cummeRbund: Visualization and Exploration of Cufflinks High-throughput Sequencing Data

Loyal A. Goff, Cole Trapnell

1 April, 2011

Contents

1 Requirements **2**

2 Introduction **2**

3 CummeRbund Classes **2**
3.1 CuffSet Class 2
3.2 CuffData Class 2
3.3 CuffFeatureSet Class 3
3.4 CuffFeature Class 4

4 Reading cuffdiff output **4**
4.1 Adding additional feature annotation 7

5 Global statistics **7**

6 Accessing Data **11**
6.1 Writing your own SQL accessors 13

7 Creating Gene Sets **15**
7.1 Geneset level plots 15

8 Individual Genes **21**
8.1 Gene-level plots 22

9 Data Exploration **25**
9.1 Finding similar genes 25

10 Miscellaneous **27**

11 Known Issues **28**

12 Session info **29**

1 Requirements

- Cufflinks $\geq$ v1.1.0 (Note: as of the release of this package, the current version of cufflinks is 1.3.3. While this package will work with >v1.1, we recommend updating your cufflinks prior to using cummeRbund)
- R $\geq$ v2.7.0
- Packages:
  - RSQLite
  - ggplot2
  - reshape
  - plyr

2 Introduction

cummeRbund is a visualization package for Cufflinks high-throughput sequencing data. The base class, *cuffSet* is a 'pointer' to cufflinks data that are stored out-of-memory in a sqlite database.

3 CummeRbund Classes

3.1 CuffSet Class

A pointer class to control access to the sqlite tables holding the Cufflinks data. The primary slot is DB which contains the RSQLite connection object. This can be accessed using the *DB()* accessor. The additional slots (genes, isoforms, TSS, and CDS) are each instances of the *CuffData* class and are pointers to sets of tables for each data subtype. They can be accessed with similar accessor wrappers. This is the default class created by *readCufflinks*. By default, *CuffData* accessor methods applied to a *CuffSet* class will operate on the 'genes' slot.

3.2 CuffData Class

The *CuffData* class is also a pointer class to the SQL backend, but each instance is specific for a data subtype (genes, isoforms, TSS, CDS). Again, there is an DB slot (accessible using *DB()*) that contains the RSQLite connection object. There are several accessor, setter, and plotting methods that allow for global analysis of all features within a *CuffData* class.Subsetting is currently being re-written, however, it is primarily done through the 'gene_id' field. Available slots for the CuffData class are:

- DB: RSQLite connection object

- tables: A *list* of tables in the SQLite DB that contain the cufflinks data.
- filters: A *list* of filters for subsetting (not implemented yet).
- type: A *character* field describing the data (ie. 'genes','isoforms','TSS','CDS','other')
- idField: The name of the identifying index field for this object (eg. 'gene_id' for type='gene', or 'isoform_id' for type='isoform')

Making the best use of either the CuffSet or CuffData classes will enable you to keep the entire dataset out of memory and significantly improve performance for large cufflinks datasets.

3.3 CuffFeatureSet Class

The *CuffFeatureSet* class is a data-storage container that holds all available data for a pre-determined list of features. Slots for FPKM data, differential regulation data, and feature-level annotation are all available. Unlike the previous classes, this class contains no connection information to the SQL database, but rather contains several slots with *data.frame* objects storing multiple-features worth of information. There are available accessors, and plotting methods that are designed to present multiple-features worth of information (eg. heatmaps, scatterplots, etc) Available slots for a *CuffFeatureSet* object include:

- annotation: Holds all feature-level annotation information for all features in object.
- fpkm: A data frame of FPKM data across all samples, for all features in object.
- diff: A data frame of differential expression/regulation data for all features in object.

A specialized sub-class of *CuffFeatureSet* is the *CuffGeneSet* class. This subclass adds additional slots to contain all isoforms, TSS, and CDS information for a given set of gene_ids. The *CuffGeneSet* class is designed to aggregate all relevant information for a set of genes into one object for easy analysis and/or manipulation. The *CuffGeneSet* object adds the following slots:

- ids: A 'character' list of all gene_ids used in object.
- isoforms: A *CuffFeatureSet* object for all isoforms of genes in object.
- TSS: A *CuffFeatureSet* object for all TSS of genes in object.
- CDS: A *CuffFeatureSet* object for all CDS of genes in object.

3.4 CuffFeature Class

The *CuffFeature* class is designed for single-feature-level data analysis and plotting. The methods available for this object are designed to analyze or visualize information about a specific feature. This is a 'data' object, as opposed to a 'pointer' object to the database backend. There is a validity requirement that a *CuffFeature* object only point to data from a single feature. Available slots for a *CuffFeature* object include:

- annotation: Holds feature-level annotation information for a given feature.
- fpkm: A data frame of FPKM data across all samples for a given feature.
- diff: A data frame of differential expression/regulation data for a given feature.

A specialized sub-class of *CuffFeature* is the *CuffGene* class. This subclass adds additional slots to contain all isoform, TSS, and CDS information for a given gene. The *CuffGene* object adds the following slots:

- id: The common 'gene_id' for all data in object
- isoforms: A *CuffFeature* object for all isoforms of a given gene.
- TSS: A *CuffFeature* object for all TSS of a given gene.
- CDS: A *CuffFeature* object for all CDS of a given gene.

Note: Future versions of cummeRbund may try to collapse the redundant functionality of the CuffFeature and CuffFeatureSet classes.

4 Reading cuffdiff output

cummeRbund was designed to process the multi-file output format for a 'cuffdiff' differential expression analysis. In this type of analysis, a user will use a reference .gtf file (either known annotation or a .gtf file created from a cufflinks assembly or merge of assemblies) and quantitate the expression values and differential regulation of the annotation(s) in the .gtf file across two or more SAM/BAM files. By design, cuffdiff produces a number of output files that contain test results for changes in expression at the level of transcripts, primary transcripts, and genes. It also tracks changes in the relative abundance of transcripts sharing a common transcription start site, and in the relative abundances of the primary transcripts of each gene. Tracking the former allows one to see changes in splicing, and the latter lets one see changes in relative promoter use within a gene.

Note: Cuffdiff requires that transcripts in the input GTF be annotated with certain attributes in order to look for changes in primary transcript expression,

splicing, coding output, and promoter use.

