GDXMRW: Exchanging Data Between GAMS and Matlab

Steve Dirkse sdirkse@gams.com

GAMS Development Corporation

www.gams.com

INFORMS San Diego 11 October 2009

Welcome/Agenda

- Background: why & what
- readgdx: overview and examples
- writegdx: overview and examples

Joint work with Michael Ferris, Rishabh Jain

GAMS Philosophy 101

- Layered architecture with separation of:
  - Model and data
  - Model and user interface
- Open architecture and interfaces to other systems
  - GDX (Gams Data eXchange) – data hugely important
  - GDX tools (from GAMS and 3rd parties)
  - GDX API to exchange data with other apps

GDX Advantages

- Platform independent
- Fully precise – data are stored in a binary format
- Efficient
  - takes advantage of sparsity
  - does multiple types of compression
- Standards-based: format is documented, tested, supported, and used in many applications
- Language independent: published interfaces for many languages, e.g. Fortran, C/C++, C#, Java, VB, Delphi
- Validated data
  - no syntax errors on read
  - consistent: no duplicates, contradictions, etc.

Why Matlab?

- Data analysis and visualization
- Matrix computations
  - Inverses, factorizations, matrix orderings
  - Eigenvalues, eigenvectors
- Programming – prototyping and algorithm development
- Application-specific toolboxes, 3rd-party apps
  - Statistics, curve-fitting
  - Finance
  - Computation biology

The GAMS Data Format

- GAMS & GDX use a relational data model
  - Parameters and sets are indexed by *labels*, not integers
  - The union of labels used forms an ordered universe
- Data is stored in *sparse form*
- Set data – a collection of labels is a set
  - One-dimensional sets make the foundation
  - *N*-dimensional tuples build (sub)sets from this
- Parameter data – numeric
  - Behave like *N*-dim sets, but with values
- Special values – INF, eps, NA (missing)

GAMS Data - Example

```
SET people /
   annika, elise, michelle, valerie
   shanti, kaoru, susan, tessa /;
SET friends(people,people) /
   annika.shanti
   elise.susan
   michelle.kaoru
   valerie.tessa /;
```

Matlab Data Format

- Rectangular matrix – the basic Matlab data type
  - Indexed by integers: 1..m -by- 1..n
  - Values stored are numbers (doubles)
  - *N*-d matrices also allowed
  - Stored in a *dense* form by default
  - Sparse storage optional for 2-d matrices
  - No scalars or 1-d matrices
- String or cell data
  - Not a primary data type in Matlab
  - used to map to/from GAMS
  - Strings are meta-data that go along with an index

Design goals & issues

- Simplicity: make the obvious choice
- Convenience: give the users what they want
- Efficiency
  - Matlab arrays are stored *dense* – major issue
  - Efficiency trumps simplicity
  - Perhaps a trade-off with convenience
- Index mapping
  - Matlab integers vs. GAMS labels
  - Matlab string data available to help bridge the gap
  - *Ordering* becomes an issue with labels – the GAMS universe ordering can conflict with Matlab orderings

- Background: why & what
- readgdx: overview and examples
- writegdx: overview and examples

Readgdx – basic syntax

- **`r = rgdx('filename.gdx',o)`**, where
  - **`r`** is a structure containing the result, i.e. output data
  - **`o`** is a structure of options controlling the read
- Each call reads a single set or parameter
  - Reading GDX this way is efficient and simple
- Option fields include
  - **`name= 'xxxxx'`** – required, symbol name in GDX
  - **`form= 'full'`** or **`'sparse'`** – default sparse
  - **`compress= 'true'`** or **`'false'`** – default false
  - **`uels`**: optional filter/mapping for each dimension
- Result fields include
  - **`val`**: the Matlab matrix, in full or sparse form
  - **`uels`**: the GAMS labels mapped to each dimension

