An Introduction to docker4seq package and 4SeqGUI

Raffaele A Calogero

Introduction

The docker4seq package was developed to facilitate the use of computing demanding applications in the field of NGS data analysis.

The docker4seq package uses docker containers that embed demanding computing tasks (e.g. short reads mapping) into isolated containers.

This approach provides multiple advantages:

- user does not need to install all the software on its local server;
- results generated by different containers can be organized in pipelines;
- reproducible research is guarantee by the possibility of sharing the docker images used for the analysis.

Requirements

The minimal hardware requirements are a 4 cores 64 bits linux computer, 32 Gb RAM, one SSD 250GB, with a folder with read/write permission for any users (chmod 777), and docker installed.

Setup

docker4seq and its graphical interface (optional) **4SeqGUI** can fit ideally in the NUC6I7KYK, Intel mini-computer equipped with Kingston Technology HyperX Impact 32GB Kit (2x16GB), 2133MHz DDR4 CL13 260-Pin SODIMM and Samsung 850 EVO - 250GB - M.2 SATA III Internal SSD.

MANDATORY: The first time *docker4seq* is installed the **downloadContainers** function needs to be executed to download, in the local repository, the docker images that are needed by *docker4seq*.

```
library(docker4seq)
downloadContainers(group="docker")
```

Dockers containers

At the present time all functions requiring some sort of calculation are embedded in the following docker images:

- docker.io/repbioinfo/demultiplexing.2017.01 used by demultiplexing
- docker.io/repbioinfo/annotate.2017.01 used by rnaseqCounts, rsemanno
- docker.io/repbioinfo/bwa.2017.01 used by bwaIndexUcsc, bwa, wrapperPdx
- docker.io/repbioinfo/chipseq.2017.01 used by chipseqCounts, chipseq
- docker.io/repbioinfo/r332.2017.01 used by experimentPower, sampleSize, wrapperDeseq2
- docker.io/repbioinfo/mirnaseq.2017.01 used by mirnaCounts
- docker.io/repbioinfo/rsemstar.2017.01 used by rnaseqCounts, rsemstarIndex, rsemstarUscsIndex
- docker.io/repbioinfo/skewer.2017.01 used by skewer, rnaseqCounts, wrapperPdx

- docker.io/repbioinfo/xenome.2017.01 used by xemone, xenomeIndex, wrapperPdx

docker container nomenclature

In case of updates required to solve bugs, which do not affect the calculation docker.io/rcaloger/XXXXX.YYYY.ZZ the fiels ZZ will be updated.

In case of updates which affect the calculation, e.g. new release of Bioconductor libraries, the field YYYY will be updated. Previous versions will be maintained to guarantee the reproducibility of any previous analisys.

Reproducibility

The file **containers.txt**, which indicates the Docker images available in the local release of docker4seq is saved within any folder generated with docker4seq functions.

In case, user would like to download a set of dockers images different from those provided as part of the package, then these images must be specified in a file with the following format **docker.repository/user/docker.name**, which has to be passed to downloadContainers function:

```
downloadContainers(group="docker", containers.file="my_containers.txt")
#an example of the my_containers.txt file content
docker.io/rcaloger/bwa.2017.01
docker.io/rcaloger/chipseq.2017.01
docker.io/rcaloger/r340.2017.01
```

Available workflows

At the present time are available the following workflows:

- **mRNAseq**, which allows:
  - adapter trimming with skewer
  - mapping with STAR
  - counting genes and isoforms with RSEM
  - as option to STAR/RSEM it is possible to use **SALMON**, which is does reference-free transcripts quantification.
  - ENSEMBL gene annotation.
  - organizing the output of RSEM in tables to be used for differential expression analysis
  - visualizing experiment data with PCA
  - evaluating experiment power and sample size
  - detecting differentially expressed genes/isoforms
  - subsetting counts/FPKM and TPM table to have only differentially expressed genes, suitable for heatmaps generation.
- **miRNAseq**, which executes the workflow described in Cordero et al. PLoS One. 2012;7(2):e31630, embedding the following steps:
  - trimming adapters with cutadapt
  - miRNAs mapping on mirbase hairpins using SHRiMP
  - quantification of mature miRNAs.
  - visualizing experiment data with PCA
  - evaluating experiment power and sample size
  - detecting differentially expressed miRNAs
- **ChIPseq**, which allows:
  - adapter trimming with skewer
  - mapping with BWA