These attributes are:

- tss_id: The ID of this transcript's inferred start site. Determines which primary transcript this processed transcript is believed to come from. Cuffcompare appends this attribute to every transcript reported in the .combined.gtf file.
- p_id The ID of the coding sequence this transcript contains. This attribute is attached by Cuffcompare to the .combined.gtf records only when it is run with a reference annotation that include CDS records. Further, differential CDS analysis is only performed when all isoforms of a gene have p_id attributes, because neither Cufflinks nor Cuffcompare attempt to assign an open reading frame to transcripts.

cuffdiff calculates the FPKM of each transcript, primary transcript, and gene in each sample. Primary transcript and gene FPKMs are computed by summing the FPKMs of transcripts in each primary transcript group or gene group. The results are output in FPKM tracking files, the structure of which can be found in the cufflinks manual.

There are four FPKM tracking files:

- *isoforms.fpkm_tracking* Transcript FPKMs
- *genes.fpkm_tracking* Gene FPKMs. Tracks the summed FPKM of transcripts sharing each gene_id
- *cds.fpkm_tracking* Coding sequence FPKMs. Tracks the summed FPKM of transcripts sharing each p_id, independent of tss_id
- *tss_groups.fpkm_tracking* Primary transcript FPKMs. Tracks the summed FPKM of transcripts sharing each tss_id

cuffdiff also performs differential expression tests between supplied conditions. This tab delimited file lists the results of differential expression testing between samples for spliced transcripts, primary transcripts, genes, and coding sequences. For detailed file structure see cufflinks manual.

Four .diff files are created:

- *isoform_exp.diff* Transcript differential FPKM.
- *gene_exp.diff* Gene differential FPKM. Tests difference sin the summed FPKM of transcripts sharing each gene_id
- *tss_group_exp.diff* Primary transcript differential FPKM. Tests differences in the summed FPKM of transcripts sharing each tss_id

- *cds_exp.diff* Coding sequence differential FPKM. Tests differences in the summed FPKM of transcripts sharing each p_id independent of tss_id

In addition, cuffdiff also performs differential splicing, CDS usage, and promoter usage tests for each gene across conditions:

- *splicing.diff* Differential splicing tests.
- *CDS.diff* Differential coding output.
- *promoters.diff* Differential promoter use.

All of these output files are related to each other through their various tracking_ids, but parsing through individual files to query for important result information requires both a good deal of patience and a strong grasp of command-line text manipulation. Enter cummeRbund, an R solution to aggregate, organize, and help visualize this multi-layered dataset.
One of the principle benefits of using cummeRbund is that data are stored in a SQLite database. This allows for out-of-memory analysis of data, quick retrieval, and only a one-time cost to setup the tables. By default, cummeRbund assumes that all output files from cuffdiff are in the current working directory. To read these files, populate the 'cuffData.db' database backend, and return the *CuffSet* pointer object, you can do the following.

```
> library(cummeRbund)

> cuff <- readCufflinks(system.file("extdata", package="cummeRbund"))
> cuff

CuffSet instance with:
         3 samples
         400 genes
         1203 isoforms
         575 TSS
         545 CDS
         960 promoters
         1725 splicing
         696 relCDS
```

Again, by default *dir* is assumed to be the current working directory and `cuff<-readCufflinks()` should work if all appropriate files are in the current working directory. Should you need to rebuild the SQLite backend for any reason, you can add the option *rebuild=T* to *readCufflinks*. Once the database is created, *readCufflinks* will default to using the SQL backend and should not need to rebuild this database. Each R session should begin with a call to *readCufflinks* so as to initialize the database connection and create an object with the appropriate RSQLite connection information.

4.1 Adding additional feature annotation

Gene- or feature-level annotation can be permanently added to the database tables for future querying. If you have a data.frame where the first column contains the 'tracking_id' (eg. 'gene_id' for genes, 'isoform_id' for isoforms, etc). You can easily add feature level annotation using the `addFeatures()` function:

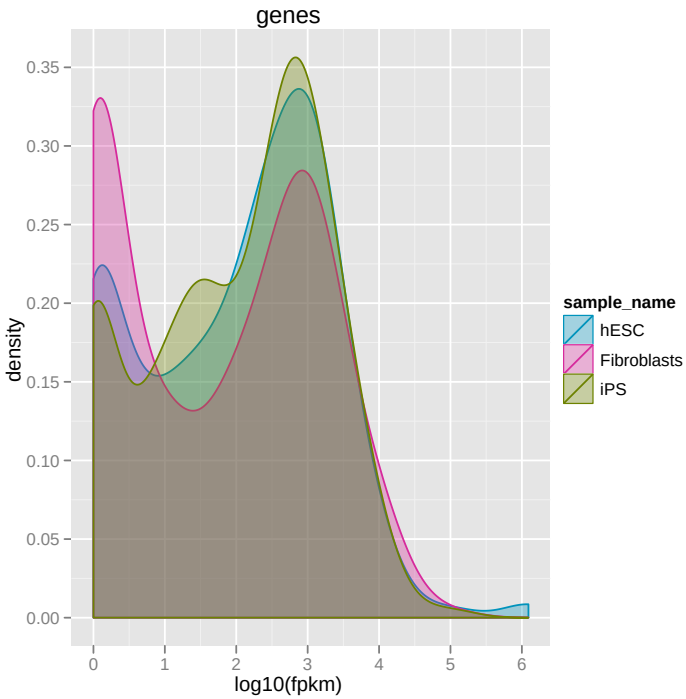
```
> #annot<-read.table("gene_annotation.tab",sep="\t",header=T,na.string="-")
> #addFeatures(cuff,annot,level="genes")
```

By default, features added to a *CuffSet* object are assumed to be gene-level annotations, but the level can selected using the argument *level*. Features added to a *CuffData* object are assumed to be of the same type as the 'type' value for that given object (e.g. gene-level features for 'genes', isoform-level features for isoforms, etc.)

