All work in this vignette should be done in the same folder as created by the download of the GitHub Summer2017SURFResearch. This folder should have been named "../Summer2017SURFResearch" after completion of the GitHub repository download.

NOTE: the following software is required: python3 (version 3.6 was tested), Matlab (ver. 2016A was tested and Octave was found not to work), R (ver. 3.4.0 was tested), and XQuartz (required for SCDE). If the user encounters an error dealing with maximal number of DLLs reached, exit R and reload the packages to continue.

The following is a list of files included in the GitHub repository:

- AlteredDropOutGraph.Rmd
- CommonData.m
- CommonNames.m
- ESGeneNames.txt
- ESGeneNamesFull.txt
- MRI.m
- ReadsPer.m
- SmartSeq.txt
- SmartSeqFull.txt
- SS1.mat
- nameParse.py
- nameParse2.py
- vignette.pdf

1. Converting Gene Names (Optional)

If the gene ID is used instead of the name then this step is needed. Some datasets will have gene IDs instead of the gene names as row names. For future analysis it is imperative that all genes are standardized in this respect (either all gene names or all IDs). In this section we will be converting gene IDs into gene names.

As the user completing this at a later date will use different files, the following is a description of the current file names:

- "ESGeneNames.txt" - The row names (of gene IDS) in the original data matrix.
- "BothNames.txt" - A file containing both the gene ID and the corresponding name.
- "OldESGeneNames.txt" - A file containing gene IDs that the MGI database found corresponding gene names for
- "NewESGeneNames.txt" - A file containing corresponding gene names
- SS1 is the data matrix where the row names are the gene names found in the database

The following are detailed steps to convert gene IDs to gene names.

(1) Go to http://www.informatics.jax.org/batch

(2) Upload "ESGeneNames.txt" from the (downloaded) GitHub Summer2017SURFResearch repository directory, and click search. In this case "ESGeneNames.txt" is a test set (used to save time in this instructional setting). The full set has also been included in the repository. If one wants to run the full set simply use "ESGeneNamesFull.txt" instead. Note: this file will differ for each dataset and it is important that the user uses their own file.

(3) Download text from the JAX web site via 'Export: Text File' into the ../Summer2017SURFResearch folder, then rename the downloaded file to "BothNames.txt". The Python programs "nameParse.py" and "nameParse2.py" should be in the same ../Summer2017SURFResearch directory already.

(4) Open terminal and navigate to the ../Summer2017SURFResearch folder. Enter **python3 nameParse.py** <RET> A new file called NewESGeneNames will be created. Next: enter **python3 nameParse2.py** <RET> A new file called OldESGeneNames will be created. That is, output is the two files "NewESGeneNames.txt" and "OldESGeneNames.txt".

(5) Invoke Matlab at the command line. Within Matlab, navigate to the ../Summer2017SURFResearch folder. All the following commands for (5) through the end (8) of this "Converting Gene Names" section are to be invoked at the Matlab prompt. Enter the following to start :
  (a) **OldESGeneNames = importdata('OldESGeneNames.txt');** <RET>
  (b) **NewESGeneNames = importdata('NewESGeneNames.txt');**<RET>
  (c) **SmartSeqNames = importdata('ESGeneNames.txt');**<RET>
  (d) **SmartSeq = importdata('SmartSeq.txt');**<RET>
  Alternativly, if the importdata function is too slow and does not seem to work (which seemed to occur on octave) the function (which is included in the GitHub repository) TextScan can be used. Simply edit the file per the instructions included as comments on lines 1,7, and 11. Run the function by entering **TextScan** <RET> in the matlab command prompt. It will need to be run 4 times, one for each of the instructions in step 5.

  Note: If the user is using "ESGeneNamesFull.txt" then instead of loading in "SmartSeq.txt" the user should load in "SmartSeqFull.txt"

(6) Enter [**CommonNames indicesOldESGeneNames indicesSmartSeq**] = **intersect(OldESGeneNames, SmartSeqNames);**<RET>.

