Gene set enrichment analysis of RNA-Seq data with the *SeqGSEA* package

Xi Wang[1,2] and Murray Cairns[1,2,3]

October 17, 2016

[1]School of Biomedical Sciences and Pharmacy, The University of Newcastle, Callaghan, New South Wales, Australia
[2]Hunter Medical Research Institute, New Lambton, New South Wales, Australia
[3]Schizophrenia Research Institute, Sydney, New South Wales, Australia

xi.wang@mdc-berlin.de

Contents

1 Introduction

1.1 Background

Transcriptome sequencing (RNA-Seq) has become a key technology in transcriptome studies because it can quantify overall expression levels and the degree of alternative splicing for each gene simultaneously. Many methods and tools, including quite a few R/Bioconductor packages, have been developed to deal with RNA-Seq data for differential expression analysis and thereafter functional analysis aiming at novel biological and biomedical discoveries. However, those tools mainly focus on each gene's overall expression and may miss the opportunities for discoveries regarding alternative splicing or the combination of the two.

SeqGSEA is novel R/Bioconductor package to derive biological insight by integrating differential expression (DE) and differential splicing (DS) from RNA-Seq data with functional gene set analysis. Due to the digital feature of RNA-Seq count data, the package utilizes negative binomial distributions for statistical modeling to first score differential expression and splicing in each gene, respectively. Then, integration strategies are applied to combine the two scores for integrated gene set enrichment analysis. See the publications **?** and **?** for more details. The *SeqGSEA* package can also give detection results of differentially expressed genes and differentially spliced genes based on sample label permutation.

1.2 Getting started

The *SeqGSEA* depends on *Biobase* for definitions of class *ReadCountSet* and class *SeqGeneSet*, *DESeq* for differential expression analysis, *biomaRt* for gene IDs/names conversion, and *doParallel* for parallelizing jobs to reduce running time. Make sure you have these dependent packages installed before you install *SeqGSEA*.

To load the *SeqGSEA* package, type `library(SeqGSEA)`. To get an overview of this package, type `?SeqGSEA`.

```
> library(SeqGSEA)
```

```
> ? SeqGSEA
```

In this Users' Guide of the *SeqGSEA* package, an analysis example is given in Section **??**, and detailed guides for DE, DS, and integrative GSEA analysis are given in Sections **??**, **??**, and **??**, respectively. A guide to parallelize those analyses is given in Section **??**.

1.3 Package citation

To cite this package, please cite the article below:
Wang X and Cairns MJ (2014). **SeqGSEA: a Bioconductor package for gene set enrichment analysis of RNA-Seq data integrating differential expression and splicing.** *Bioinformatics*, **30**(12):1777-9.

To cite/discuss the method used in this package, please cite the article below:
Wang X and Cairns MJ (2013). **Gene set enrichment analysis of RNA-Seq data: integrating differential expression and splicing.** *BMC Bioinformatics*, **14**(Suppl 5):S16.

2 Differential splicing analysis and DS scores

2.1 The *ReadCountSet* class

To facilitate differential splicing (DS) analysis, *SeqGSEA* saves exon read count data using *ReadCountSet* class, which is derived from *eSet*. While below is an example showing the steps to create a new `ReadCountSet` object, creating a `ReadCountSet` object from your own data should refer to Section **??**.

```
> rcounts <- cbind(t(sapply(1:10, function(x) {rnbinom(5, size=10, prob=runif(1))} )),
+                  t(sapply(1:10, function(x) {rnbinom(5, size=10, prob=runif(1))} )))
> colnames(rcounts) <- c(paste("S", 1:5, sep=""), paste("C", 1:5, sep=""))
> geneIDs <- c(rep("G1", 4), rep("G2", 6))
> exonIDs <- c(paste("E", 1:4, sep=""), paste("E", 1:6, sep=""))
> RCS <- newReadCountSet(rcounts, exonIDs, geneIDs)
> RCS

ReadCountSet (storageMode: environment)
assayData: 10 features, 10 samples
  element names: counts
protocolData: none
phenoData
  sampleNames: S1 S2 ... C5 (10 total)
  varLabels: label
  varMetadata: labelDescription
featureData
  featureNames: 1 2 ... 10 (10 total)
  fvarLabels: exonIDs geneIDs ... padjust (10 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

2.2 DS analysis and DS scores

To better illustrate DS analysis functions, we load an example `ReadCountSet` object from a real RNA-Seq data set as follows.

```
> data(RCS_example, package="SeqGSEA")
> RCS_example

ReadCountSet (storageMode: environment)
assayData: 5000 features, 20 samples
  element names: counts
protocolData: none
phenoData
  sampleNames: S1 S2 ... C10 (20 total)
  varLabels: label
  varMetadata: labelDescription
featureData
  featureNames: ENSG00000000003:001 ENSG00000000003:002 ...
    ENSG00000007402:038 (5000 total)
  fvarLabels: exonIDs geneIDs ... padjust (10 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

This example `ReadCountSet` object is comprised of 20 samples and 5,000 exons, part of the prostate cancer RNA-Seq data set (**?**). With the function `geneID` and the script below, we can easily check the number of genes involved in this data set.

```
> length(unique(geneID(RCS_example)))

[1] 182
```

Noticed that some exons are too short or not expressed, we should first filter out these exons from following analysis to secure the robustness of our analysis. By default, function `exonTestablity` marks exons with the sum of read counts across all samples less than `cutoff` (default: 5) to be excluded in downstream analysis. Users can also exclude genes with no or low expression from downstream analysis by checking `geneTestability`.

```
> RCS_example <- exonTestability(RCS_example, cutoff = 5)
```

Then, the main DS analysis is executed using function `estiExonNBstat` for exon DS NB-statistics and function `estiGeneNBstat` for gene DS NB-statistics by averaging exon NB-statistics. Please refer to **?** for detailed statistic analysis regarding differential splicing from exon count data.

```
> RCS_example <- estiExonNBstat(RCS_example)
> RCS_example <- estiGeneNBstat(RCS_example)
> head(fData(RCS_example)[, c("exonIDs", "geneIDs", "testable", "NBstat")])

                     exonIDs         geneIDs testable    NBstat
ENSG00000000003:001    E001 ENSG00000000003     TRUE 2.0219857
ENSG00000000003:002    E002 ENSG00000000003     TRUE 0.2486443
ENSG00000000003:003    E003 ENSG00000000003     TRUE 0.1238136
ENSG00000000003:004    E004 ENSG00000000003     TRUE 1.2058520
ENSG00000000003:005    E005 ENSG00000000003     TRUE 2.0668287
ENSG00000000003:006    E006 ENSG00000000003     TRUE 0.2678247
```