5 Global statistics

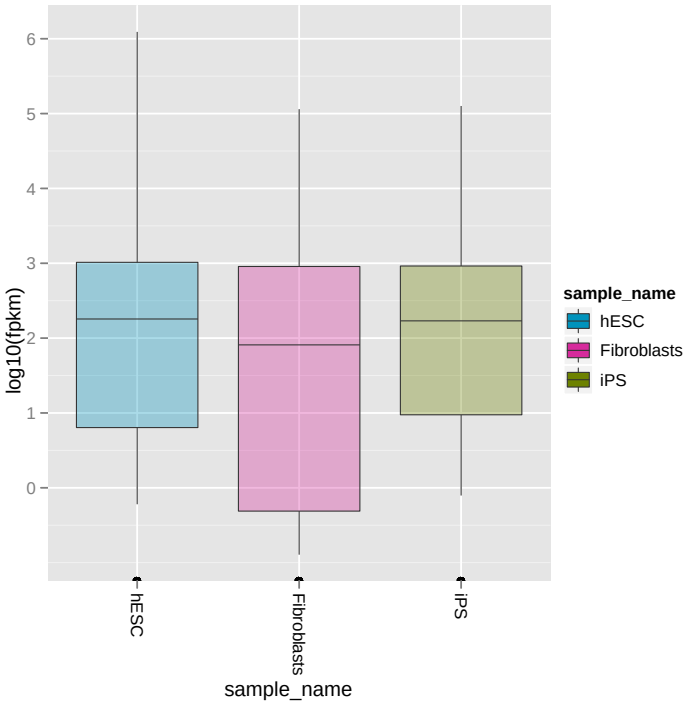
Several plotting methods are available that allow for quality-control or global analysis of cufflinks data. For example, to assess the distributions of FPKM scores across samples, you can use the *csDensity* plot (Figure 1).

```
> dens<-csDensity(genes(cuff))
> dens
```



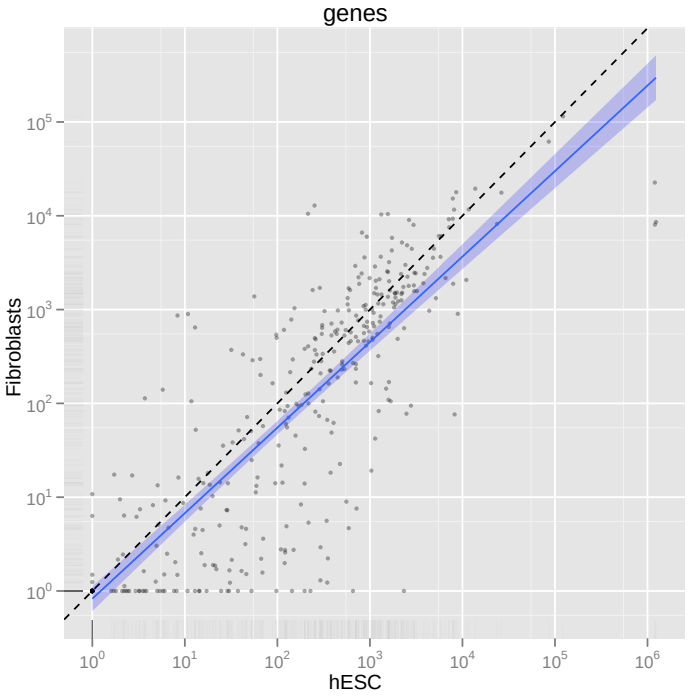
Boxplots can be visualized using the *csBoxplot* method (Figure 2).

```
> b<-csBoxplot(genes(cuff))
> b
```



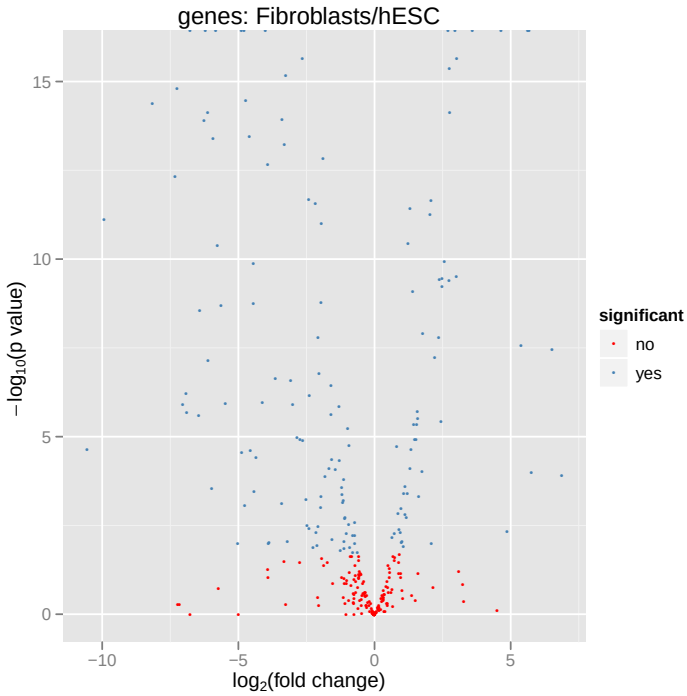
Pairwise comparisons can be made by using *csScatter*. You must specify the sample names to use for the x and y axes:

```
> s<-csScatter(genes(cuff),"hESC","Fibroblasts",smooth=T)
> s
```



Volcano plots are also available for the *CuffData* objects. Again, you must specify the comparisons by sample name.

```
> v<-csVolcano(genes(cuff),"hESC","Fibroblasts")
> v
```


genes: Fibroblasts/hESC
15
10
5
0
−10
−5
0
5
−log10(p value)
log2(fold change)
significant
no
yes

