GSEAlm Package

Assaf P. Oron, Robert Gentleman (with contributions from S. Falcon and Z. Jiang)
assaf.oron@gmail.com

April 26, 2022

Abstract

The `GSEAlm` package contains tools for linear-model inference and diagnostics, tailored for Gene Set Enrichment Analysis (GSEA). We demonstrate the package's main features, using the acute lymphoblastic leukemia (ALL) dataset, comparing the `BCR/ABL` and `NEG` phenotypes via GSEA based on chromosome-band mapping of genes. If you want to follow the text while running the code, please use the code from the `GSEAlm.R` file in the same directory where this document is found, because there are some code excerpts which were hidden from view in the tutorial below. A more up-to-date and academically oriented exposition of the tools coded in this package is available at Oron *et al.* (2008).

1 Background and Dataset

Gene set enrichment analysis (Subramanian *et al.*, 2005) is an important new approach to the analysis of gene expression data, and it has already been extended and generalized in a number of ways (Tian *et al.*, 2005; Kim and Volsky, 2005; Jiang and Gentleman, 2007).

Expression analysis in general and GSEA in particular can be viewed as a cascade of successive data reductions:

1. Biochemical hybridization information is reduced to a set of pixel images (one per sample);
2. Images are pre-processed to produce a $G \times n$ matrix of normalized average expression estimates (G genes, n samples);
3. This matrix is then filtered out to remove irrelevant genes or samples (non-specific filtering);
4. Per-gene statistics are calculated;
5. Finally, these statistics are used to calculate gene-set-level statistics, which help identify differentially expressed or otherwise interesting gene-sets.

This process is necessary, in order to bring the amount of information generated by the microrray experiment down to a scientifically relevant and manageable level, while retaining core features.

However, given the immense extent of data reduction, it makes both theoretical and practical sense to find ways to check the process. This is where the linear-model toolset comes in handy.

A linear model assumes that the mean of the response variable has a linear relationship with the explanatory variable(s). In gene-expression terminology, the typical generic model could be written as:

$$y_{gi} = \beta_{g0} + \sum_{j=1}^{p} X_{ij}\beta_{gj} + \epsilon_{gi}, \tag{1}$$

where

- y_{gi} is the gene expression value of gene g in sample i;
- p is the number of explanatory variables in the model;
- X_{ij} is the value of the j-th variable for the i-th sample. For dichotomous variables such as phenotype, one typically sets X to zero or one (e.g., `NEG` will be zero and `BCR/ABL` one);
- β_{gj} is the true value of the effect of variable j upon the expression of gene g; and
- ϵ_{gi} is a random error ("noise"), often assumed to follow a normal distribution with mean zero and variance σ^2.

The data and model are used to calculate a fitted value for each observation, denoted as $\hat{y}_{gi}$. The residuals, $e_{gi} \equiv y_{gi} - \hat{y}_{gi}$ can be naively seen as estimates for the random errors, but they play a far greater role as will be seen below.

Note that applying linear models to gene expression in the way outlined here, involves fitting the same model form to all genes separately and simultaneously.

We load the GSEAlm package. Then we load the data and perform some initial filtering. The example dataset is from a clinical trial in acute lymphoblastic leukemia (ALL). This dataset is available from Bioconductor, under the name ALL.

```
> library("GSEAlm")
```

```
> data(ALL)
> ### Some filtering;
> ### see explanation below code
> bcellIdx <- grep("^B", as.character(ALL$BT))
> bcrOrNegIdx <- which(as.character(ALL$mol.biol)
+                      %in% c("NEG", "BCR/ABL"))
> esetA <- ALL[ , intersect(bcellIdx, bcrOrNegIdx)]
> esetA$mol.biol = factor(esetA$mol.biol) # recode factor
> ### Non-specific filtering
> esetASub <- nsFilter(esetA,var.cutoff=0.6,var.func=sd)$eset
```

The ALL dataset contains 12625 genes and 128 samples. We are interested, among other things, in comparing the BCR/ABL and NEG phenotypes, appearing under the mol.biol variable (hereafter: "the phenotype effect"). There are 79 samples with one of the two phenotypes; we remove all the rest. Standard non-specific filtering of genes was also performed in order to remove most of the unexpressed genes. The common assumption is that only about 40% of genes are expressed in typical tissues, and therefore we remove the 60% with less-variable expression across samples, using standard deviation as the threshold criterion. The filtered dataset contains 79 samples and 3510 unique genes.

Next we identify gene-sets based on chromosomal location. Such an analysis is of clinical interest, as ALL has been associated with chromosomal irregularities. We mapped the chromosomal location of each gene in the filtered dataset, using tools and methods described in Falcon and Gentleman (2008) and available in the Category package. The resulting gene-set structure is hierarchical, and can be represented as a tree graph. More details follow the code excerpt.

```
> minBandSize = 5
> haveMAP = sapply(mget(featureNames(esetASub), hgu95av2MAP),
+                  function(x) !all(is.na(x)))
> workingEset = esetASub[haveMAP, ]
> entrezUniv = unlist(mget(featureNames(workingEset), hgu95av2ENTREZID))
> ### Creating incidence matrix and keeping the graph structure
>
> AgraphChr=makeChrBandGraph("hgu95av2.db",univ=entrezUniv)
> AmatChr = makeChrBandInciMat(AgraphChr)
> AmatChr3 = AmatChr[rowSums(AmatChr)>=minBandSize,]
> # Re-ordering incidence matrix columns
>
> egIds = sapply(featureNames(workingEset), function(x) hgu95av2ENTREZID[[x]])
> idx = match(egIds, colnames(AmatChr))
> AmatChr3 = AmatChr3[, idx]
> colnames(AmatChr3)=featureNames(workingEset)
> # Updating our graph to include only the bands that actually
> # appear in the matrix (doing it a bit carefully though...)
>
> # AmatChr3 = AmatChr[!duplicated(AmatChr),]
> AgraphChr3 = subGraph(c("ORGANISM:Homo sapiens",rownames(AmatChr3)),AgraphChr)
> # AgraphChr3 = subGraph(rownames(AmatChr3),AgraphChr)
>
```

There are 660 bands containing at least 5 genes, and there are 3510 genes mapped to such chromosome bands. To utilize our GSEAlm toolset, we need a chromosome-band **incidence matrix**: a matrix with as many rows as gene-sets, and as many columns as genes. Matrix elements are 1 if the gene belongs to the gene-set, and 0 otherwise. Rather than use the convenient single-step function MAPAmat to create this matrix, here we first create a hierarchical chromosome tree using makeChrBandGraph, and then generate the matrix from the tree using makeChrBandInciMat (all 3 functions are found in the Category package). This way we retain the tree in our working memory,

which will prove useful down the road. Note that the tree's trunk begins with the species-identifying node `ORGANISM:Homo sapiens`, meaning that complete chromosomes are represented as the first-level branches.