We run DS analysis on the permutation data sets as well. Here we set to run permutation 20 times for demonstration; however, in practice at least 1,000 permutations are recommended. To do so, we first generate a permutation matrix, each column corresponding to each permutation; then run DS analysis on the permutation data sets, and updated `permute_NBstat_gene` slot for results.

```
> permuteMat <- genpermuteMat(RCS_example, times=20)
> RCS_example <- DSpermute4GSEA(RCS_example, permuteMat)
> head(RCS_example@permute_NBstat_gene)
```

```
                 result.1  result.2  result.3  result.4  result.5  result.6
ENSG00000000003 0.1870815 2.8780952 0.5836988 0.5744488 1.1759294 0.6109870
ENSG00000000005 0.5141145 0.5724127 0.5255678 0.2396634 0.4281151 0.5358821
ENSG00000000419 0.4163026 1.3120959 0.7006402 0.5060594 0.4341152 0.7666005
ENSG00000000457 1.2854505 1.0512054 1.0759306 0.6710300 1.0046692 0.4547638
ENSG00000000460 0.7273873 0.7179531 0.5179566 0.9640693 0.7775632 0.4388126
ENSG00000000938 1.1177162 1.4190573 0.8655127 1.1406519 1.3489806 1.5365938
                 result.7  result.8  result.9 result.10 result.11 result.12
ENSG00000000003 2.9031828 2.7197492 0.3994873 0.8412973 0.4248052 0.3971614
ENSG00000000005 0.2395490 0.5553851 0.3659652 0.3328835 0.1075345 0.8041118
ENSG00000000419 4.1587117 1.7757371 0.8066942 0.4304273 0.4403979 0.3242032
ENSG00000000457 0.9023543 1.0173611 0.8864602 0.9360784 0.9936382 1.2755777
ENSG00000000460 0.7235042 1.1116101 0.8995847 1.1847591 0.5909996 0.8971276
ENSG00000000938 1.2824795 1.8480169 1.5957869 1.0777487 1.2950085 1.2853998
                result.13 result.14 result.15 result.16 result.17 result.18
ENSG00000000003 0.3109477 1.3921357 1.0223389 1.1972228 0.5540240 1.1687310
ENSG00000000005 0.4618394 0.3882371 0.3961347 0.9385146 0.6216721 0.1805307
ENSG00000000419 0.9724172 1.9809233 1.4456829 2.1068875 1.2140831 1.2744594
ENSG00000000457 0.7931913 2.4198590 1.9595335 0.5276219 1.0432332 1.9381738
ENSG00000000460 0.4714931 0.6836964 1.0753516 0.6644254 0.4538906 0.8588476
ENSG00000000938 1.1129227 2.0929559 1.6157833 1.8633965 1.3323693 1.7962013
                result.19 result.20
ENSG00000000003 1.1369302 1.1554512
ENSG00000000005 0.5688248 0.4717440
ENSG00000000419 1.0829093 1.8944200
ENSG00000000457 0.7442446 0.8383613
ENSG00000000460 0.9659596 0.6145833
ENSG00000000938 1.1852088 1.6387868
```

The DS NB-statistics from the permutation data sets offer an empirical background of NB-statistics on the real data set. By normalizing NB-statistics against this background, we get the DS scores, which will be used in integrated GSEA runs (Section **??**).

```
> DSscore.normFac <- normFactor(RCS_example@permute_NBstat_gene)
> DSscore <- scoreNormalization(RCS_example@featureData_gene$NBstat,
+                               DSscore.normFac)
> DSscore.perm <- scoreNormalization(RCS_example@permute_NBstat_gene,
+                                    DSscore.normFac)
> DSscore[1:5]
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
      1.3327392       0.9870764       1.4007280       2.6668844       0.9792528
```

```
> DSscore.perm[1:5,1:10]
```

```
                 result.1  result.2  result.3  result.4  result.5  result.6
ENSG00000000003 0.1729537 2.6607510 0.5396198 0.5310684 1.0871271 0.5648473
ENSG00000000005 1.1117573 1.2378254 1.1365247 0.5182649 0.9257862 1.1588291
ENSG00000000419 0.3462874 1.0914229 0.5828040 0.4209485 0.3611041 0.6376709
ENSG00000000457 1.1782996 0.9635804 0.9862446 0.6150951 0.9209233 0.4168562
```

```
ENSG00000000460 0.9483800 0.9360795 0.6753207 1.2569701 1.0138001 0.5721313
                 result.7  result.8  result.9 result.10
ENSG00000000003 2.6839441 2.5143629 0.3693194 0.7777653
ENSG00000000005 0.5180176 1.2010038 0.7913889 0.7198507
ENSG00000000419 3.4592845 1.4770872 0.6710215 0.3580365
ENSG00000000457 0.8271370 0.9325572 0.8125678 0.8580500
ENSG00000000460 0.9433171 1.4493362 1.1728939 1.5447092
```

2.3 DS permutation p-values

Besides calculating DS scores, based on the NB statistics on the real data set and the permutation data sets, we can also calculate a permutation p-value for each gene's DS significance in the studied data set.

```
> RCS_example <- DSpermutePval(RCS_example, permuteMat)
> head(DSresultGeneTable(RCS_example))
```

```
           geneID    NBstat pvalue   padjust
1 ENSG00000000003 1.4416043   0.15 0.2693878
2 ENSG00000000005 0.4564578   0.55 0.6088050
3 ENSG00000000419 1.6839390   0.25 0.3636364
4 ENSG00000000457 2.9094026   0.00 0.0000000
5 ENSG00000000460 0.7510661   0.45 0.5245033
6 ENSG00000000938 1.7949049   0.20 0.3142857
```

The adjusted p-values accounting for multiple testings were given by the BH method (**?**). Users can also apply function `topDSGenes` and function `topDSExons` to quickly get the most significant DS genes and exons, respectively.

