cnvGSA: Gene-Set Analysis of (Rare) Copy Number Variants

Joseph Lugo, Daniele Merico and Robert Ziman
The Centre for Applied Genomics
joseph.r.lugo@gmail.com, daniele.merico@gmail.com,
robert.ziman@gmail.com

April 8, 2015

Contents

1 Overview

cnvGSA is an R package for testing the rare gene-set variant burden in case-control studies of copy number variation (CNV). Earlier versions of cnvGSA, or highly similar burden tests, have been used in several high profile studies of rare CNV in neuropsychiatric disorders [1][2][3][4].

Only rare CNV (e.g. at frequency <1%) should be present in the input data: burden tests are not an appropriate method for gene-set analysis of common variants. Gene-sets need to be pre-compiled based on user-curated data or publicly available gene annotations like **Gene Ontology** or pathways.

"Competitive" gene-set over-representation tests are commonly used to analyze differentially gene expressed genes (e.g. Fisher's Exact Test, GSEA), but they are not suitable for rare CNVs [1][5]; the most appropriate choice for rare CNVs are a "self-contained" burden test with global burden correction implemented by cnvGSA and other tools [1].

Global burden correction is very important. For many disorders (including autism and schizophrenia), the disease-affected subjects (i.e. cases) are enriched in large, recurrent CNVs [2][3][4][6]. Those CNVs are not observed (or observed at very low frequency) in controls and only a minority of their genes contributes to disease risk. In the absence of global burden correction, many gene-sets would present a biologically unspecific burden, uniquely driven by those larger and recurrent CNVs [1]. Global burden correction thus helps identifying specific pathways and functional categories implicated in disease risk by rare CNVs.

In cnvGSA, subjects are treated as statistical sampling units. Subject-level covariates that may act as confounders can be provided by the user (e.g. sex, ethnicity, CNV genotyping platform, CNV genotyping site, array quality metrics, etc.). The gene-set burden is tested using a logistic regression approach. Two logistic regression models are fit: model A includes the subject-level covariates and a variable quantifying global CNV burden for each subject (total CNV length, or total number of CNV-overlapped genes per subject, etc.); model B includes all variables present in model A, plus the number of CNV-overlapped genes that are members of the gene-set being tested. Presence of significantly higher burden in cases compared to controls for the gene-set of interest is then tested by comparing the two models using a deviance chi-square test, as implemented by anova.glm.

In addition, cnvGSA provides functions to visualize gene-set burden results and also to export them to the Cytoscape plugin Enrichment Map [7].

2 Workflow outline

The typical analysis workflow consists of (1) loading CNV, phenotype/covariate and gene-set data, (2) setting filters and parameters, (3) running the analysis, (4) reviewing the results.

"CnvGSAInput" is a S4 class acting as a container data structure with slots for each of these required inputs:

```
> library("cnvGSA")
> library("cnvGSAdata")
> slotNames("CnvGSAInput")
[1] "cnvData.ls" "phData.ls" "gsData.ls" "config.ls" "params.ls"
```

The input slots should hold the following:

- cnvData.ls - CNV data
- phData.ls - Phenotype/Covariate data
- gsData.ls - Gene-set data
- config.ls - paths and file names
- params.ls - Test parameters

An example input object is available in the companion data package for this vignette:

```
> library("cnvGSA")
> library("cnvGSAdata")
> data("cnvGSA_input_example")
> ls()

[1] "cnvGSA.in"

> class(cnvGSA.in)

[1] "CnvGSAInput"
attr(,"package")
[1] "cnvGSA"

> slotNames(cnvGSA.in)
[1] "config.ls"  "params.ls"  "cnvData.ls" "phData.ls"  "gsData.ls"
```

Each of the slots can be accessed using an accessor function of the same name (e.g. cnvData.ls (cnvGSA.in) gets or sets cnvData.ls, gsData.ls(cnvGSA.in) gets or sets gsData.ls, etc.).

The input object is used by cnvGSA's main function, cnvGSAlogRegTest(), to perform the case-control gene-set burden test:

```
cnvGSAlogRegTest(cnvGSA.in)
```

This function produces an object of class "CnvGSAOutput" as its output – likewise a simple S4 class that has slots for each of the output elements.

```
> library(cnvGSA)
> library(cnvGSAdata)
> data("cnvGSA_output_example")
> ls()

[1] "cnvGSA.in" "cnvGSA.out"

> class(cnvGSA.out)

[1] "CnvGSAOutput"
attr(,"package")
[1] "cnvGSA"

> slotNames(cnvGSA.out)
[1] "res.ls"      "gsTables.ls" "gsData.ls"   "phData.ls"   "config.df"
```

The output slots contain the following:

- res.ls - List with the output from the association tests. One data frame for the test with the known loci and one without the known loci. [@JL: FIX]
- gsTables.ls - Burden analysis results for gene-sets.
- gsData.ls - Gene-set tables which look at each gene individually for each gene-set.
- phData.ls - Phenotype/covariate data which is similar to that in. cnvGSA.in

As with the slots in the input object, each of these can likewise be accessed using an accessor function of the same name (gsTables.ls() gets gsTables.ls, etc.).

Throughout the following sections, the pre-built cnvGSA.in example object will be used to illustrate each step of the typical analysis workflow.

3 Loading input data

The elements of a CnvGSAInput object – i.e. cnvData.ls, phData.ls, gsData.ls, config.ls, and params.ls – are easily filled when using the function cnvGSAIn as described above. Please ensure that the input data is formatted correctly so the script will run properly.

Creating the input data should look like the following pseudocode:

```
# Create input object
> cnvGSA.in <- CnvGSAInput()
> cnvGSA.in <- cnvGSAIn(configFile,cnvGSA.in)
```

Each of the object components can be accessed using the accessor functions.

```
# To access the params.ls run
> params.ls <- params.ls(cnvGSA.in)
```

[@JL: ADD: please provide another example]

3.1 CNV data

[@JL: CHANGE: please describe the requirements more explictly, e.g. state that the detailed format of the input is described below]

The input CNV data for `cnvGSAIn` relies on providing data with the correct structure and following the specifications described in this section. The following is an example on what the CNV file should look like. All text files that are input should be tab separated. Here are the columns that must be included for the tests to run:

- `$cnv.df` - A data frame containing the CNVs. Each row contains data for one CNV:
  - `SID` - ID assigned to the subject's DNA sample in which the CNV was found. It is assumed that the correspondence for each is always 1-to-1 with a subject.
  - `Chr` - Chromosome on which the CNV is located.
  - `BP1` - Start position of the CNV on the chromosome.
  - `BP2` - End position of the CNV on the chromosome.
  - `TYPE` - CNV type (`"1"` or `"3"` used in example, but can be any other label indicating deletions and gains).
  - `geneID` - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. `"54777;255352;84435"` for semicolon-delimited EntrezGene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. `""`) in this column.

```
> cnvFile <- system.file("extdata","cnv_AGP_demo.txt",package = "cnvGSAdata")
> cnv.df <- read.table (cnvFile, header = T, sep = "\t", quote = "\"",
+ stringsAsFactors = F)
> str(cnv.df,strict.width="cut")
'data.frame': 10666 obs. of  6 variables:
 $ SID   : chr  "1020_4" "1020_4" "1020_4" "1030_3" ...
 $ CHR   : chr  "4" "6" "3" "10" ...
 $ BP1   : int  34802932 35606076 4110452 56265896 64316996 24115481 83206919..
 $ BP2   : int  35676439 35673400 4145874 56361311 64593616 24202712 83239473..
 $ TYPE  : int  3 3 1 1 1 3 1 3 3 3 ...
 $ geneID: chr  NA "2289" NA NA ...
```

[@JL: FIX: the CNV type example in the object has values different than in the text (1, 3 vs LOSS, GAIN)]

[@JL: CHECK: is the part above just a copy and paste from Robert? `$s2class` does not exist any more, and SampleID has been replaced by SID]

After running `cnvGSAIn` the following will be included in the `$cnvData.ls` slot. There will be two columns that have to do with the known loci called `$OlpKL_CNV` and `$OlpKL_SID` which come from marking the known loci seen in section 3.3.1.