2 The t-Test as a Linear Model, and Basic Diagnostics

We arrive at step 4 of the data-reduction cascade outlined in the introduction: individual-gene statistics, which in our case are t-tests. A point often overlooked is that t-tests are identical to a linear model with a single dichotomous explanatory variable (hereafter: "factor"). This means that even for t-tests we can make use of linear-model tools. To demonstrate that the two approaches yield equivalent effect estimates, Fig. 1 plots the t-statistics produced by the `rowttests` function from the genefilter package, vs. the analogous statistics produced by a one-factor phenotype model using the `lmPerGene` function from the GSEAlm package. The values are identical except for a sign flip: the two functions have opposing conventions for factor level ordering (`rowttests` calculates "first in alphabetical order minus second", while `lmPerGene` uses the opposite which is similar to the `lm` default). In our particular application, the former convention makes more sense since the absence of mutation (i.e., `"NEG"`) is a more natural reference. We therefore reverse factor level ordering, via the standard `relevel` function.

```
> lmPhen <- lmPerGene(workingEset,~mol.biol)
> ##fit the t-tests model
> tobsChr <- rowttests(workingEset,"mol.biol")
> ## fit it via the linear-model interface
> lmEsts = lmPhen$tstat[2,]
> plot (tobsChr$stat,lmEsts,main="The t-test as a Linear Model",
+ xlab="T-test t-statistic",ylab="One-Factor Linear Model t-statistic")
> ### Re-leveling the factor
>
> workingEset$mol.biol<-relevel(workingEset$mol.biol,ref="NEG")
> lmPhen <- lmPerGene(workingEset,~mol.biol)
> lmEsts = lmPhen$tstat[2,]
>
```

Using the linear-model function instead of just a t-test, enables us to directly calculate residuals. We have as many residuals as raw data points – in our case, 3510 by 79.

For the residual analysis, it is better to use normalized residuals, so that different genes will have similar magnitude. There are several ways to achieve this, producing normalized, internally-Studentized or externally-Studentized residuals (Cook and Weisberg, 1982). The `getResidPerGene` function allows for all these options (and also the raw un-normalized residuals), with externally-Studentized residuals being the default.

Arranging Studentized residuals by sample helps identify individual samples that tend to exhibit high or low expression levels. Similarly, we can look for samples with noticeably more or less variable residuals. If a particular sample is systematically different from the rest, one may suspect

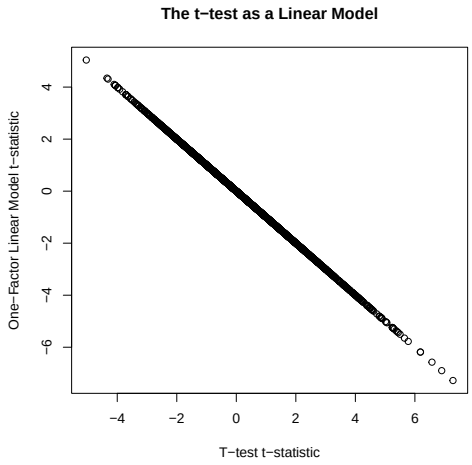

Figure 1: T-statistics for gene-level phenotype effect, from the `rowttests` and `lmPerGene` functions, plotted against each other.

imperfect normalization or other technical issues. Otherwise, this may indicate some real underlying phenomena.

Fig. 2 summarizes all Studentized residuals by sample (one box per sample), arranged by phenotype, using the `resplot` function. Several samples catch the eye, especially in the `NEG` phenotype (left) where the residuals from samples `68001`, `28001` and `16009` are predominantly negative. In the `BCR/ABL` phenotype (right), residuals from sample `84004` appear to be systematically higher than all the rest. Note that these features are **not** evident from boxplots of raw expression intensities.[1]

Note that `lmPerGene` also allows for an intercept-only model (by omitting the formula) – so you can examine residuals for patterns even without assuming any phenotype effect.

```
> lmPhenRes <- getResidPerGene(lmPhen)
> resplot(resmat=exprs(lmPhenRes),fac=workingEset$mol,cex.main=.7,cex.axis=.6,
+ horiz=TRUE,lims=c(-5,5),xname="",col=5,cex=.3)
>
```

3 Gene-Set Residuals

We proceed to the next GSEA step: producing summary statistics of the phenotype effect by chromosome band. There are several ways, all roughly equivalent, to achieve this (Subramanian *et al.*, 2005; Jiang and Gentleman, 2007; Efron and Tibshirani, 2008). Here we chose the statistic of Jiang and Gentleman (2007) (hereafter, "the J-G statistic"), for having the simplest theoretical properties, and for other reasons that will become apparent shortly.

[1] To see this, try replacing variable `lmPhenRes` by `workingEset` in the 'resplot' command below; you'll also need to remove the constraint `lims=c(-5,5)`.