6 Accessing Data

Feature-level information can be accessed directly from a *CuffData* object using the *fpkm*, *diffData*, or *features* methods:

```
> gene.features<-features(genes(cuff))
> head(gene.features)

      gene_id class_code nearest_ref_id gene_short_name
1 XLOC_000001       <NA>           <NA>            <NA>
2 XLOC_000002       <NA>           <NA>           OR4F5
3 XLOC_000003       <NA>           <NA>            <NA>
4 XLOC_000004       <NA>           <NA>            <NA>
5 XLOC_000005       <NA>           <NA>            <NA>
6 XLOC_000006       <NA>           <NA>          OR4F16
               locus length coverage gene_id
1   chr1:11873-29961     NA       NA    <NA>
2   chr1:69090-70008     NA       NA    <NA>
3 chr1:321083-321114     NA       NA    <NA>
4 chr1:321145-321223     NA       NA    <NA>
5 chr1:322036-328580     NA       NA    <NA>
6 chr1:367658-368595     NA       NA    <NA>

> gene.fpkm<-fpkm(genes(cuff))
> head(gene.fpkm)

      gene_id sample_name      fpkm    conf_hi conf_lo
1 XLOC_000001 Fibroblasts 16.401100  428.14700       0
2 XLOC_000001        hESC  0.723836    3.01108       0
3 XLOC_000001         iPS 54.067200 1402.31000       0
4 XLOC_000002 Fibroblasts  0.000000    0.00000       0
5 XLOC_000002        hESC  0.000000    0.00000       0
6 XLOC_000002         iPS  0.000000    0.00000       0
  quant_status
1      LOWDATA
2           OK
3      LOWDATA
4           OK
5           OK
6           OK

> isoform.fpkm<-fpkm(isoforms(cuff))
> head(isoform.fpkm)

      isoform_id sample_name      fpkm  conf_hi  conf_lo
1 TCONS_00000001 Fibroblasts 11.910700 19.96650  3.85498
2 TCONS_00000001        hESC  0.000000  0.00000  0.00000
```

```
3 TCONS_00000001         iPS  9.563700 23.68410  0.00000
4 TCONS_00000002 Fibroblasts  0.000000  8.55378  0.00000
5 TCONS_00000002        hESC  0.723836  3.01108  0.00000
6 TCONS_00000002         iPS 32.934400 47.93760 17.93130
  quant_status
1           OK
2           OK
3      LOWDATA
4           OK
5           OK
6           OK
```

```
> gene.diff<-diffData(genes(cuff))
> head(gene.diff)
```

```
      gene_id sample_1    sample_2 status     value_1    value_2
1 XLOC_000001     hESC Fibroblasts     OK 7.23836e-01    16.4011
2 XLOC_000002     hESC Fibroblasts NOTEST 0.00000e+00     0.0000
3 XLOC_000003     hESC Fibroblasts NOTEST 0.00000e+00     0.0000
4 XLOC_000004     hESC Fibroblasts     OK 1.20000e+06 22616.4000
5 XLOC_000005     hESC Fibroblasts     OK 1.13903e+03    41.1644
6 XLOC_000006     hESC Fibroblasts NOTEST 0.00000e+00     0.0000
  ln_fold_change test_stat  p_value  q_value significant
1        4.50198 -0.246654 0.805176 0.893616          no
2        0.00000  0.000000 1.000000 1.000000          no
3        0.00000  0.000000 1.000000 1.000000          no
4       -5.72952  1.310270 0.190105 0.300329          no
5       -4.79027 10.857600 0.000000 0.000000         yes
6        0.00000  0.000000 1.000000 1.000000          no
```

Vectors of sample names and feature names are available by using the *samples* and *featureNames* methods:

```
> sample.names<-samples(genes(cuff))
> head(sample.names)
```

```
[1] "hESC"        "Fibroblasts" "iPS"
```

```
> gene.featurenames<-featureNames(genes(cuff))
> head(gene.featurenames)
```

```
[1] "XLOC_000001" "XLOC_000002" "XLOC_000003" "XLOC_000004"
[5] "XLOC_000005" "XLOC_000006"
```

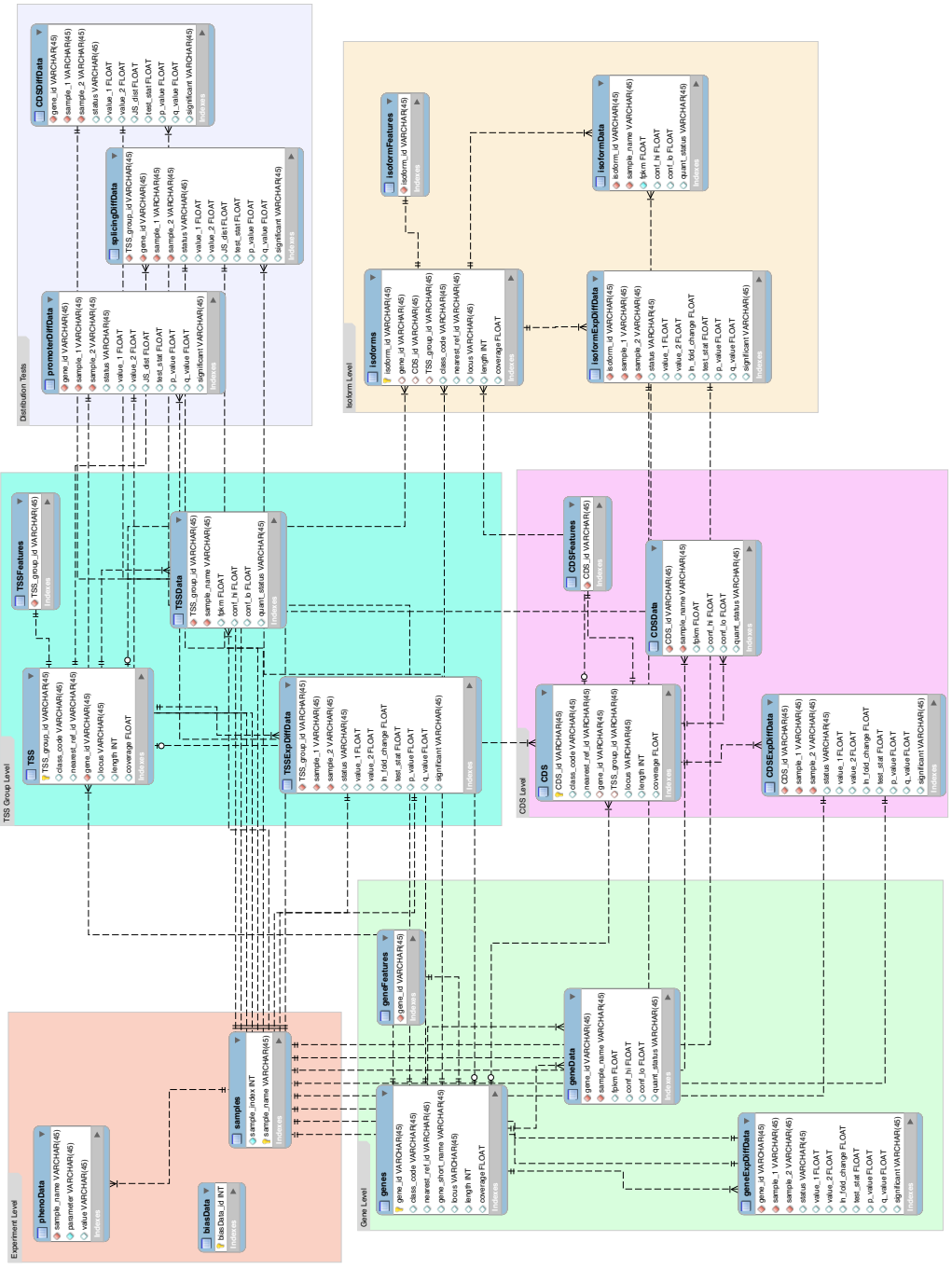
To facilitate Bioconductor-like operations, an 'FPKM-matrix' can be returned easily using the *fpkmMatrix* method:

```
> gene.matrix<-fpkmMatrix(genes(cuff))
> head(gene.matrix)
```

```
                   hESC Fibroblasts       iPS
XLOC_000001 7.23836e-01     16.4011  54.06720
XLOC_000002 0.00000e+00      0.0000   0.00000
XLOC_000003 0.00000e+00      0.0000   0.00000
XLOC_000004 1.20000e+06  22616.4000   0.00000
XLOC_000005 1.13903e+03     41.1644 944.30800
XLOC_000006 0.00000e+00      0.0000   9.00455
```