[@JL: CHANGE: you never mentioned the known loci, and now all of a sudden there's the columns that tag the overlap with known loci]

[@JL: FIX: you have the column with the gene symbols labeled as ”exon”]

[@JL: FIX: what is `geneID_DH` ?]

Its added elements are as follows:

- `$cnv.df` - A data frame containing the CNVs. Each row contains data for one CNV. It has added columns to the previous description and they are:
    - `CnvKey` - Made from combining CHR, BP1, BP2, TYPE with @ as separator
    - `$OlpKL_CNV` - Finds overlaps using the CNV data and the known loci data. It makes use of the package `GenomicRanges` to find the overlaps. It will only count those that are above the threshold named `$klOlp` which is described in more detail here 4.2.
    - `$OlpKL_SID` - Using the information from `$OlpKL_CNV`, this column will determine whether there is an overlap for each `SID`.
    - `$SubjCnvKey` - Made from combining `$SID` and `$Cnvkey` using @@
    - `CnvLength_ALL` - Length for all the CNVs for each `SID`. Adds all the lengths of each event that occurred for each `SID`
    - `$CnvLength_TYPE` - Length for the user specified CNV type. Only adds the lengths from events that are classified as the user specified CNV type.
    - `$CnvCount_TYPE` - Count for the total number of events of the user specified CNV type.

- `$cnv2gene.df` - A data frame that is used to map the CNVs to the corresponding genes.

```
# cnvData.ls after running cnvGSAIn
> str(cnvData.ls(cnvGSA.in),strict.width="cut")
List of 2
 $ cnv.df     :'data.frame': 10666 obs. of  15 variables:
  ..$ SID            : chr [1:10666] "1020_4" "1020_4" "1020_4" "1030_3" ...
  ..$ CHR            : chr [1:10666] "4" "6" "3" "10" ...
  ..$ BP1            : int [1:10666] 34802932 35606076 4110452 56265896 6431699..
  ..$ BP2            : int [1:10666] 35676439 35673400 4145874 56361311 64593616..
  ..$ TYPE           : int [1:10666] 3 3 1 1 1 3 1 3 3 3 ...
  ..$ geneID         : chr [1:10666] NA "2289" NA NA ...
  ..$ CnvKey         : chr [1:10666] "4@34802932@35676439@3" "6@35606076@35673400@3"..
  ..$ SEX            : chr [1:10666] "Male" "Male" "Male" "Male" ...
  ..$ Condition      : int [1:10666] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ OlpKL_CNV      : num [1:10666] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ OlpKL_SID      : num [1:10666] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ SubjCnvKey     : chr [1:10666] "1020_4@@4@34802932@35676439@3" "1020_4@@6@356..
  ..$ CnvLength_ALL : num [1:10666] 873508 67325 35423 95416 276621 ...
  ..$ CnvLength_TYPE: num [1:10666] 0 0 35423 95416 276621 ...
  ..$ CnvCount_TYPE : num [1:10666] 0 0 1 1 1 0 1 0 0 0 ...
$ cnv2gene.df:data.frame: 22322 obs. of  9 variables:
  ..$ SubjCnvKey : Factor w/ 10666 levels "1020_4@@3@4110452@4145874@1",..: 1 2 3 4 5 5
  ..$ geneID     : chr [1:22322] NA NA "2289" NA ...
  ..$ SEX        : chr [1:22322] "Male" "Male" "Male" "Male" ...
  ..$ CHR        : chr [1:22322] "3" "4" "6" "10" ...
  ..$ BP1        : int [1:22322] 4110452 34802932 35606076 56265896 64316996 ...
  ..$ BP2        : int [1:22322] 4145874 35676439 35673400 56361311 64593616 ...
  ..$ SID        : chr [1:22322] "1020_4" "1020_4" "1020_4" "1030_3" ...
  ..$ TYPE       : int [1:22322] 1331113333...
  ..$ geneID_TYPE: chr [1:22322] NA NA NA NA ...
```

[@JL: FIX: most of the descriptions above are incorrect]

3.1.1 Gene ID file

One convenient feature of this package is that the user can use any gene ID system. The user just has to provide a gene ID file that will specify which system they would like to use. This file is used to get the meta data on the genes like the gene symbols and the gene name.

- `$ann_eg.df` - A data frame containing information describing the gene ID system that the user would like to use:
  - ○ `geneID` - The gene identifier for the user specified system.

- ○ `Symbol` - Symbol for the gene.
- ○ `Name` - Name describing the gene `Symbol`.

[@JL: CHANGE: first state the general input file specification, then provide the example]

Here is an example of how to make the gene ID file with the Entrez ID system for Humans and how to save the data as a tab delimited text file:

```
> library (org.Hs.eg.db)

> ann_eg2sy.df   <- stack (as.list (org.Hs.egSYMBOL));
> names (ann_eg2sy.df)   <- c ("Symbol", "geneID");
> ann_eg2sy.df   <- as.data.frame (lapply (ann_eg2sy.df,   as.character),
+ stringsAsFactors = F)

> ann_eg2name.df <- stack (as.list (org.Hs.egGENENAME));
> names (ann_eg2name.df) <- c ("Name",   "geneID");
> ann_eg2name.df <- as.data.frame (lapply (ann_eg2name.df, as.character),
+ stringsAsFactors = F)

> ann_eg.df <- merge (ann_eg2sy.df, ann_eg2name.df, by = "geneID", all = T)

> str(ann_eg.df,strict.width="cut")
'data.frame': 47721 obs. of  3 variables:
 $ geneID: chr  "1" "10" "100" "1000" ...
 $ Symbol: chr  "A1BG" "NAT2" "ADA" "CDH2" ...
 $ Name  : chr  "alpha-1-B glycoprotein" "N-acetyltransf"..

# place the path to this file in the configuration file
> save ( ann_eg.df, file = "geneID_demo.txt" )
```

3.2 Phenotype/Covariate Data

After the CNV data has been loaded the next data structure that needs to be loaded is `phData`, which contains the phenotype/covariate data. Just like the CNV data there is an example of this type of file in the package `cnvGSAdata`.

[@JL: CHANGE: first state the general input file specification, then provide the example]

The data should look contain the following columns. There can be additional columns added to the data, like more covariates. This is the bare minimum format it should follow:

- `$ph.df` - A data frame containing the CNVs. Each row contains data for one CNV:

  - `SampleID` - ID assigned to the subject's DNA sample in which the CNV was found.
  - `Covariates` - The covariates that will be included in the analysis. A set of columns, which act as a character, numeric, integer or factor.
  - `Condition` - Identifies the sample as a case(1) or control(0).

```
> phFile <- system.file("extdata","ph_AGP_demo.txt",package = "cnvGSAdata")
> ph.df <- read.table (phFile, header = T, sep = "\t", quote = "\"",
+ stringsAsFactors = F)
> str(ph.df)
'data.frame': 10666 obs. of  3 variables:
 $ SID      : chr  "1020_4" "1020_4" "1020_4" "1030_3" ...
 $ SEX      : chr  "Male" "Male" "Male" "Male" ...
 $ Condition: int  1 1 1 1 1 1 1 1 1 1 ...
```

After the user runs `cnvGSAIn()` there will be some additional columns and objects added to the slot. Once again you will see that `$OlpKL_SID` is one of the columns added and is explained more in section 3.3.1.