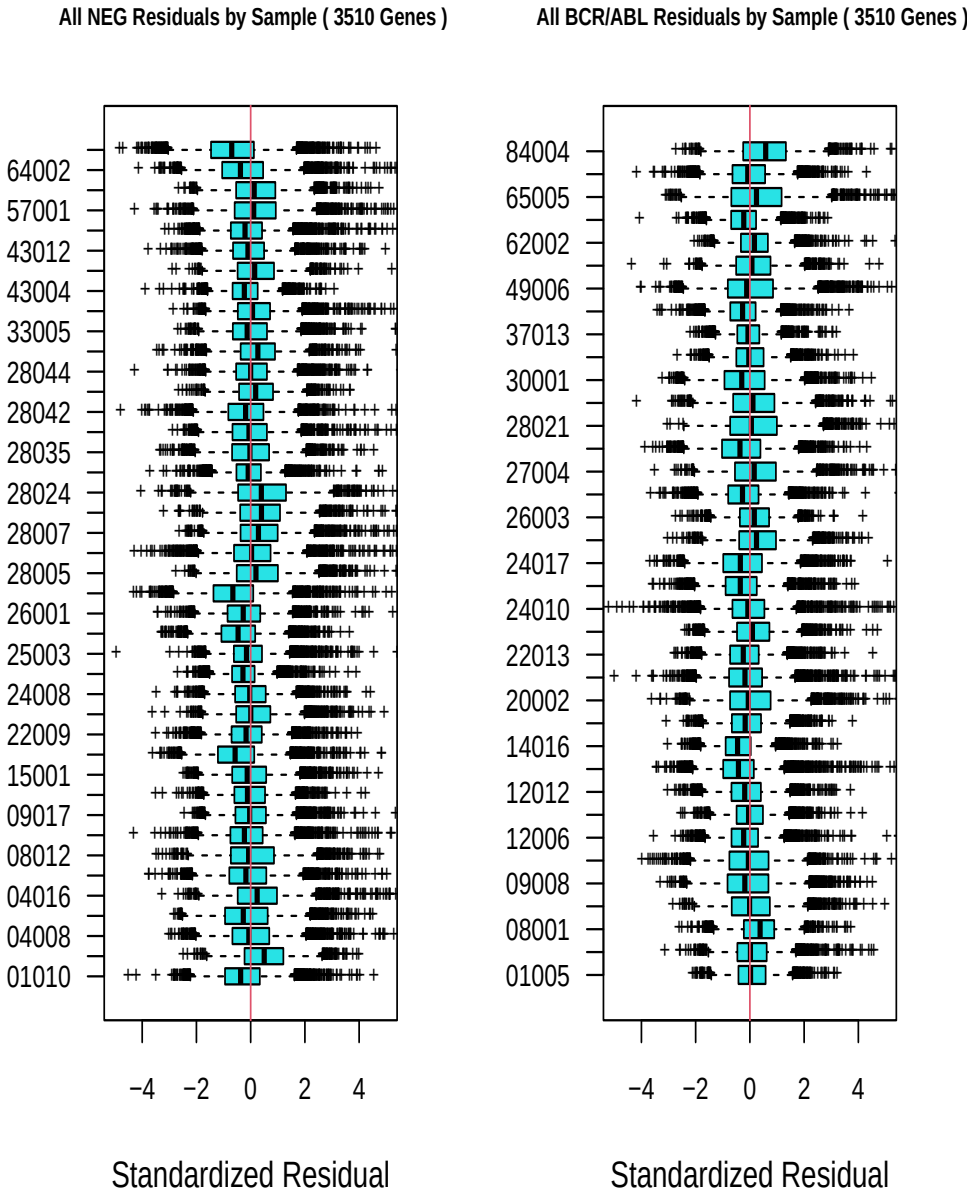

Figure 2: Raw residuals from a linear model of gene expression on phenotype for all genes of the filtered ALL dataset, grouped by sample and arranged by phenotype (NEG on left, BCR/ABL on right).

The J-G statistic can be defined as

$$\tau_S = \sum_{g \in S} (t_g - t_{ref}) / \sqrt{|S|}, \tag{2}$$

where t_g is the phenotype-effect t-statistic for gene g, t_{ref} is some estimate of the overall mean shift (e.g., the median of all t-statistics in the experiment) and $|S|$ is the size of gene set S. Since gene-gene correlations may induce a size effect on τ_S (Oron and Gentleman, forthcoming), and since the sample size is large enough, we recommend basing inference upon phenotype-label ("column") permutation p-values rather than upon comparing τ_S to standard Normal or t distributions. Function `gsealmPerm` provides a reasonably fast implementation of this test for all chromosome bands simultaneously. We skip this test for now, and focus on diagnostics instead.

Similarly to the gene-set statistics, we can create gene-set aggregate residuals by sample, using a formula analogous to (2):

$$r_{Si} = \sum_{g \in S} r_{gi} / \sqrt{|S|}, \tag{3}$$

where r_{gi} is the Studentized residual from sample i and gene g. We will refer to the r_{Si} as "GS residuals". Both GS main effect statistics and GS residuals can be generated from individual-gene information, using the `GSNormalize` function.

Figure 3 displays a heatmap of the r_{Si}, with chromosome bands in rows and samples in columns. Red indicates positive values and blue negative values. Only the lowest possible hierarchical level in each chromosome band (subject to the constraint of at least 5 genes) was used, in order to avoid redundancies and overlap-related distortions. Since we retained the chromosome's tree structure, this is easily achieved using the `leaves` function from the `graph` package. Both rows and columns were simultaneously re-ordered to identify clusters with highly correlated residual patterns. Note that we prefer to measure similarity via correlation, rather than Euclidean distance which is the default for the `heatmap` function. This is because we are interested in finding groups of samples (or chromosome bands) that exhibit the same qualitative behavior with respect to the dataset's baseline.

```
> ## now we are going to aggregate residuals over chromosome bands
>
> stdrAchr=GSNormalize(exprs(lmPhenRes),AmatChr3)
> #rAchr =  AmatChr3 %*% exprs(lmPhenRes)
> #rAsqrSums = sqrt(rowSums(AmatChr3))
> #stdrAchr = sweep(rAchr, 1, rAsqrSums, FUN="/")


> # brColors <- ifelse(colnames(stdrAchr) %in% branch1,"brown", "grey")
> # molColors <- ifelse(workingEset$mol=="BCR/ABL","brown", "grey")
> kinetColors <- ifelse(workingEset$kinet=="hyperd.","brown", "grey")
>  onecor=function(x) as.dist(1-cor(t(x))) # To get correlation-based heatmap
> ### In the heatmap we only use the lowest-level bands, or "leaves" of the graph
```

```
>
> ChrLeaves=leaves(AgraphChr3,"out")
> ### for safety
> ChrLeaves=ChrLeaves[ChrLeaves %in% rownames(AmatChr3)]
> LeafGenes=which(colSums(AmatChr3[ChrLeaves,])>0)
> bandHeatmap=heatmap(stdrAchr[ChrLeaves,],scale="row",col = HMcols,
+           ColSideColors=kinetColors,keep.dendro=TRUE,distfun=onecor,
+           labRow=FALSE,xlab="Sample",ylab="Chromosome band")
```

Fig. 3 exhibits some intriguing patterns. First, one of the samples identified above as having low residuals – `28001` – catches the eye as a narrow predominantly-blue vertical strip, again suggesting a technical normalization issue with this sample.

More interesting from a modeling perspective is the apparent block or checkerboard pattern of the heatmap. This pattern indicates a potential association between certain samples and the expression level of certain chromosomal locations, an association not explained by phenotype.

