gdxrrw:
Exchanging Data Between GAMS and R

S. Dirkse¹  M. C. Ferris  R. van Nieuwkoop

¹GAMS Development Corp
Washington, DC
sdirkse@gams.com

INFORMS Phoenix: Oct 2012
Outline

1 Background: Why & What
   • Foundation
   • Why R?
   • What is GDX?

2 Basic Usage
   • Reading data
   • Usage detail
   • Writing data

3 Example Applications
   • Euro TSP
   • Visualization with maps

4 Conclusion
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: careful coding, sparsity, compression
- Supported by freely available utilities
- Standards-based: format is documented, tested, supported, and used in many applications
- Language independent: published interfaces for many languages, e.g. C/C++, C#, Java, VB, Python, Delphi
- Validated data
  - No syntax errors in input
  - consistent: no duplicates, contradictions, etc.
R Advantages

- R is a powerful, feature-packed software package
  - Statistics
  - Data analysis, manipulation, and visualization
  - Programming - prototyping and development
  - Application-specific packages: thousands available
    - More statistics
    - Finance
    - Computational biology / bioinformatics (*Bioconductor*)
- R is free and easy to install, update, and augment
- R is fun to use: *TMTOWTDI*
  - There’s more than one way to do it
Software used in data analysis competitions in 2011

Source: http://r4stats.com/articles/articles/popularity
### Language Use Survey

What programming languages you used for data mining / data analysis in the past 12 months? [570 voters]

<table>
<thead>
<tr>
<th>Language</th>
<th>Users</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>R (257)</td>
<td></td>
<td>45%</td>
</tr>
<tr>
<td>SQL (184)</td>
<td></td>
<td>32%</td>
</tr>
<tr>
<td>Python (140)</td>
<td></td>
<td>25%</td>
</tr>
<tr>
<td>Java (139)</td>
<td></td>
<td>24%</td>
</tr>
<tr>
<td>SAS (121)</td>
<td></td>
<td>21%</td>
</tr>
<tr>
<td>MATLAB (83)</td>
<td></td>
<td>15%</td>
</tr>
<tr>
<td>C/C++ (73)</td>
<td></td>
<td>13%</td>
</tr>
<tr>
<td>Unix shell/awk/gawk/sed (59)</td>
<td></td>
<td>10%</td>
</tr>
<tr>
<td>Perl (45)</td>
<td></td>
<td>7.9%</td>
</tr>
<tr>
<td>Hadoop/Pig/Hive (35)</td>
<td></td>
<td>6.1%</td>
</tr>
<tr>
<td>Lisp (4)</td>
<td></td>
<td>0.7%</td>
</tr>
<tr>
<td>Other (70)</td>
<td></td>
<td>12.0%</td>
</tr>
<tr>
<td>None (7)</td>
<td></td>
<td>1.2%</td>
</tr>
</tbody>
</table>
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)
**rgdx Introduction**

- Reads one symbol per call
- `rgdx` argument list controls what is read and how
- Both sparse and full forms allowed for data output
- Many other options exist for handling special cases
  - user-defined UEL filters: limit and/or reorder the data
  - compression: removes zero rows and columns in the data
  - Options for handling GAMS special values like EPS
- `rgdx.set`, `rgdx.param` are convenience wrappers
  - Output is a data frame with domain sets as factors
  - Sparse format: one row per nonzero
  - Implemented in R source, they call `rgdx`
Generating a GDX file

Sets
  i       'canning plants' / seattle, san-diego, monterey /
  j       'markets' / new-york, chicago, topeka, phoenix /
  ii(i)   'active plants' / seattle, san-diego /
  jj(j)   'active markets' / new-york, chicago, topeka /
  ij(i,j) 'open routes';

ij(i,j) = yes;

Parameters
  a(i) 'capacity of plant i in cases' /
       seattle 350
       san-diego 600
       monterey 400
       /
  b(j) 'demand at market j in cases' /
       new-york 325
       chicago 300
       topeka 275
       phoenix 375
       /;

Table d(i,j) 'distance in 1K miles'

