12-2009

Analyzing Gene Expression Data in Terms of Gene Sets: Gene Set Enrichment Analysis

Wei Li
Georgia State University

ANALYZING GENE EXPRESSION DATA IN TERMS OF GENE SETS: GENE SET ENRICHMENT ANALYSIS

by

WEI LI

Under the Direction of Jiawei Liu

ABSTRACT

The DNA microarray biotechnology simultaneously monitors the expression of thousands of genes and aims to identify genes that are differently expressed under different conditions. From the statistical point of view, it can be restated as identify genes strongly associated with the response or covariant of interest. The Gene Set Enrichment Analysis (GSEA) method is one method which focuses the analysis at the functional related gene sets level instead of single genes. It helps biologists to interpret the DNA microarray data by their previous biological knowledge of the genes in a gene set. GSEA has been shown to efficiently identify gene sets containing known disease-related genes in the real experiments. Here we want to evaluate the statistical power of this method by simulation studies. The results show that the the power of GSEA is good enough to identify the gene sets highly associated with the response or covariant of interest.

INDEX WORDS: Gene Set Enrichment Analysis, Gene expression data, Satistical power, Type II error

ANALYZING GENE EXPRESSION DATA IN TERMS OF GENE SETS: GENE SET ENRICHMENT ANALYSIS

by

WEI LI

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2009

ANALYZING GENE EXPRESSION DATA IN TERMS OF GENE SETS: GENE SET ENRICHMENT ANALYSIS

by

WEI LI

Committee Chair: Dr. Jiawei Liu

Committee: Dr. Yu-sheng Hsu
Dr. Gengsheng Qin
Dr. Yuanhui Xiao

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2009

ACKNOWLEDGEMENTS

There are so many people to thank for supporting me throughout my study period. I would like to dedicate my first Thank you to my advisor, Dr. Jiawei Liu for all her encouragement and insightful instructions. There is no way I could have completed this project smoothly without Dr. Liu's detailed suggestions and patient guidance. She is always kind to help me in the every aspect as regarding to my thesis project and my study in the department.

I am also very thankful to my committee members, Dr. Yu-Sheng Hsu, Dr. Gengsheng Qin and Dr. Yuanhui Xiao. I appreciate the suggestions and critical review of my thesis draft and your constant support in all aspects during my study. I learned a lot from you, not only the knowledge learned from project or class, but the suggestions and advices to the career building and the attitude towards life.

I would also like to thank all other professors and students in the Department of Mathematics and Statistics at Georgia State University for their instruction and assistance during my study in Georgia State University. People in the department are so friendly that I feel very comfortable when talking and discussing study projects with people around.

Finally, my family is greatly appreciated for their support to achieve my goal of studying in Georgia State University. Without their love and support, I could not have gotten this far. I am so grateful and fortunate to have all of you as the light of my life!

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

DNA microarray is a new and powerful biotechnology which can simultaneously monitor the expressions of thousands of genes in different cells. It has been widely applied in biological and medical research to investigate a wide range of problems in different fields including tumor diagnosis, genetic disease, bacterial infection, and so on (Alizadeh, et al., 2000; Alon, et al., 1999; Boldrick, et al., 2002; Golub, et al., 1999; Perou, et al., 1999; Pollack, et al., 1999; Ross, et al., 2000). Depending on the experimental design, it can monitor the gene expression profiles from different samples or from same sample at different developmental stage. The microarray experiment is designed to identify the differentially expressed genes, *i.e.*, genes whose expression levels are associated with a response or covariate of interest, which can be either polytomous or continuous as described above.

In microarray experiments, thousands of hypotheses need to be tested simultaneously. So the biological question can be treated as a problem in multiple hypothesis testing. For statistical hypothesis testing, two types of errors can be used to describe the possible statistical errors made in a statistical testing procedure: Type I error, which is committed by claiming an association when it isn't, and Type II error, which is committed when the test fails to identify a true association. In the case of DNA microarray experiment, when many hypotheses are tested with each has a specified Type I error probability, the Type I errors will increase sharply with the number of hypotheses. Therefore, some generalizations of the type I error rates to the multiple hypothesis testing have been adapted to analyze DNA microarray data.

To analyze the DNA microarray data, a standard approach involves two aspects: firstly computing a test statistic for each gene; secondly, applying a multiple testing procedure to decide

which hypotheses to reject when controlling an appropriate type I error rate (Dudoit, et al., 2002; Golub, et al., 1999; Kerr, et al., 2000; Manduchi, et al., 2000; Tusher, et al., 2001). For computing the test statistic for each single gene, a variety of statistical tests has been applied according to the different experimental designs. For example, an F-statistic can be chosen for categorical responses, and the Cox proportional hazard model can be considered for survival data. For the second aspect, the key point is how to control an appropriate type I error rate.

When testing simultaneously m null hypotheses H_j, $j = 1,\ldots,m$, the number of rejected hypotheses is denote by *R*. Table 1.1 summarizes the situation (Benjamini and Hochberg, 1995). The number of total hypotheses m is known in advance, the numbers of true null hypotheses m_0 and false null hypotheses $m_1 = m - m_0$ are unknown, R is an observable random variable, and S, T, U and V are unobservable random variables. In general, one would like to minimize the number V of type I errors, and the number T of type II errors.

Table 1.1 Summary of the multiple hypothesis testing

Number of hypotheses	**Number not rejected**	**Number rejected**	
True null hypotheses	U	V	m_0
False null hypotheses	T	S	m_1
	m-R	R	m

The most standard generalizations of type I errors to the multiple testing are as follows (Shaffer, 1995):

- Per-comparison error rate (PCER), which is defined as the expected value of the number of Type I errors divided by the number of hypotheses,

$$PCER = E(V)/m$$

- Per-family error rate (PFER), which is defined as the expected number of type I errors,

$$PFER = E(V)$$

- False discovery rate (FDR), which is the expected proportion of type I errors among the rejected hypothesis,

$$FDR = E(Q), \text{ where } Q = \begin{cases} V/R, & if\ R > 0, \\ 0, & if\ R = 0. \end{cases}$$

- Family-wise error rate (FWER), which is the probability of at least one type I error,

$$FWER = pr(V \geq 1)$$

Among the multiple testing procedures that control a given Type I error rate at an acceptable level, one seeks procedures that maximize power, that is, minimize a suitably defined type II error rate. The concept of power can also be generalized in various ways when moving from single to multiple hypothesis testing. Three common definitions of power are (Dudoit, et al., 2003):

- The probability of rejecting at least one false null hypothesis, $pr(S \geq 1) = pr(T \leq m_1 - 1)$
- The average power which is the probability of rejecting the false null hypotheses, $E(S)/m_1$
- The probability of rejecting all false null hypotheses, $pr(S = m_1) = pr(T = 0)$ (Shaffer, 1995).

CHAPTER 2

METHODS FOR GENE EXPRESSION DATA ANALYSIS BY GENE SETS

A microarray experiment typically contains a long list of differentially expressed genes. The gene list is usually only the start point of the analysis of a complicated process of interpretation, in which the biologist will search for patterns in the differential expression. It is easier to analyze and interpret if the list of differentially expressed genes shares similarity in certain specific properties.

In recent years, the level of microarray data analysis has shifted from single genes to sets of related genes by grouping all genes according to their biological categories together into sets and analyzing the result of the microarray experiment in terms of these sets. Analyzing microarray data by gene sets allows biologists to link previously accumulated biological knowledge in the analysis and make a more meaningful interpretation easier. The annotation terms are usually obtained from databases such as Gene Ontology (Ashburner, et al., 2000) or KEGG (Ogata, et al., 1999). The sets of genes in this type of analysis are always created prior to the analysis, and are constructed irrelevant to the data.

Among different methods proposed for testing differential expression of a gene set with a single test, there are quite some methods derived from the most popular one, which starts from the list of differentially expressed genes and tests whether the gene set is overrepresented in this list using a test for independence in a 2 x 2 (contingency) table (Al-Shahrour, et al., 2004; Beissbarth and Speed, 2004; Boyle, et al., 2004; Hosack, et al., 2003; Lee, et al., 2005; Pehkonen, et al., 2005; Yi, et al., 2006; Zeeberg, et al., 2003; Zhang, et al., 2004).

To overcome the limitation that it requires a strict cut-off for differential expression of individual genes for the above method, alternative methods were proposed to use the whole

vector of *P*-values, or test simultaneously at many cut-off values (Al-Shahrour, et al., 2005; Breitling, et al., 2004). Mootha *et al.* tested whether the ranks of the *P*-values of the genes in the gene set differ from a uniform distribution, using a weighted Kolmogorov–Smirnov test (Mootha, et al., 2003; Subramanian, et al., 2005). Pavlidis *et al.* used a test based on the geometric mean of the *P*-values of the genes in the gene set (Pavlidis, et al., 2004).

A group of very different approaches starts the analysis from the raw expression data instead of the *P*-values per gene. Goeman *et al.* used a logistic regression model to test whether subjects with similar gene expression profiles have similar class labels (Goeman, et al., 2005; Goeman, et al., 2004). At the same time, Mansmann and Meister adopted an ANOVA model to test whether subjects with similar class labels have similar expression profiles (Mansmann and Meister, 2005). Tomfohr *et al.* used a *t*-test after reducing the gene set to its first principal component (Tomfohr, et al., 2005).

The above statistical tests proposed for analyzing microarray data in terms of gene sets are different from each other in some fundamental methodologies. Two major methodological issues are the definition of the null hypothesis and the calculation of the *P*-value (Goeman and Buhlmann, 2007).

Considering the definition of the null hypothesis, the test can be a competitive one if the expression of the gene set is compared to a standard defined by the complement of that gene set. Or it can be self-contained if the gene set is compared to a fixed standard that does not depend on the complement of genes outside the gene set. Because the self-contained null hypothesis is more restrict than the competitive null hypothesis, a test based on the self-contained null hypothesis will have higher power than one based on the competitive null; however, it is criticized to be too powerful. When there are many differentially expressed genes, almost all gene sets may be

significant. Therefore, the competitive test is more popular than the self-contained test (Goeman and Buhlmann, 2007).

On the other hand, the interpretation of a *P*-value almost depends on the sampling scheme on which the test is based.

The usual statistical setup uses the subject-sampling design. In the case of microarray data, it means the experiment is repeated for new subjects with the same genes. And it assumes that the gene expressions of the different subjects are independent and identically distributed, but they can be correlated within the same subject. Therefore, the interpretation of the *P*-value is related to the true biological replications to the new subjects and a significant *P*-value will indicate the same associations will be found for a new sample of subjects (Goeman and Buhlmann, 2007).

There are other methods based on the gene-sampling urn model. They fill the 2 x 2 (contingency) table with a sample of genes drawn randomly from a big urn of genes. Each gene is subjected to the two measurements. The first one indicates whether the gene is in the gene set or not, the second one indicates whether it is in the list of differentially expressed genes. The statistical test is then performed with the assumption that the measurements of all the different genes are all independent and identically distributed. A significant *P*-value suggests a similar association between the above two measurements (Goeman and Buhlmann, 2007). There are some problems for the gene-sampling method. Most important of all, it does not mimic the biological experiment set up and therefore difficult to interpret the *P*-value in a biological sense. And as it is undoubted that strong correlations exist among gene expressions, the independence assumption is highly unrealistic (Goeman and Buhlmann, 2007).

CHAPTER 3

GENE SET ENRICHMENT ANALYSIS METHODOLOGY

3.1 Overview of GSEA Scheme

GSEA considers experiments with expression profiles from samples belonging to two classes (class label: A or B). Given an *a priori* defined set of genes *S*, the goal of GSEA is to determine whether the members of *S* are randomly distributed throughout *L*, which is the ranked gene list according to the difference in expression for each class distinction by using any suitable metric, for example, signal-to-noise ratio (SNR) in their demonstration. The sets related to the class distinction are expected to show the pattern that members are clustered at the top or bottom of *L*. The relative location of a specific gene set can be reflected by a normalized Kolmogorov-Smirnov (K-S) running sum statistic, which is calculated by walking down the list *L* to determine whether the genes are enriched at the top of *L* for the specific gene set S. The maximum positive deviation of the running sum from zero for S across the whole list is recorded and defined as Enrichment Score (ES). At the same time, the maximum ES (MES) over all gene sets is recorded as well. Then the whole procedure is repeated by permutation of the subjects to build a histogram of the MES achieved by any gene set in a given permutation. The global *P*-value is obtained by comparing the MES in the actual data to the MES histogram and used for assessing whether any gene set is associated with the class label. And for each gene set, the *P*-value is obtained by comparing its ES in the actual data to the MES histogram (Mootha, et al., 2003).

There is a new version of GSEA which modifies the preliminary version in three aspects. In the original GSEA method, the running-sum statistic used equal weight for every step, which led to the potential problem that a gene set would have a high ES score if it clustered near the middle of the ranked gene list *L* while it would not be biological significant to the class label. In

the new version, the running-sum statistic is calculated by weighting every step according to the correlation of each gene with a class label, which can efficiently avoid the above problem. In the new GSEA, for each gene set, the *ES* for each permuted data is computed to generate a null distribution for the *ES.* The empirical, nominal *P* value of the *ES* of the actual data is then calculated relative to this null distribution. In the old GSEA method, the MES for the permutated data was used to generate a null distribution for the *ES*. When testing the entire database of gene sets, the new GSEA method uses the false discovery rate (FDR) to control the type I error as the adjustment for multiple hypothesis testing while the original method used familywise-error rate (FWER) to correct for multiple hypotheses testing which turned out to so conservative that many applications yielded no statistically significant results. Therefore, the FDR is used in the new GSEA method to focus on controlling the probability that each reported result is a false positive (Subramanian, et al., 2005).

3.2 Mathematical Description of the Methods

3.2.1 Rational of Gene Set Analysis

Consider a DNA microarray data set with samples in two classes, A and B. The size of A and B are both *n*. A gene set S shows different expression levels between samples of A and B. The data set can be modeled such that the entry D_{ij} for gene *i* and sample *j* is normally distributed with mean μ_{ij} and standard deviation σ, where

$$\mu_{ij} = \begin{cases} 0, & i \notin S \\ +\alpha, & i \in S,\ j \in A \\ -\alpha, & i \in S,\ j \in B \end{cases}$$.(Subramanian, et al., 2005)

The SNR for a single gene is proportional to $\alpha\sqrt{n}/\sigma$. If we know S and add the expression levels for all M genes in S, the SNR is proportional to $\alpha\sqrt{nM}/\sigma$.

3.2.2 Gene Set Enrichment Analysis (GSEA)

GSEA starts with a ranked gene list *L* according to the difference in expression for each class distinction by using any suitable metric, for example, signal-to-noise ratio (SNR) difference in their demonstration. The SNR difference is the difference in mean of the two classes divided by the sum of the standard deviation of the two classes. And the total N genes in the data set are rank-ordered to form $L = \{g_1, \ldots, g_N\}$ according to the correlation, $r(g_i) = r_i$ of their expression profiles with the class labels, which is exactly the obvsered SNR for all the genes in the gene list.

The fraction of genes in S ("hit") weighted by their correlation up to a given position *i* in L can be computed as

$$P_{hit}(S,i) = \sum_{\substack{g_j \in S \\ j \le i}} \frac{\left|r_j\right|^p}{N_R}, \quad \text{where} \quad N_R = \sum_{g_j \in S} \left|r_j\right|^p \quad \text{and } p \text{ is the weight indicator.}$$

The fraction of genes not in S ("misses") present up to *i* in L is given by

$$P_{miss}(S,i) = \sum_{\substack{g_j \notin S \\ j \le i}} \frac{1}{N - N_H},$$

where N is the number of genes in L and N_H is the number of genes in the gene set S.

The ES is the maximum deviation from zero of P_{hit}-P_{miss}. ES(S) will be comparably small if S is randomly distributed in L, but it will be high if S is clustered at the top or bottom of the list. When p=0, ES(S) reduces to the standard Kolmogorov-Smirnov statistic which is used in the original GSEA method; when p=1, the genes in S is weighted by their correlation with the class labels normalized by the sum of the correlation over all the genes in S, which is adopted in the new GSEA method (Subramanian, et al., 2005).

3.2.3 Estimating Significance

The significance of an observed ES is assessed by comparing it to the set of scores ES_{NULL} computed with randomly assigned classes. The procedure starts from the subject-sampling of the class labels, then reorders the gene list L according to the new SNR, and re-computes ES(S). Repeat the procedure κ (e.g. κ =1000) times to generate a histogram of the ES_{NULL}. The nominal P-value for S from ES_{NULL} can be estimated by using the positive or negative portion of the distribution of ES_{NULL} according to the sign of the observed ES(S) (Subramanian, et al., 2005).

3.2.4 Multiple Hypothesis Testing

Firstly the ES(S) is computed and recorded for each gene set in the database. Then, for each S and 1000 fixed permutations π of the class labels, the ranked gene list L is reordered and ES(S, π) is determined. The ES(S, π) and ES(S) are normalized by dividing by the mean of the ES(S, π) to get the normalized scores NES(S, π) and NES(S). Now the FDR q-value, which was introduced in Chapter 1, can be computed by using a histogram of all NES(S, π) over all S and π as the null distribution. For a given $NES(S) = NES^* \geq 0$, the FDR is the ratio of the percentage of all (S, π) with $NES(S,\pi) \geq 0$, whose $NES(S) \geq NES^*$, divided by the percentage of observed S with $NES(S) \geq 0$, whose $NES(S) \geq NES^*$. When $NES(S) = NES^* \leq 0$, the FDR can be computed in a similar way (Subramanian, et al., 2005).

CHAPTER 4

REAL EXAMPLE: MALE VS. FEMALE LYMPHOBLASTOID CELLS

To demonstrate the power of the new GSEA method, I repeated the analysis of the lymphoblastoid cells dataset because it was a simple and clear demonstration, which has been analyzed by Subramanian, *et al.* using GSEA method (Subramanian, et al., 2005). The total mRNA samples were prepared from the lymphoblastoid cell lines from 15 males and 17 females. All the samples were subjected to the DNA microarray experiment to generate the expression profiles, which were used as the initial data set for the analysis. In this experiment, we wanted to identify gene sets that are associated with the gender distinctions, which are specifically highly expressed in female or male samples.

Mootha *et al.* have created a gene set database MSigDB 1.0 which contains a total number of 1325 gene sets (Subramanian, et al., 2005). The database consists of four types of gene sets. The first type of gene sets are cytogenetic sets called C_1. It contain 319 gene sets including 24 gene sets, corresponding to each of 24 chromosomes of the human genome, and 295 gene sets corresponding to different cytogenetic bands in the human genome. The second type is functional sets called C_2, which includes 472 gene sets involved in different metabolic or signaling pathways. The third type is the regulatory-motif sets called C_3. This set contains 57 gene sets for 57 different conserved regulatory motifs in the promoter regions of human genome. The last one is the neighborhood sets C_4, which consists of 427 gene sets for expression neighborhoods centered on the known cancer-related genes.

To analyze the lymphoblastoid cells dataset, we chose to test the cytogenetic sets C_1 and expected to pick up the gene sets located on the sexual chromosome because the interested response is the gender. To be more efficient, the GSEA method tested the gene sets with at least

15 genes. The gene sets were identified to be differentially expressed between male and female samples if they yield a FDR *q*-value less than or equal to 0.25 (Subramanian, et al., 2005). For the gene sets specifically highly expressed in male samples, there were 3 gene sets satisfied the screening criteria: the Y chromosome, and the two Y chromosome bands. For the gene sets specifically highly expressed in female samples, no gene sets were able to meet the criteria. (Table 4.1 A and B) The result fits well to the biological knowledge and can be easily interpreted in a biological manner. It is undoubtedly that the genes on the sex chromosome Y expressed differently between males and females because females do not have chromosome Y. For the genes on chromosome X, as most of genes on chromosome X are involved in dosage compensation, normally they are not highly expressed. Therefore, the GSEA method could effectively identify the gene sets that are associated to the response of intests.

Table 4.1 The gene sets with top ranked NES for lymphoblastoid cells data set

A. Male vs. female.

Gene Set	Size	Source	ES	NES	NOM p-val	FDR q-val	FWER p-val	Tag %	Gene %	Signal	FDR (medium)	global p-val
chrY	**40**	**Chromosome Y**	**-0.78**	**-2.37**	**0**	**0**	**0**	**0.48**	**0.10**	**0.43**	**0**	**0**
chrYp11	**18**	**Cytogenetic band**	**-0.76**	**-2.14**	**0**	**0.00**	**0.00**	**0.50**	**0.10**	**0.45**	**0**	**0**
chrYq11	**16**	**Cytogenetic band**	**-0.89**	**-2.13**	**0**	**0.00**	**0.00**	**0.63**	**0.05**	**0.59**	**0**	**0**
chr9q21	35	Cytogenetic band	-0.43	-1.53	0.04	0.63	0.66	0.31	0.18	0.26	0.49	0.24
chr6q24	17	Cytogenetic band	-0.53	-1.44	0.09	1	0.77	0.41	0.19	0.33	0.82	0.43
chr13q13	22	Cytogenetic band	-0.45	-1.40	0.09	1	0.81	0.27	0.12	0.24	0.86	0.44
chr11q22	29	Cytogenetic band	-0.44	-1.39	0.09	1	0.81	0.28	0.15	0.23	0.82	0.42
chr2q32	22	Cytogenetic band	-0.46	-1.33	0.14	1	0.86	0.23	0.08	0.21	1	0.53
chr1q24	33	Cytogenetic band	-0.39	-1.31	0.11	1	0.87	0.49	0.25	0.36	1	0.51
chr5p13	28	Cytogenetic band	-0.40	-1.31	0.15	1	0.87	0.29	0.16	0.24	0.94	0.48

B. Female vs. male

Gene Set	Size	Source	ES	NES	NOM p-val	FDR q-val	FWER p-val	Tag %	Gene %	Signal	FDR (medium)	global p-val
chrXp22	76	Cytogenetic band	0.39	1.54	0.01	1	0.65	0.18	0.05	0.18	1	0.57
chr6q15	18	Cytogenetic band	0.55	1.53	0.05	1	0.66	0.33	0.12	0.29	0.77	0.43
chr8q11	15	Cytogenetic band	0.56	1.50	0.06	0.91	0.70	0.67	0.28	0.48	0.68	0.37
chr8p11	21	Cytogenetic band	0.49	1.50	0.06	0.69	0.70	0.38	0.15	0.32	0.52	0.28
chr12q23	29	Cytogenetic band	0.49	1.48	0.06	0.65	0.73	0.45	0.17	0.37	0.49	0.26
chr13q14	47	Cytogenetic band	0.38	1.31	0.14	1	0.87	0.34	0.20	0.27	1	0.62
chrXq23	18	Cytogenetic band	0.48	1.29	0.17	1	0.88	0.22	0.05	0.21	1	0.58
chr10q11	27	Cytogenetic band	0.37	1.26	0.17	1	0.89	0.37	0.18	0.30	1	0.59
chr2q31	41	Cytogenetic band	0.35	1.26	0.16	1	0.89	0.54	0.27	0.40	1	0.55
chr4q22	21	Cytogenetic band	0.45	1.24	0.19	1	0.89	0.29	0.09	0.26	1	0.54

CHAPTER 5

SIMULATION STUDY

5.1 Introduction

Simulations are conducted to assess the performance of the GSEA method. I set up the structure of the simulated data set by mimicking the lymphoblastoid cells real dataset which I showed the analysis in Chapter 4. The whole data set file contains a total of 15,056 genes and 32 samples with 16 samples for each of the two class labels. I also created my own gene set database. It contains 20 gene sets. Each gene set has 50 genes and they are created by grouping every 50 genes from the 1^{st} to the 1000^{th} position sequentially of the name list for the lymphoblastoid cells dataset. For the samples of class one, the expression data of all the genes were generated from the standard normal distribution N (0, 1). For the samples of class two, the expression data for genes belonging to one single predefined specific gene set S were generated from an alternative distribution while the data for the remaining genes were generated from N (0, 1) as well. Under this condition, all the genes not belonging to S would have the same null distribution of N (0, 1) regardless of the class label, but the genes belonging to S would have N (0, 1) as the null distribution for class one, and an alternative distribution for class two. Therefore, the experiment design aims to test the power of GSEA to identify the differentially expressed gene set S. Different alternative distributions were considered in the study, and I will show the result in detail in the following sections.

5.2 Comparison of Normal Distributions

Different simple conditions were considered for the expression data of the differentially expressed genes. For the genes in the gene set S, the expression data of class one were generated from N (0, 1), but those of class two were generated from different normal distributions N (μ, 1)

which are different in the mean but have the same variance of 1. The simulation was repeated 1000 times for the evaluation of the statistical power for each comparison. As can be seen from Fugure 5.2.1 and Table 5.2.1, the power is increasing approximately in a linear manner when the difference is bigger between the means of the null and the alternative distributions. When the mean difference is greater than 0.2, the gene set S can be correctly identified almost every time by GSEA that it is from a different distribution from the null one. When the mean difference is less than 0.05, GSEA, in most cases, won't be able to tell that data for this gene set of the two classes are from different distribution and will treat them as from the same population. The power of 0.5 was reached when the mean difference is around 0.1, which suggests that the chance is half that GSEA can identify the different populations.

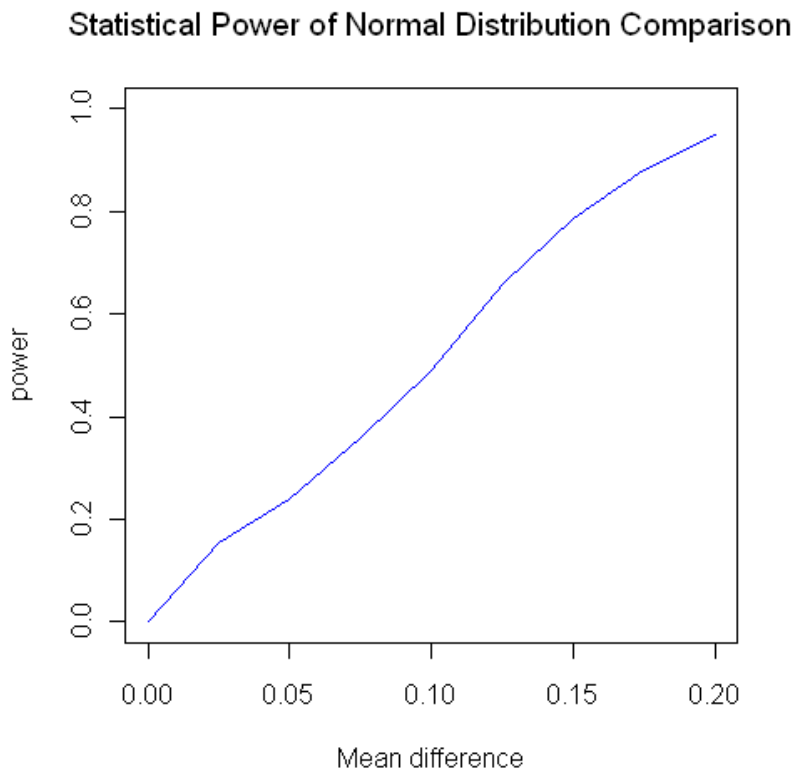

Figure 5.2.1 Plot of statistical power of the normal distribution comparison.

Table 5.2.1 Summary of the statistical power obtained from simulation studies

Null distribution Vs. Alternative distribution	Mean difference	Power
N (0, 1) Vs. N (0.2, 1)	0.2	0.948
N (0, 1) Vs. N (0.175, 1)	0.175	0.882
N (0, 1) Vs. N (0.15, 1)	0.15	0.786
N (0, 1) Vs. N (0.125, 1)	0.125	0.658
N (0, 1) Vs. N (0.1, 1)	**0.1**	**0.49**
N (0, 1) Vs. N (0.075, 1)	0.075	0.36
N (0, 1) Vs. N (0.05, 1)	0.05	0.24
N (0, 1) Vs. N (0.025, 1)	0.025	0.155

5.3 Mixture Distribution as the Alternative

In order to generalize the comparison of the expression data between the two classes, we tested different mixture distributions as the alternative. All the distributions tested were the mixture distribution of the normal distributions and the uniform distribution with the weight of 0.5. The seed normal distributions were chosen to be N (0.2, 1), N (0.1, 1) or N (0, 1).

