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

Xi Wang\textsuperscript{1,2} and Murray Cairns\textsuperscript{1,2,3}

October 24, 2023

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

xi.wang@mdc-berlin.de

1 Introduction

1.1 Background ................................................................. 2
1.2 Getting started .............................................................. 2
1.3 Package citation ............................................................. 2

2 Differential splicing analysis and DS scores

2.1 The *ReadCountSet* class ................................................. 2
2.2 DS analysis and DS scores .................................................. 3
2.3 DS permutation p-values ................................................... 5

3 Differential expression analysis and DE scores

3.1 Gene read count data from *ReadCountSet* class ....................... 6
3.2 DE analysis and DE scores ................................................ 6
3.3 DE permutation p-values .................................................. 7

4 Integrative GSEA runs

4.1 DE/DS score integration .................................................. 8
4.2 Initialization of *SeqGeneSet* objects ................................... 10
4.3 running GSEA with integrated gene scores ............................... 11
4.4 *SeqGSEA* result displays .............................................. 11

5 Running *SeqGSEA* with multiple cores

5.1 R-parallel packages ....................................................... 13
5.2 Parallelizing analysis on permutation data sets .......................... 14

6 Analysis examples

6.1 Starting from your own RNA-Seq data .................................. 14
6.2 Exemplified pipeline for integrating DE and DS .......................... 15
6.3 Exemplified pipeline for DE-only analysis .............................. 17
6.4 One-step *SeqGSEA* analysis .......................................... 18

7 Session information ................................................................ 19
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 a 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 Wang and Cairns (2013) and Wang and Cairns (2014) 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, DESeq2 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 6, and detailed guides for DE, DS, and integrative GSEA analysis are given in Sections 3, 2, and 4, respectively. A guide to parallelize those analyses is given in Section 5.

1.3 Package citation

To cite this package, please cite the article below:

To cite/discuss the method used in this package, please cite the article below:

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 6.
> 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 ... ENSG000000007402:038 (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 (Kannan et al., 2011). 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 `exonTestability` 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`.

```r
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 Wang et al. (2013) for detailed statistic analysis regarding differential splicing from exon count data.

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

```
<table>
<thead>
<tr>
<th>exonIDs</th>
<th>geneIDs</th>
<th>testable</th>
<th>NBstat</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSG00000000003:001</td>
<td>E001</td>
<td>ENSG00000000003</td>
<td>TRUE</td>
</tr>
<tr>
<td>ENSG00000000003:002</td>
<td>E002</td>
<td>ENSG00000000003</td>
<td>TRUE</td>
</tr>
<tr>
<td>ENSG00000000003:003</td>
<td>E003</td>
<td>ENSG00000000003</td>
<td>TRUE</td>
</tr>
<tr>
<td>ENSG00000000003:004</td>
<td>E004</td>
<td>ENSG00000000003</td>
<td>TRUE</td>
</tr>
<tr>
<td>ENSG00000000003:005</td>
<td>E005</td>
<td>ENSG00000000003</td>
<td>TRUE</td>
</tr>
<tr>
<td>ENSG00000000003:006</td>
<td>E006</td>
<td>ENSG00000000003</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
```

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.

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

```
<table>
<thead>
<tr>
<th>result.1</th>
<th>result.2</th>
<th>result.3</th>
<th>result.4</th>
<th>result.5</th>
<th>result.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSG00000000003:001</td>
<td>1.2079557</td>
<td>0.5570066</td>
<td>0.4402005</td>
<td>1.3517893</td>
<td>0.7693572</td>
</tr>
<tr>
<td>ENSG00000000003:002</td>
<td>0.5161164</td>
<td>0.3955556</td>
<td>0.4200555</td>
<td>0.3246275</td>
<td>0.4231841</td>
</tr>
<tr>
<td>ENSG00000000003:003</td>
<td>1.7563215</td>
<td>0.4803296</td>
<td>0.8376798</td>
<td>0.9492975</td>
<td>0.7967375</td>
</tr>
<tr>
<td>ENSG00000000003:004</td>
<td>0.4690467</td>
<td>0.6868281</td>
<td>0.6487009</td>
<td>1.0135191</td>
<td>0.6430321</td>
</tr>
<tr>
<td>ENSG00000000003:005</td>
<td>0.8511979</td>
<td>0.6261922</td>
<td>0.5182736</td>
<td>1.3106612</td>
<td>0.3963787</td>
</tr>
</tbody>
</table>
```

4
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 4).

