Moment based gene set enrichment testing – the npGSEA package

***Jessica L. Larson*[1*] *and Art Owen*[2]**

[1em] [1] Department of Bioinformatics and Computational Biology, Genentech, Inc.
[2] Department of Statistics, Stanford University
[*]larson.jessica (at) gene.com

October 7, 2025

Contents

1 Introduction

Gene set methods are critical to the analysis of gene expression data. The npGSEA package provides methods to run permutation-based gene set enrichment analyses without the typically computationally expensive permutation cost. These methods allow users to adjust for covariates and approximate corresponding permutation distributions. We are currently evaluating the applicability and accuracy of our method for RNA-seq expression data.

Our methods find the exact relevant moments of a weighted sum of (squared) test statistics under permutation, taking into account correlations among the test statistics. We find moment-based gene set enrichment p-values that closely approximate the permutation method p-values.

This vignette describes a typical analysis workflow and includes some information about the statistical theory behind npGSEA. For more technical details, please see Larson and Owen, 2015 .

2 Example workflow for GSEA

2.1 Preparing our gene sets and our dataset for analysis

For our example, we will use the ALL dataset. We begin by loading relevant libraries, subsetting the data, and running `featureFilter` on this data set. For details on these methods, please see the `limma` manual.

```
> library(ALL)
> library(hgu95av2.db)
> library(genefilter)
> library(limma)
> library(GSEABase)
> library(npGSEA)
> data(ALL)
> ALL <- ALL[, ALL$mol.biol %in% c('NEG','BCR/ABL') &
+      !is.na(ALL$sex)]
> ALL$mol.biol <- factor(ALL$mol.biol,
+      levels = c('NEG', 'BCR/ABL'))
> ALL <- featureFilter(ALL)
```

We adjust the feature names of the ALL dataset so that they match the names of our gene sets below. We convert them to entrez ids.

```
> featureNames(ALL) <- select(hgu95av2.db, featureNames(ALL),
+                             "ENTREZID", "PROBEID")$ENTREZID
```

We now make four arbitrary gene sets by randomly selecting from the genes in our universe.

```
> xData <- exprs(ALL)
> geneEids <- rownames(xData)
> set.seed(12345)
> set1 <- GeneSet(geneIds=sample(geneEids,15, replace=FALSE),
+              setName="set1",
+              shortDescription="This is set1")
> set2 <- GeneSet(geneIds=sample(geneEids,50, replace=FALSE),
+              setName="set2",
+              shortDescription="This is set2")
> set3 <- GeneSet(geneIds=sample(geneEids,100, replace=FALSE),
+              setName="set3",
+              shortDescription="This is set3")
> set4 <- GeneSet(geneIds=sample(geneEids,500, replace=FALSE),
+             setName="set4",
+             shortDescription="This is set4")
```

As a positive control, we also make three gene sets that include our top differentially expressed genes.

```
> model <- model.matrix(~mol.biol, ALL)
> fit  <-  eBayes(lmFit(ALL, model))
> tt <- topTable(fit, coef=2, n=200)
> ttUp <- tt[which(tt$logFC >0), ]
> ttDown <- tt[which(tt$logFC <0), ]
> set5 <- GeneSet(geneIds=rownames(ttUp)[1:20],
+         setName="set5",
+         shortDescription="This is a true set of the top 20 DE
+         genes with a positive fold change")
> set6 <- GeneSet(geneIds=rownames(ttDown)[1:20],
+         setName="set6",
+         shortDescription="This is a true set of the top 20 DE genes
+         with a negative fold change")
> set7 <- GeneSet(geneIds=c(rownames(ttUp)[1:10], rownames(ttDown)[1:10]),
+         setName="set7",
+         shortDescription="This is a true set of the top 10 DE genes
+         with a positive and a negative fold change")
```

We then collapse all of our gene sets into a `GeneSetCollection`. For more information on `GeneSets` and `GeneSetCollections`, see the `GSEABase` manual.

```
> gsc <- GeneSetCollection( c(set1, set2, set3, set4, set5, set6, set7) )
> gsc
```

```
GeneSetCollection
  names: set1, set2, ..., set7 (7 total)
  unique identifiers: 933, 10970, ..., 6767 (693 total)
  types in collection:
    geneIdType: NullIdentifier (1 total)
    collectionType: NullCollection (1 total)
```