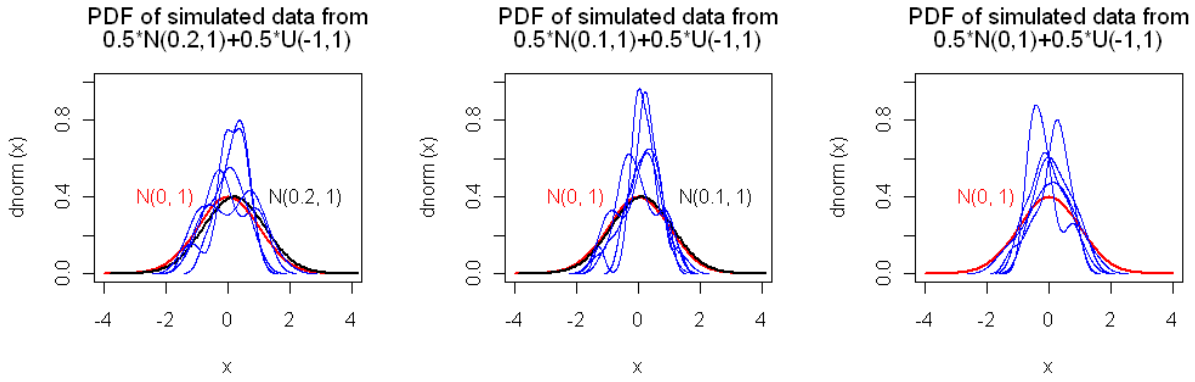

Figure 5.3.1 The empirical probability density plots of the simulated data from different mixture distributions. The curves are shown in blue color. The probability density plot for the seed normal distributions and the null distribution are shown in black and red color as the reference.

Figure 5.3.1 showed the differences between the original seed distributions and the mixture distributions. It presented the empirical probability density plot for 10000 random numbers generated from the mixture distributions. The seed normal distributions and the null distribution were also presented as the reference to show difference between them. We can see that the mixture

distributions generally showed sharper peaks than the seed distributions and most of them have two peaks. The positions of the mixture distribution peaks are near the peaks of the seed distribution which can be shifted to the left or right side of them.

Table 5.3.1 summarized the results for the power study of the mixture distributions as the alternative hypothesis under these three conditions. The highest power was obtained for the mixture distribution using N (0.2, 1) as the seed normal distribution. The power for the mixture distribution of N (0.1, 1) was in the middle. And the power was very low with the N (0, 1) seed normal distribution. The power study of the mixture distributions as the alternative showed similar pattern to previous simple comparison. Moreover, the powers did not differ much to those for the seed normal distributions as alternatives, which were shown in Table 5.2.1. This could be explained by the fact that the generated data from the mixture distributions can be closer or farther to the null distribution compared to the seed normal distribution, which is shown in Figure 5.3.1. According to our previously showed result that the power is lower for closer alternative distribution and higher for farther one, the changes of the power would be mostly canceled out with the alternative hypothesis which is a collection of both kind.

Table 5.3.1 The estimated power for the mixture distribution as the alternative distribution.

Null distribution Vs. Alternative distribution	**Seed distribution**	**Power**
N (0, 1) vs. 0.5N (0, 1)+0.5U (-1, 1)	N (0, 1)	0.113
N (0, 1) vs. 0.5N (0.1, 1)+0.5U (-1, 1)	N (0.1, 1)	0.43
N (0, 1) vs. 0.5N (0.2, 1)+0.5U (-1, 1)	N (0.2, 1)	0.933

5.4 Neighborhood of a Distribution as the Alternative

To further generalize the comparison of the expression data between the two classes, we tested different distribution neighborhoods as the alternatives. A statistical neighborhood can be built about a distribution F by including all distribution P that satisfies $\rho(P, F) \le c$, where ρ is a

statistical distance measurement and *c* is some specified level of tolerance. In my case, the Kolmogorov distance was selected to build the neighborhood about the seed normal distribution. A sample was drawn from the neighborhood distributions of different tolerance level. To simplify the simulation procedure, we considered a distribution that has the mixture form but is restricted within the neighborhood of the null distribution.

In the previous section, we have showed that the power for the mixture distribution as alternative of different normal distributions with U (-1, 1) did not exhibit big difference compared to the simple normal distribution comparison. So we started to consider the one sided situation by mixing with the U (-1, 0) so that the neighborhood distributions were closer to the null distribution compared to the original seed normal distribution.

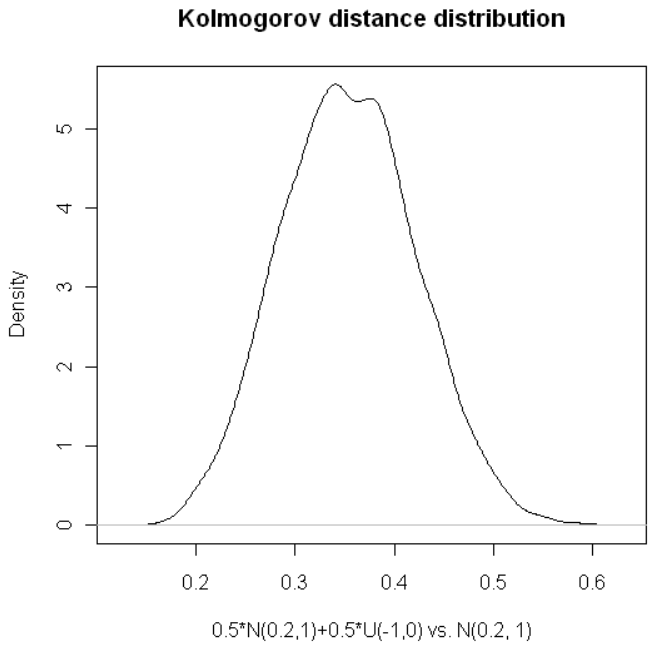

Figure 5.4.1 The empirical probability density of the Kolmogorov distance between 0.5×N (0.2, 1) + 0.5×U (-1, 0) and N (0.2, 1).

First we considered the mixture distribution of 0.5×N (0.2, 1) + 0.5× U (-1, 0) to simulate the distribution neighborhood. Figure 5.4.1 showed that the empirical Kolmogorov distance

distribution, which is centered on about 0.35 and ranges from 0 to 0.6. Figure 5.4.2 showed the neighborhood distributions exhibited different variations to the original seed distribution of N (0.2, 1) in the probability plots.

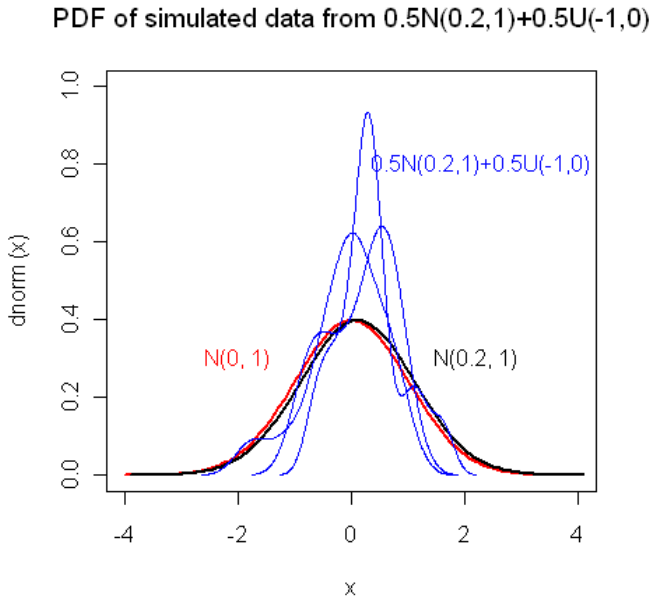

Figure 5.4.2 The empirical probability density plots for 0.5×N (0.2, 1) + 0.5×U (-1, 0) with the Kolmogorov distance to N (0.2, 1) between 0.2 and 0.25. The plots are in blue color. Three plots were generated from three series of random numbers to represent the diversity of the neighborhood distribution. The probability density plots for N (0, 1) and N (0.2, 1) were shown in black and red color as the reference.

Table 5.4.1 The estimated power for the distribution neighborhood (0.5N (0.2, 1)+0.5 U(-1, 0))

Kolmogorov distance (Data vs. N (0.2, 1))	**Power**
<0.15	0.84
0.15-0.2	0.49
0.2-0.25	0.14
0.25-0.3	0.19
0.3-0.35	0.71
>0.35	1

The powers for the differences neighborhoods were summarized in Table 5.4.1. Because the neighborhood data were simulated from the mixture distribution of N (0.2, 1) and U (-1, 0), the

Kolmogorov distances were controlled against N (0.2, 1). The power of the GSEA test was not monotonely correlated with the Kolmogorov distances of the neighborhood distribution. The result was conceivable because they were not the two distributions that were compared to in the statistical hypothesis testing. The correlation was expected between the power and the distance between the alternative and the null distributions because these were the two distributions that were compared to in the statistical hypothesis testing. The distances between the alternative and neighborhood distributions may only indirectly reflect those between the alternative and null distributions.

In order to confirm that the result we have observed is a common pattern, we have done the tests again by using the neighborhood simulated from the same mixture distribution but changing the weight from 0.5 to 0.8. Figure 5.4.3 showed that the empirical Kolmogorov distance distribution was centered on 0.2 and ranged from 0 to 0.55. Figure 5.4.4 presented the diversity of the neighborhood distributions.

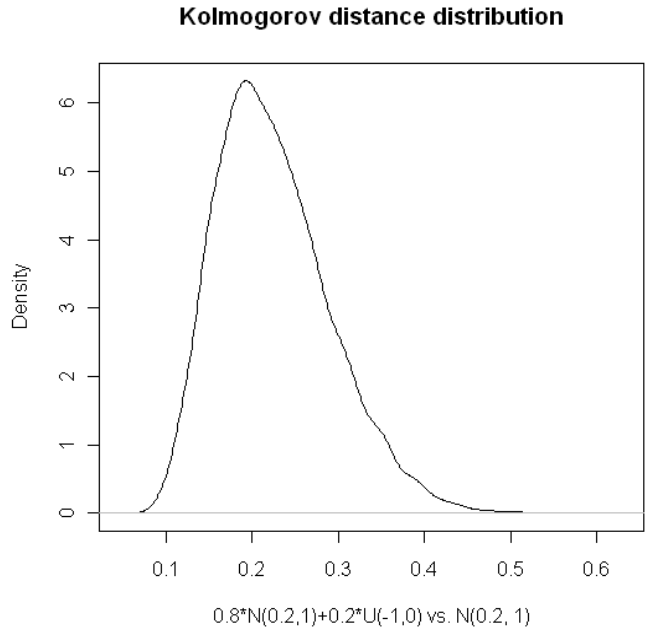

Figure 5.4.3 The empirical probability density of the Kolmogorov distance between 0.8×N (0.2, 1) + 0.2×U (-1, 0) and N (0.2, 1).

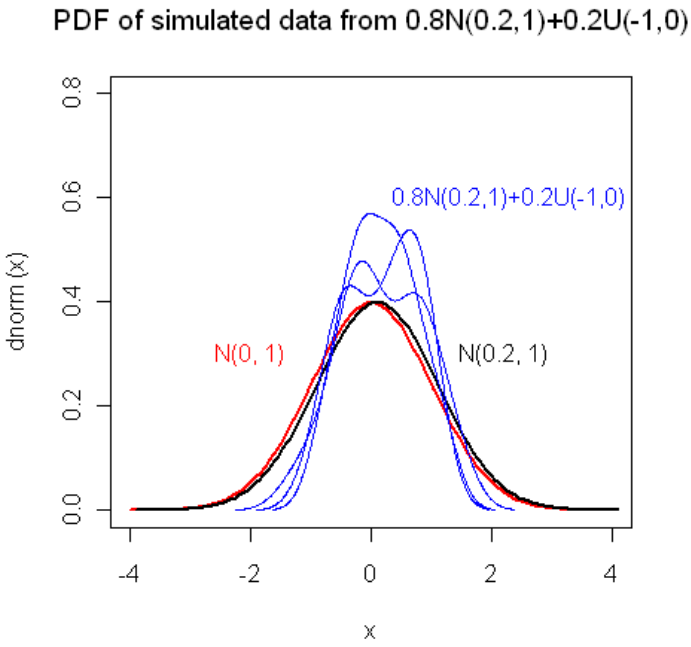

Figure 5.4.4 The empirical probability density plots for 0.8×N (0.2, 1) + 0.2×U (-1, 0) with the Kolmogorov distance to N (0.2, 1) between 0.15 and 0.25. The plots are shown in blue color. Three plots were generated from three series of random numbers to represent the diversity of the distribution. The probability density plots for N (0, 1) and N (0.2, 1) were shown in black and red color as the reference.

Table 5.4.2 The estimated power for the distribution neighborhood (0.8N (0.2, 1)+0.2 U(-1, 0)).

Kolmogorov distance (Data vs. N (0.2, 1))	**Power**
<0.15	0.926
0.15-0.2	0.853
0.2-0.25	0.45
0.25-0.3	0.148
0.3-0.35	0.595
>0.35	1

Table 5.4.2 summarized the results for the power estimation. It revealed the simlar pattern as the previous results. Nevertheless, the information in Table 5.4.1 and 5.4.2 was not sufficient to illustrate the real statistical hypothesis testing conditions, so we check the two empirical probability density functions of the two Kolmogorov distances. The result was shown in Figure 5.4.5, which

showed clearly the positive correlation between the power of GSEA and the Kolmogorov distance between the neighborhood and the null distributions. For both cases studied above, the power of GSEA decreased as the location of the peak for the probability density plot shifted to the left side; and the power increased as the location of the peak shifted to the right side. In addition, it worthes knowing that the lowest power for both cases were similar (0.14 and 0.148 respectively), and both they were both obtained with the peaks located very close to one another for the two conditions (near 0.2).

It is easy to explain the relationshop bewteen the power and two Kolmogorov distances if we focus on the Kolmogorov distance between the neighborhood and the null distributions because these are the two distributions used in the statistical hypothesis testing. Therefore, the Kolmogorov distance between the neighborhood and the seed distributions can only indirectly reflect the Kolmogorov distance between the neighborhood and the null distributions. At first, when the neighborhood distribution shifts toward the left side of the seed distribution of N (0.2, 1) as the Kolmogorov distance between the neighborhood and the seed normal distribution increases, the Kolmogorov distance decreases between the neighborhood and the null distributions, therefore the power of the test is dropped until the means of the two distributions equal to each other. But when the neighborhood distribution moves further, the Kolmogorov distance begins to increase between the neighborhood and the null distributions, and the power of the test will go up again as the mean difference of the two distributions increases. Figure 5.4.5 showed exactly the trend of the changes of the Kolmogorov distances.

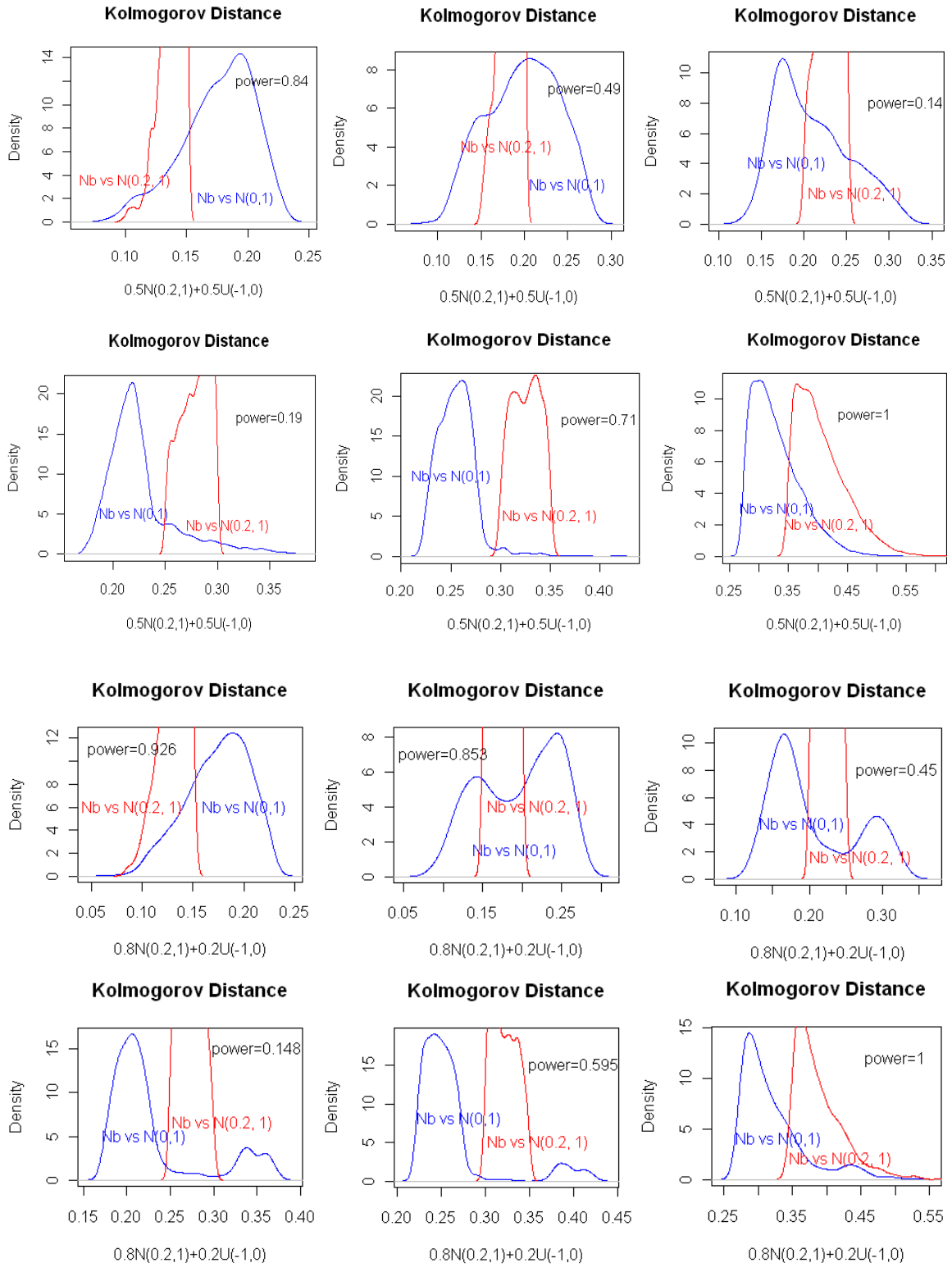

Figure 5.4.5 The empirical probability density plots of the Kolmogorov distances between the simulated neighborhood and the null distribution or the seed normal distribution. The

neighborhood were simulated from the mixture distribution of 0.5×N (0.2, 1) + 0.5×U (-1, 0) with different Kolmogorov distance ranges to N (0.2, 1). The plots are shown in blue color for the Kolmogorov distances between the neighborhood and the null distributions. The plots are shown in red color for the Kolmogorov distances between the neighborhood and the seed distributions. Fifity plots were generated from fifty series of random numbers, and they are shown to represent the neighborhood distribution population. The probability density plot for N (0.2, 1) was shown in red color as the reference. Upper six plots: the neighborhood were simulated from the mixture distribution of 0.5×N (0.2, 1) + 0.5×U (-1, 0); Lower six plots: the neighborhood were simulated from the mixture distribution of 0.8×N (0.2, 1) + 0.2×U (-1, 0).

CHAPTER 6

CONCLUSIONS AND DISCUSSION

GSEA is a computational method that determines whether an a priori defined set of genes shows statistically significant, concordant differences between two biological states from the Genechip microarray expression profiles. GSEA is claimed to have relatively low power because the suitable FDR threshold is set to 0.25. The main purpose of this thesis is to evaluate the power of GSEA method.

All the study including the real example and simulations in this thesis was done with FDR is controlled at 0.25. With this value, the statistical power of the GSEA method is high enough to identify small differences between different distributions. In the real data analysis in Chapter 4, the GSEA method effectively picked up the responsible gene sets. And in the simulation studied shown in Chapter 5, the GSEA method could differentiate the two distributions of N (0, 1) and N (0.2, 1) to a power of 0.95. When the two distributions get closer to N (0, 1) and N (0.1, 1), GSEA still can maintain the power at the level of 0.5, which means GSEA can pick the right gene up at half of the chance. When we generalize the alternative distribution of normal distribution to its neighborhood distribution, GSEA still kept the power to a satisfactory level, and did not show dramatically decrease. Under this condition, the power of GSEA is rather determined by the distance between the neighborhood and null distributions than the neighborhood and the original seed distributions, which could be reflected from the above results that GSEA tended to lose its power only when the neighborhood distribution population moved closer to the null distribution. It is notable that when the Kolmogorov metric between 0.2 and 0.25 for $0.8\times N(0.2, 1) + 0.2\times U(-1, 0)$, the power of GSEA decreased to 0.45, which is similar to the result from the simple comparison of N (0.1, 1). It suggested that the GSEA

method cannot tell the difference between these two populations and will treat them as the similar alternative distribution.

To specifically addressing the power of the methods for DNA microarray data analysis, we need to consider the nature of the DNA microarray experiment, which aims to identify the differentially expressed genes, i.e., it favors the alternative hypothesis for the statistical hypothesis testing. In this case, controlling the power is much more important than the control of type I error. Therefore, it is conceivable to improve the power of the GSEA method by loosen the type I error control.

The GSEA has other advantages as well. Instead of focusing the analysis on identifying individual differentially expressed genes, GSEA analyzes DNA microarray data at the level of gene sets. The new strategies for DNA microarray data analysis by gene sets, which enables the statistical analysis to be integrated with biological information, can detect the biological processes in one shot, which are distributed across the entire gene network including those genes expressed only subtly different.

The original GSEA method has been shown to efficiently discover the metabolic pathways linked in human diabetes, or involved in diffuse large B cell lymphoma, and nutrient-sensing pathways involved in prostate cancer and so on. The new GSEA method is more efficient with improvement from three aspects. The ES score is computed in a way that it is strongly associated with the positions in the gene rank list of gene members of a specific gene sets. Each gene set will have its own null distribution based on the permutations. And the type I error control was changed from the conservation FWER to FDR. The new version of GSEA is more sensitive and robust, and has much broader applications. The GSEA method is very flexible in that it can test any gene set. The gene sets tested can be from the gene set database MsigDB 1.0,

or they can be created through other genomic information and additional biological annotation, or they can be several lists of genes that you are interested to analyze. In summary, the GSEA methodology is an efficient, flexible and powerful tool for the DNA microarray data analysis.

Introducing the neighborhood distribution would be a good correction for the self-contained test for the DNA microarray data analysis, which has the problem of over powerful to reject any similarity between different class labels. This problem can be overcomed by constructing a neighborhood about one class and relaxing the null hypothesis as that another class is in a close neighborhood of it. This will be investigated in our future work.

REFERENCES

Al-Shahrour, F., Diaz-Uriarte, R. and Dopazo, J. (2004) FatiGO: a web tool for finding significant associations of Gene Ontology terms with groups of genes, *Bioinformatics*, **20**, 578-580.

Al-Shahrour, F., Diaz-Uriarte, R. and Dopazo, J. (2005) Discovering molecular functions significantly related to phenotypes by combining gene expression data and biological information, *Bioinformatics*, **21**, 2988-2993.

Alizadeh, A.A., Eisen, M.B., Davis, R.E., Ma, C., Lossos, I.S., Rosenwald, A., Boldrick, J.C., Sabet, H., Tran, T., Yu, X., Powell, J.I., Yang, L., Marti, G.E., Moore, T., Hudson, J., Jr., Lu, L., Lewis, D.B., Tibshirani, R., Sherlock, G., Chan, W.C., Greiner, T.C., Weisenburger, D.D., Armitage, J.O., Warnke, R., Levy, R., Wilson, W., Grever, M.R., Byrd, J.C., Botstein, D., Brown, P.O. and Staudt, L.M. (2000) Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling, *Nature*, **403**, 503-511.

Alon, U., Barkai, N., Notterman, D.A., Gish, K., Ybarra, S., Mack, D. and Levine, A.J. (1999) Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, *Proceedings of the National Academy of Sciences of the United States of America*, **96**, 6745-6750.

Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G. and Gene Ontology, C. (2000) Gene Ontology: tool for the unification of biology, *Nature Genetics*, **25**, 25-29.

Beissbarth, T. and Speed, T.P. (2004) GOstat: find statistically overrepresented Gene Ontologies within a group of genes, *Bioinformatics*, **20**, 1464-1465.

Benjamini, Y. and Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing, *Journal of the Royal Statistical Society. Series B (Methodological)*, **57**, 289-300.

Boldrick, J.C., Alizadeh, A.A., Diehn, M., Dudoit, S., Liu, C.L., Belcher, C.E., Botstein, D., Staudt, L.M., Brown, P.O. and Relman, D.A. (2002) Stereotyped and specific gene expression programs in human innate immune responses to bacteria, *Proceedings of the National Academy of Sciences of the United States of America*, **99**, 972-977.

Boyle, E.I., Weng, S., Gollub, J., Jin, H., Botstein, D., Cherry, J.M. and Sherlock, G. (2004) GO::TermFinder--open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes, *Bioinformatics*, **20**, 3710-3715.

Breitling, R., Amtmann, A. and Herzyk, P. (2004) Iterative Group Analysis (iGA): A simple tool to enhance sensitivity and facilitate interpretation of microarray experiments, *BMC Bioinformatics*, **5**, 34.

Dudoit, S., Shaffer, J.P. and Boldrick, J.C. (2003) Multiple Hypothesis Testing in Microarray Experiments, *Statistical Science*, **18**, 71-103.

Dudoit, S., Yang, Y.H., Callow, M.J. and Speed, T.P. (2002) Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments, *Statistica Sinica*, **12**, 111-139.

Goeman, J.J. and Buhlmann, P. (2007) Analyzing gene expression data in terms of gene sets: methodological issues, *Bioinformatics*, **23**, 980-987.

Goeman, J.J., Oosting, J., Cleton-Jansen, A.-M., Anninga, J.K. and van Houwelingen, H.C. (2005) Testing association of a pathway with survival using gene expression data, *Bioinformatics*, **21**, 1950-1957.

Goeman, J.J., van de Geer, S.A., de Kort, F. and van Houwelingen, H.C. (2004) A global test for groups of genes: testing association with a clinical outcome, *Bioinformatics*, **20**, 93-99.

Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D. and Lander, E.S. (1999) Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring, *Science*, **286**, 531-537.

Hosack, D., Dennis, G., Sherman, B., Lane, H. and Lempicki, R. (2003) Identifying biological themes within lists of genes with EASE, *Genome Biology*, **4**, R70.

Kerr, M.K., Martin, M. and Churchill, G.A. (2000) Analysis of variance for gene expression microarray data, *Journal of Computational Biology*, **7**, 819-837.

Lee, H., Braynen, W., Keshav, K. and Pavlidis, P. (2005) ErmineJ: Tool for functional analysis of gene expression data sets, *BMC Bioinformatics*, **6**, 269.

Manduchi, E., Grant, G.R., McKenzie, S.E., Overton, G.C., Surrey, S. and Stoeckert, C.J. (2000) Generation of patterns from gene expression data by assigning confidence to differentially expressed genes, *Bioinformatics*, **16**, 685-698.

Mansmann, U. and Meister, R. (2005) Testing differential gene expression in functional groups. Goeman's global test versus an ANCOVA approach, *Methods Inf Med*, **44**, 449-453.

Mootha, V.K., Lindgren, C.M., Eriksson, K.F., Subramanian, A., Sihag, S., Lehar, J., Puigserver, P., Carlsson, E., Ridderstrale, M., Laurila, E., Houstis, N., Daly, M.J., Patterson, N., Mesirov, J.P., Golub, T.R., Tamayo, P., Spiegelman, B., Lander, E.S., Hirschhorn, J.N.,

Altshuler, D. and Groop, L.C. (2003) PGC-1 alpha-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes, *Nature Genetics*, **34**, 267-273.

Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H. and Kanehisa, M. (1999) KEGG: Kyoto Encyclopedia of Genes and Genomes, *Nucl. Acids Res.*, **27**, 29-34.

Pavlidis, P., Qin, J., Arango, V., Mann, J.J. and Sibille, E. (2004) Using the gene ontology for microarray data mining: A comparison of methods and application to age effects in human prefrontal cortex, *Neurochem. Res.*, **29**, 1213-1222.

Pehkonen, P., Wong, G. and Toronen, P. (2005) Theme discovery from gene lists for identification and viewing of multiple functional groups, *BMC Bioinformatics*, **6**, 162.

Perou, C.M., Jeffrey, S.S., van de Rijn, M., Rees, C.A., Eisen, M.B., Ross, D.T., Pergamenschikov, A., Williams, C.F., Zhu, S.X., Lee, J.C.F., Lashkari, D., Shalon, D., Brown, P.O. and Botstein, D. (1999) Distinctive gene expression patterns in human mammary epithelial cells and breast cancers, *Proceedings of the National Academy of Sciences of the United States of America*, **96**, 9212-9217.

Pollack, J.R., Perou, C.M., Alizadeh, A.A., Eisen, M.B., Pergamenschikov, A., Williams, C.F., Jeffrey, S.S., Botstein, D. and Brown, P.O. (1999) Genome-wide analysis of DNA copy-number changes using cDNA microarrays, *Nat Genet*, **23**, 41-46.

Ross, D.T., Scherf, U., Eisen, M.B., Perou, C.M., Rees, C., Spellman, P., Iyer, V., Jeffrey, S.S., Van de Rijn, M., Waltham, M., Pergamenschikov, A., Lee, J.C., Lashkari, D., Shalon, D., Myers, T.G., Weinstein, J.N., Botstein, D. and Brown, P.O. (2000) Systematic variation in gene expression patterns in human cancer cell lines, *Nature Genetics*, **24**, 227-235.