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
<th>phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.404</td>
<td>1.733</td>
<td>1.455</td>
<td>1.112</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.429</td>
<td>1.729</td>
<td>1.274</td>
<td>0.298</td>
</tr>
<tr>
<td>monterey</td>
<td>2.570</td>
<td>1.856</td>
<td>1.435</td>
<td>0.594</td>
</tr>
</tbody>
</table>

Scalar f 'freight: $/case/1K miles' /90/ ;

execute_unload 'trans';
### Loading the package

```r
library(gdxrrw)
igdx()

## The GDX library has been loaded
## GDX library load path: /gams_64/phoenix

rgdx("?")

## R-file source info: Id: gdxrrw.c 35863 2012-10-18 17:54:51Z sdirkse

(fdf <- rgdx.scalar("trans", "f"))

## [1] 90
## attr(,"symName")
## [1] "f"
```
```r
info <- gdxInfo("trans", dump = F, returnDF = T)
info[c("sets", "parameters")]
```

### $sets

<table>
<thead>
<tr>
<th>name</th>
<th>index</th>
<th>dim</th>
<th>card</th>
<th>text</th>
<th>doms</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>canning plants</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>markets</td>
<td>0</td>
</tr>
<tr>
<td>ii</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>active plants</td>
<td>1</td>
</tr>
<tr>
<td>jj</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>active markets</td>
<td>2</td>
</tr>
<tr>
<td>ij</td>
<td>5</td>
<td>2</td>
<td>12</td>
<td>open routes</td>
<td>1, 2</td>
</tr>
</tbody>
</table>

### $parameters

<table>
<thead>
<tr>
<th>name</th>
<th>index</th>
<th>dim</th>
<th>card</th>
<th>text</th>
<th>doms</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>capacity of plant i in cases</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>demand at market j in cases</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>8</td>
<td>2</td>
<td>12</td>
<td>distance in 1K miles</td>
<td>1, 2</td>
</tr>
<tr>
<td>f</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>freight: $/case/1K miles</td>
<td></td>
</tr>
</tbody>
</table>
rgdx.set: reading sets as dataframes

(idf <- rgdx.set('trans','i'))

##
## i
## 1 seattle
## 2 san-diego
## 3 monterey

str(idf) ; idf$i

## 'data.frame': 3 obs. of 1 variable:
## $ i: Factor w/ 7 levels "seattle","san-diego",...: 1 2 3
## - attr(*, "symName")= chr "i"
## - attr(*, "domains")= chr "*

## [1] seattle san-diego monterey
## Levels: seattle san-diego monterey new-york chicago topeka phoenix
rgdx.set: reading subsets as dataframes

(iidf <- rgdx.set('trans','ii'))

##
## i
## 1  seattle
## 2  san-diego

str(iidf) ; iidf$i

## 'data.frame': 2 obs. of 1 variable:
## $ i: Factor w/ 3 levels "seattle","san-diego",..: 1 2
## - attr(*, "symName")= chr "ii"
## - attr(*, "domains")= chr "i"

## [1] seattle  san-diego
## Levels: seattle san-diego monterey
rgdx: reading sets as lists

```r
jlst <- rgdx("trans", list(name = "j"))
str(jlst)
```

## List of 7
### $ name : chr "j"
### $ type : chr "set"
### $ dim : int 1
### $ val : num [1:4, 1] 4 5 6 7
### $ form : chr "sparse"
### $ uels : List of 1
#### ..$: chr [1:7] "seattle" "san-diego" "monterey" "new-york" ...
### $ domains: chr "*"
rgdx: reading sets as lists

```r
jlst[c("val", "uels")]
```

```r
# $val
# [,1]
# [1,] 4
# [2,] 5
# [3,] 6
# [4,] 7

# $uels
# $uels[[1]]
# [1] "seattle" "san-diego" "monterey" "new-york" "chicago" "topeka"
# [7] "phoenix"
```
rgdx: reading subsets as lists

jjlst <- rgdx("trans", list(name = "jj"))
jjlst[c("val", "uels", "domains")]