  - peak calling using either MACS v 1.4 or SICER v 1.1
  - associating peaks to the nearest gene, UCSC annotation
  - full annotation of the nearest gene
- **PDX Exomeseq**, which allows:
  - mouse sequence removal with xenome
  - adapter trimming with skewer
  - mapping with BWA
  - annotating the output of oncoSNP

The most expensive computing steps of the analyses are embedded in the following docker4seq functions: **rnaseqCounts**, **mirnaCounts**, **chipseqCounts**. These functions are also the only having RAM and computing power requirements not usually available in consumer computers. Hereafter it is shown the time required to run the above three functions increasing the number of sequenced reads.

testSeqbox

In *docker4seq* is now present the function *testSeqbox*, which allows to evaluate if the software required for docker4seq functionalities is properly installed. Results of the tests are saved in testSeqBox.out file.

rnaseqCounts performances

Counts generation from fastq files is the most time consuming step in RNAseq data analysis and it is usually calculated using high-end servers. We compare the behabiour of **rnaseqCounts** on SeqBox and on a high-end server:

```
+ SeqBox: NUC6I7KYK CPU i7-6770HQ 3.5 GHz (1 core, 8 threads), 32 Gb RAM, HD 250 Gb SSD
+ SGI UV200 server: CPU E5-4650 v2 2.40GHz (8 cores, 160 threads), 1 Tb RAM, RAID 6, 100 Tb SATA
```

We run respectively 26, 52, 78, and 105 million reads using different number of threads, values shown in parenthesis in *Figure 1*. It is notable that SeqBox, mapping in 5 hours more than 100 milion reads, it is able to handle in 20 hours the throughput of the Illumina benchtop sequencer NextSeq 500, which produces up to 400 milion reads in a run of 30 hours.

mirnaCounts performances

We run resepctively 3, 6, 12, and 24 miRNA samples in parallel using **mirnaCounts**, with different number of threads, values shown in parenthesis in *Figure 2*.

chipseqCounts performances

We run respectively 37, 70, 111, and 149 million reads using different number of threads, values shown in parenthesis in *Figure 3*.

From the point of view of parallelization the **rnaseqCounts** is the one that embeds the most computing demanding tools: i) mapping with STAR and ii) quantifying transcripts with RSEM. Both these tools were design to take advantage of multiple cores hardware architecture and they also require massive I/O. On the basis of the results shown in *Figure 1* parallelization does not improve very much the overall performances, even if it can mitigate the gap w.r.t. SeqBox due to the poor I/O performance of the SATA disk array. On the other side the presence of a SSD with very high I/O performance can remedy the limited amount of cores of SeqBox.

In the case of **mirnaCounts** and **chipseqCounts** the parallelization is very little and it is only available for the reads mapping procedure. Moreover, both functions have a massive I/O. The reduced parallelization