Shaffer, J.P. (1995) MULTIPLE HYPOTHESIS-TESTING, *Annu. Rev. Psychol.*, **46**, 561-584.

Subramanian, A., Tamayo, P., Mootha, V.K., Mukherjee, S., Ebert, B.L., Gillette, M.A., Paulovich, A., Pomeroy, S.L., Golub, T.R., Lander, E.S. and Mesirov, J.P. (2005) Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles, *Proceedings of the National Academy of Sciences of the United States of America*, **102**, 15545-15550.

Tomfohr, J., Lu, J. and Kepler, T. (2005) Pathway level analysis of gene expression using singular value decomposition, *BMC Bioinformatics*, **6**, 225.

Tusher, V.G., Tibshirani, R. and Chu, G. (2001) Significance analysis of microarrays applied to the ionizing radiation response, *Proceedings of the National Academy of Sciences of the United States of America*, **98**, 5116-5121.

Yi, M., Horton, J., Cohen, J., Hobbs, H. and Stephens, R. (2006) WholePathwayScope: a comprehensive pathway-based analysis tool for high-throughput data, *BMC Bioinformatics*, **7**, 30.

Zeeberg, B., Feng, W., Wang, G., Wang, M., Fojo, A., Sunshine, M., Narasimhan, S., Kane, D., Reinhold, W., Lababidi, S., Bussey, K., Riss, J., Barrett, J. and Weinstein, J. (2003) GoMiner: a resource for biological interpretation of genomic and proteomic data, *Genome Biology*, **4**, R28.

Zhang, B., Schmoyer, D., Kirov, S. and Snoddy, J. (2004) GOTree Machine (GOTM): a web-based platform for interpreting sets of interesting genes using Gene Ontology hierarchies, *BMC Bioinformatics*, **5**, 16.