2.2 Running npGSEA

Now that we have both our gene sets and experiment, we are ready to run npGSEA and determine the level of enrichment in our experiment. We can use npGSEA with our eset or expression data (xData) directly. We call npGSEASummary to get a summary of the results. T_Gw is explained in more detail in Section 3.2.

```
> yFactor <- ALL$mol.biol
> res1 <- npGSEA(x = ALL, y = yFactor, set = set1)  ##with the eset
> res1
```

```
Normal Approximation for set1
T_Gw = 0.457
var(T_Gw) = 0.0364
pLeft = 0.992, pRight = 0.00824, pTwoSided = 0.0165
```

```
> res2_exprs <- npGSEA(xData, ALL$mol.biol, gsc[[2]])  ##with the expression data
> res2_exprs
```

```
Normal Approximation for set2
T_Gw = 1.17
var(T_Gw) = 0.138
pLeft = 0.999, pRight = 0.000793, pTwoSided = 0.00159
```

npGSEA has several built in accessor functions to gather more information about the analysis of your set of interest in your experiment.

```
> res3 <- npGSEA(ALL, yFactor, set3)
> res3
```

```
Normal Approximation for set3
T_Gw = -0.645
var(T_Gw) = 0.223
pLeft = 0.0861, pRight = 0.914, pTwoSided = 0.172
```

```
> geneSetName(res3)
```

```
[1] "set3"
> stat(res3)
[1] -0.6449809
> sigmaSq(res3)
[1] 0.223123
> zStat(res3)
[1] -1.365447
> pTwoSided(res3)
[1] 0.1721127
> pLeft(res3)
[1] 0.08605636
> pValues(res3)
pLeft = 0.0861, pRight = 0.914, pTwoSided = 0.172
> dim(xSet(res3))
[1] 100 109
```

There is also a npGSEA specific plot function (npGSEAPlot) to visualize the results of your analysis. Highlighted in red on the plot is the corresponding zStat of our analysis.

```
> npGSEAPlot(res3)
```

2.3 Running npGSEA with the beta and chi-sq approximations

There are three types of approximation methods in npGSEA: "norm", "beta", and "chiSq". Each method is discussed in brief in Section 3. The "norm" approximation method is the default. Note that each of these methods has the same $\hat{\beta}_g$ (see methods section).

```
> res5_norm <- npGSEA(ALL, yFactor, set5, approx= "norm")
> res5_norm
Normal Approximation for set5
T_Gw = 5
var(T_Gw) = 0.394
pLeft = 1, pRight = 8.72e-16, pTwoSided = 1.74e-15
```

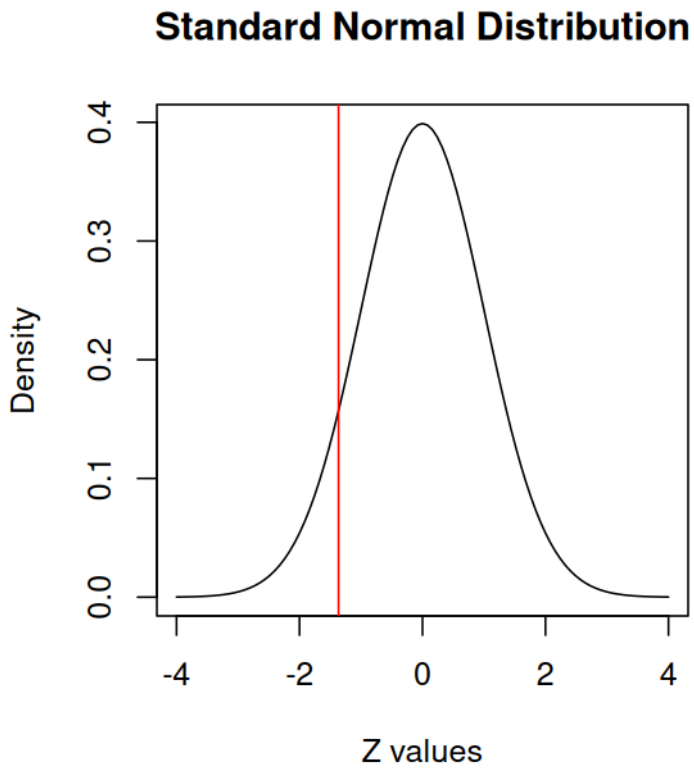

Figure 1: Set3 normal approximation results This plot displays the standard normal curve and our observed zStat for set3 in this analysis.

```
> betaHats(res5_norm)
```

```
             [,1]
1490   0.53814745
168544 0.10183475
1893   0.23559626
2013   0.11853248
2022   0.23358385
216    0.19554569
2273   0.35939613
23179  0.24618755
25     0.21597396
2549   0.21100011
2934   0.33968229
5445   0.23247513
55884  0.16183870
57556  0.37650630
```

```
687    0.46075522
6915   0.10897784
7277   0.34083849
92     0.15933797
9369   0.09865998
9788   0.26243503
```

```
> npGSEAPlot(res5_norm)
```

