goseq: Gene Ontology testing for RNA-seq datasets

Matthew D. Young Nadia Davidson Matthew J. Wakefield
Gordon K. Smyth Alicia Oshlack

30 March 2016

1 Introduction

This document gives an introduction to the use of the goseq R Bioconductor package (?). This package provides methods for performing Gene Ontology analysis of RNA-seq data, taking length bias into account (?). The methods and software used by goseq are equally applicable to other category based test of RNA-seq data, such as KEGG pathway analysis.

Once installed, the goseq package can be easily loaded into R using:

```
> library(goseq)
```

In order to perform a GO analysis of your RNA-seq data, goseq only requires a simple named vector, which contains two pieces of information.

1. `Measured genes`: all genes for which RNA-seq data was gathered for your experiment. Each element of your vector should be named by a unique gene identifier.

2. `Differentially expressed genes`: each element of your vector should be either a 1 or a 0, where 1 indicates that the gene is differentially expressed and 0 that it is not.

If the organism, gene identifier or category test is currently not natively supported by goseq, it will also be necessary to supply additional information regarding the genes length and/or the association between categories and genes.

Bioconductor R packages such as Rsubread allow for the summarization of mapped reads into a table of counts, such as reads per gene. From there, several packages exist for performing differential expression analysis on summarized data (eg. edgeR (???)). goseq will work with any method for determining differential expression and as such differential expression analysis is outside the scope of this document, but in order to facilitate ease of use, we will make use of the edgeR package to calculate differentially expressed (DE) genes in all the case studies in this document.

2 Reading data

We assume that the user can use appropriate in-built `R` functions (such as `read.table` or `scan`) to obtain two vectors, one containing all genes assayed in the RNA-seq experiment, the other containing all genes which are DE. If we assume that the vector of genes being assayed is named `assayed.genes` and the vector of DE genes is named `de.genes` we can construct a named vector suitable for use with `goseq` using the following:

```
> gene.vector=as.integer(assayed.genes%in%de.genes)
> names(gene.vector)=assayed.genes
> head(gene.vector)
```

It may be that the user can already read in a vector in this format, in which case it can then be immediately used by `goseq`.

3 GO testing of RNA-seq data

To begin the analysis, `goseq` first needs to quantify the length bias present in the dataset under consideration. This is done by calculating a Probability Weighting Function or PWF which can be thought of as a function which gives the probability that a gene will be differentially expressed (DE), based on its length alone. The PWF is calculated by fitting a monotonic spline to the binary data series of differential expression (1=DE, 0=Not DE) as a function of gene length. The PWF is used to weight the chance of selecting each gene when forming a null distribution for GO category membership. The fact that the PWF is calculated directly from the dataset under consideration makes this approach robust, only correcting for the length bias present in the data. For example, if `goseq` is run on a microarray dataset, for which no length bias exists, the calculated PWF will be nearly flat and all genes will be weighted equally, resulting in no length bias correction.

In order to account for the length bias inherent to RNA-seq data when performing a GO analysis (or other category based tests), one cannot simply use the hypergeometric distribution as the null distribution for category membership, which is appropriate for data without DE length bias, such as microarray data. GO analysis of RNA-seq data requires the use of random sampling in order to generate a suitable null distribution for GO category membership and calculate each categories significance for over representation amongst DE genes.

However, this random sampling is computationally expensive. In most cases, the Wallenius distribution can be used to approximate the true null distribution, without any significant loss in accuracy. The `goseq` package implements this approximation as its default option. The option to generate the null distribution using random sampling is also included as an option, but users should be aware that the default number of samples generated will not be enough to accurately call enrichment when there are a large number of go terms.

Having established a null distribution, each GO category is then tested for over and under representation amongst the set of differentially expressed genes and the null is used to calculate a p-value for under and over representation.

4 Natively supported Gene Identifiers and category tests

goseq needs to know the length of each gene, as well as what GO categories (or other categories of interest) each gene is associated with. goseq relies on the UCSC genome browser to provide the length information for each gene. However, because the process of fetching the length of every transcript is slow and bandwidth intensive, goseq relies on an offline copy of this information stored in the data package geneLenDataBase. To see which genome/gene identifier combinations are in the local database, simply run:

```
> supportedOrganisms()
```

The leftmost columns in the output of this command list the genomes and gene identifiers respectively. If length data exists in the local database it is indicated in the second last column. If your genome/ID combination is not in the local database, it may be downloaded from the UCSC genome browser or taken from a TxDb annotation package (if installed). If your genome/ID combination is not found in any database, you will have to manually specify the gene lengths. We encourage all users to manually specify their gene lengths if provided by upstream summarization programs. e.g. featureCounts, as these lengths will be more accurate.

In order to link GO categories to genes, goseq uses the organism packages from Bioconductor. These packages are named org.<Genome>.<ID>.db, where <Genome> is a short string identifying the genome and <ID> is a short string identifying the gene identifier. Currently, goseq will automatically retrieve the mapping between GO categories and genes from the relevant package (as long as it is installed) for commonly used genome/ID combinations. If GO mappings are not automatically available for your genome/ID combination, you will have to manually specify the relationship between genes and categories. Although the Genome/ID naming conventions used by the organism packages differ from the UCSC, goseq is able to convert between the two, so the user need only ever specify the UCSC genome/ID in most cases. The final column indicates whether the Genome/ID combination is supported for GO categories.

5 Non-native Gene Identifier or category test

If the organism, Gene Identifier or category test you wish to perform is not in the native goseq database, you will have to supply one or all of the following:

- Length data: the length of each gene in your gene identifier format.

- `Category mappings`: the mapping (usually many-to-many) between the categories you wish to test for over/under representation amongst DE genes and genes in your gene identifier format.

5.1 Length data format

The length data must be formatted as a numeric vector, of the same length as the main named vector specifying gene names/DE genes. Each entry should give the length of the corresponding gene in bp. If length data is unavailable for some genes, that entry should be set to NA.

5.2 Category mapping format

The mapping between category names and genes should be given as a data frame with two columns. One column should contain the gene IDs and the other the name of an associated category. As the mapping between categories and genes is usually many-to-many, this data frame will usually have multiple rows with the same gene name and category name.