APPENDIX A: R code for GSEA methodology

```
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.


# G S E A -- Gene Set Enrichment Analysis

# Auxiliary functions and definitions

GSEA.GeneRanking <- function(A, class.labels, gene.labels, nperm,
      permutation.type = 0, sigma.correction = "GeneCluster", fraction=1.0,
      replace=F, reverse.sign= F) {

# This function ranks the genes according to the signal to noise ratio for the
      actual phenotype and also random permutations and bootstrap
# subsamples of both the observed and random phenotypes. It uses matrix
      operations to implement the signal to noise calculation
# in stages and achieves fast execution speed. It supports two types of
      permutations: random (unbalanced) and balanced.
# It also supports subsampling and bootstrap by using masking and multiple-
      count variables.  When "fraction" is set to 1 (default)
# the there is no subsampling or boostrapping and the matrix of observed
      signal to noise ratios will have the same value for
# all permutations. This is wasteful but allows to support all the multiple
      options with the same code. Notice that the second
# matrix for the null distribution will still have the values for the random
      permutations
# (null distribution). This mode (fraction = 1.0) is the defaults, the
      recommended one and the one used in the examples.
# It is also the one that has be tested more thoroughly. The resampling and
      boostrapping options are intersting to obtain
# smooth estimates of the observed distribution but its is left for the expert
      user who may want to perform some sanity
# checks before trusting the code.
#
# Inputs:
#   A: Matrix of gene expression values (rows are genes, columns are samples)
#   class.labels: Phenotype of class disticntion of interest. A vector of
      binary labels having first the 1's and then the 0's
#   gene.labels: gene labels. Vector of probe ids or accession numbers for the
      rows of the expression matrix
#   nperm: Number of random permutations/bootstraps to perform
#   permutation.type: Permutation type: 0 = unbalanced, 1 = balanced. For
      experts only (default: 0)
#   sigma.correction: Correction to the signal to noise ratio (Default =
      GeneCluster, a choice to support the way it was handled in a previous
      package)
#   fraction: Subsampling fraction. Set to 1.0 (no resampling). For experts
      only (default: 1.0)
#   replace: Resampling mode (replacement or not replacement). For experts
      only (default: F)
#   reverse.sign: Reverse direction of gene list (default = F)
```

```
#
# Outputs:
#  s2n.matrix: Matrix with random permuted or bootstraps signal to noise
     ratios (rows are genes, columns are permutations or bootstrap
     subsamplings
#  obs.s2n.matrix: Matrix with observed signal to noise ratios (rows are
     genes, columns are boostraps subsamplings. If fraction is set to 1.0
     then all the columns have the same values
#  order.matrix: Matrix with the orderings that will sort the columns of the
     obs.s2n.matrix in decreasing s2n order
#  obs.order.matrix: Matrix with the orderings that will sort the columns of
     the s2n.matrix in decreasing s2n order
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

    A <- A + 0.00000001

    N <- length(A[,1])
    Ns <- length(A[1,])

    subset.mask <- matrix(0, nrow=Ns, ncol=nperm)
    reshuffled.class.labels1 <- matrix(0, nrow=Ns, ncol=nperm)
    reshuffled.class.labels2 <- matrix(0, nrow=Ns, ncol=nperm)
    class.labels1 <- matrix(0, nrow=Ns, ncol=nperm)
    class.labels2 <- matrix(0, nrow=Ns, ncol=nperm)

    order.matrix <- matrix(0, nrow = N, ncol = nperm)
    obs.order.matrix <- matrix(0, nrow = N, ncol = nperm)
    s2n.matrix <- matrix(0, nrow = N, ncol = nperm)
    obs.s2n.matrix <- matrix(0, nrow = N, ncol = nperm)

    obs.gene.labels <- vector(length = N, mode="character")
    obs.gene.descs <- vector(length = N, mode="character")
    obs.gene.symbols <- vector(length = N, mode="character")

    M1 <- matrix(0, nrow = N, ncol = nperm)
    M2 <- matrix(0, nrow = N, ncol = nperm)
    S1 <- matrix(0, nrow = N, ncol = nperm)
    S2 <- matrix(0, nrow = N, ncol = nperm)

    gc()

    C <- split(class.labels, class.labels)
    class1.size <- length(C[[1]])
    class2.size <- length(C[[2]])
    class1.index <- seq(1, class1.size, 1)
    class2.index <- seq(class1.size + 1, class1.size + class2.size, 1)

    for (r in 1:nperm) {
       class1.subset <- sample(class1.index, size =
      ceiling(class1.size*fraction), replace = replace)
       class2.subset <- sample(class2.index, size =
      ceiling(class2.size*fraction), replace = replace)
```

```
class1.subset.size <- length(class1.subset)
class2.subset.size <- length(class2.subset)
subset.class1 <- rep(0, class1.size)
for (i in 1:class1.size) {
    if (is.element(class1.index[i], class1.subset)) {
       subset.class1[i] <- 1
    }
}
subset.class2 <- rep(0, class2.size)
for (i in 1:class2.size) {
    if (is.element(class2.index[i], class2.subset)) {
       subset.class2[i] <- 1
    }
}
subset.mask[, r] <- as.numeric(c(subset.class1, subset.class2))
fraction.class1 <- class1.size/Ns
fraction.class2 <- class2.size/Ns

if (permutation.type == 0) { # random (unbalanced) permutation
   full.subset <- c(class1.subset, class2.subset)
   label1.subset <- sample(full.subset, size = Ns * fraction.class1)
   reshuffled.class.labels1[, r] <- rep(0, Ns)
   reshuffled.class.labels2[, r] <- rep(0, Ns)
   class.labels1[, r] <- rep(0, Ns)
   class.labels2[, r] <- rep(0, Ns)
   for (i in 1:Ns) {
      m1 <- sum(!is.na(match(label1.subset, i)))
      m2 <- sum(!is.na(match(full.subset, i)))
      reshuffled.class.labels1[i, r] <- m1
      reshuffled.class.labels2[i, r] <- m2 - m1
      if (i <= class1.size) {
        class.labels1[i, r] <- m2
        class.labels2[i, r] <- 0
      } else {
         class.labels1[i, r] <- 0
         class.labels2[i, r] <- m2
      }
   }

} else if (permutation.type == 1) { # proportional (balanced)
permutation

   class1.label1.subset <- sample(class1.subset, size =
ceiling(class1.subset.size*fraction.class1))
   class2.label1.subset <- sample(class2.subset, size =
floor(class2.subset.size*fraction.class1))
   reshuffled.class.labels1[, r] <- rep(0, Ns)
   reshuffled.class.labels2[, r] <- rep(0, Ns)
   class.labels1[, r] <- rep(0, Ns)
   class.labels2[, r] <- rep(0, Ns)
   for (i in 1:Ns) {
      if (i <= class1.size) {
         m1 <- sum(!is.na(match(class1.label1.subset, i)))
         m2 <- sum(!is.na(match(class1.subset, i)))
         reshuffled.class.labels1[i, r] <- m1
         reshuffled.class.labels2[i, r] <- m2 - m1
         class.labels1[i, r] <- m2
         class.labels2[i, r] <- 0
      } else {
         m1 <- sum(!is.na(match(class2.label1.subset, i)))
         m2 <- sum(!is.na(match(class2.subset, i)))
```

```
                reshuffled.class.labels1[i, r] <- m1
                reshuffled.class.labels2[i, r] <- m2 - m1
                class.labels1[i, r] <- 0
                class.labels2[i, r] <- m2
              }
            }
          }
        }

# compute S2N for the random permutation matrix

    P <- reshuffled.class.labels1 * subset.mask
    n1 <- sum(P[,1])
    M1 <- A %*% P
    M1 <- M1/n1
    gc()
    A2 <- A*A
    S1 <- A2 %*% P
    S1 <- S1/n1 - M1*M1
    S1 <- sqrt(abs((n1/(n1-1)) * S1))
    gc()
    P <- reshuffled.class.labels2 * subset.mask
    n2 <- sum(P[,1])
    M2 <- A %*% P
    M2 <- M2/n2
    gc()
    A2 <- A*A
    S2 <- A2 %*% P
    S2 <- S2/n2 - M2*M2
    S2 <- sqrt(abs((n2/(n2-1)) * S2))
    rm(P)
    rm(A2)
    gc()

    if (sigma.correction == "GeneCluster") {  # small sigma "fix" as used in
      GeneCluster
       S2 <- ifelse(0.2*abs(M2) < S2, S2, 0.2*abs(M2))
       S2 <- ifelse(S2 == 0, 0.2, S2)
       S1 <- ifelse(0.2*abs(M1) < S1, S1, 0.2*abs(M1))
       S1 <- ifelse(S1 == 0, 0.2, S1)
       gc()
    }

    M1 <- M1 - M2
    rm(M2)
    gc()
    S1 <- S1 + S2
    rm(S2)
    gc()

    s2n.matrix <- M1/S1

    if (reverse.sign == T) {
      s2n.matrix <- - s2n.matrix
    }
    gc()

    for (r in 1:nperm) {
      order.matrix[, r] <- order(s2n.matrix[, r], decreasing=T)
    }
```

```
# compute S2N for the "observed" permutation matrix

    P <- class.labels1 * subset.mask
    n1 <- sum(P[,1])
    M1 <- A %*% P
    M1 <- M1/n1
    gc()
    A2 <- A*A
    S1 <- A2 %*% P
    S1 <- S1/n1 - M1*M1
    S1 <- sqrt(abs((n1/(n1-1)) * S1))
    gc()
    P <- class.labels2 * subset.mask
    n2 <- sum(P[,1])
    M2 <- A %*% P
    M2 <- M2/n2
    gc()
    A2 <- A*A
    S2 <- A2 %*% P
    S2 <- S2/n2 - M2*M2
    S2 <- sqrt(abs((n2/(n2-1)) * S2))
    rm(P)
    rm(A2)
    gc()

    if (sigma.correction == "GeneCluster") {  # small sigma "fix" as used in
      GeneCluster
       S2 <- ifelse(0.2*abs(M2) < S2, S2, 0.2*abs(M2))
       S2 <- ifelse(S2 == 0, 0.2, S2)
       S1 <- ifelse(0.2*abs(M1) < S1, S1, 0.2*abs(M1))
       S1 <- ifelse(S1 == 0, 0.2, S1)
       gc()
    }

    M1 <- M1 - M2
    rm(M2)
    gc()
    S1 <- S1 + S2
    rm(S2)
    gc()

    obs.s2n.matrix <- M1/S1
    gc()

    if (reverse.sign == T) {
       obs.s2n.matrix <- - obs.s2n.matrix
    }

    for (r in 1:nperm) {
       obs.order.matrix[,r] <- order(obs.s2n.matrix[,r], decreasing=T)
    }

    return(list(s2n.matrix = s2n.matrix,
                obs.s2n.matrix = obs.s2n.matrix,
                order.matrix = order.matrix,
                obs.order.matrix = obs.order.matrix))
}

GSEA.EnrichmentScore <- function(gene.list, gene.set, weighted.score.type = 1,
      correl.vector = NULL) {
#
```

```
# Computes the weighted GSEA score of gene.set in gene.list.
# The weighted score type is the exponent of the correlation
# weight: 0 (unweighted = Kolmogorov-Smirnov), 1 (weighted), and 2 (over-
      weighted). When the score type is 1 or 2 it is
# necessary to input the correlation vector with the values in the same order
      as in the gene list.
#
# Inputs:
#  gene.list: The ordered gene list (e.g. integers indicating the original
      position in the input dataset)
#  gene.set: A gene set (e.g. integers indicating the location of those genes
      in the input dataset)
#  weighted.score.type: Type of score: weight: 0 (unweighted = Kolmogorov-
      Smirnov), 1 (weighted), and 2 (over-weighted)
#  correl.vector: A vector with the coorelations (e.g. signal to noise scores)
      corresponding to the genes in the gene list
#
# Outputs:
#  ES: Enrichment score (real number between -1 and +1)
#  arg.ES: Location in gene.list where the peak running enrichment occurs
      (peak of the "mountain")
#  RES: Numerical vector containing the running enrichment score for all
      locations in the gene list
#  tag.indicator: Binary vector indicating the location of the gene sets
      (1's) in the gene list
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

   tag.indicator <- sign(match(gene.list, gene.set, nomatch=0))   # notice
      that the sign is 0 (no tag) or 1 (tag)
   no.tag.indicator <- 1 - tag.indicator
   N <- length(gene.list)
   Nh <- length(gene.set)
   Nm <-  N - Nh
   if (weighted.score.type == 0) {
      correl.vector <- rep(1, N)
   }
   alpha <- weighted.score.type
   correl.vector <- abs(correl.vector**alpha)
   sum.correl.tag    <- sum(correl.vector[tag.indicator == 1])
   norm.tag    <- 1.0/sum.correl.tag
   norm.no.tag <- 1.0/Nm
   RES <- cumsum(tag.indicator * correl.vector * norm.tag - no.tag.indicator *
      norm.no.tag)
   max.ES <- max(RES)
   min.ES <- min(RES)
   if (max.ES > - min.ES) {
#     ES <- max.ES
      ES <- signif(max.ES, digits = 5)
      arg.ES <- which.max(RES)
   } else {
#     ES <- min.ES
      ES <- signif(min.ES, digits=5)
```

```
    arg.ES <- which.min(RES)
  }
  return(list(ES = ES, arg.ES = arg.ES, RES = RES, indicator =
      tag.indicator))
}


OLD.GSEA.EnrichmentScore <- function(gene.list, gene.set) {
#
# Computes the original GSEA score from Mootha et al 2003 of gene.set in
      gene.list
#
# Inputs:
#  gene.list: The ordered gene list (e.g. integers indicating the original
      position in the input dataset)
#  gene.set: A gene set (e.g. integers indicating the location of those genes
      in the input dataset)
#
# Outputs:
#  ES: Enrichment score (real number between -1 and +1)
#  arg.ES: Location in gene.list where the peak running enrichment occurs
      (peak of the "mountain")
#  RES: Numerical vector containing the running enrichment score for all
      locations in the gene list
#  tag.indicator: Binary vector indicating the location of the gene sets
      (1's) in the gene list
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

  tag.indicator <- sign(match(gene.list, gene.set, nomatch=0))   # notice
      that the sign is 0 (no tag) or 1 (tag)
  no.tag.indicator <- 1 - tag.indicator
  N <- length(gene.list)
  Nh <- length(gene.set)
  Nm <-  N - Nh

  norm.tag    <- sqrt((N - Nh)/Nh)
  norm.no.tag <- sqrt(Nh/(N - Nh))

  RES <- cumsum(tag.indicator * norm.tag - no.tag.indicator * norm.no.tag)
  max.ES <- max(RES)
  min.ES <- min(RES)
  if (max.ES > - min.ES) {
     ES <- signif(max.ES, digits=5)
     arg.ES <- which.max(RES)
  } else {
     ES <- signif(min.ES, digits=5)
     arg.ES <- which.min(RES)
  }
  return(list(ES = ES, arg.ES = arg.ES, RES = RES, indicator =
      tag.indicator))
}
```

```
GSEA.EnrichmentScore2 <- function(gene.list, gene.set, weighted.score.type =
      1, correl.vector = NULL) {
#
# Computes the weighted GSEA score of gene.set in gene.list. It is the same
      calculation as in
# GSEA.EnrichmentScore but faster (x8) without producing the RES, arg.RES and
      tag.indicator outputs.
# This call is intended to be used to asses the enrichment of random
      permutations rather than the
# observed one.
# The weighted score type is the exponent of the correlation
# weight: 0 (unweighted = Kolmogorov-Smirnov), 1 (weighted), and 2 (over-
      weighted). When the score type is 1 or 2 it is
# necessary to input the correlation vector with the values in the same order
      as in the gene list.
#
# Inputs:
#  gene.list: The ordered gene list (e.g. integers indicating the original
      position in the input dataset)
#  gene.set: A gene set (e.g. integers indicating the location of those genes
      in the input dataset)
#  weighted.score.type: Type of score: weight: 0 (unweighted = Kolmogorov-
      Smirnov), 1 (weighted), and 2 (over-weighted)
#  correl.vector: A vector with the coorelations (e.g. signal to noise scores)
      corresponding to the genes in the gene list
#
# Outputs:
#  ES: Enrichment score (real number between -1 and +1)
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

   N <- length(gene.list)
   Nh <- length(gene.set)
   Nm <-  N - Nh

   loc.vector <- vector(length=N, mode="numeric")
   peak.res.vector <- vector(length=Nh, mode="numeric")
   valley.res.vector <- vector(length=Nh, mode="numeric")
   tag.correl.vector <- vector(length=Nh, mode="numeric")
   tag.diff.vector <- vector(length=Nh, mode="numeric")
   tag.loc.vector <- vector(length=Nh, mode="numeric")

   loc.vector[gene.list] <- seq(1, N)
   tag.loc.vector <- loc.vector[gene.set]

   tag.loc.vector <- sort(tag.loc.vector, decreasing = F)

   if (weighted.score.type == 0) {
      tag.correl.vector <- rep(1, Nh)
   } else if (weighted.score.type == 1) {
      tag.correl.vector <- correl.vector[tag.loc.vector]
      tag.correl.vector <- abs(tag.correl.vector)
   } else if (weighted.score.type == 2) {
```

```
      tag.correl.vector <-
       correl.vector[tag.loc.vector]*correl.vector[tag.loc.vector]
      tag.correl.vector <- abs(tag.correl.vector)
   } else {
      tag.correl.vector <- correl.vector[tag.loc.vector]**weighted.score.type
      tag.correl.vector <- abs(tag.correl.vector)
   }

   norm.tag <- 1.0/sum(tag.correl.vector)
   tag.correl.vector <- tag.correl.vector * norm.tag
   norm.no.tag <- 1.0/Nm
   tag.diff.vector[1] <- (tag.loc.vector[1] - 1)
   tag.diff.vector[2:Nh] <- tag.loc.vector[2:Nh] - tag.loc.vector[1:(Nh - 1)]
       - 1
   tag.diff.vector <- tag.diff.vector * norm.no.tag
   peak.res.vector <- cumsum(tag.correl.vector - tag.diff.vector)
   valley.res.vector <- peak.res.vector - tag.correl.vector
   max.ES <- max(peak.res.vector)
   min.ES <- min(valley.res.vector)
   ES <- signif(ifelse(max.ES > - min.ES, max.ES, min.ES), digits=5)

   return(list(ES = ES))

}

GSEA.HeatMapPlot <- function(V, row.names = F, col.labels, col.classes,
      col.names = F, main = " ", xlab=" ", ylab=" ") {
#
# Plots a heatmap "pinkogram" of a gene expression matrix including phenotype
      vector and gene, sample and phenotype labels
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

      n.rows <- length(V[,1])
      n.cols <- length(V[1,])
      row.mean <- apply(V, MARGIN=1, FUN=mean)
      row.sd <- apply(V, MARGIN=1, FUN=sd)
      row.n <- length(V[,1])
      for (i in 1:n.rows) {
         if (row.sd[i] == 0) {
            V[i,] <- 0
          } else {
            V[i,] <- (V[i,] - row.mean[i])/(0.5 * row.sd[i])
          }
          V[i,] <- ifelse(V[i,] < -6, -6, V[i,])
          V[i,] <- ifelse(V[i,] > 6, 6, V[i,])
       }

       mycol <- c("#0000FF", "#0000FF", "#4040FF", "#7070FF", "#8888FF",
      "#A9A9FF", "#D5D5FF", "#EEE5EE", "#FFAADA", "#FF9DB0", "#FF7080",
      "#FF5A5A", "#FF4040", "#FF0D1D", "#FF0000") # blue-pinkogram colors. The
      first and last are the colors to indicate the class vector (phenotype).
      This is the 1998-vintage, pre-gene cluster, original pinkogram color map
```

```
        mid.range.V <- mean(range(V)) - 0.1
        heatm <- matrix(0, nrow = n.rows + 1, ncol = n.cols)
        heatm[1:n.rows,] <- V[seq(n.rows, 1, -1),]
        heatm[n.rows + 1,] <- ifelse(col.labels == 0, 7, -7)
        image(1:n.cols, 1:(n.rows + 1), t(heatm), col=mycol, axes=FALSE,
       main=main, xlab= xlab, ylab=ylab)

        if (length(row.names) > 1) {
            numC <- nchar(row.names)
            size.row.char <- 35/(n.rows + 5)
            size.col.char <- 25/(n.cols + 5)
            maxl <- floor(n.rows/1.6)
            for (i in 1:n.rows) {
               row.names[i] <- substr(row.names[i], 1, maxl)
            }
            row.names <- c(row.names[seq(n.rows, 1, -1)], "Class")
            axis(2, at=1:(n.rows + 1), labels=row.names, adj= 0.5, tick=FALSE,
       las = 1, cex.axis=size.row.char, font.axis=2, line=-1)
        }

        if (length(col.names) > 1) {
           axis(1, at=1:n.cols, labels=col.names, tick=FALSE, las = 3,
       cex.axis=size.col.char, font.axis=2, line=-1)
        }

        C <- split(col.labels, col.labels)
        class1.size <- length(C[[1]])
        class2.size <- length(C[[2]])
        axis(3, at=c(floor(class1.size/2),class1.size + floor(class2.size/2)),
       labels=col.classes, tick=FALSE, las = 1, cex.axis=1.25, font.axis=2,
       line=-1)

       return()
}

GSEA.Res2Frame <- function(filename = "NULL") {
#
# Reads a gene expression dataset in RES format and converts it into an R data
      frame
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

   header.cont <- readLines(filename, n = 1)
   temp <- unlist(strsplit(header.cont, "\t"))
   colst <- length(temp)
   header.labels <- temp[seq(3, colst, 2)]
   ds <- read.delim(filename, header=F, row.names = 2, sep="\t", skip=3,
       blank.lines.skip=T, comment.char="", as.is=T)
   colst <- length(ds[1,])
   cols <- (colst - 1)/2
   rows <- length(ds[,1])
   A <- matrix(nrow=rows - 1, ncol=cols)
```

```
   A <- ds[1:rows, seq(2, colst, 2)]
   table1 <- data.frame(A)
   names(table1) <- header.labels
   return(table1)
}

GSEA.Gct2Frame <- function(filename = "NULL") {
#
# Reads a gene expression dataset in GCT format and converts it into an R data
      frame
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.
   ds <- read.delim(filename, header=T, sep="\t", skip=2, row.names=1,
      blank.lines.skip=T, comment.char="", as.is=T)
   ds <- ds[-1]
   return(ds)
}

GSEA.Gct2Frame2 <- function(filename = "NULL") {
#
# Reads a gene expression dataset in GCT format and converts it into an R data
      frame
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.
     content <- readLines(filename)
     content <- content[-1]
     content <- content[-1]
     col.names <- noquote(unlist(strsplit(content[1], "\t")))
     col.names <- col.names[c(-1, -2)]
     num.cols <- length(col.names)
     content <- content[-1]
     num.lines <- length(content)


     row.nam <- vector(length=num.lines, mode="character")
     row.des <- vector(length=num.lines, mode="character")
     m <- matrix(0, nrow=num.lines, ncol=num.cols)

     for (i in 1:num.lines) {
        line.list <- noquote(unlist(strsplit(content[i], "\t")))
        row.nam[i] <- noquote(line.list[1])
        row.des[i] <- noquote(line.list[2])
        line.list <- line.list[c(-1, -2)]
        for (j in 1:length(line.list)) {
           m[i, j] <- as.numeric(line.list[j])
```

```
      }
    }
    ds <- data.frame(m)
    names(ds) <- col.names
    row.names(ds) <- row.nam
    return(ds)
}

GSEA.ReadClsFile <- function(file = "NULL") {
#
# Reads a class vector CLS file and defines phenotype and class labels vectors
     for the samples in a gene expression file (RES or GCT format)
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

    cls.cont <- readLines(file)
    num.lines <- length(cls.cont)
    class.list <- unlist(strsplit(cls.cont[[3]], " "))
    s <- length(class.list)
    t <- table(class.list)
    l <- length(t)
    phen <- vector(length=l, mode="character")
    phen.label <- vector(length=l, mode="numeric")
    class.v <- vector(length=s, mode="numeric")
    for (i in 1:l) {
       phen[i] <- noquote(names(t)[i])
       phen.label[i] <- i - 1
    }
    for (i in 1:s) {
       for (j in 1:l) {
          if (class.list[i] == phen[j]) {
             class.v[i] <- phen.label[j]
          }
       }
    }
    return(list(phen = phen, class.v = class.v))
}

GSEA.Threshold <- function(V, thres, ceil) {
#
# Threshold and ceiling pre-processing for gene expression matrix
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

      V[V < thres] <- thres
```

```
       V[V > ceil] <- ceil
       return(V)
}

GSEA.VarFilter <- function(V, fold, delta, gene.names = "NULL") {
#
# Variation filter pre-processing for gene expression matrix
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

       cols <- length(V[1,])
       rows <- length(V[,1])
       row.max <- apply(V, MARGIN=1, FUN=max)
             row.min <- apply(V, MARGIN=1, FUN=min)
       flag <- array(dim=rows)
       flag <- (row.max /row.min > fold) & (row.max - row.min > delta)
       size <- sum(flag)
       B <- matrix(0, nrow = size, ncol = cols)
       j <- 1
       if (gene.names == "NULL") {
          for (i in 1:rows) {
             if (flag[i]) {
                B[j,] <- V[i,]
                j <- j + 1
              }
          }
       return(B)
       } else {
           new.list <- vector(mode = "character", length = size)
           for (i in 1:rows) {
             if (flag[i]) {
                B[j,] <- V[i,]
                new.list[j] <- gene.names[i]
                j <- j + 1
             }
           }
       return(list(V = B, new.list = new.list))
       }
}

GSEA.NormalizeRows <- function(V) {
#
# Stardardize rows of a gene expression matrix
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.
```

```
       row.mean <- apply(V, MARGIN=1, FUN=mean)
             row.sd <- apply(V, MARGIN=1, FUN=sd)
       row.n <- length(V[,1])
       for (i in 1:row.n) {
           if (row.sd[i] == 0) {
               V[i,] <- 0
         } else {
            V[i,] <- (V[i,] - row.mean[i])/row.sd[i]
         }
       }
       return(V)
}

GSEA.NormalizeCols <- function(V) {
#
# Stardardize columns of a gene expression matrix
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

       col.mean <- apply(V, MARGIN=2, FUN=mean)
             col.sd <- apply(V, MARGIN=2, FUN=sd)
       col.n <- length(V[1,])
       for (i in 1:col.n) {
           if (col.sd[i] == 0) {
               V[i,] <- 0
         } else {
            V[,i] <- (V[,i] - col.mean[i])/col.sd[i]
         }
       }
       return(V)
}

# end of auxiliary functions

# ------------------------------------------------------------------------
     ------------
# Main GSEA Analysis Function that implements the entire methodology

GSEA <- function(
input.ds,
input.cls,
gene.ann = "",
gs.db,
gs.ann = "",
output.directory = "",
doc.string = "GSEA.analysis",
non.interactive.run = F,
reshuffling.type = "sample.labels",
nperm = 1000,
weighted.score.type = 1,
nom.p.val.threshold = -1,
fwer.p.val.threshold = -1,
```

```
fdr.q.val.threshold = 0.25,
topgs = 10,
adjust.FDR.q.val = F,
gs.size.threshold.min = 25,
gs.size.threshold.max = 500,
reverse.sign = F,
preproc.type = 0,
random.seed = 123456,
perm.type = 0,
fraction = 1.0,
replace = F,
save.intermediate.results = F,
OLD.GSEA = F,
use.fast.enrichment.routine = T) {

# This is a methodology for the analysis of global molecular profiles called
      Gene Set Enrichment Analysis (GSEA). It determines
# whether an a priori defined set of genes shows statistically significant,
      concordant differences between two biological
# states (e.g. phenotypes). GSEA operates on all genes from an experiment,
      rank ordered by the signal to noise ratio and
# determines whether members of an a priori defined gene set are nonrandomly
      distributed towards the top or bottom of the
# list and thus may correspond to an important biological process. To assess
      significance the program uses an empirical
# permutation procedure to test deviation from random that preserves
      correlations between genes.
#
# For details see Subramanian et al 2005
#
# Inputs:
#   input.ds: Input gene expression Affymetrix dataset file in RES or GCT
      format
#   input.cls:  Input class vector (phenotype) file in CLS format
#   gene.ann.file: Gene microarray annotation file (Affymetrix Netaffyx *.csv
      format) (default: none)
#   gs.file: Gene set database in GMT format
#   output.directory: Directory where to store output and results (default: .)
#   doc.string:  Documentation string used as a prefix to name result files
      (default: "GSEA.analysis")
#   non.interactive.run: Run in interactive (i.e. R GUI) or batch (R command
      line) mode (default: F)
#   reshuffling.type: Type of permutation reshuffling: "sample.labels" or
      "gene.labels" (default: "sample.labels")
#   nperm: Number of random permutations (default: 1000)
#   weighted.score.type: Enrichment correlation-based weighting: 0=no weight
      (KS), 1=standard weigth, 2 = over-weigth (default: 1)
#   nom.p.val.threshold: Significance threshold for nominal p-vals for gene
      sets (default: -1, no thres)
#   fwer.p.val.threshold: Significance threshold for FWER p-vals for gene sets
      (default: -1, no thres)
#   fdr.q.val.threshold: Significance threshold for FDR q-vals for gene sets
      (default: 0.25)
#   topgs: Besides those passing test, number of top scoring gene sets used
      for detailed reports (default: 10)
#   adjust.FDR.q.val: Adjust the FDR q-vals (default: F)
#   gs.size.threshold.min: Minimum size (in genes) for database gene sets to
      be considered (default: 25)
#   gs.size.threshold.max: Maximum size (in genes) for database gene sets to
      be considered (default: 500)
```

```
#   reverse.sign: Reverse direction of gene list (pos. enrichment becomes
      negative, etc.) (default: F)
#   preproc.type: Preprocessing normalization: 0=none, 1=col(z-score).,
      2=col(rank) and row(z-score)., 3=col(rank). (default: 0)
#   random.seed: Random number generator seed. (default: 123456)
#   perm.type: Permutation type: 0 = unbalanced, 1 = balanced. For experts
      only (default: 0)
#   fraction: Subsampling fraction. Set to 1.0 (no resampling). For experts
      only (default: 1.0)
#   replace: Resampling mode (replacement or not replacement). For experts
      only (default: F)
#   OLD.GSEA: if TRUE compute the OLD GSEA of Mootha et al 2003
#   use.fast.enrichment.routine: if true it uses a faster version to compute
      random perm. enrichment "GSEA.EnrichmentScore2"
#
#   Output:
#    The results of the method are stored in the "output.directory" specified
      by the user as part of the input parameters.
#     The results files are:
#    - Two tab-separated global result text files (one for each phenotype).
      These files are labeled according to the doc
#     string prefix and the phenotype name from the CLS file:
      <doc.string>.SUMMARY.RESULTS.REPORT.<phenotype>.txt
#    - One set of global plots. They include a.- gene list correlation
      profile, b.- global observed and null densities, c.- heat map
#     for the entire sorted dataset, and d.- p-values vs. NES plot. These
      plots are in a single JPEG file named
#     <doc.string>.global.plots.<phenotype>.jpg. When the program is run
      interactively these plots appear on a window in the R GUI.
#    - A variable number of tab-separated gene result text files according to
      how many sets pass any of the significance thresholds
#     ("nom.p.val.threshold," "fwer.p.val.threshold," and
      "fdr.q.val.threshold") and how many are specified in the "topgs"
#     parameter. These files are named: <doc.string>.<gene set
      name>.report.txt.
#   - A variable number of gene set plots (one for each gene set report file).
      These plots include a.- Gene set running enrichment
#     "mountain" plot, b.- gene set null distribution and c.- heat map for
      genes in the gene set. These plots are stored in a
#     single JPEG file named <doc.string>.<gene set name>.jpg.
#  The format (columns) for the global result files is as follows.
#  GS : Gene set name.
# SIZE : Size of the set in genes.
# SOURCE : Set definition or source.
# ES : Enrichment score.
# NES : Normalized (multiplicative rescaling) normalized enrichment score.
# NOM p-val : Nominal p-value (from the null distribution of the gene set).
# FDR q-val: False discovery rate q-values
# FWER p-val: Family wise error rate p-values.
# Tag %: Percent of gene set before running enrichment peak.
# Gene %: Percent of gene list before running enrichment peak.
# Signal : enrichment signal strength.
# FDR (median): FDR q-values from the median of the null distributions.
# glob.p.val: P-value using a global statistic (number of sets above the set's
      NES).
#
# The rows are sorted by the NES values (from maximum positive or negative NES
      to minimum)
#
# The format (columns) for the gene set result files is as follows.
#
```

```
# #: Gene number in the (sorted) gene set
# GENE : gene name. For example the probe accession number, gene symbol or the
      gene identifier gin the dataset.
# SYMBOL : gene symbol from the gene annotation file.
# DESC : gene description (title) from the gene annotation file.
# LIST LOC : location of the gene in the sorted gene list.
# S2N : signal to noise ratio (correlation) of the gene in the gene list.
# RES : value of the running enrichment score at the gene location.
# CORE_ENRICHMENT: is this gene is the "core enrichment" section of the list?
      Yes or No variable specifying in the gene location is before (positive
      ES) or after (negative ES) the running enrichment peak.
#
# The rows are sorted by the gene location in the gene list.
# The function call to GSEA returns a  two element list containing the two
      global result reports as data frames ($report1, $report2).
#
# results1: Global output report for first phenotype
# result2:  Global putput report for second phenotype
#
# The Broad Institute
# SOFTWARE COPYRIGHT NOTICE AGREEMENT
# This software and its documentation are copyright 2003 by the
# Broad Institute/Massachusetts Institute of Technology.
# All rights are reserved.
#
# This software is supplied without any warranty or guaranteed support
# whatsoever. Neither the Broad Institute nor MIT can be responsible for
# its use, misuse, or functionality.

  print(" *** Running GSEA Analysis...")

if (OLD.GSEA == T) {
   print("Running OLD GSEA from Mootha et al 2003")
}

# Copy input parameters to log file

if (output.directory != "")  {

filename <- paste(output.directory, doc.string, "_params.txt", sep="",
      collapse="")

time.string <- as.character(as.POSIXlt(Sys.time(),"GMT"))
write(paste("Run of GSEA on ", time.string), file=filename)

if (is.data.frame(input.ds)) {
#     write(paste("input.ds=", quote(input.ds), sep=" "), file=filename,
      append=T)
} else {
     write(paste("input.ds=", input.ds, sep=" "), file=filename, append=T)
}
if (is.list(input.cls)) {
#     write(paste("input.cls=", input.cls, sep=" "), file=filename, append=T)
} else {
     write(paste("input.cls=", input.cls, sep=" "), file=filename, append=T)
}
if (is.data.frame(gene.ann)) {
#   write(paste("gene.ann =", gene.ann, sep=" "), file=filename, append=T)
} else {
    write(paste("gene.ann =", gene.ann, sep=" "), file=filename, append=T)
}
```

```
 if (regexpr(pattern=".gmt", gs.db[1]) == -1) {
#   write(paste("gs.db=", gs.db, sep=" "), file=filename, append=T)
} else {
   write(paste("gs.db=", gs.db, sep=" "), file=filename, append=T)
}
if (is.data.frame(gs.ann)) {
#    write(paste("gene.ann =", gene.ann, sep=" "), file=filename, append=T)
} else {
    write(paste("gs.ann =", gs.ann, sep=" "), file=filename, append=T)
}
write(paste("output.directory =", output.directory, sep=" "), file=filename,
      append=T)
write(paste("doc.string = ", doc.string, sep=" "), file=filename, append=T)
write(paste("non.interactive.run =", non.interactive.run, sep=" "),
      file=filename, append=T)
write(paste("reshuffling.type =", reshuffling.type, sep=" "), file=filename,
      append=T)
write(paste("nperm =", nperm, sep=" "), file=filename, append=T)
write(paste("weighted.score.type =", weighted.score.type, sep=" "),
      file=filename, append=T)
write(paste("nom.p.val.threshold =", nom.p.val.threshold, sep=" "),
      file=filename, append=T)
write(paste("fwer.p.val.threshold =", fwer.p.val.threshold, sep=" "),
      file=filename, append=T)
write(paste("fdr.q.val.threshold =", fdr.q.val.threshold, sep=" "),
      file=filename, append=T)
write(paste("topgs =", topgs, sep=" "), file=filename, append=T)
write(paste("adjust.FDR.q.val =", adjust.FDR.q.val, sep=" "), file=filename,
      append=T)
write(paste("gs.size.threshold.min =", gs.size.threshold.min, sep=" "),
      file=filename, append=T)
write(paste("gs.size.threshold.max =", gs.size.threshold.max, sep=" "),
      file=filename, append=T)
write(paste("reverse.sign =", reverse.sign, sep=" "), file=filename, append=T)
write(paste("preproc.type =", preproc.type, sep=" "), file=filename, append=T)
write(paste("random.seed =", random.seed, sep=" "), file=filename, append=T)
write(paste("perm.type =", perm.type, sep=" "), file=filename, append=T)
write(paste("fraction =", fraction, sep=" "), file=filename, append=T)
write(paste("replace =", replace, sep=" "), file=filename, append=T)
}

# Start of GSEA methodology

  if (.Platform$OS.type == "windows") {
#    memory.limit(6000000000)
     memory.limit()
#     print(c("Start memory size=",  memory.size()))
  }

  # Read input data matrix

  set.seed(seed=random.seed, kind = NULL)
  adjust.param <- 0.5

  gc()

  time1 <- proc.time()

  if (is.data.frame(input.ds)) {
     dataset <- input.ds
  } else {
```

```
    if (regexpr(pattern=".gct", input.ds) == -1) {
       dataset <- GSEA.Res2Frame(filename = input.ds)
    } else {
#       dataset <- GSEA.Gct2Frame(filename = input.ds)
       dataset <- GSEA.Gct2Frame2(filename = input.ds)
    }
  }
  gene.labels <- row.names(dataset)
  sample.names <- names(dataset)
  A <- data.matrix(dataset)
  dim(A)
  cols <- length(A[1,])
  rows <- length(A[,1])

# preproc.type control the type of pre-processing: threshold, variation
      filter, normalization

 if (preproc.type == 1) {  # Column normalize (Z-score)
   A <- GSEA.NormalizeCols(A)
 } else if (preproc.type == 2) { # Column (rank) and row (Z-score) normalize
   for (j in 1:cols) {  # column rank normalization
      A[,j] <- rank(A[,j])
   }
   A <- GSEA.NormalizeRows(A)
 } else if (preproc.type == 3) { # Column (rank) norm.
   for (j in 1:cols) {  # column rank normalization
      A[,j] <- rank(A[,j])
   }
 }

  # Read input class vector

  if(is.list(input.cls)) {
     CLS <- input.cls
  } else {
     CLS <- GSEA.ReadClsFile(file=input.cls)
  }
  class.labels <- CLS$class.v
  class.phen <- CLS$phen

  if (reverse.sign == T) {
     phen1 <- class.phen[2]
     phen2 <- class.phen[1]
  } else {
     phen1 <- class.phen[1]
     phen2 <- class.phen[2]
  }

  # sort samples according to phenotype

 col.index <- order(class.labels, decreasing=F)
 class.labels <- class.labels[col.index]
 sample.names <- sample.names[col.index]
 for (j in 1:rows) {
    A[j, ] <- A[j, col.index]
 }
 names(A) <- sample.names

  # Read input gene set database

 if (regexpr(pattern=".gmt", gs.db[1]) == -1) {
```

```
    temp <- gs.db
} else {
    temp <- readLines(gs.db)
}

    max.Ng <- length(temp)
    temp.size.G <- vector(length = max.Ng, mode = "numeric")
    for (i in 1:max.Ng) {
       temp.size.G[i] <- length(unlist(strsplit(temp[[i]], "\t"))) - 2
    }

    max.size.G <- max(temp.size.G)
    gs <- matrix(rep("null", max.Ng*max.size.G), nrow=max.Ng, ncol=
     max.size.G)
    temp.names <- vector(length = max.Ng, mode = "character")
    temp.desc <- vector(length = max.Ng, mode = "character")
    gs.count <- 1
    for (i in 1:max.Ng) {
       gene.set.size <- length(unlist(strsplit(temp[[i]], "\t"))) - 2
       gs.line <- noquote(unlist(strsplit(temp[[i]], "\t")))
       gene.set.name <- gs.line[1]
       gene.set.desc <- gs.line[2]
       gene.set.tags <- vector(length = gene.set.size, mode = "character")
       for (j in 1:gene.set.size) {
          gene.set.tags[j] <- gs.line[j + 2]
       }
       existing.set <- is.element(gene.set.tags, gene.labels)
       set.size <- length(existing.set[existing.set == T])
       if ((set.size < gs.size.threshold.min) || (set.size >
     gs.size.threshold.max)) next
       temp.size.G[gs.count] <- set.size
       gs[gs.count,] <- c(gene.set.tags[existing.set], rep("null",
     max.size.G - temp.size.G[gs.count]))
       temp.names[gs.count] <- gene.set.name
       temp.desc[gs.count] <- gene.set.desc
       gs.count <- gs.count + 1
    }
    Ng <- gs.count - 1
    gs.names <- vector(length = Ng, mode = "character")
    gs.desc <- vector(length = Ng, mode = "character")
    size.G <- vector(length = Ng, mode = "numeric")
    gs.names <- temp.names[1:Ng]
    gs.desc <- temp.desc[1:Ng]
    size.G <- temp.size.G[1:Ng]

  N <- length(A[,1])
  Ns <- length(A[1,])

  print(c("Number of genes:", N))
  print(c("Number of Gene Sets:", Ng))
  print(c("Number of samples:", Ns))
  print(c("Original number of Gene Sets:", max.Ng))
  print(c("Maximum gene set size:", max.size.G))

# Read gene and gene set annotations if gene annotation file was provided

  all.gene.descs <- vector(length = N, mode ="character")
  all.gene.symbols <- vector(length = N, mode ="character")
  all.gs.descs <- vector(length = Ng, mode ="character")

  if (is.data.frame(gene.ann)) {
```

```
   temp <- gene.ann
   a.size <- length(temp[,1])
   print(c("Number of gene annotation file entries:", a.size))
   accs <- as.character(temp[,1])
   locs <- match(gene.labels, accs)
   all.gene.descs <- as.character(temp[locs, "Gene.Title"])
   all.gene.symbols <- as.character(temp[locs, "Gene.Symbol"])
   rm(temp)
} else  if (gene.ann == "") {
   for (i in 1:N) {
      all.gene.descs[i] <- gene.labels[i]
      all.gene.symbols[i] <- gene.labels[i]
   }
} else {
   temp <- read.delim(gene.ann, header=T, sep=",", comment.char="", as.is=T)
   a.size <- length(temp[,1])
   print(c("Number of gene annotation file entries:", a.size))
   accs <- as.character(temp[,1])
   locs <- match(gene.labels, accs)
   all.gene.descs <- as.character(temp[locs, "Gene.Title"])
   all.gene.symbols <- as.character(temp[locs, "Gene.Symbol"])
   rm(temp)
}

if (is.data.frame(gs.ann)) {
   temp <- gs.ann
   a.size <- length(temp[,1])
   print(c("Number of gene set annotation file entries:", a.size))
   accs <- as.character(temp[,1])
   locs <- match(gs.names, accs)
   all.gs.descs <- as.character(temp[locs, "SOURCE"])
   rm(temp)
} else if (gs.ann == "") {
   for (i in 1:Ng) {
      all.gs.descs[i] <- gs.desc[i]
   }
} else {
   temp <- read.delim(gs.ann, header=T, sep="\t", comment.char="", as.is=T)
   a.size <- length(temp[,1])
   print(c("Number of gene set annotation file entries:", a.size))
   accs <- as.character(temp[,1])
   locs <- match(gs.names, accs)
   all.gs.descs <- as.character(temp[locs, "SOURCE"])
   rm(temp)
}


Obs.indicator <- matrix(nrow= Ng, ncol=N)
Obs.RES <- matrix(nrow= Ng, ncol=N)

Obs.ES <- vector(length = Ng, mode = "numeric")
Obs.arg.ES <- vector(length = Ng, mode = "numeric")
Obs.ES.norm <- vector(length = Ng, mode = "numeric")

time2 <- proc.time()

# GSEA methodology

# Compute observed and random permutation gene rankings

obs.s2n <- vector(length=N, mode="numeric")
```

```
 signal.strength <- vector(length=Ng, mode="numeric")
 tag.frac <- vector(length=Ng, mode="numeric")
 gene.frac <- vector(length=Ng, mode="numeric")
 coherence.ratio <- vector(length=Ng, mode="numeric")
 obs.phi.norm <- matrix(nrow = Ng, ncol = nperm)
 correl.matrix <- matrix(nrow = N, ncol = nperm)
 obs.correl.matrix <- matrix(nrow = N, ncol = nperm)
 order.matrix <- matrix(nrow = N, ncol = nperm)
 obs.order.matrix <- matrix(nrow = N, ncol = nperm)

  nperm.per.call <- 100
  n.groups <- nperm %/% nperm.per.call
  n.rem <- nperm %% nperm.per.call
  n.perms <- c(rep(nperm.per.call, n.groups), n.rem)
  n.ends <- cumsum(n.perms)
  n.starts <- n.ends - n.perms + 1

  if (n.rem == 0) {
    n.tot <- n.groups
  } else {
    n.tot <- n.groups + 1
  }

for (nk in 1:n.tot) {
  call.nperm <- n.perms[nk]

  print(paste("Computing ranked list for actual and permuted
      phenotypes.......permutations: ", n.starts[nk], "--", n.ends[nk], sep="
      "))

  O <- GSEA.GeneRanking(A, class.labels, gene.labels, call.nperm,
      permutation.type = perm.type, sigma.correction = "GeneCluster",
      fraction=fraction, replace=replace, reverse.sign = reverse.sign)
  gc()

  order.matrix[,n.starts[nk]:n.ends[nk]] <- O$order.matrix
  obs.order.matrix[,n.starts[nk]:n.ends[nk]] <- O$obs.order.matrix
  correl.matrix[,n.starts[nk]:n.ends[nk]] <- O$s2n.matrix
  obs.correl.matrix[,n.starts[nk]:n.ends[nk]] <- O$obs.s2n.matrix
   rm(O)
}

 obs.s2n <- apply(obs.correl.matrix, 1, median)  # using median to assign
     enrichment scores
 obs.index <- order(obs.s2n, decreasing=T)
 obs.s2n  <- sort(obs.s2n, decreasing=T)

 obs.gene.labels <- gene.labels[obs.index]
 obs.gene.descs <- all.gene.descs[obs.index]
 obs.gene.symbols <- all.gene.symbols[obs.index]

 for (r in 1:nperm) {
    correl.matrix[, r] <- correl.matrix[order.matrix[,r], r]
 }
 for (r in 1:nperm) {
    obs.correl.matrix[, r] <- obs.correl.matrix[obs.order.matrix[,r], r]
 }

 gene.list2 <- obs.index
 for (i in 1:Ng) {
```

```
      print(paste("Computing observed enrichment for gene set:", i,
      gs.names[i], sep=" "))
      gene.set <- gs[i,gs[i,] != "null"]
      gene.set2 <- vector(length=length(gene.set), mode = "numeric")
      gene.set2 <- match(gene.set, gene.labels)
      if (OLD.GSEA == F) {
        GSEA.results <- GSEA.EnrichmentScore(gene.list=gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector = obs.s2n)
      } else {
        GSEA.results <- OLD.GSEA.EnrichmentScore(gene.list=gene.list2,
      gene.set=gene.set2)
      }
      Obs.ES[i] <- GSEA.results$ES
      Obs.arg.ES[i] <- GSEA.results$arg.ES
      Obs.RES[i,] <- GSEA.results$RES
      Obs.indicator[i,] <- GSEA.results$indicator
      if (Obs.ES[i] >= 0) {  # compute signal strength
          tag.frac[i] <- sum(Obs.indicator[i,1:Obs.arg.ES[i]])/size.G[i]
          gene.frac[i] <- Obs.arg.ES[i]/N
      } else {
          tag.frac[i] <- sum(Obs.indicator[i, Obs.arg.ES[i]:N])/size.G[i]
          gene.frac[i] <- (N - Obs.arg.ES[i] + 1)/N
      }
      signal.strength[i] <- tag.frac[i] * (1 - gene.frac[i]) * (N / (N -
      size.G[i]))
   }

# Compute enrichment for random permutations

   phi <- matrix(nrow = Ng, ncol = nperm)
   phi.norm <- matrix(nrow = Ng, ncol = nperm)
   obs.phi <- matrix(nrow = Ng, ncol = nperm)

   if (reshuffling.type == "sample.labels") { # reshuffling phenotype labels
      for (i in 1:Ng) {
       print(paste("Computing random permutations' enrichment for gene set:",
      i, gs.names[i], sep=" "))
       gene.set <- gs[i,gs[i,] != "null"]
       gene.set2 <- vector(length=length(gene.set), mode = "numeric")
       gene.set2 <- match(gene.set, gene.labels)
       for (r in 1:nperm) {
           gene.list2 <- order.matrix[,r]
           if (use.fast.enrichment.routine == F) {
             GSEA.results <- GSEA.EnrichmentScore(gene.list=gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=correl.matrix[, r])
           } else {
             GSEA.results <- GSEA.EnrichmentScore2(gene.list=gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=correl.matrix[, r])
           }
           phi[i, r] <- GSEA.results$ES
       }
       if (fraction < 1.0) { # if resampling then compute ES for all observed
      rankings
           for (r in 1:nperm) {
              obs.gene.list2 <- obs.order.matrix[,r]
              if (use.fast.enrichment.routine == F) {
```

```
            GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
           } else {
            GSEA.results <-
    GSEA.EnrichmentScore2(gene.list=obs.gene.list2, gene.set=gene.set2,
    weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
           }
           obs.phi[i, r] <- GSEA.results$ES
        }
     } else { # if no resampling then compute only one column (and fill the
    others with the same value)
         obs.gene.list2 <- obs.order.matrix[,1]
        if (use.fast.enrichment.routine == F) {
          GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
        } else {
          GSEA.results <- GSEA.EnrichmentScore2(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
        }
        obs.phi[i, 1] <- GSEA.results$ES
        for (r in 2:nperm) {
          obs.phi[i, r] <- obs.phi[i, 1]
        }
     }
     gc()
  }

} else if (reshuffling.type == "gene.labels") { # reshuffling gene labels
   for (i in 1:Ng) {
     gene.set <- gs[i,gs[i,] != "null"]
     gene.set2 <- vector(length=length(gene.set), mode = "numeric")
     gene.set2 <- match(gene.set, gene.labels)
     for (r in 1:nperm) {
        reshuffled.gene.labels <- sample(1:rows)
        if (use.fast.enrichment.routine == F) {
          GSEA.results <-
    GSEA.EnrichmentScore(gene.list=reshuffled.gene.labels,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.s2n)
        } else {
          GSEA.results <-
    GSEA.EnrichmentScore2(gene.list=reshuffled.gene.labels,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.s2n)
        }
        phi[i, r] <- GSEA.results$ES
     }
     if (fraction < 1.0) { # if resampling then compute ES for all observed
    rankings
       for (r in 1:nperm) {
         obs.gene.list2 <- obs.order.matrix[,r]
         if (use.fast.enrichment.routine == F) {
           GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
         } else {
```

```
            GSEA.results <- GSEA.EnrichmentScore2(gene.list=obs.gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=obs.correl.matrix[, r])
          }
          obs.phi[i, r] <- GSEA.results$ES
        }
       } else { # if no resampling then compute only one column (and fill the
      others with the same value)
        obs.gene.list2 <- obs.order.matrix[,1]
        if (use.fast.enrichment.routine == F) {
          GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=obs.correl.matrix[, r])
        } else {
          GSEA.results <- GSEA.EnrichmentScore2(gene.list=obs.gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=obs.correl.matrix[, r])
        }
        obs.phi[i, 1] <- GSEA.results$ES
        for (r in 2:nperm) {
          obs.phi[i, r] <- obs.phi[i, 1]
        }
       }
       gc()
     }
   }

# Compute 3 types of p-values

# Find nominal p-values

print("Computing nominal p-values...")

p.vals <- matrix(0, nrow = Ng, ncol = 2)

if (OLD.GSEA == F) {
   for (i in 1:Ng) {
     pos.phi <- NULL
     neg.phi <- NULL
     for (j in 1:nperm) {
       if (phi[i, j] >= 0) {
         pos.phi <- c(pos.phi, phi[i, j])
       } else {
         neg.phi <- c(neg.phi, phi[i, j])
       }
     }
     ES.value <- Obs.ES[i]
     if (ES.value >= 0) {
       p.vals[i, 1] <- signif(sum(pos.phi >= ES.value)/length(pos.phi),
      digits=5)
     } else {
       p.vals[i, 1] <- signif(sum(neg.phi <= ES.value)/length(neg.phi),
      digits=5)
     }
   }
} else {  # For OLD GSEA compute the p-val using positive and negative values
       in the same histogram
   for (i in 1:Ng) {
     if (Obs.ES[i] >= 0) {
       p.vals[i, 1] <-  sum(phi[i,] >= Obs.ES[i])/length(phi[i,])
       p.vals[i, 1] <-  signif(p.vals[i, 1], digits=5)
```

```
     } else {
        p.vals[i, 1] <-  sum(phi[i,] <= Obs.ES[i])/length(phi[i,])
        p.vals[i, 1] <-  signif(p.vals[i, 1], digits=5)
     }
   }
}

# Find effective size

 erf <- function (x)
 {
    2 * pnorm(sqrt(2) * x)
 }

 KS.mean <- function(N) { # KS mean as a function of set size N
      S <- 0
      for (k in -100:100) {
         if (k == 0) next
         S <- S + 4 * (-1)**(k + 1) * (0.25 * exp(-2 * k * k * N) - sqrt(2 *
      pi) *  erf(sqrt(2 * N) * k)/(16 * k * sqrt(N)))
      }
     return(abs(S))
 }

# KS.mean.table <- vector(length=5000, mode="numeric")

# for (i in 1:5000) {
#    KS.mean.table[i] <- KS.mean(i)
# }

# KS.size <-  vector(length=Ng, mode="numeric")

# Rescaling normalization for each gene set null

print("Computing rescaling normalization for each gene set null...")

if (OLD.GSEA == F) {
   for (i in 1:Ng) {
        pos.phi <- NULL
        neg.phi <- NULL
        for (j in 1:nperm) {
           if (phi[i, j] >= 0) {
              pos.phi <- c(pos.phi, phi[i, j])
           } else {
              neg.phi <- c(neg.phi, phi[i, j])
           }
        }
        pos.m <- mean(pos.phi)
        neg.m <- mean(abs(neg.phi))

#        if (Obs.ES[i] >= 0) {
#           KS.size[i] <- which.min(abs(KS.mean.table - pos.m))
#        } else {
#           KS.size[i] <- which.min(abs(KS.mean.table - neg.m))
#        }

        pos.phi <- pos.phi/pos.m
        neg.phi <- neg.phi/neg.m
        for (j in 1:nperm) {
           if (phi[i, j] >= 0) {
               phi.norm[i, j] <- phi[i, j]/pos.m
```

```
          } else {
             phi.norm[i, j] <- phi[i, j]/neg.m
          }
        }
        for (j in 1:nperm) {
          if (obs.phi[i, j] >= 0) {
             obs.phi.norm[i, j] <- obs.phi[i, j]/pos.m
          } else {
             obs.phi.norm[i, j] <- obs.phi[i, j]/neg.m
          }