The beta approximation yields results quite similar to the normal approximation.

```
> res5_beta <- npGSEA(ALL, yFactor, set5, approx= "beta")
> res5_beta
```

```
Beta Approximation for set5
T_Gw = 5
var(T_Gw) = 0.394
pLeft = 1, pRight = 5.83e-29, pTwoSided = 1.17e-28
```

```
> betaHats(res5_beta)
```

```
             [,1]
1490   0.53814745
168544 0.10183475
1893   0.23559626
2013   0.11853248
2022   0.23358385
216    0.19554569
2273   0.35939613
23179  0.24618755
25     0.21597396
2549   0.21100011
2934   0.33968229
5445   0.23247513
55884  0.16183870
57556  0.37650630
687    0.46075522
6915   0.10897784
7277   0.34083849
92     0.15933797
9369   0.09865998
9788   0.26243503
```

```
> npGSEAPlot(res5_beta)
```

The chi-sq approximation method is only available for the two-sided test. Here we call `npGSEA` and then show how the `chiSqStat` is related to `C_Gw`. `C_Gw` is explained in more detail in Section 3.2.

```
> res5_chiSq <- npGSEA(ALL, yFactor, set5, approx= "chiSq")
> res5_chiSq
```

```
Chi-sq Approximation for set5
C_Gw = 1.52
df = 2.42, sigmaSq = 0.0172
pTwoSided = 0
```

```
> betaHats(res5_chiSq)
```

```
             [,1]
1490   0.53814745
168544 0.10183475
1893   0.23559626
2013   0.11853248
2022   0.23358385
216    0.19554569
2273   0.35939613
23179  0.24618755
25     0.21597396
2549   0.21100011
2934   0.33968229
5445   0.23247513
55884  0.16183870
57556  0.37650630
687    0.46075522
6915   0.10897784
7277   0.34083849
92     0.15933797
9369   0.09865998
9788   0.26243503
```

```
> chiSqStat(res5_chiSq)
```

```
[1] 88.81123
```

```
> stat(res5_chiSq)
```

```
[1] 1.524982
```

```
> stat(res5_chiSq)/sigmaSq(res5_chiSq)
```

```
[1] 88.81123
```

```
> npGSEAPlot(res5_chiSq)
```

Note that, as we expected, set5 is a significantly enriched in all three methods. In each of the three corresponding plots, the observed statistic is a very rare event.

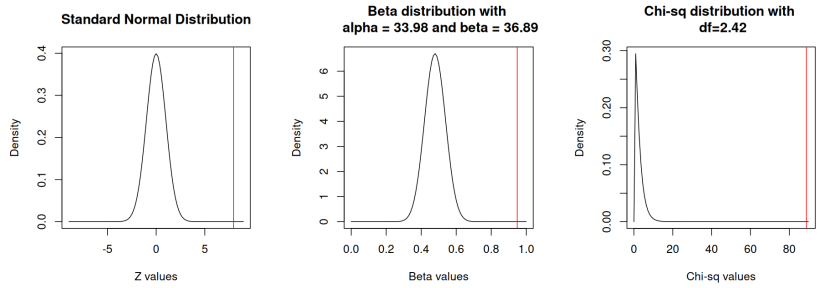

Figure 2: Set5 normal, beta, and chi-sq approximation results These plots displays the reference normal, beta, and chi-sq curves, and our observed zStat, betaStat, and chiSq-Stat for set5 in this analysis.

2.4 Adding weights to the model

Sometimes we do not want to weigh each gene in our set equally. We want to assign a larger weight to genes that are of a particular interest, and a lower weight to genes that we know may behave poorly. In this example, we weight the genes in set7 by their variance.

```
> res7_nowts <- npGSEA(x = ALL, y= yFactor, set = set7)
> res7_nowts
```

```
Normal Approximation for set7
T_Gw = 1.4
var(T_Gw) = 0.0583
pLeft = 1, pRight = 3.04e-09, pTwoSided = 6.09e-09
```

```
> wts <- apply(exprs(ALL)[match(geneIds(set7), featureNames(ALL)), ],
+                                                  1, var)
> wts <- 1/wts
> res7_wts <- npGSEA(x = ALL, y = yFactor, set = set7,  w = wts, approx= "norm")
> res7_wts
```

```
Normal Approximation for set7
T_Gw = 8.33
var(T_Gw) = 1.47
pLeft = 1, pRight = 3.15e-12, pTwoSided = 6.3e-12
```

By adding these weights, we get a slightly more significant result. We can add weights for the beta and chi-sq approximations, too. By default, npGSEA assigns a weight of 1 for all genes.