A skeptic might ask whether perhaps the block pattern is only an artifact of the heatmap's graphical method, which allows for grouping and reordering of both rows and columns. For this purpose, Fig. 4 displays the heatmap of a similarly sized, randomly-generated dataset. Since from the residual plot on Fig. 2 we know that different samples are prone to some normalization-related offset, the data were generated by superimposing independent errors on sample-specific baselines that (randomly) differ between columns. This error structure generates a heatmap with strong vertical features. On the other hand, in the absence of any real correlation between gene-sets and expression, horizontal features, if any, are mild and localized – and therefore there is no apparent block pattern.

So it seems that there may be unaccounted-for factors associated with groups of chromosome bands. Among the 21 descriptive variables available for each sample, we found the "`kinet`" variable to be strongly correlated with the observed pattern. This variable indicates whether the samples exhibit hyperdiploidy, i.e., having substantially more than 46 chromosomes. Clearly, this property could be associated with increased expression of certain chromosomes – and if there are specific hyperdiploidy patterns common in ALL, this could help explain the block pattern of Fig. 3. The `kinet` variable is illustrated as a colored band at the top of the heatmap, with red indicating hyperdiploid samples, grey diploid samples, and white samples with unknown status. Even though hyperdiploid subjects are less than a quarter of the sample size, they form a strong majority in a cluster of samples characterized by predominantly negative residuals over most chromosome bands, and strongly positive residuals over the remaining bands.

We conclude that adding "`kinet`" to the model may be an idea worth pursuing. It should be noted, though, that while such a model expansion might help account for most of the block pattern, it will not be of any use for minority members – especially the 6 diploid samples in "sample cluster 1", and the 8 hyperdiploid samples not belonging to either cluster. Hence, some remnant (or a milder mirror-image) of the block pattern would probably still show after model expansion.

Another factor of interest, which can be used as a benchmark because it certainly plays a role in the expression of some genes, is sex. We conclude that adding sex, too, as a covariate in the model may be of use (a more detailed discussion of Y-chromosome expression and sex assignment in this

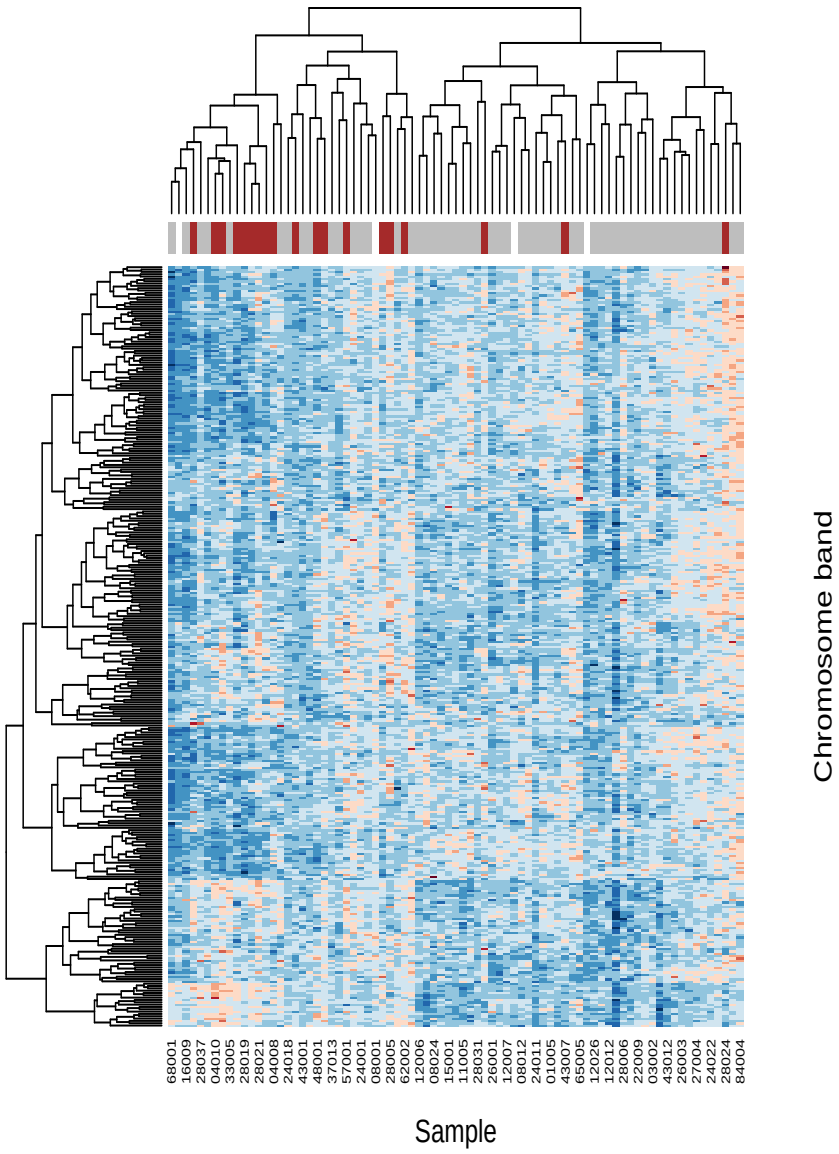

Figure 3: GS residuals from the linear model of gene expression on phenotype for each chromosome band (row) and sample (column). Red colors indicate positive residuals, and blues indicate negative residuals. Both rows and columns are clustered according to a correlation-based similarity measure. The colored band above the map indicates the value of the `kinet` variable – grey for diploid, red for hyperdiploid and white for unknown.