Alternatively, mappings between genes and categories can be given as a list. The names of list entries should be gene IDs and the entries themselves should be a vector of category names to which the gene ID corresponds.

5.3 Some additional tips

Any organism for which there is an annotation on either Ensembl or the UCSC, can be easily turned into length data using the GenomicFeatures package. To do this, first create a TranscriptDb object using either makeTxDbFromBiomart or makeTxDbFromUCSC (see the help in the GenomicFeatures package on using these commands). Once you have a transcriptDb object, you can get a vector named by gene ID containing the median transcript length of each gene simply by using the command.

```
> txsByGene=transcriptsBy(txdb,"gene")
> lengthData=median(width(txsByGene))
```

The relationship between gene identifier and GO category can usually be obtained from the Gene Ontology website (www.geneontology.org) or from the NCBI. Additionally, the bioconductor AnnotationDbi library has recently added a function "makeOrgPackageFromNCBI", which can be used to create an organism package from within R, using the NCBI data. Once created, this package can then be used to obtain the mapping between genes and gene ontology.

6 Case study: Prostate cancer data

6.1 Introduction

This section provides an analysis of data from an RNA-seq experiment to illustrate the use of `goseq` for GO analysis.

This experiment examined the effects of androgen stimulation on a human prostate cancer cell line, LNCaP. The data set includes more than 17 million short cDNA reads obtained for both the treated and untreated cell line and sequenced on Illumina's 1G genome analyzer.

For each sample we were provided with the raw 35 bp RNA-seq reads from the authors. For the untreated prostate cancer cells (LNCaP cell line) there were 4 lanes totaling 10 million, 35 bp reads. For the treated cells there were 3 lanes totaling 7 million, 35 bp reads. All replicates were technical replicates. Reads were mapped to NCBI version 36.3 of the human genome using bowtie. Any read with which mapped to multiple locations was discarded. Using the ENSEMBL 54 annotation from biomart, each mapped read was associated with an ENSEMBL gene. This was done by associating any read that overlapped with any part of the gene (not just the exons) with that gene. Reads that did not correspond to genes were discarded.

6.2 Source of the data

The data set used in this case study is taken from (**?**) and was made available from the authors upon request.

6.3 Determining the DE genes using `edgeR`

To begin with, we load in the text data and convert it the appropriate `edgeR` DGEList object.

```
> library(edgeR)
> table.summary=read.table(system.file("extdata","Li_sum.txt",package='goseq'),
+                          sep='\t',header=TRUE,stringsAsFactors=FALSE)
> counts=table.summary[,-1]
> rownames(counts)=table.summary[,1]
> grp=factor(rep(c("Control","Treated"),times=c(4,3)))
> summarized=DGEList(counts,lib.size=colSums(counts),group=grp)
```

Next, we use `edgeR` to estimate the biological dispersion and calculate differential expression using a negative binomial model.

```
> disp=estimateCommonDisp(summarized)
> disp$common.dispersion

[1] 0.05688364
```

```
> tested=exactTest(disp)
> topTags(tested)

Comparison of groups:  Treated-Control
                    logFC   logCPM       PValue          FDR
ENSG00000127954 11.557868 6.680748 2.574972e-80 1.274766e-75
ENSG00000151503  5.398963 8.499530 1.781732e-65 4.410322e-61
ENSG00000096060  4.897600 9.446705 7.983756e-60 1.317479e-55
ENSG00000091879  5.737627 6.282646 1.207655e-54 1.494654e-50
ENSG00000132437 -5.880436 7.951910 2.950042e-52 2.920896e-48
ENSG00000166451  4.564246 8.458467 7.126763e-52 5.880292e-48
ENSG00000131016  5.254737 6.607957 1.066807e-51 7.544766e-48
ENSG00000163492  7.085400 5.128514 2.716461e-45 1.681014e-41
ENSG00000113594  4.051053 8.603264 9.272066e-44 5.100255e-40
ENSG00000116285  4.108522 7.864773 6.422468e-43 3.179507e-39
```

Finally, we Format the DE genes into a vector suitable for use with `goseq`

```
> genes=as.integer(p.adjust(tested$table$PValue[tested$table$logFC!=0],
+                  method="BH")<.05)
> names(genes)=row.names(tested$table[tested$table$logFC!=0,])
> table(genes)

genes
    0     1
19535  3208
```

6.4 Determining Genome and Gene ID

In order to allow for automatic data retrieval, the user has to tell `goseq` what genome and gene ID format were used to summarize the data. In our case we will use the hg19 build of the human genome, we check what code this corresponds to by running:

```
> head(supportedOrganisms())

    Genome         Id  Id Description Lengths in geneLeneDataBase
12 anoCar1    ensGene Ensembl gene ID                         TRUE
13 anoGam1    ensGene Ensembl gene ID                         TRUE
14 apiMel2    ensGene Ensembl gene ID                         TRUE
56 bosTau2 geneSymbol     Gene Symbol                         TRUE
15 bosTau3    ensGene Ensembl gene ID                         TRUE
57 bosTau3 geneSymbol     Gene Symbol                         TRUE
   GO Annotation Available
```

```
12                    FALSE
13                     TRUE
14                    FALSE
56                     TRUE
15                     TRUE
57                     TRUE
```

Which lists the genome codes in the far left column, headed "Genome". As we are using "hg19" and we also know that we used ENSEMBL Gene ID to summarize our read data, we check what code this corresponds to by running:

```
> supportedOrganisms()[supportedOrganisms()$Genome=="hg19",]
```

```
   Genome          Id  Id Description Lengths in geneLeneDataBase
4    hg19  knownGene  Entrez Gene ID                         TRUE
36   hg19    ensGene Ensembl gene ID                         TRUE
81   hg19 geneSymbol     Gene Symbol                         TRUE
   GO Annotation Available
4                     TRUE
36                    TRUE
81                    TRUE
```

The gene ID codes are listed in the column second from left, titled "Id". We find that our gene ID code is "ensGene". We will use these strings whenever we are asked for a genome or id. If the gene ID is missing for your Genome (for example this is the case for hg38), then the genome is not supported in the geneLengthDatabase. Gene lengths will either be automatically fetched from TxDB, UCSC or you will need to provide them manually. Supported Gene IDs to automatically fetch GO terms should usually either be Entrez ("knownGene"), Ensembl ("ensGene") or gene symbols ("geneSymbol").

