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), R (ver. 3.4.0 was tested), and XQuartz (required for SCDE)

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
- 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>

(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. Calculating RPM

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

3. Calculating Center Log Ratio (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>

(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 following packages using the install.packages() and library() methods: R.matlab and scone.

(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.

4. Single Cell Differential Expression (SCDE) Analysis

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 download and load packages in R

(a) SCDE 1.99.4

  (i) **require(devtools)** <RET>

  (ii) **devtools::install_github(’hms-dbmi/scde’, build_vignettes = FALSE)** <RET>

  (iii) **library(scde)** <RET>

(b) flexmix version 2.3-13 (note: a more updated version of flexmix will be downloaded with SCDE, however that version does not work with SCDE, so make sure you either download this version of flexmix after SCDE or that you do not update flexmix when downloading SCDE).

  (i) **require(devtools)** <RET>

  (ii) **packageurl = ”https://cran.r-project.org/src/contrib/Archive/flexmix/flexmix 13.tar.gz”** <RET>

  (iii) **install.packages(packageurl, repos=NULL, type=”source”)** <RET>

  (iv) **library(flexmix)**<RET>

(c) data.table

  (i) **install.packages(”data.table”)** <RET>

  (ii) **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, when changing the number of cores (n.cores) to a value greater than one the package sometimes does not work (it seems to be computer dependent however it did not seem to affect the 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. We have already loaded in the data matrix SS1 in the third section, which will be the matrix we use for this portion.

(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.

(4) (Optional but advised) Open the R terminal on the cluster. The following steps pertain to the 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 following code will take some time to run (hence the cluster use). Enter
o.ifm = scde.error.models(counts = SS1, n.cores = X, min.size.entries = Y, max.pairs = Z)<RET>

  (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.)

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

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

(8) 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>

(9) 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, ”../Summer2017SURFResearch/SS1Prior.txt”, sep=”\t”)** <RET>

(b) **write.table(SS1fc, ”../Summer2017SURFResearch/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, ”../Summer2017SURFResearch/SS1Fail.txt”, sep=”\t”)** <RET>
(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.