6.1 Writing your own SQL accessors

Since the cufflinks is a SQLite database backend, if you are familiar with SQL and/or RSQLite query construction, you can simply design your own SQL queries to access the data that you are after.



7 Creating Gene Sets

Gene Sets (stored in a *CuffGeneSet* object) can be created using the *getGenes* method on a CuffSet object. You must first create a vector of 'gene_id' or 'gene_short_name' values to identify the genes you wish to select:

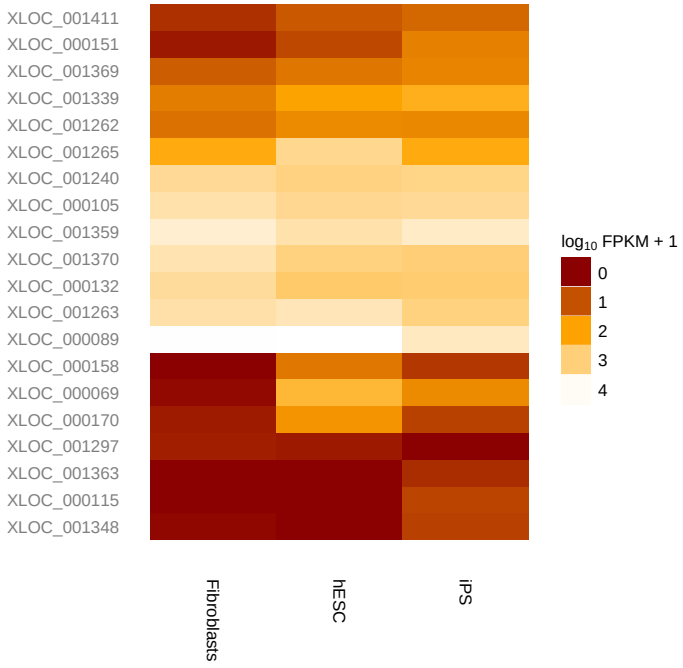
```
> data(sampleData)
> myGeneIds<-sampleIDs
> myGeneIds

 [1] "XLOC_001363" "XLOC_001297" "XLOC_001339" "XLOC_000132"
 [5] "XLOC_001265" "XLOC_000151" "XLOC_001359" "XLOC_000069"
 [9] "XLOC_000170" "XLOC_000105" "XLOC_001262" "XLOC_001348"
[13] "XLOC_001411" "XLOC_001369" "XLOC_000158" "XLOC_001370"
[17] "XLOC_001263" "XLOC_000115" "XLOC_000089" "XLOC_001240"

> myGenes<-getGenes(cuff,myGeneIds)
> myGenes

CuffGeneSet instance for genes c("XLOC_000069", "XLOC_000089", "XLOC_000105", "XLOC_000115",
Short name:         ESPN PGD MFN2 PRAMEF1 EFHD2 PADI1 NA FAM43B UBE2J2 C1orf86 SLC2A7 SPATA2
Slots:
         annotation
         fpkm
         diff
         isoforms         CuffFeatureSet instance of size 45
         TSS                 CuffFeatureSet instance of size 18
         CDS                 CuffFeatureSet instance of size 31
```

The same *fpkm*, *fpkmMatrix*, *features*, *diffData*, *samples*, and *featureNames* are available for instances of the *CuffGeneSet* class.

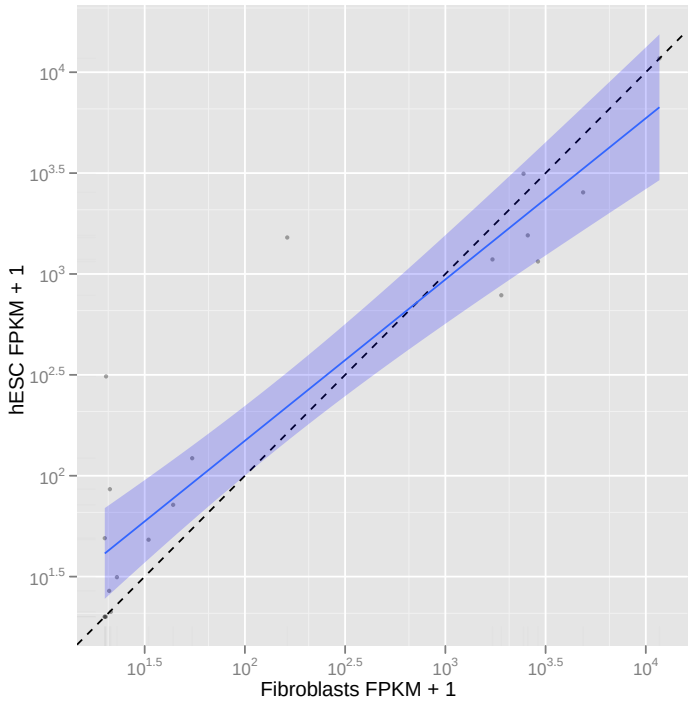
7.1 Geneset level plots

There are several plotting functions available for gene-set-level visualization:

