Pathway Fingerprinting - Altschuler et al - Leukemia Stem Pathways

Gabriel Altschuler

May 31, 2013

Contents

This Sweave document contains the code used to produce the comparison of stem cell pathways in human and mouse leukemia, and the associated survival analysis contained within Altschuler et al.

1 Settings and Libraries

```
> # define environment settings and load packages and scripts
> options(width=60, stringsAsFactors = FALSE)
> options(continue=" ")
> tpbEnv <- new.env()
> #assign("cat", function(...) cat(file=stderr(),...), tpbEnv)
> environment(txtProgressBar) <- tpbEnv
> library(pathprint)
> library(GEOmetadb)
> source("supplementaryFunctions.R")
```

2 Data

2.1 Human leukemia stem cells

Gentles et al (Majeti) JAMA 2010 provides data on leukemic and normal stem and differentiated cell types. The Gentles data is not carried by the pathprint package as it was uploaded to Gene Expression Omnibus (GEO) using a custom CDF. However, the raw cell files can be obtained by ftp from GEO, normalized to produce an expression matrix, and fingerprinted. Corresponding meta data can be retrieved using the package GEOmetadb.

```
> # Load and fingerprint Gentles et al data
> # This has been done already in advance
> # library(hgu133plus2.db)
> # library(simpleaffy)
> # GSE24006<-read.affy(covdesc = "GSE24006.covdesc", path = "data")
> # GSE24006.eset<-call.exprs(GSE24006.data, "rma")
> # GSE24006.expression<-exprs(GSE24006.eset)
> # save(GSE24006.expression, file = "GSE24006.expression.RData")
> # Load locally saved version of this file
> load("data/GSE24006.expression.RData")
> Gentles.fingerprint <- exprs2fingerprint(exprs = GSE24006.expression,
                                           platform = "GPL570",
                                           species = "human",
                                           progressBar = FALSE)
> colnames(Gentles.fingerprint)<-gsub(".CEL", "", colnames(Gentles.fingerprint))
> rm(GSE24006.expression)


> # Retrieve metadata using GEOmetaDB
> # download database if required
> # getSQLiteFile()
> # adjust path as necessary
> con <- dbConnect(
   SQLite(),
   "../../../../../../Databases/GEOmetadb/GEOmetadb.sqlite")
> Gentles.sql<-paste("SELECT DISTINCT gsm.gsm,",
                     "gsm.title,",
                     "gsm.source_name_ch1,",
                     "gsm.characteristics_ch1",
                     "FROM",
                     "gsm JOIN gse_gsm ON gsm.gsm=gse_gsm.gsm",
                     "JOIN gse ON gse_gsm.gse=gse.gse",
                     "JOIN gse_gpl ON gse_gpl.gse=gse.gse",
                    "JOIN gpl ON gse_gpl.gpl=gpl.gpl",
                    "WHERE",
                    "gse.gse = 'GSE24006'",
                     sep = " ")
> Gentles.meta <- dbGetQuery(con, Gentles.sql)
> dbDisconnect(con)
> Gentles.meta$origin <- sapply(Gentles.meta$characteristics_ch1,
       function(x){if(grepl("AML",x)){"AML"} else {"normal"}})
> Gentles.meta$fraction <- sapply(Gentles.meta$characteristics_ch1,
       function(x){substring(x,
     regexpr("cell type \\(purified cell subset\\): ", x) +
     attr(regexpr("cell type \\(purified cell subset\\): ", x),
          "match.length"),
                              1000)})
> # define cell types
> Gentles.meta$type = paste(Gentles.meta$origin,
                            Gentles.meta$fraction,
                            sep = "_")
> Gentles.meta$cellType<-sapply(Gentles.meta$title,
           function(x){unlist(strsplit(
             unlist(strsplit(x, split = "_"))[[2]],
                              split = "_"))[[1]]})
> Gentles.meta$cellType[Gentles.meta$cellType == "34neg"] <- "Blast"
> Gentles.meta$cellType[Gentles.meta$cellType == "34pos38pos"] <- "LPC"
> Gentles.cellTypes<-levels(as.factor(Gentles.meta$cellType))
> Gentles.meta$cellType2<-NA
> Gentles.meta$cellType2[grep("HSC", Gentles.meta$cellType)]<-"stem"
> Gentles.meta$cellType2[grep("MPP", Gentles.meta$cellType)]<-"stem"
> Gentles.meta$cellType2[grep("LSC", Gentles.meta$cellType)]<-"stem"
> Gentles.meta$cellType2[grep("MEP", Gentles.meta$cellType)]<-"non-stem"
> Gentles.meta$cellType2[grep("GMP", Gentles.meta$cellType)]<-"non-stem"
> Gentles.meta$cellType2[grep("CMP", Gentles.meta$cellType)]<-"non-stem"
```