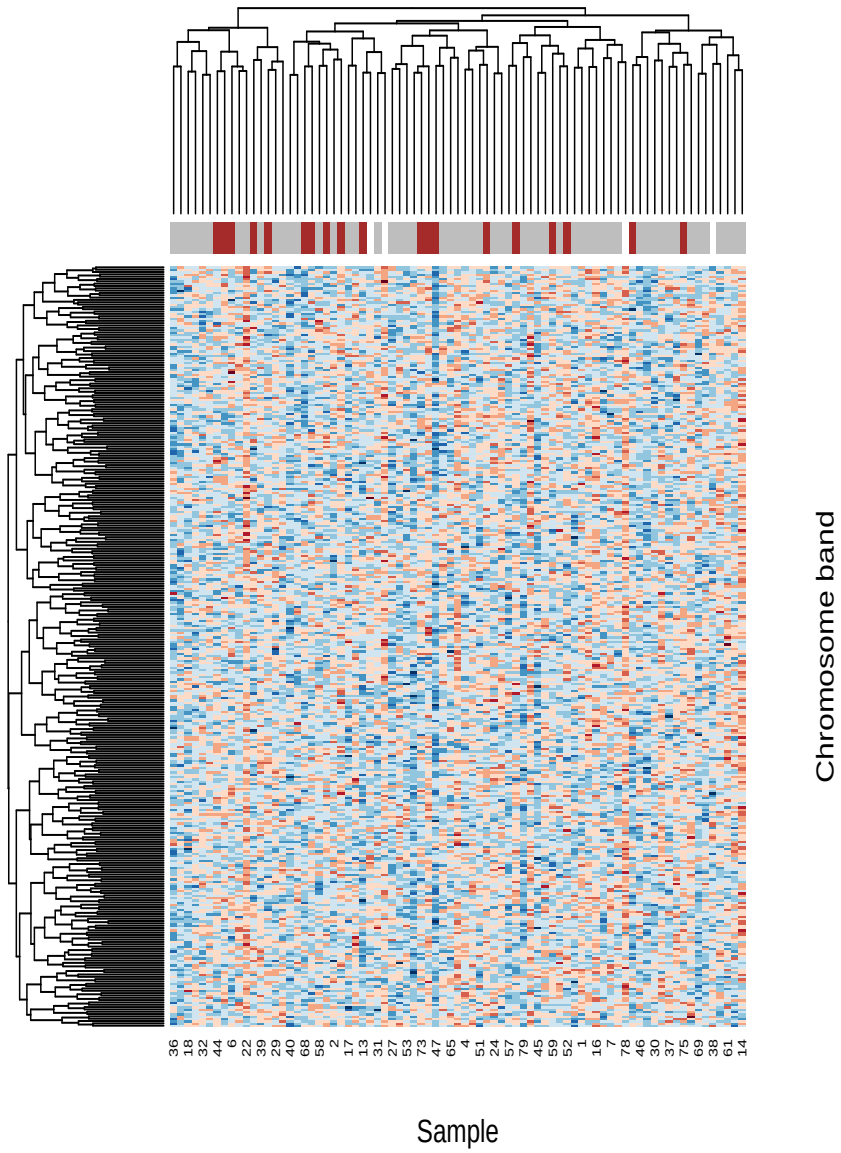

Figure 4: Randomly generated GS residuals with a different baseline for each sample (column). Sample baselines were generated from an exponential distribution, and individual-point noise from a normal distribution. Otherwise, layout and details are identical to those of Fig. 3.

dataset is found in Oron *et al.* (2008)).

4 GSEA and Diagnostics, Using a 3-Variable Model

The previous section's findings suggest a 3-variable model with phenotype, hyperdiploidy and sex; we'd like to implement it using `lmPerGene`. There are 4 samples with missing data for this model; we can either give them up and use the remaining 75, or **impute** the data by artificially filling the missing values, and thus retain all 79 samples. Here we choose the former, mostly because it is very unclear whether to assign sample `25006` as diploid or hyperdiploid. However, if one chooses to impute, the following code excerpts creates a mirror `expressionSet` object named `imputeEset`; substitute for `workingEset` in all subsequent code to see the difference (there should be some difference in residual patterns, but very little in bottom-line inference).

```
> sampleIDs=rownames(pData(workingEset))
> imputeEset=workingEset
> imputeEset$sex[is.na(workingEset$sex)] <- 'M'
> imputeEset$kinet[is.na(workingEset$kinet)] <- 'dyploid'
> ### This is the questionable sample; try once as diploid
> ### (by skipping the following line), and once as hyperdiploid
> imputeEset$kinet[which(sampleIDs=="25006")]<-'hyperd.'
```

We continue with the original (un-imputed) dataset; `lmPerGene` as a default removes samples with missing observations, just like `lm`. We also generate residuals, GS τ statistics and GS residuals.

```
> lmExpand <- lmPerGene(workingEset,~mol.biol+sex+kinet)
> lmExpandRes <- getResidPerGene(lmExpand,type="extStudent")
> lmExpandTees <- t(lmExpand$tstat[2:4,])
> lmExpandBandTees<-GSNormalize(lmExpandTees,AmatChr3)
> GSresidExpand=GSNormalize(exprs(lmExpandRes),AmatChr3)
```

The GS residual heatmap is shown on Fig. 5. The patterns are more predominantly vertical than in Fig. 3, but not as exclusively vertical as in Fig. 4. Most of the remnant patterns resembling Fig. 3's blocks seems to be related to minority members, as explained earlier. Hyperdiploidy (depicted on the top bar) does not appear to be associated with any sample cluster anymore.[2]

```
> kinetColors2 <- ifelse(lmExpandRes$kinet=="hyperd.","brown", "grey")
> bandHeatmapExp=heatmap(GSresidExpand[ChrLeaves,], scale="row",col = HMcols,
+           ColSideColors=kinetColors2,keep.dendro=TRUE,distfun=onecor,
+           labRow=FALSE,xlab="Sample",ylab="Chromosome band")
```

[2]Note that we have to be a bit careful in setting up the heatmap, since we have 4 columns less than in the single-factor one; we take advantage of the fact that residuals are an `expressionSet` type object to redefine the color vector on the top bar.