2.5 Adding covariates to model

Often we want to correct for confounders in our model. To do this with npGSEA, we provide a vector or matrix in the covars slot of our function. npGSEA then projects both the data (x) and the outcome of interest (y) against our covariate matrix/vector. The resulting residuals are used for further analysis.

In this example, we correct for the age and sex of the subjects in our experiment. For more details on model selection and its relation to inference, please see the limma manual.

```
> res3_age <- npGSEA(x = ALL, y = yFactor, set = set3, covars = ALL$age)
> res3_age
```

```
Normal Approximation for set3
T_Gw = -0.404
var(T_Gw) = 0.185
pLeft = 0.174, pRight = 0.826, pTwoSided = 0.348
```

```
> res3_agesex <- npGSEA(x = ALL, y = yFactor, set = set3, covars = cbind(ALL$age, ALL$sex
> res3_agesex
```

```
Normal Approximation for set3
T_Gw = -0.382
var(T_Gw) = 0.182
pLeft = 0.185, pRight = 0.815, pTwoSided = 0.37
```

By adjusting for these variables, we get a slight different result than above. Note that we can adjust for covariates in the beta and chi-sq approximation methods, too.

2.6 Running npGSEA with multiple gene sets

To explore multiple gene sets, we let set be a GeneSetCollection. This returns a list of npGSEAResultNorm objects, called a npGSEAResultNormCollection. We can access statistics for each GeneSet in our analysis through accessors of npGSEAResultNormCollection.

```
> resgsc_norm <- npGSEA(x = ALL, y = yFactor, set = gsc)
> unlist( pLeft(resgsc_norm) )
```

```
         set1         set2         set3         set4         set5         set6
9.917587e-01 9.992073e-01 8.605636e-02 9.874496e-01 1.000000e+00 2.660288e-11
         set7
1.000000e+00
```

```
> unlist( stat (resgsc_norm) )
```

```
      set1       set2       set3       set4       set5       set6       set7
 0.4574477  1.1712563 -0.6449809  3.2438833  4.9973052 -3.7548816  1.4034573
```

```
> unlist( zStat (resgsc_norm) )
```

```
     set1      set2      set3      set4      set5      set6      set7
 2.398051  3.158578 -1.365447  2.239848  7.958358 -6.561678  5.814316
```

Note how quick our method is. We get results as accurate as permutation methods in a fraction of the time, even for multiple gene sets.

Using the `ReportingTools` package, we can publish these results to a HTML page for exploration. We first adjust for multiple testing.

```
> pvals <- p.adjust( unlist(pTwoSided(resgsc_norm)), method= "BH" )
> library(ReportingTools)
> npgseaReport <- HTMLReport (shortName = "npGSEA",
+         title = "npGSEA Results", reportDirectory = "./reports")
> publish(gsc, npgseaReport, annotation.db = "org.Hs.eg",
+         setStats = unlist(zStat (resgsc_norm)), setPValues = pvals)
> finish(npgseaReport)
```

3 Methods in brief

3.1 Disadvantages to a permutation approach

There are three main disadvantages to permutation-based analyses: cost, randomness, and granularity.

Testing many sets of genes becomes computationally expensive for two reasons. First, there are many test statistics to calculate in each permuted version of the data. Second, to allow for multiplicity adjustment, we require small nominal p-values to draw inference about our sets, which in turn requires a large number of permutations.

Permutations are also subject random inference. Because permutations are based on a random shuffling of the data, there is a chance that we will obtain a different p-value for our set of interest each time we run our permutation analysis.

Permutations also have a granularity problem. If we do M permutations, then the smallest possible p-value we can attain is $1/(M+1)$. When it is necessary to adjust for multiplicity, the permutation approach becomes very computationally expensive. Another aspect of the granularity problem is that permutations give us no basis to distinguish between two gene sets that both have the same p-value $1/(M+1)$. There may be many such gene sets, and they have meaningfully different effect sizes.

Because of each of these limitations of permutation testing, there is a need to move beyond permutation-based GSEA methods. The methods we present in npGSEA and discuss in brief below are not as computationally expensive, random, or granular than their permutation counterparts. More details on our method can be found in Larson and Owen (2015).

3.2 Test statistics

We present our notation using the language of gene expression experiments.