2.2 Mouse leukemia stem cells

Krivtsov et al. Nature (2006) provides data on HSC, hematopoeitic progenitors (CMP, GMP, MEP), as well as a leukemic stem cell type, L-GMP. The fingerprinted data and metadata associated with this publication are contained within the Pathprint package

```
> Krivtsov.meta<-GEO.metadata.matrix[
    GEO.metadata.matrix$GSE %in% "GSE3722",]
> Krivtsov.data<-GEO.fingerprint.matrix[,Krivtsov.meta$GSM]
> # define cell types
> Krivtsov.meta$cellType<-sapply(Krivtsov.meta$Title,
           function(x){unlist(strsplit(
             unlist(strsplit(x, split = "\\)"))[[1]],
                              split = "\\("))[[2]]})
> Krivtsov.meta$cellType<-gsub(" enriched", "", Krivtsov.meta$cellType)
> Krivtsov.cellTypes<-levels(as.factor(Krivtsov.meta$cellType))
> Krivtsov.meta$cellType2<-NA
> Krivtsov.meta$cellType2[grep("HSC", Krivtsov.meta$cellType)]<-"stem"
> Krivtsov.meta$cellType2[grep("MEP", Krivtsov.meta$cellType)]<-"non-stem"
> Krivtsov.meta$cellType2[grep("GMP", Krivtsov.meta$cellType)]<-"non-stem"
> Krivtsov.meta$cellType2[grep("L-GMP", Krivtsov.meta$cellType)]<-"stem"
> Krivtsov.meta$cellType2[grep("CMP", Krivtsov.meta$cellType)]<-"non-stem"
```

3 Stem Cell Characteristic Pathways

We will define the set of stem cell characteristic pathways (SCCP) in human and mouse. The SCCP are the pathways that are shared between normal and leukemic stem cells. First, the consensus fingerprints are defined for each of the cell types.

```
> threshold = 0.4
> Gentles.consensus<-sapply(Gentles.cellTypes, function(x){
    consensusFingerprint(Gentles.fingerprint[,
        Gentles.meta$gsm[Gentles.meta$cellType == x]],
    threshold = threshold)
    })
> Krivtsov.consensus<-sapply(Krivtsov.cellTypes, function(x){
    consensusFingerprint(Krivtsov.data[,
        Krivtsov.meta$GSM[Krivtsov.meta$cellType == x]],
    threshold = threshold)
    })
```

```
> # Define SCCP as pathways that differ between stem and non-stem samples
> # Apply additional filter to ensure that the SCCPs do not include any
> # pathways with values of both 1 and -1 within the stem or non-stem groups
>
> # Human SCCP
> Gentles.SCCP <- diffPathways(
   Gentles.consensus[,c(2,3,4,6,7)], c(1,1,2,2,1), 0.75)
> Gentles.SCCP.valid <- intersect(Gentles.SCCP,
                                  rownames(Gentles.consensus)[
    (apply(Gentles.consensus[,c("HSC", "LSC")],1,max) -
     apply(Gentles.consensus[,c("HSC", "LSC")],1,min)
     ) < 2])
> Gentles.SCCP.valid <- intersect(Gentles.SCCP.valid,
                                   rownames(Gentles.consensus)[
    (apply(Gentles.consensus[,c("CMP", "GMP", "MEP")],1,max) -
     apply(Gentles.consensus[,c("CMP", "GMP", "MEP")],1,min)
     ) < 2])
> # Mouse SCCP
> Krivtsov.SCCP <- diffPathways(Krivtsov.consensus, c(1,1,2,2,1), 0.75)
> Krivtsov.SCCP.valid <- intersect(Krivtsov.SCCP,
                                 rownames(Krivtsov.consensus)[
    (apply(Krivtsov.consensus[,c("HSC", "L-GMP")],1,max) -
     apply(Krivtsov.consensus[,c("HSC", "L-GMP")],1,min)
     ) < 2])
> Krivtsov.SCCP.valid <- intersect(Krivtsov.SCCP.valid,
                                 rownames(Krivtsov.consensus)[
    (apply(Krivtsov.consensus[,c("CMP", "GMP", "MEP")],1,max) -
     apply(Krivtsov.consensus[,c("CMP", "GMP", "MEP")],1,min)
     ) < 2])
> # Common SCCP
> (commonSCCP <-intersect(Krivtsov.SCCP.valid, Gentles.SCCP.valid))
```

```
[1] "Translation Factors (Wikipathways)"
[2] "GPCRs, Class B Secretin-like (Wikipathways)"
[3] "{PLCG2,30} (Static Module)"
[4] "{RAN,17} (Static Module)"
```

4 Survival Analysis

4.1 Clinical Datasets

A survival analysis was conducted across 4 independent clinical datasets, corresponding to GEO IDs GSE12417, GSE10358, GSE14468, and GSE1159. The data and metadata for each of these datasets is contained within the pathprint package, with additional data from the publications used to match sample IDs where required.

