Stat M254 Homework 1

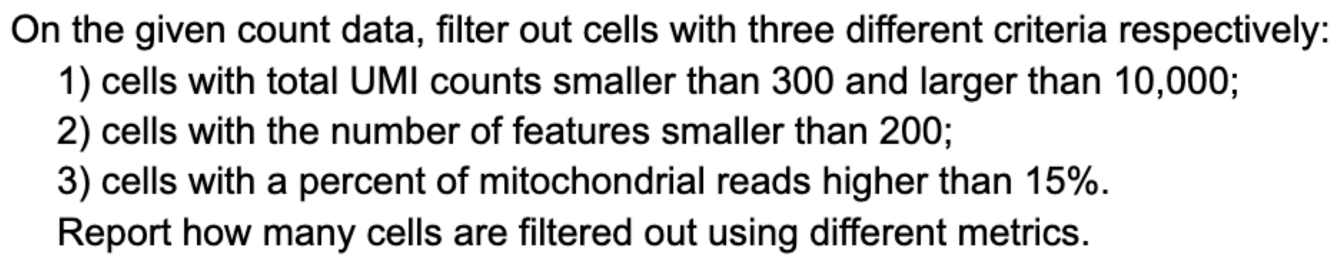
Due Apr 27 @ 11:59PM

{r})

```
        library(tidyverse)
        library(dplyr)
        library(Seurat)
        library(patchwork)
        library(Matrix)
        library(VennDiagram)
        library(sctransform)
        library(ggplot2)
        library(BayesTools)
        library(gridExtra)
```

Part 1: Real data [using the data from pbmc.csv]

Problem 1



On the given count data, filter out cells with three different criteria respectively:

1) cells with total UMI counts smaller than 300 and larger than 10,000;
2) cells with the number of features smaller than 200;
3) cells with a percent of mitochondrial reads higher than 15%.

Report how many cells are filtered out using different metrics.

Answer:

```
        cell_meta <- read.csv("data/pbmc.csv", row.names = 1)
```

```
        pbmc <- CreateSeuratObject(counts = cell_meta, project = "pbmc")
```

```
        pbmc
```

```
An object of class Seurat
33694 features across 5000 samples within 1 assay
Active assay: RNA (33694 features, 0 variable features)
 1 layer present: counts
```

```
        # metric 1
        pbmc1 <- subset(pbmc, subset = nCount_RNA > 300 & nCount_RNA < 10000)
```

```r
pbmc1
```

```
An object of class Seurat 
33694 features across 4844 samples within 1 assay 
Active assay: RNA (33694 features, 0 variable features)
 1 layer present: counts
```

```r
# metric 2
pbmc2 <- CreateSeuratObject(counts = cell_meta, project = "pbmc",
                           min.features = 200)
pbmc2
```

```
An object of class Seurat 
33694 features across 4900 samples within 1 assay 
Active assay: RNA (33694 features, 0 variable features)
 1 layer present: counts
```

```r
# metric 3
pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")
pbmc3 <- subset(pbmc, subset = percent.mt < 15)
pbmc3
```

```
An object of class Seurat 
33694 features across 4944 samples within 1 assay 
Active assay: RNA (33694 features, 0 variable features)
 1 layer present: counts
```

`Metric 1(UMI)` filtered 156 samples. `Metric 2(gene counts)` filtered 100 samples. `Metric 3(mitochondrial reads)` filterd 56 samples.