- `$ph.df` - A data frame with the gene-sets as columns. Each of these columns will show how many total events occurred with for each gene-set and the corresponding `$SID`. There are additional covariates and columns added once `cnvGSAIn()` is run:
  - `OlpKL_SID` - Finding those that have overlap in the CNV data frame. Maps it to the corresponding `$SID`.
  - `CnvCount_TYPE` - Count for the number of events of the user specified type that has occurred for each `$SID`.
  - `CnvTotLength_TYPE` - Total length of all the CNV events of the user specified type that occurred for each `$SID`.
  - `CnvMeanLength_TYPE` - Mean length of all the CNV events of the user specified type that occurred for each `$SID`.
- `$ph_TYPE.df` - A data frame with the gene-sets as additional columns. Each of these columns will show how many total events occurred with for each gene-set and the corresponding `$SID`

```
> str(cnvGSA.in@phData.ls,max.level=2,list.len=10)
List of 2
 $ ph.df:'data.frame': 4227 obs. of  7 variables:
  ..$ SID               : chr [1:4227] "1020_4" "1030_3" "1045_3" "1050_3" ...
  ..$ SEX               : chr [1:4227] "Male" "Male" "Male" "Female" ...
  ..$ Condition         : int [1:4227] 1 1 1 1 0 0 0 0 0 0 ...
  ..$ OlpKL_SID         : num [1:4227] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ CnvCount_TYPE     : num [1:4227] 1 2 1 1 0 0 2 1 3 0 ...
  ..$ CnvTotLength_TYPE : num [1:4227] 35423 372037 32555 41098 0 ...
  ..$ CnvMeanLength_TYPE: num [1:4227] 11808 186018 10852 6850 0 ...
 $ ph_TYPE.df:'data.frame': 4227 obs. of  59 variables:
  ..$ SID               : chr [1:4227] "1020_4" "1030_3" "1045_3" "1050_3" ...
  ..$ SEX               : chr [1:4227] "Male" "Male" "Male" "Female" ...
  ..$ Condition         : int [1:4227] 1 1 1 1 0 0 0 0 0 0 ...
  ..$ OlpKL_SID         : num [1:4227] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ CnvCount_TYPE     : num [1:4227] 1 2 1 1 0 0 2 1 3 0 ...
  ..$ CnvTotLength_TYPE : num [1:4227] 35423 372037 32555 41098 0 ...
  ..$ CnvMeanLength_TYPE: num [1:4227] 11808 186018 10852 6850 0 ...
  ..$ GS1               : int [1:4227] 0 1 0 1 0 0 1 0 1 0 ...
  ..$ GS10              : int [1:4227] 0 1 0 0 0 0 1 0 1 0 ...
  ..$ GS11              : int [1:4227] 0 0 0 1 0 0 0 0 2 0 ...
  .. [list output truncated]
```

[@JL: CHANGE: `$ph_TYPE.df` should not be a subset]

3.3 Gene sets

The gene-set data should be one list containing all the gene sets that the user wants to include in the analysis. This list should have gene-set names which will act as the `$GsID`'s for the later tables and will be the ID's for a certain gene set.

The object containing the gene-sets **must** be called `gs_all.ls` and will be a list with each entry being the gene-set named and a vector or the corresponding genes in the gene-set. There should also be a separate object that will contain the gene set names. This object **must** be named `gsid2name.chv` and will be a vector simply containing all the gene-set names. Here is an example of what they should look like:

[@JL: FIX: why do you state ”file” ?]

```
> library(cnvGSAdata)
> data("gs_data_example") # gs_all.ls and gsid2name.chv will be loaded
> str(gs_all.ls,max.level=1,list.len=5)
List of 51
 $ BspanVHM_PreNat            : int [1:3038] 7105 4800 81887 6405 1595 3927 ...
 $ BspanVHM_EqlNat            : int [1:3038] 8813 54467 9957 23072 8379 56603 ...
 $ BspanVHM_PstNat            : int [1:3131] 2519 2729 90529 57185 22875 572 ...
 $ BspanVH_lg2rpkm4.74        : int [1:4600] 7105 4800 57185 1595 572 9957 3927 ...
 $ BspanHM_lg2rpkm3.21        : int [1:4605] 8813 2519 2729 90529 81887 22875 ...
  [list output truncated]
```

```
> str(gsid2name.chv)
 Named chr [1:51] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" ...
 - attr(*, "names")= chr [1:51] "BspanVHM_PreNat" "BspanVHM_EqlNat" ...
```

After the gene-set data is formatted correctly, `gsData.ls`, which is the slot that contains the gene-set data can be loaded. As stated before, the `gsData.ls()` function below is just an accessor function for this slot. There is an example of this in the `CnvGSAInput()` object. Too see an example, just load the `CnvGSAInput()` object included in this package.
The list elements are:

- `$gs_info.df` - A data frame with the columns `$GsKey`, `$GsID` ,`$GsName`, `$GsSize` that is used to map the CNVs to the appropriate gene-set.
- `$gs_sel_U.df` - A data frame with the columns `$GsKey`, `$gID`, `$GsName` that is used to map each `$GsKey` with the corresponding `$geneID`. It splits all the genes apart so that there is a one to one relationship.
- `$gs_colnames_TYPE.chv` - A single character vector that contains all the `$GsKey` elements.
- `$gs.ls` - A list containing all the gene sets and may be filtered depending on what the user specifies. The names for this list are the `$GsKeys`.
- `$geneCount.tab` - A table that is used to obtain the geneCount.df data frame. It uses the table function in R to compare the `$geneID` with the `$Condition`.
- `$geneCount.df` - A data frame with the columns `$GsID`, `$Name`, `$Controls`, `$Cases`. This shows how many cases and controls are present for each gene.
- `$gs_all.ls` - A list containing all the gene-sets and their corresponding genes. The names for this list are the `$GsIDs`.

```
> library(cnvGSAdata)
> data("cnvGSA_input_example")
> str(cnvGSA.in@gsData.ls,strict.width="cut",list.len = 10)
List of 7
 $ gs_info.df          :'data.frame': 52 obs. of  4 variables:
  ..$ GsKey : chr [1:52] "GS1" "GS2" "GS3" "GS4" ...
  ..$ GsID  : chr [1:52] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" ..
  ..$ GsName: chr [1:52] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" ..
  ..$ GsSize: int [1:52] 3037 3037 3131 4600 4605 4596 4601 4131 4185 4190 ...
 $ gs_sel_U.df         :'data.frame': 120486 obs. of  4 variables:
  ..$ GsKey : Factor w/ 52 levels "GS1","GS10","GS11",..: 1 1 1 1 1 1 1 1 1 1 ...
  ..$ gID   : chr [1:120486] "7105" "4800" "81887" "6405" ...
  ..$ GsID  : chr [1:120486] "BspanVHM_PreNat" "BspanVHM_PreNat" "BspanVHM_PreNat" ..
  ..$ GsName: chr [1:120486] "BspanVHM_PreNat" "BspanVHM_PreNat" "BspanVHM_PreNat" ..
 $ gs_colnames_TYPE.chv: chr [1:51] "GS1" "GS10" "GS11" "GS12" ...
 $ gs.ls               :List of 52
  ..$ GS1 : int [1:3037] 7105 4800 81887 6405 1595 3927 90293 57679 7035 10181 ...
  ..$ GS2 : int [1:3037] 8813 54467 9957 23072 8379 56603 79007 8837 9108 381 ...
  ..$ GS3 : int [1:3131] 2519 2729 90529 57185 22875 572 51056 29916 55365 4074 ...
  ..$ GS4 : int [1:4600] 7105 4800 57185 1595 572 9957 3927 90293 56603 79007 ...
  ..$ GS5 : int [1:4605] 8813 2519 2729 90529 81887 22875 6405 54467 51056 23072 ...
  ..$ GS6 : int [1:4596] 57147 55732 2268 3075 889 5893 4267 843 55471 952 ...
  ..$ GS7 : int [1:4601] 64102 1080 26 51384 6542 85413 340273 799 3207 4222 ...
  ..$ GS8 : chr [1:4131] "1" "8086" "65985" "343066" ...
  ..$ GS9 : chr [1:4185] "29974" "53947" "13" "344752" ...
  ..$ GS10: chr [1:4190] "2" "195827" "79719" "14" ...
  .. [list output truncated]
 $ geneCount.df        :'data.frame': 6153 obs. of  4 variables:
  ..$ geneID  : chr [1:6153] "1" "10" "10003" "100033413" ...
  ..$ Name    : chr [1:6153] "alpha-1-B glycoprotein" "N-acetyltransferase 2"..
  ..$ Controls: int [1:6153] 1 1 0 0 0 0 0 0 0 0 ...
  ..$ Cases   : int [1:6153] 1 0 1 6 6 6 6 6 6 6 ...
 $ geneCount.tab       : 'table' int [1:6203, 1:2] 1 1 0 0 0 0 0 0 0 0 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ geneID   : chr [1:6203] "1" "10" "10003" "100033413" ...
  .. ..$ Condition: chr [1:2] "0" "1"
 $ gs_all.ls           :List of 51
  ..$ BspanVHM_PreNat            : int [1:3038] 7105 4800 81887 6405 1595 3927 ...
  ..$ BspanVHM_EqlNat            : int [1:3038] 8813 54467 9957 23072 8379 56603 ...
  ..$ BspanVHM_PstNat            : int [1:3131] 2519 2729 90529 57185 22875 ...
  ..$ BspanVH_lg2rpkm4.74        : int [1:4600] 7105 4800 57185 1595 572 9957 ...
  ..$ BspanHM_lg2rpkm3.21        : int [1:4605] 8813 2519 2729 90529 81887 ...
  ..$ BspanML_lg2rpkm0.93        : int [1:4596] 57147 55732 2268 3075 889 5893 ...
  ..$ BspanLA_lg2rpkm.MIN        : int [1:4601] 64102 1080 26 51384 6542 ...
  ..$ GI_q4                      : chr [1:4131] "1" "8086" "65985" "343066" ...
  ..$ GI_q3                      : chr [1:4185] "29974" "53947" "13" "344752" ...
  ..$ GI_q2                      : chr [1:4190] "2" "195827" "79719" "14" ...
  .. [list output truncated]
```