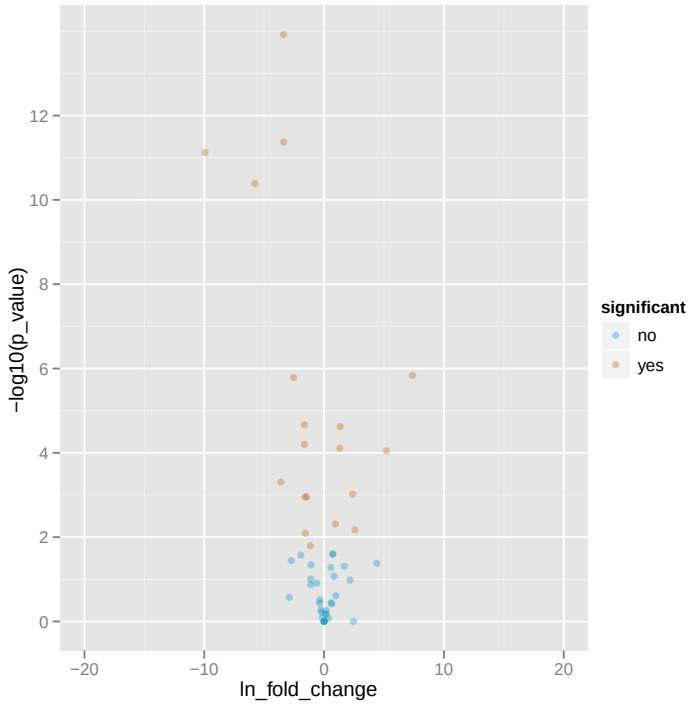
```
> h<-csHeatmap(myGenes,cluster='both')
> h
```




```
> s<-csScatter(myGenes,"Fibroblasts","hESC",smooth=T)
> s
```

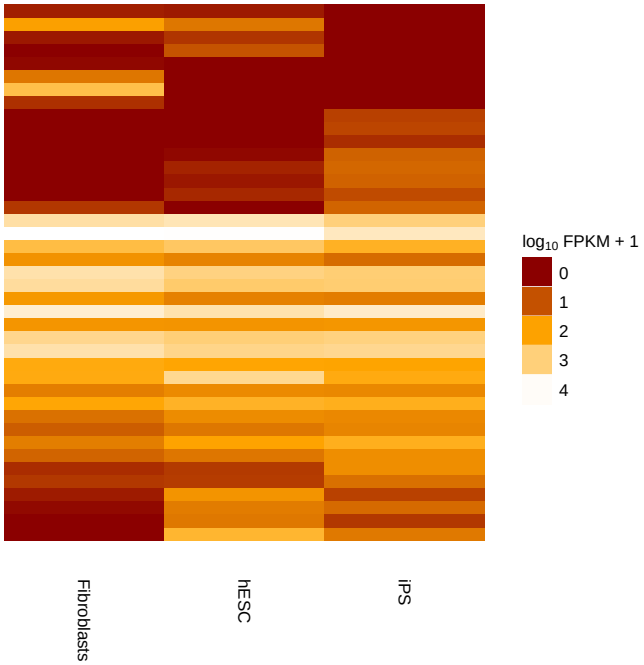



```
> v<-csVolcano(myGenes,cluster='both')
> v
```



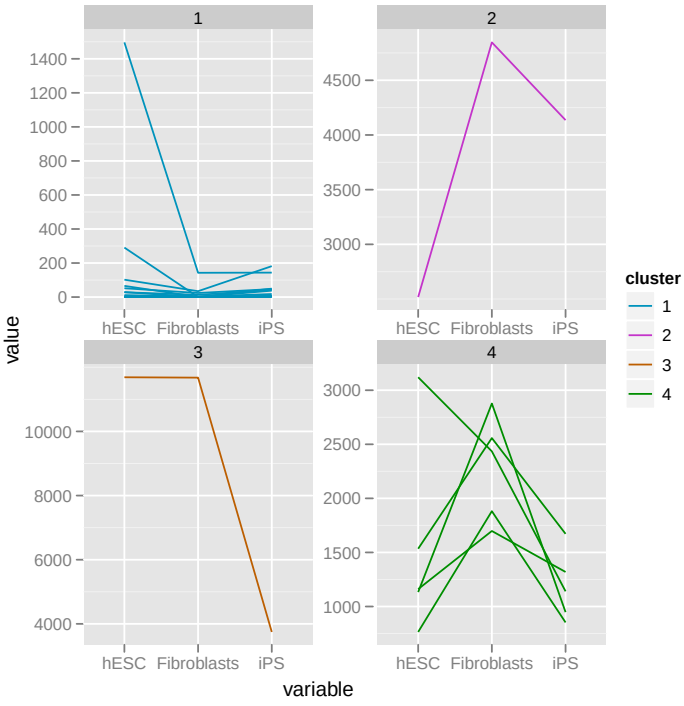
Similar plots can be made for all sub-level features of a *CuffGeneSet* class by specifying which slot you would like to plot (eg. *isoforms(myGenes)*, *TSS(myGenes)*, *CDS(myGenes)*).

```
> ih<-csHeatmap(isoforms(myGenes),cluster='both',labRow=F)
> ih
```



Rudimentary k-means clustering is implemented as well.

```
> ic<-csCluster(myGenes,k=4)
> ic
```