## $val
## [,1]
## [1,] 1
## [2,] 2
## [3,] 3

## $uels
## $uels[[1]]
## [1] "new-york" "chicago" "topeka" "phoenix"

## $domains
## [1] "j"
rgdx.set: sparse form

(ijdf <- rgdx.set("trans", "ij"))

```r
## i   j
## 1  seattle new-york
## 2  seattle chicago
## 3  seattle topeka
## 4  seattle phoenix
## 5  san-diego new-york
## 6  san-diego chicago
## 7  san-diego topeka
## 8  san-diego phoenix
## 9  monterey new-york
## 10 monterey chicago
## 11 monterey topeka
## 12 monterey phoenix
```
rgdx.set: sparse form

```r
str(ijdf)
```

```r
## 'data.frame': 12 obs. of  2 variables:
## $ i: Factor w/ 3 levels "seattle","san-diego",..: 1 1 1 1 2 2 2 2 3 3 ...
## $ j: Factor w/ 4 levels "new-york","chicago",..: 1 2 3 4 1 2 3 4 1 2 ...
## - attr(*, "symName")= chr "ij"
## - attr(*, "domains")= chr  "i" "j"

ijdf$i

```r
## [1] seattle seattle seattle seattle seattle san-diego san-diego san-diego san-diego san-diego monterey monterey monterey monterey
## Levels: seattle san-diego monterey
```
**rgdx: full form**

```r
ijlst <- rgdx("trans", list(name = "ij", form = "full"))
# str(ijlst)
ijlst$val
```

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
<th>phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>san-diego</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>monterey</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
rgdx.param: reading parameters as dataframes

```r
adf <- rgdx.param("trans", "a")
```

```r
## i value
## 1 seattle 350
## 2 san-diego 600
## 3 monterey 400
```

```r
str(adf)
```

```r
## 'data.frame': 3 obs. of 2 variables:
## $ i     : Factor w/ 3 levels "seattle","san-diego",...: 1 2 3
## $ value: num 350 600 400
## - attr(*, "symName")= chr "a"
## - attr(*, "domains")= chr "i"
```
rgdx: reading parameters as lists

```r
blst <- rgdx("trans", list(name = "b"))
str(blst)
```

### List of 7

## $ name : chr "b"
## $ type : chr "parameter"
## $ dim : int 1
## $ val : num [1:4, 1:2] 1 2 3 4 325 300 275 375
## $ form : chr "sparse"
## $ uels : List of 1
## ..$ : chr [1:4] "new-york" "chicago" "topeka" "phoenix"
## $ domains: chr "j"
rgdx: reading parameters as lists

blst[c("val", "uels")]

## $val
## [,1] [,2]
## [1,] 1 325
## [2,] 2 300
## [3,] 3 275
## [4,] 4 375

## $uels
## $uels[[1]]
## [1] "new-york" "chicago" "topeka" "phoenix"
### rgdx.param: sparse form

```r
(ddf <- rgdx.param("trans", "d"))
```

```
#  i     j       value
#  1 seattle new-york 2.404
#  2 seattle chicago 1.733
#  3 seattle topeka 1.455
#  4 seattle phoenix 1.112
#  5 san-diego new-york 2.429
#  6 san-diego chicago 1.729
#  7 san-diego topeka 1.274
#  8 san-diego phoenix 0.298
#  9 monterey new-york 2.570
# 10 monterey chicago 1.856
# 11 monterey topeka 1.435
# 12 monterey phoenix 0.594
```
Usage detail

**rgdx: full form**

dlst <- rgdx("trans", list(name = "d", form = "full"))
# str(dlst)
dlst$val

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
<th>phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.404</td>
<td>1.733</td>
<td>1.455</td>
<td>1.112</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.429</td>
<td>1.729</td>
<td>1.274</td>
<td>0.298</td>
</tr>
<tr>
<td>monterey</td>
<td>2.570</td>
<td>1.856</td>
<td>1.435</td>
<td>0.594</td>
</tr>
</tbody>
</table>
Introduction to \texttt{wgdx}

- All symbols are written at once: no appending to existing GDX files is supported
- Think of \texttt{wgdx} as the inverse of \texttt{rgdx}
  - The same data is written & read
  - Each works with data in full or sparse form
- \texttt{wgdx} is the basic function
- various convenience wrappers exist
Reproducing the input

