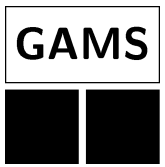
gdxrrw: Exchanging Data Between GAMS and R

S. Dirkse[1] M. C. Ferris R. van Nieuwkoop

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

ICS Santa Fe: Jan 2013



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

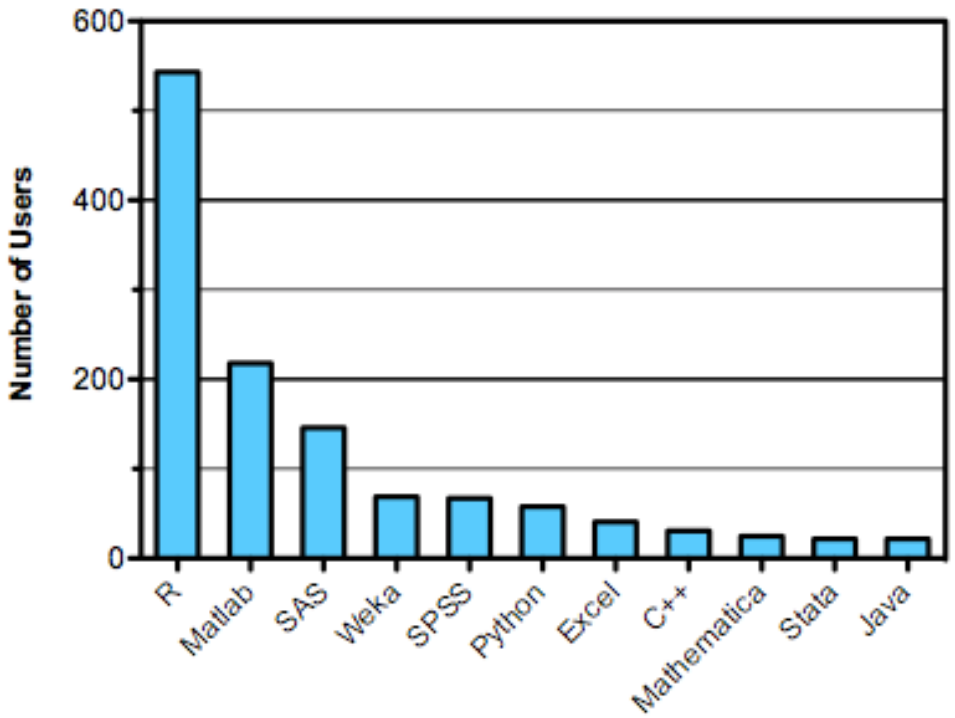
GAMS Philosophy 101

- 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

R Advantages

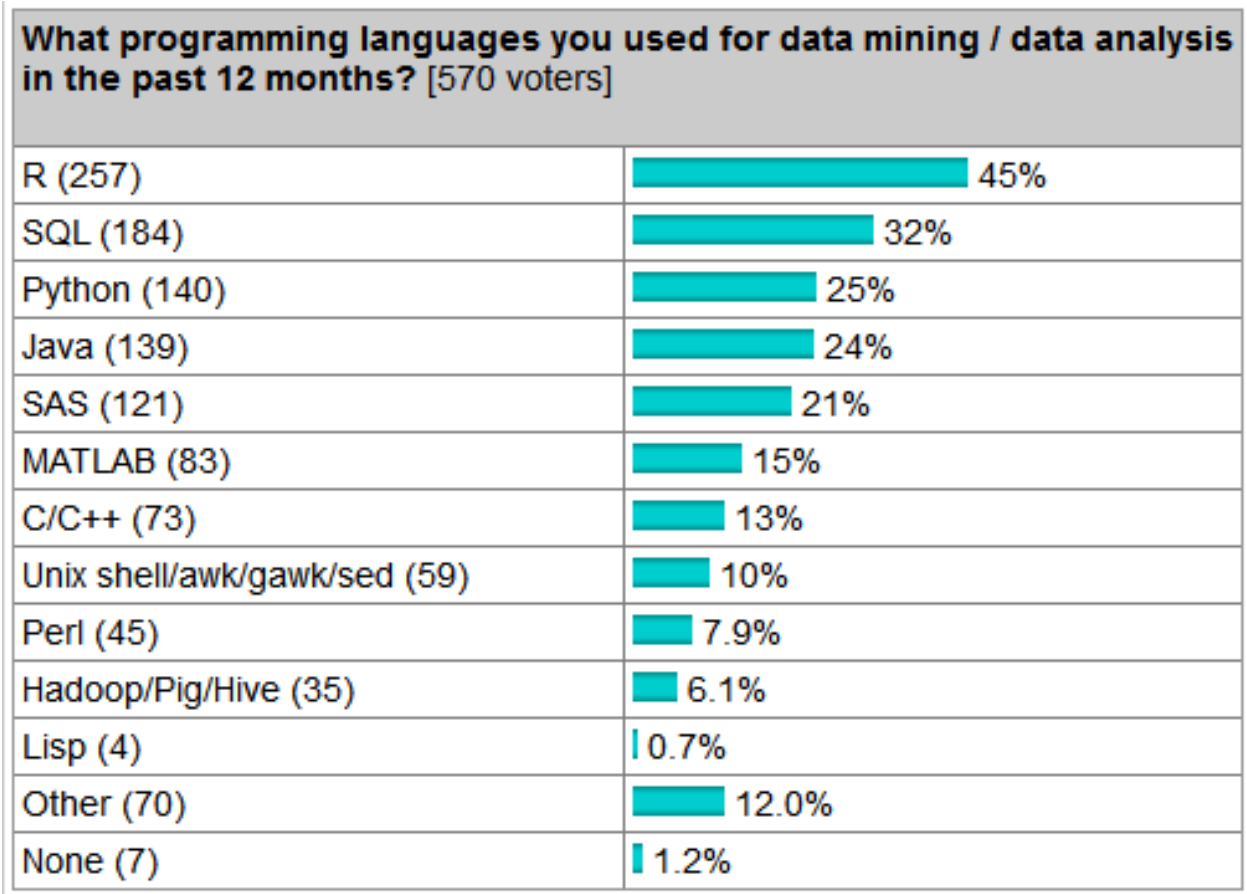
- R is a powerful, feature-packed software package
  - Statistics
  - Data analysis, manipulation, and visualization
  - Programming - prototyping and development
  - Application-specific packages: thousands available
    - More statistics
    - Finance
    - Computational biology / bioinformatics (*Bioconductor*)
- 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 Use in Data Analysis Competitions



