Next Generation Sequencing Technologies: Data Analysis and Applications

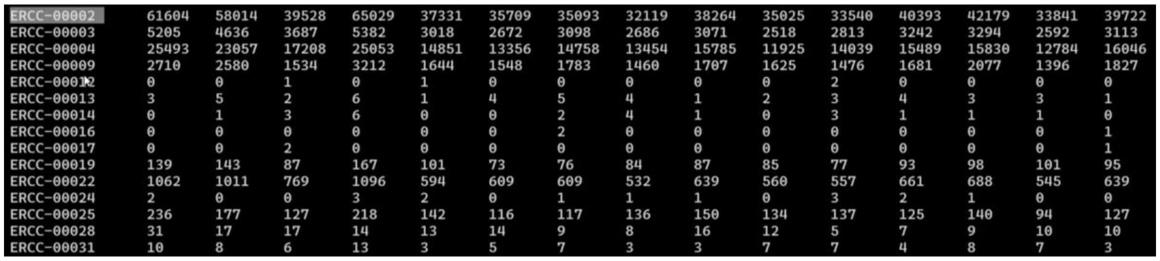
Analysis with spike-ins

Dr. Riddhiman Dhar, Department of Biotechnology
Indian Institute of Technology Kharagpur

Good day, everyone. Welcome to the course on nationwide sequencing technologies, data analysis, and applications. We have been working on an RNA-Seq dataset where we have been analyzing the data. The goal is to identify the genes that show differences in expression across two different conditions. So, just to remind you about the data very briefly, So, this is a dataset with 16 samples, and we have about 6800 genes, ok?

And the 16 samples are organized into four groups, and each group contains four technical replicates. So, if you remember, the groups are group 1, group 2, group 3, and group 4. And in the last class we have been looking at, we have been looking at the differential expression analysis using DESeq2. So, this is the tool that we used, and if you remember, we followed all the steps, starting with the count data.

What we are going to do today is look at the same differential gene expression analysis, but with the spike in data in there, ok. So, these are synthetic RNA molecules called ERCC spike in or ERCC spike in mix that will be used for normalization. So, the dataset that we have been working with also contains this ERCC spike. Just to remind you again about the spike ins that we talked about in the theory classes, these spike ins are synthetic RNA molecules; they contain RNA molecules of different lengths. And they are added in different quantities or different numbers in each sample, ok?



```
ERCC-00002     61604   58014   39528   65029   37331   35709   35093   32119   38264   35025   33540   40393   42179   33841   39722
ERCC-00003     5205    4636    3687    5382    3018    2672    3098    2686    3071    2518    2813    3242    3294    2592    3113
ERCC-00004     25493   23057   17208   25053   14851   13356   14758   13454   15785   11925   14039   15489   15830   12784   16046
ERCC-00009     2710    2580    1534    3212    1644    1548    1783    1460    1707    1625    1476    1681    2077    1396    1827
ERCC-00012     0       0       1       0       1       0       0       0       0       0       2       0       0       0       0
ERCC-00013     3       5       2       6       1       4       5       4       1       2       3       4       3       3       1
ERCC-00014     0       1       3       6       0       0       2       4       1       0       3       1       1       1       0
ERCC-00016     0       0       0       0       0       0       2       0       0       0       0       0       0       0       1
ERCC-00017     0       0       2       0       0       0       0       0       0       0       0       0       0       0       1
ERCC-00019     139     143     87      167     101     73      76      84      87      85      77      93      98      101     95
ERCC-00022     1062    1011    769     1096    594     609     609     532     639     560     557     661     688     545     639
ERCC-00024     2       0       0       3       2       0       1       1       1       0       3       2       1       0       0
ERCC-00025     236     177     127     218     142     116     117     136     150     134     137     125     140     94      127
ERCC-00028     31      17      17      14      13      14      9       8       16      12      5       7       9       10      10
ERCC-00031     10      8       6       13      3       5       7       3       3       7       7       4       8       7       3
```

So, just to clarify that each sample will get exactly the same amount of these RNA molecules, ok? So, there is no technical variation here, right? So, there is no biological variation here. What you will get is purely a technical variation, right? So, what you do is add exactly the same amount of spike to each sample, and at the end we get the read data. If you see variation in the read data, that will be technical variation, and we can then estimate the technical variation that arises because of the differences in sample preparation, etcetera.

So, this will help us differentiate between technical variation and biological variation better than the other methods, where we have to assume some sort of distribution, perhaps, or assume that most of the genes are not showing any change in expression, etcetera. If you can also extend this using these spikes, you can also estimate the absolute RNA molecule number in the sample. So, this is possible because you know the number of molecules in each type of synthetic RNA that you are adding, right? So, based on that, you can calculate the RNA molecule of the sample that might be present, ok? So, this is something we will not go into again, but this is also more challenging because of the differences in the RNA sequence, right?

So, there is also sequence-specific bias; there are other differences in GC content, etcetera. So, we will not go into that part of the absolute quantification; what we will work with is the relative quantification, right? So, relative abundance between two samples or two different conditions is okay. So, this is the agenda for today's class. We will be working with the GC analysis, where the data contains these ERCC spikes, ok? So, again, the steps we have completed are the first few steps that are shown in green, right?