```
jout <- list(name='j', form='sparse', uels=jlst$uels, val=jlst$val)
jjout <- jjlst ; jjout$domains <- NULL
bout <- list(name='b', uels=blst$uels, val=blst$val, type='parameter')
dout <- list(name='d', form='full', uels=dlst$uels, val=dlst$val, type='parameter')
wgdx.lst('tmp', list(fdf, idf, iidf, jout, jjout, ijdf, adf, bout, dout))
system('gdxdiff trans tmp')
```

- Data frames work as inverses wrt. input and output
Visualizing a TSP on R’s eurodist data

- Start with R’s eurodist data
- Write data (cities and distances) to GDX
- Solve TSP in GAMS
- Dump tour to GDX
- Read tour into R
- Plot cities and tour
TSP model in GAMS

```gams
sets   ii 'set of cities';
parameter c(ii,ii) 'inter-city distances';

$gdxin eurodist
$load ii=cities c=dist
$gdxin

$include tsp_dseX

scalars modelstat, solvestat;
modelstat = DSE.modelstat;
solvestat = DSE.solvestat;
execute_unload 'eurosol', modelstat, solvestat, ii, tour;
```
Write GDX and run GAMS

```r
fnData <- "eurodist.gdx"
fnSol <- "eurosol.gdx"
invisible(suppressWarnings(file.remove(fnData, fnSol)))
n <- attr(eurodist, "Size")
cities <- attr(eurodist, "Labels")
uu <- list(c(cities))
dd <- as.matrix(eurodist)
clst <- list(name='cities', type='set', uels=uu,
             ts='cities from stats::eurodist')
dlst <- list(name='dist', type='parameter', dim=2, form='full',
             ts='distance', val=dd, uels=c(uu, uu))
wgdx (fnData, clst, dlst)
gams('tsp_dse.gms')
```

## [1] 0
v <- rgdx(fnSol, list(name = "tour", form = "sparse"))
nxt <- v$val[, 2]
# compute the sequence of cities, based on nxt
solSeq <- NA * c(1:n + 1)
k <- 1
for (j in c(1:n)) {
  solSeq[j] <- k
  k <- nxt[k]
}
solSeq[n + 1] <- 1
if (k != 1) stop("Bogus tour specified")
loc <- cmdscale(eurodist)
rx <- range(x <- loc[, 1])
ry <- range(y <- -loc[, 2])
tspres <- loc[solSeq, ]
s <- seq(n)
Plot tour

```r
plot(x, y, type="n", asp=1, xlab="", ylab="")
arrows(tspres[s,1], -tspres[s,2], tspres[s+1,1], -tspres[s+1,2],
       angle=10, col="blue")
text(x, y, labels(eurodist), cex=0.8)
```
Optimal TSP Tour

The diagram illustrates the optimal tour for the TSP (Traveling Salesman Problem) in Europe, with cities such as Athens, Barcelona, Brussels, Calais, Cherbourg, Cologne, Copenhagen, Geneva, Gibraltar, Hamburg, Hook of Holland, Lisbon, Lyons, Madrid, Marseilles, Milan, Munich, Paris, Rome, Stockholm, and Vienna. The coordinates range from -2000 to 2000 on both axes.
Visualization with maps

Integrating model results with maps

- Start with model results: regions and regional data
- Read result into R
- Use R plotting capability to generate map outline
  - R package "maps" includes US state data
  - similar regional data are available for all countries
- Add pie charts to map to show regional data
Send model results to GDX