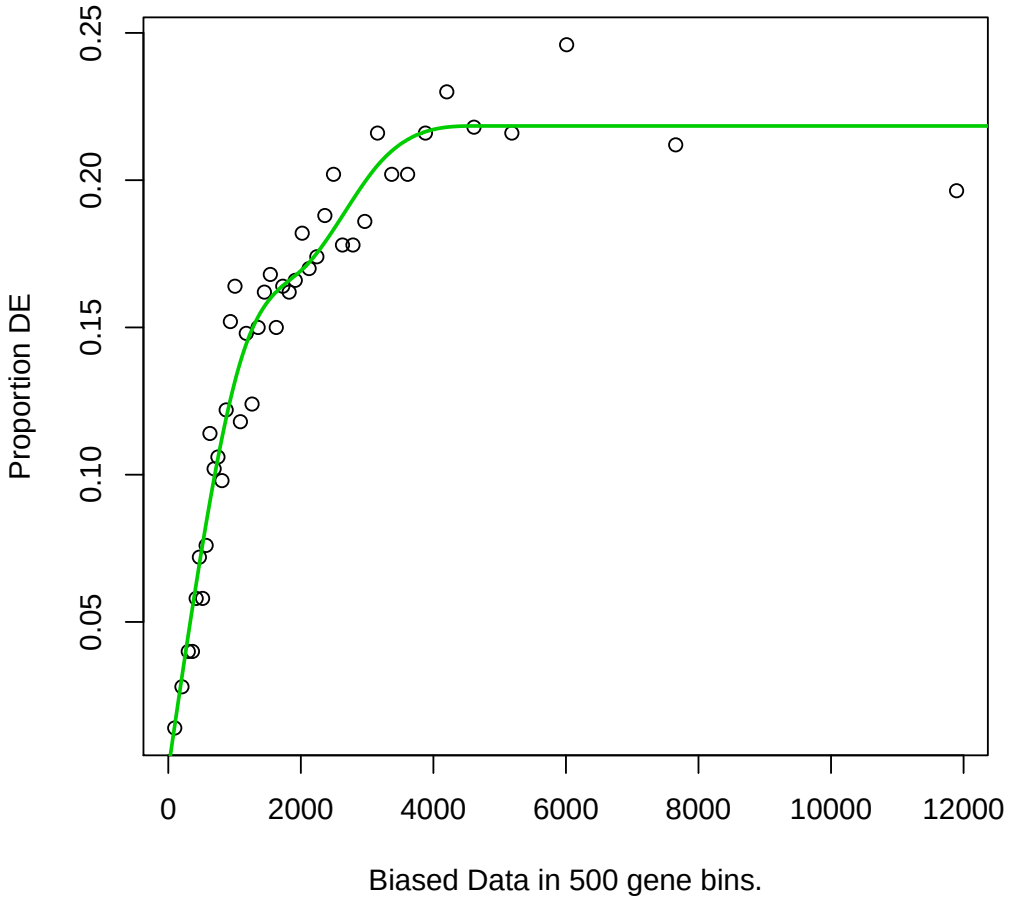
6.5 GO analysis

6.5.1 Fitting the Probability Weighting Function (PWF)

We first need to obtain a weighting for each gene, depending on its length, given by the PWF. As you may have noticed when running supportedGenomes or supportedGeneIDs, length data is available in the local database for our gene ID, "ensGene" and our genome, "hg19". We will let `goseq` automatically fetch this data from its databases.

```
> pwf=nullp(genes,"hg19","ensGene")
> head(pwf)
```

```
                DEgenes bias.data        pwf
ENSG00000230758       0       247 0.03757470
ENSG00000182463       0      3133 0.20436865
ENSG00000124208       0      1978 0.16881769
ENSG00000230753       0       466 0.06927243
ENSG00000224628       0      1510 0.15903532
ENSG00000125835       0       954 0.12711992
```



`nullp` plots the resulting fit, allowing verification of the goodness of fit before continuing the analysis. Further plotting of the pwf can be performed using the `plotPWF` function.

The output of nullp contains all the data used to create the PWF, as well as the PWF itself. It is a data frame with 3 columns, named "DEgenes", "bias.data" and "pwf" with the rownames set to the gene names. Each row corresponds to a gene with the DEgenes column specifying if the gene is DE (1 for DE, 0 for not DE), the bias.data column giving the numeric value of the DE bias

being accounted for (usually the gene length or number of counts) and the pwf column giving the genes value on the probability weighting function.

6.5.2 Using the Wallenius approximation

To start with we will use the default method, to calculate the over and under expressed GO categories among DE genes. Again, we allow `goseq` to fetch data automatically, except this time the data being fetched is the relationship between ENSEMBL gene IDs and GO categories.

```
> GO.wall=goseq(pwf,"hg19","ensGene")
> head(GO.wall)
```

```
        category over_represented_pvalue under_represented_pvalue numDEInCat
11078 GO:0044763            1.366974e-13                         1       1849
11050 GO:0044699            2.221801e-13                         1       2006
2525  GO:0005737            1.694844e-11                         1       1863
135   GO:0000278            1.180017e-08                         1        245
7698  GO:0031988            1.705328e-08                         1        628
10996 GO:0044444            2.011938e-08                         1       1431
      numInCat                              term ontology
11078     8182 single-organism cellular process       BP
11050     9000          single-organism process       BP
2525      8418                         cytoplasm       CC
135        865               mitotic cell cycle       BP
7698      2640         membrane-bounded vesicle       CC
10996     6443                 cytoplasmic part       CC
```

The resulting object is ordered by GO category over representation amongst DE genes.