So, we have been actually working on the differential gene expression analysis. We have done the first differential gene expression analysis using these two. Today we are going to use the ERCC spike-in data to do the same analysis and see if we get similar results or not. This is the agenda. So, the tools that we will be using are these tools. So, one is a RUVSeq, ok, and here is the link to the package, ok.

We will see in a moment where this package is, how we can install it, and how to use it. And once we have normalized using RUVSeq, we can then use DEseq to actually do the differential expression analysis, right? So, the second part we have already done, the first part would be the novel part, ok? So, let us now move on to the terminal, where we can look at the first look at the data, and let us find out right where this spiking data is, ok. So, here we are in the terminal, in the folder where we have the data and the codes, right?

So, let us first look at the data file, ok? This is the data file, if you remember correctly. If you see that there will be 16 samples from S1 to S16 and on the left side you have the gene names, ok. So, the first column is the gene names, right? So, you have about 6800 genes here, and then you have the sample names on top.

So, this is the header line, right? So, when you load the data, we actually mention that this is the header line, right? So, we have been doing that in R. So, what I will do is, in this file, now we can see where these spike-ins are, ok? So, I will search for this ERCC, and you can see it, right?

There are some rows on the right, which are named ERCC 2, ERCC 3, and so on. You can see these numbers, right? So, these are the synthetic RNA molecules; these are not the gene names, ok? So, all the gene names you can see there look different. These are the synthetic RNA or spike-in molecules that have been added to these samples, ok?

So, these RNA molecules are different RNA molecules, and they are also added in different quantities to make the mix, ok? Why different quantities? To estimate, right, this variation in the actual number and length can affect the technical variation of the read data count. So, that is why we have this combination of these different RNA molecules, not just one molecule. And then you can see these numbers, like the abundance numbers that map to them, which is about 61,000, 5200, and so on. And for some spike-ins, you can see the numbers are very low, ok?

So, these are present in very small numbers in the sample, ok? So, you can see this huge variation, and then you can also probably notice that we have a lot of these as engineering molecules, and you can see this end here, right? So, you have this ERCC 171, ok? So, out of the 6800 genes, not all of them are genes. So, we have about 100 or more, 150 or so, right?

You can probably count how many there are actually of these ERCC spike-ins; ok, and the rest of them are genes, alright. So, we can use this ERCC data now to actually normalize the data and count, and we can counter or minimize the technical variation, right? So, you get the idea; hopefully we have added these spike-in mixes, right? So, each mix contains these multiple RNA molecules in different numbers, ok, and we add this mix exactly the same amount across all the samples, ok. So, there is no variation due to environmental conditions, etcetera. When we are preparing the library we have isolated RNA, and we add this molecule to this library to this sample.

And then we get this variation in the data, which means this is purely a technical variation. We can now look into the pattern, etcetera, and try to correct for this technical variation as much as possible. So, this is what we are going to do. We will import this data in R, and we will try to normalize the data based on these ERCC spike-ins, ok? So, one of the things you probably now realize is that when you are doing the mapping in RNA-Seq, ok, when you are building the index, you would also have to add this ERCC data right in the index, ok. So, you have to have the ERCC FASTA files that are available, and they will be combined with your transcriptome or genome data, reference genome data, or reference transcriptome data, and the program will build an index for that, right?

Otherwise, you will not get this mapping against these molecules, ok? So, this is something we need to remember, ok? So, we will use this package, DEseq2. I showed you this in the last class. So, here is the Bioconductor Package, where you can go to this page and download it. We have also talked about how we can look at the manuals here, the right reference manual and find out the commands that we can use.

The package that we will use today is called RUVseq. So, as you can see, the name of this package is Remove Unwanted Variation from RNA-Seq Data. So, this is what we want to do, right? So, we want to remove this unwanted technical variation that might be present in our sample, and the citation is given. This is the paper where this method has been proposed or the package has been published.

To install this Bioconductor package, we will have to use this Biocmanager, right? We utilized this last time, if you hopefully remember. So, here it is, right? Biocmanager installs RUVseq. This will take a bit of time, ok? And in the meantime, we can, ok?

So, this is actually already there, and I probably do not want to update any packages at this moment. So, it is telling me that this package is already installed, ok? So, if it is not, then it will be installed step by step, ok? So, what we will do is now have a look at what this package can do. And we have the manual around here, ok?

So, I clicked on this link, and you can have all the commands, right? These are the commands that are available for this package, ok? And we will use one of these for our data, ok? And again, we have all the descriptions, all the attributes that a command can take, and sometimes some examples, right? So, the usage attributes and examples will be given.

The methods that we will use are, so there are three methods you can see at the top if you go. You have this RUVg method, you have RUVr, and you have RUVs, ok? So, what are these? Let us have a look, right? We can click on this link.