4.1.1 GSE12417

```
> # GSE12417
> # Retrieve metadata from GEO metadata matrix
> GSE12417.meta<-GEO.metadata.matrix[
     GEO.metadata.matrix$GSE %in% "GSE12417",]
> GSE12417.meta$ID<-GSE12417.meta$GSM
> # remove HGU133B data
> GSE12417.meta<-GSE12417.meta[-grep("U133B", GSE12417.meta$Title),]
> # extract survival time and censoring data
> GSE12417.meta$survivalTime<-sapply(
   GSE12417.meta$Characteristics, function(x){
     unlist(strsplit(unlist(strsplit(x, "OS = "))[2], "days"))[1]
     })
> GSE12417.meta$survivalTime<-gsub(" ", "", GSE12417.meta$survivalTime)
> GSE12417.meta$survivalTime<-as.numeric(GSE12417.meta$survivalTime)
> GSE12417.meta$death<-sapply(
   GSE12417.meta$Characteristics, function(x){
     unlist(strsplit(x, ": "))[2]
     })
> GSE12417.meta$death<-as.numeric(GSE12417.meta$death)
```

4.1.2 GSE10358

```
> # GSE10358
> # Retrieve metadata from GEO metadata matrix
> GSE10358<-GEO.metadata.matrix[
     GEO.metadata.matrix$GSE %in% "GSE10358",]
> # combine with curated metadata to obtain sample reference IDs
> GSE10358.meta<-read.delim("data/GSE10358_annotation_short.txt",
                            stringsAsFactors = FALSE)
> GSE10358.meta$UPN<-gsub("-", "", GSE10358.meta$UPN)
> # which do not match?
> #sum(GSE10358.meta$UPN %in% GSE10358$Title)
> #GSE10358.meta$UPN[!(GSE10358.meta$UPN %in% GSE10358$Title)]
> #GSE10358$Title[!(GSE10358$Title %in% GSE10358.meta$UPN)]
> #GSE10358.meta[GSE10358.meta$UPN %in% GSE10358.meta$UPN[
> #  !(GSE10358.meta$UPN %in% GSE10358$Title)],]
> #GSE10358[GSE10358$Title %in% GSE10358$Title[
> #  !(GSE10358$Title %in% GSE10358.meta$UPN)],]
>
> # the non-matching samples are different patients
> GSE10358.meta$ID<-GSE10358$GSM[match(GSE10358.meta$UPN, GSE10358$Title)]
> colnames(GSE10358.meta)[12]<-"death"
> colnames(GSE10358.meta)[14]<-"survivalTime"
> # NAs are duplicated
> GSE10358.meta$ID[duplicated(GSE10358.meta$ID)]<-paste(
   NA, 1:sum(duplicated(GSE10358.meta$ID)), sep = ".")
```

4.1.3 GSE14468

```
> # GSE14468
> GSE14468.meta<-GEO.metadata.matrix[
     GEO.metadata.matrix$GSE %in% "GSE14468",]
> GSE14468.meta$REF <- sapply(GSE14468.meta$Title, function(x){
      unlist(strsplit(x," "))[[1]]
         })
> GSE14468.meta$Karyotype <- NA
> GSE14468.meta$Karyotype[1:461] <- sapply(
   GSE14468.meta$Characteristics[1:461], function(x){
      unlist(strsplit(
        unlist(strsplit(x,"karyotype: "))[[2]], ";\\tnpm1"))[[1]]
         }, USE.NAMES = F)
> GSE14468.meta$Karyotype <- as.character(GSE14468.meta$Karyotype)
> normalGSE14468.meta <- GSE14468.meta[GSE14468.meta$Karyotype == "NN",]
> normalGSE14468.meta <- normalGSE14468.meta[
   !is.na(normalGSE14468.meta$Karyotype),]
> # also import mapping data from external tables
> GSE14468.datatable <- read.csv("data/OS_EFS_GSE6891.csv",
                                 stringsAsFactors = F)
```

```
> # find matching records in the
> GSE14468.datatable$ID <- GSE14468.meta$GSM[
   match(GSE14468.datatable$volgnummer,
         GSE14468.meta$REF)]
> GSE14468.datatable$survivalTime <- GSE14468.datatable$os
> GSE14468.datatable$death <- as.numeric(GSE14468.datatable$osi == 'dead')
> # remove NAs
> GSE14468.datatable <- GSE14468.datatable[!is.na(GSE14468.datatable$ID),]
```

4.1.4 GSE1159

```
> # GSE1159
> GSE1159.meta<-GEO.metadata.matrix[
     GEO.metadata.matrix$GSE %in% "GSE1159",]
> GSE1159.meta$REF <- sapply(GSE1159.meta$Title, function(x){
      unlist(strsplit(x," "))[[2]]
         })
> # also import mapping data from external table
> GSE1159.datatable <- read.csv("data/OS_EFS_GSE1159.csv", stringsAsFactors = F)
> GSE1159.datatable$ID <- GSE1159.meta$GSM[
   match(GSE1159.datatable$volgnummer,
         GSE1159.meta$REF)]
> GSE1159.datatable$survivalTime <- GSE1159.datatable$os
> GSE1159.datatable$death <- as.numeric(GSE1159.datatable$osi == 'dead')
> GSE1159.datatable <- GSE1159.datatable[!is.na(GSE1159.datatable$ID),]
```