Readgdx – r.val and r.uels

- The interesting results are the values and UELs

```
Set i / seattle, sandiego /
        j / topeka, chicago /;
Set link(i,j) 'set to read' /
   seattle.topeka, seattle.chicago,
   sandiego.topeka, sandiego.chicago /;
```

```
>> r.val, r.uels, r.uels{1}

ans =

     0     0     1     1
     0     0     1     1
     0     0     0     0
     0     0     0     0


ans =

    {1x4 cell}    {1x4 cell}


ans =

    'seattle'    'sandiego'    'topeka'    'chicago'
```

Readgdx – full vs. sparse

```
link(i,j) 'set to read' /
  seattle.topeka, seattle.chicago,
  sandiego.topeka, sandiego.chicago /;
```

```
ans =

     0     0     1     1
     0     0     1     1
     0     0     0     0
     0     0     0     0
```

Full (i.e dense) result

```
ans =

     1     3
     1     4
     2     3
     2     4
```

Sparse result

Readgdx – compressed reads

- A compressed read removes empty rows/cols from the result
  - This saves significant space
  - The UELs corresponding to the remaining indices are returned in r.uels
  - The result may be more convenient also
- Our example from the previous slides returns:
  - a 2x2 dense matrix of ones as r.val
  - r.uels{1} = ‘seattle’,’sandiego’ - the sources
  - r.uels{2} – ‘topeka’,’chicago’ – the sinks

Readgdx – filtered reads

- A filter is a list of labels - values whose labels are outside of this list are ignored.
- Filters are specified using the **`uels`** field of the options
  - Each dimension has its own filter
  - To filter nothing out, use the universe as a filter
  - The result **`uels`** field duplicates the input options
- Filters provide additional flexibility
  - E.g. compressed reads remove all zero rows/cols
  - Filters can reorder data as well as filter it

Readgdx – filtered reads

- The Matlab code below does a filtered read on our example data, returning an empty set.

```
o.name = 'link';
dom = cell(1,2);
dom{1} = { 'chicago','topeka' };
dom{2} = { 'seattle','sandiego' };
o.uels = dom;
r = rgdx('rgdx.gdx',o);
```

