1. Converting Gene Names

(1) Go to http://www.informatics.jax.org/batch
(2) Upload "ESGeneNames.txt" (or any list of gene ID names)
(3) Download text, rename to "BothNames.txt" and place in the same folder as "nameParse.py"
(4) Open terminal and navigate to the above folder. Enter **python3 nameParse.py** and **python3 nameParse2.py**
(5) Output is "NewESGeneNames.txt" and "OldESGeneNames.txt"
(6) Go to matlab and enter the following:
    (a) **OldESGeneNames = importdata('OldESGeneNames.txt');**
    (b) **NewESGeneNames = importdata('NewESGeneNames.txt');**
    (c) **SmartSeqNames = importdata('ESGeneNames.txt');**
    (d) **SmartSeq = importdata('SmartSeq.txt');**
(7) Enter **[CommonNames indicesOldESGeneNames indicesSmartSeq] = intersect(OldESGeneNames, SmartSeqNames);**
(8) Use the function CommonData. Type **SS1 = CommonData(SmartSeq, indicesSmartSeq);**. Now SS1 is the new data matrix for the updated names. The next step is to update the names so they match with SS1.
(9) Run "CommonNames.m"
(10) Now SS1 is the new data matrix and commonNames is the gene list

2. Calculating RPM

(1) Navigate to the functions folder
(2) Enter **SS1RPM = ReadsPer(SS1,1000000)**
(3) SS1RPM is now the RPM values of SS1

3. Calculating CLR

(1) CLR is calculated in R and therefore we need to transfer the data. Enter **save('SS1.mat','SS1');**
(2) Open R and download the following packages: R.matlab and Scone
(3) Navigate to the folder where SS1.mat is saved. In R, enter **SS1 = readMat("SS1.mat")**
(4) Then enter **SS1 = SS1$SS1**
(5) To clr normalize the data type **SS1CLR = CLR_FN(SS1)**
(6) SS1CLR is now the normalized data. To move it back to matlab enter **write.table(SS1CLR, "SS1CLR.txt", quote = FALSE, sep = "\t")**. This file can then be imported using the importdata function

4. SCDE

SCDE Preparation

Note that this is done in R

(1) Download XQuartz
(2) Download the following R packages
    (a) SCDE version 1.99.4
    (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)

(c) data.table

Running SCDE
http://hms-dbmi.github.io/scde/diffexp.html is the vignette for SCDE. 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). 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 will assume that the data is called SS1
(2) Even if the data look like integers they may not be. It may be necessary to run the following code: **SS1 = apply(SS1,2,function(x) {storage.mode(x) = 'integer'; x})**.
(3) The following code will take some time to run and should be done on the cluster.
o.ifm = scde.error.models(counts = SS1, n.cores = X, min.size.entries = Y, max.pairs = Z)
    (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
    (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
(4) Now enter **valid.cells = o.ifm$corr.a > 0** and **o.ifm = o.ifm[valid.cells,]**
(5) Now to calculate the prior type **SS1p = scde.expression.prior(models = o.ifm, counts = SS1, length.out = 400, show.plot = FALSE)**
(6) To calculate the matrix where each entry is the probability of a dropout type in **SS1f = scde.failure.probability(models = o.ifm, counts = SS1)**
(7) Finally in preparation to produce the fail curves (probability of drop out on the y axis and gene magnitude expression on the x) enter **SS1fc = scde.failure.probability(o.ifm, magnitudes = log((10^SS1p$x)-1))**

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, "/Users/arozu7/Desktop/SS1Prior.txt", sep="\t")**
    (b) **write.table(SS1fc, "/Users/arozu7/Desktop/SS1FailCurves.txt", sep="\t")**
(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 the folder containing the files we just printed out and enter:
    (a) **SS1fc = fread("SS1failCurves.txt")**
    SS1fc = SS1fc[,-1]
    (b) **SS1p = fread("SS1prior.txt")**
    SS1p = SS1p[,-1]
    colnames(SS1p) = c("x", "y", "lp", "grid.weight")
(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)

plot(c(), c(), xlim=range(SS1p$x), ylim=c(0,1), xlab="expression magnitude (log10)", ylab="drop-out probability", main = "SS1 Dropout")
invisible(apply(SS1fc, 2, function(y) lines(x = SS1p$x, y = y,col = "orange")))

(4) Ideally the graph should overall have a negative correlation. 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".

Multiple Random Imputations

(1) The first step is to print out the fail matrix. This is done by entering **write.table(SS1f, "/Users/arozu7/Desktop/SS1Fail.txt", sep="\t")**
(2) Now open matlab and import this file (for our purposes it will be called SS1f)
(3) To do multiple random imputations use the function "MRI". Type in matlab **imputedSS1 = MRI(SS1.data,SS1f.data, X)** where X is the number of desired imputations (75 was used).
(4) imputedSS1 is the the final matrix