- Software used in data analysis competitions in 2011
- Source: `http://r4stats.com/articles/popularity`

Language Use Survey



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

R (257)	45%
SQL (184)	32%
Python (140)	25%
Java (139)	24%
SAS (121)	21%
MATLAB (83)	15%
C/C++ (73)	13%
Unix shell/awk/gawk/sed (59)	10%
Perl (45)	7.9%
Hadoop/Pig/Hive (35)	6.1%
Lisp (4)	0.7%
Other (70)	12.0%
None (7)	1.2%

- Language use survey
- Source: `http://r4stats.com/articles/popularity`

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.

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 , santaFe /
  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
    santaFe  375
  /;
Table d(i,j) 'distance in 1K miles'
            new-york    chicago     topeka    santaFe
seattle        2.404      1.733      1.455      1.176
san-diego      2.429      1.729      1.274      0.670
monterey       2.570      1.856      1.435      0.890  ;

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

Loading the package

```
library(gdxrrw)
igdx()

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

rgdx("?")

## R-file source info: Id: gdxrrw.c 36421 2012-11-10 00:18:39Z sdirkse

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

## [1] 90
## attr(,"symName")
## [1] "f"
```

gdxInfo: reading meta-data

```
info <- gdxInfo("trans", dump = F, returnDF = T)
info[c("sets", "parameters")]

## $sets
##   name index dim card           text doms
## 1    i     1   1    3 canning plants    0
## 2    j     2   1    4        markets    0
## 3   ii     3   1    2  active plants    1
## 4   jj     4   1    3 active markets    2
## 5   ij     5   2   12    open routes 1, 2
##
## $parameters
##   name index dim card                        text doms
## 1    a     6   1    3 capacity of plant i in cases    1
## 2    b     7   1    4  demand at market j in cases    2
## 3    d     8   2   12         distance in 1K miles 1, 2
## 4    f     9   0    1     freight: $/case/1K miles
```

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 santaFe
```

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

```
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

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

## $val
##      [,1]
## [1,]    4
## [2,]    5
## [3,]    6
## [4,]    7
##
## $uels
## $uels[[1]]
## [1] "seattle"   "san-diego" "monterey"  "new-york"  "chicago"   "topeka"
## [7] "santaFe"
```

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"   "santaFe"
##
##
## $domains
## [1] "j"
```

rgdx.set: sparse form

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

```
##            i        j
## 1    seattle new-york
## 2    seattle  chicago
## 3    seattle   topeka
## 4    seattle  santaFe
## 5  san-diego new-york
## 6  san-diego  chicago
## 7  san-diego   topeka
## 8  san-diego  santaFe
## 9   monterey new-york
## 10  monterey  chicago
## 11  monterey   topeka
## 12  monterey  santaFe
```

rgdx.set: sparse form

```
str(ijdf)

## '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

##  [1] seattle   seattle   seattle   seattle   san-diego san-diego san-die
##  [8] san-diego monterey  monterey  monterey  monterey
## Levels: seattle san-diego monterey
```

rgdx: full form

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

##           new-york chicago topeka santaFe
## seattle          1       1      1       1
## san-diego        1       1      1       1
## monterey         1       1      1       1
```

rgdx.param: reading parameters as dataframes

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

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

str(adf)

## '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