A Difficult Model

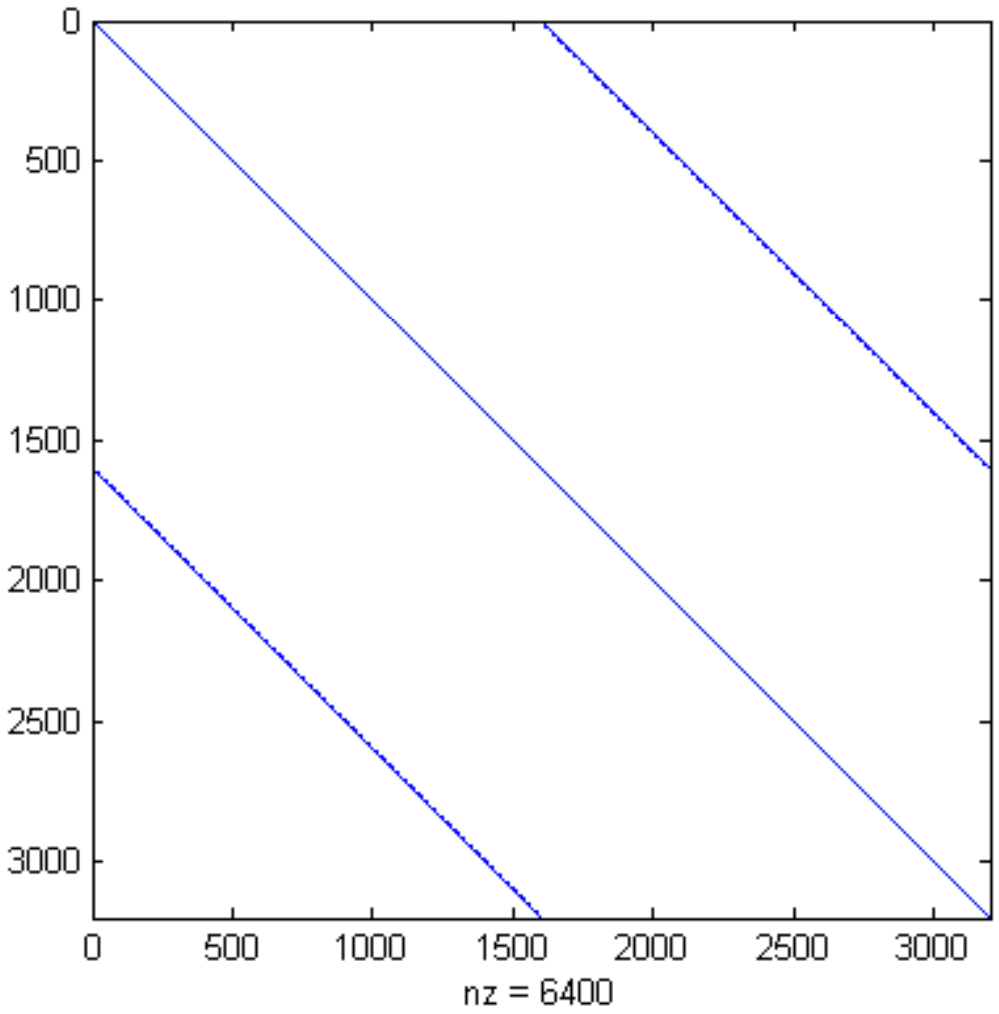
```
eqV1(m1,m2)..
V1(m1,m2) =e=
* profit1(m1,m2)
  q1(m1,m2)*(A - q1(m1,m2) - q2(m1,m2)) - sqr(q1(m1,m2))/ord(m1)
  - (gamma1*U1(m1,m2) + phi1*sqr(U1(m1,m2)))
  + beta*sum(n1(m1,m1p),
      (((1-delta)*alpha*U1(m1,m2))$(sameas(m1p,m1+1) and ord(m1) lt %n%)
      + (1-delta+delta*alpha*U1(m1,m2))$(sameas(m1p,m1) and ord(m1) lt %n% and ord(m1) gt 1)
      + (1+delta*alpha*U1(m1,m2))$(sameas(m1p,m1) and ord(m1) eq 1)
      + (1-delta+alpha*U1(m1,m2))$(sameas(m1p,m1) and ord(m1) eq %n%)
      + delta$(sameas(m1p,m1-1) and ord(m1) gt 1))
    *sum(n2(m2,m2p),
      (((1-delta)*alpha*U2(m1,m2))$(sameas(m2p,m2+1) and ord(m2) lt %n%)
      + (1-delta+delta*alpha*U2(m1,m2))$(sameas(m2p,m2) and ord(m2) lt %n% and ord(m2) gt 1)
      + (1+delta*alpha*U2(m1,m2))$(sameas(m2p,m2) and ord(m2) eq 1)
      + (1-delta+alpha*U2(m1,m2))$(sameas(m2p,m2) and ord(m2) eq %n%)
      + delta$(sameas(m2p,m2-1) and ord(m2) gt 1))
      *V1(m1p,m2p)))/((1 + alpha*U1(m1,m2))*(1 + alpha*U2(m1,m2)));

eqV2(m1,m2)..
V2(m1,m2) =e=
* profit2(m1,m2)
  q2(m1,m2)*(A - q1(m1,m2) - q2(m1,m2)) - sqr(q2(m1,m2))/ord(m2)
  - (gamma2*U2(m1,m2) + phi2*sqr(U2(m1,m2)))
  + beta*sum(n1(m1,m1p),
      (((1-delta)*alpha*U1(m1,m2))$(sameas(m1p,m1+1) and ord(m1) lt %n%)
      + (1-delta+delta*alpha*U1(m1,m2))$(sameas(m1p,m1) and ord(m1) lt %n% and ord(m1) gt 1)
      + (1+delta*alpha*U1(m1,m2))$(sameas(m1p,m1) and ord(m1) eq 1)
      + (1-delta+alpha*U1(m1,m2))$(sameas(m1p,m1) and ord(m1) eq %n%)
      + delta$(sameas(m1p,m1-1) and ord(m1) gt 1))
    *sum(n2(m2,m2p),
      (((1-delta)*alpha*U2(m1,m2))$(sameas(m2p,m2+1) and ord(m2) lt %n%)
      + (1-delta+delta*alpha*U2(m1,m2))$(sameas(m2p,m2) and ord(m2) lt %n% and ord(m2) gt 1)
      + (1+delta*alpha*U2(m1,m2))$(sameas(m2p,m2) and ord(m2) eq 1)
      + (1-delta+alpha*U2(m1,m2))$(sameas(m2p,m2) and ord(m2) eq %n%)
      + delta$(sameas(m2p,m2-1) and ord(m2) gt 1))
      *V2(m1p,m2p)))/((1 + alpha*U1(m1,m2))*(1 + alpha*U2(m1,m2)));

eqV1foc(m1,m2)..
-(-(gamma1 + 2*phi1*U1(m1,m2))
  + beta*sum(n1(m1,m1p),
```