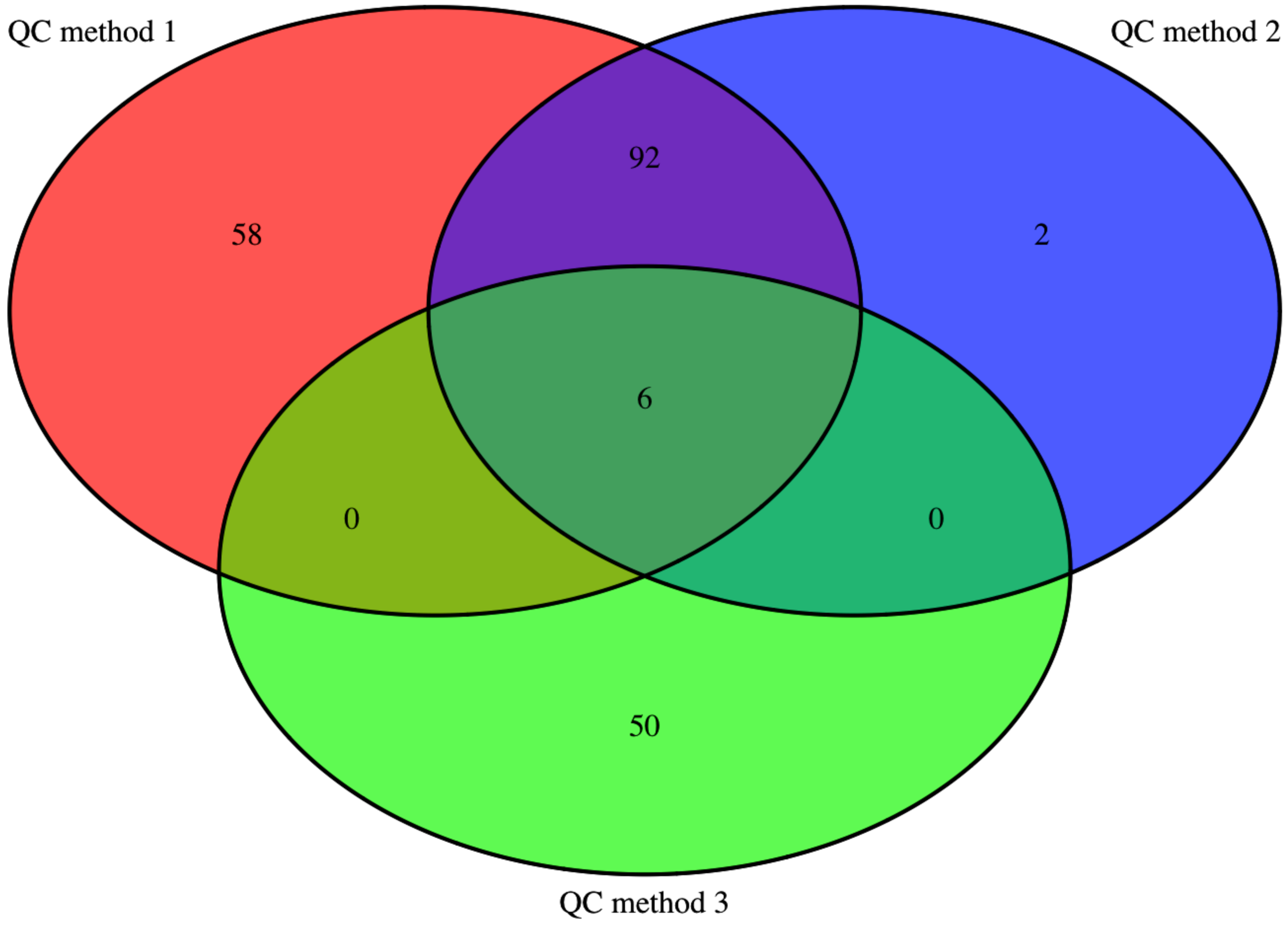
Problem 2

From question 1, you get three different sets of filtered cells. Plot the Venn diagram of these three sets. From the diagram, do the three different quality control methods filter out similar cells or not? Explain the reason why they have similar or dissimilar results.

Answer:

```r
allcells <- colnames(pbmc)
QC1 <- colnames(pbmc1)
QC2 <- colnames(pbmc2)
QC3 <- colnames(pbmc3)

cells1 <- setdiff(allcells, QC1)
cells2 <- setdiff(allcells, QC2)
cells3 <- setdiff(allcells, QC3)
```

```
        grid.newpage()

        venn.plot <- draw.triple.venn(
          area1 = length(cells1),
          area2 = length(cells2),
          area3 = length(cells3),
          n12 = length(intersect(cells1, cells2)),
          n23 = length(intersect(cells2, cells3)),
          n13 = length(intersect(cells1, cells3)),
          n123 = length(Reduce(intersect, list(cells1, cells2, cells3))),
          category = c("QC method 1", "QC method 2", "QC method 3"),
          fill = c("red", "blue", "green")
        )

        grid.draw(venn.plot)
```



Since it is difficult to interpret the Venn diagram with three sets of filtering, I plot the Venn diagram with three set of what had been filtered out for each group.

All these three methods filtered out some similar cells but they also filtered dissimilar cells. In particular,

- `QC method 1` and `QC method 2` filtered out the 98 same cells
- `QC method 1` and `QC method 3` filtered out the 6 same cells

- `QC method 2` and `QC method 3` filtered out the 6 same cells
- `QC method 1` filtered out 58 cells which are not filtered by `QC method 2` and `QC method 3`
- `QC method 2` filtered out 2 cells which are not filtered by `QC method 1` and `QC method 3`
- `QC method 3` filtered out 50 cells which are not filtered by `QC method 1` and `QC method 2`

There are 6 cells that all three method filtered out.

It is reasonable that cells filtered by `QC method 1` and `QC method 2` are similar because total UMI counts sums up UMI count in all genes and cells with less genes are likely to have less total UMI counts. It also makes sense that cells filtered by `QC method 3` are likely dissimilar to the other 2 QC methods since mitochondrial reads are not directly related to UMI counts or gene occurrance.