3 Differential expression analysis and DE scores

3.1 Gene read count data from *ReadCountSet* class

For gene DE analysis, read counts on each gene should be first calculated. With *SeqGSEA*, users usually analyze DE and DS simultaneously, so the package includes the function `getGeneCount` to facilitate gene read count calculation from a `ReadCountSet` object.

```
> geneCounts <- getGeneCount(RCS_example)
> dim(geneCounts) # 182  20
```

```
[1] 182  20
```

```
> head(geneCounts)
```

```
                 S1  S2  S3  S4  S5  S6   S7  S8  S9 S10  C1  C2  C3  C4  C5
ENSG00000000003 495 235 386 272 255 815 1065 803 839 885 278 270 238 175 292
ENSG00000000005  19   1   0   2   2  12    7   3   1   4   4   4   2   0   1
ENSG00000000419 196 134 165 184 132 344  343 307 342 280 179 156 100 120 126
ENSG00000000457  97  78 141  72 102 219  344 277 337 249  62  48  40  43  52
ENSG00000000460  52  35  48  25  47 105  124  80 156 145  48  36  21  34  19
ENSG00000000938  27  44  57  43  14  71   74 146 148 165  48  59  32  79  20
                 C6   C7  C8  C9 C10
ENSG00000000003 432  519 621 475 560
ENSG00000000005   9    3   1  14  46
ENSG00000000419 169  255 171 164 201
```

```
ENSG00000000457 170  165 131 183 185
ENSG00000000460  68   90  48  72  38
ENSG00000000938 103 1285 137 156  90
```

This function results in a matrix of 182 rows and 20 columns, corresponding to 182 genes and 20 samples.

3.2 DE analysis and DE scores

DE analysis has been implemented in several R/`Bioconductor` packages, of which *DESeq* (**?**) is mainly utilized in *SeqGSEA* for DE analysis. With *DESeq*, we can model count data with negative binomial distributions for accounting biological variations and various biases introduced in RNA-Seq. Given the read count data on individual genes and sample grouping information, basic DE analysis based on *DESeq* including size factor estimation and dispersion estimation, is encapsulated in the function `runDESeq`.

```
> label <- label(RCS_example)
> DEG <- runDESeq(geneCounts, label)
```

The function `runDESeq` returns a `CountDataSet` object, which is defined in the *DESeq* package. The DE analysis in the *DESeq* package continues with the output `CountDataSet` object and conducts negative-binomial-based statistical tests for DE genes (using `nbinomTest` or `nbinomGLMTest`). However, in this *SeqGSEA* package, we define NB statistics to quantify each gene's expression difference between sample groups.

The NB statistics for DE can be achieved by the following scripts.

```
> DEGres <- DENBStat4GSEA(DEG)
> DEGres[1:5, "NBstat"]
```

```
[1]  0.5426504  0.2503510  0.0231052 14.3384053  1.4101270
```

Similarly, we run DE analysis on the permutation data sets as well. The `permuteMat` should be the same as used in DS analysis on the permutation data sets.

```
> DEpermNBstat <- DENBStatPermut4GSEA(DEG, permuteMat)
> DEpermNBstat[1:5, 1:10]
```

```
        result.1  result.2   result.3  result.4    result.5  result.6
[1,] 1.485677100 0.0148533 1.04735957 0.4544516 1.110738338 0.1471130
[2,] 0.005320307 0.8255690 1.37737723 0.2644379 0.001813971 0.4341747
[3,] 0.050652609 0.3055821 0.40921181 0.3204775 2.456744443 0.4531558
[4,] 2.312731130 0.1264944 1.33024171 0.6121379 2.188813948 1.5402304
[5,] 0.078569764 2.0488329 0.09252033 7.0810633 0.514573497 8.0257154
        result.7  result.8   result.9  result.10
[1,]  7.10052374 1.0716235 0.62586806 0.04377268
[2,]  0.09361178 1.0026635 0.08657665 1.84394362
[3,] 13.02274116 0.8282924 0.91689975 0.01211893
[4,]  3.14502538 0.3627286 4.03839445 0.05286897
[5,]  6.38156410 0.3861598 1.21823753 0.02399327
```

Once again, the DE NB-statistics from the permutation data sets offer an empirical background, so we can normalize NB-statistics against this background. By doing so, we get the DE scores, which will also be used in integrated GSEA runs (Section **??**).

```
> DEscore.normFac <- normFactor(DEpermNBstat)
> DEscore <- scoreNormalization(DEGres$NBstat, DEscore.normFac)
> DEscore.perm <- scoreNormalization(DEpermNBstat, DEscore.normFac)
> DEscore[1:5]
```

```
[1] 0.43285802 0.24160117 0.01910237 9.27957222 0.75545277
```

```
> DEscore.perm[1:5, 1:10]
```

```
        result.1   result.2   result.3  result.4    result.5  result.6
[1,] 1.185085759 0.01184809 0.83545133 0.3625042 0.886006917 0.1173482
[2,] 0.005134361 0.79671515 1.32923760 0.2551958 0.001750572 0.4190002
[3,] 0.041877367 0.25264196 0.33831847 0.2649568 2.031129124 0.3746495
[4,] 1.496760279 0.08186503 0.86090982 0.3961652 1.416563185 0.9968110
[5,] 0.042092483 1.09762917 0.04956627 3.7935654 0.275674449 4.2996475
        result.7  result.8   result.9  result.10
[1,]  5.66390205 0.8548060 0.49923858 0.03491633
[2,]  0.09034002 0.9676202 0.08355078 1.77949739
[3,] 10.76663425 0.6847960 0.75805271 0.01001940
[4,]  2.03540697 0.2347518 2.61358025 0.03421590
[5,]  3.41882001 0.2068789 0.65265110 0.01285401
```

3.3 DE permutation p-values

Similar to DS analysis, comparing NB-statistics on the real data set and those on the permutation data sets, we can get permutation p-values for each gene's DE significance.

```
> DEGres <- DEpermutePval(DEGres, DEpermNBstat)
> DEGres[1:6, c("NBstat", "perm.pval", "perm.padj")]
```

```
                   NBstat perm.pval perm.padj
ENSG00000000003  0.5426504      0.45         1
ENSG00000000005  0.2503510      0.70         1
ENSG00000000419  0.0231052      0.95         1
ENSG00000000457 14.3384053      0.00         0
ENSG00000000460  1.4101270      0.30         1
ENSG00000000938  2.1976989      0.00         0
```