```
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" "santaFe"
##  $ 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"   "santaFe"
```

rgdx.param: sparse form

```
(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  santaFe 1.176
## 5   san-diego new-york 2.429
## 6   san-diego  chicago 1.729
## 7   san-diego   topeka 1.274
## 8   san-diego  santaFe 0.670
## 9    monterey new-york 2.570
## 10   monterey  chicago 1.856
## 11   monterey   topeka 1.435
## 12   monterey  santaFe 0.890
```

rgdx: full form

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

##           new-york chicago topeka santaFe
## seattle      2.404   1.733  1.455   1.176
## san-diego    2.429   1.729  1.274   0.670
## monterey     2.570   1.856  1.435   0.890
```

Introduction to wgdx

- All symbols are written at once: no appending to existing GDX files is supported
- Think of `wgdx` as the inverse of `rgdx`
  - The same data is written & read
  - Each works with data in full or sparse form
- `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

```
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

```
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
```

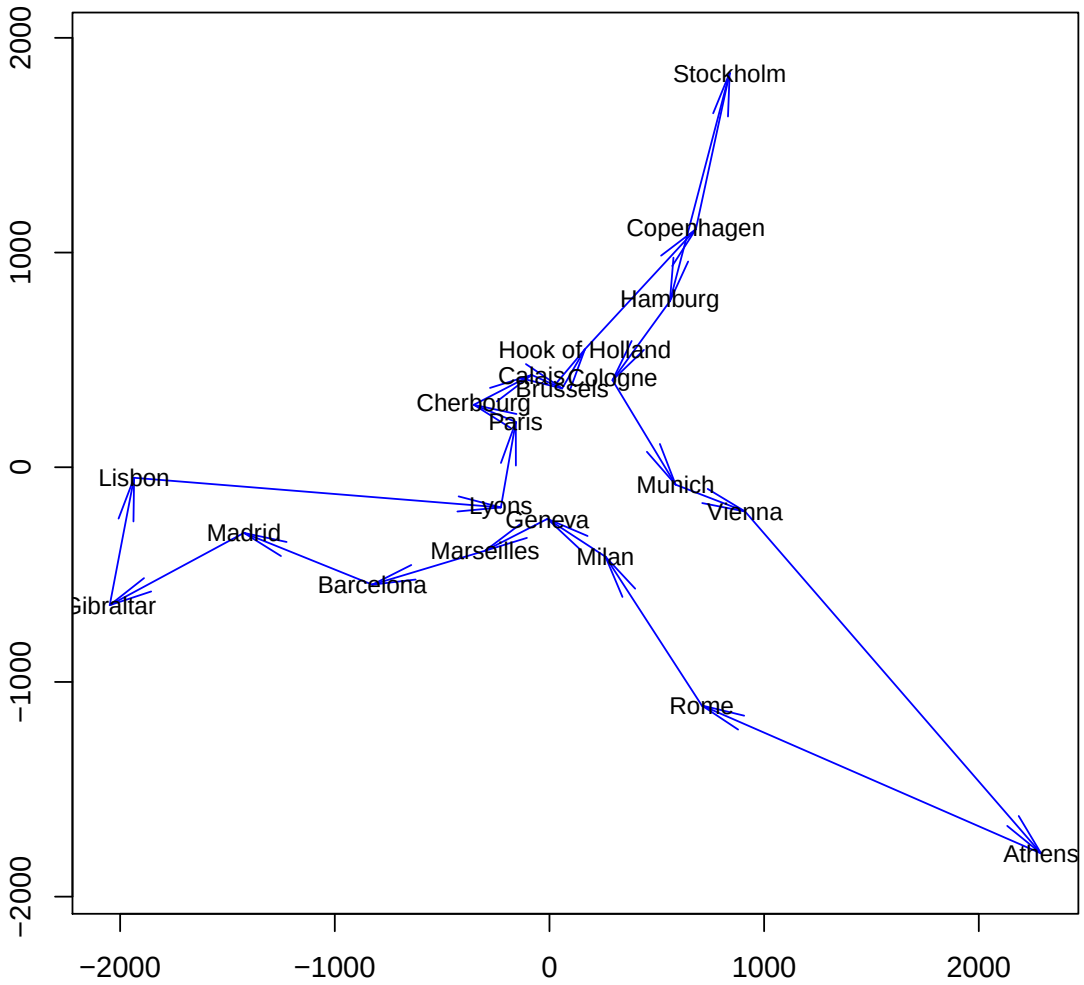
Read tour and compute locations