Problem 3

Filtered out the cells using all three criteria. Then, also filtered out genes expressed in less than 10 cells. On the filtered data, perform four different normalization methods: Seurat default log-normalization (log1pCP10k), log1pCPM, SCTransform [you need to set variable.features.n = the number genes after filtering so that SCTransform doesn't select highly variable genes], and log1pPF (Booeshaghi et al., 2022 and Github)

```
log1pPF = function(counts){
  # counts data is the #genes by #cells matrix
  ls = colSums(counts) # library size for each cell
  meanLS = mean(ls) # mean library size
  mat = t(log1p(apply(counts, 1, function(x){ x/ls * meanLS })))
  return(mat)
}
```

Answer:

```
log1pPF = function(counts){
  ls = colSums(counts)
  meanLS = mean(ls)
  mat = t(log1p(apply(counts, 1, function(x) x / ls * meanLS)))
  return(mat)
}
```

```
#filter data
pbmc <- CreateSeuratObject(counts = cell_meta, project = "pbmc",
                           min.cells = 10, min.features = 200)

pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")

pbmc_filtered <- subset(pbmc,
                        subset = nCount_RNA > 300 & nCount_RNA < 10000
                        & percent.mt < 15)
```

```
pbmc_filtered
```

```
An object of class Seurat 
15708 features across 4790 samples within 1 assay 
Active assay: RNA (15708 features, 0 variable features)
 1 layer present: counts
```

```
# Normalize the data
pbmc_norm1 <- NormalizeData(pbmc_filtered,
                            normalization.method = "LogNormalize",
                            scale.factor = 10000)
pbmc_norm2 <- NormalizeData(pbmc_filtered,
                            normalization.method = "LogNormalize",
                            scale.factor = 1000000)
# run sctransform
pbmc_norm3 <- SCTransform(pbmc_filtered, variable.features.n = 15708,
                          verbose = FALSE)

pbmc_norm4 <- log1pPF(pbmc_filtered[['RNA']]$counts)
pbmc_norm4 <- CreateSeuratObject(counts = pbmc_norm4, project = "pbmc")
```

Problem 4

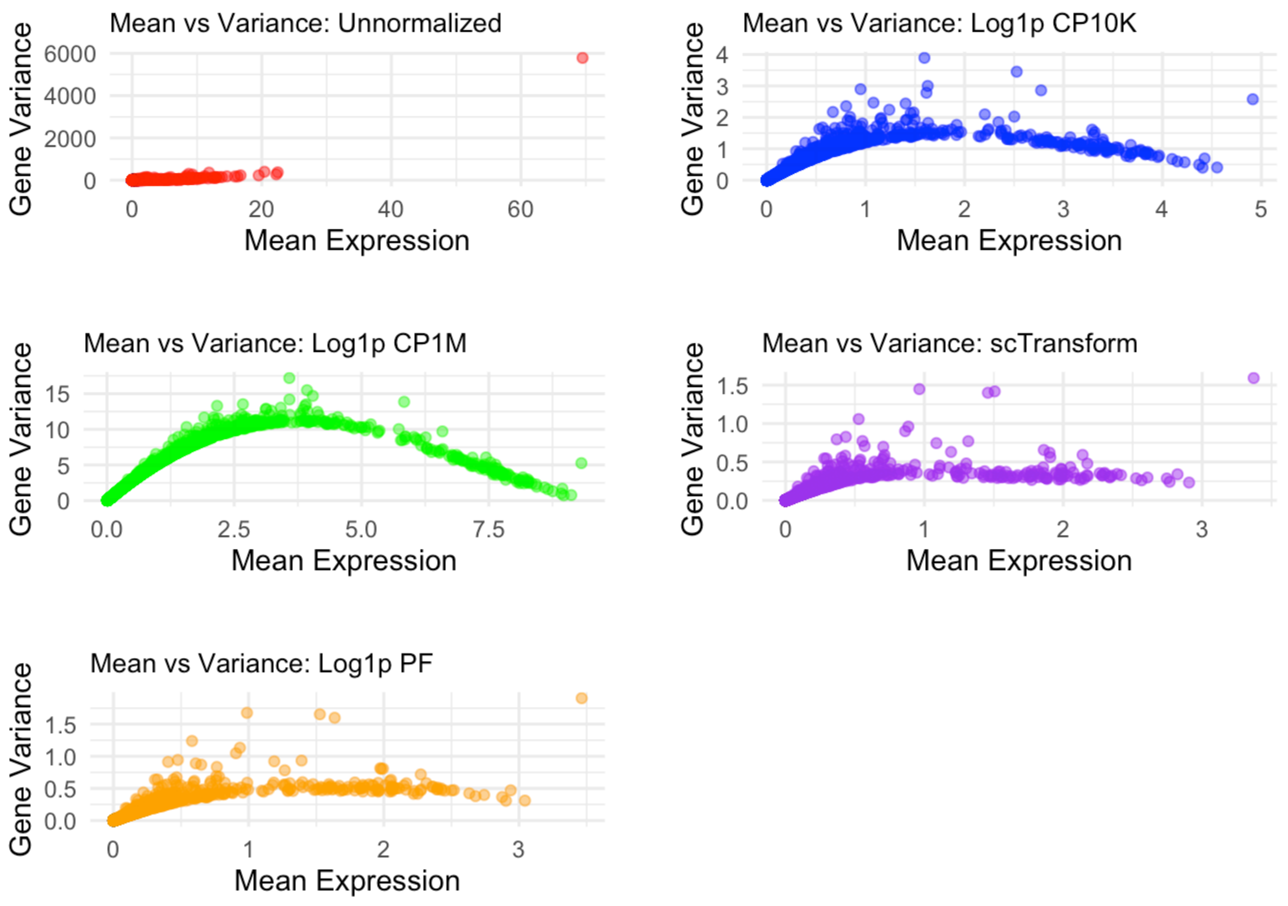
Calculate gene variances on (1) filtered count data and (2) four normalized data, respectively. Also, calculate the gene means before normalization. Draw five different plots showing the gene means vs. different variances. One of the plots should use the variances from the filtered count data. Each of the other four plots shows the variances derived from each normalized data. Compare the plots before and after normalization and explain whether the normalization methods could stabilize the variance.