4.2 Clinical data - fingerprints

```
> # Retrieve fingerprint data from GEO fingerprint matrix
> GSE12417.fingerprints<-GEO.fingerprint.matrix[,GSE12417.meta$GSM]
> GSE10358.fingerprints <- GEO.fingerprint.matrix[,GSE10358.meta$ID[
   grep("GSM", GSE10358.meta$ID)]]
> GSE14468.fingerprints <- GEO.fingerprint.matrix[,GSE14468.datatable$ID]
> GSE1159.fingerprints <- GEO.fingerprint.matrix[,GSE1159.datatable$ID]
```

4.3 Kaplan-Meier Plots - SCCP pathways

The function survivalAnalysis, contained in the Appendix, is used to produce Kaplan-Meier plots. This script relies on the R package Survival. Patients are stratified by the sum of the fingerprint scores across the common human/mouse SCCP pathways. In the output plots, blue represents the low expression group and red the high expression group. In addition to the KM plot for the SCCP pathways, a p-values from 1000 randomly selected pathway sets (with the same number of member pathways as the SCCP set) are used to produce a background against which the SCCP p-value can be compared. A boxplot (or beanplot if the beanplot package is installed) is produced, where the box or bean represents the distrubution of (log) p-values from randomly selected pathways, a red dot represents the p-value for the specified pathway set, and a blue line represents a p-value of 0.05 (a value often used in studies to denote a 'significant result'). Comparing the p-value of the specified pathway set to the background distribution provides a more rigorous test of significance.

```
> # survival analysis script
> # In this case input exprs is the pathway fingerprint
> # Input genesets is list of Pathway sets (only 1 in this case)
> nPermutations = 1000
> GSE10358.survivalAnalysis<-survivalAnalysis(
   exprs = GSE10358.fingerprints,
   geneset = list(CommonSCCP = commonSCCP,
                  HumanSCCP = Gentles.SCCP.valid,
                  MouseSCCP = Krivtsov.SCCP.valid),
   clinicalData = GSE10358.meta,
   grouping = "kmeans",
   nClusters = 2,
   method = "raw",
   pdf = "output/GSE10358.pdf",
   randomBackground = TRUE,
   nSamples = nPermutations,
   plotMax = 5,
   what = c(0,1,1,0)
   )
> GSE12417.survivalAnalysis<-survivalAnalysis(
   exprs = GSE12417.fingerprints,
   geneset = list(CommonSCCP = commonSCCP,
                  HumanSCCP = Gentles.SCCP.valid,
                  MouseSCCP = Krivtsov.SCCP.valid),
   clinicalData = GSE12417.meta,
   grouping = "kmeans",
   nClusters = 2,
   method = "raw",
   pdf = "output/GSE12417.pdf",
   randomBackground = TRUE,
   nSamples = nPermutations,
   plotMax = 5,
   what = c(0,1,1,0)
   )
> GSE1159.survivalAnalysis<-survivalAnalysis(
   exprs = GSE1159.fingerprints,
   geneset = list(CommonSCCP = commonSCCP,
                  HumanSCCP = Gentles.SCCP.valid,
                  MouseSCCP = Krivtsov.SCCP.valid),
   clinicalData = GSE1159.datatable,
   grouping = "kmeans",
   nClusters = 2,
   method = "raw",
   pdf = "output/GSE1159.pdf",
   randomBackground = TRUE,
   nSamples = nPermutations,
   plotMax = 5,
   what = c(0,1,1,0)
   )
> GSE14468.survivalAnalysis<-survivalAnalysis(
   exprs = GSE14468.fingerprints,
   geneset = list(CommonSCCP = commonSCCP,
                  HumanSCCP = Gentles.SCCP.valid,
                  MouseSCCP = Krivtsov.SCCP.valid),
   clinicalData = GSE14468.datatable,
   grouping = "kmeans",
   nClusters = 2,
   method = "raw",
   pdf = "output/GSE14468.pdf",
   randomBackground = TRUE,
   nSamples = nPermutations,
   plotMax = 5,
   what = c(0,1,1,0)
   )
```