```r
> 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 1.7255445 0.8382363 2.1679863 3.2707076 0.9690078

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

result.1 result.2 result.3 result.4 result.5 result.6 result.7 result.8 result.9 result.10
ENSG00000000003 1.4458762 0.6667154 0.5269030 1.618039 0.9208907 2.1682966
ENSG00000000005 0.9477930 0.7263959 0.7841503 0.730351 0.5961439 0.7771327
ENSG00000000019 0.9723044 0.6184000 1.0784704 1.222173 1.0267593 2.0267335
ENSG000000000419 0.7296848 0.7721221 0.7292600 1.139383 0.6104687 1.2874863
ENSG000000000460 1.0981724 0.8079895 0.6686464 1.690984 0.5154743 0.8006381
```

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.

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

geneID NBstat pvalue padjust
1  ENSG00000000003 1.4416043 0.15 0.2112000
2  ENSG00000000005 0.4564578 0.90 0.5629733
3  ENSG00000000019 1.6839390 0.00 0.0000000
4  ENSG000000000419 1.2637404 0.00 0.0000000
5  ENSG000000000460 0.7510661 0.90 0.5629733
6  ENSG000000000938 1.6676791 1.5500467
```

The adjusted p-values accounting for multiple tests were given by the BH method (Benjamini and Hochberg, 1995). 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.

```r
> geneCounts <- getGeneCount(RCS_example)
> dim(geneCounts) # 182 20
[1] 182 20
```

```r
> 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 156 134 166 184 132 364 343 307 280 179 156 100 120 126
ENSG00000000467  97  7  8  1  1  19  74 304 347 249  62  40  43  52
ENSG00000000460  52 35  7 12  7  47 105 124  80 156 145  48  36  21  34 19
ENSG00000000938  27 44  7 13  4  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  26 171 164  201
ENSG00000000467 170 165 131 183 185
ENSG00000000460  68  90  48 72  38
ENSG00000000938 103 1286 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 DESeq2 (Anders and Huber, 2010) is mainly utilized in SeqGSEA for DE analysis. With DESeq2, 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 DESeq2 including size factor estimation and dispersion estimation, is encapsulated in the function runDESeq.

```r
> label <- label(RCS_example)
> dds <- runDESeq(geneCounts, label)
```

The function runDESeq returns a DESeqDataSet object, which is defined in the DESeq2 package. The DE analysis in the DESeq2 package continues with the output DESeqDataSet 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.

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

```
0.51449518  0.32697507  0.02206611 13.22540371  1.24725584
```

6
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.

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

result.1 result.2 result.3 result.4 result.5 result.6 result.7 result.8 result.9 result.10
[1,] 0.1127507 0.01642736 0.3483761 1.4015411 0.2698296 2.67376271 0.02698876
[2,] 2.5915185 2.13647340 0.2865709 4.4274418 5.1725082 0.06830376 0.13664817
[3,] 0.0213238 0.13972657 0.9917461 0.2656078 0.2450678 4.37588504 0.03773060
[4,] 0.4534209 0.04154632 0.9339290 1.0678160 0.0200093 0.89788526 0.0171969
[5,] 0.1035362 0.5663182 2.6642083 1.1210781 2.2143580 1.56841992 1.05018798
```

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 4).

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

[1] 0.6410503 0.2285285 0.0278913 25.19098498 1.17883758
```

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

result.1 result.2 result.3 result.4 result.5 result.6 result.7 result.8 result.9 result.10
[1,] 0.14048500 0.02046815 0.4340694 1.7462911 0.33620202 3.33145271 0.03362743
[2,] 1.8125646 1.49321771 0.2002893 3.0944146 3.6154525 0.04773866 0.0955073
[3,] 0.02696527 0.17669294 1.2541246 0.3358775 0.30990349 5.53876858 0.04771266
[4,] 0.8634985 0.7913563 1.7788941 2.0339143 0.03812551 1.71386520 0.03272269
[5,] 0.0978669 0.3534921 2.5180630 1.0598613 2.09288930 1.48238420 0.99257989
```

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.