For a comparison to the nominal p-values from exact testing and forming comprehensive results, users can run `DENBTest` first and then `DEpermutePval`, which generates results as follows.

```
> DEGres <- DENBTest(DEG)
> DEGres <- DEpermutePval(DEGres, DEpermNBstat)
> DEGres[1:6, c("NBstat", "pval", "padj", "perm.pval", "perm.padj")]
```

```
                   NBstat         pval         padj perm.pval perm.padj
ENSG00000000003  0.5426504 3.956985e-01 5.408276e-01      0.45         1
ENSG00000000005  0.2503510 3.300042e-01 4.943803e-01      0.70         1
ENSG00000000419  0.0231052 9.244775e-01 9.839468e-01      0.95         1
ENSG00000000457 14.3384053 9.960426e-05 2.589711e-03      0.00         0
ENSG00000000460  1.4101270 1.370959e-01 2.970412e-01      0.30         1
ENSG00000000938  2.1976989 7.309013e-07 4.434134e-05      0.00         0
```

4 Integrative GSEA runs

4.1 DE/DS score integration

We have proposed two strategies for integrating normalized DE and DS scores (**?**), one of which is the weighted summation of the two scores and the other is a rank-based strategy. The functions `geneScore` and `genePermuteScore` implement two methods for the weighted summation strategy: weighted linear

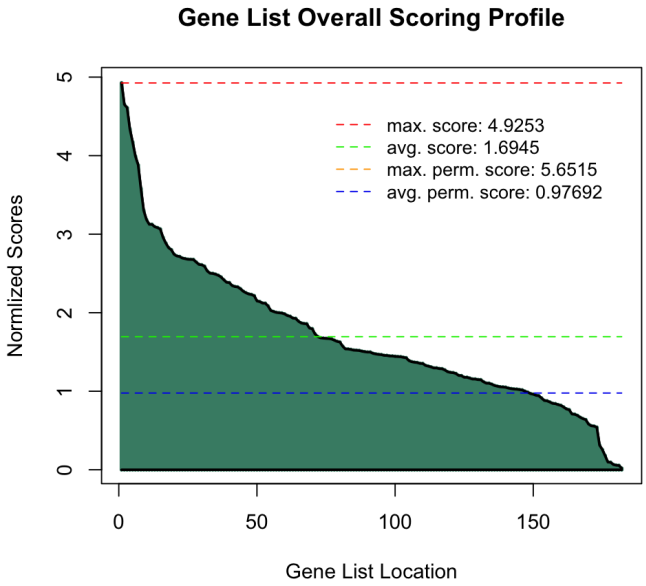

Figure 1: Gene scores resulted from linear combination. Scores are sorted from the largest to the smallest. Red, green, orange, blue dotted horizontal lines represent the maximum score, average score on the real data set, and the maximum score, average score on the permutation data sets.

combination and weighted quadratic combination. Scripts below show a linear combination of DE and DS scores with weight for DE equal to 0.3. Users should keep the weight for DE in `geneScore` and `genePermuteScore` the same, and the weight rangs from 0 (i.e., DS only) to 1 (i.e., DE only). Visualization of gene scores can be made by applying the `plotGeneScore` function.

```
> gene.score <- geneScore(DEscore, DSscore, method="linear", DEweight = 0.3)
> gene.score.perm <- genePermuteScore(DEscore.perm, DSscore.perm,
+                                     method="linear",  DEweight=0.3)
> plotGeneScore(gene.score, gene.score.perm)
```

The plot generated by the `plotGeneScore` function (Fig. **??**) can also be saved as a PDF file easily with the `pdf` argument of `plotGeneScore`.

The functions `geneScore` and `genePermuteScore` also implement one method for the rank-based integration strategy: using data-set-specific ranks. The plot for integrated gene scores is shown in Fig. **??**.

```
> gene.score <- geneScore(DEscore, DSscore, method="rank", DEweight = 0.3)
> gene.score.perm <- genePermuteScore(DEscore.perm, DSscore.perm,
+                                     method="rank",  DEweight=0.3)
> plotGeneScore(gene.score, gene.score.perm)
```

Rather than the above method to integrate scores with data-set-specific ranks, an alternative method is implemented with the `rankCombine` function, which takes only the ranks from the real data set for integrating DE and DS scores on both real and permutation data sets. This provides a method in a global manner. The plot of gene scores is shown in Fig. **??**.

```
> combine <- rankCombine(DEscore, DSscore, DEscore.perm, DSscore.perm, DEweight=0.3)
> gene.score <- combine$geneScore
```

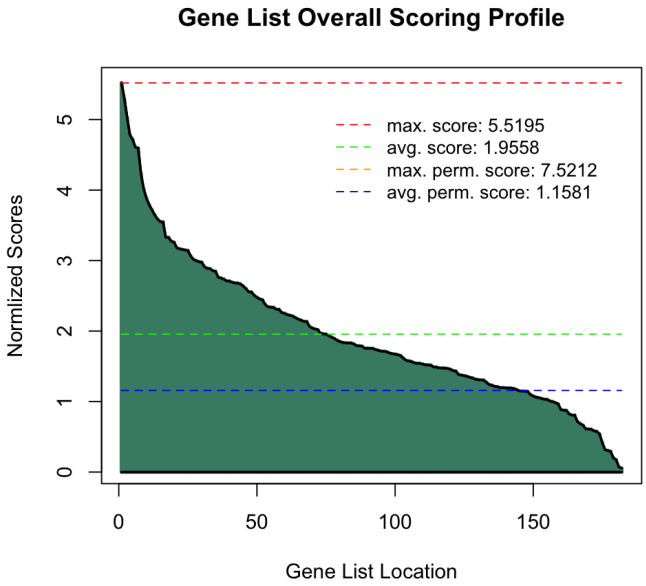

Figure 2: Gene scores resulted from rank-based combination with data-set-specific ranks. Scores are sorted from the largest to the smallest. Red, green, orange, blue dotted horizontal lines represent the maximum score, average score on the real data set, and the maximum score, average score on the permutation data sets.

```
> gene.score.perm <- combine$genePermuteScore
> plotGeneScore(gene.score, gene.score.perm)
```

Basically the integrated gene scores are distributed similarly with the three integration methods at DE weight 0.3 (Figs. **??**, **??**, and **??**); however, according to the analysis in **?**, SeqGSEA can detect slightly more significant gene sets with rank-based integration strategy than with linear combination.

4.2 Initialization of *SeqGeneSet* objects

To facilitate running gene set enrichment analysis, *SeqGSEA* implements a *SeqGeneSet* class. The *SeqGeneSet* class has several slots for accommodating a category of gene sets derived from any biological knowledge-based databases such as Kyoto Encyclopedia of Genes and Genomes (KEGG). However, we recommend to start with the formatted gene-set files from the well-maintained resource Molecular Signatures Database (MSigDB, `http://www.broadinstitute.org/gsea/msigdb/index.jsp`) (**?**). After downloading a `gmt` file from the above URL, users can use `loadGenesets` to initialize a `SeqGeneSet` object easily. Please note that with the current version of *SeqGSEA*, only gene sets with gene symbols are supported, though read count data's gene IDs can be either gene symbols or Ensembl Gene IDs.