        }
        if (Obs.ES[i] >= 0) {
          Obs.ES.norm[i] <- Obs.ES[i]/pos.m
        } else {
          Obs.ES.norm[i] <- Obs.ES[i]/neg.m
        }
   }
} else {  # For OLD GSEA does not normalize using empirical scaling
   for (i in 1:Ng) {
      for (j in 1:nperm) {
         phi.norm[i, j] <- phi[i, j]/400
      }
      for (j in 1:nperm) {
         obs.phi.norm[i, j] <- obs.phi[i, j]/400
      }
      Obs.ES.norm[i] <- Obs.ES[i]/400
   }
}

# Save intermedite results

      if (save.intermediate.results == T) {

        filename <- paste(output.directory, doc.string, ".phi.txt", sep="",
      collapse="")
        write.table(phi, file = filename, quote=F, col.names= F, row.names=F,
      sep = "\t")

        filename <- paste(output.directory, doc.string, ".obs.phi.txt",
      sep="", collapse="")
        write.table(obs.phi, file = filename, quote=F, col.names= F,
      row.names=F, sep = "\t")

        filename <- paste(output.directory, doc.string, ".phi.norm.txt",
      sep="", collapse="")
        write.table(phi.norm, file = filename, quote=F, col.names= F,
      row.names=F, sep = "\t")

        filename <- paste(output.directory, doc.string, ".obs.phi.norm.txt",
      sep="", collapse="")
        write.table(obs.phi.norm, file = filename, quote=F, col.names= F,
      row.names=F, sep = "\t")

        filename <- paste(output.directory, doc.string, ".Obs.ES.txt",
      sep="", collapse="")
        write.table(Obs.ES, file = filename, quote=F, col.names= F,
      row.names=F, sep = "\t")

        filename <- paste(output.directory, doc.string, ".Obs.ES.norm.txt",
      sep="", collapse="")
```

```
         write.table(Obs.ES.norm, file = filename, quote=F, col.names= F,
       row.names=F, sep = "\t")
      }

# Compute FWER p-vals

     print("Computing FWER p-values...")

if (OLD.GSEA == F) {
    max.ES.vals.p <- NULL
    max.ES.vals.n <- NULL
    for (j in 1:nperm) {
       pos.phi <- NULL
       neg.phi <- NULL
       for (i in 1:Ng) {
          if (phi.norm[i, j] >= 0) {
             pos.phi <- c(pos.phi, phi.norm[i, j])
          } else {
             neg.phi <- c(neg.phi, phi.norm[i, j])
          }
       }
       if (length(pos.phi) > 0) {
          max.ES.vals.p <- c(max.ES.vals.p, max(pos.phi))
       }
       if (length(neg.phi) > 0) {
          max.ES.vals.n <- c(max.ES.vals.n, min(neg.phi))
       }
     }
   for (i in 1:Ng) {
       ES.value <- Obs.ES.norm[i]
       if (Obs.ES.norm[i] >= 0) {
          p.vals[i, 2] <- signif(sum(max.ES.vals.p >=
       ES.value)/length(max.ES.vals.p), digits=5)
       } else {
          p.vals[i, 2] <- signif(sum(max.ES.vals.n <=
       ES.value)/length(max.ES.vals.n), digits=5)
       }
     }
} else {  # For OLD GSEA compute the FWER using positive and negative values
       in the same histogram
    max.ES.vals <- NULL
    for (j in 1:nperm) {
       max.NES <- max(phi.norm[,j])
       min.NES <- min(phi.norm[,j])
       if (max.NES > - min.NES) {
          max.val <- max.NES
       } else {
          max.val <- min.NES
       }
       max.ES.vals <- c(max.ES.vals, max.val)
    }
    for (i in 1:Ng) {
       if (Obs.ES.norm[i] >= 0) {
          p.vals[i, 2] <- sum(max.ES.vals >=
       Obs.ES.norm[i])/length(max.ES.vals)
       } else {
          p.vals[i, 2] <- sum(max.ES.vals <=
       Obs.ES.norm[i])/length(max.ES.vals)
       }
       p.vals[i, 2] <-  signif(p.vals[i, 2], digits=4)
     }
```

```
}

# Compute FDRs

     print("Computing FDR q-values...")

     NES <- vector(length=Ng, mode="numeric")
     phi.norm.mean  <- vector(length=Ng, mode="numeric")
     obs.phi.norm.mean  <- vector(length=Ng, mode="numeric")
     phi.norm.median  <- vector(length=Ng, mode="numeric")
     obs.phi.norm.median  <- vector(length=Ng, mode="numeric")
     phi.norm.mean  <- vector(length=Ng, mode="numeric")
     obs.phi.mean  <- vector(length=Ng, mode="numeric")
     FDR.mean <- vector(length=Ng, mode="numeric")
     FDR.median <- vector(length=Ng, mode="numeric")
     phi.norm.median.d <- vector(length=Ng, mode="numeric")
     obs.phi.norm.median.d <- vector(length=Ng, mode="numeric")

     Obs.ES.index <- order(Obs.ES.norm, decreasing=T)
     Orig.index <- seq(1, Ng)
     Orig.index <- Orig.index[Obs.ES.index]
     Orig.index <- order(Orig.index, decreasing=F)
     Obs.ES.norm.sorted <- Obs.ES.norm[Obs.ES.index]
     gs.names.sorted <- gs.names[Obs.ES.index]

     for (k in 1:Ng) {
        NES[k] <- Obs.ES.norm.sorted[k]
        ES.value <- NES[k]
        count.col <- vector(length=nperm, mode="numeric")
        obs.count.col <- vector(length=nperm, mode="numeric")
        for (i in 1:nperm) {
           phi.vec <- phi.norm[,i]
           obs.phi.vec <- obs.phi.norm[,i]
           if (ES.value >= 0) {
              count.col.norm <- sum(phi.vec >= 0)
              obs.count.col.norm <- sum(obs.phi.vec >= 0)
              count.col[i] <- ifelse(count.col.norm > 0, sum(phi.vec >=
      ES.value)/count.col.norm, 0)
              obs.count.col[i] <- ifelse(obs.count.col.norm > 0,
      sum(obs.phi.vec >= ES.value)/obs.count.col.norm, 0)
           } else {
              count.col.norm <- sum(phi.vec < 0)
              obs.count.col.norm <- sum(obs.phi.vec < 0)
              count.col[i] <- ifelse(count.col.norm > 0, sum(phi.vec <=
      ES.value)/count.col.norm, 0)
              obs.count.col[i] <- ifelse(obs.count.col.norm > 0,
      sum(obs.phi.vec <= ES.value)/obs.count.col.norm, 0)
           }
         }
        phi.norm.mean[k] <- mean(count.col)
        obs.phi.norm.mean[k] <- mean(obs.count.col)
        phi.norm.median[k] <- median(count.col)
        obs.phi.norm.median[k] <- median(obs.count.col)
        FDR.mean[k] <- ifelse(phi.norm.mean[k]/obs.phi.norm.mean[k] < 1,
      phi.norm.mean[k]/obs.phi.norm.mean[k], 1)
        FDR.median[k] <- ifelse(phi.norm.median[k]/obs.phi.norm.median[k] < 1,
      phi.norm.median[k]/obs.phi.norm.median[k], 1)
     }

# adjust q-values
```

```
  if (adjust.FDR.q.val == T) {
     pos.nes <- length(NES[NES >= 0])
     min.FDR.mean <- FDR.mean[pos.nes]
     min.FDR.median <- FDR.median[pos.nes]
     for (k in seq(pos.nes - 1, 1, -1)) {
         if (FDR.mean[k] < min.FDR.mean) {
            min.FDR.mean <- FDR.mean[k]
         }
         if (min.FDR.mean < FDR.mean[k]) {
            FDR.mean[k] <- min.FDR.mean
         }
     }

     neg.nes <- pos.nes + 1
     min.FDR.mean <- FDR.mean[neg.nes]
     min.FDR.median <- FDR.median[neg.nes]
     for (k in seq(neg.nes + 1, Ng)) {
         if (FDR.mean[k] < min.FDR.mean) {
            min.FDR.mean <- FDR.mean[k]
         }
         if (min.FDR.mean < FDR.mean[k]) {
            FDR.mean[k] <- min.FDR.mean
         }
     }
  }

  obs.phi.norm.mean.sorted <- obs.phi.norm.mean[Orig.index]
  phi.norm.mean.sorted <- phi.norm.mean[Orig.index]
  FDR.mean.sorted <- FDR.mean[Orig.index]
  FDR.median.sorted <- FDR.median[Orig.index]

#   Compute global statistic

  glob.p.vals <- vector(length=Ng, mode="numeric")
  NULL.pass <- vector(length=nperm, mode="numeric")
  OBS.pass <- vector(length=nperm, mode="numeric")

  for (k in 1:Ng) {
    NES[k] <- Obs.ES.norm.sorted[k]
    if (NES[k] >= 0) {
       for (i in 1:nperm) {
           NULL.pos <- sum(phi.norm[,i] >= 0)
           NULL.pass[i] <- ifelse(NULL.pos > 0, sum(phi.norm[,i] >=
     NES[k])/NULL.pos, 0)
           OBS.pos <- sum(obs.phi.norm[,i] >= 0)
           OBS.pass[i] <- ifelse(OBS.pos > 0, sum(obs.phi.norm[,i] >=
     NES[k])/OBS.pos, 0)
       }
    } else {
       for (i in 1:nperm) {
           NULL.neg <- sum(phi.norm[,i] < 0)
           NULL.pass[i] <- ifelse(NULL.neg > 0, sum(phi.norm[,i] <=
     NES[k])/NULL.neg, 0)
           OBS.neg <- sum(obs.phi.norm[,i] < 0)
           OBS.pass[i] <- ifelse(OBS.neg > 0, sum(obs.phi.norm[,i] <=
     NES[k])/OBS.neg, 0)
       }
    }
    glob.p.vals[k] <- sum(NULL.pass >= mean(OBS.pass))/nperm
  }
  glob.p.vals.sorted <- glob.p.vals[Orig.index]
```

```
# Produce results report

     print("Producing result tables and plots...")

      Obs.ES <- signif(Obs.ES, digits=5)
      Obs.ES.norm <- signif(Obs.ES.norm, digits=5)
      p.vals <- signif(p.vals, digits=4)
      signal.strength <- signif(signal.strength, digits=3)
      tag.frac <- signif(tag.frac, digits=3)
      gene.frac <- signif(gene.frac, digits=3)
      FDR.mean.sorted <- signif(FDR.mean.sorted, digits=5)
      FDR.median.sorted <-  signif(FDR.median.sorted, digits=5)
      glob.p.vals.sorted <- signif(glob.p.vals.sorted, digits=5)

      report <- data.frame(cbind(gs.names, size.G, all.gs.descs, Obs.ES,
      Obs.ES.norm, p.vals[,1], FDR.mean.sorted, p.vals[,2], tag.frac,
      gene.frac, signal.strength, FDR.median.sorted, glob.p.vals.sorted))
      names(report) <- c("GS", "SIZE", "SOURCE", "ES", "NES", "NOM p-val",
      "FDR q-val", "FWER p-val", "Tag \%", "Gene \%", "Signal", "FDR
      (median)", "glob.p.val")
#      print(report)
      report2 <- report
      report.index2 <- order(Obs.ES.norm, decreasing=T)
      for (i in 1:Ng) {
         report2[i,] <- report[report.index2[i],]
      }
      report3 <- report
      report.index3 <- order(Obs.ES.norm, decreasing=F)
      for (i in 1:Ng) {
         report3[i,] <- report[report.index3[i],]
      }
      phen1.rows <- length(Obs.ES.norm[Obs.ES.norm >= 0])
      phen2.rows <- length(Obs.ES.norm[Obs.ES.norm < 0])
      report.phen1 <- report2[1:phen1.rows,]
      report.phen2 <- report3[1:phen2.rows,]

if (output.directory != "")  {
      if (phen1.rows > 0) {
         filename <- paste(output.directory, doc.string,
      ".SUMMARY.RESULTS.REPORT.", phen1,".txt", sep="", collapse="")
         write.table(report.phen1, file = filename, quote=F, row.names=F, sep
      = "\t")
      }
      if (phen2.rows > 0) {
         filename <- paste(output.directory, doc.string,
      ".SUMMARY.RESULTS.REPORT.", phen2,".txt", sep="", collapse="")
         write.table(report.phen2, file = filename, quote=F, row.names=F, sep
      = "\t")
      }
}

# Global plots

if (output.directory != "")  {
     if (non.interactive.run == F) {
         if (.Platform$OS.type == "windows") {
            glob.filename <- paste(output.directory, doc.string,
       ".global.plots", sep="", collapse="")
            windows(width = 10, height = 10)
         } else if (.Platform$OS.type == "unix") {
```

```
            glob.filename <- paste(output.directory, doc.string,
      ".global.plots.pdf", sep="", collapse="")
            pdf(file=glob.filename, height = 10, width = 10)
          }
     } else {
          if (.Platform$OS.type == "unix") {
             glob.filename <- paste(output.directory, doc.string,
      ".global.plots.pdf", sep="", collapse="")
            pdf(file=glob.filename, height = 10, width = 10)
          } else if (.Platform$OS.type == "windows") {
             glob.filename <- paste(output.directory, doc.string,
      ".global.plots.pdf", sep="", collapse="")
            pdf(file=glob.filename, height = 10, width = 10)
          }
     }
}

     nf <- layout(matrix(c(1,2,3,4), 2, 2, byrow=T), c(1,1), c(1,1), TRUE)

# plot S2N correlation profile

    location <- 1:N
    max.corr <- max(obs.s2n)
    min.corr <- min(obs.s2n)

    x <- plot(location, obs.s2n, ylab = "Signal to Noise Ratio (S2N)", xlab =
      "Gene List Location", main = "Gene List Correlation (S2N) Profile", type
      = "l", lwd = 2, cex = 0.9, col = 1)
    for (i in seq(1, N, 20)) {
      lines(c(i, i), c(0, obs.s2n[i]), lwd = 3, cex = 0.9, col = colors()[12])
      # shading of correlation plot
    }
    x <- points(location, obs.s2n, type = "l", lwd = 2, cex = 0.9, col = 1)
    lines(c(1, N), c(0, 0), lwd = 2, lty = 1, cex = 0.9, col = 1) # zero
      correlation horizontal line
    temp <- order(abs(obs.s2n), decreasing=T)
    arg.correl <- temp[N]
    lines(c(arg.correl, arg.correl), c(min.corr, 0.7*max.corr), lwd = 2, lty
      = 3, cex = 0.9, col = 1) # zero correlation vertical line

    area.bias <- signif(100*(sum(obs.s2n[1:arg.correl]) +
      sum(obs.s2n[arg.correl:N]))/sum(abs(obs.s2n[1:N])), digits=3)
    area.phen <- ifelse(area.bias >= 0, phen1, phen2)
    delta.string <- paste("Corr. Area Bias to \"", area.phen, "\" =",
      abs(area.bias), "\%", sep="", collapse="")
    zero.crossing.string <- paste("Zero Crossing at location ", arg.correl, "
      (",  signif(100*arg.correl/N, digits=3), " \%)")
    leg.txt <- c(delta.string, zero.crossing.string)
    legend(x=N/10, y=max.corr, bty="n", bg = "white", legend=leg.txt, cex =
      0.9)

    leg.txt <- paste("\"", phen1, "\" ", sep="", collapse="")
    text(x=1, y=-0.05*max.corr, adj = c(0, 1), labels=leg.txt, cex = 0.9)

    leg.txt <- paste("\"", phen2, "\" ", sep="", collapse="")
    text(x=N, y=0.05*max.corr, adj = c(1, 0), labels=leg.txt, cex = 0.9)

  if (Ng > 1) { # make these plots only if there are multiple gene sets.

    # compute plots of actual (weighted) null and observed
```

```
phi.densities.pos <- matrix(0, nrow=512, ncol=nperm)
phi.densities.neg <- matrix(0, nrow=512, ncol=nperm)
obs.phi.densities.pos <- matrix(0, nrow=512, ncol=nperm)
obs.phi.densities.neg <- matrix(0, nrow=512, ncol=nperm)
phi.density.mean.pos <- vector(length=512, mode = "numeric")
phi.density.mean.neg <- vector(length=512, mode = "numeric")
obs.phi.density.mean.pos <- vector(length=512, mode = "numeric")
obs.phi.density.mean.neg <- vector(length=512, mode = "numeric")
phi.density.median.pos <- vector(length=512, mode = "numeric")
phi.density.median.neg <- vector(length=512, mode = "numeric")
obs.phi.density.median.pos <- vector(length=512, mode = "numeric")
obs.phi.density.median.neg <- vector(length=512, mode = "numeric")
x.coor.pos <-  vector(length=512, mode = "numeric")
x.coor.neg <-  vector(length=512, mode = "numeric")

for (i in 1:nperm) {
   pos.phi <- phi.norm[phi.norm[, i] >= 0, i]
   if (length(pos.phi) > 2) {
      temp <- density(pos.phi, adjust=adjust.param, n = 512, from=0,
 to=3.5)
   } else {
      temp <- list(x = 3.5*(seq(1, 512) - 1)/512, y = rep(0.001, 512))
   }
   phi.densities.pos[, i] <- temp$y
   norm.factor <- sum(phi.densities.pos[, i])
   phi.densities.pos[, i] <- phi.densities.pos[, i]/norm.factor
   if (i == 1) {
      x.coor.pos <- temp$x
   }

   neg.phi <- phi.norm[phi.norm[, i] < 0, i]
   if (length(neg.phi) > 2) {
      temp <- density(neg.phi, adjust=adjust.param, n = 512, from=-3.5,
 to=0)
   } else {
      temp <- list(x = 3.5*(seq(1, 512) - 1)/512, y = rep(0.001, 512))
   }
   phi.densities.neg[, i] <- temp$y
   norm.factor <- sum(phi.densities.neg[, i])
   phi.densities.neg[, i] <- phi.densities.neg[, i]/norm.factor
   if (i == 1) {
      x.coor.neg <- temp$x
   }
   pos.phi <- obs.phi.norm[obs.phi.norm[, i] >= 0, i]
   if (length(pos.phi) > 2) {
      temp <- density(pos.phi, adjust=adjust.param, n = 512, from=0,
 to=3.5)
   } else {
      temp <- list(x = 3.5*(seq(1, 512) - 1)/512, y = rep(0.001, 512))
   }
   obs.phi.densities.pos[, i] <- temp$y
   norm.factor <- sum(obs.phi.densities.pos[, i])
   obs.phi.densities.pos[, i] <- obs.phi.densities.pos[, i]/norm.factor

   neg.phi <- obs.phi.norm[obs.phi.norm[, i] < 0, i]
   if (length(neg.phi)> 2) {
      temp <- density(neg.phi, adjust=adjust.param, n = 512, from=-3.5,
 to=0)
   } else {
      temp <- list(x = 3.5*(seq(1, 512) - 1)/512, y = rep(0.001, 512))
   }
```

```
      obs.phi.densities.neg[, i] <- temp$y
      norm.factor <- sum(obs.phi.densities.neg[, i])
      obs.phi.densities.neg[, i] <- obs.phi.densities.neg[, i]/norm.factor

    }
    phi.density.mean.pos <- apply(phi.densities.pos, 1, mean)
    phi.density.mean.neg <- apply(phi.densities.neg, 1, mean)

    obs.phi.density.mean.pos <- apply(obs.phi.densities.pos, 1, mean)
    obs.phi.density.mean.neg <- apply(obs.phi.densities.neg, 1, mean)

    phi.density.median.pos <- apply(phi.densities.pos, 1, median)
    phi.density.median.neg <- apply(phi.densities.neg, 1, median)

    obs.phi.density.median.pos <- apply(obs.phi.densities.pos, 1, median)
    obs.phi.density.median.neg <- apply(obs.phi.densities.neg, 1, median)

    x <- c(x.coor.neg, x.coor.pos)
    x.plot.range <- range(x)
    y1 <- c(phi.density.mean.neg, phi.density.mean.pos)
    y2 <- c(obs.phi.density.mean.neg, obs.phi.density.mean.pos)
    y.plot.range <- c(-0.3*max(c(y1, y2)),  max(c(y1, y2)))

    print(c(y.plot.range, max(c(y1, y2)), max(y1), max(y2)))

    plot(x, y1, xlim = x.plot.range, ylim = 1.5*y.plot.range, type = "l",
     lwd = 2, col = 2, xlab = "NES", ylab = "P(NES)", main = "Global Observed
     and Null Densities (Area Normalized)")

   y1.point <- y1[seq(1, length(x), 2)]
   y2.point <- y2[seq(2, length(x), 2)]
   x1.point <- x[seq(1, length(x), 2)]
   x2.point <- x[seq(2, length(x), 2)]

#    for (i in 1:length(x1.point)) {
#      lines(c(x1.point[i], x1.point[i]), c(0, y1.point[i]), lwd = 3, cex =
     0.9, col = colors()[555]) # shading
#    }
#
#    for (i in 1:length(x2.point)) {
#      lines(c(x2.point[i], x2.point[i]), c(0, y2.point[i]), lwd = 3, cex =
     0.9, col = colors()[29]) # shading
#    }

    points(x, y1, type = "l", lwd = 2, col = colors()[555])
    points(x, y2, type = "l", lwd = 2, col = colors()[29])

    for (i in 1:Ng) {
       col <- ifelse(Obs.ES.norm[i] > 0, 2, 3)
       lines(c(Obs.ES.norm[i], Obs.ES.norm[i]), c(-0.2*max(c(y1, y2)), 0),
     lwd = 1, lty = 1, col = 1)
    }
    leg.txt <- paste("Neg. ES: \"", phen2, " \" ", sep="", collapse="")
    text(x=x.plot.range[1], y=-0.25*max(c(y1, y2)), adj = c(0, 1),
     labels=leg.txt, cex = 0.9)
    leg.txt <- paste(" Pos. ES: \"", phen1, "\" ", sep="", collapse="")
    text(x=x.plot.range[2], y=-0.25*max(c(y1, y2)), adj = c(1, 1),
     labels=leg.txt, cex = 0.9)

    leg.txt <- c("Null Density", "Observed Density", "Observed NES values")
    c.vec <- c(colors()[555], colors()[29], 1)
```

```
      lty.vec <- c(1, 1, 1)
      lwd.vec <- c(2, 2, 2)
      legend(x=0, y=1.5*y.plot.range[2], bty="n", bg = "white",
       legend=leg.txt, lty = lty.vec, lwd = lwd.vec, col = c.vec, cex = 0.9)

      B <- A[obs.index,]
      if (N > 300) {
        C <- rbind(B[1:100,], rep(0, Ns), rep(0, Ns), B[(floor(N/2) - 50 +
       1):(floor(N/2) + 50),], rep(0, Ns), rep(0, Ns), B[(N - 100 + 1):N,])
      }
      rm(B)
      GSEA.HeatMapPlot(V = C, col.labels = class.labels, col.classes =
       class.phen, main = "Heat Map for Genes in Dataset")

# p-vals plot
      nom.p.vals <- p.vals[Obs.ES.index,1]
      FWER.p.vals <- p.vals[Obs.ES.index,2]
      plot.range <- 1.25*range(NES)
      plot(NES, FDR.mean, ylim = c(0, 1), xlim = plot.range, col = 1, bg = 1,
       type="p", pch = 22, cex = 0.75, xlab = "NES", main = "p-values vs. NES",
       ylab ="p-val/q-val")
      points(NES, nom.p.vals, type = "p", col = 2, bg = 2, pch = 22, cex =
       0.75)
      points(NES, FWER.p.vals, type = "p", col = colors()[577], bg =
       colors()[577], pch = 22, cex = 0.75)
      leg.txt <- c("Nominal p-value", "FWER p-value", "FDR q-value")
      c.vec <- c(2, colors()[577], 1)
      pch.vec <- c(22, 22, 22)
      legend(x=-0.5, y=0.5, bty="n", bg = "white", legend=leg.txt, pch =
       pch.vec, col = c.vec, pt.bg = c.vec, cex = 0.9)
      lines(c(min(NES), max(NES)), c(nom.p.val.threshold,
       nom.p.val.threshold), lwd = 1, lty = 2, col = 2)
      lines(c(min(NES), max(NES)), c(fwer.p.val.threshold,
       fwer.p.val.threshold), lwd = 1, lty = 2, col = colors()[577])
      lines(c(min(NES), max(NES)), c(fdr.q.val.threshold,
       fdr.q.val.threshold), lwd = 1, lty = 2, col = 1)

      if (non.interactive.run == F) {
          if (.Platform$OS.type == "windows") {
              savePlot(filename = glob.filename, type ="jpeg", device =
       dev.cur())
          } else if (.Platform$OS.type == "unix") {
              dev.off()
          }
      } else {
          dev.off()
      }

  } # if Ng > 1

#-------------------------------------------------------------------------
# Produce report for each gene set passing the nominal, FWER or FDR test or
       the top topgs in each side

      if (topgs > floor(Ng/2)) {
         topgs <- floor(Ng/2)
      }

      for (i in 1:Ng) {
         if ((p.vals[i, 1] <= nom.p.val.threshold) ||
             (p.vals[i, 2] <= fwer.p.val.threshold) ||
```

```
            (FDR.mean.sorted[i] <= fdr.q.val.threshold) ||
            (is.element(i, c(Obs.ES.index[1:topgs], Obs.ES.index[(Ng - topgs +
      1): Ng])))) {

#  produce report per gene set

          kk <- 1
          gene.number <- vector(length = size.G[i], mode = "character")
          gene.names <- vector(length = size.G[i], mode = "character")
          gene.symbols <- vector(length = size.G[i], mode = "character")
          gene.descs <- vector(length = size.G[i], mode = "character")
          gene.list.loc <- vector(length = size.G[i], mode = "numeric")
          core.enrichment <- vector(length = size.G[i], mode = "character")
          gene.s2n <- vector(length = size.G[i], mode = "numeric")
          gene.RES <- vector(length = size.G[i], mode = "numeric")
          rank.list <- seq(1, N)

          if (Obs.ES[i] >= 0) {
            set.k <- seq(1, N, 1)
            phen.tag <- phen1
            loc <- match(i, Obs.ES.index)
          } else {
            set.k <- seq(N, 1, -1)
            phen.tag <- phen2
            loc <- Ng - match(i, Obs.ES.index) + 1
          }

          for (k in set.k) {
             if (Obs.indicator[i, k] == 1) {
                gene.number[kk] <- kk
                gene.names[kk] <- obs.gene.labels[k]
                gene.symbols[kk] <- substr(obs.gene.symbols[k], 1, 15)
                gene.descs[kk] <- substr(obs.gene.descs[k], 1, 40)
                gene.list.loc[kk] <- k
                gene.s2n[kk] <- signif(obs.s2n[k], digits=3)
                gene.RES[kk] <- signif(Obs.RES[i, k], digits = 3)
                if (Obs.ES[i] >= 0) {
                   core.enrichment[kk] <- ifelse(gene.list.loc[kk] <=
      Obs.arg.ES[i], "YES", "NO")
                } else {
                   core.enrichment[kk] <- ifelse(gene.list.loc[kk] >
      Obs.arg.ES[i], "YES", "NO")
                }
                kk <- kk + 1
             }
          }

      gene.report <- data.frame(cbind(gene.number, gene.names, gene.symbols,
      gene.descs, gene.list.loc, gene.s2n, gene.RES, core.enrichment))
      names(gene.report) <- c("#", "GENE", "SYMBOL", "DESC", "LIST LOC",
      "S2N", "RES", "CORE_ENRICHMENT")

#      print(gene.report)

if (output.directory != "")  {

      filename <- paste(output.directory, doc.string, ".", gs.names[i],
      ".report.", phen.tag, ".", loc, ".txt", sep="", collapse="")
      write.table(gene.report, file = filename, quote=F, row.names=F, sep =
      "\t")
```

```
    if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
          gs.filename <- paste(output.directory, doc.string, ".",
    gs.names[i], ".plot.", phen.tag, ".", loc, sep="", collapse="")
          windows(width = 14, height = 6)
       } else if (.Platform$OS.type == "unix") {
          gs.filename <- paste(output.directory, doc.string, ".",
    gs.names[i], ".plot.", phen.tag, ".", loc, ".pdf", sep="", collapse="")
          pdf(file=gs.filename, height = 6, width = 14)
       }
    } else {
       if (.Platform$OS.type == "unix") {
         gs.filename <- paste(output.directory, doc.string, ".",
    gs.names[i], ".plot.", phen.tag, ".", loc, ".pdf", sep="", collapse="")
         pdf(file=gs.filename, height = 6, width = 14)
       } else if (.Platform$OS.type == "windows") {
         gs.filename <- paste(output.directory, doc.string, ".",
    gs.names[i], ".plot.", phen.tag, ".", loc, ".pdf", sep="", collapse="")
         pdf(file=gs.filename, height = 6, width = 14)
       }
    }

}

        nf <- layout(matrix(c(1,2,3), 1, 3, byrow=T), 1, c(1, 1, 1), TRUE)
        ind <- 1:N
        min.RES <- min(Obs.RES[i,])
        max.RES <- max(Obs.RES[i,])
        if (max.RES < 0.3) max.RES <- 0.3
        if (min.RES > -0.3) min.RES <- -0.3
        delta <- (max.RES - min.RES)*0.50
        min.plot <- min.RES - 2*delta
        max.plot <- max.RES
        max.corr <- max(obs.s2n)
        min.corr <- min(obs.s2n)
        Obs.correl.vector.norm <- (obs.s2n - min.corr)/(max.corr -
    min.corr)*1.25*delta + min.plot
        zero.corr.line <- (- min.corr/(max.corr - min.corr))*1.25*delta +
    min.plot
        col <- ifelse(Obs.ES[i] > 0, 2, 4)

        # Running enrichment plot

        sub.string <- paste("Number of genes: ", N, " (in list), ",
    size.G[i], " (in gene set)", sep = "", collapse="")

        main.string <- paste("Gene Set ", i, ":", gs.names[i])
        plot(ind, Obs.RES[i,], main = main.string, sub = sub.string, xlab =
    "Gene List Index", ylab = "Running Enrichment Score (RES)", xlim=c(1,
    N), ylim=c(min.plot, max.plot), type = "l", lwd = 2, cex = 1, col = col)
        for (j in seq(1, N, 20)) {
           lines(c(j, j), c(zero.corr.line, Obs.correl.vector.norm[j]), lwd
    = 1, cex = 1, col = colors()[12]) # shading of correlation plot
        }
        lines(c(1, N), c(0, 0), lwd = 1, lty = 2, cex = 1, col = 1) # zero
    RES line
        lines(c(Obs.arg.ES[i], Obs.arg.ES[i]), c(min.plot, max.plot), lwd =
    1, lty = 3, cex = 1, col = col) # max enrichment vertical line
        for (j in 1:N) {
          if (Obs.indicator[i, j] == 1) {
```

```
         lines(c(j, j), c(min.plot + 1.25*delta, min.plot +
1.75*delta), lwd = 1, lty = 1, cex = 1, col = 1)  # enrichment tags
       }
    }
    lines(ind, Obs.correl.vector.norm, type = "l", lwd = 1, cex = 1,
col = 1)
    lines(c(1, N), c(zero.corr.line, zero.corr.line), lwd = 1, lty = 1,
cex = 1, col = 1) # zero correlation horizontal line
    temp <- order(abs(obs.s2n), decreasing=T)
    arg.correl <- temp[N]
    lines(c(arg.correl, arg.correl), c(min.plot, max.plot), lwd = 1,
lty = 3, cex = 1, col = 3) # zero crossing correlation vertical line

    leg.txt <- paste("\"", phen1, "\" ", sep="", collapse="")
    text(x=1, y=min.plot, adj = c(0, 0), labels=leg.txt, cex = 1.0)

    leg.txt <- paste("\"", phen2, "\" ", sep="", collapse="")
    text(x=N, y=min.plot, adj = c(1, 0), labels=leg.txt, cex = 1.0)

    adjx <- ifelse(Obs.ES[i] > 0, 0, 1)

    leg.txt <- paste("Peak at ", Obs.arg.ES[i], sep="", collapse="")
    text(x=Obs.arg.ES[i], y=min.plot + 1.8*delta, adj = c(adjx, 0),
labels=leg.txt, cex = 1.0)

    leg.txt <- paste("Zero crossing at ", arg.correl, sep="",
collapse="")
    text(x=arg.correl, y=min.plot + 1.95*delta, adj = c(adjx, 0),
labels=leg.txt, cex = 1.0)

    # nominal p-val histogram

    sub.string <- paste("ES =", signif(Obs.ES[i], digits = 3), " NES
=", signif(Obs.ES.norm[i], digits=3), "Nom. p-val=", signif(p.vals[i,
1], digits = 3),"FWER=", signif(p.vals[i, 2], digits = 3), "FDR=",
signif(FDR.mean.sorted[i], digits = 3))
    temp <- density(phi[i,], adjust=adjust.param)
    x.plot.range <- range(temp$x)
    y.plot.range <- c(-0.125*max(temp$y), 1.5*max(temp$y))
    plot(temp$x, temp$y, type = "l", sub = sub.string, xlim =
x.plot.range, ylim = y.plot.range, lwd = 2, col = 2, main = "Gene Set
Null Distribution", xlab = "ES", ylab="P(ES)")
    x.loc <- which.min(abs(temp$x - Obs.ES[i]))
    lines(c(Obs.ES[i], Obs.ES[i]), c(0, temp$y[x.loc]), lwd = 2, lty =
1, cex = 1, col = 1)
    lines(x.plot.range, c(0, 0), lwd = 1, lty = 1, cex = 1, col = 1)

    leg.txt <- c("Gene Set Null Density", "Observed Gene Set ES value")
    c.vec <- c(2, 1)
    lty.vec <- c(1, 1)
    lwd.vec <- c(2, 2)
    legend(x=-0.2, y=y.plot.range[2], bty="n", bg = "white",
legend=leg.txt, lty = lty.vec, lwd = lwd.vec, col = c.vec, cex = 1.0)

    leg.txt <- paste("Neg. ES \"", phen2, "\" ", sep="", collapse="")
    text(x=x.plot.range[1], y=-0.1*max(temp$y), adj = c(0, 0),
labels=leg.txt, cex = 1.0)
    leg.txt <- paste(" Pos. ES: \"", phen1, "\" ", sep="", collapse="")
    text(x=x.plot.range[2], y=-0.1*max(temp$y), adj = c(1, 0),
labels=leg.txt, cex = 1.0)
```

```
          # create pinkogram for each gene set

          kk <- 1

          pinko <- matrix(0, nrow = size.G[i], ncol = cols)
          pinko.gene.names <- vector(length = size.G[i], mode = "character")
          for (k in 1:rows) {
             if (Obs.indicator[i, k] == 1) {
                pinko[kk,] <- A[obs.index[k],]
                pinko.gene.names[kk] <- obs.gene.symbols[k]
                kk <- kk + 1
             }
          }
          GSEA.HeatMapPlot(V = pinko, row.names = pinko.gene.names,
       col.labels = class.labels, col.classes = class.phen, col.names =
       sample.names, main =" Heat Map for Genes in Gene Set", xlab=" ", ylab="
       ")

      if (non.interactive.run == F) {
          if (.Platform$OS.type == "windows") {
             savePlot(filename = gs.filename, type ="jpeg", device =
       dev.cur())
          } else if (.Platform$OS.type == "unix") {
             dev.off()
          }
      } else {
          dev.off()
      }

        } # if p.vals thres

      } # loop over gene sets


  return(list(report1 = report.phen1, report2 = report.phen2))

}  # end of definition of GSEA.analysis


GSEA.write.gct <- function (gct, filename)
{
    f <- file(filename, "w")
    cat("#1.2", "\n", file = f, append = TRUE, sep = "")
    cat(dim(gct)[1], "\t", dim(gct)[2], "\n", file = f, append = TRUE, sep =
       "")
    cat("Name", "\t", file = f, append = TRUE, sep = "")
    cat("Description", file = f, append = TRUE, sep = "")
    names <- names(gct)
    cat("\t", names[1], file = f, append = TRUE, sep = "")
    for (j in 2:length(names)) {
       cat("\t", names[j], file = f, append = TRUE, sep = "")
    }
    cat("\n", file = f, append = TRUE, sep = "\t")
    oldWarn <- options(warn = -1)
    m <- matrix(nrow = dim(gct)[1], ncol = dim(gct)[2] +  2)
    m[, 1] <- row.names(gct)
    m[, 2] <- row.names(gct)
    index <- 3
    for (i in 1:dim(gct)[2]) {
       m[, index] <- gct[, i]
       index <- index + 1
```

```
    }
    write.table(m, file = f, append = TRUE, quote = FALSE, sep = "\t", eol =
       "\n", col.names = FALSE, row.names = FALSE)
    close(f)
    options(warn = 0)
    return(gct)
}


GSEA.ConsPlot <- function(V, col.names, main = " ", sub = " ", xlab=" ",
      ylab=" ") {

# Plots a heatmap plot of a consensus matrix

     cols <- length(V[1,])
     B <- matrix(0, nrow=cols, ncol=cols)
     max.val <- max(V)
     min.val <- min(V)
     for (i in 1:cols) {
        for (j in 1:cols) {
           k <- cols - i + 1
          B[k, j] <-  max.val - V[i, j] + min.val
         }
     }



#    col.map <- c(rainbow(100, s = 1.0, v = 0.75, start = 0.0, end = 0.75,
       gamma = 1.5), "#BBBBBB", "#333333", "#FFFFFF")
     col.map <- rev(c("#0000FF", "#4040FF", "#7070FF", "#8888FF", "#A9A9FF",
       "#D5D5FF", "#EEE5EE", "#FFAADA", "#FF9DB0", "#FF7080", "#FF5A5A",
       "#FF4040", "#FF0D1D"))

#    max.size <- max(nchar(col.names))
     par(mar = c(5, 15, 15, 5))
     image(1:cols, 1:cols, t(B), col = col.map, axes=FALSE, main=main,
       sub=sub, xlab= xlab, ylab=ylab)

     for (i in 1:cols) {
        col.names[i]  <- substr(col.names[i], 1, 25)
     }
     col.names2 <- rev(col.names)

     size.col.char <- ifelse(cols < 15, 1, sqrt(15/cols))

     axis(2, at=1:cols, labels=col.names2, adj= 0.5, tick=FALSE, las = 1,
       cex.axis=size.col.char, font.axis=1, line=-1)
     axis(3, at=1:cols, labels=col.names, adj= 1, tick=FALSE, las = 3,
       cex.axis=size.col.char, font.axis=1, line=-1)

     return()
}

GSEA.HeatMapPlot2 <- function(V, row.names = "NA", col.names = "NA", main = "
      ", sub = " ", xlab=" ", ylab=" ", color.map = "default") {
#
# Plots a heatmap of a matrix

      n.rows <- length(V[,1])
      n.cols <- length(V[1,])
```

```
      if (color.map == "default") {
        color.map <- rev(rainbow(100, s = 1.0, v = 0.75, start = 0.0, end =
      0.75, gamma = 1.5))
      }

       heatm <- matrix(0, nrow = n.rows, ncol = n.cols)
       heatm[1:n.rows,] <- V[seq(n.rows, 1, -1),]

       par(mar = c(7, 15, 5, 5))
       image(1:n.cols, 1:n.rows, t(heatm), col=color.map, axes=FALSE,
      main=main, sub = sub, xlab= xlab, ylab=ylab)

       if (length(row.names) > 1) {
           size.row.char <- ifelse(n.rows < 15, 1, sqrt(15/n.rows))
           size.col.char <- ifelse(n.cols < 15, 1, sqrt(10/n.cols))
#           size.col.char <- ifelse(n.cols < 2.5, 1, sqrt(2.5/n.cols))
           for (i in 1:n.rows) {
              row.names[i] <- substr(row.names[i], 1, 40)
           }
           row.names <- row.names[seq(n.rows, 1, -1)]
           axis(2, at=1:n.rows, labels=row.names, adj= 0.5, tick=FALSE, las =
      1, cex.axis=size.row.char, font.axis=1, line=-1)
       }

       if (length(col.names) > 1) {
          axis(1, at=1:n.cols, labels=col.names, tick=FALSE, las = 3,
      cex.axis=size.col.char, font.axis=2, line=-1)
       }

      return()
}


GSEA.Analyze.Sets <- function(
   directory,
   topgs = "",
   non.interactive.run = F,
   height = 12,
   width = 17) {

   file.list <- list.files(directory)
   files <- file.list[regexpr(pattern = "\.report\.", file.list) > 1]
   max.sets <- length(files)

   set.table <- matrix(nrow = max.sets, ncol = 5)

   for (i in 1:max.sets) {
      temp1 <-  strsplit(files[i], split="\.report\.")
      temp2 <-  strsplit(temp1[[1]][1], split="\\.")
      s <- length(temp2[[1]])
      prefix.name <- paste(temp2[[1]][1:(s-1)], sep="", collapse="")
      set.name <- temp2[[1]][s]
      temp3 <-  strsplit(temp1[[1]][2], split="\\.")
      phenotype <- temp3[[1]][1]
      seq.number <- temp3[[1]][2]
      dataset <- paste(temp2[[1]][1:(s-1)], sep="", collapse="\.")

      set.table[i, 1] <- files[i]

      set.table[i, 3] <- phenotype
      set.table[i, 4] <- as.numeric(seq.number)
```

```
      set.table[i, 5] <- dataset

#     set.table[i, 2] <- paste(set.name, dataset, sep ="", collapse="")
      set.table[i, 2] <- substr(set.name, 1, 20)
   }

   print(c("set name=", prefix.name))
   doc.string <- prefix.name

   set.table <- noquote(set.table)
   phen.order <- order(set.table[, 3], decreasing = T)
   set.table <- set.table[phen.order,]
   phen1 <- names(table(set.table[,3]))[1]
   phen2 <- names(table(set.table[,3]))[2]
   set.table.phen1 <- set.table[set.table[,3] == phen1,]
   set.table.phen2 <- set.table[set.table[,3] == phen2,]

   seq.order <- order(as.numeric(set.table.phen1[, 4]), decreasing = F)
   set.table.phen1 <- set.table.phen1[seq.order,]
   seq.order <- order(as.numeric(set.table.phen2[, 4]), decreasing = F)
   set.table.phen2 <- set.table.phen2[seq.order,]

#  max.sets.phen1 <- length(set.table.phen1[,1])
#  max.sets.phen2 <- length(set.table.phen2[,1])

   if (topgs == "") {
      max.sets.phen1 <- length(set.table.phen1[,1])
      max.sets.phen2 <- length(set.table.phen2[,1])
   } else {
      max.sets.phen1 <- ifelse(topgs > length(set.table.phen1[,1]),
       length(set.table.phen1[,1]), topgs)
      max.sets.phen2 <- ifelse(topgs > length(set.table.phen2[,1]),
       length(set.table.phen2[,1]), topgs)
   }

   # Analysis for phen1

   leading.lists <- NULL
   for (i in 1:max.sets.phen1) {
      inputfile <- paste(directory, set.table.phen1[i, 1], sep="",
       collapse="")
      gene.set <- read.table(file=inputfile, sep="\t", header=T,
       comment.char="", as.is=T)
      leading.set <- as.vector(gene.set[gene.set[,"CORE_ENRICHMENT"] == "YES",
       "SYMBOL"])
      leading.lists <- c(leading.lists, list(leading.set))
      if (i == 1) {
         all.leading.genes <- leading.set
      } else{
         all.leading.genes <- union(all.leading.genes, leading.set)
      }
   }
   max.genes <- length(all.leading.genes)
   M <- matrix(0, nrow=max.sets.phen1, ncol=max.genes)
   for (i in 1:max.sets.phen1) {
      M[i,] <- sign(match(all.leading.genes, as.vector(leading.lists[[i]]),
       nomatch=0))  # notice that the sign is 0 (no tag) or 1 (tag)
   }

   Inter <- matrix(0, nrow=max.sets.phen1, ncol=max.sets.phen1)
   for (i in 1:max.sets.phen1) {
```

```
   for (j in 1:max.sets.phen1) {
      Inter[i, j] <- length(intersect(leading.lists[[i]],
    leading.lists[[j]]))/length(union(leading.lists[[i]],
    leading.lists[[j]]))
   }
}

Itable <- data.frame(Inter)
names(Itable) <- set.table.phen1[1:max.sets.phen1, 2]
row.names(Itable) <- set.table.phen1[1:max.sets.phen1, 2]

if (non.interactive.run == F) {
     if (.Platform$OS.type == "windows") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen1,
    sep="", collapse="")
        windows(height = width, width = width)
     } else if (.Platform$OS.type == "unix") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen1,
    ".pdf", sep="", collapse="")
        pdf(file=filename, height = width, width = width)
     }
} else {
     if (.Platform$OS.type == "unix") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen1,
    ".pdf", sep="", collapse="")
        pdf(file=filename, height = width, width = width)
     } else if (.Platform$OS.type == "windows") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen1,
    ".pdf", sep="", collapse="")
        pdf(file=filename, height = width, width = width)
     }
}

GSEA.ConsPlot(Itable, col.names = set.table.phen1[1:max.sets.phen1, 2],
    main = " ", sub=paste("Leading Subsets Overlap ", doc.string, " - ",
    phen1, sep=""), xlab=" ", ylab=" ")

if (non.interactive.run == F) {
     if (.Platform$OS.type == "windows") {
         savePlot(filename = filename, type ="jpeg", device = dev.cur())
     } else if (.Platform$OS.type == "unix") {
         dev.off()
     }
} else {
     dev.off()
}

# Save leading subsets in a GCT file

D.phen1 <- data.frame(M)
names(D.phen1) <- all.leading.genes
row.names(D.phen1) <- set.table.phen1[1:max.sets.phen1, 2]
output <- paste(directory, doc.string, ".leading.genes.", phen1, ".gct",
    sep="")
GSEA.write.gct(D.phen1, filename=output)

# Save leading subsets as a single gene set in a .gmt file

row.header <- paste(doc.string, ".all.leading.genes.", phen1, sep="")
output.line <- paste(all.leading.genes, sep="\t", collapse="\t")
```

```
   output.line <- paste(row.header, row.header, output.line, sep="\t",
       collapse="")
   output <- paste(directory, doc.string, ".all.leading.genes.", phen1,
       ".gmt", sep="")
   write(noquote(output.line), file = output, ncolumns = length(output.line))

   if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
          filename <- paste(directory, doc.string, ".leading.assignment.",
       phen1, sep="", collapse="")
          windows(height = height, width = width)
       } else if (.Platform$OS.type == "unix") {
          filename <- paste(directory, doc.string, ".leading.assignment.",
       phen1, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       }
   } else {
       if (.Platform$OS.type == "unix") {
          filename <- paste(directory, doc.string, ".leading.assignment.",
       phen1, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       } else if (.Platform$OS.type == "windows") {
          filename <- paste(directory, doc.string, ".leading.assignment.",
       phen1, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       }
   }

   cmap <-  c("#AAAAFF", "#111166")
   GSEA.HeatMapPlot2(V = data.matrix(D.phen1), row.names = row.names(D.phen1),
       col.names = names(D.phen1), main = "Leading Subsets Assignment", sub =
       paste(doc.string, " - ", phen1, sep=""), xlab=" ", ylab=" ", color.map =
       cmap)

   if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
           savePlot(filename = filename, type ="jpeg", device = dev.cur())
       } else if (.Platform$OS.type == "unix") {
           dev.off()
       }
   } else {
       dev.off()
   }

   DT1.phen1 <- data.matrix(t(D.phen1))
   DT2.phen1 <- data.frame(DT1.phen1)
   names(DT2.phen1) <- set.table.phen1[1:max.sets.phen1, 2]
   row.names(DT2.phen1) <- all.leading.genes
#   GSEA.write.gct(DT2.phen1, filename=outputfile2.phen1)

   # Analysis for phen2

   leading.lists <- NULL
   for (i in 1:max.sets.phen2) {
      inputfile <- paste(directory, set.table.phen2[i, 1], sep="",
       collapse="")
      gene.set <- read.table(file=inputfile, sep="\t", header=T,
       comment.char="", as.is=T)
      leading.set <- as.vector(gene.set[gene.set[,"CORE_ENRICHMENT"] == "YES",
       "SYMBOL"])
      leading.lists <- c(leading.lists, list(leading.set))
```

```
   if (i == 1) {
      all.leading.genes <- leading.set
   } else{
      all.leading.genes <- union(all.leading.genes, leading.set)
   }
}
max.genes <- length(all.leading.genes)
M <- matrix(0, nrow=max.sets.phen2, ncol=max.genes)
for (i in 1:max.sets.phen2) {
   M[i,] <- sign(match(all.leading.genes, as.vector(leading.lists[[i]]),
    nomatch=0))   # notice that the sign is 0 (no tag) or 1 (tag)
}

Inter <- matrix(0, nrow=max.sets.phen2, ncol=max.sets.phen2)
for (i in 1:max.sets.phen2) {
  for (j in 1:max.sets.phen2) {
     Inter[i, j] <- length(intersect(leading.lists[[i]],
    leading.lists[[j]]))/length(union(leading.lists[[i]],
    leading.lists[[j]]))
  }
}

Itable <- data.frame(Inter)
names(Itable) <- set.table.phen2[1:max.sets.phen2, 2]
row.names(Itable) <- set.table.phen2[1:max.sets.phen2, 2]

if (non.interactive.run == F) {
     if (.Platform$OS.type == "windows") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen2,
    sep="", collapse="")
        windows(height = width, width = width)
     } else if (.Platform$OS.type == "unix") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen2,
    ".pdf", sep="", collapse="")
        pdf(file=filename, height = width, width = width)
     }
} else {
     if (.Platform$OS.type == "unix") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen2,
    ".pdf", sep="", collapse="")
        pdf(file=filename, height = width, width = width)
     } else if (.Platform$OS.type == "windows") {
        filename <- paste(directory, doc.string, ".leading.overlap.", phen2,
    ".pdf", sep="", collapse="")
        pdf(file=filename, height = width, width = width)
     }
}

GSEA.ConsPlot(Itable, col.names = set.table.phen2[1:max.sets.phen2, 2],
    main = " ", sub=paste("Leading Subsets Overlap ", doc.string, " - ",
    phen2, sep=""), xlab=" ", ylab=" ")

if (non.interactive.run == F) {
     if (.Platform$OS.type == "windows") {
         savePlot(filename = filename, type ="jpeg", device = dev.cur())
     } else if (.Platform$OS.type == "unix") {
         dev.off()
     }
} else {
     dev.off()
}
```

```
# Save leading subsets in a GCT file

D.phen2 <- data.frame(M)
names(D.phen2) <- all.leading.genes
row.names(D.phen2) <- set.table.phen2[1:max.sets.phen2, 2]
output <- paste(directory, doc.string, ".leading.genes.", phen2, ".gct",
    sep="")
GSEA.write.gct(D.phen2, filename=output)

# Save primary subsets as a single gene set in a .gmt file

row.header <- paste(doc.string, ".all.leading.genes.", phen2, sep="")
output.line <- paste(all.leading.genes, sep="\t", collapse="\t")
output.line <- paste(row.header, row.header, output.line, sep="\t",
    collapse="")
output <- paste(directory, doc.string, ".all.leading.genes.", phen2,
    ".gmt", sep="")
write(noquote(output.line), file = output, ncolumns = length(output.line))

if (non.interactive.run == F) {
     if (.Platform$OS.type == "windows") {
        filename <- paste(directory, doc.string, ".leading.assignment.",
    phen2, sep="", collapse="")
        windows(height = height, width = width)
     } else if (.Platform$OS.type == "unix") {
        filename <- paste(directory, doc.string, ".leading.assignment.",
    phen2, ".pdf", sep="", collapse="")
        pdf(file=filename, height = height, width = width)
     }
} else {
     if (.Platform$OS.type == "unix") {
        filename <- paste(directory, doc.string, ".leading.assignment.",
    phen2, ".pdf", sep="", collapse="")
        pdf(file=filename, height = height, width = width)
     } else if (.Platform$OS.type == "windows") {
        filename <- paste(directory, doc.string, ".leading.assignment.",
    phen2, ".pdf", sep="", collapse="")
        pdf(file=filename, height = height, width = width)
     }
}

cmap <-  c("#AAAAFF", "#111166")
GSEA.HeatMapPlot2(V = data.matrix(D.phen2), row.names = row.names(D.phen2),
    col.names = names(D.phen2), main = "Leading Subsets Assignment", sub =
    paste(doc.string, " - ", phen2, sep=""), xlab=" ", ylab=" ", color.map =
    cmap)

if (non.interactive.run == F) {
     if (.Platform$OS.type == "windows") {
         savePlot(filename = filename, type ="jpeg", device = dev.cur())
     } else if (.Platform$OS.type == "unix") {
         dev.off()
     }
} else {
     dev.off()
}

DT1.phen2 <- data.matrix(t(D.phen2))
DT2.phen2 <- data.frame(DT1.phen2)
names(DT2.phen2) <- set.table.phen2[1:max.sets.phen2, 2]
```

```
   row.names(DT2.phen2) <- all.leading.genes
#   GSEA.write.gct(DT2.phen2, filename=outputfile2.phen2)

   # Resort columns and rows for phen1

   A <- data.matrix(D.phen1)
   A.row.names <- row.names(D.phen1)
   A.names <- names(D.phen1)

   # Max.genes

#   init <- 1
#   for (k in 1:max.sets.phen1) {
#      end <- which.max(cumsum(A[k,]))
#      if (end - init > 1) {
#         B <- A[,init:end]
#         B.names <- A.names[init:end]
#        dist.matrix <- dist(t(B))
#         HC <- hclust(dist.matrix, method="average")
##         B <- B[,HC$order] + 0.2*(k %% 2)
#        B <- B[,HC$order]
#         A[,init:end] <- B
#         A.names[init:end] <- B.names[HC$order]
#         init <- end + 1
#     }
#   }

#   windows(width=14, height=10)
#   GSEA.HeatMapPlot2(V = A, row.names = A.row.names, col.names = A.names, sub
      = "  ", main = paste("Primary Sets Assignment - ", doc.string, " - ",
      phen1, sep=""), xlab=" ", ylab=" ")

   dist.matrix <- dist(t(A))
   HC <- hclust(dist.matrix, method="average")
   A <- A[, HC$order]
   A.names <- A.names[HC$order]

   dist.matrix <- dist(A)
   HC <- hclust(dist.matrix, method="average")
   A <- A[HC$order,]
   A.row.names <- A.row.names[HC$order]

   if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
          filename <- paste(directory, doc.string,
      ".leading.assignment.clustered.", phen1, sep="", collapse="")
          windows(height = height, width = width)
       } else if (.Platform$OS.type == "unix") {
          filename <- paste(directory, doc.string,
      ".leading.assignment.clustered.", phen1, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       }
   } else {
       if (.Platform$OS.type == "unix") {
          filename <- paste(directory, doc.string,
      ".leading.assignment.clustered.", phen1, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       } else if (.Platform$OS.type == "windows") {
          filename <- paste(directory, doc.string,
      ".leading.assignment.clustered.", phen1, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
```

```
      }
   }

   cmap <-  c("#AAAAFF", "#111166")
#   GSEA.HeatMapPlot2(V = A, row.names = A.row.names, col.names = A.names,
      main = "Leading Subsets Assignment (clustered)", sub = paste(doc.string,
      " - ", phen1, sep=""), xlab=" ", ylab=" ", color.map = cmap)

   GSEA.HeatMapPlot2(V = t(A), row.names = A.names, col.names = A.row.names,
      main = "Leading Subsets Assignment (clustered)", sub = paste(doc.string,
      " - ", phen1, sep=""), xlab=" ", ylab=" ", color.map = cmap)

    text.filename <- paste(directory, doc.string,
      ".leading.assignment.clustered.", phen1, ".txt", sep="", collapse="")
    line.list <- c("Gene", A.row.names)
    line.header <- paste(line.list, collapse="\t")
    line.length <- length(A.row.names) + 1
    write(line.header, file = text.filename, ncolumns = line.length)
    write.table(t(A), file=text.filename, append = T, quote=F, col.names= F,
      row.names=T, sep = "\t")

   if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
          savePlot(filename = filename, type ="jpeg", device = dev.cur())
       } else if (.Platform$OS.type == "unix") {
          dev.off()
       }
   } else {
       dev.off()
   }


# resort columns and rows for phen2

   A <- data.matrix(D.phen2)
   A.row.names <- row.names(D.phen2)
   A.names <- names(D.phen2)

   # Max.genes

#   init <- 1
#   for (k in 1:max.sets.phen2) {
#      end <- which.max(cumsum(A[k,]))
#      if (end - init > 1) {
#         B <- A[,init:end]
#         B.names <- A.names[init:end]
#         dist.matrix <- dist(t(B))
#         HC <- hclust(dist.matrix, method="average")
##         B <- B[,HC$order] + 0.2*(k %% 2)
#        B <- B[,HC$order]
#         A[,init:end] <- B
#         A.names[init:end] <- B.names[HC$order]
#         init <- end + 1
#      }
#   }

#  windows(width=14, height=10)
#  GESA.HeatMapPlot2(V = A, row.names = A.row.names, col.names = A.names, sub
      = "  ", main = paste("Primary Sets Assignment - ", doc.string, " - ",
      phen2, sep=""), xlab=" ", ylab=" ")
```

```
   dist.matrix <- dist(t(A))
   HC <- hclust(dist.matrix, method="average")
   A <- A[, HC$order]
   A.names <- A.names[HC$order]

   dist.matrix <- dist(A)
   HC <- hclust(dist.matrix, method="average")
   A <- A[HC$order,]
   A.row.names <- A.row.names[HC$order]

   if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
          filename <- paste(directory, doc.string,
       ".leading.assignment.clustered.", phen2, sep="", collapse="")
          windows(height = height, width = width)
       } else if (.Platform$OS.type == "unix") {
          filename <- paste(directory, doc.string,
       ".leading.assignment.clustered.", phen2, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       }
   } else {
       if (.Platform$OS.type == "unix") {
          filename <- paste(directory, doc.string,
       ".leading.assignment.clustered.", phen2, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       } else if (.Platform$OS.type == "windows") {
          filename <- paste(directory, doc.string,
       ".leading.assignment.clustered.", phen2, ".pdf", sep="", collapse="")
          pdf(file=filename, height = height, width = width)
       }
   }

   cmap <-  c("#AAAAFF", "#111166")

#   GSEA.HeatMapPlot2(V = A, row.names = A.row.names, col.names = A.names,
       main = "Leading Subsets Assignment (clustered)", sub = paste(doc.string,
       " - ", phen2, sep=""), xlab=" ", ylab=" ", color.map = cmap)
   GSEA.HeatMapPlot2(V = t(A), row.names =A.names , col.names = A.row.names,
       main = "Leading Subsets Assignment (clustered)", sub = paste(doc.string,
       " - ", phen2, sep=""), xlab=" ", ylab=" ", color.map = cmap)

    text.filename <- paste(directory, doc.string,
       ".leading.assignment.clustered.", phen2, ".txt", sep="", collapse="")
    line.list <- c("Gene", A.row.names)
    line.header <- paste(line.list, collapse="\t")
    line.length <- length(A.row.names) + 1
    write(line.header, file = text.filename, ncolumns = line.length)
    write.table(t(A), file=text.filename, append = T, quote=F, col.names= F,
       row.names=T, sep = "\t")

   if (non.interactive.run == F) {
       if (.Platform$OS.type == "windows") {
           savePlot(filename = filename, type ="jpeg", device = dev.cur())
       } else if (.Platform$OS.type == "unix") {
           dev.off()
       }
   } else {
       dev.off()
   }

}
```