6.5.3 Using random sampling

It may sometimes be desirable to use random sampling to generate the null distribution for category membership. For example, to check consistency against results from the Wallenius approximation. This is easily accomplished by using the `method` option to specify sampling and the `repcnt` option to specify the number of samples to generate:

```
> GO.samp=goseq(pwf,"hg19","ensGene",method="Sampling",repcnt=1000)
```

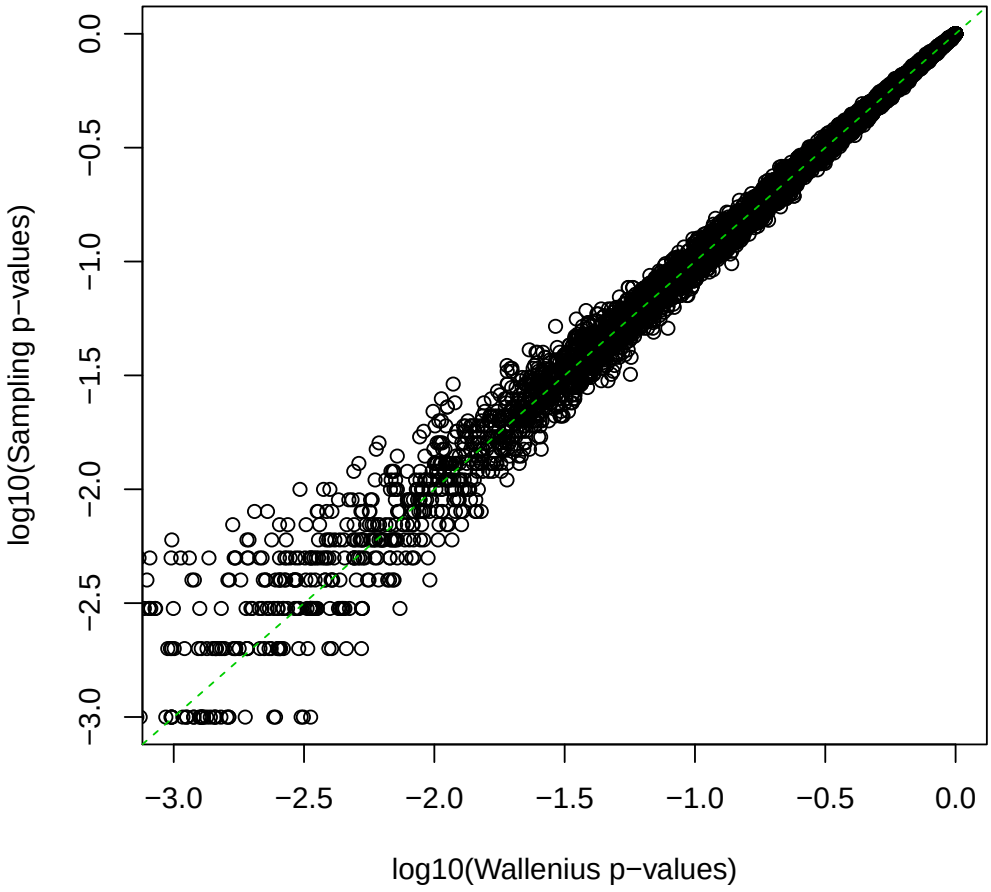
```
> head(GO.samp)
```

```
      category over_represented_pvalue under_represented_pvalue numDEInCat numInCat
93  GO:0000184            0.000999001                         1         32      118
110 GO:0000226            0.000999001                         1        105      363
135 GO:0000278            0.000999001                         1        245      865
```

```
136 GO:0000280             0.000999001                          1        138      479
272 GO:0000779             0.000999001                          1         34      106
285 GO:0000793             0.000999001                          1         56      184
                                                                                term ontology
93  nuclear-transcribed mRNA catabolic process, nonsense-mediated decay       BP
110                              microtubule cytoskeleton organization       BP
135                                                   mitotic cell cycle       BP
136                                                     nuclear division       BP
272                            condensed chromosome, centromeric region       CC
285                                                 condensed chromosome       CC
```

You will notice that this takes far longer than the Wallenius approximation. Plotting the p-values against one another, we see that there is little difference between the two methods. However, the accuracy of the sampling method is limited by the number of samples generated, `repcnt`, such that very low p-values will not be correctly calculated. Significantly enriched GO terms may then be missed after correcting for multiple testing.

```
> plot(log10(GO.wall[,2]), log10(GO.samp[match(GO.samp[,1],GO.wall[,1]),2]),
+      xlab="log10(Wallenius p-values)",ylab="log10(Sampling p-values)",
+      xlim=c(-3,0))
> abline(0,1,col=3,lty=2)
```



6.5.4 Ignoring length bias

`goseq` also allows for one to perform a GO analysis without correcting for RNA-seq length bias. In practice, this is only useful for assessing the effect of length bias on your results. You should NEVER use this option as your final analysis. If length bias is truly not present in your data, `goseq` will produce a nearly flat PWF and no length bias correction will be applied to your data and all methods will produce the same results.