Answer:

```
# Function to plot mean vs. variance
plotMVP <- function(SeuratObj, name, type, color){
  counts_matrix <- GetAssayData(SeuratObj, slot = type)
  counts_matrix <- as.matrix(counts_matrix)

  gene_means = rowMeans(counts_matrix)
  gene_variances = apply(counts_matrix, 1, var)

  gene_data = data.frame(
    Mean = gene_means,
    Variance = gene_variances
  )
```

```r
  p <- ggplot(gene_data, aes(x = Mean, y = Variance, color = name)) +
    geom_point(alpha = 0.5) +
    labs(x = "Mean Expression", y = "Gene Variance",
         title = paste0("Mean vs Variance: ", name)) +
    theme_minimal() +
    scale_color_manual(values = color) +
    theme(plot.title = element_text(size = 10),  # Reduce title font size
          plot.margin = unit(c(1, 1, 1, 1), "lines"),  # Adjust plot margins
          legend.position = "none")  # Hide the legend

  return(p)
}
```

```r
# Generate plots with specific colors
p0 <- plotMVP(pbmc_filtered, "Unnormalized", "counts", "red")
p1 <- plotMVP(pbmc_norm1, "Log1p CP10K", "data", "blue")
p2 <- plotMVP(pbmc_norm2, "Log1p CP1M", "data", "green")
p3 <- plotMVP(pbmc_norm3, "scTransform", "data", "purple")
p4 <- plotMVP(pbmc_norm4, "Log1p PF", "counts", "orange")

# Arrange plots in a grid
grid.arrange(p0, p1, p2, p3, p4, ncol = 2, nrow = 3)
```



Before normalization, There exists outlier genes with extreme variance, extreme mean, or both. This makes it difficult to observe the true mean variance relationship from the data. After normalization, the extreme outliers are being normalized so the mean and variance are within a reasonale scale for us to observe the relationship. Both four normalization show s a pattern that variance starts increasing with mean and then decrease at a certain point with increase of the mean. `log1pCP10K` and `log1pCP1M` show more extreme concave pattern than the other two normalization methods. It seems like all four normalization methods do not stabilize the variance. As mean increases, the variance changes according to the mean. This is a typical sign that variance is not stable across the mean.

Problem 5

Use the Spearman correlation between gene means and variances to evaluate the variance stabilization for each normalization method. And compare it to the Spearman correlation between gene means and variances using count data. Do you think normalization methods could stabilize variance, and which method has the best performance?

Answer:

```
# Function to calculate Spearman correlation between mean and variance of gene ex
calculateSpearman <- function(seurat_obj, type) {
  # Get the data
  data_matrix <- GetAssayData(seurat_obj, slot = type)
  data_matrix <- as.matrix(data_matrix)
  # Calculate mean and variance
  means = rowMeans(data_matrix)
  variances = apply(data_matrix, 1, var)

  # Calculate Spearman correlation
  correlation = cor(means, variances, method = "spearman")
  return(correlation)
}
```

```
# Calculate Spearman correlation for each normalization method
spearman_filtered <- calculateSpearman(pbmc_filtered, "counts")
spearman_norm1 <- calculateSpearman(pbmc_norm1, "data")
spearman_norm2 <- calculateSpearman(pbmc_norm2, "data")
spearman_norm3 <- calculateSpearman(pbmc_norm3, "data")
spearman_norm4 <- calculateSpearman(pbmc_norm4, "counts")

# Print the Spearman correlations
print(paste("Spearman correlation for unnormalized data:", spearman_filtered))
```

```
[1] "Spearman correlation for unnormalized data: 0.995211040210693"
```

```
print(paste("Spearman correlation for log1pCP10K:", spearman_norm1))
```

```
[1] "Spearman correlation for log1pCP10K: 0.998688010272964"
```

```
print(paste("Spearman correlation for log1pCP1M:", spearman_norm2))
```

```
[1] "Spearman correlation for log1pCP1M: 0.9987682882677"
```

```
print(paste("Spearman correlation for scTransform:", spearman_norm3))
```

```
[1] "Spearman correlation for scTransform: 0.999359445729681"
```

```
print(paste("Spearman correlation for log1pPF:", spearman_norm3))
```

```
[1] "Spearman correlation for log1pPF: 0.999359445729681"
```