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

<table>
<thead>
<tr>
<th>NBstat</th>
<th>perm.pval</th>
<th>perm.padj</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSG00000000003</td>
<td>0.51449518</td>
<td>0.501</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th></th>
<th>NBstat</th>
<th>pval</th>
<th>padj</th>
<th>perm.pval</th>
<th>perm.padj</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENSG00000000003</td>
<td>0.51449518</td>
<td>0.1112841241</td>
<td>0.24121931</td>
<td>0.50</td>
<td>1</td>
</tr>
<tr>
<td>ENSG00000000005</td>
<td>0.32697507</td>
<td>0.2310245778</td>
<td>0.38574746</td>
<td>0.40</td>
<td>1</td>
</tr>
<tr>
<td>ENSG000000000005</td>
<td>0.02205611</td>
<td>0.3643336232</td>
<td>0.52272077</td>
<td>0.85</td>
<td>1</td>
</tr>
<tr>
<td>ENSG0000000000419</td>
<td>13.22540371</td>
<td>0.0008268416</td>
<td>0.01588092</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ENSG0000000000460</td>
<td>1.24725584</td>
<td>0.0816750168</td>
<td>0.21235504</td>
<td>0.30</td>
<td>1</td>
</tr>
<tr>
<td>ENSG0000000000938</td>
<td>7.73813817</td>
<td>0.0006694334</td>
<td>0.01588092</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>
```

4 Integrative GSEA runs

4.1 DE/DS score integration

We have proposed two strategies for integrating normalized DE and DS scores (Wang and Cairns, 2013), 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 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 ranges from 0 (i.e., DS only) to 1 (i.e., DE only). Visualization of gene scores can be made by applying the `plotGeneScore` function.

```r
> 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. 1) 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. 2.

```r
> 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. 3.
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.

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.
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.

```r
> combine <- rankCombine(DEscore, DSscore, DEscore.perm, DSscore.perm, DEweight=0.3)
> gene.score <- combine$geneScore
> gene.score.perm <- combine$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. 1, 2, and 3); however, according to the analysis in Wang and Cairns (2013), 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](http://www.broadinstitute.org/gsea/msigdb/index.jsp)) (Subramanian et al., 2005). 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.

```r
> data(GS_example, package="SeqGSEA")
> GS_example
```
SeqGeneSet object: gs_symb.txt
GeneSetSourceFile: /Library/Frameworks/R.framework/Versions/2.15/Resources/library/SeqGSEA/extdata/gs_symb.txt
GeneSets: ERB2.UP.V1.DN
    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.

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

<table>
<thead>
<tr>
<th>GSName</th>
<th>GSSize</th>
<th>ES</th>
<th>ES.pos</th>
<th>pval</th>
<th>FDR</th>
<th>FWER</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>1.769722</td>
<td>41</td>
<td>0.05</td>
<td>0.00000</td>
<td>0.25</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>1.897786</td>
<td>33</td>
<td>0.00</td>
<td>0.00000</td>
<td>0.05</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.566208</td>
<td>57</td>
<td>0.05</td>
<td>0.33333</td>
<td>0.65</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>1.371925</td>
<td>35</td>
<td>0.20</td>
<td>0.50000</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1.179402</td>
<td>37</td>
<td>0.25</td>
<td>0.80000</td>
<td>1.00</td>
</tr>
</tbody>
</table>

The main GSEA includes several steps detailed in Wang and Cairns (2013) and its original paper Subramanian et al. (2005). 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. 4).

```r
> 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.
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.

> 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. 5 for an example.

> 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 Wang and Cairns (2013)). 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:
genename gs_symb.txt:TBK1.DF_UP
gensetSize 5
genDesc http://www.broadinstitute.org/gsea/msigdb/cards/TBK1.DF_UP
NES 1.89778620725375
Pos 33
pvalue 0
FDR 0
FWER 0.05

Leading set:
ENSG00000002919 6.47006365396346
ENSG00000005194 6.26880890936921
ENSG00000001167 4.81782694513746
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.

Whole gene set:
ENSG00000002919 6.47006365396346
ENSG00000005194 6.26880890936921
ENSG00000001167 4.81782694513746
ENSG00000006576 1.39964571133245
ENSG00000005059 1.12835495156424

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 6.

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

```r
> library(doParallel)
> a <- matrix(1:16, 4, 4)
```
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

```r
library(doParallel)
c1 <- makeCluster(2) # specify 2 cores to be used in this computing
registerDoParallel(c1)
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,] 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` vignette ([http://cran.r-project.org/web/packages/doParallel/index.html](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. These 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).