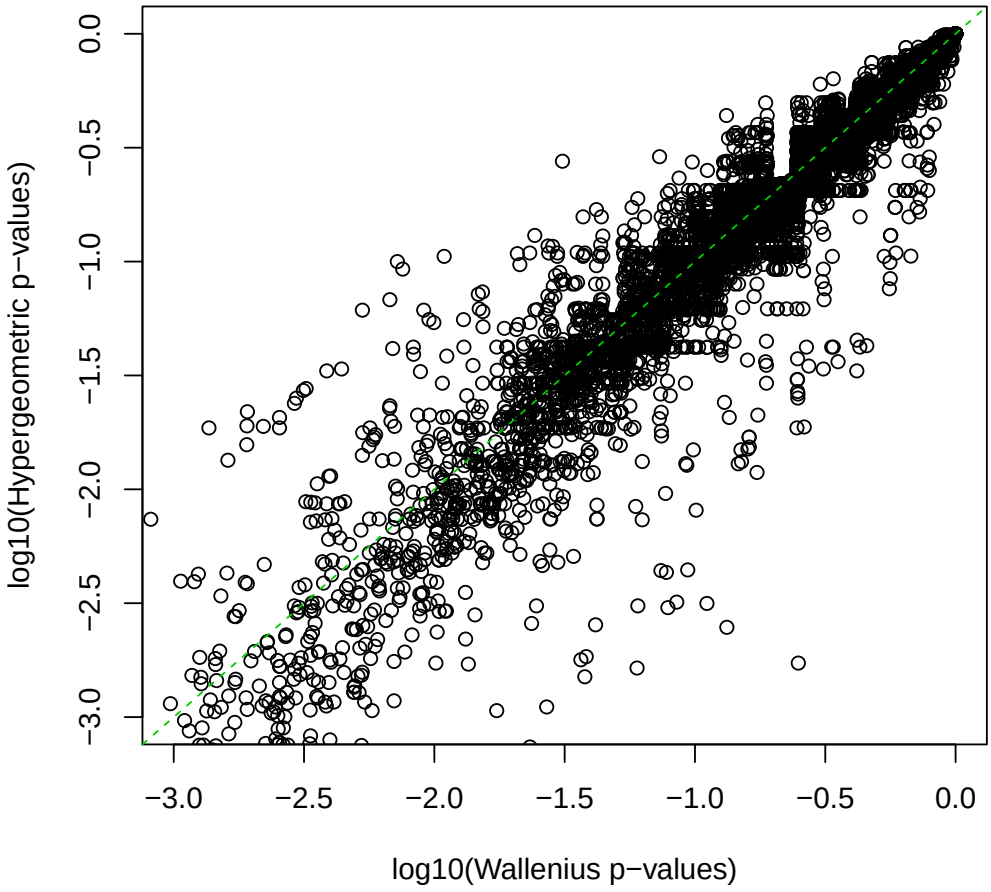
However, if you still wish to ignore length bias in calculating GO category enrichment, this is again accomplished using the `method` option.

```
> GO.nobias=goseq(pwf,"hg19","ensGene",method="Hypergeometric")
> head(GO.nobias)
```

```
        category over_represented_pvalue under_represented_pvalue numDEInCat
11078 GO:0044763            7.376357e-15                1.0000000       1849
11050 GO:0044699            1.884542e-14                1.0000000       2006
2525  GO:0005737            3.153001e-10                1.0000000       1863
3370  GO:0007049            8.736949e-09                1.0000000        382
135   GO:0000278            9.220328e-09                1.0000000        245
13282 GO:0051726            1.008192e-07                0.9999999        225
      numInCat                              term ontology
11078     8182 single-organism cellular process       BP
11050     9000          single-organism process       BP
2525      8418                         cytoplasm       CC
3370      1450                        cell cycle       BP
135        865                mitotic cell cycle       BP
13282      803         regulation of cell cycle       BP
```

Ignoring length bias gives very different results from a length bias corrected analysis.

```
> plot(log10(GO.wall[,2]), log10(GO.nobias[match(GO.nobias[,1],GO.wall[,1]),2]),
+      xlab="log10(Wallenius p-values)", ylab="log10(Hypergeometric p-values)",
+      xlim=c(-3,0), ylim=c(-3,0))
> abline(0,1,col=3,lty=2)
```



6.5.5 Limiting GO categories and other category based tests

By default, `goseq` tests all three major Gene Ontology branches; Cellular Components, Biological Processes and Molecular Functions. However, it is possible to limit testing to any combination of the major branches by using the `test.cats` argument to the goseq function. This is done by specifying a vector consisting of some combination of the strings "GO:CC", "GO:BP" and "GO:MF". For example, to test for only Molecular Function GO categories:

```
> GO.MF=goseq(pwf,"hg19","ensGene",test.cats=c("GO:MF"))
> head(GO.MF)
```

```
       category over_represented_pvalue under_represented_pvalue numDEInCat
1065 GO:0005198           7.885214e-06                0.9999954        138
258  GO:0003735           3.854837e-05                0.9999816         52
```

```
1222 GO:0008092            1.181744e-04                0.9999177        188
1179 GO:0005515            1.313342e-04                0.9998894       1502
2299 GO:0030215            4.127806e-04                0.9999417         10
965  GO:0005001            4.543717e-04                0.9999489          9
     numInCat                                                          term ontology
1065      513                                  structural molecule activity       MF
258       194                               structural constituent of ribosome       MF
1222      689                                     cytoskeletal protein binding       MF
1179     6864                                                  protein binding       MF
2299       16                                   semaphorin receptor binding       MF
965        13 transmembrane receptor protein tyrosine phosphatase activity       MF
```

Native support for other category tests, such as KEGG pathway analysis are also made available via this argument. See the man `goseq` function man page for up to date information on what category tests are natively supported.

6.5.6 Making sense of the results

Having performed the GO analysis, you may now wish to interpret the results. If you wish to identify categories significantly enriched/unenriched below some p-value cutoff, it is necessary to first apply some kind of multiple hypothesis testing correction. For example, GO categories over enriched using a .05 FDR cutoff (**?**) are:

```
> enriched.GO=GO.wall$category[p.adjust(GO.wall$over_represented_pvalue,
+             method="BH")<.05]
> head(enriched.GO)

[1] "GO:0044763" "GO:0044699" "GO:0005737" "GO:0000278" "GO:0031988" "GO:0044444"
```