Readgdx - Visualizing the Jacobian

- Use CONVERT to dump the Jacobian to GDX
- A few Matlab lines give us an interesting picture

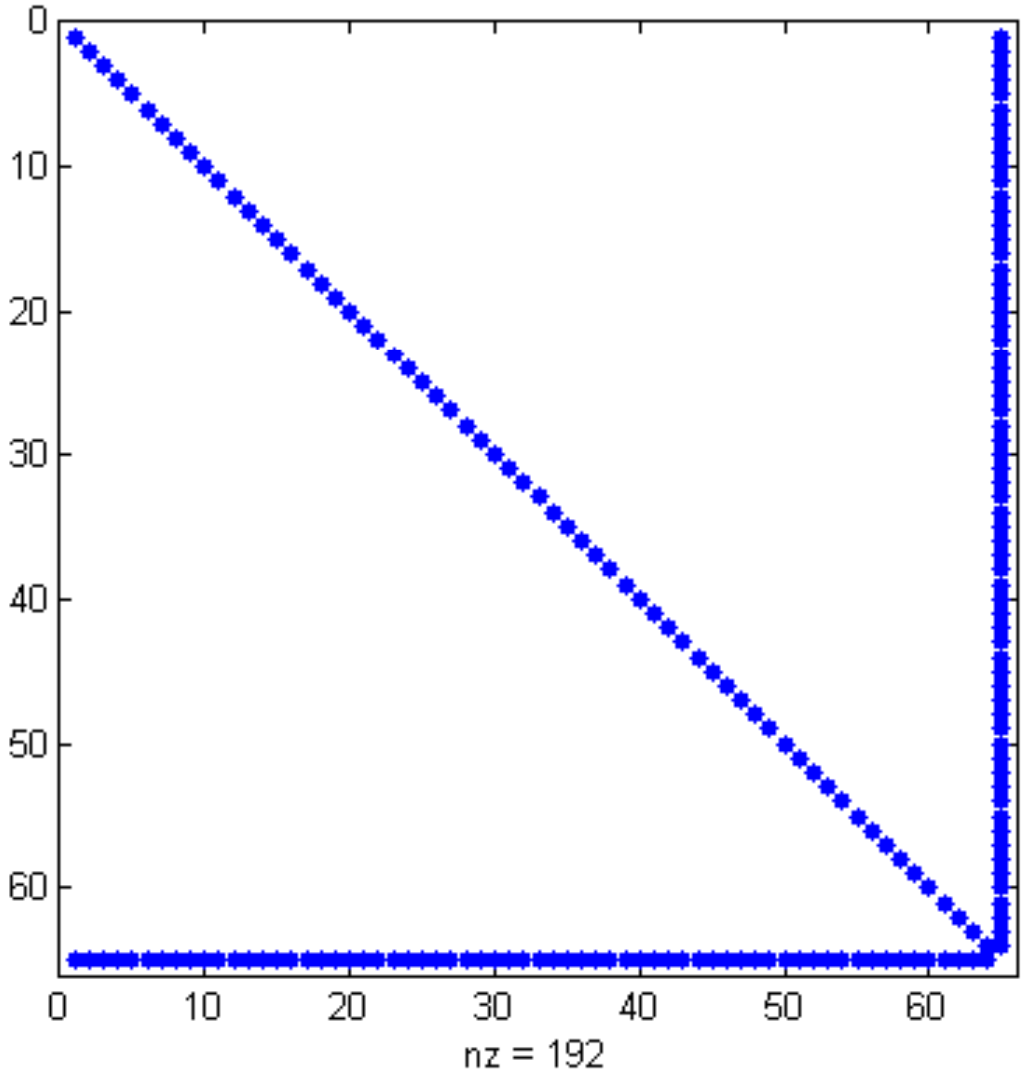
```
u = readgdx('dyngame.gdx');      universe = u.uels;
rFlags.name = 'I';
i = readgdx('dyngame.gdx',rFlags);
rFlags.name = 'J';
j = readgdx('dyngame.gdx',rFlags);
Adom{1} = {universe{i.val}};
Adom{2} = {universe{j.val}};
rFlags.name = 'A';
rFlags.uels = Adom;
rFlags.form = 'full';
A = readgdx('dyngame.gdx',rFlags);
% spy(A.val)
```