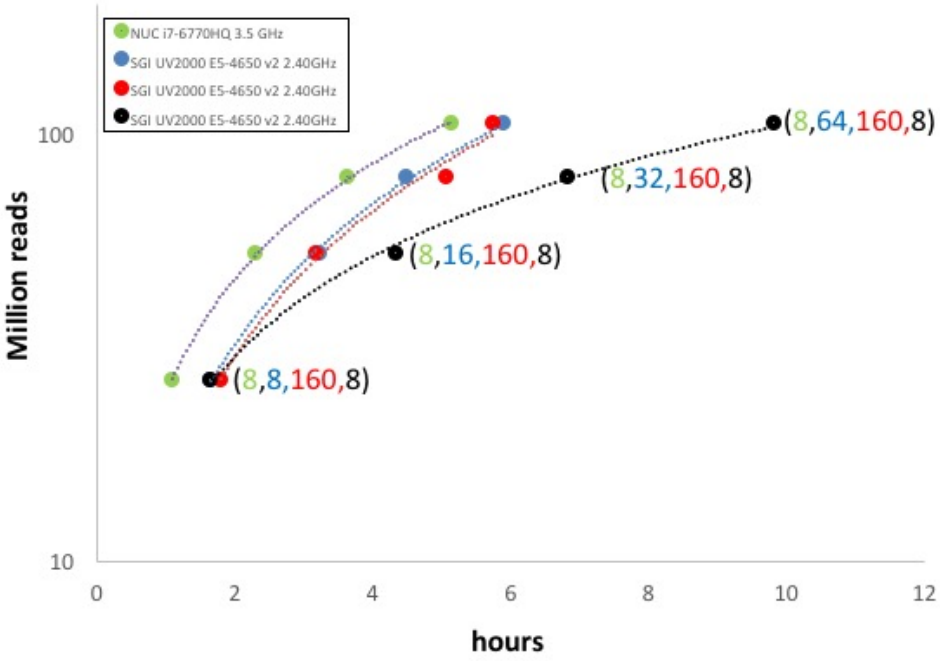

Figure 1: rnaseqCounts overall performance

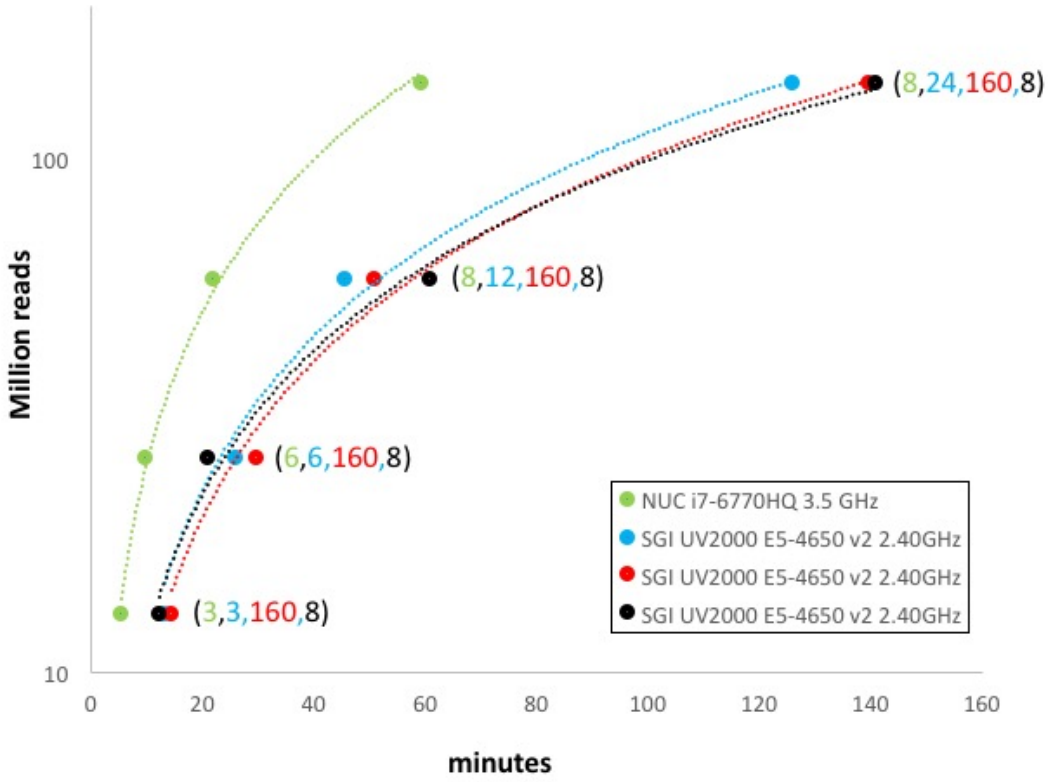

Figure 2: mirnaCounts overall performance

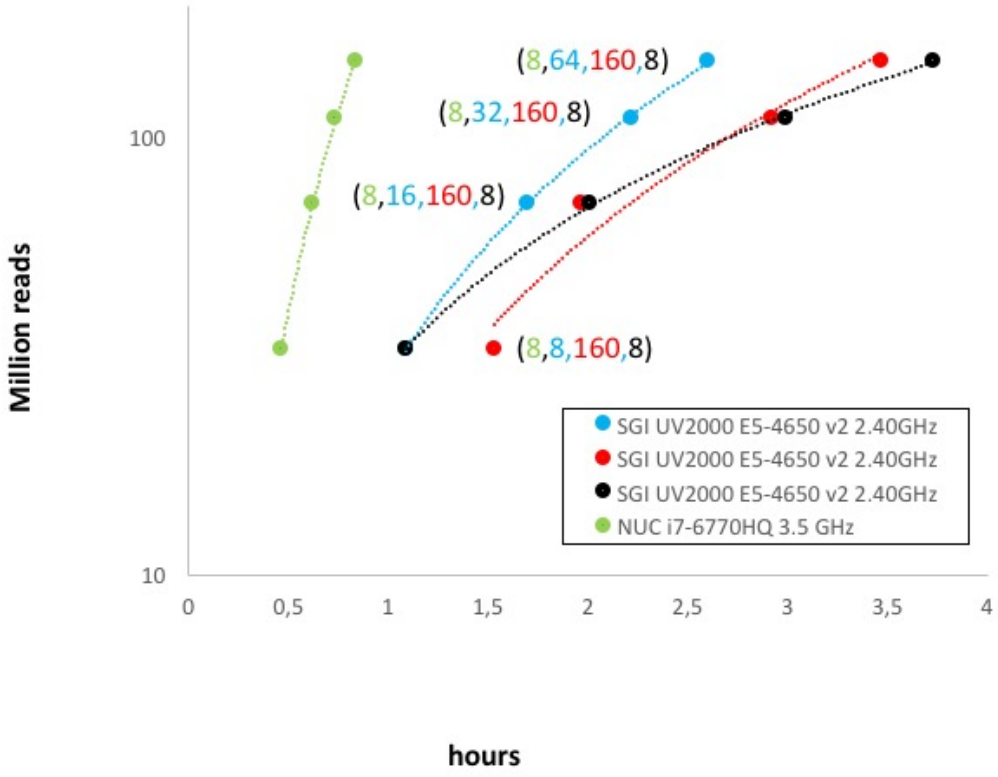

Figure 3: chipseqCounts overall performance