The permutation analysis can also be run for the common human/mouse SCCP against a background of pathways randomly selected from the set of human SCCP.

```
> GSE10358.survivalAnalysis<-survivalAnalysis(
    exprs = GSE10358.fingerprints[Gentles.SCCP.valid, ],
    geneset = list(CommonSCCP = commonSCCP),
    clinicalData = GSE10358.meta,
    grouping = "kmeans",
    nClusters = 2,
    method = "raw",
    pdf = "output/humanSCCPBackgroundGSE10358.pdf",
    randomBackground = TRUE,
    nSamples = nPermutations,
    plotMax = 5,
    what = c(0,1,1,0)
    )
> GSE12417.survivalAnalysis<-survivalAnalysis(
    exprs = GSE12417.fingerprints[Gentles.SCCP.valid, ],
    geneset = list(CommonSCCP = commonSCCP),
    clinicalData = GSE12417.meta,
    grouping = "kmeans",
    nClusters = 2,
    method = "raw",
    pdf = "output/humanSCCPBackgroundGSE12417.pdf",
    randomBackground = TRUE,
    nSamples = nPermutations,
    plotMax = 5,
    what = c(0,1,1,0)
    )
 > GSE1159.survivalAnalysis<-survivalAnalysis(
    exprs = GSE1159.fingerprints[Gentles.SCCP.valid, ],
    geneset = list(CommonSCCP = commonSCCP),
    clinicalData = GSE1159.datatable,
    grouping = "kmeans",
    nClusters = 2,
    method = "raw",
    pdf = "output/humanSCCPBackgroundGSE1159.pdf",
    randomBackground = TRUE,
    nSamples = nPermutations,
    plotMax = 5,
    what = c(0,1,1,0)
    )
 > GSE14468.survivalAnalysis<-survivalAnalysis(
    exprs = GSE14468.fingerprints[Gentles.SCCP.valid, ],
    geneset = list(CommonSCCP = commonSCCP),
    clinicalData = GSE14468.datatable,
    grouping = "kmeans",
    nClusters = 2,
    method = "raw",
    pdf = "output/humanSCCPBackgroundGSE14468.pdf",
    randomBackground = TRUE,
    nSamples = nPermutations,
    plotMax = 5,
    what = c(0,1,1,0)
    )
```

5 Kaplan-Meier analysis - Individual Pathways

The clinical significance of individual SCCP can also be assessed by stratifying patients according to single pathway scores. In the paper, the PGLC2 module is used as an example. In the output plots red represents high expression, 1, yellow represents intermediate, 0, and blue low expression, -1.

```
> # In this case, the pathways are entered as a vector rather than a list
> # The script treats each separately instead of as a group
> names(commonSCCP) = commonSCCP
> GSE10358.survivalAnalysis<-survivalAnalysis(
   exprs = GSE10358.fingerprints,
   geneset = commonSCCP,
   clinicalData = GSE10358.meta,
   method = "ternary",
   pdf = "output/ternaryGSE10358.pdf",
   randomBackground = FALSE
   )
> GSE12417.survivalAnalysis<-survivalAnalysis(
   exprs = GSE12417.fingerprints,
   geneset = commonSCCP,
   clinicalData = GSE12417.meta,
   method = "ternary",
   pdf = "output/ternaryGSE12417.pdf",
   randomBackground = FALSE
   )
> GSE1159.survivalAnalysis<-survivalAnalysis(
   exprs = GSE1159.fingerprints,
   geneset = commonSCCP,
   clinicalData = GSE1159.datatable,
   method = "ternary",
   pdf = "output/ternaryGSE1159.pdf",
   randomBackground = FALSE
   )
> GSE14468.survivalAnalysis<-survivalAnalysis(
   exprs = GSE14468.fingerprints,
   geneset = commonSCCP,
   clinicalData = GSE14468.datatable,
   method = "ternary",
   pdf = "output/ternaryGSE14468.pdf",
   randomBackground = FALSE
   )
```