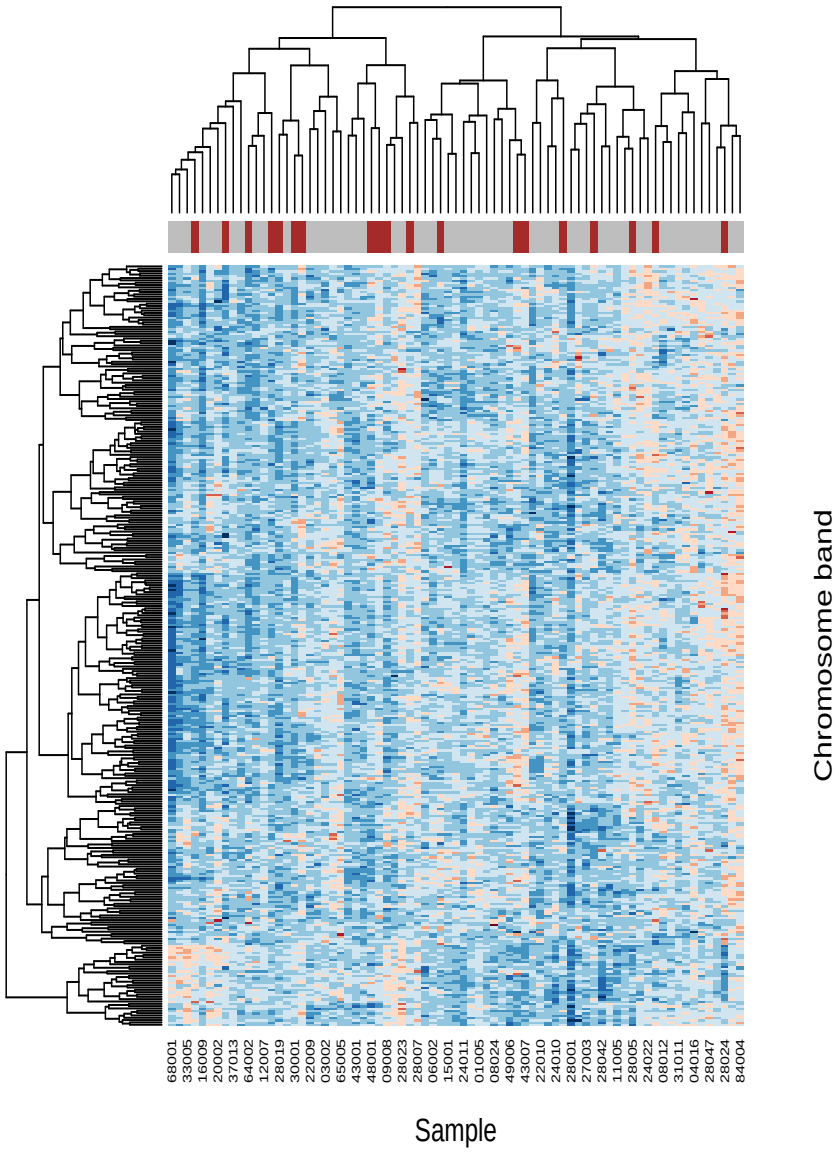

Figure 5: GS residuals from the 3-factor linear model. Layout and details are similar to those of Fig. 3.

We proceed to inference. As mentioned above, due to within-sample correlations the safe approach to inference is via label permutation tests. For a multiple regression model, we can infer about one factor of interest only by permuting its labels **within each subgroup defined by a level combination of the adjusting factors**. For example, the phenotype effect will be evaluated by separately and simultaneously permuting the phenotype labels of hyperdiploid females, hyperdiploid males, diploid females and diploid males, without mixing labels between the four groups. Using this approach the adjusting factors must be categorical (the factor of interest itself can be continuous). Function `gsealmPerm` performs such a permutation analysis. The factors are given as the RHS of a standard `R` model formula, with the first factor being the one of interest. If inference is needed for other factors, the same function can be called with a different factor ordering in the formula. In this document, we run only 125 permutations to save compilation time.

Below is a list of flagged chromosome bands, at the 0.01 level on either side, for the phenotype effect. Theoretically we approach the p-value as an alert flag, rather than an exact probability value (which it is not, due to both the hierarchical structure and multiple testing). In any case, to make the list less redundant we look only at the chromosome-tree "leaves" as before. The first list is of bands down-expressed on the `BCR/ABL` phenotype, followed by a list of those up-expressed on `BCR/ABL`. Note that we have set `removeShift=TRUE` (this is not an argument of `gsealmPerm` itself, but rather passed on to `GSNormalize`) – meaning that differential expression is estimated vs. the baseline gene-level expression gap between the phenotypes (calculated by default as the median of gene-level t-statistics).

```
> pvalsExpand=gsealmPerm(workingEset[LeafGenes,],~mol.biol+sex+kinet,AmatChr3[ChrLeaves,LeafGene
> pvalsExpand[pvalsExpand[,1]<flagp,1]

      13q2       13q33       18q12        9p24     16q22.1     18q21.1
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
      1p33    20q13.12    20q13.33    22q11.23     22q12.1    22q13.31
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
      7p13      7p15.2      7p22.3      7q22.1      8p23.1       Xq22.1
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
      Xq28
0.007936508

> pvalsExpand[pvalsExpand[,2]<flagp,2]

    10p11.2        5q23        6q24        7q31       12q22     14q22.2
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
    15q21.2    16p13.13    18p11.22      1p21.3      1p22.1     1p36.23
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
     1q21.2      1q32.2     20p12.3        2p14      2q11.2      2q31.2
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
     3q25.1        4p14      4q13.3        6q21      6q23.3     9q21.13
0.007936508 0.007936508 0.007936508 0.007936508 0.007936508 0.007936508
     9q33.2
0.007936508
```

These lists are not very different from the ones obtained via a 1-factor model (not shown here; you can try and compare by running `gsealmPerm` on the phenotype-only model). So adjusting with two of the most obvious factors in the dataset does not substantially change inference about the phenotype effect.

The sex effect permutation test is omitted here for brevity. In any case, there are little surprises there - the only locations flagged as differentially expressed at the 0.01 level sit on the Y chromosome.

Finally, when getting to the hyperdiploidy effect our hard work in expanding the model bears fruit. Given the nature of the hyperdiploidy phenomenon, here we focus on complete chromosomes. It turns out – perhaps not so surprisingly – that quite a few chromosomes are flagged as overexpressed among the hyperdiploid samples. Here we leave `removeShift` in its default `FALSE` value, because we'd like to directly gauge the absolute chromosome-level expression differences between the hyperdiploid and diploid groups.

```
> chrNames=c(as.character(1:22),"X")
> pvalsHyper=gsealmPerm(workingEset,~kinet+mol.biol+sex,AmatChr3[chrNames,],nperm=nperm)
> pvalsHyper[pvalsHyper[,2]<flagp,2] ### over-expressed list for hyperdiploids
```

```
         19          21          22           X
0.007936508 0.007936508 0.007936508 0.007936508
```

```
>
> # browser()
>
> # oldflags=c(table(pvals[,1]<flagp)[2],table(pvals[,2]<flagp)[2])
> # newflags=c(table(pvals.Exp[,1]<flagp)[2],table(pvals.Exp[,2]<flagp)[2])
>
> #downtable=table(pvals[,1]<flagp,pvals.Exp[,1]<flagp)
> #uptable=table(pvals[,2]<flagp,pvals.Exp[,2]<flagp)
```

If we relax the alert level to 0.1, the over-expressed list expands to nearly ten chromosomes. Inspecting chromosome-level hyperdiploidy patterns among hyperdiploid subjects more closely (Fig. 6), it can be seen that chromosome X expression levels are closely correlated with chromosomes 6, 14 and 21, while chromosomes 19 and 22 follow a different pattern and are closely correlated with each other. Chromosome X in particular seems to dichotomize samples into two groups with strongly higher-than-average and lower-than-average responses. These observations may have biological significance.

```
> GSChromeResid=GSresidExpand[chrNames,]
> hyperHmap=heatmap(GSChromeResid[pvalsHyper[,2]<0.1,lmExpandRes$kinet=="hyperd."],
+ scale="row",col = HMcols,keep.dendro=TRUE,distfun=onecor,xlab="Sample ID",ylab="Chromosome")
```