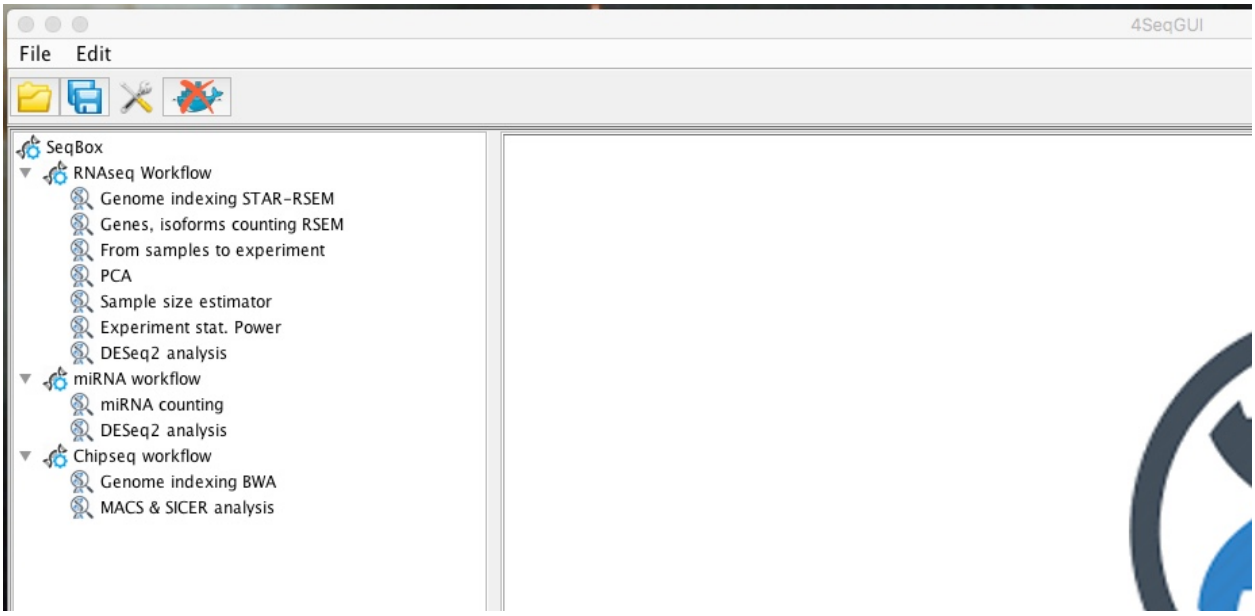

Figure 4: mRNAseq workflow

of these two analyses combined with the higher I/O throughput of the SSD with respect to the SATA array makes SeqBox extremely effective even with very high number of reads to be processed, *Figure 2 and 3*.