Unless you are a machine, GO accession identifiers are probably not very meaningful to you. Information about each term can be obtained from the Gene Ontology website, `http://www.geneontology.org/`, or using the R package `GO.db`.

```
> library(GO.db)
> for(go in enriched.GO[1:10]){
+         print(GOTERM[[go]])
+         cat("--------------------------------------\n")
+ }

GOID: GO:0044763
Term: single-organism cellular process
Ontology: BP
Definition: Any process that is carried out at the cellular level,
```

```
    occurring within a single organism.
-------------------------------------
GOID: GO:0044699
Term: single-organism process
Ontology: BP
Definition: A biological process that involves only one organism.
Synonym: single organism process
-------------------------------------
GOID: GO:0005737
Term: cytoplasm
Ontology: CC
Definition: All of the contents of a cell excluding the plasma membrane
    and nucleus, but including other subcellular structures.
-------------------------------------
GOID: GO:0000278
Term: mitotic cell cycle
Ontology: BP
Definition: Progression through the phases of the mitotic cell cycle, the
    most common eukaryotic cell cycle, which canonically comprises four
    successive phases called G1, S, G2, and M and includes replication of
    the genome and the subsequent segregation of chromosomes into daughter
    cells. In some variant cell cycles nuclear replication or nuclear
    division may not be followed by cell division, or G1 and G2 phases may
    be absent.
-------------------------------------
GOID: GO:0031988
Term: membrane-bounded vesicle
Ontology: CC
Definition: Any small, fluid-filled, spherical organelle enclosed by a
    lipid bilayer.
Synonym: membrane-enclosed vesicle
-------------------------------------
GOID: GO:0044444
Term: cytoplasmic part
Ontology: CC
Definition: Any constituent part of the cytoplasm, all of the contents of
    a cell excluding the plasma membrane and nucleus, but including other
    subcellular structures.
Synonym: cytoplasm component
-------------------------------------
GOID: GO:0006614
Term: SRP-dependent cotranslational protein targeting to membrane
```

```
Ontology: BP
Definition: The targeting of proteins to a membrane that occurs during
    translation and is dependent upon two key components, the
    signal-recognition particle (SRP) and the SRP receptor. SRP is a
    cytosolic particle that transiently binds to the endoplasmic reticulum
    (ER) signal sequence in a nascent protein, to the large ribosomal
    unit, and to the SRP receptor in the ER membrane.
Synonym: ER translocation
Synonym: SRP-dependent cotranslational membrane targeting
Synonym: SRP-dependent cotranslational protein-membrane targeting
-------------------------------------
GOID: GO:0007049
Term: cell cycle
Ontology: BP
Definition: The progression of biochemical and morphological phases and
    events that occur in a cell during successive cell replication or
    nuclear replication events. Canonically, the cell cycle comprises the
    replication and segregation of genetic material followed by the
    division of the cell, but in endocycles or syncytial cells nuclear
    replication or nuclear division may not be followed by cell division.
Synonym: cell-division cycle
-------------------------------------
GOID: GO:0005576
Term: extracellular region
Ontology: CC
Definition: The space external to the outermost structure of a cell. For
    cells without external protective or external encapsulating structures
    this refers to space outside of the plasma membrane. This term covers
    the host cell environment outside an intracellular parasite.
Synonym: extracellular
-------------------------------------
GOID: GO:0031982
Term: vesicle
Ontology: CC
Definition: Any small, fluid-filled, spherical organelle enclosed by
    membrane.
-------------------------------------
```

6.5.7 Understanding goseq internals

The situation may arise where it is necessary for the user to perform some of the data processing steps usually performed automatically by `goseq` themselves. With this in mind, it will be useful

to step through the preprocessing steps performed automatically by goseq to understand what is happening.

To start with, when nullp is called, goseq uses the genome and gene identifiers supplied to try and retrieve length information for all genes given to the genes argument. To do this, it retrieves the data from the database of gene lengths maintained in the package geneLenDataBase. This is performed by the getlength function in the following way:

```
> len=getlength(names(genes),"hg19","ensGene")
> length(len)

[1] 22743

> length(genes)

[1] 22743

> head(len)

[1]  247 3133 1978  466 1510  954
```

After some data cleanup, the length data and the DE data is then passed to the makespline function to produce the PWF. The nullp returns a data frame which has 3 columns, the original DEgenes vector, the length bias data (in a column called bias.data) and the PWF itself (in a column named pwf). The names of the genes are also kept in this data frame as the names of the rows. If length data could not be obtained for a certain gene the corresponding entries in the "bias.data" and "pwf" columns are set to NA.

Next we call the goseq function to determine over/under representation of GO categories amongst DE genes. When we do this, goseq looks for the appropriate organism package and tries to obtain the mapping from genes to GO categories from it. This is done using the getgo function as follows:

```
> go=getgo(names(genes),"hg19","ensGene")
> length(go)

[1] 22743

> length(genes)

[1] 22743

> head(go)
```

```
$<NA>
NULL

$ENSG00000182463
  [1] "GO:0000122" "GO:0006139" "GO:0006351" "GO:0006355" "GO:0006357" "GO:0006366"
  [7] "GO:0006725" "GO:0006807" "GO:0007275" "GO:0008150" "GO:0008152" "GO:0009058"
 [13] "GO:0009059" "GO:0009889" "GO:0009890" "GO:0009892" "GO:0009987" "GO:0010467"
 [19] "GO:0010468" "GO:0010556" "GO:0010558" "GO:0010605" "GO:0010629" "GO:0016070"
 [25] "GO:0018130" "GO:0019219" "GO:0019222" "GO:0019438" "GO:0031323" "GO:0031324"
 [31] "GO:0031326" "GO:0031327" "GO:0032501" "GO:0032502" "GO:0032774" "GO:0034641"
 [37] "GO:0034645" "GO:0034654" "GO:0043170" "GO:0044237" "GO:0044238" "GO:0044249"
 [43] "GO:0044260" "GO:0044271" "GO:0044699" "GO:0044707" "GO:0044767" "GO:0045892"
 [49] "GO:0045934" "GO:0046483" "GO:0048519" "GO:0048523" "GO:0048856" "GO:0050789"
 [55] "GO:0050794" "GO:0051171" "GO:0051172" "GO:0051252" "GO:0051253" "GO:0060255"
 [61] "GO:0065007" "GO:0071704" "GO:0080090" "GO:0090304" "GO:0097659" "GO:1901360"
 [67] "GO:1901362" "GO:1901576" "GO:1902679" "GO:1903506" "GO:1903507" "GO:2000112"
 [73] "GO:2000113" "GO:2001141" "GO:0005575" "GO:0005622" "GO:0005623" "GO:0005634"
 [79] "GO:0043226" "GO:0043227" "GO:0043229" "GO:0043231" "GO:0044424" "GO:0044464"
 [85] "GO:0000981" "GO:0000982" "GO:0001071" "GO:0001078" "GO:0001227" "GO:0003674"
 [91] "GO:0003676" "GO:0003677" "GO:0003682" "GO:0003700" "GO:0005488" "GO:0005515"
 [97] "GO:0043167" "GO:0043169" "GO:0044877" "GO:0046872" "GO:0097159" "GO:1901363"

$<NA>
NULL

$<NA>
NULL

$<NA>
NULL

$ENSG00000125835
  [1] "GO:0000375" "GO:0000377" "GO:0000387" "GO:0000398" "GO:0006139" "GO:0006351"
  [7] "GO:0006353" "GO:0006366" "GO:0006369" "GO:0006396" "GO:0006397" "GO:0006464"
 [13] "GO:0006479" "GO:0006725" "GO:0006807" "GO:0006810" "GO:0006913" "GO:0008150"
 [19] "GO:0008152" "GO:0008213" "GO:0008334" "GO:0008380" "GO:0009058" "GO:0009059"
 [25] "GO:0009987" "GO:0010467" "GO:0016043" "GO:0016070" "GO:0016071" "GO:0018130"
 [31] "GO:0019438" "GO:0019538" "GO:0022607" "GO:0022613" "GO:0022618" "GO:0032259"
 [37] "GO:0032774" "GO:0034622" "GO:0034641" "GO:0034645" "GO:0034654" "GO:0036211"
 [43] "GO:0043170" "GO:0043412" "GO:0043414" "GO:0043933" "GO:0044085" "GO:0044237"
 [49] "GO:0044238" "GO:0044249" "GO:0044260" "GO:0044267" "GO:0044271" "GO:0046483"
 [55] "GO:0046907" "GO:0051169" "GO:0051170" "GO:0051179" "GO:0051234" "GO:0051641"
```

```
 [61] "GO:0051649" "GO:0065003" "GO:0071704" "GO:0071826" "GO:0071840" "GO:0090304"
 [67] "GO:0097659" "GO:1901360" "GO:1901362" "GO:1901576" "GO:0005575" "GO:0005576"
 [73] "GO:0005622" "GO:0005623" "GO:0005634" "GO:0005654" "GO:0005681" "GO:0005682"
 [79] "GO:0005683" "GO:0005684" "GO:0005685" "GO:0005686" "GO:0005687" "GO:0005689"
 [85] "GO:0005697" "GO:0005737" "GO:0005829" "GO:0030529" "GO:0030532" "GO:0031974"
 [91] "GO:0031981" "GO:0031982" "GO:0031988" "GO:0032991" "GO:0034708" "GO:0034709"
 [97] "GO:0034719" "GO:0043226" "GO:0043227" "GO:0043229" "GO:0043230" "GO:0043231"
[103] "GO:0043233" "GO:0043234" "GO:0044421" "GO:0044422" "GO:0044424" "GO:0044428"
[109] "GO:0044444" "GO:0044446" "GO:0044464" "GO:0046540" "GO:0070013" "GO:0070062"
[115] "GO:0071004" "GO:0071010" "GO:0071013" "GO:0071204" "GO:0097525" "GO:0097526"
[121] "GO:1903561" "GO:1990904" "GO:0003674" "GO:0003676" "GO:0003723" "GO:0005488"
[127] "GO:0005515" "GO:0017069" "GO:0030620" "GO:0044822" "GO:0070034" "GO:0071208"
[133] "GO:0097159" "GO:1901363"
```