6 Appendix

Supplementary functions used in the analysis.

```
> survivalAnalysis<-function(exprs,
                             geneset,
                             genesetDown = vector("list", 0),
                             twoWay = FALSE,
                             clinicalData,
                             method = "zScore",
                             type = "KM",
                             grouping = "kmeans",
                             nClusters = 2,
                             quantile = 0.25,
                             pdf = "none",
                             tableFile = "none",
                             output = TRUE,
                             randomBackground = FALSE,
                             nSamples = 1000,
                             pValplotType = "bean",
                             plotMax = NULL,...)
 #########
 # function for survival analysis based on clinical data and an expression matrix
 # exprs - rownames are entrez IDs (unique), colnames are samples
 # geneset - list of gene sets to be tested
 #   if twoWay = TRUE, these are the up-regulated gene sets
 # genesetDown - if twoWay = TRUE, these are the list of down regulated gene sets
 # twoWay - defines whether statistic should account for up and down regulated genes
 # clinicalData - matrix with the following column names
 # ID - sample IDs matching the colnames of exprs
 # survivalTime - survival or follow-up time
 # death (optional) - status or censored, vector or 0 (alive) or 1 (deceased)
 # method - one of "zScore" (default), "medianZeroedZScore", "SCG"
 # zScore - scores each gene across all samples (inc those without clinical data)
 #        expression is evaluating by summing zScore
 # medianZeroed_zScore - as zScore but each sample column is first median zeroed
 # SCG - uses single chip GSEA to assess gene set expression scores # assume human data
 # raw - uses raw expression values - can be used for fingerprint analysis
 # ternary - divides into the ternary fingerprint score
 # type  - one of "KM" - default or "Cox"
 #                 "KM" computes an estimate of a survival curve using the Kaplan-Meier method
 #                 "Cox" computes predicted survivor function for a Cox proportional hazards model
 # grouping - one of "kmeans" or "quantile"
 #                "kmeans" groups the data into n groups, accoding to nClusters
 #                "quantile" defines groups as samples above an upper or below a lower quantile
 # nClusters - scores are clustered by kmeans into n groups, default = 2
 # quantile - upper and lower quantile for grouping samples, default = 0.25 # to be added
 # N.B. quantile splits sample into equal sized groups if = 0.5, split is at the median score
 # pdf (optional) - filename for graphs or "none" for quatrz or other default viewer (default)
 # tableFile (optional) - filename to output table of samples with their groupings
 # output (optional) - produce plots and print stats
 # randomBackground (optional) - for KM only, permute genesets to get p-value background
 # nSamples (optional) - number of permutations for background
 # pValplotType (optional) - use beanplot or boxplot for the pvalue distribution
 # plotMax (optional) - maximum x value for p-value plot, useful if want to standardize output
 # ... - additional comments to be passed onto bean or boxplot
 #########
 {
   library(survival) # load survival analysis package

   # check parameters
   if (!("ID" %in% colnames(clinicalData))) stop(
     "no ID column in clinical data")
   if (!("survivalTime" %in% colnames(clinicalData))) stop(
     "no survivalTime column in clinical data")
   if (!(method %in% c("SCG", "zScore", "medianZeroed_zScore", "raw", "ternary"))) stop (
     "please select one method from SCG, zScore, medianZeroed_zScore")
   if (twoWay == TRUE & !(length(geneset) == length(genesetDown))) stop(
     "Up and down regulated genesets must be of the same length")
   ID <- as.character(clinicalData$ID)
   exprsID<-as.character(colnames(exprs))
   if ((sum(duplicated(ID)) > 0 | sum(duplicated(exprsID)) > 0) == TRUE) stop
   ("duplicated sample names")
   matchedSamples<-intersect(ID, exprsID)
   if (length(matchedSamples) == 0) stop(
   "no IDs in exprs match the clinical data")
 if (output){print(paste(length(matchedSamples), "matching samples"))}
 if (!(class(nClusters) == "numeric")) stop("define number of clusters")
 if (quantile > 0.5) stop("define quantile between 0 and 0.5")
 # create data frame        for combining the clincal data and the expression matrix
 functionOutput <- ""
 dataframe<-as.data.frame(matrix(nrow = length(matchedSamples), ncol = (3+length(geneset))))
 colnames(dataframe)<-c("ID", "survivalTime", "death", names(geneset))
 rownames(dataframe)<-matchedSamples
 dataframe$ID<-matchedSamples
 dataframe$survivalTime<-clinicalData$survivalTime[match(matchedSamples, clinicalData$ID)]
 # optional to add death status for right censored data, otherwise assume all deceased
 if ("death" %in% colnames(clinicalData)){
   dataframe$death<-clinicalData$death[match(matchedSamples, clinicalData$ID)]
 }
 if (!("death" %in% colnames(clinicalData))){
   print("No censoring data present - assume all samples deceased")
   dataframe$death<-rep(1, nrow(dataframe))
 }
 if (method == "ternary"){
   grouping = ""; nClusters = 3} # skip grouping step and assign 3 clusters
 # score samples for each geneset according to the selected method
 if (method == "SCG") # run SCG, assume species is human