3.3.1 Marking Known Loci

The option to include known loci into the analysis is available and should resemble the format discussed below. These files to create the objects are defined in the configuration file. There are two objects needed to mark the known loci. By including known loci, overlaps can be calculated for the gene-sets and the CNVs. All this is done in the `cnvGSAIn` function. This will help uncover more information in the analysis. The objects needed are:

kl_loci.df - Data frame that contains all the known locus for the. It should include the following columns:

- CHR - Chromosome on which the CNV is located.
- BP1 - Start position of the CNV on the chromosome.
- BP2 - End position of the CNV on the chromosome.
- TYPE - CNV type (typically "LOSS" or "GAIN", but can be any other label indicating deletions and gains).
- geneID - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited EntrezGene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column.

```
> kl_lociFile <- system.file("extdata","kl_loci_AGP_demo.txt",package = "cnvGSAdata")
> kl_loci.df <- read.table (kl_lociFile,  sep = "", header = T, comment.char = "",
+ quote = "\"", stringsAsFactors = F)
> str(kl_loci.df,strict.width="cut")
'data.frame': 56 obs. of  5 variables:
 $ CHR   : chr  "2" "2" "15" "9" ...
 $ BP1   : int  50539877 51002576 21190624 98998 50990306 50968208 28705540..
 $ BP2   : int  50730546 51157742 26203954 3682923 51222043 51214171 30436163..
 $ TYPE  : int  1 1 3 1 1 1 1 3 1 3 ...
 $ geneID: chr  "9378" "9378" "100033413;100033414;100033415;100033416;..
```

kl_gene.df - Data frame that contains the genes of interest for the analysis.

- geneID - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited EntrezGene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column..
- Symbol - Gene Symbol.
- TYPE - CNV type (typically "LOSS" or "GAIN", but can be any other label indicating deletions and gains).

```
> kl_geneFile <- system.file("extdata","kl_gene_AGP_demo.txt",package = "cnvGSAdata")
> kl_gene.df <- read.table (kl_geneFile, sep = "", header = T, comment.char = "",
+ quote = "\"", stringsAsFactors = F)
```

```
> str(kl_gene.df,strict.width="cut")
'data.frame':	1 obs. of  3 variables:
 $ geneID: int 9378
 $ Symbol: chr "NRXN1"
 $ TYPE  : int 1
```

4 Configuring test parameters

Running the association tests requires that the user provides a configuration file to control how the tests will be run. The script will go through the configuration file that the user will input and use these set up for different functions in the package. There are four blocks in the configuration file that serve different purposes. The following section will go through each block in and explain how to fill in each value.

4.1 config.ls configuration

The `config.ls()` accessor function can be used to interact with this list. The main association test procedure accepts several files and this list will contain all the paths to the necessary input data (note that the `config.ls()` function below is just an accessor function for this slot).

The fields for `config.ls` are as follows:

- `cnvFile` - File name for the CNV file user would like to read in.
- `phfile` - File name for the phenotype/covariate file user would like to read in.
- `geneIDFile` - File name for the file describing the ID system used for the data.
- `klGeneFile` - File name for the file with the genes of interest.
- `klLocifile` - File name for the file with containing the loci of the genes.
- `gsFile` - File name for the gene-set data the user would like to read in.
- `outputPath` - Path for the folder where the user would like to output everything.
- `geneListFile` - Path for the folder containing the list of genes that the user would like to include in the regression test. This is optional and will only look a the genes in the list. Use this to look at specific genes of interest.

```
> str( config.ls(cnvGSA.in) )
List of 9
 $ cnvFile     : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/cnv_AGP_demo.tx
 $ phFile      : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/ph_AGP_demo.txt
 $ geneIDFile  : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/geneID_demo.txt
 $ klGeneFile  : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/kl_gene_AGP_dem
 $ klLociFile  : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/kl_loci_AGP_dem
 $ gsFile      : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/gs_data_demo.RD
 $ outputPath  : chr "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/te
 $ geneListFile: chr ""
 $ config.df   :'data.frame': 37 obs. of  2 variables:
  ..$ param: chr [1:37] "cnvFile" "phFile" "geneIDFile" "klGeneFile" ...
  ..$ value: chr [1:37] "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/cnv_AGP_demo
```

4.2 params.ls configuration

The params.ls() accessor function can be used to interact with this list.The main association test procedure accepts several parameters which specify how the user would like to run the test.

The fields for `params.ls` are as follows:

- `Kl` - Whether or not the user would like to include known loci in the analyses. If the user chooses `"ALL"`, the tests will be run twice; once with the known loci, and once without the known loci. The default value for this parameter is `"ALL"`. Possible values are:

  - `"YES"` - Include known loci in analysis
  - `"NO"` - Do not include known loci in analysis
  - `"ALL"` - Produce results from both ”YES” and ”NO”. Will create two separate data frames.

- `projectName` - This is where the user defines the type of gene-set they are using (ex. for Neurosets the user can put ”NS”). This helps organize the output file names.

- `gsUset` - Gene-set representing the universe-set for the corresponding global burden correction. *This is optional.*

- `cnvType` - The type of CNV that the user would like to use for testing. Possible values are:

  - Any value in the `$TYPE` column which may include (”1”, ”0”, ”LOSS”, ”GAIN”, etc.). The user may have multiple levels for this column but they should only choose one for the analysis.
  - `"ALL"` - Includes all types of CNVs in the data.

- `covariates` - The covariates the user would like to include in the regression model. This will be a list of all the desired covariates without any spaces in between (ex. ”SEX,CNV_platform”). The user can choose not to include covariates by puting ”NONE”.

- `klOlp` - The threshold of the percentage of overlap having to do with the known loci. The default value for this is 0.5.

- `corrections` - The correction models that they would like to include in the test. Each correction model will provide different results and comparisons can be made to see the difference between all the different corrections. More than one correction can be included in the analysis. Enter as a list of corrections without any space in between (ex. ”no_corr,uni_gc”). The default value for this is to use all the correction models. Possible values are:

  - `"no_corr"` - Do not include any correction in the analysis.
  - `"uni_gc"` - Add the universe count correction model.
  - `"tot_l"` - Add the total count correction model.
  - `"cnvn_ml"` - Add the mean length and number correction model.
  - `"ALL"` - Include all the correction models.

- `geneSep` - The separator that will be used to separate the gene list in the `$geneID` column of the `cnv.df` object. The default value for this is ”;”.

- `keySep` - The separator that will be used to separate the values in the `$SubjCnvKey` column of the `cnv.df` object. The default value for this is ”@”.