I do not think normalization could stabilize mean in this example. The Spearman correlation between mean and variance is still high after normalization. This indicates that the variance is still highly dependent on the mean. Reversely, unnormalized data has relatively the least correlation. Among the four normalization method, `log1pCP1M` gives relatively the lowest correlation. However, all correlations are super high and very close to each other.

Part 2: Simulated data

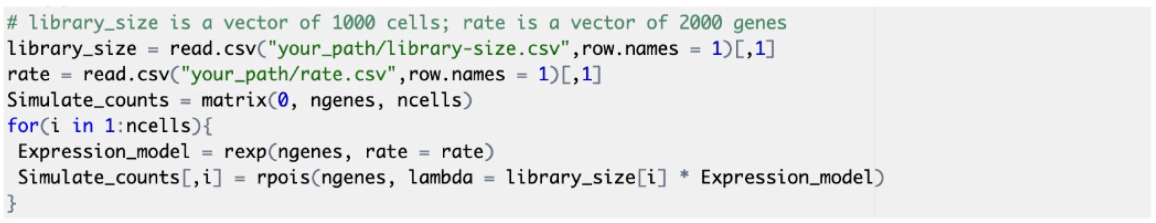
Problem 6

6. In the article Sarkar and Stephens, 2021, they proposed a general scRNA-seq model using the measurement model and expression model, i.e.,

$$\begin{aligned}
&\text{Measurement model: } x_{gc}|x_{c+}, \lambda_{gc} \sim \text{Poisson}(x_{c+}\lambda_{gc}),\\
&\text{Expression model: } \lambda_{gc} \sim G_g(\cdot),\\
&\text{where,}\\
&x_{gc} \text{ is the observed expression level of gene } g \text{ for cell } c\\
&x_{c+} \text{ is the observed library size of cell } c
\end{aligned}$$

With different assumed expression models $G_g(\cdot)$, the observed expression x_{gc} will also have different distributions. And we assume that genes are independent of each other. For the following questions, use the provided parameter files to simulate different count data with 1000 cells and 2000 genes.

Hint: Suppose the expression model follows the exponential distribution. The demo code for simulating count data in R is as follows.



```
# library_size is a vector of 1000 cells; rate is a vector of 2000 genes
library_size = read.csv("your_path/library-size.csv",row.names = 1)[,1]
rate = read.csv("your_path/rate.csv",row.names = 1)[,1]
Simulate_counts = matrix(0, ngenes, ncells)
for(i in 1:ncells){
 Expression_model = rexp(ngenes, rate = rate)
 Simulate_counts[,i] = rpois(ngenes, lambda = library_size[i] * Expression_model)
}
```

1)

Assume the expression model is the point mass distribution, i.e., $\lambda_{gc} \sim \delta_{\alpha_g}$ [this means that λ_{gc} is a constant of α_g ; you could check the definition on page 7 of the slides]. Use the *point-mass.csv* [stores the point mass parameter for each gene] and *library-size.csv* files to simulate the count data. If the expression model is a point mass distribution, what is the distribution of simulated genes? [Hint: R function: rpois]

Answer:

```
#Simulate data
set.seed(321)

ncells = 1000
ngenes = 2000

library_size = read.csv("data/library-size.csv", row.names = 1)[, 1]
point_mass = read.csv("data/point-mass.csv", row.names = 1)[, 1]
Simulate_counts_1 = matrix(0, ngenes, ncells)
for(i in 1:ncells){
  Expression_model = point_mass
  Simulate_counts_1[,i] = rpois(ngenes,
                                lambda = library_size[i] * Expression_model)
}
```

Based on definition of point mass distribution, $P(\lambda_{gc} = \alpha_g) = 1$

Thus, λ_{gc} is a constant of α_g for all g and c

$X_{gc} \sim Poisson(x_{c+}\alpha_g)$

2)

Assume the expression model is the Gamma distribution, i.e., $\lambda_{gc} \sim Gamma(\alpha_g, \beta_g)$. Use the *shape+scale.csv* [stores the shape and scale parameters of Gamma distribution for each gene] and *library-size.csv* files to simulate the count data. If the expression model is a Gamma distribution, what is the distribution of simulated genes? [Hint: R function: rgamma]

Answer:

```
set.seed(321)

ncells = 1000
ngenes = 2000

#Simulate data
library_size = read.csv("data/library-size.csv", row.names = 1)[, 1]
```

```r
        shape_scale = read.csv("data/shape+scale.csv", row.names = 1)[, c(1, 2)]
        Simulate_counts_2 = matrix(0, ngenes, ncells)
        for(i in 1:ncells){
          Expression_model = rgamma(ngenes, scale = shape_scale[, 1],
                                    shape = shape_scale[, 2])
          Simulate_counts_2[,i] = rpois(ngenes,
                                     lambda = library_size[i] * Expression_model)
        }
```

Since gamma distribution has the scaling property.