Note that some of the gene categories have been returned as "NULL". This means that a GO category could not be found in the database for one of the genes. In the `goseq` command, enrichment will only be calculated using genes with a GO category by default. However, in older versions of goseq (below 1.15.2), we counted all genes. i.e. genes with no categories still counted towards the total number of gene outside of any single category. It is possible to switch between these two behaviors using the `use_genes_without_cat` flag in `goseq`.

The first thing the getgo function does is to convert the UCSC genome/ID namings into the naming convention used by the organism packages. This is done using two hard coded conversion vectors that are included in the `goseq` package but usually hiden from the user.

```
> goseq:::.ID_MAP
```

```
 knownGene    refGene    ensGene geneSymbol        sgd     plasmo       tair
      "eg"       "eg"  "ENSEMBL"   "SYMBOL"      "sgd"   "plasmo"     "tair"
```

```
> goseq:::.ORG_PACKAGES
```

```
          anoGam      Arabidopsis           bosTau               ce
     "org.Ag.eg"    "org.At.tair"      "org.Bt.eg"      "org.Ce.eg"
          canFam               dm           danRer      E. coli K12
     "org.Cf.eg"      "org.Dm.eg"      "org.Dr.eg"   "org.EcK12.eg"
   E. coli Sakai           galGal               hg               mm
"org.EcSakai.eg"      "org.Gg.eg"      "org.Hs.eg"      "org.Mm.eg"
          rheMac          Malaria           panTro               rn
    "org.Mmu.eg"  "org.Pf.plasmo"      "org.Pt.eg"      "org.Rn.eg"
          sacCer           susScr           xenTro
    "org.Sc.sgd"      "org.Ss.eg"      "org.Xl.eg"
```

It is just as valid to run the length and GO category fetching as separate steps and then pass the result to the `nullp` and `goseq` functions using the `bias.data` and `gene2cat` arguments. Thus the following two blocks of code are equivalent:

```
> pwf=nullp(genes,"hg19","ensGene")
> go=goseq(pwf,"hg19","ensGene")
```

and

```
> gene_lengths=getlength(names(genes),"hg19","ensGene")
> pwf=nullp(genes,bias.data=gene_lengths)
> go_map=getgo(names(genes),"hg19","ensGene")
> go=goseq(pwf,"hg19","ensGene",gene2cat=go_map)
```

6.6 KEGG pathway analysis

In order to illustrate performing a category test not present in the goseq database, we perform a KEGG pathway analysis. For human, the mapping from KEGG pathways to genes are stored in the package org.Hs.eg.db, in the object org.Hs.egPATH. In order to test for KEGG pathway over representation amongst DE genes, we need to extract this information and put it in a format that goseq understands. Unfortunately, the org.Hs.eg.db package does not contain direct mappings between ENSEMBL gene ID and KEGG pathway. Therefore, we have to construct this map by combining the ENSEMBL <-> Entrez and Entrez <-> KEGG mappings. This can be done using the following code:

```
> # Get the mapping from ENSEMBL 2 Entrez
> en2eg=as.list(org.Hs.egENSEMBL2EG)
> # Get the mapping from Entrez 2 KEGG
> eg2kegg=as.list(org.Hs.egPATH)
> # Define a function which gets all unique KEGG IDs
> # associated with a set of Entrez IDs
> grepKEGG=function(id,mapkeys){unique(unlist(mapkeys[id],use.names=FALSE))}
> # Apply this function to every entry in the mapping from
> # ENSEMBL 2 Entrez to combine the two maps
> kegg=lapply(en2eg,grepKEGG,eg2kegg)
> head(kegg)
```

Note that this step is quite time consuming. The code written here is not the most efficient way of producing this result, but the logic is much clearer than faster algorithms. The source code for getgo contains a more efficient routine.