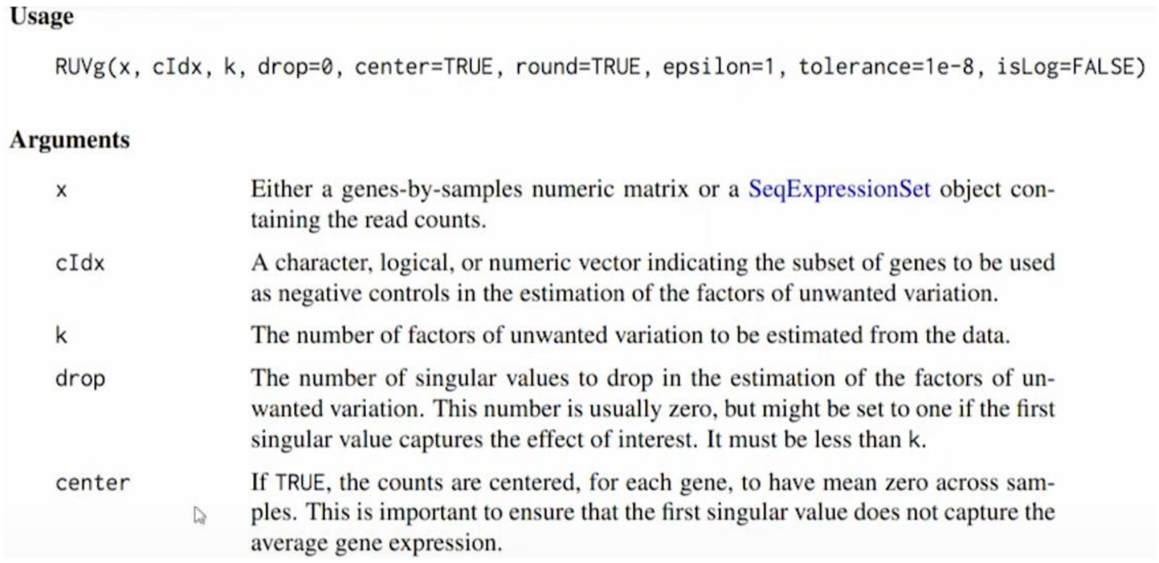
So, this is RUVg, right? So, in the full form you can see here, remove unwanted variation using control genes, ok? So, if you have a set of control genes that you know are not differentially expressed between samples, you can use that set for normalization. So, this is the RUVg method. Then, of course, you have this RUV, right? So again, this is to remove unwanted variation using residuals, ok?

So, here, you do not have these control genes. So, you do not have to specify that. And the last one is RUVs, ok? This is to remove unwanted variation using replicate negative control samples, ok? So, this is again a different method with a different requirement.

But what you probably have realized now is that in our case, we can use this RUV-G, right? Because this removes this variation using control genes, we can consider these spike genes as control genes, right? So, these are control genes, which means they do not show any difference in expression across samples. So, there is no biological variation. And so, ERCC spike genes feed that criteria, right?

So, we are not seeing any; we should not see any variation in this ERCC data across samples, right? So, the ERCC gene will be added later on after we have isolated the library from the samples, and they will be added in equal amounts, ok? So, we can use this RUVg, and we can say, OK, we can use this ERCC as the control gene. So, we just have to specify these ERCC genes as the control genes for this method, ok?

So, this is what we are going to do. Then we have the usage that is mentioned, right? So, you have RUVg; you have these attributes that are given. This X is usually a gene-by-sample matrix, or it could be a SeqExpressionSet. Again, it can take a specific data format, right? And we have seen how we can create this SeqExpressionSet in the last few labs, I think, right?



Usage

```
RUVg(x, cIdx, k, drop=0, center=TRUE, round=TRUE, epsilon=1, tolerance=1e-8, isLog=FALSE)
```

Arguments

x	Either a genes-by-samples numeric matrix or a SeqExpressionSet object containing the read counts.
cIdx	A character, logical, or numeric vector indicating the subset of genes to be used as negative controls in the estimation of the factors of unwanted variation.
k	The number of factors of unwanted variation to be estimated from the data.
drop	The number of singular values to drop in the estimation of the factors of unwanted variation. This number is usually zero, but might be set to one if the first singular value captures the effect of interest. It must be less than k.
center	If TRUE, the counts are centered, for each gene, to have mean zero across samples. This is important to ensure that the first singular value does not capture the average gene expression.

So, when you are doing this hands-on, we have seen that we have created this SeqExpressionSet data format using this new SeqExpressionSet command, ok? So, then you have these other attributes that we also sometimes need to specify, right? For example, whether you have rounded normalized measures, right, or epsilon, So, if you want to add this number, right before you block transform, because sometimes if your count is 0, you cannot take log 0. So, you can specify what would be the number that you want to use, right, and so on, right.

So, there are so many things, right? And then you have the K, which is the most important part, right? This is the number of factors of unwanted variation to be estimated from the data, ok? So, we can probably end up with just one or so, right? So, we can say one, two, etc. We will see, maybe we can just specify it one and we will try out this method, and what I will tell you is that you can try different cases and see how that will change the results, ok?

So, then there are some details and little expressions, as you can see, right? Here it clearly mentions several types of controls that can be used, right? So, you can use high-staging genes. If you know, you are sure that these are not showing any difference in expression, or you can use spike-ins, ok? So, specifically, it mentions ERCC spike-ins, and we have these ERCC spike-ins in our data.

So, we can use those for this RUVg method, ok? And then, of course, there are others, so here are references and then that example set, ok? So, we will use these commands. Now, we can see how this process will go, and we can identify or try out this method for our data. Once we have done that, we will use DEseq2 to do the differential expression analysis. So, let us try the first part, and as you can see here in the example, they also used k equal to 1.

So, we will try k equals 1. You can, and we can increase this later on. You can try it yourself, right? When you are doing this, ok? So, again, what I will do is actually collect these commands, right? So, we have this data, right?