- `geneSetSizeMax` - The max number of genes that the gene sets are allowed to have. The default value for this is 1500.

- `geneSetSizeMin` - The minimum number of genes that the gene sets are allowed to have. The default value for this is 25.

- `filtGs` - Once the `$geneSetSizeMax` and `$geneSetSizeMax` parameter’s have been set the user can choose whether or not to include this filter in the analysis. The default value for this is ”NO” Possible values are:

  - `"YES"` - Include the filter in the analysis.
  - `"NO"` - Do not include the filter in the analysis.

- `covInterest` - What covariate will be used to report stratified case and control subject counts.

- `eventThreshold` - The threshold for the number of events that the user chooses to be accepted. The default value for this is 1.

- `flevels` - The max amount of levels the factor `$covInterest` is allowed to have. The default value for this is 10.

- `cores` - User will set the amount of cores they want to use for the parallelization. The default value for this is the max amount of cores on the machine.

- parallel - Option to run the association test in parallel or not. The default value is "YES". The options include:
  - "YES" - Allow parallelization in tests.
  - "NO" - Do not allow parallelization in tests.

```
> str( params.ls(cnvGSA.in) )
List of 18
 $ Kl            : chr "ALL"
 $ projectName   : chr "NS"
 $ gsUSet        : chr ""
 $ cnvType       : chr "1"
 $ covariates    : chr "SEX"
 $ klOlp         : num 0.5
 $ corrections   : chr [1:4] "no_corr" "uni_gc" "tot_l" "cnvn_ml"
 $ geneSep       : chr ";"
 $ keySep        : chr "@"
 $ geneSetSizeMin: num 25
 $ geneSetSizeMax: num 1500
 $ filtGs        : chr "NO"
 $ covInterest   : chr "SEX"
 $ eventThreshold: num 1
 $ fLevels       : num 10
 $ check_type    : chr "1"
 $ cores         : num 4
 $ parallel      : chr "YES"
```

[@JL: CHECK: do we really need "projectName" ?]

[@JL: CHANGE: the "thresholdSzCt" parameter should be labeled differently; the definition of "gsUset" is: geen-set representing the universe-set for the corresponding global burden correction; the definition of "covInterest" is: what covariate will be used to report stratified case and control subject counts]

4.3 Visualization configuration

This package includes the ability to create visualizations from the output of the main association test. The function `f.makeViz` is described in the workflow example and it takes in these fields:

- FDRThreshold - Will only include those that are under this number for the plots. The default value is 0.1.

- plotHeight - Vertical length of the plots. The default value is 13.

- `gsList` - Text file containing list of genes that the user is interested in looking at plotting. Should be comma separated with no space in between (ex. "PSD_BayesGrant_fullset, FMR1_Targets_Darnell").

- `outputPathViz` - Path for the output of the visualization script.

- `labelSize` - How long the user would like the labels to be on the plots. The default value is 0.7.

- `correctionViz` - Corrections used in creating the plots. Multiple corrections may be selected but since there is filtering being done with the FDR value `no_corr` cannot be chosen. The default value for this is "ALL". The valid options include:

  ◦ `"uni_gc"` - Add the universe count correction model.
  ◦ `"tot_l"` - Add the total count correction model.
  ◦ `"cnvn_ml"` - Add the mean length and number correction model.
  ◦ `"ALL"` - Include all the correction models.

- `cnvGSAViz` - also takes in some fields that are defined in `params.ls`. It is not necessary to restate the following fields:

  ◦ `Kl` - look at description in previous section.
  ◦ `cnvType` - look at description in previous section.

4.4 Enrichment configuration

Along with the visualization function, there is also a function that will create the files necessary to create an Enrichment Map. This function takes in these fields:

- `pVal` - the p-value the user would like to use for the enrichment map. The default value is "Pvalue_U_dev".

- `FDR` - the FDR value the user would like to use for the enrichment map. The default value is "FDR_BH_U".

- `coeff` - the coefficient the user would like to use for the enrichment map. The default value is "Coeff_U".

- `keepCoeff` - whether or not the user would like to keep the negative coefficients. The default value is "YES".

- `outputPathEnr` - Path for the output of the enr script.

- `minCaseCount` - the min number of cases each gene is allowed to have. The default value is 0.

- `minControlCount` - the min number of controls each gene is allowed to have. The default value is 0.
- `minRatio` - the min case/control ratio each gene is allowed to have. The default value is 0.
- `filtGsEnr` - whether or not the user would like to filter the gene-sets alowed in the GMT file. The default value is ”YES”.
- `f.enrFiles` also takes in some fields that are defined in `params.ls`. It is not necessary to restate the following fields:
  - `Kl` - look at description in previous section.
  - `cnvType` - look at description in previous section.

Note: larger gene-sets have more statistical power; this can be taken into account by separately testing gene-sets with different sizes.

4.5 Loading test parameters from a file

To make it easier to integrate the association test into a larger bioinformatics pipeline, it is convenient to read in the parameters from an external source such as a text file. One such implementation is to record each parameter on its own line using the tab separated syntax:

[@JL: CHANGE: it is better to provide first the definition of the content of the parameters, and then the example]

```
param value
################
# LogReg Config
################
## CONFIG.LS ##
cnvFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/cnv_AGP_demo.txt"
phFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/ph_AGP_demo.txt"
geneIDFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/geneID_demo.txt"
klGeneFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/kl_gene_AGP_demo.txt"
klLociFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/kl_loci_AGP_demo.txt"
gsFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/gs_data_demo.RData"
outputPath "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/testResults/
geneListFile ""
## PARAMS.LS ##
Kl "ALL"
projectName "NS"
gsUSet ""
cnvType 1
```

```
covariates "SEX"
klOlp 0.5
corrections "no_corr,uni_gc,tot_l,cnvn_ml"
geneSep ";"
keySep "@"
geneSetSizeMax 1500
geneSetSizeMin 25
filtGs "NO"
covInterest "SEX"
eventThreshold 1
fLevels 10
cores   4
parallel    "YES"
############
# Viz Config
############
FDRThreshold 0.1
plotHeight 13
gsList "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/gsList.txt"
outputPathViz "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/Viz/"
labelSize 0.7
correctionViz   "uni_gc,tot_l,cnvn_ml"
############
# ENR Config
############
pVal "Pvalue_U_dev"
FDR "FDR_BH_U"
coeff "Coeff_U"
keepCoeff "YES"
filtGsEnr "NO"
outputPathEnr "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/enr/"
minCaseCount 0
minControlCount 0
minRatio 0
```

To update the parameters in the S4 object the user simply needs to change the config file and then run the function below. Once this function is run, all the changed parameters will be updated in the CnvGSAInput S4 object.

```
> cnvGSA.in <- f.readConfig(configFile,cnvGSA.in)
```

5 Full workflow example: case-control analysis of rare CNVs from the Pinto et al. 2014 ASD study

5.1 Loading the data and running the association test

The following code performs an analysis of approx. 5500 CNVs from 2000 subjects against approx. The CNV data-set consists of rare CNVs as described in Pinto et al., AJHG 2014 [4], and the gene-sets are a collection imported from the Gene Ontology, KEGG, Biocarta, Reactome, and PFAM.