```
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

```
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



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

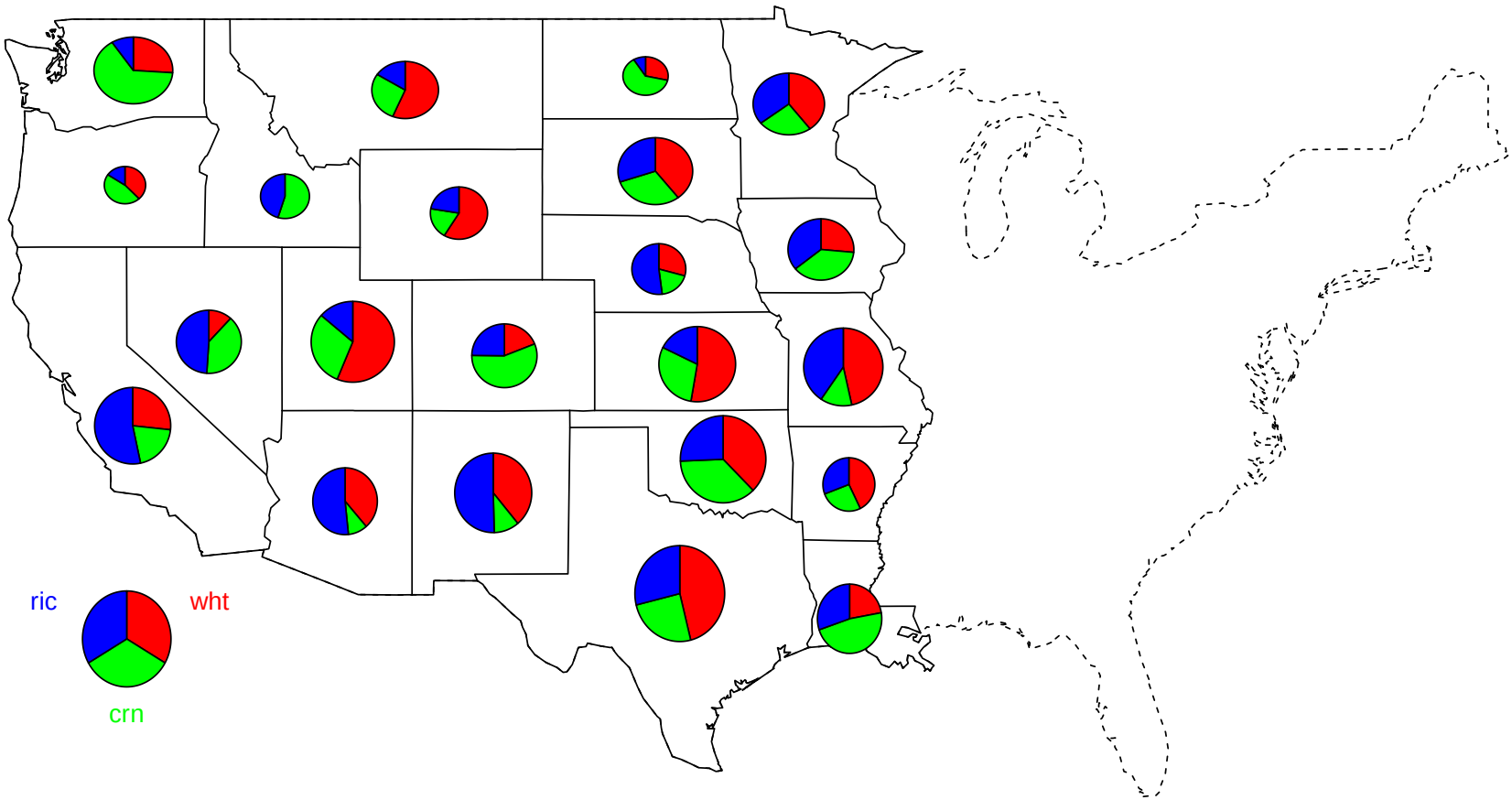
```
sets
   s 'states to map'
   c 'commodities produced'
   ;
parameters
   sPrd(c,s) 'scaled production'
   w(s)      'scale factors based on total production'
   ;

$include genStateDataX

execute_unload 'stateRes', s, c, sPrd, w;
```

Generate output

```
if (!file.exists("stateRes.gdx")) invisible(gams("genStateData.gms"))
```

```
source("statePieChart.R")
```

Commodity production, by state



Reading results and data prep

```
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)
```

data prep

```
# 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
```

Plot states and pie charts

```
map("state", interior = FALSE,lty=2)
map("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])
  }
}
```

Plot legend

```
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` (Wiki, downloads, FAQ, etc.)
  - `http://www.gams.com/download` (free GAMS downloads)
  - `http://blog.modelworks.ch` (Renger's blog)