We produce the PWF as before. Then, to perform a KEGG analysis, we simply make use of the gene2cat option in goseq:

```
> pwf=nullp(genes,"hg19","ensGene")
> KEGG=goseq(pwf,gene2cat=kegg)
> head(KEGG)
```

Note that we do not have to tell the goseq function what organism and gene ID we are using as we are manually supplying the mapping between genes and categories.

KEGG analysis is shown as an illustration of how to supply your own mapping between gene ID and category, KEGG analysis is actually natively supported by GOseq and we could have performed it with the following code.

```
> pwf=nullp(genes,'hg19','ensGene')
> KEGG=goseq(pwf,'hg19','ensGene',test.cats="KEGG")
> head(KEGG)
```

```
    category over_represented_pvalue under_represented_pvalue numDEInCat numInCat
88     03010            8.215777e-06                0.9999974         29       88
77     00900            2.412873e-04                0.9999707         10       15
113    04115            8.297560e-04                0.9996779         26       64
175    04964            2.168168e-03                0.9995887         10       17
27     00330            3.711464e-03                0.9986421         18       44
20     00250            5.247804e-03                0.9984194         13       28
```

Noting that this time it was necessary to tell the goseq function that we are using HG19 and ENSEMBL gene ID, as the function needs this information to automatically construct the mapping from geneid to KEGG pathway.

6.7 Extracting mappings from organism packages

If you know that the information mapping gene ID to your categories of interest is contained in the organism packages, but `goseq` fails to fetch it automatically, you may want to extract it yourself and then pass it to the `goseq` function using the `gene2cat` argument. This is done in exactly the same way as extracting the KEGG to ENSEMBL mappings in the section "KEGG pathway analysis" above. This example is actually the worst case, where it is necessary to combine two mappings to get the desired list. If we had instead wanted the association between Entrez gene IDs and KEGG pathways, the following code would have been sufficient:

```
> kegg=as.list(org.Hs.egPATH)
> head(kegg)
```

```
$`1`
[1] NA

$`2`
[1] "04610"

$`3`
```

```
[1] NA

$`9`
[1] "00232" "00983" "01100"

$`10`
[1] "00232" "00983" "01100"

$`11`
[1] NA
```

A note on fetching GO mappings from the organism. The data structure of GO is a directed acyclic graph. This means that in addition to each GO category being associated with a set of genes, it may also have children that are associated to other genes. It is important to use the org.Hs.egGO2ALLEGS and NOT the org.Hs.egGO object to create the mapping between GO categories and gene identifiers, as the latter does not include the links to genes arising from "child" GO categories. Thank you to Christopher Fjell for pointing this out.

6.8 Correcting for other biases

It is possible that in some circumstances you will wish to correct not just for length bias, but for the total number of counts. This can make sense because power to detect DE depends on the total number of counts a gene receives, which is the product of gene length and gene expression. So correcting for read count bias will compensate for all biases, known and unknown, in power to detect DE. On the other hand, it will also remove bias resulting from differences in expression level, which may not be desirable.

Correcting for count bias will produce a different PWF. Therefore, we need to tell `goseq` about the data on which the fraction DE depends when calculating the PWF using the `nullp` function. We then simply pass the result to `goseq` as usual.

So, in order to tell `goseq` to correct for read count bias instead of length bias, all you need to do is supply a numeric vector, containing the number of counts for each gene to `nullp`.

```
> countbias=rowSums(counts)[rowSums(counts)!=0]
> length(countbias)

[1] 22743

> length(genes)

[1] 22743
```

To use the count bias when doing GO analysis, simply pass this vector to `nullp` using the `bias.data` option. Note that we have to supply "hg19" and "ensGene" to `goseq` as it is not used by nullp and hence not in the pwf.counts object.

```
> pwf.counts=nullp(genes,bias.data=countbias)
> GO.counts=goseq(pwf.counts,"hg19","ensGene")
> head(GO.counts)
```

```
        category over_represented_pvalue under_represented_pvalue numDEInCat
5369  GO:0016021            6.008040e-13                         1        701
11050 GO:0044699            6.400832e-13                         1       2006
7337  GO:0031224            1.812125e-12                         1        708
2626  GO:0005886            1.876015e-12                         1        700
11078 GO:0044763            3.448584e-12                         1       1849
15572 GO:0071944            5.093866e-12                         1        716
      numInCat                              term ontology
5369      3102   integral component of membrane       CC
11050     9000          single-organism process       BP
7337      3163  intrinsic component of membrane       CC
2626      2950                  plasma membrane       CC
11078     8182 single-organism cellular process       BP
15572     3031                   cell periphery       CC
```

Note that if you want to correct for length bias, but your organism/gene identifier is not natively supported, then you need to follow the same procedure as above, only the numeric vector supplied will contain each gene's length instead of its number of reads.

7 Setup

This vignette was built on:

```
> 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  stats4    stats     graphics  grDevices utils     datasets  methods
```

```
[9] base

other attached packages:
 [1] GO.db_3.4.0           org.Hs.eg.db_3.4.0    AnnotationDbi_1.36.0
 [4] Biobase_2.34.0        rtracklayer_1.34.0    GenomicRanges_1.26.0
 [7] GenomeInfoDb_1.10.0   IRanges_2.8.0         S4Vectors_0.12.0
[10] BiocGenerics_0.20.0   edgeR_3.16.0          limma_3.30.0
[13] goseq_1.26.0          geneLenDataBase_1.9.2 BiasedUrn_1.07

loaded via a namespace (and not attached):
 [1] XVector_0.14.0             zlibbioc_1.20.0
 [3] GenomicAlignments_1.10.0   BiocParallel_1.8.0
 [5] lattice_0.20-34            tools_3.3.1
 [7] SummarizedExperiment_1.4.0 grid_3.3.1
 [9] nlme_3.1-128               mgcv_1.8-15
[11] DBI_0.5-1                  Matrix_1.2-7.1
[13] bitops_1.0-6               biomaRt_2.30.0
[15] RCurl_1.95-4.8             RSQLite_1.0.0
[17] GenomicFeatures_1.26.0     Biostrings_2.42.0
[19] Rsamtools_1.26.0           locfit_1.5-9.1
[21] XML_3.98-1.4
```

8 Acknowledgments

Christopher Fjell for a series of bug fixes and pointing out the difference between the egGO and egGO2ALLEGS objects in the organism packages.