[@JL: FIX: as your label states above, this is the AJHG 2014, not the Nature 2010 dataset]

```
> library( "cnvGSA" )
> library( "cnvGSAdata" )

##
## Update the config file
##

## This will give the user all the paths needed to update the config file
> cnvFile    <- system.file( "extdata", "cnv_AGP_demo.txt", package="cnvGSAdata" )
> phFile     <- system.file( "extdata", "ph_AGP_demo.txt", package="cnvGSAdata" )
> geneIDFile <- system.file( "extdata", "gene_ID_demo.txt", package="cnvGSAdata" )
> klGeneFile <- system.file( "extdata", "kl_gene_AGP_demo.txt", package="cnvGSAdata" )
> klLociFile <- system.file( "extdata", "kl_loci_AGP_demo.txt", package="cnvGSAdata" )
> gsFile     <- system.file( "extdata", "gs_data_demo.txt", package="cnvGSAdata" )
## make sure everything is tab separated in the config file
## look at the params above to see how to fill everything in

##
## Create the input object
##

> configFile <- "INPUT PATH TO CONFIG FILE"
> cnvGSA.in  <- CnvGSAInput() # need S4 object
> cnvGSA.in  <- cnvGSAIn(configFile,cnvGSA.in)

## If this error is seen:
> Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :
  line 2 did not have 2 elements
## Check the config file to make sure that everything but the comments are tab
## separated.
```

```
##
## Run association test and save the output
##

> cnvGSA.out <- CnvGSAOutput() # need S4 object
> cnvGSA.out <- cnvGSAlogRegTest(cnvGSA.in, cnvGSA.out)
```

5.2 Reviewing the results

As stated in the workflow outline, the output object is a simple S4 class containing a slot for each output data structure:

```
> slotNames(cnvGSA.out)
[1] "res.ls"      "gsTables.ls" "gsData.ls"   "phData.ls"
```

Similar to the input, each of these is a list structure containing further data structures: `res.ls` contains the the results of the logistic regression tests, and `gsTables.ls` contains the gene-set tables for all the gene-sets. `gsData.ls` contains all the gene-set data, and `phData.ls` contains all the pheontype/covariate data. Just as with the input object, each of these can be accessed using an accessor function of the same name.

5.2.1 res.ls results

Taking a look first at `res.ls`:

```
List of 2
 $ covAll_chipAll_TYPE_KLy.df :'data.frame': 50 obs. of  39 variables:
 $ covAll_chipAll_TYPE_KLn.df:'data.frame': 50 obs. of  39 variables:
```

The data frames `$covAll_chipAll_TYPE_KLy.df` and `$covAll_chipAll_TYPE_KLn.df` contain the actual logistic regression results. `$covAll_chipAll_TYPE_KLy.df` has the results for the tests where the known loci were included, whereas `$covAll_chipAll_TYPE_KLn.df` has the results where the known loci are not included. The TYPE is specified by the user and will vary. The types used in the sample data are "1"(LOSS) and "3"(GAIN).

The structure of both data frames is shown in the listing below:

```
> str( cnvGSA.out@res.ls$covAll_chipAll_TYPE_KLy.df )
data.frame': 50 obs. of  38 variables:
 $ GsID             : chr  "FMR1_Targets_Darnell" "PSD_BayesGrant_fullset" ...
 $ GsName           : chr  "FMR1_Targets_Darnell" "PSD_BayesGrant_fullset" ...
 $ GsSize           : int  840 1407 4600 1230 3131 4605 927 1424 3037 116 ...
 $ Coeff            : num  1.461 1.03 0.41 0.821 0.392 ...
 $ Pvalue_glm       : num  9.19e-09 2.15e-08 5.92e-07 2.09e-07 8.70e-07 ...
 $ Pvalue_dev       : num  8.67e-13 2.08e-12 4.45e-10 9.15e-09 1.96e-09 ...
 $ Pvalue_dev_s     : num  12.06 11.68 9.35 8.04 8.71 ...
 $ Coeff_U          : num  1.328 0.935 0.366 0.647 0.337 ...
 $ Pvalue_U_glm     : num  9.33e-07 2.81e-06 2.93e-04 2.88e-04 6.15e-04 ...
 $ Pvalue_U_dev     : num  4.19e-08 1.73e-07 7.35e-05 1.71e-04 2.82e-04 ...
 $ Pvalue_U_dev_s   : num  7.38 6.76 4.13 3.77 3.55 ...
 $ FDR_BH_U         : num  2.09e-06 4.33e-06 9.19e-04 1.71e-03 1.91e-03 ...
 $ Coeff_TL         : num  1.349 0.944 0.358 0.71 0.344 ...
 $ Pvalue_TL_glm    : num  2.33e-07 7.15e-07 4.46e-05 2.49e-05 3.67e-05 ...
 $ Pvalue_TL_dev    : num  4.63e-10 2.00e-09 5.31e-07 6.53e-06 1.86e-06 ...
 $ Pvalue_TL_dev_s  : num  9.33 8.7 6.27 5.19 5.73 ...
 $ FDR_BH_TL        : num  2.31e-08 4.99e-08 8.85e-06 5.44e-05 2.32e-05 ...
 $ Coeff_CNML       : num  1.375 0.967 0.366 0.753 0.353 ...
 $ Pvalue_CNML_glm  : num  9.72e-08 2.55e-07 1.66e-05 5.06e-06 1.20e-05 ...
 $ Pvalue_CNML_dev  : num  8.52e-11 2.51e-10 1.02e-07 8.20e-07 3.03e-07 ...
 $ Pvalue_CNML_dev_s: num  10.07 9.6 6.99 6.09 6.52 ...
 $ FDR_BH_CNML      : num  4.26e-09 6.27e-09 1.70e-06 6.83e-06 3.79e-06 ...
 $ CASE_g1n         : num  3.86 5.34 9.25 4.92 7.24 ...
 $ CTRL_g1n         : num  0.899 1.627 4.88 2.269 4.409 ...
 $ CASE_g2n         : num  0.74 1.16 2.7 1.22 2.7 ...
 $ CTRL_g2n         : num  0.0428 0.1284 0.8134 0.0856 0.8134 ...
 $ CASE_g3n         : num  0.529 0.687 1.639 0.687 1.745 ...
 $ CTRL_g3n         : num  0 0 0.342 0 0.171 ...
 $ CASE_g4n         : num  0.37 0.582 1.163 0.106 1.322 ...
 $ CTRL_g4n         : num  0 0 0.1712 0 0.0856 ...
 $ CASE_g5n         : num  0.106 0.423 0.793 0.106 1.005 ...
 $ CTRL_g5n         : num  0 0 0.1284 0 0.0856 ...
 $ CASE_gTT         : num  1891 1891 1891 1891 1891 ...
 $ CTRL_gTT         : num  2336 2336 2336 2336 2336 ...
 $ CASE_Female      : num  6.3 6.3 10.37 5.19 7.04 ...
 $ CASE_Male        : num  3.45 5.18 9.07 4.87 7.28 ...
 $ CTRL_Female      : num  0.962 1.603 4.567 2.484 5.048 ...
 $ CTRL_Male        : num  0.827 1.654 5.239 2.022 3.676 ...
```

[@JL: CHANGE: the ordering in this table is not always helpful for the user: it would be better to put the GsID, Name, Size first, and grouping the BH_FDRs with the corresponding p-values]

Each row contains results for a single gene-set. The columns are as follows:

- `GsID`, `GsName`, and `GsSize` show the gene-set's identifier, name, and number of member genes respectively.
- `Coeff`, `Coeff_U`, `Coeff_TL`, and `Coeff_CNML` show the gene-set's coefficients for each of the different correction models that were used. This represents the log odds of being a

case in the exposed vs. unexposed, adjusted for the covariates and whichever correction models were specified.

- Pvalue_glm, Pvalue_U_glm, Pvalue_TL_glm, and Pvalue_CNML_glmL show the gene-set's p-values for each of the different correction models that were used. These p-values were obtained using the z-test or more specifically Wald's test.
- Pvalue_dev, Pvalue_U_dev, Pvalue_TL_dev, and Pvalue_CNML_dev show the gene-set's p-values for each of the different correction models that were used.These p-values were obtained using a chi-square deviance test.
- Pvalue_dev_s, Pvalue_U_dev_s, Pvalue_TL_dev_s, and Pvalue_CNML_dev_s show the gene-set's p-values for each of the different correction models that were used. These p-values were obtained by taking the log of the deviance p-values and multiplying it by the sign of the coefficient.
- FDR_BH_U, FDR_BH_TL, and FDR_BH_CNML show FDR values using three methods. FDR_BH_U is the Benjamini-Hochberg FDR for the p-values calculated using the uni_gc correction for each gene-set calculated in relation to all the gene-sets in the input. FDR_BH_TL is the same as the previous, except with the tot_l correction. FDR_BH_CNML is the same as the previous, except with the cnvn_ml correction.
- CASE_g1n, CASE_g2n, CASE_g3n, CASE_g4n, and CASE_g5n show the percentage of samples from the cases that each gene-set has at least 1,2,3,4 and 5 events in.
- CTRL_g1n, CTRL_g2n, CTRL_g3n, CTRL_g4n, and CTRL_g5n show the percentage of samples from the controls that each gene-set has at least 1,2,3,4 and 5 events in.
- CASE_gTT and CTRL_gTT show the total number of cases and controls there are.
- CASE_Female and CASE_male show the percentage of samples from the cases that must be >= a user specified number of events and also is part of the SEX covariate. This will change depending on which covariate the user would like to see in the output.
- CTRL_Female and CTRL_male show the percentage of samples from the controls that must be >= a user specified number of events and also is part of the SEX covariate.This will change depending on which covariate the user would like to see in the output.