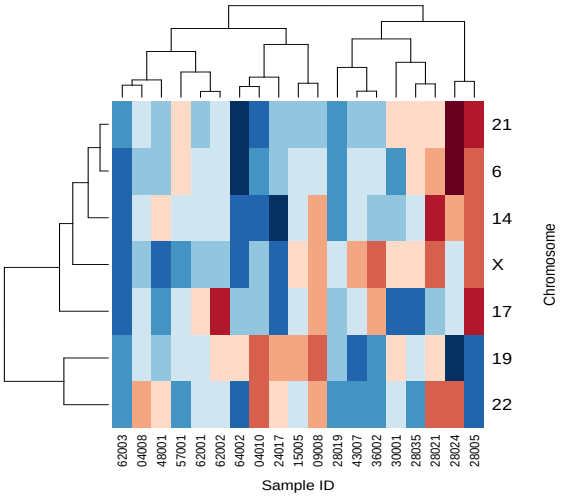

Figure 6: Complete-chromosome GS residuals from the 3-factor linear model, for chromosomes flagged as over-expressed under hyperdiploidy ($p < 0.1$), and for the hyperdiploid subjects only.

4.0.1 Influence Analysis

Recall that some samples (notably `28001`) were visually flagged as potentially problematic outliers. What should we do about them? If there is still access to earlier-stage data (physical samples or raw images), it is always a good idea to inspect questionable samples directly. In the absence of such access, one needs to answer the practical question: how strongly does a single outlying observation affect model inference?

Since our model includes only dichotomous factors, fitting it boils down to calculating group averages – for 2 groups in the 1-factor case and for 8 groups in the 3-factor case. The smaller the group, the more leverage each sample in the group exerts (sample leverage values are obtained via the `Leverage` function). A measure known as Cook's D (Cook and Weisberg, 1982) incorporates a sample's leverage together with the magnitude of its residual, to calculate the square distance by which a single observation "moves" the model's parameter estimates.[3] The `CooksDPerGene` function performs this calculation for all samples and genes simultaneously.

In the GSEA framework, we would probably like to aggregate individual-gene D values within each chromosome band, in an analogous manner to τ_{Si} and r_{Si}. However, normalizing by square-root makes little sense since Cook's D is positive, not pivotal around zero. Instead, we suggest using

$$D_{Si} = \sqrt{\sum_{g \in S} D_{gi}/|S|}, \tag{4}$$

which is simply the root-mean D value over the gene-set. This, too, can be accomplished via the `GSNormalize` function.

D_{Si} provides a measure of the typical amount by which the sample in question affects t-statistics for genes in the band. We are looking for samples whose overall D_{Si} values, across different gene-sets,

[3] The distance is measured in p-dimensional parameter space and normalized by the standard error.

are substantially larger than the overall baseline.

Since the 1-factor model splits the dataset into two roughly equal-sized groups, the leverage of all samples is roughly the same, and we expect this model's D_{Si} values to convey a similar message to Fig. 3. For that model, even the strongest outlier samples do not appear to be substantially away from the pack (Fig. 7, top). The story is different for the 3-factor model (Fig. 7, bottom). There is at least one highly influential sample (relative to the baseline): sample `28024`. This sample has become influential for the expanded model, because it belongs to a female subject with hyperdiploidy – one of the smallest among the 8 groups (by contrast, sample `28001` belongs to the largest group, and hence its diminished influence on the 3-factor model). Additionally, `28024`'s expression levels were quite variable to begin with: its Studentized residuals have the second-highest IQR (see Fig. 2). Again, if there is access to raw data in some form, it is a good idea to inspect them and see why this sample is so highly variable. Otherwise, a sensitivity analysis (repeating the regression without `28024` and comparing the results) may be in order – although in our particular case, the sample's overall influence does not appear to warrant taking this step.

```
> cookie1=CooksDPerGene(lmPhen)
> cookie3=CooksDPerGene(lmExpand)
> BandCooks1=GSNormalize(cookie1,AmatChr3,fun2=identity)
> BandCooks3=GSNormalize(cookie3,AmatChr3,fun2=identity)
> BandCooks1Base=BandCooks1[ChrLeaves,]
> BandCooks3Base=BandCooks3[ChrLeaves,]
> layout(1:2)
> boxplot(sqrt(BandCooks1Base)~col(BandCooks1Base),names=sampleIDs,pch='+',
+         cex=.2,las=3,cex.axis=.4,
+         main="Influence by sample and Chromosome Band: 1-Factor Model",
+         xlab="Sample ID",ylab="Cook's D (Band Root-Mean)",boxwex=.5,
+         cex.main=0.8,cex.lab=0.7,col=5)
> boxplot(sqrt(BandCooks3Base)~col(BandCooks3Base),
+         names=sampleIDs[!is.na(workingEset$kinet)],pch='+',
+         cex=.2,las=3,cex.axis=.4,
+         main="Influence by Sample and Chromosome Band: 3-Factor Model",
+         xlab="Sample ID",ylab="Cook's D (Band Root-Mean)",boxwex=.5,
+         cex.main=0.8,cex.lab=0.7,col=5,ylim=c(0,0.4))
```

The D_{Si} measure can also be used to check whether there are any problem gene-sets, i.e., chromosome bands influenced by an unduly large number of outlying samples. In our case, there are no such bands.[4]

The GSEAlm package also offers two other commonly encountered diagnostics: `dffitsPerGene` and `dfbetasPerGene`, each using a different measure of how individual samples affect estimation. According to recent theory (LaMotte, 1999; Jensen, 2001), all three influence diagnostics convey probabilistically equivalent information from the strict perspective of outlier identification. However, these functions are still useful for visual problem identification.

[4]Graph not shown here; it can be easily produced by constructing a boxplot by rows rather than columns in the code above – you'll also need to set `names=rownames(AmatChr3[ChrLeaves,])`.