(7) Use the function CommonData. Type **SS1 = CommonData(SmartSeq, indicesSmartSeq);**<RET>. (This may take awhile.) Now SS1 is the new data matrix for the updated names. The next step is to update the names so they match with SS1.

(8) Run "CommonNames.m" by typing **CommonNames** <RET> at the Matlab prompt. Now SS1 is the new data matrix and commonNames is the gene list. We are done for this section.

2. Single Cell Differential Expression (SCDE) Analysis

The Paper for SCDE can be found at https://www.ncbi.nlm.nih.gov/pubmed/24836921

SCDE Preparation

Note that this entire section is done in R until the final section on multiple random imputations, which is done in Matlab.

(1) Download XQuartz from web site at https://www.xquartz.org.

(2) The following steps detail how to install and load SCDE and flexmix. We need to download SCDE version 1.99.4 and flexmix version 2.3-13. One problem is that we need to download flexmix before SCDE and that SCDE has the possibility of updating flexmix. In the following steps, SCDE and its dependencies are downloaded. Then SCDE is removed (leaving its dependencies). Then flexmix is downloaded, and finally SCDE is re-downloaded.

   (a) **install.packages("devtools")** <RET>
   (b) **require(devtools)** <RET>
   (c) **devtools::install_github('hms-dbmi/scde', build_vignettes = FALSE)** <RET>

   If an error dealing with a fortran library is encountered go to the SCDE website at http://hms-dbmi.github.io/scde/package.html and go to the troubleshooting section. It will tell you to enter the following two lines of code at the terminal:
      (i) **curl -O http://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2** <RET>
      (ii) **sudo tar fvxz gfortran-4.8.2-darwin13.tar.bz2 -C /** <RET>

   (d) **library(scde)** <RET>
   (e) **remove.packages("scde")**
   (f) **packageurl = "https://cran.r-project.org/src/contrib/Archive/flexmix/flexmix_2.3-13.tar.gz"** <RET>
   (g) **install.packages(packageurl, repos=NULL, type="source")** <RET>
   (h) **library(flexmix)**<RET>
   (i) **devtools::install_github('hms-dbmi/scde', build_vignettes = FALSE, dep = FALSE)** <RET>
   (j) **library(scde)** <RET>

   At this point the user should check the versions of SCDE and flexmix. Enter **packageVersion("scde")** <RET> and **packageVersion("flexmix")** <RET>. The output should be 1.99.4 and 2.3-13 respectively. As of August 2017 the most recent version of flexmix is 2.3-14, however this version is known to not work with the current version of SCDE.

(3) Download and load the R package R.matlab
   (a) **install.packages("R.matlab")** <RET>
   (b) **library(R.matlab)** <RET>

(4) Download and load the R package data.table
   (a) **install.packages("data.table")** <RET>
   (b) **library(data.table)** <RET>

Running SCDE
http://hms-dbmi.github.io/scde/diffexp.html is the vignette for SCDE and https://www.bioconductor.org/packages/devel/bioc/manuals/scde/man/scde.pdf is the package description. For this project the output desired for imputation is "p.self.fail". It is also important to note that setting the number of cores to greater than one may not work on a single machine, however this error does not seem to happen when SCDE is run an a cluster. The authors are still unsure of what causes this. If any other error comes up there is an active SCDE Google group where the authors frequently answer questions at

https://groups.google.com/forum/#!forum/singlecellstats

(1) Load in the raw read counts of the data. SCDE only accepts integer values as input so normalized data will not work. Enter **readMat("SS1.mat")** <RET> and **SS1 = SS1$SS1** <RET>

(2) Even if the data looks like integers that may not be the case. It may be necessary to run the following code: **SS1 = apply(SS1,2,function(x) {storage.mode(x) = 'integer'; x})**.