1
2
3
4
value
variable
hESC
Fibroblasts
iPS
cluster
1
2
3
4

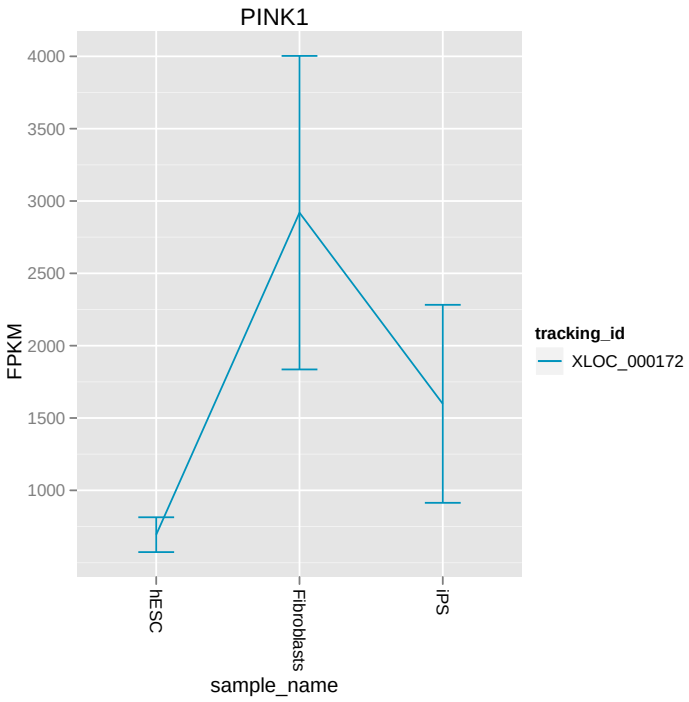
8 Individual Genes

An individual CuffGene object can be created by using the `getGene` function for a given 'gene_id'.

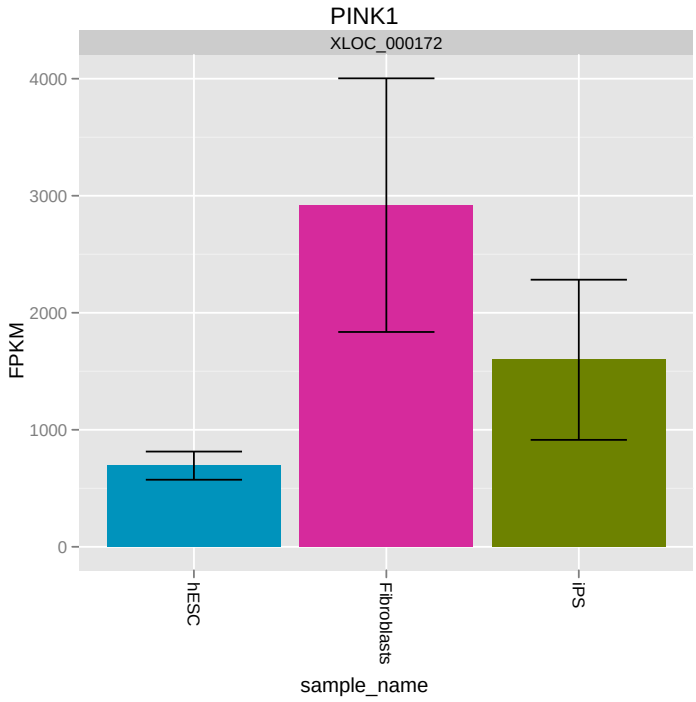
```
> myGeneId<-"PINK1"
> myGene<-getGene(cuff,myGeneId)
> myGene

CuffGene instance for gene PINK1
Short name:         PINK1
Slots:
         annotation
         fpkm
         diff
         isoforms         CuffFeature instance of size 2
         TSS                 CuffFeature instance of size 2
         CDS                 CuffFeature instance of size 2

> head(fpkm(myGene))

      gene_id sample_name     fpkm  conf_hi  conf_lo
1 XLOC_000172 Fibroblasts 2919.340 4002.960 1835.730
2 XLOC_000172        hESC  693.465  813.869  573.062
3 XLOC_000172         iPS 1598.040 2282.380  913.710
  quant_status
1           OK
2           OK
3           OK

> head(fpkm(isoforms(myGene)))

      isoform_id sample_name     fpkm  conf_hi   conf_lo
1 TCONS_00000480 Fibroblasts 2101.640 3111.330 1091.9400
2 TCONS_00000480        hESC  573.512  668.688  478.3370
3 TCONS_00000480         iPS 1598.040 2282.380  913.7100
4 TCONS_00000481 Fibroblasts  817.704 1391.700  243.7120
5 TCONS_00000481        hESC  119.953  152.675   87.2311
6 TCONS_00000481         iPS    0.000    0.000    0.0000
  quant_status
1           OK
2           OK
3           OK
4           OK
5           OK
6           OK
```

8.1 Gene-level plots

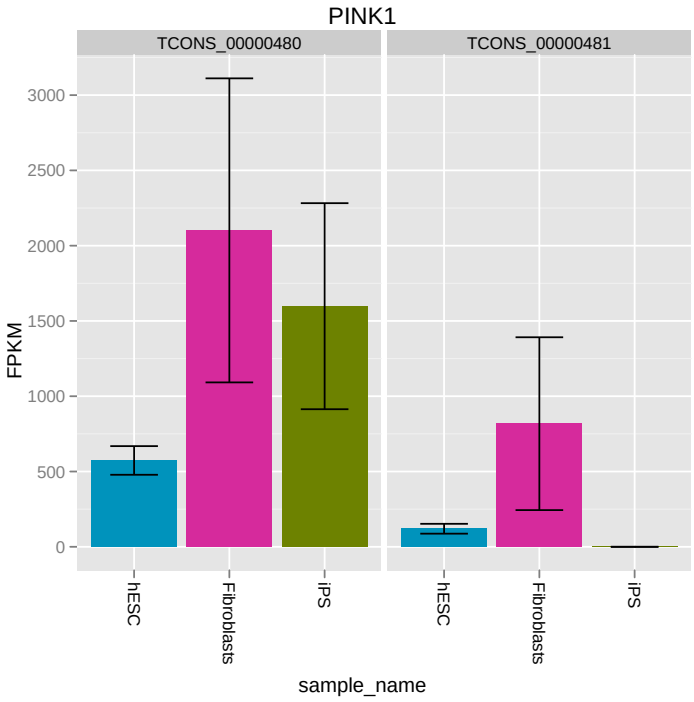
```
> gl<-expressionPlot(myGene)
> gl
```




```
> gb<-expressionBarplot(myGene)
> gb
```




```
> igb<-expressionBarplot(isoforms(myGene))
> igb
```


PINK1
TCONS_00000480
TCONS_00000481
FPKM
3000
2500
2000
1500
1000
500
0
hESC
Fibroblasts
iPS
hESC
Fibroblasts
iPS
sample_name