APPENDIX B: R code for real example: Male vs. Female Lymphoblastoid Cells

```
GSEA.program.location <- "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
      R.1.0/GSEA.1.0.R"  #  R source program (change pathname to the rigth
      location in local machine)
source(GSEA.program.location, verbose=T, max.deparse.length=9999)

GSEA(
      # Input/Output Files :-------------------------------------------
 input.ds =  "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
      R.1.0/Datasets/Gender.gct",            # Input gene expression Affy
      dataset file in RES or GCT format
 input.cls = "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
      R.1.0/Datasets/Gender.cls",            # Input class vector (phenotype)
      file in CLS format
 gs.db =    "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
      R.1.0/GeneSetDatabases/tryC1.gmt",         # Gene set database in GMT
      format
 output.directory     = "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
      R.1.0/Gender_C1/",          # Directory where to store output and
      results (default: "")
#  Program parameters :----------------------------------------------------
 doc.string          = "try_C1",    # Documentation string used as a prefix
      to name result files (default: "GSEA.analysis")
 non.interactive.run   = F,            # Run in interactive (i.e. R GUI) or
      batch (R command line) mode (default: F)
 reshuffling.type     = "sample.labels", # Type of permutation reshuffling:
      "sample.labels" or "gene.labels" (default: "sample.labels"
 nperm              = 50,          # Number of random permutations (default:
      1000)
 weighted.score.type   =  1,           # Enrichment correlation-based
      weighting: 0=no weight (KS), 1= weigthed, 2 = over-weigthed (default: 1)
 nom.p.val.threshold   = -1,           # Significance threshold for nominal
      p-vals for gene sets (default: -1, no thres)
 fwer.p.val.threshold  = -1,           # Significance threshold for FWER p-
      vals for gene sets (default: -1, no thres)
 fdr.q.val.threshold   = 0.25,         # Significance threshold for FDR q-
      vals for gene sets (default: 0.25)
 topgs               = 20,           # Besides those passing test, number of
      top scoring gene sets used for detailed reports (default: 10)
 adjust.FDR.q.val     = F,            # Adjust the FDR q-vals (default: F)
 gs.size.threshold.min = 5,           # Minimum size (in genes) for database
      gene sets to be considered (default: 25)
 gs.size.threshold.max = 500,          # Maximum size (in genes) for database
      gene sets to be considered (default: 500)
 reverse.sign        = F,            # Reverse direction of gene list (pos.
      enrichment becomes negative, etc.) (default: F)
 preproc.type        = 0,            # Preproc.normalization: 0=none,
      1=col(z-score)., 2=col(rank) and row(z-score)., 3=col(rank). (def: 0)
 random.seed         = 111,          # Random number generator seed.
      (default: 123456)
 perm.type           = 0,            # For experts only. Permutation type: 0
      = unbalanced, 1 = balanced (default: 0)
 fraction            = 1.0,          # For experts only. Subsampling
      fraction. Set to 1.0 (no resampling) (default: 1.0)
 replace             = F,            # For experts only, Resampling mode
      (replacement or not replacement) (default: F)
 save.intermediate.results = F,        # For experts only, save intermediate
      results (e.g. matrix of random perm. scores) (default: F)
```

```
 OLD.GSEA            = F,             # Use original (old) version of GSEA
      (default: F)
 use.fast.enrichment.routine = T         # Use faster routine to compute
      enrichment for random permutations (default: T)
)
```