Below is shown an example of the `SeqGeneSet` object, which contains information such as how many gene sets in this object and the names/sizes/descriptions of each gene set.

```
> data(GS_example, package="SeqGSEA")
> GS_example

SeqGeneSet object: gs_symb.txt
GeneSetSourceFile: /Library/Frameworks/R.framework/Versions/2.15/Resources/library/SeqGSEA/extdat
GeneSets: ERB2_UP.V1_DN
```

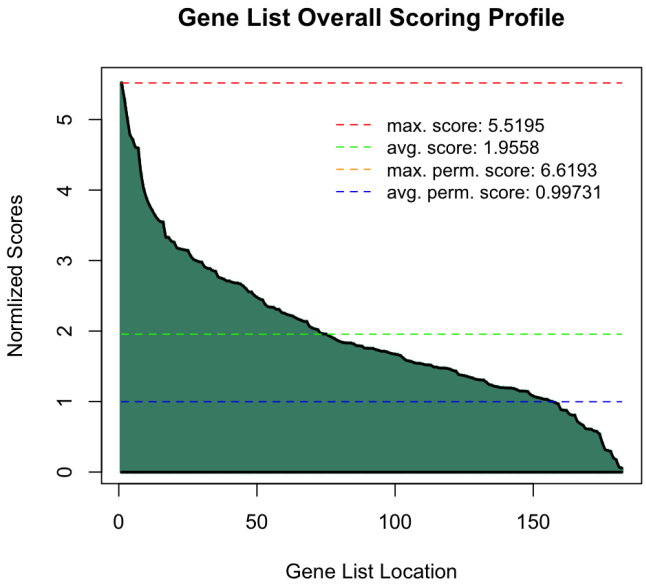

Figure 3: Gene scores resulted from rank-based combination with the same rank got from the real data set. Scores are sorted from the largest to the smallest. Red, green, orange, blue dotted horizontal lines represent the maximum score, average score on the real data set, and the maximum score, average score on the permutation data sets.

```
          AKT_UP_MTOR_DN.V1_UP
          ...
          KRAS.600.LUNG.BREAST_UP.V1_DN
  with the number of genes in respective sets: 6, 6, ..., 5
  brief descriptions:
          http://www.broadinstitute.org/gsea/msigdb/cards/ERB2_UP.V1_DN
          http://www.broadinstitute.org/gsea/msigdb/cards/AKT_UP_MTOR_DN.V1_UP
          ...
          http://www.broadinstitute.org/gsea/msigdb/cards/KRAS.600.LUNG.BREAST_UP.V1_DN
  # gene sets passed filter: 11  (#genes >= 5 AND <= 1000)
  # gene sets excluded: 178  (#genes < 5 OR > 1000)
ES scores: not computed
ES postions: not computed
Permutated ES scores: not performed
ES scores normalized: No
ES p-value: not computed
ES FWER: not computed
ES FDR: not computed
```

4.3 running GSEA with integrated gene scores

With the initialized `SeqGeneSet` object and integrated gene scores as well as gene scores on the permutation data sets, the main `GSEnrichAnlyze` can be executed; and the `topGeneSets` allows users promptly access to the top significant gene sets.

```
> GS_example <- GSEnrichAnalyze(GS_example, gene.score, gene.score.perm)
> topGeneSets(GS_example, 5)
```

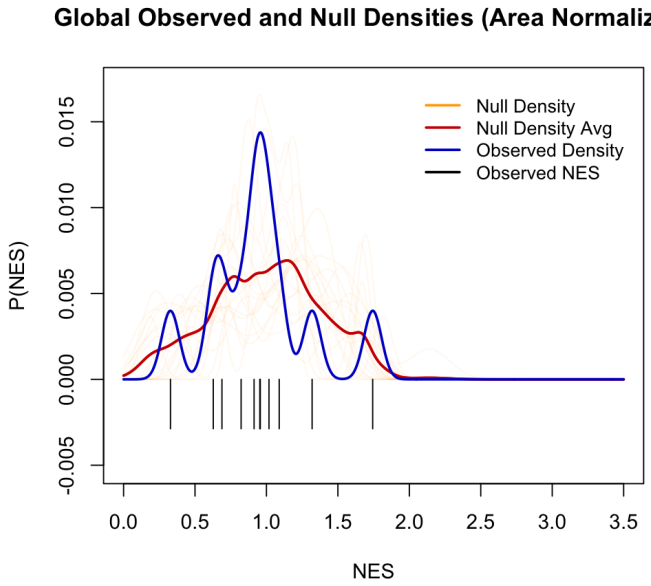

Figure 4: Distribution of normalized enrichment scores (NES) on the observed and permutation (null) data sets. Blue: observed NES density; Orange: each for NES density on one permutation data set; Red: the average density on all permutation data sets; Black: observed NES values.

```
                  GSName GSSize        ES ES.pos pval     FDR FWER
9              TBK1.DF_UP      5 1.7435652     24 0.00 0.00000  0.1
10             NFE2L2.V2      9 0.9133443     88 0.50 0.85714  1.0
3           AKT_UP.V1_UP      6 0.8225888     95 0.70 0.87500  1.0
2  AKT_UP_MTOR_DN.V1_UP      6 0.6277070     18 0.80 0.90000  1.0
1          ERB2_UP.V1_DN      6 0.9534551      8 0.65 1.00000  1.0
```

The main GSEA includes several steps detailed in **?** and its original paper **?**. In *SeqGSEA*, functions `calES`, `calES.perm`, `normES` and `signifES` are implemented to complete the analysis. Advanced users may set up customized pipelines with the functions above themselves.

4.4 *SeqGSEA* result displays

Several functions in *SeqGSEA* can be employed for visualization of gene set enrichment analysis running results. The `plotES` function is to plot the distribution of normalized enrichment scores (NES) of all gene sets in a `SeqGeneSet` object on the observed data set versus its empirical background provided by the NES on the permutation data sets (Fig. **??**).

```
> plotES(GS_example)
```

The `plotSig` function plots the distributions of permutation p-value, false discovery rate (FDR) and family-wise error rate (FWER) versus NES. The example plot is not shown in this vignette as the distributions can be far from the real ones due to the limited permutation times.

```
> plotSig(GS_example)
```

The `plotSigGS` function is to plot detailed results of a particular gene set that has been analyzed. Information in the plot includes running enrichment scores, null NES on the permutation data sets. See Fig. **??** for an example.