(3) It also may be useful to apply some quality control before running SCDE. Luckily the SCDE package includes a function to help with that. If QC is desired (which depends on the quality of the matrix) look up the clean.counts function in the package description included above. The included test data does not need any QC.

(4) If running the test data set skip this step and go directly to step (5). If running the full dataset (this should not be necessary for the test case) open the R terminal on the cluster. The following steps pertain to the Ohio Supercomputing Center (OSC) (it is important to note that this should be done either in an interactive session or in a job submission, not on the login nodes):
   (a) **module load gnu/5.2.0** <RET>
   (b) **module load R** <RET>
   (c) **R** <RET>

(5) The last step before running SCDE is to add row names and column names. Enter **rownames(SS1) = 1:50** <RET> and **colnames(SS1) = 1:130** <RET>. The numbers should be changed to reflect the size (number of rows and columns) of the dataset in use.

(6) The following code will take some time to run (a couple of hours if using a cluster with a full dataset with 12 nodes). Enter
o.ifm = scde.error.models(counts = SS1, n.cores = X, min.size.entries = Y, max.pairs = Z)<RET> . Where:
   (a) o.ifm is the negative binomial and Poisson process model that is the output of SCDE
   (b) n.cores is the number of cores used (if using a laptop I recommend 1 otherwise an error may pop up).
   (c) max.pairs is the maximum number of pairs used to create the model. The default is 5,000 however many different levels should be tried. I found success by setting it extremely high (i.e. 500,000) as then the model would incorporate every possible pairing.
   (d) min.size.entries is the minimum number of genes (rows) needed. (Make sure to set this to less than 50 if you are using the test set.)

(7) Now enter **valid.cells = o.ifm$corr.a > 0** <RET> and **o.ifm = o.ifm[valid.cells,]** <RET>

(8) Now to calculate the prior type **SS1p = scde.expression.prior(models = o.ifm, counts = SS1, length.out = 400, show.plot = FALSE)** <RET>

(9) To calculate the matrix where each entry is the probability of a dropout type in **SS1f = scde.failure.probability(models = o.ifm, counts = SS1)** <RET>

(10) Finally in preparation to produce the fail curves (probability of drop out on the y axis and gene magnitude expression on the x) enter **SS1fc = scde.failure.probability(o.ifm, magnitudes = log((10^SS1p$x)-1))** <RET>

Creating fail curves. If the previous steps were not done on the cluster skip to step 3.

(1) First we need to print out the prior and the fail curve matrix. Do this by typing in the following (and changing the path when needed):
   (a) **write.table(SS1p, "SS1Prior.txt", sep="\t")** <RET>
   (b) **write.table(SS1fc, "SS1FailCurves.txt", sep="\t")** <RET>

(2) Then we have to read them back in to R on the computer (as compared to cluster where we originally did the work). Navigate to ../Summer2017SURFResearch:
   (a) **SS1fc = fread("SS1failCurves.txt")** <RET>
   SS1fc = SS1fc[,-1] <RET>

   (b) **SS1p = fread("SS1prior.txt")** <RET>
   SS1p = SS1p[,-1] <RET>
   colnames(SS1p) = c("x", "y", "lp", "grid.weight") <RET>

(3) Now to plot the fail curves, type in
   par(mfrow = c(1,1), mar = c(3.5,3.5,0.5,0.5), mgp = c(2.0,0.65,0), cex = 1) <RET>

   plot(c(), c(), xlim=range(SS1p$x), ylim=c(0,1), xlab="expression magnitude (log10)", ylab="drop-out probability", main = "SS1 Dropout") <RET>

   invisible(apply(SS1fc, 2, function(y) lines(x = SS1p$x, y = y,col = "orange"))) <RET>

(4) Ideally the graph should overall have a negative slope. If this is not the case it may be best to graph a subset to see if that portion has a negative correlation. For this we will only graph curves where the first value is sufficiently high (i.e. non-zero). Simply run the r markdown document "AlteredDropOutGraph" by pressing the green arrow in the top right of both code blocks (start with the top one).