$x_{c+}\lambda_{gc} \sim Gamma(x_{c+}\alpha_g, \beta_g)$

$X_{gc} \sim NB(x_{c+}\alpha_g, \frac{\beta_g}{1+\beta_g})$

Here is the proof:

Without loss of generality, we will redefine random variables.

Let $X \sim Poisson(\boldsymbol{\theta})$, $\boldsymbol{\theta} \sim Gamma(\alpha, \frac{p}{1-p})$

By definition,

$p(x;\boldsymbol{\theta}) = \int_0^\infty p(x;\theta)p(\theta)dx$

Since pdf of poisson and gamma distributions are known, we have:

$$\begin{aligned}
p(x;\boldsymbol{\theta}) &= \int_0^\infty \frac{\theta^x e^{-\theta}}{x!}\frac{1}{\Gamma(\alpha)(\frac{p}{1-p})^\alpha}\theta^{\alpha-1}e^{-\theta(\frac{1-p}{p})}dx \\
&= \frac{(1-p)^\alpha p^{-\alpha}}{\Gamma(\alpha)x!}\int_0^\infty \theta^{x+\alpha-1}e^{-\frac{\theta}{p}}dx \\
&= \frac{(1-p)^\alpha p^{-\alpha}}{\Gamma(\alpha)x!}\Gamma(x+\alpha)p^{x+\alpha} \\
&= \frac{\Gamma(x+\alpha)}{\Gamma(\alpha)x!}(1-p)^\alpha p^x \\
&= \binom{x+\alpha-1}{x}(1-p)^\alpha p^x \quad \text{where } x = 0, 1, 2, \cdots
\end{aligned}$$

Since $p(x;\boldsymbol{\theta})$ is a pmf of negative binomial distribution, we have $X \sim NB(\alpha, p)$

Problem 7. Answering the following questions

1) Apply four different normalization methods in question 3 on each simulated dataset.

Answer:

```
sim1 <- CreateSeuratObject(counts = Simulate_counts_1,
                           min.cells = 10, min.features = 200)
```

```
sim1_norm1 <- NormalizeData(sim1,
                            normalization.method = "LogNormalize",
                            scale.factor = 10000)
sim1_norm2 <- NormalizeData(sim1,
                            normalization.method = "LogNormalize",
                            scale.factor = 1000000)
# run sctransform
sim1_norm3 <- Seurat::SCTransform(
  sim1,
  variable.features.n = nrow(sim1@assays$RNA@layers$counts),
  return.only.var.genes = FALSE)

sim1_norm4 <- log1pPF(sim1[['RNA']]$counts)
sim1_norm4 <- CreateSeuratObject(counts = sim1_norm4, project = "sim1")
```

```
sim2 <- CreateSeuratObject(counts = Simulate_counts_2,
                           min.cells = 10, min.features = 200)
```