```plaintext
sets
  s 'states to map'
  c 'commodities produced'
;
parameters
  sPrd(c,s) 'scaled production'
  w(s)   'scale factors based on total production'
;
$include genStateDataX

execute_unload 'stateRes', s, c, sPrd, w;
```
if (!file.exists("stateRes.gdx")) invisible(gams("genStateData.gms"))

source("statePieChart.R")
Visualization with maps

Commodity production, by state
library(maps)
fnData <- "stateRes.gdx"
sdf <- rgdx.set(fnData, "s")
cdf <- rgdx.set(fnData, "c")
s <- (rgdx(fnData, list(name = "sPrd", form = "full")))$val
w <- (rgdx(fnData, list(name = "w", form = "full")))$val
slist <- as.vector(sdf$i)
ns <- length(slist)  # number of states
idx <- match(slist, state.name)
x <- state.center$x[idx]
y <- state.center$y[idx]
nc <- dim(cdf)[1]  # number of commodities: each commodity gets a color
colors <- rainbow(nc)
# define the maximum size of the chart elements
minradius <- +Inf
for (i in 1:ns) {
  for (j in 1:ns) {
    if (i != j) {
      r <- (x[i] - x[j])^2 + (y[i] - y[j])^2
      minradius <- min(minradius, r)
    }
  }
}
minradius <- sqrt(minradius)

np <- 200  # define the number of polygon points on the arc
npp <- np + 1
n <- np - 1
xx <- rep(0, npp)
yy <- xx
rad0 <- max(0.4, 0.5 * minradius)  # maximum radius of pie chart
Visualization with maps

Plot states and pie charts

```r
cmap("state", interior = FALSE, lty=2)
cmap("state", boundary = TRUE, add = TRUE, region=slist)
for (k in 1:ns) {
  xx[npp] <- x[k]
  yy[npp] <- y[k]
  xfac <- 1 / cos((y[k]*pi)/180)
  r <- rad0 * w[k] ; beta <- 0
  for (c in 1:nc) {
    alpha <- s[c,k]*2*pi
    for (i in 1:np) {
      xx[i] <- xx[npp] + r*sin(beta + alpha *(i-1)/n)*xfac
      yy[i] <- yy[npp] + r*cos(beta + alpha *(i-1)/n)
    }
    beta <- beta + alpha
    coord <- cbind(xx,yy)
    polygon (coord, col=colors[c])
  }
}
```
Background: Why & What

Basic Usage

Example Applications

Conclusion

Visualization with maps

Plot legend

```r
xpac <- -120 ; ypac <- 30       # a nice pacific location
xfac <- 1 / cos((ypac*pi)/180)
xleg <- vector(mode='numeric',length=nc)
yleg <- vector(mode='numeric',length=nc)
cleg <- vector(mode="character",length=nc)
xx[npp] <- xpac ; yy[npp] <- ypac
r <- rad0 ; beta <- 0 ; for (c in 1:nc) {
  alpha <- (2/3)*pi ; theta <- beta + alpha * 0.5
  xleg[c] <- xpac + 2.5*r*sin(theta)
  yleg[c] <- ypac + 1.6*r*cos(theta)
  cleg[c] <- paste0("commodity",as.character(c))
  cleg[c] <- as.character(cdf$i[c])
  for (i in 1:np) {
    xx[i] <- xx[npp] + r*sin(beta + alpha *(i-1)/n)*xfac
    yy[i] <- yy[npp] + r*cos(beta + alpha *(i-1)/n)  }
  beta <- beta + alpha ; coord <- cbind(xx,yy)
  polygon (coord,col=colors[c])
  text(xleg,yleg,labels=cleg,col=colors)
}
```
Concluding Remarks

- **gdxrrw** bridges the gap between R and GAMS
  - Fits into the ecosystem of existing GDX utilities
  - Presents data in a natural form for R users
- Development/enhancement is ongoing
  - Reading/writing equations and variables
  - Suggestions welcome!!
- For more information and to get started, visit:
  - [http://support.gams.com/doku.php?id=gdxrrw:interfacing_gams_and_r](http://support.gams.com/doku.php?id=gdxrrw:interfacing_gams_and_r) (Wiki, downloads, FAQ, etc.)
  - [http://www.gams.com/download](http://www.gams.com/download) (free GAMS downloads)
  - [http://blog.modelworks.ch](http://blog.modelworks.ch) (Renger’s blog)