### 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 gene 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/](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.
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 (Kannan et al., 2011). We divide the process into five steps for a complete SeqGSEA run.

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

```r
> 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

```r
> # 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)
> dds <- runDESeq(geneCounts, label)
> DEGres <- DENBStat4GSEA(dds)
> # calculate DE NB statistics on the permutation data sets
> DEpermNBstat <- DENBStatPermut4GSEA(dds, permuteMat) # permutation

**Step 3: score integration**

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

```r
> 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

```r
> # load gene read count data
> geneCounts <- read.table(count.file)
> # specify the labels of each sample
> label <- as.factor(c(rep(1,10), rep(0,10)))
> # calculate DE NB statistics
> dds <-runDESeq(geneCounts, label)
> DEGres <- DENBStat4GSEA(dds)
> # calculate DE NB statistics on the permutation data sets
> permuteMat <- genpermuteMat(label, times=perm.times)
> DEpermNBstat <- DENBStatPermut4GSEA(dds, permuteMat) # permutation
```

Step 2: score normalization

```r
> # 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) # visualization of scores
> # NUT 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 4.3.1 (2023-06-16)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.18-bioc/R/lib/libRblas.so
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0

locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
[3] LC_TIME=en_GB LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8 LC_NAME=C
[9] LC_ADDRESS=C LC_TELEPHONE=C

time zone: America/New_York
tzcode source: system (glibc)

attached base packages:
[1] stats4 parallel stats graphics grDevices utils datasets
[8] methods base

other attached packages:
[1] SeqGSEA_1.42.0 DESeq2_1.42.0
[3] SummarizedExperiment_1.32.0 MatrixGenerics_1.14.0
[5] matrixStats_1.0.0 GenomicRanges_1.54.0
[7] GenomeInfoDb_1.38.0 IRanges_2.36.0
[9] S4Vectors_0.40.0 doParallel_1.0.17
[11] iterators_1.0.14 foreach_1.5.2
[13] Biobase_2.62.0 BiocGenerics_0.48.0

loaded via a namespace (and not attached):
[1] RGGREST_1.42.0 gtable_0.3.4 ggplot2_3.4.4
[4] lattice_0.22-5 vctrs_0.6.4 tools_4.3.1
[7] bitops_1.0-7 generics_0.1.3 curl_5.1.0
[10] tibble_3.2.1 fansi_1.0.5 AnnotationDbi_1.64.0
[13] RSQLite_2.3.1 blob_1.2.4 pkgconfig_2.0.3
[16] Matrix_1.6-1.1 dbplyr_2.3.4 lifecycle_1.0.3
[19] GenomeInfoDbData_1.2.11 stringr_1.5.0 compiler_4.3.1
[22] progress_1.2.2 Biostrings_2.70.0 munsell_0.5.0
[25] codetools_0.2-19 RCurl_1.98-1.12 pillar_1.9.0
[28] crayon_1.5.2 BiocParallel_1.36.0 DelayedArray_0.28.0
[31] cachem_1.0.8 abind_1.4-5 digest_0.6.33
[34] tidyselect_1.2.0 locfit_1.5-9.8 stringi_1.7.12
[37] purrr_1.0.2 dplyr_1.1.3 biomaRt_2.58.0
[40] fastmap_1.1.1 grid_4.3.1 colorspace_2.1-0
[43] cli_3.6.1 SparseArray_1.2.0 magrittr_2.0.3
[46] S4Arrays_1.2.0 XML_3.99-0.14 utf8_1.2.4
This is a clean up step for the vignette on Windows; typically not needed for users.

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

**References**