APPENDIX C: Modified GSEA for power study

```
# G S E A -- Gene Set Enrichment Analysis

# Auxiliary functions and definitions

GSEA.GeneRanking <- function(A, class.labels, gene.labels, nperm,
      permutation.type = 0, sigma.correction = "GeneCluster", fraction=1.0,
      replace=F, reverse.sign= F) {

    A <- A + 0.00000001

    N <- length(A[,1])
    Ns <- length(A[1,])

    subset.mask <- matrix(0, nrow=Ns, ncol=nperm)
    reshuffled.class.labels1 <- matrix(0, nrow=Ns, ncol=nperm)
    reshuffled.class.labels2 <- matrix(0, nrow=Ns, ncol=nperm)
    class.labels1 <- matrix(0, nrow=Ns, ncol=nperm)
    class.labels2 <- matrix(0, nrow=Ns, ncol=nperm)

    order.matrix <- matrix(0, nrow = N, ncol = nperm)
    obs.order.matrix <- matrix(0, nrow = N, ncol = nperm)
    s2n.matrix <- matrix(0, nrow = N, ncol = nperm)
    obs.s2n.matrix <- matrix(0, nrow = N, ncol = nperm)

    obs.gene.labels <- vector(length = N, mode="character")
    obs.gene.descs <- vector(length = N, mode="character")
    obs.gene.symbols <- vector(length = N, mode="character")

    M1 <- matrix(0, nrow = N, ncol = nperm)
    M2 <- matrix(0, nrow = N, ncol = nperm)
    S1 <- matrix(0, nrow = N, ncol = nperm)
    S2 <- matrix(0, nrow = N, ncol = nperm)

    gc()

    C <- split(class.labels, class.labels)
    class1.size <- length(C[[1]])
    class2.size <- length(C[[2]])
    class1.index <- seq(1, class1.size, 1)
    class2.index <- seq(class1.size + 1, class1.size + class2.size, 1)

    for (r in 1:nperm) {
       class1.subset <- sample(class1.index, size =
      ceiling(class1.size*fraction), replace = replace)
       class2.subset <- sample(class2.index, size =
      ceiling(class2.size*fraction), replace = replace)
       class1.subset.size <- length(class1.subset)
       class2.subset.size <- length(class2.subset)
       subset.class1 <- rep(0, class1.size)
       for (i in 1:class1.size) {
           if (is.element(class1.index[i], class1.subset)) {
               subset.class1[i] <- 1
           }
       }
       subset.class2 <- rep(0, class2.size)
       for (i in 1:class2.size) {
           if (is.element(class2.index[i], class2.subset)) {
               subset.class2[i] <- 1
```

```
           }
        }
        subset.mask[, r] <- as.numeric(c(subset.class1, subset.class2))
        fraction.class1 <- class1.size/Ns
        fraction.class2 <- class2.size/Ns

        if (permutation.type == 0) { # random (unbalanced) permutation
           full.subset <- c(class1.subset, class2.subset)
           label1.subset <- sample(full.subset, size = Ns * fraction.class1)
           reshuffled.class.labels1[, r] <- rep(0, Ns)
           reshuffled.class.labels2[, r] <- rep(0, Ns)
           class.labels1[, r] <- rep(0, Ns)
           class.labels2[, r] <- rep(0, Ns)
           for (i in 1:Ns) {
              m1 <- sum(!is.na(match(label1.subset, i)))
              m2 <- sum(!is.na(match(full.subset, i)))
              reshuffled.class.labels1[i, r] <- m1
              reshuffled.class.labels2[i, r] <- m2 - m1
              if (i <= class1.size) {
                class.labels1[i, r] <- m2
                class.labels2[i, r] <- 0
              } else {
                 class.labels1[i, r] <- 0
                 class.labels2[i, r] <- m2
              }
           }

        }
     }

# compute S2N for the random permutation matrix

     P <- reshuffled.class.labels1 * subset.mask
     n1 <- sum(P[,1])
     M1 <- A %*% P
     M1 <- M1/n1
     gc()
     A2 <- A*A
     S1 <- A2 %*% P
     S1 <- S1/n1 - M1*M1
     S1 <- sqrt(abs((n1/(n1-1)) * S1))
     gc()
     P <- reshuffled.class.labels2 * subset.mask
     n2 <- sum(P[,1])
     M2 <- A %*% P
     M2 <- M2/n2
     gc()
     A2 <- A*A
     S2 <- A2 %*% P
     S2 <- S2/n2 - M2*M2
     S2 <- sqrt(abs((n2/(n2-1)) * S2))
     rm(P)
     rm(A2)
     gc()

     if (sigma.correction == "GeneCluster") {  # small sigma "fix" as used in
       GeneCluster
        S2 <- ifelse(0.2*abs(M2) < S2, S2, 0.2*abs(M2))
        S2 <- ifelse(S2 == 0, 0.2, S2)
        S1 <- ifelse(0.2*abs(M1) < S1, S1, 0.2*abs(M1))
        S1 <- ifelse(S1 == 0, 0.2, S1)
```

```
        gc()
    }

    M1 <- M1 - M2
    rm(M2)
    gc()
    S1 <- S1 + S2
    rm(S2)
    gc()

    s2n.matrix <- M1/S1

    if (reverse.sign == T) {
      s2n.matrix <- - s2n.matrix
    }
    gc()

    for (r in 1:nperm) {
      order.matrix[, r] <- order(s2n.matrix[, r], decreasing=T)
    }

# compute S2N for the "observed" permutation matrix

    P <- class.labels1 * subset.mask
    n1 <- sum(P[,1])
    M1 <- A %*% P
    M1 <- M1/n1
    gc()
    A2 <- A*A
    S1 <- A2 %*% P
    S1 <- S1/n1 - M1*M1
    S1 <- sqrt(abs((n1/(n1-1)) * S1))
    gc()
    P <- class.labels2 * subset.mask
    n2 <- sum(P[,1])
    M2 <- A %*% P
    M2 <- M2/n2
    gc()
    A2 <- A*A
    S2 <- A2 %*% P
    S2 <- S2/n2 - M2*M2
    S2 <- sqrt(abs((n2/(n2-1)) * S2))
    rm(P)
    rm(A2)
    gc()

    if (sigma.correction == "GeneCluster") {  # small sigma "fix" as used in
      GeneCluster
       S2 <- ifelse(0.2*abs(M2) < S2, S2, 0.2*abs(M2))
       S2 <- ifelse(S2 == 0, 0.2, S2)
       S1 <- ifelse(0.2*abs(M1) < S1, S1, 0.2*abs(M1))
       S1 <- ifelse(S1 == 0, 0.2, S1)
       gc()
    }

    M1 <- M1 - M2
    rm(M2)

    gc()
    S1 <- S1 + S2
    rm(S2)
```

```
    gc()

    obs.s2n.matrix <- M1/S1
    gc()

    if (reverse.sign == T) {
       obs.s2n.matrix <- - obs.s2n.matrix
    }

    for (r in 1:nperm) {
       obs.order.matrix[,r] <- order(obs.s2n.matrix[,r], decreasing=T)
    }

    return(list(s2n.matrix = s2n.matrix,
              obs.s2n.matrix = obs.s2n.matrix,
              order.matrix = order.matrix,
              obs.order.matrix = obs.order.matrix))
}

GSEA.EnrichmentScore <- function(gene.list, gene.set, weighted.score.type = 1,
      correl.vector = NULL) {
   tag.indicator <- sign(match(gene.list, gene.set, nomatch=0))   # notice
      that the sign is 0 (no tag) or 1 (tag)
   no.tag.indicator <- 1 - tag.indicator
   N <- length(gene.list)
   Nh <- length(gene.set)
   Nm <-  N - Nh
   if (weighted.score.type == 0) {
      correl.vector <- rep(1, N)
   }
   alpha <- weighted.score.type
   correl.vector <- abs(correl.vector**alpha)
   sum.correl.tag   <- sum(correl.vector[tag.indicator == 1])
   norm.tag   <- 1.0/sum.correl.tag
   norm.no.tag <- 1.0/Nm
   RES <- cumsum(tag.indicator * correl.vector * norm.tag - no.tag.indicator *
      norm.no.tag)
   max.ES <- max(RES)
   min.ES <- min(RES)
   if (max.ES > - min.ES) {
#     ES <- max.ES
     ES <- signif(max.ES, digits = 5)
     arg.ES <- which.max(RES)
   } else {
#     ES <- min.ES
     ES <- signif(min.ES, digits=5)
     arg.ES <- which.min(RES)
   }
   return(list(ES = ES, arg.ES = arg.ES, RES = RES, indicator =
      tag.indicator))
}


OLD.GSEA.EnrichmentScore <- function(gene.list, gene.set) {
#
# Computes the original GSEA score from Mootha et al 2003 of gene.set in
      gene.list
#
# Inputs:
#  gene.list: The ordered gene list (e.g. integers indicating the original
      position in the input dataset)
```

```
#  gene.set: A gene set (e.g. integers indicating the location of those genes
     in the input dataset)
#
# Outputs:
#  ES: Enrichment score (real number between -1 and +1)
#  arg.ES: Location in gene.list where the peak running enrichment occurs
     (peak of the "mountain")
#  RES: Numerical vector containing the running enrichment score for all
     locations in the gene list
#  tag.indicator: Binary vector indicating the location of the gene sets
     (1's) in the gene list
#

  tag.indicator <- sign(match(gene.list, gene.set, nomatch=0))   # notice
     that the sign is 0 (no tag) or 1 (tag)
  no.tag.indicator <- 1 - tag.indicator
  N <- length(gene.list)
  Nh <- length(gene.set)
  Nm <-  N - Nh

  norm.tag    <- sqrt((N - Nh)/Nh)
  norm.no.tag <- sqrt(Nh/(N - Nh))

  RES <- cumsum(tag.indicator * norm.tag - no.tag.indicator * norm.no.tag)
  max.ES <- max(RES)
  min.ES <- min(RES)
  if (max.ES > - min.ES) {
     ES <- signif(max.ES, digits=5)
     arg.ES <- which.max(RES)
  } else {

     ES <- signif(min.ES, digits=5)
     arg.ES <- which.min(RES)
  }
  return(list(ES = ES, arg.ES = arg.ES, RES = RES, indicator =
     tag.indicator))
}

GSEA.EnrichmentScore2 <- function(gene.list, gene.set, weighted.score.type =
     1, correl.vector = NULL) {

  N <- length(gene.list)
  Nh <- length(gene.set)
  Nm <-  N - Nh

  loc.vector <- vector(length=N, mode="numeric")
  peak.res.vector <- vector(length=Nh, mode="numeric")
  valley.res.vector <- vector(length=Nh, mode="numeric")
  tag.correl.vector <- vector(length=Nh, mode="numeric")
  tag.diff.vector <- vector(length=Nh, mode="numeric")
  tag.loc.vector <- vector(length=Nh, mode="numeric")

  loc.vector[gene.list] <- seq(1, N)
  tag.loc.vector <- loc.vector[gene.set]

  tag.loc.vector <- sort(tag.loc.vector, decreasing = F)

  if (weighted.score.type == 0) {
     tag.correl.vector <- rep(1, Nh)
  } else if (weighted.score.type == 1) {
     tag.correl.vector <- correl.vector[tag.loc.vector]
```

```
      tag.correl.vector <- abs(tag.correl.vector)
   } else if (weighted.score.type == 2) {
      tag.correl.vector <-
       correl.vector[tag.loc.vector]*correl.vector[tag.loc.vector]
      tag.correl.vector <- abs(tag.correl.vector)
   } else {
      tag.correl.vector <- correl.vector[tag.loc.vector]**weighted.score.type
      tag.correl.vector <- abs(tag.correl.vector)
   }

   norm.tag <- 1.0/sum(tag.correl.vector)
   tag.correl.vector <- tag.correl.vector * norm.tag
   norm.no.tag <- 1.0/Nm
   tag.diff.vector[1] <- (tag.loc.vector[1] - 1)
   tag.diff.vector[2:Nh] <- tag.loc.vector[2:Nh] - tag.loc.vector[1:(Nh - 1)]
       - 1
   tag.diff.vector <- tag.diff.vector * norm.no.tag
   peak.res.vector <- cumsum(tag.correl.vector - tag.diff.vector)
   valley.res.vector <- peak.res.vector - tag.correl.vector
   max.ES <- max(peak.res.vector)
   min.ES <- min(valley.res.vector)
   ES <- signif(ifelse(max.ES > - min.ES, max.ES, min.ES), digits=5)

   return(list(ES = ES))

}


GSEA.Res2Frame <- function(filename = "NULL") {
#
# Reads a gene expression dataset in RES format and converts it into an R data
      frame
#
   header.cont <- readLines(filename, n = 1)
   temp <- unlist(strsplit(header.cont, "\t"))
   colst <- length(temp)
   header.labels <- temp[seq(3, colst, 2)]
   ds <- read.delim(filename, header=F, row.names = 2, sep="\t", skip=3,
       blank.lines.skip=T, comment.char="", as.is=T)
   colst <- length(ds[1,])
   cols <- (colst - 1)/2
   rows <- length(ds[,1])
   A <- matrix(nrow=rows - 1, ncol=cols)
   A <- ds[1:rows, seq(2, colst, 2)]
   table1 <- data.frame(A)
   names(table1) <- header.labels
   return(table1)
}

GSEA.Gct2Frame <- function(filename = "NULL") {
#
# Reads a gene expression dataset in GCT format and converts it into an R data
      frame
#
   ds <- read.delim(filename, header=T, sep="\t", skip=2, row.names=1,
       blank.lines.skip=T, comment.char="", as.is=T)
   ds <- ds[-1]
   return(ds)
}

GSEA.Gct2Frame2 <- function(filename = "NULL") {
```

```
#
# Reads a gene expression dataset in GCT format and converts it into an R data
      frame
     content <- readLines(filename)
     content <- content[-1]
     content <- content[-1]
     col.names <- noquote(unlist(strsplit(content[1], "\t")))
     col.names <- col.names[c(-1, -2)]
     num.cols <- length(col.names)
     content <- content[-1]
     num.lines <- length(content)


     row.nam <- vector(length=num.lines, mode="character")
     row.des <- vector(length=num.lines, mode="character")
     m <- matrix(0, nrow=num.lines, ncol=num.cols)

     for (i in 1:num.lines) {
        line.list <- noquote(unlist(strsplit(content[i], "\t")))
        row.nam[i] <- noquote(line.list[1])
        row.des[i] <- noquote(line.list[2])
        line.list <- line.list[c(-1, -2)]

        for (j in 1:length(line.list)) {
           m[i, j] <- as.numeric(line.list[j])
        }
     }
     ds <- data.frame(m)
     names(ds) <- col.names
     row.names(ds) <- row.nam
     return(ds)
}

GSEA.ReadClsFile <- function(file = "NULL") {
#
# Reads a class vector CLS file and defines phenotype and class labels vectors
      for the samples in a gene expression file (RES or GCT format)

     cls.cont <- readLines(file)
     num.lines <- length(cls.cont)
     class.list <- unlist(strsplit(cls.cont[[3]], " "))
     s <- length(class.list)
     t <- table(class.list)
     l <- length(t)
     phen <- vector(length=l, mode="character")
     phen.label <- vector(length=l, mode="numeric")
     class.v <- vector(length=s, mode="numeric")
     for (i in 1:l) {
        phen[i] <- noquote(names(t)[i])
        phen.label[i] <- i - 1
     }
     for (i in 1:s) {
        for (j in 1:l) {
            if (class.list[i] == phen[j]) {
               class.v[i] <- phen.label[j]
            }
        }
     }
     return(list(phen = phen, class.v = class.v))
}
```

```
GSEA.Threshold <- function(V, thres, ceil) {
#
# Threshold and ceiling pre-processing for gene expression matrix
#

       V[V < thres] <- thres
       V[V > ceil] <- ceil
       return(V)
}

GSEA.VarFilter <- function(V, fold, delta, gene.names = "NULL") {
#
# Variation filter pre-processing for gene expression matrix
#

       cols <- length(V[1,])
       rows <- length(V[,1])
       row.max <- apply(V, MARGIN=1, FUN=max)
             row.min <- apply(V, MARGIN=1, FUN=min)
       flag <- array(dim=rows)
       flag <- (row.max /row.min > fold) & (row.max - row.min > delta)
       size <- sum(flag)
       B <- matrix(0, nrow = size, ncol = cols)
       j <- 1
       if (gene.names == "NULL") {
          for (i in 1:rows) {
             if (flag[i]) {
                B[j,] <- V[i,]
                j <- j + 1
              }
          }
       return(B)
       } else {
           new.list <- vector(mode = "character", length = size)
           for (i in 1:rows) {
             if (flag[i]) {
                B[j,] <- V[i,]
                new.list[j] <- gene.names[i]
                j <- j + 1
             }
           }
       return(list(V = B, new.list = new.list))
       }
}

GSEA.NormalizeRows <- function(V) {
#
# Stardardize rows of a gene expression matrix
       row.mean <- apply(V, MARGIN=1, FUN=mean)
             row.sd <- apply(V, MARGIN=1, FUN=sd)
       row.n <- length(V[,1])
       for (i in 1:row.n) {
           if (row.sd[i] == 0) {
               V[i,] <- 0
          } else {
             V[i,] <- (V[i,] - row.mean[i])/row.sd[i]
          }
       }
       return(V)
}
```

```
GSEA.NormalizeCols <- function(V) {
#
# Stardardize columns of a gene expression matrix

       col.mean <- apply(V, MARGIN=2, FUN=mean)
             col.sd <- apply(V, MARGIN=2, FUN=sd)
       col.n <- length(V[1,])
       for (i in 1:col.n) {
           if (col.sd[i] == 0) {
               V[i,] <- 0
         } else {
            V[,i] <- (V[,i] - col.mean[i])/col.sd[i]
         }
       }
       return(V)
}

#function generate random numbers of tube distribution

tube=function(n, nb, pi, low, up) {
      dis=100
      while ((dis>up) || (dis<low)) {
      a=rnorm(n)
#     b=pi*rnorm(n, nb, 1)+(1-pi)*runif(n, -1, 1)
      b=pi*rnorm(n, nb, 1)+(1-pi)*runif(n, -1, 0)
      s=ks.test(b,"pnorm", nb, 1)
      dis=s$statistic}
      x=c(a,b)
      return(x)
}

# end of auxiliary functions

# ------------------------------------------------------------------------
# Main GSEA Analysis Function that implements the entire methodology

GSEA <- function(
input.ds,
input.cls,
gene.ann = "",
gs.db,
gs.ann = "",

output.directory = "",
doc.string = "GSEA.analysis",
non.interactive.run = F,
reshuffling.type = "sample.labels",
nperm = 1000,
weighted.score.type = 1,
nom.p.val.threshold = -1,
fwer.p.val.threshold = -1,
fdr.q.val.threshold = 0.25,
topgs = 10,
adjust.FDR.q.val = F,
gs.size.threshold.min = 25,
gs.size.threshold.max = 500,
reverse.sign = F,
preproc.type = 0,
random.seed = 123456,
perm.type = 0,
fraction = 1.0,
```

```
replace = F,
save.intermediate.results = F,
OLD.GSEA = F,
use.fast.enrichment.routine = T) {

#
  print(" *** Running GSEA Analysis...")


# Start of GSEA methodology

  if (.Platform$OS.type == "windows") {
#    memory.limit(3000000000)
     memory.limit()
#     print(c("Start memory size=",  memory.size()))
  }

  # Read input data matrix

  adjust.param <- 0.5

  gc()

  time1 <- proc.time()

  # Read input class vector

  if(is.list(input.cls)) {
     CLS <- input.cls
  } else {
     CLS <- GSEA.ReadClsFile(file=input.cls)
  }
  class.labels <- CLS$class.v
  class.phen <- CLS$phen

  if (reverse.sign == T) {
     phen1 <- class.phen[2]
     phen2 <- class.phen[1]
  } else {
     phen1 <- class.phen[1]
     phen2 <- class.phen[2]
  }

  # sort samples according to phenotype

 col.index <- order(class.labels, decreasing=F)
 class.labels <- class.labels[col.index]
 sample.names <- sample.names[col.index]
 for (j in 1:rows) {
    A[j, ] <- A[j, col.index]
 }
 names(A) <- sample.names

  # Read input gene set database

 if (regexpr(pattern=".gmt", gs.db[1]) == -1) {
     temp <- gs.db
 } else {
     temp <- readLines(gs.db)
 }
```

```
    max.Ng <- length(temp)
    temp.size.G <- vector(length = max.Ng, mode = "numeric")
    for (i in 1:max.Ng) {
       temp.size.G[i] <- length(unlist(strsplit(temp[[i]], "\t"))) - 2
    }

    max.size.G <- max(temp.size.G)
    gs <- matrix(rep("null", max.Ng*max.size.G), nrow=max.Ng, ncol=
     max.size.G)
    temp.names <- vector(length = max.Ng, mode = "character")
    temp.desc <- vector(length = max.Ng, mode = "character")
    gs.count <- 1
    for (i in 1:max.Ng) {
       gene.set.size <- length(unlist(strsplit(temp[[i]], "\t"))) - 2
       gs.line <- noquote(unlist(strsplit(temp[[i]], "\t")))
       gene.set.name <- gs.line[1]
       gene.set.desc <- gs.line[2]
       gene.set.tags <- vector(length = gene.set.size, mode = "character")
       for (j in 1:gene.set.size) {
          gene.set.tags[j] <- gs.line[j + 2]
       }
       existing.set <- is.element(gene.set.tags, gene.labels)
       set.size <- length(existing.set[existing.set == T])
       if ((set.size < gs.size.threshold.min) || (set.size >
     gs.size.threshold.max)) next
       temp.size.G[gs.count] <- set.size
       gs[gs.count,] <- c(gene.set.tags[existing.set], rep("null",
     max.size.G - temp.size.G[gs.count]))
       temp.names[gs.count] <- gene.set.name
       temp.desc[gs.count] <- gene.set.desc
       gs.count <- gs.count + 1
    }
    Ng <- gs.count - 1
    gs.names <- vector(length = Ng, mode = "character")
    gs.desc <- vector(length = Ng, mode = "character")
    size.G <- vector(length = Ng, mode = "numeric")
    gs.names <- temp.names[1:Ng]
    gs.desc <- temp.desc[1:Ng]
    size.G <- temp.size.G[1:Ng]

  N <- length(A[,1])
  Ns <- length(A[1,])


#--------------------------------------------------------------------------
  for (i in 1:rows) {
      if (is.element(gene.labels[i],gs[3,])) {
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.2,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.175,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.15,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.125,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.1,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.075,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.05,1))
#           A[i,]=c(rnorm(cols/2,0,1),rnorm(cols/2, 0.025,1))
#           A[i,]=tube(cols/2, 0.2, 0.5, 0,0.3)
#           A[i,]=tube(cols/2, 0.1, 0.5, 0,0.14)
#           A[i,]=tube(cols/2, 0.2, 0.8, 0,0.15)
#           A[i,]=tube(cols/2, 0.2, 0.8, 0.15,0.2)
#           A[i,]=tube(cols/2, 0.2, 0.8, 0.2,0.25)
#           A[i,]=tube(cols/2, 0.2, 0.8, 0.15,0.25)
```

```
#           A[i,]=tube(cols/2, 0.2, 0.8, 0.25,0.35)
            A[i,]=tube(cols/2, 0.2, 0.8, 0.35,1)
} else {
            A[i,]=rnorm(cols,0,1)
      }
  }

  nrgs=gs.names[3]
#-------------------------------------------------------------------------

  print(c("Number of genes:", N))
  print(c("Number of Gene Sets:", Ng))
  print(c("Number of samples:", Ns))
  print(c("Original number of Gene Sets:", max.Ng))
  print(c("Maximum gene set size:", max.size.G))

# Read gene and gene set annotations if gene annotation file was provided

  all.gene.descs <- vector(length = N, mode ="character")
  all.gene.symbols <- vector(length = N, mode ="character")
  all.gs.descs <- vector(length = Ng, mode ="character")

  for (i in 1:N) {
       all.gene.descs[i] <- gene.labels[i]
       all.gene.symbols[i] <- gene.labels[i]
  }

  for (i in 1:Ng) {
       all.gs.descs[i] <- gs.desc[i]
  }

  Obs.indicator <- matrix(nrow= Ng, ncol=N)
  Obs.RES <- matrix(nrow= Ng, ncol=N)

  Obs.ES <- vector(length = Ng, mode = "numeric")
  Obs.arg.ES <- vector(length = Ng, mode = "numeric")
  Obs.ES.norm <- vector(length = Ng, mode = "numeric")


  time2 <- proc.time()

  # GSEA methodology

  # Compute observed and random permutation gene rankings

  obs.s2n <- vector(length=N, mode="numeric")
  signal.strength <- vector(length=Ng, mode="numeric")
  tag.frac <- vector(length=Ng, mode="numeric")
  gene.frac <- vector(length=Ng, mode="numeric")
  coherence.ratio <- vector(length=Ng, mode="numeric")
  obs.phi.norm <- matrix(nrow = Ng, ncol = nperm)
  correl.matrix <- matrix(nrow = N, ncol = nperm)
  obs.correl.matrix <- matrix(nrow = N, ncol = nperm)
  order.matrix <- matrix(nrow = N, ncol = nperm)
  obs.order.matrix <- matrix(nrow = N, ncol = nperm)

   nperm.per.call <- 100
   n.groups <- nperm %/% nperm.per.call
   n.rem <- nperm %% nperm.per.call
   n.perms <- c(rep(nperm.per.call, n.groups), n.rem)
   n.ends <- cumsum(n.perms)
```

```
  n.starts <- n.ends - n.perms + 1

  if (n.rem == 0) {
    n.tot <- n.groups
  } else {
    n.tot <- n.groups + 1
  }

for (nk in 1:n.tot) {
  call.nperm <- n.perms[nk]

  print(paste("Computing ranked list for actual and permuted
      phenotypes.......permutations: ", n.starts[nk], "--", n.ends[nk], sep="
      "))

  O <- GSEA.GeneRanking(A, class.labels, gene.labels, call.nperm,
      permutation.type = perm.type, sigma.correction = "GeneCluster",
      fraction=fraction, replace=replace, reverse.sign = reverse.sign)
  gc()

  order.matrix[,n.starts[nk]:n.ends[nk]] <- O$order.matrix
  obs.order.matrix[,n.starts[nk]:n.ends[nk]] <- O$obs.order.matrix
  correl.matrix[,n.starts[nk]:n.ends[nk]] <- O$s2n.matrix
  obs.correl.matrix[,n.starts[nk]:n.ends[nk]] <- O$obs.s2n.matrix
   rm(O)
}

 obs.s2n <- apply(obs.correl.matrix, 1, median)  # using median to assign
     enrichment scores
 obs.index <- order(obs.s2n, decreasing=T)
 obs.s2n  <- sort(obs.s2n, decreasing=T)

 obs.gene.labels <- gene.labels[obs.index]
 obs.gene.descs <- all.gene.descs[obs.index]
 obs.gene.symbols <- all.gene.symbols[obs.index]

 for (r in 1:nperm) {
    correl.matrix[, r] <- correl.matrix[order.matrix[,r], r]
 }
 for (r in 1:nperm) {
    obs.correl.matrix[, r] <- obs.correl.matrix[obs.order.matrix[,r], r]
 }

 gene.list2 <- obs.index
 for (i in 1:Ng) {
     print(paste("Computing observed enrichment for gene set:", i,
     gs.names[i], sep=" "))
     gene.set <- gs[i,gs[i,] != "null"]
     gene.set2 <- vector(length=length(gene.set), mode = "numeric")
     gene.set2 <- match(gene.set, gene.labels)
     if (OLD.GSEA == F) {
        GSEA.results <- GSEA.EnrichmentScore(gene.list=gene.list2,
     gene.set=gene.set2, weighted.score.type=weighted.score.type,
     correl.vector = obs.s2n)
     } else {
        GSEA.results <- OLD.GSEA.EnrichmentScore(gene.list=gene.list2,
     gene.set=gene.set2)
     }
     Obs.ES[i] <- GSEA.results$ES
     Obs.arg.ES[i] <- GSEA.results$arg.ES
     Obs.RES[i,] <- GSEA.results$RES
```

```
      Obs.indicator[i,] <- GSEA.results$indicator
      if (Obs.ES[i] >= 0) {  # compute signal strength
         tag.frac[i] <- sum(Obs.indicator[i,1:Obs.arg.ES[i]])/size.G[i]
         gene.frac[i] <- Obs.arg.ES[i]/N
      } else {
         tag.frac[i] <- sum(Obs.indicator[i, Obs.arg.ES[i]:N])/size.G[i]
         gene.frac[i] <- (N - Obs.arg.ES[i] + 1)/N
      }
      signal.strength[i] <- tag.frac[i] * (1 - gene.frac[i]) * (N / (N -
      size.G[i]))
   }

# Compute enrichment for random permutations

   phi <- matrix(nrow = Ng, ncol = nperm)
   phi.norm <- matrix(nrow = Ng, ncol = nperm)
   obs.phi <- matrix(nrow = Ng, ncol = nperm)

   if (reshuffling.type == "sample.labels") { # reshuffling phenotype labels
      for (i in 1:Ng) {
       print(paste("Computing random permutations' enrichment for gene set:",
      i, gs.names[i], sep=" "))
       gene.set <- gs[i,gs[i,] != "null"]
       gene.set2 <- vector(length=length(gene.set), mode = "numeric")
       gene.set2 <- match(gene.set, gene.labels)
       for (r in 1:nperm) {
           gene.list2 <- order.matrix[,r]
           if (use.fast.enrichment.routine == F) {
              GSEA.results <- GSEA.EnrichmentScore(gene.list=gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=correl.matrix[, r])
           } else {
              GSEA.results <- GSEA.EnrichmentScore2(gene.list=gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=correl.matrix[, r])
           }
           phi[i, r] <- GSEA.results$ES
       }
       if (fraction < 1.0) { # if resampling then compute ES for all observed
      rankings
           for (r in 1:nperm) {
              obs.gene.list2 <- obs.order.matrix[,r]
              if (use.fast.enrichment.routine == F) {
                 GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=obs.correl.matrix[, r])
              } else {
                 GSEA.results <-
      GSEA.EnrichmentScore2(gene.list=obs.gene.list2, gene.set=gene.set2,
      weighted.score.type=weighted.score.type,
      correl.vector=obs.correl.matrix[, r])
              }
              obs.phi[i, r] <- GSEA.results$ES
           }
       } else { # if no resampling then compute only one column (and fill the
      others with the same value)

            obs.gene.list2 <- obs.order.matrix[,1]
           if (use.fast.enrichment.routine == F) {
```

```
            GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
        } else {
            GSEA.results <- GSEA.EnrichmentScore2(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
        }
        obs.phi[i, 1] <- GSEA.results$ES
        for (r in 2:nperm) {
            obs.phi[i, r] <- obs.phi[i, 1]
        }
     }
     gc()
  }

} else if (reshuffling.type == "gene.labels") { # reshuffling gene labels
   for (i in 1:Ng) {
     gene.set <- gs[i,gs[i,] != "null"]
     gene.set2 <- vector(length=length(gene.set), mode = "numeric")
     gene.set2 <- match(gene.set, gene.labels)
     for (r in 1:nperm) {
        reshuffled.gene.labels <- sample(1:rows)
        if (use.fast.enrichment.routine == F) {
            GSEA.results <-
    GSEA.EnrichmentScore(gene.list=reshuffled.gene.labels,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.s2n)
        } else {
            GSEA.results <-
    GSEA.EnrichmentScore2(gene.list=reshuffled.gene.labels,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.s2n)
        }
        phi[i, r] <- GSEA.results$ES
     }
     if (fraction < 1.0) { # if resampling then compute ES for all observed
    rankings
        for (r in 1:nperm) {
           obs.gene.list2 <- obs.order.matrix[,r]
           if (use.fast.enrichment.routine == F) {
              GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
           } else {
              GSEA.results <- GSEA.EnrichmentScore2(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
           }
           obs.phi[i, r] <- GSEA.results$ES
        }
     } else { # if no resampling then compute only one column (and fill the
    others with the same value)
        obs.gene.list2 <- obs.order.matrix[,1]
        if (use.fast.enrichment.routine == F) {
           GSEA.results <- GSEA.EnrichmentScore(gene.list=obs.gene.list2,
    gene.set=gene.set2, weighted.score.type=weighted.score.type,
    correl.vector=obs.correl.matrix[, r])
        } else {
```

```
            GSEA.results <- GSEA.EnrichmentScore2(gene.list=obs.gene.list2,
      gene.set=gene.set2, weighted.score.type=weighted.score.type,
      correl.vector=obs.correl.matrix[, r])
         }
         obs.phi[i, 1] <- GSEA.results$ES
         for (r in 2:nperm) {
            obs.phi[i, r] <- obs.phi[i, 1]
         }
       }
       gc()
     }
   }

# Compute 3 types of p-values

# Find nominal p-values

print("Computing nominal p-values...")

p.vals <- matrix(0, nrow = Ng, ncol = 2)

if (OLD.GSEA == F) {
   for (i in 1:Ng) {
      pos.phi <- NULL
      neg.phi <- NULL
      for (j in 1:nperm) {
         if (phi[i, j] >= 0) {
            pos.phi <- c(pos.phi, phi[i, j])
         } else {
            neg.phi <- c(neg.phi, phi[i, j])
         }
      }
      ES.value <- Obs.ES[i]
      if (ES.value >= 0) {
         p.vals[i, 1] <- signif(sum(pos.phi >= ES.value)/length(pos.phi),
       digits=5)
      } else {
         p.vals[i, 1] <- signif(sum(neg.phi <= ES.value)/length(neg.phi),
       digits=5)
      }
   }
} else {  # For OLD GSEA compute the p-val using positive and negative values
       in the same histogram
   for (i in 1:Ng) {
      if (Obs.ES[i] >= 0) {
         p.vals[i, 1] <-  sum(phi[i,] >= Obs.ES[i])/length(phi[i,])
         p.vals[i, 1] <-  signif(p.vals[i, 1], digits=5)
      } else {
         p.vals[i, 1] <-  sum(phi[i,] <= Obs.ES[i])/length(phi[i,])
         p.vals[i, 1] <-  signif(p.vals[i, 1], digits=5)
      }
   }
}

# Find effective size

 erf <- function (x)
 {
    2 * pnorm(sqrt(2) * x)
 }
```

```
 KS.mean <- function(N) { # KS mean as a function of set size N
      S <- 0
      for (k in -100:100) {
         if (k == 0) next
         S <- S + 4 * (-1)**(k + 1) * (0.25 * exp(-2 * k * k * N) - sqrt(2 *
      pi) *  erf(sqrt(2 * N) * k)/(16 * k * sqrt(N)))
      }
     return(abs(S))
 }

# KS.mean.table <- vector(length=5000, mode="numeric")

# for (i in 1:5000) {
#    KS.mean.table[i] <- KS.mean(i)
# }

# KS.size <-  vector(length=Ng, mode="numeric")

# Rescaling normalization for each gene set null

print("Computing rescaling normalization for each gene set null...")

if (OLD.GSEA == F) {
   for (i in 1:Ng) {
        pos.phi <- NULL
        neg.phi <- NULL
        for (j in 1:nperm) {
           if (phi[i, j] >= 0) {
              pos.phi <- c(pos.phi, phi[i, j])
           } else {
              neg.phi <- c(neg.phi, phi[i, j])
           }
        }
        pos.m <- mean(pos.phi)
        neg.m <- mean(abs(neg.phi))

#        if (Obs.ES[i] >= 0) {
#           KS.size[i] <- which.min(abs(KS.mean.table - pos.m))
#        } else {
#           KS.size[i] <- which.min(abs(KS.mean.table - neg.m))
#        }

        pos.phi <- pos.phi/pos.m
        neg.phi <- neg.phi/neg.m
        for (j in 1:nperm) {
           if (phi[i, j] >= 0) {
              phi.norm[i, j] <- phi[i, j]/pos.m
           } else {
              phi.norm[i, j] <- phi[i, j]/neg.m
           }
         }
         for (j in 1:nperm) {
            if (obs.phi[i, j] >= 0) {
               obs.phi.norm[i, j] <- obs.phi[i, j]/pos.m
            } else {
               obs.phi.norm[i, j] <- obs.phi[i, j]/neg.m
            }
         }
         if (Obs.ES[i] >= 0) {
            Obs.ES.norm[i] <- Obs.ES[i]/pos.m
         } else {
```

```
            Obs.ES.norm[i] <- Obs.ES[i]/neg.m
          }
   }
} else {  # For OLD GSEA does not normalize using empirical scaling
   for (i in 1:Ng) {
      for (j in 1:nperm) {
         phi.norm[i, j] <- phi[i, j]/400
      }
      for (j in 1:nperm) {
         obs.phi.norm[i, j] <- obs.phi[i, j]/400
      }
      Obs.ES.norm[i] <- Obs.ES[i]/400
   }
}

# Compute FWER p-vals

     print("Computing FWER p-values...")

if (OLD.GSEA == F) {
    max.ES.vals.p <- NULL
    max.ES.vals.n <- NULL
    for (j in 1:nperm) {
       pos.phi <- NULL
       neg.phi <- NULL
       for (i in 1:Ng) {
          if (phi.norm[i, j] >= 0) {
             pos.phi <- c(pos.phi, phi.norm[i, j])
          } else {
             neg.phi <- c(neg.phi, phi.norm[i, j])
          }
       }
       if (length(pos.phi) > 0) {
          max.ES.vals.p <- c(max.ES.vals.p, max(pos.phi))
       }
       if (length(neg.phi) > 0) {
          max.ES.vals.n <- c(max.ES.vals.n, min(neg.phi))
       }
     }
   for (i in 1:Ng) {
       ES.value <- Obs.ES.norm[i]
       if (Obs.ES.norm[i] >= 0) {
          p.vals[i, 2] <- signif(sum(max.ES.vals.p >=
       ES.value)/length(max.ES.vals.p), digits=5)
       } else {
          p.vals[i, 2] <- signif(sum(max.ES.vals.n <=
       ES.value)/length(max.ES.vals.n), digits=5)
       }
     }
} else {  # For OLD GSEA compute the FWER using positive and negative values
       in the same histogram
    max.ES.vals <- NULL
    for (j in 1:nperm) {
       max.NES <- max(phi.norm[,j])
       min.NES <- min(phi.norm[,j])
       if (max.NES > - min.NES) {
          max.val <- max.NES
       } else {
          max.val <- min.NES
       }
       max.ES.vals <- c(max.ES.vals, max.val)
```

```
    }
    for (i in 1:Ng) {
       if (Obs.ES.norm[i] >= 0) {
          p.vals[i, 2] <- sum(max.ES.vals >=
       Obs.ES.norm[i])/length(max.ES.vals)
       } else {
          p.vals[i, 2] <- sum(max.ES.vals <=
       Obs.ES.norm[i])/length(max.ES.vals)
       }
       p.vals[i, 2] <-  signif(p.vals[i, 2], digits=4)
     }
}

# Compute FDRs

      print("Computing FDR q-values...")

      NES <- vector(length=Ng, mode="numeric")
      phi.norm.mean  <- vector(length=Ng, mode="numeric")
      obs.phi.norm.mean  <- vector(length=Ng, mode="numeric")
      phi.norm.median  <- vector(length=Ng, mode="numeric")
      obs.phi.norm.median  <- vector(length=Ng, mode="numeric")
      phi.norm.mean  <- vector(length=Ng, mode="numeric")
      obs.phi.mean  <- vector(length=Ng, mode="numeric")
      FDR.mean <- vector(length=Ng, mode="numeric")
      FDR.median <- vector(length=Ng, mode="numeric")
      phi.norm.median.d <- vector(length=Ng, mode="numeric")
      obs.phi.norm.median.d <- vector(length=Ng, mode="numeric")

      Obs.ES.index <- order(Obs.ES.norm, decreasing=T)
      Orig.index <- seq(1, Ng)
      Orig.index <- Orig.index[Obs.ES.index]
      Orig.index <- order(Orig.index, decreasing=F)
      Obs.ES.norm.sorted <- Obs.ES.norm[Obs.ES.index]
      gs.names.sorted <- gs.names[Obs.ES.index]

      for (k in 1:Ng) {
         NES[k] <- Obs.ES.norm.sorted[k]
         ES.value <- NES[k]
         count.col <- vector(length=nperm, mode="numeric")
         obs.count.col <- vector(length=nperm, mode="numeric")
         for (i in 1:nperm) {
            phi.vec <- phi.norm[,i]
            obs.phi.vec <- obs.phi.norm[,i]
            if (ES.value >= 0) {
               count.col.norm <- sum(phi.vec >= 0)
               obs.count.col.norm <- sum(obs.phi.vec >= 0)
               count.col[i] <- ifelse(count.col.norm > 0, sum(phi.vec >=
       ES.value)/count.col.norm, 0)
               obs.count.col[i] <- ifelse(obs.count.col.norm > 0,
       sum(obs.phi.vec >= ES.value)/obs.count.col.norm, 0)
            } else {
               count.col.norm <- sum(phi.vec < 0)
               obs.count.col.norm <- sum(obs.phi.vec < 0)
               count.col[i] <- ifelse(count.col.norm > 0, sum(phi.vec <=
       ES.value)/count.col.norm, 0)
               obs.count.col[i] <- ifelse(obs.count.col.norm > 0,
       sum(obs.phi.vec <= ES.value)/obs.count.col.norm, 0)
            }
          }
         phi.norm.mean[k] <- mean(count.col)
```

```
       obs.phi.norm.mean[k] <- mean(obs.count.col)
       phi.norm.median[k] <- median(count.col)
       obs.phi.norm.median[k] <- median(obs.count.col)
       FDR.mean[k] <- ifelse(phi.norm.mean[k]/obs.phi.norm.mean[k] < 1,
      phi.norm.mean[k]/obs.phi.norm.mean[k], 1)
       FDR.median[k] <- ifelse(phi.norm.median[k]/obs.phi.norm.median[k] < 1,
      phi.norm.median[k]/obs.phi.norm.median[k], 1)
     }

# adjust q-values

     if (adjust.FDR.q.val == T) {
        pos.nes <- length(NES[NES >= 0])
        min.FDR.mean <- FDR.mean[pos.nes]
        min.FDR.median <- FDR.median[pos.nes]
        for (k in seq(pos.nes - 1, 1, -1)) {
            if (FDR.mean[k] < min.FDR.mean) {
               min.FDR.mean <- FDR.mean[k]
            }
            if (min.FDR.mean < FDR.mean[k]) {
               FDR.mean[k] <- min.FDR.mean
            }
        }

        neg.nes <- pos.nes + 1
        min.FDR.mean <- FDR.mean[neg.nes]
        min.FDR.median <- FDR.median[neg.nes]
        for (k in seq(neg.nes + 1, Ng)) {
           if (FDR.mean[k] < min.FDR.mean) {
              min.FDR.mean <- FDR.mean[k]
           }
           if (min.FDR.mean < FDR.mean[k]) {
              FDR.mean[k] <- min.FDR.mean
           }
        }
     }

     obs.phi.norm.mean.sorted <- obs.phi.norm.mean[Orig.index]
     phi.norm.mean.sorted <- phi.norm.mean[Orig.index]
     FDR.mean.sorted <- FDR.mean[Orig.index]
     FDR.median.sorted <- FDR.median[Orig.index]

#   Compute global statistic

    glob.p.vals <- vector(length=Ng, mode="numeric")
    NULL.pass <- vector(length=nperm, mode="numeric")
    OBS.pass <- vector(length=nperm, mode="numeric")

    for (k in 1:Ng) {
      NES[k] <- Obs.ES.norm.sorted[k]
      if (NES[k] >= 0) {
         for (i in 1:nperm) {
            NULL.pos <- sum(phi.norm[,i] >= 0)
            NULL.pass[i] <- ifelse(NULL.pos > 0, sum(phi.norm[,i] >=
       NES[k])/NULL.pos, 0)
            OBS.pos <- sum(obs.phi.norm[,i] >= 0)
            OBS.pass[i] <- ifelse(OBS.pos > 0, sum(obs.phi.norm[,i] >=
       NES[k])/OBS.pos, 0)
         }
      } else {
         for (i in 1:nperm) {
```

```
            NULL.neg <- sum(phi.norm[,i] < 0)
            NULL.pass[i] <- ifelse(NULL.neg > 0, sum(phi.norm[,i] <=
       NES[k])/NULL.neg, 0)
            OBS.neg <- sum(obs.phi.norm[,i] < 0)
            OBS.pass[i] <- ifelse(OBS.neg > 0, sum(obs.phi.norm[,i] <=
       NES[k])/OBS.neg, 0)
          }
       }
       glob.p.vals[k] <- sum(NULL.pass >= mean(OBS.pass))/nperm
     }
     glob.p.vals.sorted <- glob.p.vals[Orig.index]

# Produce results report

      print("Producing result tables and plots...")

       Obs.ES <- signif(Obs.ES, digits=5)
       Obs.ES.norm <- signif(Obs.ES.norm, digits=5)
       p.vals <- signif(p.vals, digits=4)
       signal.strength <- signif(signal.strength, digits=3)
       tag.frac <- signif(tag.frac, digits=3)
       gene.frac <- signif(gene.frac, digits=3)
       FDR.mean.sorted <- signif(FDR.mean.sorted, digits=5)
       FDR.median.sorted <-  signif(FDR.median.sorted, digits=5)
       glob.p.vals.sorted <- signif(glob.p.vals.sorted, digits=5)

       report <- data.frame(cbind(gs.names, size.G, all.gs.descs, Obs.ES,
       Obs.ES.norm, p.vals[,1], FDR.mean.sorted, p.vals[,2], tag.frac,
       gene.frac, signal.strength, FDR.median.sorted, glob.p.vals.sorted))
       names(report) <- c("GS", "SIZE", "SOURCE", "ES", "NES", "NOM p-val",
       "FDR q-val", "FWER p-val", "Tag \%", "Gene \%", "Signal", "FDR
       (median)", "glob.p.val")
       report2 <- report
       report.index2 <- order(Obs.ES.norm, decreasing=T)
       for (i in 1:Ng) {
          report2[i,] <- report[report.index2[i],]
       }
       report3 <- report
       report.index3 <- order(Obs.ES.norm, decreasing=F)
       for (i in 1:Ng) {
          report3[i,] <- report[report.index3[i],]
       }
       phen1.rows <- length(Obs.ES.norm[Obs.ES.norm >= 0])
       phen2.rows <- length(Obs.ES.norm[Obs.ES.norm < 0])
       report.phen1 <- report2[1:phen1.rows,]
       report.phen2 <- report3[1:phen2.rows,]

#if (output.directory != "")  {
#     if (phen1.rows > 0) {
#       filename <- paste(output.directory, doc.string,
       ".SUMMARY.RESULTS.REPORT.", phen1,".txt", sep="", collapse="")
#      write.table(report.phen1, file = filename, quote=F, row.names=F, sep =
       "\t")
#   }
#  if (phen2.rows > 0) {
#         filename <- paste(output.directory, doc.string,
       ".SUMMARY.RESULTS.REPORT.", phen2,".txt", sep="", collapse="")
#         write.table(report.phen2, file = filename, quote=F, row.names=F, sep
       = "\t")
#      }
#}
```

```
#------------------------------------------------------------------------
# Produce report for each gene set passing the nominal, FWER or FDR test or
      the top topgs in each side

     if (topgs > floor(Ng/2)) {
        topgs <- floor(Ng/2)
     }

     result.geneset=vector(length=2*topgs,mode="character")
     k=1

     for (i in 1:Ng) {
        if ((p.vals[i, 1] <= nom.p.val.threshold) ||
            (p.vals[i, 2] <= fwer.p.val.threshold) ||
            (FDR.mean.sorted[i] <= fdr.q.val.threshold) ||
            (is.element(i, c(Obs.ES.index[1:topgs], Obs.ES.index[(Ng - topgs +
      1): Ng])))) {
                result.geneset[k]=gs.names[i]
                  k=k+1
       } # if p.vals thres

     } # loop over gene sets

    result=sum(!is.na(match(nrgs,result.geneset)))

    return(result)

}  # end of definition of GSEA.analysis
```