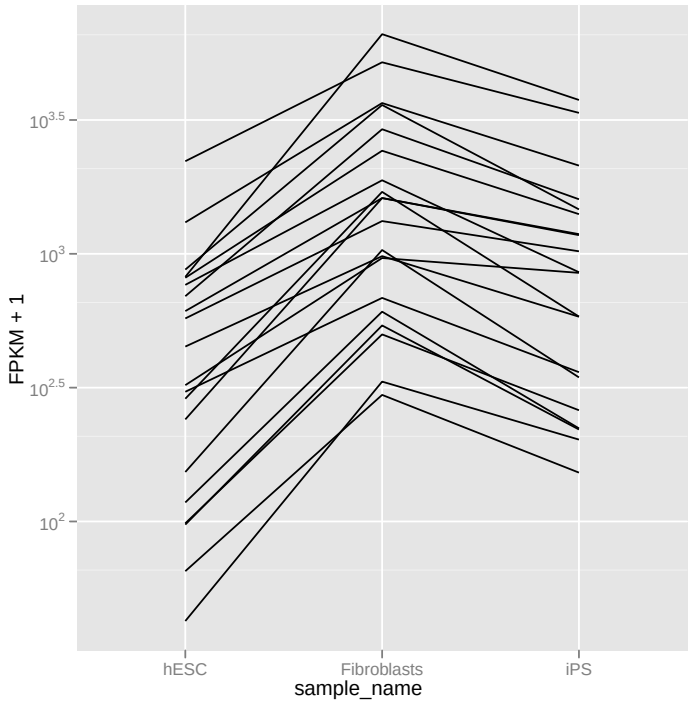
9 Data Exploration

The cummeRbund package is more than just a visualization tool as well. We are working to implement several different means of data exploration from gene and condition clustering, finding features with similar expression profiles, as well as incorporating Gene Ontology analysis.

9.1 Finding similar genes

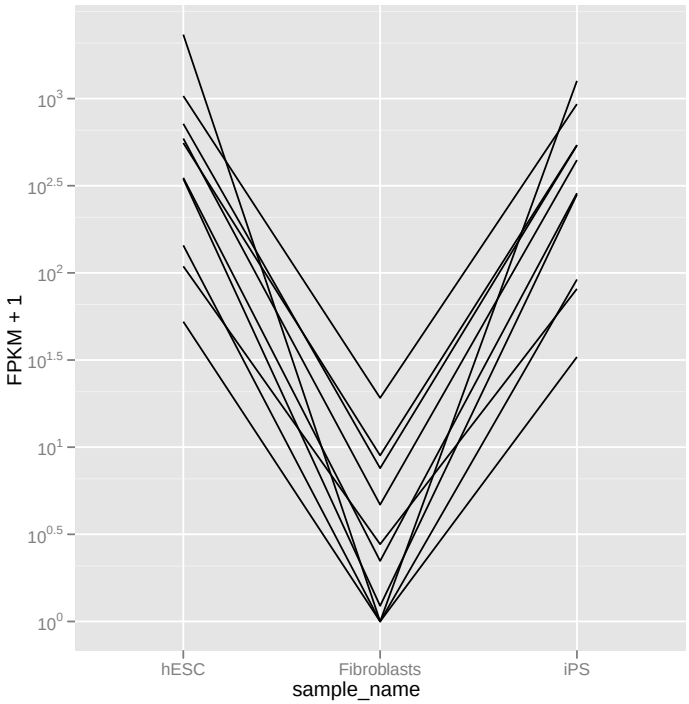
One common question in large-scale gene expression analyses is 'How can I find genes with similar expression profiles to gene x?'. We have implemented a method, *findSimilar* to allow you to identify a fixed number of the most similar genes to a given gene of interest. For example, if you wanted to find the 20 genes most similar to "PINK1", you could do the following:

```
> mySimilar<-findSimilar(cuff,"PINK1",n=20)
> mySimilar.expression<-expressionPlot(mySimilar,logMode=T,showErrorbars=F)
```



You are also able to provide your own expression profile in lieu of a 'gene_id'. The vector provided must match the order and length of *samples()*.

```
> myProfile<-c(500,0,400)
> mySimilar2<-findSimilar(cuff,myProfile,n=10)
> mySimilar2.expression<-expressionPlot(mySimilar2,logMode=T,showErrorbars=F)
```



findSimilar() uses the Jensen-Shannon distance between the probability distributions of each gene across conditions to determine the similarity. We have found this to be a more robust way to determine distance between genes using the high dynamic range of FPKM data. Future versions may allow for other dissimilarity measures to be used instead.

10 Miscellaneous

- All plotting functions return ggplot objects and the resulting objects can be manipulated/faceted/altered using standard ggplot2 methods.
- There are occasional DB connectivity issues that arise. Not entirely sure why yet. If necessary, just `readCufflinks` again and this should solve connectivity issues with a new RSQLite connection object. If connectivity continues to be a problem, try `cuff<-readCufflinks(rebuild=T)`
- I am still working on fully documenting each of the methods. There are a good number of arguments that exist, but might be hard to find without looking at the source.

11 Known Issues

- You must have at least one p_id field (see cufflinks manual) in your cuffdiff reference gtf file. Otherwise no results will be populated for the CDS.diff files and nothing will be available for cummeRbund to parse. This is described in more detail in the cuffdiff section of the cufflinks user guide.
- Large cuffdiff runs (e.g. $\geq$10 conditions) produce very large results files. These will take some time to parse and populate the cuffData.db sqlite database. While this is only a one time cost, the process can take a while. We are working on making the table writes and indexing significantly faster.
- Cuffdiff does not 'require' that gene_ids, isoform_ids, TSS_group_ids, or CDS_ids be unique in your reference gtf file. In fact, duplicate IDs will be aggregated by cummeRbund in the indexing phase and will produce undesireable effects. Please ensure that all of your IDs are unique prior to running cuffdiff (see cuffmerge for help) to avoid this issue.

12 Session info

```
> sessionInfo()

R version 2.14.0 (2011-10-31)
Platform: x86_64-unknown-linux-gnu (64-bit)

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

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets
[7] methods   base

other attached packages:
[1] cummeRbund_1.0.0 ggplot2_0.8.9    proto_0.3-9.2
[4] reshape_0.8.4    plyr_1.6         RSQLite_0.10.0
[7] DBI_0.2-5

loaded via a namespace (and not attached):
[1] digest_0.5.1 tools_2.14.0
```