So, let us go to this command line, ok? Here we are using this RUVg set package. We will load the data, and then we will complete the analysis, ok? So, the first part will be the new part, and then we will do the differential expression analysis using DEseq2. So, that is something you are already familiar with.

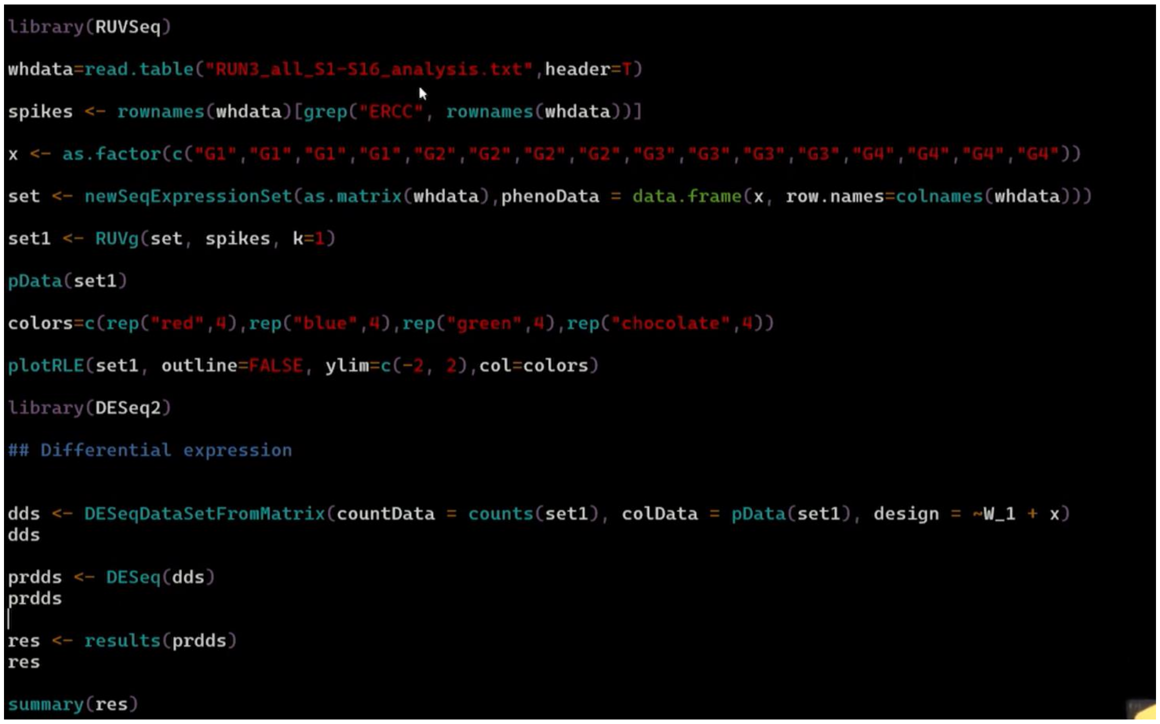
We did this in the last lab, ok? So, I will load the library first, ok? So, library RUVseq. So, it will load. It will give us some output or some commands, right, and then we need to load the data, right? So, here is the data—the bridge data—which I am going to load, ok?

So, here it is. It is giving some commands and some comments, right? It is saying it is masking some objects, etcetera. So, we do not have to worry about it. There is no error, at least.

So, what I am going to do is I am going to load the data. We have the header. So, we have to give header equals to true, ok, and the data is loaded correctly. You can see by this head command, right? So, we have the sample names, which are the column names, and then you have the gene names, which are the row names for our sample. So, the next step is to define what these control genes or spikes are, ok?

So, from this data, we have to mention, right, these are the rows that represent the control genes and the corresponding counts, ok? So, this is what we are going to do here with this command. As you can see, this is what we will do, ok? So, spikes, row names, and whdata So, we are actually getting the row names of the whdata and matching them with this grep ERCC, ok?

So, what grep ERCC means is that we are matching this word ERCC, right, for the gene name or row name in the row names here, right? So, this command is searching for this grep ERCC, row names, and whdata, right? It is searching for this data in this whdata and where there is a match that will be stored as spikes, ok. Those row names will be given as spikes, ok? So, let us try this command, ok, and see what we get, and then you will realize how this actually works, ok.



```
library(RUVSeq)

whdata=read.table("RUN3_all_S1-S16_analysis.txt",header=T)

spikes <- rownames(whdata)[grep("ERCC", rownames(whdata))]

x <- as.factor(c("G1","G1","G1","G1","G2","G2","G2","G2","G3","G3","G3","G3","G4","G4","G4","G4"))

set <- newSeqExpressionSet(as.matrix(whdata),phenoData = data.frame(x, row.names=colnames(whdata)))

set1 <- RUVg(set, spikes, k=1)

pData(set1)

colors=c(rep("red",4),rep("blue",4),rep("green",4),rep("chocolate",4))

plotRLE(set1, outline=FALSE, ylim=c(-2, 2),col=colors)

library(DESeq2)

## Differential expression

dds <- DESeqDataSetFromMatrix(countData = counts(set1), colData = pData(set1), design = ~W_1 + x)
dds

prdds <- DESeq(dds)
prdds

res <- results(prdds)
res

summary(res)
```

So, this first part, this part here, is looking and searching for this ERCC in the row name, ok, and the second part is only taking those row names where this match is found, ok. So, maybe I can try this out separately. You can see, right, if I write grep ERCC, row names, whdata, ok, it is giving me the row numbers where it is finding this match, ok, in the data and the spikes we can now try, ok, it gives us the names of these spike ends, ok. So, these spike ends represent the data. We have now identified those using this very simple command using the grep command, right?