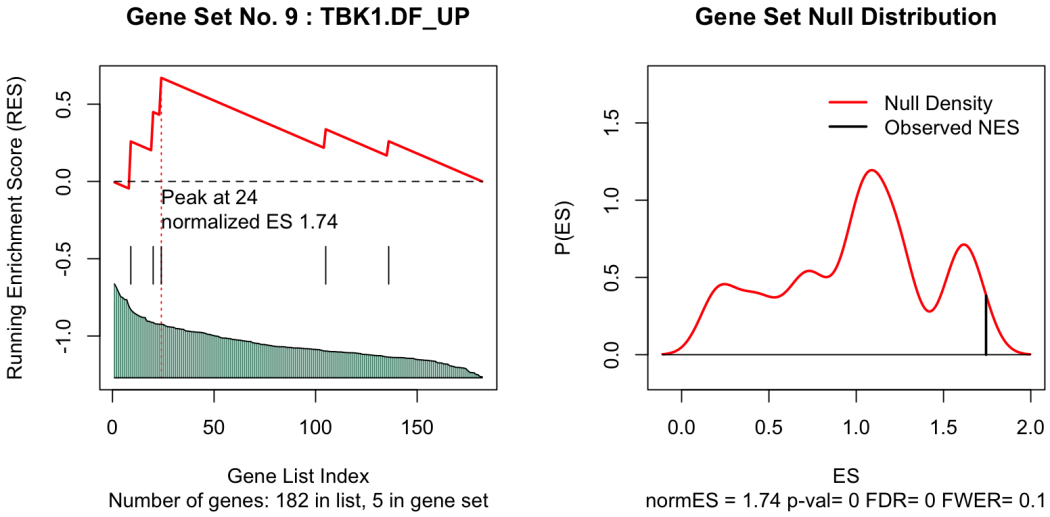

Figure 5: Left: gene locations of a particular gene set according to the gene score rank and running enrichment scores; Right: null NES distribution and the relative position of the observed NES.

```
> plotSigGeneSet(GS_example, 9, gene.score) # 9th gene set is the most significant one.
```

Besides the functions to generate plots, the `writeSigGeneSet` function can write the detailed information of any analyzed gene sets, including NES, p-values, FDR, and the leading set (see the definition in **?**). An example is shown below.

```
> writeSigGeneSet(GS_example, 9, gene.score) # 9th gene set is the most significant one.
```

```
GSEA result for gene set No. 9:
genesetName         gs_symb.txt:TBK1.DF_UP
genesetSize         5
genesetDesc         http://www.broadinstitute.org/gsea/msigdb/cards/TBK1.DF_UP
NES         1.74356520275095
Pos         24
pvalue         0
FDR         0
FWER         0.1

Leading set:
ENSG00000005194         4.01404510265715
ENSG00000002919         3.25899451088124
ENSG00000001167         3.14787736063795

Whole gene set:
ENSG00000005194         4.01404510265715
ENSG00000002919         3.25899451088124
ENSG00000001167         3.14787736063795
ENSG00000005059         1.57467401250473
ENSG00000006576         1.21405041598915
```

The `GSEAresultTable` generates a summary table of the GSEA analysis, which can also be output with customized scripts. An example can be found in Section **??**.

5 Running *SeqGSEA* with multiple cores

5.1 R-parallel packages

There are many R packages for facilitating users in running R scripts in parallel, including *parallel*, *snowfall*, *multicore*, and many others. While experienced users may parallelize *SeqGSEA* runnings with the above packages themselves to reduce the running time, we provide with in the *SeqGSEA* package vignette an general way for users to parallelize their runnings utilizing the *doParallel* package (which depends on *parallel*).

First, we show a toy example for a basic idea how *doParallel* works. Basically, *doParallel* is a *parallel backend* for the *foreach* package using *parallel*, which provides a mechanism to execute *foreach* loops in parallel. With the `foreach` function in the *foreach* package, we can specify which `foreach` loops need to be parallelized using the `%dopar%` operator. However, without a registered parallel backend, the `foreach` loops will be executed sequentially even if the `%dopar%` operator is used. In those cases, the *foreach* package will issue a warning that it is running sequentially. Below are two running examples showing how the task is running sequentially and in parallel, respectively.

Run sequentially without parallel backend registered

```
> library(doParallel)
> a <- matrix(1:16, 4, 4)
> b <- t(a)
> foreach(b=iter(b, by='col'), .combine=cbind) %dopar%
+   (a %*% b)
```

```
     [,1] [,2] [,3] [,4]
[1,]  276  304  332  360
[2,]  304  336  368  400
[3,]  332  368  404  440
[4,]  360  400  440  480
```

Although the warning message didn't appear here, you would definitely see a warning message when you run the scripts above, like:
Warning message:
executing %dopar% sequentially: no parallel backend registered

Run in parallel with two cores

```
> library(doParallel)
> cl <- makeCluster(2) # specify 2 cores to be used in this computing
> registerDoParallel(cl)
> getDoParWorkers() # 2
```

```
[1] 2
```

```
> a <- matrix(1:16, 4, 4)
> b <- t(a)
> foreach(b=iter(b, by='col'), .combine=cbind) %dopar%
+   (a %*% b)
```

```
     [,1] [,2] [,3] [,4]
[1,]  276  304  332  360
[2,]  304  336  368  400
[3,]  332  368  404  440
[4,]  360  400  440  480
```

The parallel backend registration was done with `registerdoParallel`. For more details please refer to *doParallel*'s vignette (`http://cran.r-project.org/web/packages/doParallel/index.html`).

5.2 Parallelizing analysis on permutation data sets

In *SeqGSEA*, the loops for analyzing permutation data sets are implemented by `foreach` with `%dopar%` operator used. Those loops include DS, DE, and GSEA analyses, which are the most time consuming parts. Although there are three parts can take the advantage of parallel running, users only need to register parallel backend once at the beginning of all analyses. See an analysis example in the next section (Section **??**).