5.2.2 Detailed analysis of gene-set associations

As a further aid in understanding the CNVs and genes contributing to the association results, there is gsTables.ls – a list of data frames, one for each of the gene-sets, containing information about the CNVs and corresponding genes affecting the gene-set:

```
> str( gsTables.ls, max.level=1, list.len=5 )
List of 52
 $ BspanHM_lg2rpkm3.21        :’data.frame’: 707 obs. of  10 variables:
  .. [list output truncated]
 $ BspanLA_lg2rpkm.MIN        :’data.frame’: 1883 obs. of  10 variables:
  .. [list output truncated]
 $ BspanML_lg2rpkm0.93        :’data.frame’: 1026 obs. of  10 variables:
  .. [list output truncated]
 $ BspanVH_lg2rpkm4.74        :'data.frame': 546 obs. of  10 variables:
  .. [list output truncated]
 $ BspanVHM_EqlNat            :'data.frame': 418 obs. of  10 variables:
  .. [list output truncated]
  [list output truncated]
```

To get gsTables.ls the user need to run the following:

```
> gsTables.ls <- cnvGSA.out@gsTables.ls
> gsTables.ls <- cnvGSAgsTables(cnvGSA.in,cnvGSA.out)
```

The structure of the data frame for a particular gene-set is similar to that of cnvData$cnv in the input:

```
> gsTables.ls[[2]][10:19,]
    geneID_TYPE                                SubjCnvKey    SEX CHR       BP1
227   100310812 HABC_900437_900437@@7@101900904@102126404@1 Female   7 101900904
229   100310812 HABC_900216_900216@@7@101900914@102126404@1 Female   7 101900914
231   100310812 HABC_900320_900320@@7@101901092@102121342@1 Female   7 101901092
233   100310812 HABC_900736_900736@@7@101900904@102126404@1 Female   7 101900904
235   100310812 HABC_902946_902946@@7@101900904@102139865@1   Male   7 101900904
237   100310812 HABC_901157_901157@@7@101900846@102146951@1 Female   7 101900846
239   100310812 HABC_900766_900766@@7@101900904@102126404@1 Female   7 101900904
241   100310812 HABC_902681_902681@@7@101900846@102126404@1 Female   7 101900846
243   100310812       20011_1048001@@7@101900846@102139865@1   Male   7 101900846
245   100310812    16086_1571084001@@7@101900914@102126404@1   Male   7 101900914
          BP2                SID TYPE              GsID              GsName
227 102126404 HABC_900437_900437    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
229 102126404 HABC_900216_900216    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
231 102121342 HABC_900320_900320    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
233 102126404 HABC_900736_900736    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
235 102139865 HABC_902946_902946    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
237 102146951 HABC_901157_901157    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
239 102126404 HABC_900766_900766    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
241 102126404 HABC_902681_902681    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
243 102139865      20011_1048001    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
245 102126404   16086_1571084001    1 BspanLA_lg2rpkm.MIN BspanLA_lg2rpkm.MIN
```

(Note that the selection above shows only 10 rows from a single data frame in the list.).

Examining the data frame for a particular gene-set may reveal that its association due to certain genes may actually be better explained by *other* genes (those that have a clearer functional impact or that have previously been associated with the cases under consideration).

5.2.3 Visualizations

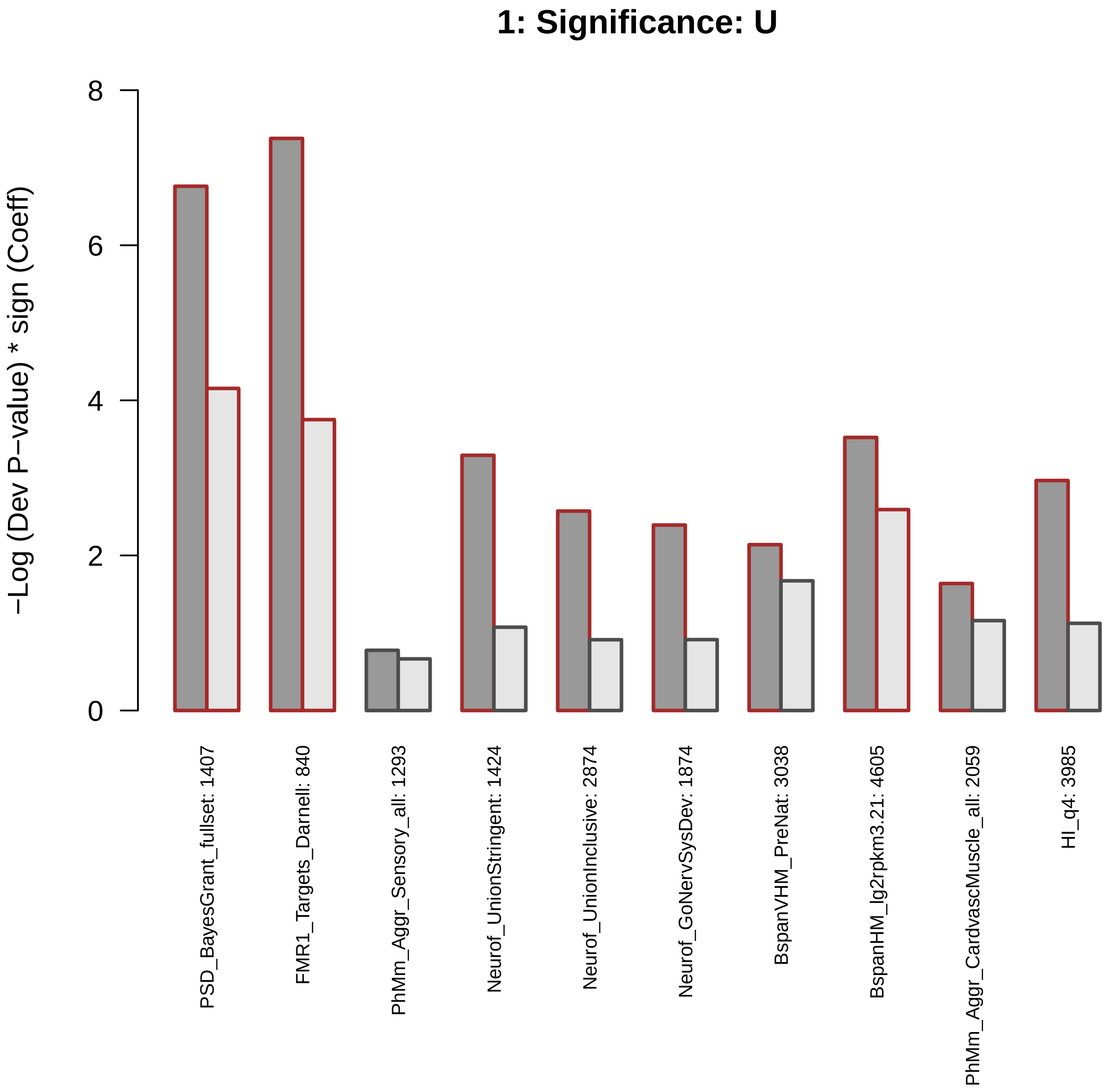
Once the CnvGSAOutput and CnvGSAInput S4 objects are made, the user can run the f.makeViz to create a collection of visualizations. An example of how to run it is shown below:

```
> f.makeViz(cnvGSA.in,cnvGSA.out)
```