 {
   if(output){print("method = single chip geneset enrichment")}
   scores <- single.chip.GSEA(
     exprs = exprs,
     species = "human",
     gsdb = geneset,
     gene.set.selection = "ALL",
     sample.norm.type = "rank",
     output.score.type = "ES")
   if (twoWay == TRUE)
   {
     if(output){print("subtracting down regulated scores")}
     scores.down <- single.chip.GSEA(
       exprs = exprs,
       species = "human",
       gsdb = genesetDown,
       gene.set.selection = "ALL",
       sample.norm.type = "rank",
       output.score.type = "ES")
     rownames(scores.down)<-rownames(scores)
     scores<-scores-scores.down
   }
 }
 if (method == "medianZeroed_zScore")
   # zero expression levels about sample median
 {
   if(output){print("method = median zeroed z score")}
```

```
    rNames<-rownames(exprs)
    exprs<-apply(exprs, 2, function(x){x-median(x)})
    rownames(exprs)<-rNames
  }
  if ((method == "zScore") | (method == "medianZeroed_zScore"))
    # Z score across samples
    # score each geneset by the sum of the Z scores
  {
    if (!(method == "medianZeroed_zScore")){if(output){print("method = z score")}}
    # print(exprs) # for error checking
    zScore <- t(apply(exprs, 1, function(x){(x-mean(x))/sd(x)}))
    # print(head(zScore[,1:5])) # line inserted for error checking
    rownames(zScore)<-rownames(exprs)
    scores <- matrix(nrow = length(geneset), ncol = ncol(exprs))
    rownames(scores)<-names(geneset)
    colnames(scores)<-colnames(zScore)
    scores.down<-scores
    for (i in 1:length(geneset)){
      scores[i,]<-apply(
        zScore, 2, function(x){sum(x[rownames(zScore) %in% geneset[[i]]])})
    }
    if (twoWay == TRUE){
      if(output){print("subtracting down regulated scores")}
      for (i in 1:length(geneset)){
        scores.down[i,]<-apply(zScore, 2, function(x){
          sum(x[rownames(zScore) %in% genesetDown[[i]]])})
      }
      scores<-scores-scores.down
    }
  }
  if (method == "raw")
    # sum raw expression across samples
  {
    if(output){print("using raw expression values")}
    scores <- matrix(nrow = length(geneset), ncol = ncol(exprs))
    rownames(scores)<-names(geneset)
    colnames(scores)<-colnames(exprs)
    scores.down<-scores
    for (i in 1:length(geneset)){
      scores[i,]<-apply(exprs, 2, function(x){
        sum(x[rownames(exprs) %in% geneset[[i]]])})
    }
    if (twoWay == TRUE){
      if(output){print("subtracting down regulated scores")}
      for (i in 1:length(geneset)){
        scores.down[i,]<-apply(exprs, 2, function(x){
          sum(x[rownames(exprs) %in% genesetDown[[i]]])})
      }
      scores<-scores-scores.down
    }
```

```
}
# print(head(scores[,1:5])) # line inserted for error checking
if(output){print("")} # bug
# sometimes seems to need a line break to re-start text printing after SCG

if (grouping == "kmeans"){
  # cluster samples into groups using kmeans clustering
  if(output){print(paste("kmeans clustering samples into",
                         nClusters, "groups", sep = " "))}
  groups<-t(apply(scores, 1,
                  function(x){
 # print(quantile(x, (1:nClusters)/nClusters)) # for error checking
                    (kmeans(
                      unlist(x),
                      centers = quantile(x, (1:nClusters)/nClusters),
                      nstart = 5)
                    )$cluster
                  }
  ))

  # centers<-t(apply(scores, 1, function(x){
  #  quantile(x, (1:nClusters)/nClusters)})) # for error checking
  # print(groups) # line inserted for error checking
}
if (grouping == "quantile"){
  # define upper and lower boundaries according to defined quantile
  if(output){print(paste("Group samples as high (2) if >",
                         1-quantile, "quantile and low (1) if <",
                         quantile, "quantile", sep = " "))}

  # assign a score of 2 if > 1-quantile and 1 if < quantile
  groups<-t(apply(scores, 1,
                  function(x){
                    # print(quantile(x, c(quantile,1-quantile))) # for error checking
                    (2*(x > quantile(x, 1-quantile))+(x < quantile(x, quantile)))
  ))
  # set remaining samples to NA so they are omitted from survival plots
  groups[groups == 0]<-NA
  #   print("debugging")
  #                print(groups)
  #                print(class(groups))
  #                print(dim(groups))
}

if (method == "ternary"){
  # simply set groups corresponding to 2 + ternary score
  # so -1,0,1 becomes 1,2,3
  scores <- exprs[unlist(geneset),]
  #                groups<-as.matrix(t(2+scores))
  # edited on 31/5/12 to remove transpose and naming
  groups<-as.matrix(2+scores)
  #                print(groups)
  # rownames(groups)<-names(geneset)

  #                print(class(groups))
  #                print(dim(groups))
}


# combine groups with clinical data


colnames(dataframe) <- c(colnames(dataframe)[1:3], names(geneset))
for (i in 1:length(geneset)){
  #                print(table(groups[i,matchedSamples]))
  #                print(head(groups[i,matchedSamples]))
  #                print(class(matchedSamples))
  dataframe[,(3+i)]<-groups[i, matchedSamples]
}
#        print(head(dataframe)) # for error checking
#        print(table(dataframe[,4]))
# write table to file if selected
if (!(tableFile == "none")){
  write.table(dataframe, file = tableFile,
              quote = FALSE, row.names = FALSE, sep = "\t")
}

