specified. It is also possible to retrieve sets in this manner:

```
[hIndices pIndices UEL]=gams2('WindMixDemo', 'getsets', 'h', 'p')
```

Note that the set identifiers supplied as the right-hand input arguments must match the identifiers used in the GAMS declarations. The order of the left-hand output arguments is important, and it needs to match the order of the right-hand input arguments. The last left-hand output argument must be the UEL. One-dimensional parameters may then be formatted as follows:

```
%Fixed O&M costs in $'000s per MW of installed capacity.
fixedcost = [492 800 220 105 84];
fixedcost = [pIndices fixedcost'];
```

No multi-dimensional parameters are passed to GAMS, but to illustrate how they would be formatted, suppose there are two additional sets and parameters:

```
SETS
  h        hours              /1*744/
  p        Power plants       /hydro, nuke, coal, gas, peak/
  q                           /100*103/
  r                           /a, b, c, d/
;

PARAMETERS
newParam1(p,h,q)
newParam2(q,p,r)
;
```

If the set indices are retrieved from GAMS, they will be 1-744, 745-749, 100-103 and 750-753 for *hIndices*, *pIndices*, *qIndices* and *rIndices*, respectively.[11] *full2sp* returns an (m×4) matrix when its input arguments are a 3D array and 'parameter.' If using the UEL retrieved from GAMS, the modifications to the indices include:

```
newParam1(:,1) = newParam1(:,1) + min(pIndices) - 1;
newParam1(:,3) = newParam1(:,3) + min(qIndices) – 1;

newParam2(:,1) = newParam2(:,1) + min(qIndices) - 1;
newParam2(:,2) = newParam2(:,2) + min(pIndices) - 1;
newParam2(:,3) = newParam2(:,3) + min(rIndices) - 1;
```

However, retrieving sets from GAMS is not necessary as long as users are careful to pair the correct UEL for the indices that are used when invoking *gams2* or *writegdx*. The GAMS optimization can be run using

```
gams2('WindMixDemo', 'carbtax', 'demand', 'fuel', 'OMcost', 'emission', ...
      'plantcap', 'fixedcost', 'ramup', 'ramdown', 'UEL')
```

Then, the results may be retrieved as follows:

```
exitflag = readgdx('output.gdx', 'exitflag');
fval = readgdx('output.gdx', 'fval');

temp = readgdx('output.gdx', 'total');
```

[11] GAMS stores elements in the UEL based on their order of appearance (McCarl 2007).

```
temp(:,1) = temp(:,1)-max(temp(:,2)); %modify indices before calling sp2full
total = sp2full(temp, 'param',[length(pIndices) length(hIndices)]);
clear temp;
```

To improve efficiency, the indices for the two-dimensional parameter, *total*, were modified prior to calling *sp2full*. Again, for large data sets, there is a risk of encountering an "out of memory" error if this is not done.

5. Passing Values without Labels: WindMixDemo_f2sp

GAMS

In this program, the sets are declared as

```
SETS
  h        hours            /1*744/
  p        Power plants     /1*5/
;
```

and all subsequent set references were changed accordingly.

Matlab

Because the default UEL will be used, the process for formatting the input data matrices can be simplified. One-dimensional parameters are specified as column vectors as follows:

```
%Fixed O&M costs in $'000s per MW of installed capacity.
fixedcost = [492 800 220 105 84]';
```

Two-dimensional parameters are specified as two-dimensional matrices, and so on. No additional formatting is required because *gams2* can apply *full2sp* before creating the GDX file:

```
f2sp='y'; %allow gams2 to apply full2sp

%Run GAMS optimization
gams2('WindMixDemo_f2sp', 'carbtax', 'demand', 'fuel', 'OMcost', ...
      'emission', 'plantcap', 'fixedcost', 'ramup', 'ramdown')
```

Users also have the option of using *gams2* to retrieve the results of the GAMS optimization:

```
%Since no left-hand output arguments are specified,
%the results will be named exitflag, fval and total
gams2('output.gdx', 'exitflag', 'fval', 'total')
```

When used in this manner, the items returned to Matlab are the output arguments of *sp2full*. To obtain the *readgdx* output arguments instead, define *sp2f='no'* prior to invoking *gams2*.

6. Troubleshooting[12]

The GAMS exit codes that are identified include

```
case 0: Normal completion
case 2: Compilation error(s)
case 6: File not found or permission denied
```

All other exit codes are considered by *gams2* to be execution errors. When the user encounters an exit code of 2, the most likely reason are misnamed parameters or mismatched dimensions.

An error code of 3 normally indicates an arithmetic error arising from division by zero, improper exponentiation, etc.

When an exit code of 6 is encountered, the user should ensure that the filename is specified correctly and that both the Matlab and GAMS files are in the current working directory. Otherwise, the user will need to ensure that he or she has the proper directory permissions.

In all cases, it is useful to review the LST and/or LOG file for further details.

Additionally, users may encounter a "Warning: File not found or permission denied" message when using the 'getsets' option. This will happen if *readgdx* does not terminate properly. The message is unimportant and can be disregarded. However, to correct the issue, users will need to exit and reopen Matlab. Alternatively, users may opt to merely disable the warning using

```
>> warning off MATLAB:DELETE:Permission
```

Acknowledgments

The author would like to thank Professor G. C. van Kooten for supplying the WindMix model and contributing to the improvement of this document.

[12] Most of this information can be found in McCarl (2007)

References

Ferris, Michael C. *MATLAB and GAMS: Interfacing Optimization and Visualization Software.* Computer Sciences Department, University of Wisconsin-Madison, 2005.

MathWorks, Inc. *MATLAB® Getting Started Guide.* Natick, 2009.

McCarl, Bruce A. "McCarl Expanded GAMS User Guide Version 22.5." Texas A&M University, 2007.