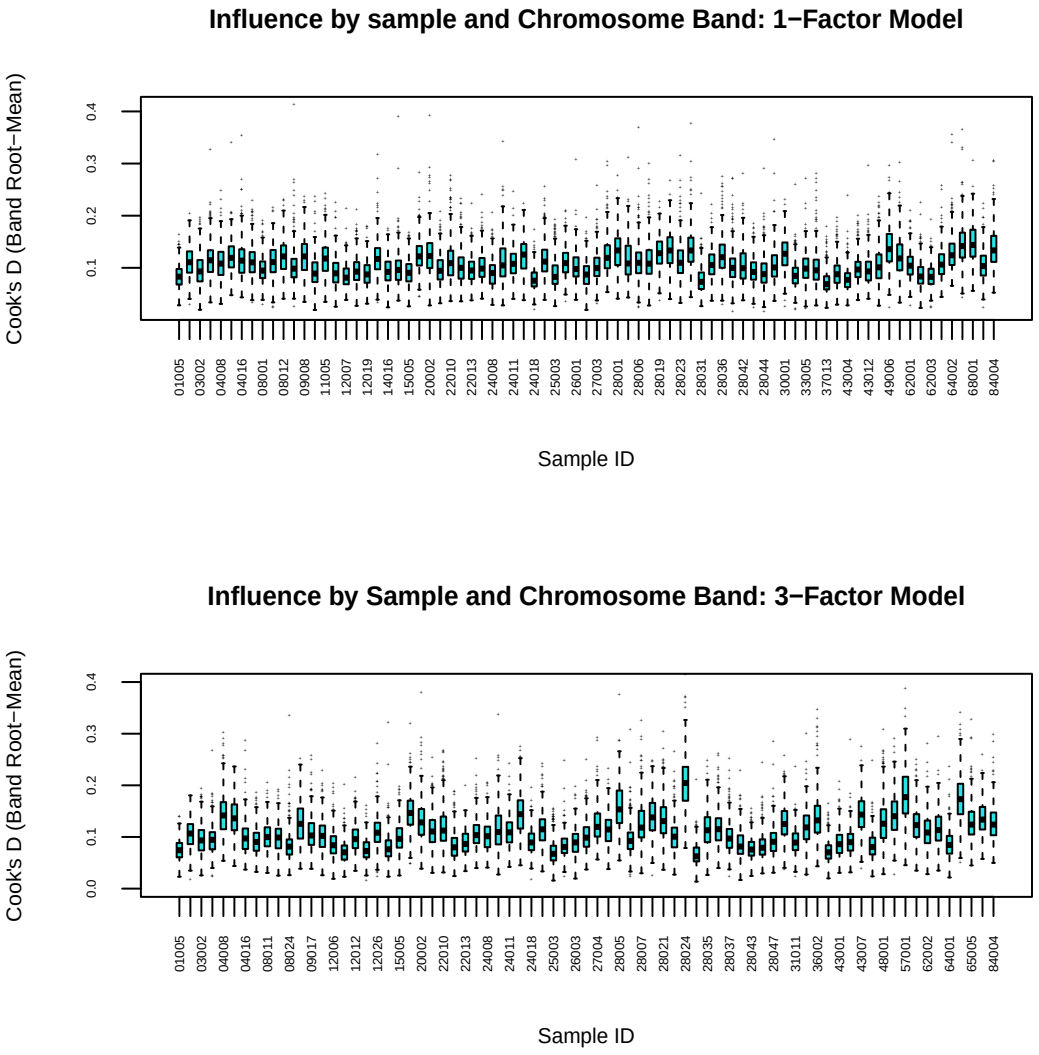

Figure 7: Chromosome-band root-mean Cook's D values, summarized by sample, for the 1-factor (top) and 3-factor (bottom) models.

References

Cook, R. and Weisberg, S. (1982). *Residuals and Influence in Regression*. Monographs on Statistics and Applied Probability. Chapman and Hall.

Efron, B. and Tibshirani, R. (2008). On testing the significance of sets of genes. *Ann. Appl. Stat.*, **1**. In Press.

Falcon, S. and Gentleman, R. (2008). Modeling gene expression data via chromosome bands. *Submitted*, **1**.

Jensen, D. (2001). Properties of selected subset diagnostics in regression. *Statistics and Probability Letters*, **51**, 377–388.

Jiang, Z. and Gentleman, R. (2007). Extensions to gene set enrichment analysis. *Bioinformatics*, **23**, 306–313.

Kim, S.-Y. and Volsky, D. J. (2005). Page: Parametric analysis of gene set enrichment. *BMC Bioinformatics*, **6;144**.

LaMotte, L. (1999). Collapsibility hypotheses and diagnostic bounds in regression analysis. *Metrika*, **50**, 109–119.

Oron, A., Jiang, Z., and Gentleman, R. (2008). Gene set enrichment analysis using linear models and diagnostics. *Bioinformatics*, **24**, 2586–2591.

Subramanian, A., Tamayo, P., Mootha, V. K., *et al.* (2005). Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, **102**(43), 15545–15550.

Tian, L., Greenberg, S. A., Kong, S. W., *et al.* (2005). Discovering statistically significant pathways in expression profiling studies. *Proceedings of the National Academy of Sciences*, **102**(38), 13544–13549.

Session Information

- R version 4.2.0 RC (2022-04-19 r82224 ucrt), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.utf8`, `LC_MONETARY=English_United States.utf8`, `LC_NUMERIC=C`, `LC_TIME=English_United States.utf8`
- Running under: `Windows Server x64 (build 20348)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, stats, stats4, utils
- Other packages: ALL 1.37.0, AnnotationDbi 1.58.0, Biobase 2.56.0, BiocGenerics 0.42.0, Category 2.62.0, GOstats 2.62.0, GSEAlm 1.56.0, IRanges 2.30.0, Matrix 1.4-1, RColorBrewer 1.1-3, S4Vectors 0.34.0, XML 3.99-0.9, annotate 1.74.0, genefilter 1.78.0, graph 1.74.0, hgu95av2.db 3.13.0, lattice 0.20-45, org.Hs.eg.db 3.15.0
- Loaded via a namespace (and not attached): AnnotationForge 1.38.0, Biostrings 2.64.0, DBI 1.1.2, GO.db 3.15.0, GSEABase 1.58.0, GenomeInfoDb 1.32.0, GenomeInfoDbData 1.2.8, KEGGREST 1.36.0, R6 2.5.1, RBGL 1.72.0, RCurl 1.98-1.6, RSQLite 2.2.12, Rcpp 1.0.8.3, Rgraphviz 2.40.0, XVector 0.36.0, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.3, cachem 1.0.6, cli 3.3.0, compiler 4.2.0, crayon 1.5.1, fastmap 1.1.0, grid 4.2.0, httr 1.4.2, memoise 2.0.1, pkgconfig 2.0.3, png 0.1-7, rlang 1.0.2, splines 4.2.0, survival 3.3-1, tools 4.2.0, vctrs 0.4.1, xtable 1.8-4, zlibbioc 1.42.0