6 Analysis examples

6.1 Starting from your own RNA-Seq data

With this *SeqGSEA* package, we provide complementary Python scripts for counting reads on exons of each genes from SAM/BAM files: two scripts `prepare_exon_annotation_refseq.py` and `prepare_exon_annotation_ensembl.py` for preparing (sub-)exon annotation, and `count_in_exons.py` for counting reads. The scripts are based on the `HTSeq` Python package (`http://www-huber.embl.de/users/anders/HTSeq/`). Please install it before using the Python scripts provided. The scripts can be found in the directory given by the following command.

```
> system.file("extscripts", package="SeqGSEA", mustWork=TRUE)
```

```
[1] "/private/tmp/Rtmp3Osb1g/Rinst6ba54f216e04/SeqGSEA/extscripts"
```

Simply by typing "python" + the file name of echo script in your shell console, the help documentation will be on your screen.

Other than the Python scripts provided, users who prefer playing with R/`Bioconductor` packages can also use `easyRNASeq` in *easyRNASeq*, `summarizeOverlaps` in *GenomicRanges*, and `featureCounts` in *Rsubread* to count reads that mapped to each exon. Please refer to respective packages for detailed usage.

For users who are not familiar with RNA-Seq data processing, the upstream steps of counting reads are (1) data preprocessing, including adapter removal, low-quality read filtering, data quality-control analysis, and (2) read mapping. R/`Bioconductor` users can apply *Rsubread* to map reads based on a seed-and-vote approach, as well as a few QC analysis. Users familiar with command-line can choose from a wide range of tools, such as already widely used ones including `TopHat` (`http://tophat.cbcb.umd.edu`), `START` (`http://code.google.com/p/rna-star`), and etc..

6.2 Exemplified pipeline for integrating DE and DS

Below is shown a typical SeqGSEA running example with the data enclosed with the *SeqGSEA* package, which are a part of the prostate cancer data set (**?**). We divide the process into five steps for a complete SeqGSEA run.

Step 0: Initialization. (Users should change values in this part accordingly.)

```
> rm(list=ls())
> # input count data files
> data.dir <- system.file("extdata", package="SeqGSEA", mustWork=TRUE)
> case.pattern <- "^SC"  # file name starting with "SC"
> ctrl.pattern <- "^SN"  # file name starting with "SN"
> case.files <- dir(data.dir, pattern=case.pattern, full.names = TRUE)
> control.files <- dir(data.dir, pattern=ctrl.pattern, full.names = TRUE)
> # gene set file
> geneset.file <- system.file("extdata", "gs_symb.txt",
+                             package="SeqGSEA", mustWork=TRUE)
```

```
> # output file prefix
> output.prefix <- "SeqGSEA.test"
> # setup parallel backend
> library(doParallel)
> cl <- makeCluster(2) # specify 2 cores to be used in computing
> registerDoParallel(cl) # parallel backend registration
> # setup permutation times
> perm.times <- 20 # change the number to >= 1000 in your analysis
```

Step 1: DS analysis

```
> # load exon read count data
> RCS <- loadExonCountData(case.files, control.files)
> # remove genes with low expression
> RCS <- exonTestability(RCS, cutoff=5)
> geneTestable <- geneTestability(RCS)
> RCS <- subsetByGenes(RCS, unique(geneID(RCS))[ geneTestable ])
> # get gene IDs, which will be used in initialization of gene set
> geneIDs <- unique(geneID(RCS))
> # calculate DS NB statistics
> RCS <- estiExonNBstat(RCS)
> RCS <- estiGeneNBstat(RCS)
> # calculate DS NB statistics on the permutation data sets
> permuteMat <- genpermuteMat(RCS, times=perm.times)
> RCS <- DSpermute4GSEA(RCS, permuteMat)
```

Step 2: DE analysis

```
> # get gene read counts
> geneCounts <- getGeneCount(RCS)
> # calculate DE NB statistics
> label <- label(RCS)
> DEG <-runDESeq(geneCounts, label)
> DEGres <- DENBStat4GSEA(DEG)
> # calculate DE NB statistics on the permutation data sets
> DEpermNBstat <- DENBStatPermut4GSEA(DEG, permuteMat) # permutation
```

Step 3: score integration

```
> # DE score normalization
> DEscore.normFac <- normFactor(DEpermNBstat)
> DEscore <- scoreNormalization(DEGres$NBstat, DEscore.normFac)
> DEscore.perm <- scoreNormalization(DEpermNBstat, DEscore.normFac)
> # DS score normalization
> DSscore.normFac <- normFactor(RCS@permute_NBstat_gene)
> DSscore <- scoreNormalization(RCS@featureData_gene$NBstat, DSscore.normFac)
> DSscore.perm <- scoreNormalization(RCS@permute_NBstat_gene, DSscore.normFac)
> # score integration
> gene.score <- geneScore(DEscore, DSscore, DEweight=0.5)
> gene.score.perm <- genePermuteScore(DEscore.perm, DSscore.perm, DEweight=0.5)
> # visilization of scores
> # NOT run in the example; users to uncomment the following 6 lines to run
> #plotGeneScore(DEscore, DEscore.perm, pdf=paste(output.prefix,".DEScore.pdf",sep=""),
> #              main="Expression")
> #plotGeneScore(DSscore, DSscore.perm, pdf=paste(output.prefix,".DSScore.pdf",sep=""),
```

```
> #              main="Splicing")
> #plotGeneScore(gene.score, gene.score.perm,
> #              pdf=paste(output.prefix,".GeneScore.pdf",sep=""))
```

Step 4: main GSEA

```
> # load gene set data
> gene.set <- loadGenesets(geneset.file, geneIDs, geneID.type="ensembl",
+                          genesetsize.min = 5, genesetsize.max = 1000)
> # enrichment analysis
> gene.set <- GSEnrichAnalyze(gene.set, gene.score, gene.score.perm, weighted.type=1)
> # format enrichment analysis results
> GSEAres <- GSEAresultTable(gene.set, TRUE)
> # output results
> # NOT run in the example; users to uncomment the following 4 lines to run
> #write.table(GSEAres, paste(output.prefix,".GSEA.result.txt",sep=""),
> #            quote=FALSE, sep="\t", row.names=FALSE)
> #plotES(gene.set, pdf=paste(output.prefix,".GSEA.ES.pdf",sep=""))
> #plotSig(gene.set, pdf=paste(output.prefix,".GSEA.FDR.pdf",sep=""))
```

For gene sets used in Step 4, while we recommend users directly download and use those already well-formatted gene sets from MSigDB (`http://www.broadinstitute.org/gsea/msigdb/index.jsp`), users can also feed whatever gene sets to SeqGSEA as long as they are in the GMT format. Please refer to the following URL for details: `http://www.broadinstitute.org/cancer/software/gsea/wiki/index.php/Data_formats`.