So, this grep command we have talked about in the terminal is what we use here, ok? All right, so once we have now defined the spikes, what we are going to do is set up something called factors, ok? This is simply defining the design of the experiment, ok? So, what are these factors? So, this is simply saying, OK, so the order of the samples and the condition to which they belong is okay. So, in our case, the first four are group 1, the next four are group 2, the next four are group 3, and the last four are group 4, right?

That is what we are defining here, right? As a factor, we are defining this variable x, and as a factor, right, we are making this list from and the order of this should match the name of the samples, ok? So, it is simply getting this order of samples, right? So, which group they belong to, and the four levels, as you can see, are defined as g 1, g 2, g 3, and g 4, right? Four factors, ok? So, this is what we have done now. Now what we are going to do is create this sake expression set, ok?

This is something we have done before, and here we are going to do the same thing, ok? So, this is the command, right? If you go back now to how to create this sake expression set, we have this count data part, and we have the pheno data part, right? So, again, go back to the last class or maybe go back to the file, right? How to create this new sake expression set?

You will see we need the count data and the pheno data, right? So, that is what we are doing, and we are assigning this variable set with this data, ok? So, I will run this in here, ok? And so this is simply coming from the whole data that we have loaded, right? This is done in matrix form, which will be the count data, and then the pheno data is in the data frame format, right?

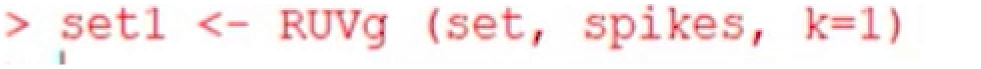
So, it is a x, right? So, the x is the factors that are given, right, the groups that we have defined just now. Then we have the row names, ok? So, the row names will be the column names of this data, right? So, the column names of this data are the sample names.

So, these sample names would be the row names of this sake expression matrix, ok? And let us see, we have done this now, and then if we look at the set, it will tell you, right, this is the sake expression set format. We have 6800 genes. Of course, this is not a gene; this includes ERCC. We have features, right: 16 samples, counts, normalized counts, and element names.

We have not defined the normalized counts here. So, we do not have to worry about that. We have 16 samples, and this one is 16 variable levels: x, ok. So, this is, ok, this is what we have defined now for this set. Now that we have defined this set, we can now run RUBg.

As you can see, RUVg requires new sake expression set data, right? Here we can see this in this example. It requires new sake expression set data, which we have defined as the list of spikes and the number of factors, ok? Of course, there are other attributes, but these are actually required. Even I mean these are mandatory, right? You have to mention how many factors you are looking at, which are the spike genes, and what the data is that you are looking at, ok?

And these should be enough to run your RUVg method, ok? So, that is what we are going to do, right? We are running; we are going to run this RUVg method now, ok? And let us go to the terminal, ok?

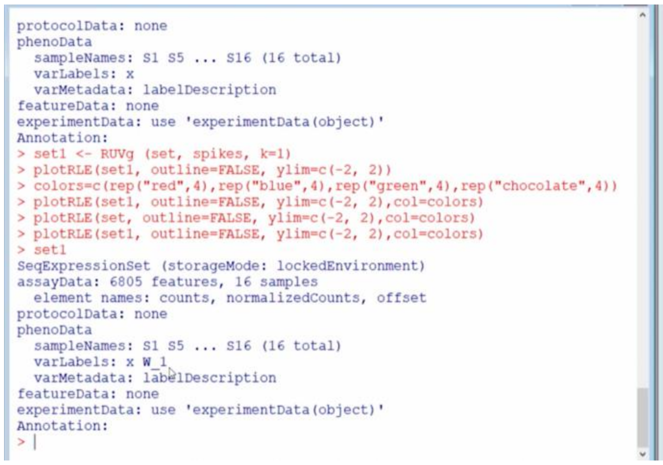


```
> set1 <- RUVg (set, spikes, k=1)
```

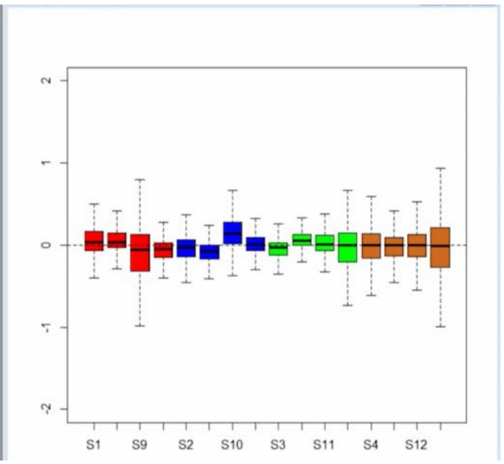
So, set 1, ok. So, it has run this RUVg method now, ok, and has got this set 1, ok. So, what we want to do is see whether set 1 is slightly different from set 2 at all. Is there any difference in the data that has been obtained right after this RUVg method? So, this is something we will do first using this plotRLE method, ok? So, plotRLE will tell us, but before we do that, I will define this color set, ok?