Let g and h denote individual genes and G be a set of genes. Our experiment has n subjects. The subjects may represent patients, cell cultures, or tissue samples. The expression level for gene g in subject i is X_{gi}, and Y_i is the target variable on subject i. Y_i is often a treatment, disease, or genotype. We center the variables so that $\sum_{i=1}^n Y_i = \sum_{i=1}^n X_{gi} = 0, \forall g$.

Our measure of association for gene g on our treatment of interest is

$$\hat{\beta}_g = \frac{1}{n}\sum_{i=1}^{n} X_{gi}Y_i.$$

We consider the linear statistic

$$T_{G,w} = \sum_{g\in G} w_g \hat{\beta}_g$$

and the quadratic statistic

$$C_{G,w} = \sum_{g\in G} w_g \hat{\beta}_g^2,$$

where w_g corresponds to the weight given to gene g in set G.

3.3 Moment based reference distributions

To avoid the issues discussed above, we approximate the distribution of the permuted test statistics $T_{G,w}$ by Gaussian or by rescaled beta distributions. For the quadratic statistic $C_{G,w}$ we use a distribution of the form $\sigma^2\chi^2_{(\nu)}$.

For the Gaussian treatment of $T_{G,w}$ we calculate $\sigma^2 = Var(T_{G,w})$ under permutation, and then report the p-value

$$p = Pr(N(0, \sigma^2) \leq T_{G,w}).$$

The above is a left tail p-value. Two-sided and right tailed p-values are analogous.

When we want something sharper than the normal distribution, we can use a scaled Beta distribution, of the form $A + (B - A)Beta(\alpha, \beta)$. The $Beta(\alpha, \beta)$ distribution has a continuous density function on $0 < x < 1$ for $\alpha, \beta > 0$. We choose A, B, α and β by matching the upper and lower limits of $T_{G,w}$ under permutation, as well as its mean and variance. The observed left tailed p-value is

$$p = Pr\Big(Beta(\alpha, \beta) \leq \frac{T_{G,w} - A}{B - A}\Big).$$

For the quadratic test statistic $C_{G,w}$ we use a $\sigma^2\chi^2_{(\nu)}$ reference distribution reporting the p-value

$$\Pr(\sigma^2\chi^2_{(\nu)} \geq C_{G,w}),$$

after matching the first and second moments of $\sigma^2\chi^2_{(\nu)}$ to $E(C_{G,w})$ and $E(C^2_{G,w})$ under permutation, respectively.

Additional details on how σ^2, A, B, α, β, $E(C_{G,w})$, $E(C^2_{G,w})$, and ν are derived can be found in Larson and Owen (2015).

4 Session Info

- R version 4.5.1 Patched (2025-08-23 r88802), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Time zone: `America/New_York`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 24.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.22-bioc/R/lib/libRblas.so`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0`

- Base packages: base, datasets, grDevices, graphics, methods, stats, stats4, utils
- Other packages: ALL 1.51.0, AnnotationDbi 1.71.1, Biobase 2.69.1, BiocGenerics 0.55.1, GSEABase 1.71.1, IRanges 2.43.5, S4Vectors 0.47.4, XML 3.99-0.19, annotate 1.87.0, genefilter 1.91.0, generics 0.1.4, graph 1.87.0, hgu95av2.db 3.13.0, limma 3.65.5, npGSEA 1.45.0, org.Hs.eg.db 3.21.0
- Loaded via a namespace (and not attached): BiocManager 1.30.26, BiocStyle 2.37.1, Biostrings 2.77.2, DBI 1.2.3, KEGGREST 1.49.1, Matrix 1.7-4, MatrixGenerics 1.21.0, R6 2.6.1, RSQLite 2.4.3, Seqinfo 0.99.2, XVector 0.49.1, bit 4.6.0, bit64 4.6.0-1, blob 1.2.4, cachem 1.1.0, cli 3.6.5, compiler 4.5.1, crayon 1.5.3, digest 0.6.37, evaluate 1.0.5, fastmap 1.2.0, grid 4.5.1, htmltools 0.5.8.1, httr 1.4.7, knitr 1.50, lattice 0.22-7, matrixStats 1.5.0, memoise 2.0.1, pkgconfig 2.0.3, png 0.1-8, rlang 1.1.6, rmarkdown 2.30, splines 4.5.1, statmod 1.5.0, survival 3.8-3, tools 4.5.1, vctrs 0.6.5, xfun 0.53, xtable 1.8-4, yaml 2.3.10

5 References

Larson and Owen. (2015). Moment based gene set tests. *BMC Bioinformatics.* **16**:132. http://www.biomedcentral.com/1471-2105/16/132