The Jacobian Revealed



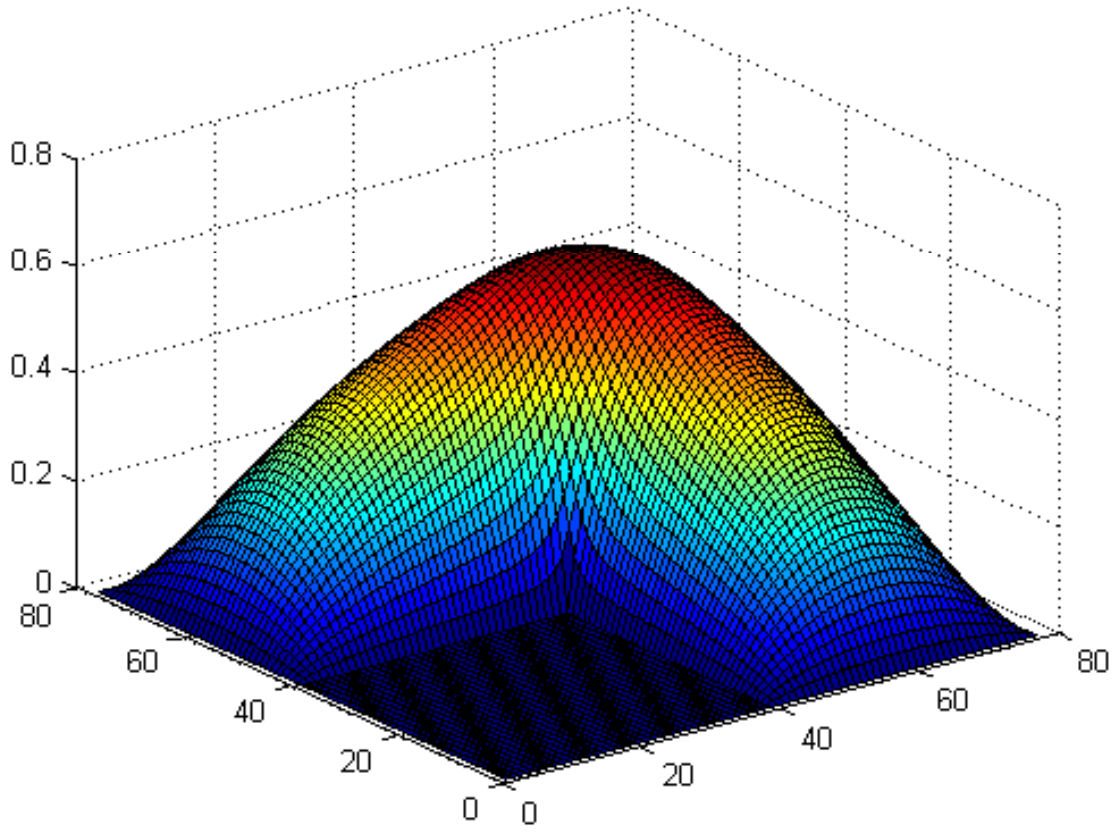
Another Model, Another Jacobian

```
o.name = 'A';
o.form = 'full';
o.compress='true'
A = rgdx(‘jac’,o)
spy(A.val);
```