6.3 Exemplified pipeline for DE-only analysis

For the demanding of DE-only analysis, such as for organisms without much alternative splicing annotated, here we show an exemplified pipeline for such analysis. It includes 4 steps as follows.

Step 0: Initialization. (Users should change values in this part accordingly.)

```
> rm(list=ls())
> # input count data files
> data.dir <- system.file("extdata", package="SeqGSEA", mustWork=TRUE)
> count.file <- paste(data.dir,"geneCounts.txt",sep="/")
> # gene set file
> geneset.file <- system.file("extdata", "gs_symb.txt",
+                            package="SeqGSEA", mustWork=TRUE)
> # output file prefix
> output.prefix <- "SeqGSEA.test"
> # setup parallel backend
> library(doParallel)
> cl <- makeCluster(2) # specify 2 cores to be used in computing
> registerDoParallel(cl) # parallel backend registration
> # setup permutation times
> perm.times <- 20 # change the number to >= 1000 in your analysis
```

Step 1: DE analysis

```
> # load gene read count data
> geneCounts <- read.table(count.file)
> # speficify the labels of each sample
> label <- as.factor(c(rep(1,10), rep(0,10)))
```

```
> # calculate DE NB statistics
> DEG <-runDESeq(geneCounts, label)
> DEGres <- DENBStat4GSEA(DEG)
> # calculate DE NB statistics on the permutation data sets
> permuteMat <- genpermuteMat(label, times=perm.times)
> DEpermNBstat <- DENBStatPermut4GSEA(DEG, permuteMat) # permutation
```

Step 2: score normalization

```
> # DE score normalization
> DEscore.normFac <- normFactor(DEpermNBstat)
> DEscore <- scoreNormalization(DEGres$NBstat, DEscore.normFac)
> DEscore.perm <- scoreNormalization(DEpermNBstat, DEscore.normFac)
> # score integration - DSscore can be null
> gene.score <- geneScore(DEscore, DEweight=1)
> gene.score.perm <- genePermuteScore(DEscore.perm, DEweight=1)  # visilization of scores
> # NOT run in the example; users to uncomment the following 6 lines to run
> #plotGeneScore(DEscore, DEscore.perm, pdf=paste(output.prefix,".DEScore.pdf",sep=""),
> #              main="Expression")
> #plotGeneScore(gene.score, gene.score.perm,
> #              pdf=paste(output.prefix,".GeneScore.pdf",sep=""))
```

Step 3: main GSEA

```
> # load gene set data
> geneIDs <- rownames(geneCounts)
> gene.set <- loadGenesets(geneset.file, geneIDs, geneID.type="ensembl",
+                          genesetsize.min = 5, genesetsize.max = 1000)
> # enrichment analysis
> gene.set <- GSEnrichAnalyze(gene.set, gene.score, gene.score.perm, weighted.type=1)
> # format enrichment analysis results
> GSEAres <- GSEAresultTable(gene.set, TRUE)
> # output results
> # NOT run in the example; users to uncomment the following 4 lines to run
> #write.table(GSEAres, paste(output.prefix,".GSEA.result.txt",sep=""),
> #            quote=FALSE, sep="\t", row.names=FALSE)
> #plotES(gene.set, pdf=paste(output.prefix,".GSEA.ES.pdf",sep=""))
> #plotSig(gene.set, pdf=paste(output.prefix,".GSEA.FDR.pdf",sep=""))
```

6.4 One-step SeqGSEA analysis

While users can choose to run SeqGSEA step by step in a well-controlled manner (see above), the one-step SeqGSEA analysis with an all-in `runSeqGSEA` function enables users to run SeqGSEA in the easiest way. With the `runSeqGSEA` function, users can also test multiple weights for integrating DE and DS scores. DE-only analysis starting with exon read counts is also supported in the all-in function.

Follow the example below to start your first SeqGSEA analysis now!

```
> ### Initialization ###
> # input file location and pattern
> data.dir <- system.file("extdata", package="SeqGSEA", mustWork=TRUE)
> case.pattern <- "^SC" # file name starting with "SC"
> ctrl.pattern <- "^SN" # file name starting with "SN"
> # gene set file and type
> geneset.file <- system.file("extdata", "gs_symb.txt",
```

```
+                             package="SeqGSEA", mustWork=TRUE)
> geneID.type <- "ensembl"
> # output file prefix
> output.prefix <- "SeqGSEA.example"
> # analysis parameters
> nCores <- 8
> perm.times <- 1000 # >= 1000 recommended
> DEonly <- FALSE
> DEweight <- c(0.2, 0.5, 0.8)  # a vector for different weights
> integrationMethod <- "linear"
>
> ### one step SeqGSEA running ###
> # NOT run in the example; uncomment the following 4 lines to run
> # CAUTION: running the following lines will generate lots of files in your working dir
> #runSeqGSEA(data.dir=data.dir, case.pattern=case.pattern, ctrl.pattern=ctrl.pattern,
> #           geneset.file=geneset.file, geneID.type=geneID.type, output.prefix=output.prefix,
> #           nCores=nCores, perm.times=perm.times, integrationMethod=integrationMethod,
            DEonly=DEonly, DEweight=DEweight)
```

7 Session information

```
> sessionInfo()

R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] SeqGSEA_1.14.0      DESeq_1.26.0        lattice_0.20-34
[4] locfit_1.5-9.1      doParallel_1.0.10   iterators_1.0.8
[7] foreach_1.4.3       Biobase_2.34.0      BiocGenerics_0.20.0

loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.36.0 splines_3.3.1        IRanges_2.8.0
 [4] xtable_1.8-2         tools_3.3.1          grid_3.3.1
 [7] DBI_0.5-1            genefilter_1.56.0    survival_2.39-5
[10] Matrix_1.2-7.1       RColorBrewer_1.1-2   geneplotter_1.52.0
[13] S4Vectors_0.12.0     bitops_1.0-6         codetools_0.2-15
[16] biomaRt_2.30.0       RCurl_1.95-4.8       RSQLite_1.0.0
[19] compiler_3.3.1       stats4_3.3.1         XML_3.98-1.4
[22] annotate_1.52.0
```

Cleanup

This is a cleanup step for the vignette on Windows; typically not needed for users.

```
> allCon <- showConnections()
> socketCon <- as.integer(rownames(allCon)[allCon[, "class"] == "sockconn"])
> sapply(socketCon, function(ii) close.connection(getConnection(ii)) )
```