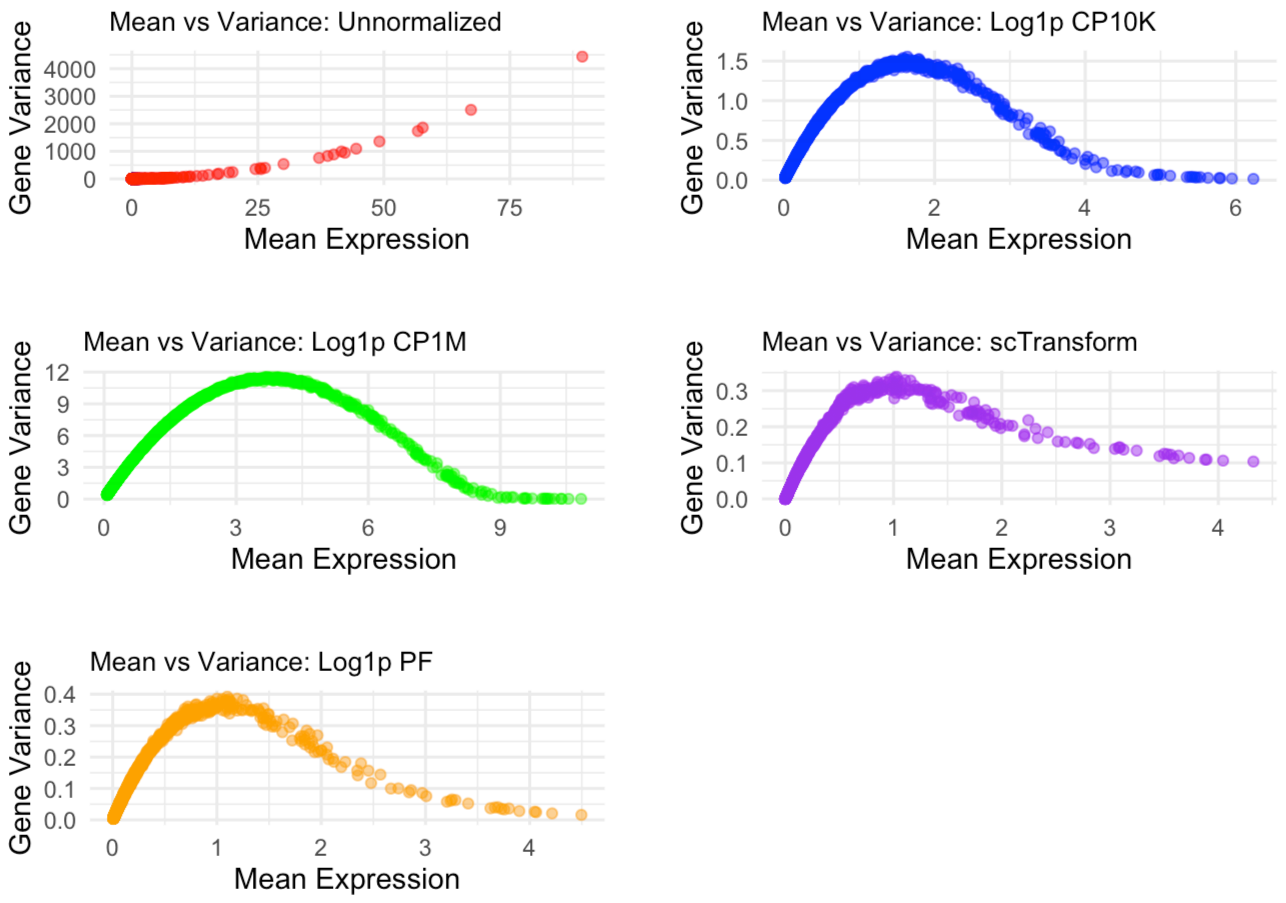
```
sim2_norm1 <- NormalizeData(sim2,
                            normalization.method = "LogNormalize",
                            scale.factor = 10000)
sim2_norm2 <- NormalizeData(sim2,
                            normalization.method = "LogNormalize",
                            scale.factor = 1000000)
# run sctransform
sim2_norm3 <- SCTransform(sim2,
                          variable.features.n = nrow(sim2@assays$RNA@layers$count
                          verbose = FALSE, return.only.var.genes = FALSE)

sim2_norm4 <- log1pPF(sim2[['RNA']]$counts)
sim2_norm4 <- CreateSeuratObject(counts = sim2_norm4, project = "sim2")
```

2) Draw 5 figures with gene means vs variances as in question 4 for each simulated dataset.

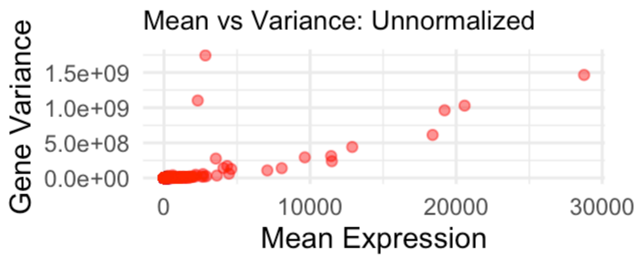
Answer:

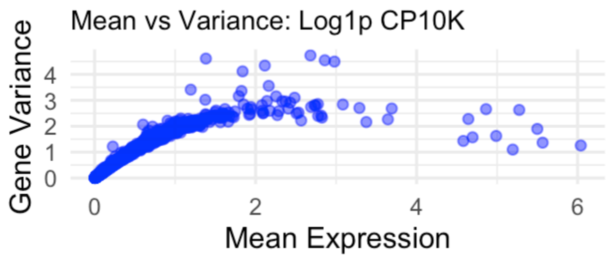
```
# Generate plots with specific colors
p0 <- plotMVP(sim1, "Unnormalized", "counts", "red")
p1 <- plotMVP(sim1_norm1, "Log1p CP10K", "data", "blue")
p2 <- plotMVP(sim1_norm2, "Log1p CP1M", "data", "green")
p3 <- plotMVP(sim1_norm3, "scTransform", "data", "purple")
p4 <- plotMVP(sim1_norm4, "Log1p PF", "counts", "orange")
```