Visualizing a surface

```
fname = 'bratu';
u = rgdx(fname);
uni = u.uels;
o.name = 'I';
i = rgdx(fname,o);
o.name = `J';
j = rgdx(fname,o);
o.name = 'vsol';
o.form = 'full';
d{1} = {uni{i.val}};
d{2} = {uni{j.val}};
o.uels = dom;
v = rgdx(fname,o);
surf(v.val)
```



- Background: why & what
- readgdx: overview and examples
- writegdx: overview and examples

Writegdx – basic syntax

- **`wgdx('filename.gdx',d1,d2,…,dn)`**, where
  - **`di`** is a structure containing a symbol to write
  - One call writes the entire GDX file
- Data fields include
  - **`name= 'xxxx'`** – required, symbol name in GDX
  - **`type= 'set'`** or **`'parameter'`** – required
  - **`form= 'full'`** or **`'sparse'`** – default sparse
  - **`val:`** the Matlab matrix, in full or sparse form
  - **`uels`**: optional mapping for each dimension

Writegdx – d.val and d.uels

- The interesting inputs are the values and UELs
- For example, how can we write a GDX equivalent to the data below:

```
Set i / seattle, sandiego /
    j / topeka, chicago /;
Set link(i,j) /
  seattle.topeka, seattle.chicago,
  sandiego.topeka, sandiego.chicago /;
```

Writegdx – Simple Example

```
i.val  = [1:2]';
i.uels = { 'seattle', 'sandiego' };

j.val  = [1:2]';
j.uels = { 'topeka', 'chicago' };

link.val  = [1 1 ; 1 2 ; 2 1 ; 2 2];
link.uels{1} = i.uels;
link.uels{2} = j.uels;

wgdx('wgdx', i, j, link);
```

Writegdx – Alternate Example

```
i.val  = [1:2]';
i.uels = { 'seattle', 'sandiego' };

j.val  = [4:5]';
j.uels = { ‘ignore’, ‘this’, ‘unused’, 'topeka', 'chicago' };

link.val  = [1 4 ; 1 5 ; 2 4 ; 2 5];
link.uels{1} = i.uels;
link.uels{2} = j.uels;

wgdx('wgdx', i, j, link);
```

Genetics Example – Old

```
[s,metabs,rxns,output]=SIMPHENY2GAMS('Sim12');

% remove possible strings that are duplicate
[uels,I,J] = unique( lower({ metabs{:} rxns{:} })', 'first' );
s1 = length(metabs); s2 = length(rxns);
% array J: mapping between origin numbers and UEL list
m1 = J(1:s1); m2 = J(s1+1:s1+s2);
[r,c,v] = find(s); smat = [J(r) J(c+s1) v];
writegdx('data.gdx','set','metabs',m1,'set','rxns',m2,'par','s',s
   mat,uels);
```

Genetics Example – New

```
[s,metabs,rxns,output]=SIMPHENY2GAMS('Sim12');
s1 = length(metabs); s2 = length(rxns);
d1.name = 'metabs';  d1.type = 'set';
d1.val = [1:s1]';
d1.uels = metabs';
d2.name = 'rxns';    d2.type = 'set';
d2.val = [1:s2]';
d2.uels = rxns';
d3.name = 's';       d3.type = 'parameter';
d3.val  = s;
d3.uels = { metabs' rxns' };
wgdx('newData', d1, d2, d3);
```

Concluding Remarks

- Updated tools improve on convenience and simplicity
- Additional enhancements are in progress
  - Reading equations and variables
  - Writing Matlab logicals and integers
  - Suggestions welcome!!
- GDXMRW is available for download:
  - www.gams.com/~steve/gdxmrw
- We invite & encourage you to use these tools