# produce survival graphs or compute proportional hazard model

if (type == "KM"){
  # create pdf if required
  pval.vector = vector("numeric", length(geneset))
  if(output){print("Plotting graphs")}
  if(output){pb <- txtProgressBar(min = 0, max = length(geneset), style = 3)}
  if (!(pdf == "none")){pdf(file = pdf)}
  for (i in 1:length(geneset)){
    # fit data to Kaplan-Meier survival curve
    #                        print(table(dataframe[,names(geneset)[i]]))
    fit<-survfit(Surv(dataframe$survivalTime, dataframe$death) ~
      dataframe[,names(geneset)[i]])
    diff<-survdiff(Surv(dataframe$survivalTime, dataframe$death) ~
      dataframe[,names(geneset)[i]])
    # print(diff) # inserted for error checking
    pval<-signif((1 - pchisq(diff$chisq, length(diff$n) - 1)), 3)
    col = colorRampPalette(c("blue", "green", "yellow", "red"),
                           nClusters)(nClusters)
    names<-vector("character", nClusters)
    # As centers were chosen incrementally this should result in
    # mean(grp1) < mean(grp2) < etc.
    for (j in 1:length(diff$n)){
      names[j]<-paste("Group", j, "n =", diff$n[[j]], sep  =" ")
    }
    # plot data if output flag is TRUE
    if(output == TRUE){
      plot(fit, col = col, xlab = "Survival time",
           main = paste(
             names(geneset)[i],
             length(intersect(geneset[[i]], rownames(exprs))),
             "genes",
             sep = " "
           )
      )
      if(min(fit$surv) > 0.2){xpos<-2; ypos <- 0.2}
      if(min(fit$surv) < 0.2){xpos<-0.5*max(fit$time); ypos <- 1}
      legend(xpos, ypos,        c(names, paste("p-value =", pval, sep = " ")),
             bty = "n", col = c(col, "white"), lty = 1)

      print(fit)}
    pval.vector[i] <- pval
    if(output){setTxtProgressBar(pb, i)}

  }
  if (!(pdf == "none")){dev.off()}
  functionOutput <- pval.vector
}


if (type == "Cox"){
  cox<-as.data.frame(matrix(nrow = length(geneset), ncol = 5))
  rownames(cox)<-names(geneset)
  colnames(cox)<-c("coef", "expCoef", "seCoef", "z", "p")
  for (i in 1:length(geneset)){
    fit<-coxph(Surv(dataframe$survivalTime, dataframe$death) ~
      dataframe[,names(geneset)[i]])
    cox$coef[i]<-fit$coefficients
    cox$expCoef[i]<-exp(fit$coefficients)
    cox$seCoef[i]<-(fit$var)^0.5
    # need to put in z and p-value
  }
  print(cox)
}

# create background of p-value built of random permutations
if (randomBackground == TRUE){
  if (type != "KM" | twoWay == TRUE) stop(
    "Random background only valid for one way, KM analysis")
  print("Running background p-value distribution")
  pval.matrix <- matrix(nrow = nSamples, ncol = length(geneset))
  colnames(pval.matrix) <- names(geneset)
  for (i in (1:nSamples)){
```

```
  # create a randomized geneset
  genesetRandom = lapply(geneset,
                        function(x){
                          sample(rownames(exprs),
                                 length(intersect(x,
                                                  rownames(exprs))))
                        })
  # run analysis
  # Error if cannot split into groups
  pval.output <- 1
  try(pval.output<-survivalAnalysis(
    exprs,
    genesetRandom, genesetDown, twoWay,
    clinicalData, method, type, grouping,
    nClusters, quantile, pdf = FALSE, tableFile,
    output = FALSE, randomBackground = FALSE))
  pval.matrix[i,]<-pval.output
}
# take logs
log.pval.matrix <- -log(pval.matrix, 10)
log.pval.vector <- -log(pval.vector, 10)
#    print(c(0:(1+length(pval.vector))))

# produce plot of p-value distribution overlaid onto values
if (!(pdf == "none")){
  pval.pdf <- paste(pdf, "pvalPlot.pdf", sep = ".")
  pval.pdf <- gsub(".pdf.", ".", pval.pdf)
  pdf(pval.pdf)}
# create default max unless specified
if (is.null(plotMax))
{
  plotMax = max(max(log.pval.matrix),max(log.pval.vector))
}
plotMin = 0
try(plot(log.pval.vector, c(1:length(pval.vector)),
     yaxt='n',
     ylim = c(0,(1+length(pval.vector))),
     xlim = c(plotMin, plotMax),
     pch = 19, col = "red",
     ylab = "", cex.axis = 0.5, cex.lab = 0.5,
     xlab = expression(-log[10]~"p-value")))
try(axis(2, labels = colnames(log.pval.matrix),
         at = 1:length(pval.vector), cex.axis = 0.5))
if(is.element("beanplot", installed.packages()[,1]) &
  pValplotType == "bean"){
  library(beanplot)
  try(beanplot(as.vector(as.matrix(log.pval.matrix)) ~
               factor(rep(colnames(log.pval.matrix),
                          each = nrow(log.pval.matrix)),
                      levels = colnames(log.pval.matrix)),
                   horizontal = TRUE, add = TRUE,
                   col = c("#606060FF", "#FFFFFFFF", "#000000FF", "#606060FF"),
                   cutmin = plotMin, method = "jitter", show.names = F,...))
    }
    else {
      print(
  "This might look nicer if the beanplot package was selected or installed")
      try(boxplot(as.vector(log.pval.matrix) ~
        factor(rep(colnames(log.pval.matrix),
                   each = nrow(log.pval.matrix)),
               levels = colnames(log.pval.matrix)),
                  horizontal = TRUE, add = TRUE,
                  col = "#FFFFFF00", names = F, axes = F,...))
    }
    # re-plot points as can get lost behind box/bean
    try(points(log.pval.vector, c(1:length(pval.vector)),
               pch = 19, col = "red"))
    abline(v = -log(0.05,10), col = "blue", lwd = 2)
    if (!(pdf == "none")){dev.off()}
    functionOutput <- pval.matrix
  }
  return(functionOutput)
}
```