So, this is simply saying what the colors of these boxes are in the RLE plot, ok? So, if maybe I can just run this RLE plot with set 1, this is the data that we are running. Then we do not want to outline a limit, right? That is the limit of the y axis that is given the range, ok? We can simply run this, and then we will see, right?

You can see this will be in gray, or this whole box plot is shown in grey. We have these samples here. So, it would be nice if we could define some colors for these groups, right? So, if we can group them by color, ok. So, to do that, we will define these colors as variables, and we are using 4 colors.



```
protocolData: none
phenoData
  sampleNames: S1 S5 ... S16 (16 total)
  varLabels: x
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
> set1 <- RUVg (set, spikes, k=1)
> plotRLE(set1, outline=FALSE, ylim=c(-2, 2))
> colors=c(rep("red",4),rep("blue",4),rep("green",4),rep("chocolate",4))
> plotRLE(set1, outline=FALSE, ylim=c(-2, 2),col=colors)
> plotRLE(set, outline=FALSE, ylim=c(-2, 2),col=colors)
> plotRLE(set1, outline=FALSE, ylim=c(-2, 2),col=colors)
> set1
SeqExpressionSet (storageMode: lockedEnvironment)
assayData: 6805 features, 16 samples
  element names: counts, normalizedCounts, offset
protocolData: none
phenoData
  sampleNames: S1 S5 ... S16 (16 total)
  varLabels: x W_1
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
> |
```



You can choose anything you like. So, this is red, blue, green, and chocolate, ok? So, red means you have 4 reds, right? So, these are groups 1. So, if you see the order, we have 4 group 1 samples. So, this would be 4 reds. Then you have 4 group 2, then 4 blue, then 4 group 3, 4 green, right, and then 4 group 4, 4 chocolate colors.

So, you can define any color you like. I am just defining them, ok? And then we run this plot early again, ok, but this time color with our colors that we have just defined, ok. And you can see this now; these samples are colored, ok? So, the red one is for group 1, the blue is for group 2, right, and green is for group 3, and the fourth one is for group 4. Now, the question is, is there any difference with the original data? So, let us check that, ok?

Let us run this with set 1 instead of set 1. Let us run this with set, right? Set is the original data that we created. This is a new set of expressions that we created. So, let us run this with that new set of data, right, and see if we can see some difference, if we can observe some differences, right. So, here is the original data, the raw count data, ok, and this is the early plot.

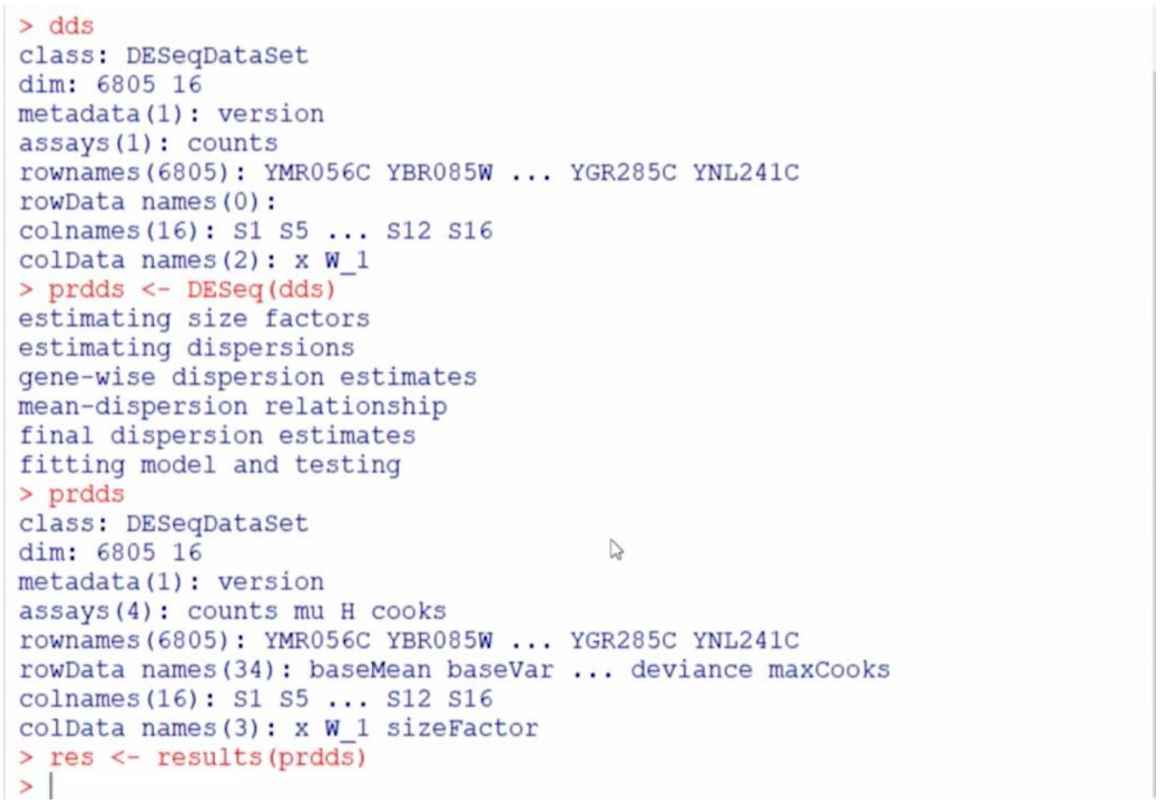
That is why the y axis is transformed. And then, if we plot set 1, again, you can see that the difference between the raw data and the normalized data is okay. So, now we have done this. We have done the normalization, ok? So, it is complete, and in the next step, we can do DEseq2, ok?