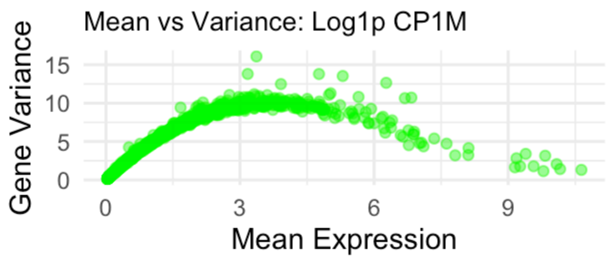
```
# Arrange plots in a grid
grid.arrange(p0, p1, p2, p3, p4, ncol = 2, nrow = 3)
```

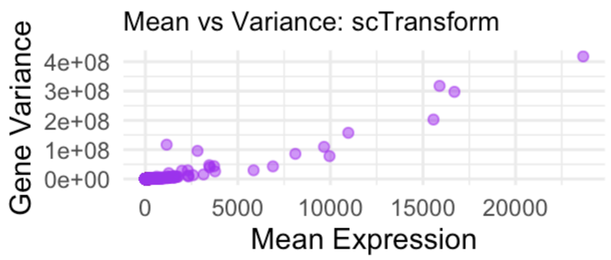


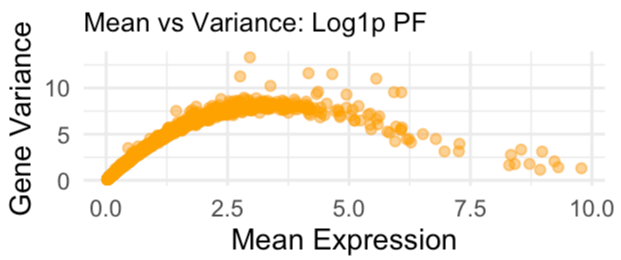
```
# Generate plots with specific colors
p0 <- plotMVP(sim2, "Unnormalized", "counts", "red")
p1 <- plotMVP(sim2_norm1, "Log1p CP10K", "data", "blue")
p2 <- plotMVP(sim2_norm2, "Log1p CP1M", "data", "green")
p3 <- plotMVP(sim2_norm3, "scTransform", "counts", "purple")
p4 <- plotMVP(sim2_norm4, "Log1p PF", "counts", "orange")

# Arrange plots in a grid
grid.arrange(p0, p1, p2, p3, p4, ncol = 2, nrow = 3)
```











3) Use Spearman correlation to evaluate normalization methods as in question 5 for each simulated dataset.

```
# Calculate Spearman correlation for each normalization method
spearman_filtered <- calculateSpearman(sim1, "counts")
spearman_norm1 <- calculateSpearman(sim1_norm1, "data")
spearman_norm2 <- calculateSpearman(sim1_norm2, "data")
spearman_norm3 <- calculateSpearman(sim1_norm3, "data")
spearman_norm4 <- calculateSpearman(sim1_norm4, "counts")

# Print the Spearman correlations
print(paste("Spearman correlation for unnormalized data:", spearman_filtered))
```

```
[1] "Spearman correlation for unnormalized data: 0.999244989771759"
```

```
print(paste("Spearman correlation for log1pCP10K:", spearman_norm1))
```

```
[1] "Spearman correlation for log1pCP10K: 0.906945581355425"
```

```
print(paste("Spearman correlation for log1pCP1M:", spearman_norm2))
```

```
[1] "Spearman correlation for log1pCP1M: 0.837004724535116"
```

```r
print(paste("Spearman correlation for scTransform:", spearman_norm3))
```

```
[1] "Spearman correlation for scTransform: 0.992175528934755"
```

```r
print(paste("Spearman correlation for log1pPF:", spearman_norm3))
```

```
[1] "Spearman correlation for log1pPF: 0.992175528934755"
```

```r
# Calculate Spearman correlation for each normalization method
spearman_filtered <- calculateSpearman(sim2, "counts")
spearman_norm1 <- calculateSpearman(sim2_norm1, "data")
spearman_norm2 <- calculateSpearman(sim2_norm2, "data")
spearman_norm3 <- calculateSpearman(sim2_norm3, "data")
spearman_norm4 <- calculateSpearman(sim2_norm4, "counts")

# Print the Spearman correlations
print(paste("Spearman correlation for unnormalized data:", spearman_filtered))
```

```
[1] "Spearman correlation for unnormalized data: 0.969916316626626"
```

```r
print(paste("Spearman correlation for log1pCP10K:", spearman_norm1))
```

```
[1] "Spearman correlation for log1pCP10K: 0.996779222818654"
```

```r
print(paste("Spearman correlation for log1pCP1M:", spearman_norm2))
```

```
[1] "Spearman correlation for log1pCP1M: 0.970198820511449"
```

```r
print(paste("Spearman correlation for scTransform:", spearman_norm3))
```

```
[1] "Spearman correlation for scTransform: 0.979206170967878"
```

```r
print(paste("Spearman correlation for log1pPF:", spearman_norm3))
```

```
[1] "Spearman correlation for log1pPF: 0.979206170967878"
```