Multiple Random Imputations

It is important to note that for whatever matrix is chosen to impute with SCDE the non-imputed matrix is needed for this step. For instance we decided to impute the values of SS1 (created in the first section) and thus we need SS1 (which should already be a variable, if not follow the instructions from the first step) for this final step.

(1) The first step is to print out the fail matrix. This is done by entering **write.table(SS1f, "SS1Fail.txt", sep="\t")** <RET> in R

(2) Now open matlab and type **SS1f = importdata('SS1Fail.txt');**<RET>

(3) To do multiple random imputations use the function "MRI". Type in matlab **imputedSS1 = MRI(SS1,SS1f.data, X)** <RET> where X is the number of desired imputations (75 was used). (Note: make sure SS1 is also imported into Matlab, if not go to the first section). ImputedSS1 is the the final matrix.

3. Calculating Reads Per Million (RPM)

This step will normalize a data matrix using RPM

(1) In your Matlab session, enter **SS1RPM = ReadsPer(SS1,1000000)** <RET>. SS1RPM is now the RPM values of SS1 (where SS1 was created in the first section)

4. Calculating Center Log Ratio (CLR)

This section will normalize a data matrix using CLR

(1) CLR is calculated in R and therefore we need to move the data into R. In Matlab, enter **save('SS1.mat','SS1');** <RET>. Notice that this file was actually included in the GitHub repository and thus this step can be skipped if the user is using the test set.

(2) Open R. Change the directory to the ../Summer2017SURFResearch directory using setwd(). The following steps in this section will all be in R. Download and load the R package scone. This step may take some time too if the user has not previously used biocLite
   (a) **source("https://bioconductor.org/biocLite.R")** <RET>
   (b) **biocLite("scone")**<RET>
   (c) **libary(scone)** <RET>

(3) Enter **SS1 = readMat("SS1.mat")**<RET>

(4) Then type **SS1 = SS1$SS1** <RET>

(5) To normalize the data type: **SS1CLR = CLR_FN(SS1)** <RET>. (You can ignore the warning msgs that this command may output.)

(6) SS1CLR is now the normalized data. To move it back to Matlab type **write.table(SS1CLR, "SS1CLR.txt", quote = FALSE, sep = "\t")** <RET>. This file can then be imported into Matlab using the importdata function as done before.

5. Miscellaneous Functions

- BulkToSparse - Takes bulk data matrix and turns it into a sparse one by randomly removing entries. Enter cutoff as a whole number. A cutoff of 15 means a 15% sparse matrix (assuming the matrix is originally 0% sparse). Useful to see how a bulk matrix would act as a single-cell one.

- CleanZeros - Given data matrix, gene names for that matrix, and a cutoff. Deletes rows (genes) if they have a high enough percent zero. Percent should be given in decimal form. If percent is equal to 0.1 then all genes that are more than 10% zero are removed. Line 8 removes the first line in the genes which is a common problem in Matlab. If the input is simply the genelist comment it out.

- Count2Graph - Outputs different histograms. nbins is the number of bins forh istogram. If log is 1 then the logarithm of the data is plotted. If type is 1 plots the amount of reads (i.e. total mRNA found) per cell and if type is2 plots number of zeros per cell. Y axis will be number of cells and Xaxis will be either amount of reads (type 1) or number of zeros (type 2).

- createZeroGraphs - Plots percent zero against number of genes. Type "hold on" between function calls if you want to plot multiple graphs in the same window. Depends on RealPercentZero.

- GetTopGenes - Creates datamatrix with corresponding gene names of the top x expressed genes (i.e. genes with the most number of reads). Input is data matrix, gene list and the amount of top genes desired.

- PCAPrep - genesc is the row and column names matrix. If you only have the gene names and no cell names you actually dont need this function and can just set genes to be the gene list and ncount to be the data matrix. Preps data for PCA code. If included the PCA code will color based on percent zeros.