So, that part you are familiar with, right? We have done this before. So, we have to load the DEseq2 library. Then we have to create this DESeq data set from the matrix. This is again a command that we have to use because we need this DESeq data set for running DEseq2. The count data now would come from this set 1 matrix, right?

So, not the set matrix, but the set 1 matrix. This is the normalized count data, right after the RUVg normalization, ok? So, this will come, and then we have the column data that will also come from set 1 and the design part, ok? This is very important here, ok? So, in this design part, if you look at this p data set 1, we can write set 1, ok? So, when you see the normalized data here, ok, so set 1, you see there are two variable levels, right, x and w 1, ok.

So, w 1 is the factor, right? This factor has been estimated by RUVg, ok? So, you remember, right, how many factors you want to estimate from the control genes? So, this is the factor. If you actually estimate two factors, you will get w 1, w 2, three factors, and so on, right? So, since you have estimated this factor, this should also be part of your design, ok? And x should be part of your design because x determines the groupings, i.e., which sample belongs to which group.

So, we will have these two variable variables, right, and this will be part of the design in DEseq2, ok? So, let us clear this window now and let us first load this library, DEseq2. Sorry, yeah. So, we need to remember the capital letters and the small letters. Also, I will close this. Okay, alright. So, now we want to create this DESeq data set, right, from the matrix after this count normalization, ok?



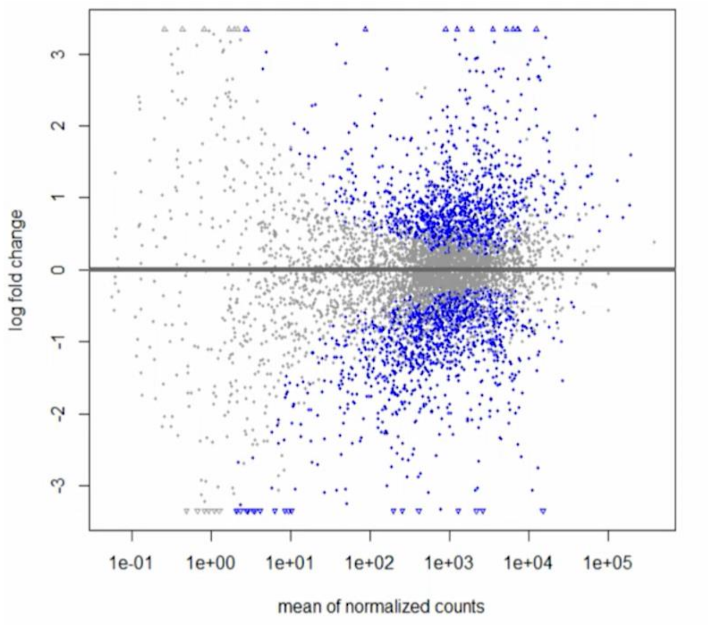
```
> dds
class: DESeqDataSet
dim: 6805 16
metadata(1): version
assays(1): counts
rownames(6805): YMR056C YBR085W ... YGR285C YNL241C
rowData names(0):
colnames(16): S1 S5 ... S12 S16
colData names(2): x W_1
> prdds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
> prdds
class: DESeqDataSet
dim: 6805 16
metadata(1): version
assays(4): counts mu H cooks
rownames(6805): YMR056C YBR085W ... YGR285C YNL241C
rowData names(34): baseMean baseVar ... deviance maxCooks
colnames(16): S1 S5 ... S12 S16
colData names(3): x W_1 sizeFactor
> res <- results(prdds)
> |
```

The difference is that the counts will come from normalized data and the design will be different, ok? So, the design will consider both of these factors, right? So, the x is the group, right? That will define the groups, and w 1 has been estimated by the RUVg method, ok? So, this is the command for, ok? So, we have now created this data set, the D set data set. And as you can see as before, we have this DEseq data set class, whose dimensions are given and fixed in samples 600, 800 genes.

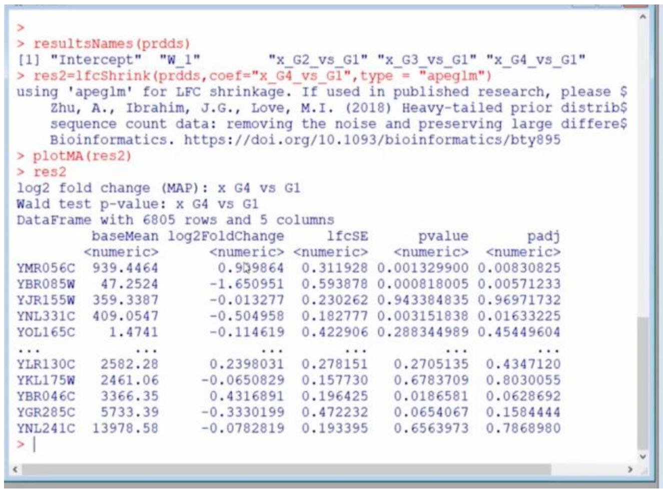
Then you have these row names that are given and the column names; these are the sample names and then we have these two factors that are also given. So, now it is easy, right? Once you have run this, once you have created this, right, so we can now run DEseq, ok, on this. It will again give you the steps, right? So, we have discussed these steps: estimate size factors, then dispersions. Right, this alpha is in the theory class.