APPENDIX D: R code for power study

```
GSEA.program.location <- "e:/Wei/Wei study/Fall 2008/thesis/my code/my test
GSEA6-try.R"  #  R source program (change pathname to the rigth location in
local machine)
source(GSEA.program.location, verbose=T, max.deparse.length=9999)

dataset <- GSEA.Gct2Frame2(filename = "e:/Wei/Wei study/Fall 2008/thesis/GSEA-
P-R.1.0/Datasets/Gender.gct")
gene.labels <- row.names(dataset)
sample.names <- names(dataset)
A <- data.matrix(dataset)
dim(A)
cols <- length(A[1,])
rows <- length(A[,1])

names <- row.names(dataset)

output.a=matrix(nrow=20, ncol=1)
output.b=matrix(nrow=20, ncol=1)
output.c=matrix(names[1:1000],nrow=20, ncol=50, byrow=T)

for (i in 1:20) {
      output.a[i]=paste("L50_", i, sep="")
      output.b[i]="Sequence"
}

output=cbind(output.a, output.b, output.c)

outputdir=paste("E:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
R.1.0/GeneSetDatabases/", "my20",".gmt", sep="")

write(t(output), file = outputdir, sep="\t",ncolumns=52)


smtimes=1000

nonrandom=vector(length=smtimes, mode="numeric")

for (i in 1:smtimes)  {

print(c("Number of simulation:", i))

nonrandom[i]=GSEA(                                                    #
Input/Output Files :-------------------------------------------
 input.ds =  "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
R.1.0/Datasets/Gender.gct",            # Input gene expression Affy dataset
file in RES or GCT format
 input.cls = "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
R.1.0/Datasets/Gender.cls",            # Input class vector (phenotype) file
in CLS format
 gs.db =    "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
R.1.0/GeneSetDatabases/my20.gmt",         # Gene set database in GMT format
 output.directory     = "e:/Wei/Wei study/Fall 2008/thesis/GSEA-P-
R.1.0/Gender_C1/",          # Directory where to store output and results
(default: "")
#  Program parameters :-----------------------------------------------------
-----------------------------------------------------------------
 doc.string          = "try_C1",    # Documentation string used as a prefix
to name result files (default: "GSEA.analysis")
```

```
 non.interactive.run   = F,             # Run in interactive (i.e. R GUI) or
batch (R command line) mode (default: F)
 reshuffling.type     = "sample.labels", # Type of permutation reshuffling:
"sample.labels" or "gene.labels" (default: "sample.labels"
 nperm              = 1000,          # Number of random permutations
(default: 1000)
 weighted.score.type   =  1,            # Enrichment correlation-based
weighting: 0=no weight (KS), 1= weigthed, 2 = over-weigthed (default: 1)
 nom.p.val.threshold   = -1,            # Significance threshold for nominal
p-vals for gene sets (default: -1, no thres)
 fwer.p.val.threshold  = -1,            # Significance threshold for FWER p-
vals for gene sets (default: -1, no thres)
 fdr.q.val.threshold   = 0.25,          # Significance threshold for FDR q-
vals for gene sets (default: 0.25)
 topgs              = 1,           # Besides those passing test, number of
top scoring gene sets used for detailed reports (default: 10)
 adjust.FDR.q.val     = F,             # Adjust the FDR q-vals (default: F)
 gs.size.threshold.min = 5,            # Minimum size (in genes) for database
gene sets to be considered (default: 25)
 gs.size.threshold.max = 500,           # Maximum size (in genes) for database
gene sets to be considered (default: 500)
 reverse.sign        = F,             # Reverse direction of gene list (pos.
enrichment becomes negative, etc.) (default: F)
 preproc.type        = 0,             # Preproc.normalization: 0=none,
1=col(z-score)., 2=col(rank) and row(z-score)., 3=col(rank). (def: 0)
 random.seed         = 111,           # Random number generator seed.
(default: 123456)
 perm.type           = 0,             # For experts only. Permutation type: 0
= unbalanced, 1 = balanced (default: 0)
 fraction            = 1.0,           # For experts only. Subsampling
fraction. Set to 1.0 (no resampling) (default: 1.0)
 replace             = F,             # For experts only, Resampling mode
(replacement or not replacement) (default: F)
 save.intermediate.results = F,         # For experts only, save intermediate
results (e.g. matrix of random perm. scores) (default: F)
 OLD.GSEA            = F,             # Use original (old) version of GSEA
(default: F)
 use.fast.enrichment.routine = T        # Use faster routine to compute
enrichment for random permutations (default: T)
)
}

power=sum(nonrandom)/smtimes

power
```