There are three different plots that will be made by this function. They show the significance, effect, and support of the gene-sets that the user input. Depending on which corrections the user chooses to include there can be up to 7 plots made. Only the support plots do not vary with the correction choice. The gene-sets that will be displayed on the plots should only include ones that are particularly interesting. Valid gene-set choices include all those that are in the gene-sets in the $res.ls data frame list elements. The user must input a comma separated file with no spaces containing all the gene-sets that they would like to look at. The file should resemble this:

```
PSD_BayesGrant_fullset,FMR1_Targets_Darnell,PhMm_Aggr_Sensory_all,
Neurof_UnionStringent,Neurof_UnionInclusive,Neurof_GoNervSysDev,BspanVHM_PreNat,
BspanHM_lg2rpkm3.21,PhMm_Aggr_CardvascMuscle_all,HI_q4
```

For the plots, anything with an FDR below the user specified cut off will have a brown border. This way the user can easily see how reliable the results are. An example of one of the plots with an FDR cut off of 0.1, the universe correction, and the gene-sets mentioned above is included below:


1: Significance: U
−Log (Dev P−value) * sign (Coeff)
0
2
4
6
8
PSD_BayesGrant_fullset: 1407
FMR1_Targets_Darnell: 840
PhMm_Aggr_Sensory_all: 1293
Neurof_UnionStringent: 1424
Neurof_UnionInclusive: 2874
Neurof_GoNervSysDev: 1874
BspanVHM_PreNat: 3038
BspanHM_lg2rpkm3.21: 4605
PhMm_Aggr_CardvascMuscle_all: 2059
HI_q4: 3985

5.2.4 Enrichment Map pre-processing

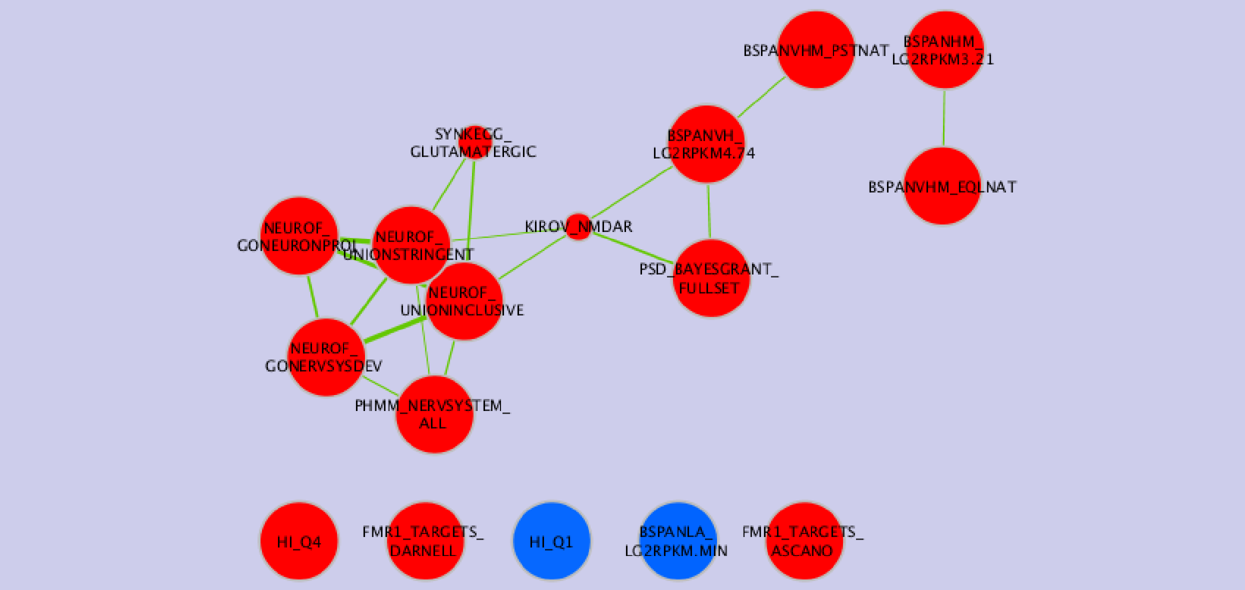
To conduct further analysis of the results, this package has the function `f.enrFiles` that will do all the pre-processing for the files necessary to make an enrichment map. The enrichment map is created using Cytoscape and displays a network of gene-sets and how they interact with one another. It will produce clusters that will convey the data in a meaningful way. The `$klOlp` parameter in the configuration file will influence the generation of the gene-set overlaps by providing a cuttoff. It will only include those with an overlap percentage above that cuttoff and this will be reflected in the generation of the Enrichment Map.

[@JL: could you add something about how the config parameters influence the generation of gene-set overlaps?]

The `f.enrFiles` function will create two files (GMT and generic) needed to make the enrichment map. For more information on these types of files, see the **Enrichment Map User Guide**. Once the function is run, these two files will be output and will be available to use with the **Cytoscape** software. It is recommended to use the Jaccard+Overlap combined option when creating the map. There is a basic example below.

```
## To create the two files, simply run this function
> f.enrFiles(cnvGSA.in,cnvGSA.out)
```

The output will allow the user to use Cytoscape to create a network map that will look similar to this:



The map represents the gene-sets as nodes(circles) and links them together with edges(lines connecting circles). The closer the gene-sets are, the more similar the gene-sets are with one another. This creates clusters which make it easy to determine how the gene-sets compare to one another. The size of the node represents the size of the gene-set and the thickness of the edges is proportional to the overlap between the gene-sets. This overlap refers to the number of genes two gene-sets share with one another. In a two-class experiment such as this one, red nodes display high enrichment in one class and blue nodes display high enrichment in the second class. For a more in depth look into how to interpret these Enrichment Maps, refer to the Enrichment Map paper [7].

If the user would like to use their own files to make the enrichment map they should follow the examples below. This can be found on the **Enrichment Map User Guide** as described above.

The GMT file should follow somehthing like this:

- Each row of the geneset file represents one gene-set and consists of:
    - gene-set name (–tab–) description (–tab–) a list of tab-delimited genes that are part of that gene-set.

The generic file should follow this format:

- The generic results file is a tab delimited file with enriched gene-sets and their corresponding p-values (and optionally, FDR corrections)
- The Generic Enrichment Results file needs:
    - gene-set ID (must match the gene-set ID in the GMT file),
    - gene-set name or description
    - p-value
    - FDR correction value
    - Phenotype: +1 or -1, to identify enrichment in up- and down-regulation, or, more in general, in either of the two phenotypes being compared in the two-class analysis (+1 maps to red and -1 maps to blue).

6 References

[1] Raychaudhuri S et al. Accurately assessing the risk of schizophrenia conferred by rare copy-number variation affecting genes with brain function. *PLoS Genet.* 2010 Sep 9; **6**(9): e1001097.

[2] Pinto, D et al. Functional impact of global rare copy number variation in autism spectrum disorders. *Nature.* 2010 Jul 15; **466**(7304): 368–72.

[3] Kirov G et al. De novo CNV analysis implicates specific abnormalities of postsynaptic signalling complexes in the pathogenesis of schizophrenia. *Mol Psychiatry.* 2012 Feb; **17**(2): 142–53.

[4] Pinto, D et al. Convergence of Genes and Cellular Pathways Dysregulated in Autism Spectrum Disorders. *Am J Hum Genet.* 2014 May 1; **94**(5): 677–694.

[5] Goeman JJ, Bhlmann P. Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics.* 2007 Apr 15; **23**(8): 980–7.

[6] C. R. Marshall et al. Structural Variation of Chromosomes in Autism Spectrum Disorder. *Am J Hum Genet.* 2008 Feb 8; **82**(2): 477–488.

[7] Merico D et al. Enrichment map: a network-based method for gene-set enrichment visualization and interpretation. *PLoS One.* 2010 Nov 15; **5**(11): e13984.