If you remember, then you have these gene-wise dispersion estimates, ok? So, those would be the genes running for all. Then you have the mean dispersion relationship, which is used for dispersion shrinkage, and that will give you the final dispersion estimate. Once you have this dispersion estimate, you can fit this model into this negative binomial distribution and then you can do the test. So, the testing is the wild test, ok?

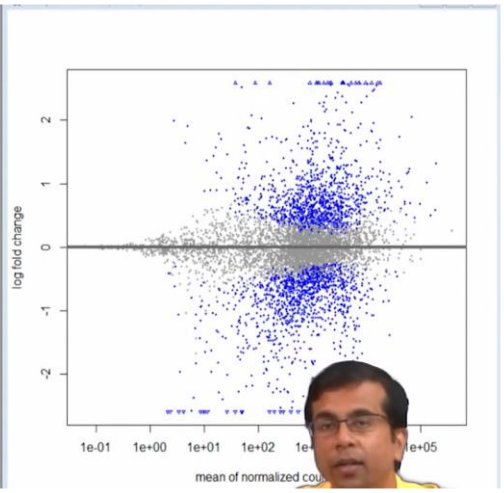
So, here it is, right? Again, the size factor is now determined for the samples, ok? So, the simple part now is to look at the results, ok, and we create these results by this command, and we can simply say this. So, we get this base mean, log pole change, log statistic, p value, and p adjusted, ok? Now, as before, you can go ahead and do the same analysis. So, what I will do is simply plot this now, ok, and this M-A plot will give you this log fold change versus mean counts, right, and we got something very similar in the last class if you go back, ok.



There would be some changes, of course, but you can see this looks pretty similar. The only thing that we can probably do to actually make it better is shrink the LFC. So, you remember this again from our last class? I am skipping some of these steps, perhaps because we have done this before. So, we will load the library app. We will look at this data, and then we will do the LFC shrinkage, and we will again plot, right, and we will see, right, what are the results that we get from here, ok?



```
>
> resultsNames(prdds)
[1] "Intercept"  "W_1"        "x_G2_vs_G1" "x_G3_vs_G1" "x_G4_vs_G1"
> res2=lfcShrink(prdds,coef="x_G4_vs_G1",type = "apeglm")
using 'apeglm' for LFC shrinkage. If used in published research, please $
    Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distrib$
    sequence count data: removing the noise and preserving large differe$
    Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895
> plotMA(res2)
> res2
log2 fold change (MAP): x G4 vs G1
Wald test p-value: x G4 vs G1
DataFrame with 6805 rows and 5 columns
         baseMean log2FoldChange     lfcSE      pvalue       padj
        <numeric>      <numeric> <numeric>   <numeric>  <numeric>
YMR056C  939.4464       0.9[illegible]9864  0.311928 0.001329900 0.00830825
YBR085W   47.2524      -1.650951  0.593878 0.000818005 0.00571233
YJR155W  359.3387      -0.013277  0.230262 0.943384835 0.96971732
YNL331C  409.0547      -0.504958  0.182777 0.003151838 0.01633225
YOL165C    1.4741      -0.114619  0.422906 0.288344989 0.45449604
...           ...            ...       ...         ...        ...
YLR130C   2582.28      0.2398031  0.278151   0.2705135  0.4347120
YKL175W   2461.06     -0.0650829  0.157730   0.6783709  0.8030055
YBR046C   3366.35      0.4316891  0.196425   0.0186581  0.0628692
YGR285C   5733.39     -0.3330199  0.472232   0.0654067  0.1584444
YNL241C  13978.58     -0.0782819  0.193395   0.6563973  0.7868980
> |
```



So, we are loading this library, APEGLM, and once it is done, we can, ok? So, it is telling us that this design thing intercepts w 1 x g 2 versus g 1 x g 4 versus g 1. So, this is not a bin anymore, right?

So, that was in the earlier class, right? So, we have seen. So, this will not be a bin. So, we will use this command to generate these results after LFC shrinkage. The coefficient would be x g 4 versus g 1, right, and the type would be apeglm. We can use other methods as well, and once this is done, we can plot MA and see if that looks okay.

So, this is done. We use the plot MA command, and we get better results, ok? So, one thing you probably have to do now is actually compare, right, how this comes, going back to the earlier DEseq2 analysis, and see how these results look. There would be some differences. They look very similar, but there would be some differences, especially when you export the data, and if you can look at this data here, you will see there is probably some difference. So, I can plot this. So, if I remember correctly, right, so for this gene, right, in the earlier analysis, it was a slightly different value for lock for change, and this has changed.

This will change because we are using this correction or normalization using the RUVg method. The rest of the things you can then filter, right, and identify the list of genes. You can export them to a file, right, in a text file tab-separated file, and you can work with those files, ok? So, this completes our differential expression analysis, and what we want to do is look beyond this MA plot. We want to look at some visualizations, right? For example, the volcano plot or the heat map, and we also want to look at the gene set enrichment analysis, right?

So, the functional enrichment analysis we have talked about So, very briefly, I want to show you how we can do that. There are different tools, but we will use one tool and complete that analysis, but that is for the next class. Thank you very much.