Test sets

A folder including a set do dataset to test each of the workflows available in docker4seq/4SeqGUI can be found **here**

RNAseq workflow: Howto

Demultiplexing

demultiplexing function is used to convert in fastq bcl files generated by an Illumina sequencer. The function requires that in the bcl folder the SampleSheet.csv is present. An example fo SampleSheet for a pair-end run is present in docker4seq examples folder. The function require that the full path to the bcl file folder is provided, **data.folder**, the scratch folder where temporary analysis is run and the number of cores that will be used by the program.

```
demultiplexing(group="docker",
      data.folder="/home/calogero/Documents/data/lollini/3a_run/170712_NB501050_0097_AH3FGNBGX3",
      scratch.folder="/data/scratch", threads=24)
```

The function will return, in the folder containing the bcl files folder, e.g. /home/calogero/Documents/data/lollini/3a_run/, the fastq files generated by the analysis.

This function is not implemented in **4SeqGUI** because this step is generally done by a core lab. Thus only a limited group of users require the use of this function.

The **mRNAseq workflow**, that can be handled using **4SeqGUI** graphical interface (linux/MAC), starts from the availability of fastq files.

Sample quantification is made of these steps:

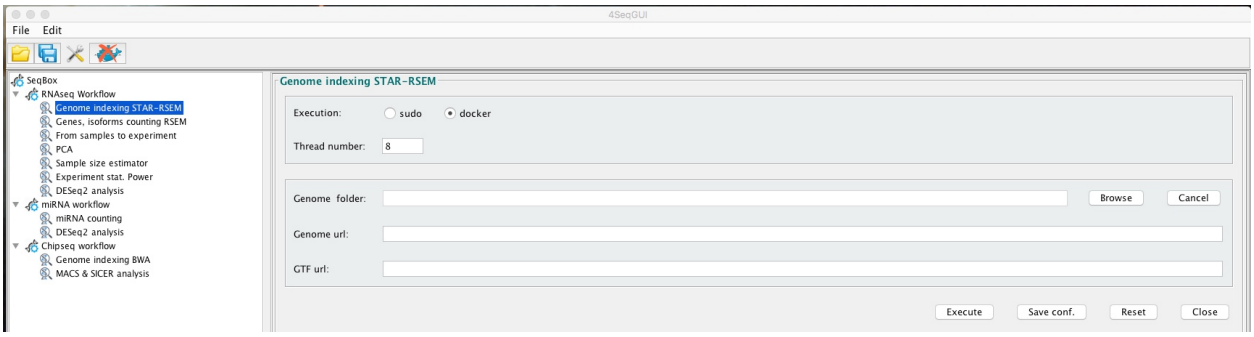

Figure 5: Creating a STAR genome index

- Creating a genome index for STAR (see end of this paragraph)
- Running removing sequencing adapters
- Mapping reads to the reference genome
- Quantify gene and transcript expression level
- Annotating genes.

All the parameters can be setup using 4SeqGUI

Creating a STAR index file for mRNAseq

The index can be easily created using the graphical interface:

A detailed description of the parameters is given below.

Creating a STAR index file by line command

```
rsemstarIndex(group="docker",genome.folder="/data/scratch/hg38star",
ensembl.urlgenome="ftp://ftp.ensembl.org/pub/release-87/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.toplevel.fa.gz",
ensembl.urlgtf="ftp://ftp.ensembl.org/pub/release-87/gtf/homo_sapiens/Homo_sapiens.GRCh38.87.gtf.gz")
```

In brief, **rsemstarIndex** uses ENSEMBL genomic data. User has to provide the URL (**ensembl.urlgenome**) for the file XXXXX_dna.toplevel.fa.gz related to the organism of interest, the URL (**ensembl.urlgtf**) for the annotation GTF XXX.gtf.gz and the path to the folder where the index will be generated (**genome.folder**). The parameter **threads** indicate the number of cores dedicated to this task.

Precompiled index folders are available:

- hg38star
- mm10star

Quantifying genes/isoforms

A detailed description of the parameters is given below.

Sample quantification by line command

The sample quantification can be also executed using R and it is completely embedded in a single function:

```
#test example
system("wget http://130.192.119.59/public/test.mrnaCounts.zip")
unzip("test.mrnaCounts.zip")
setwd("./test.mrnaCounts")
library(docker4seq)
rnaseqCounts(group="docker",fastq.folder=getwd(), scratch.folder=getwd(),
adapter5="AGATCGGAAGAGCACACGTCTGAACTCCAGTCA",
adapter3="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT",
```

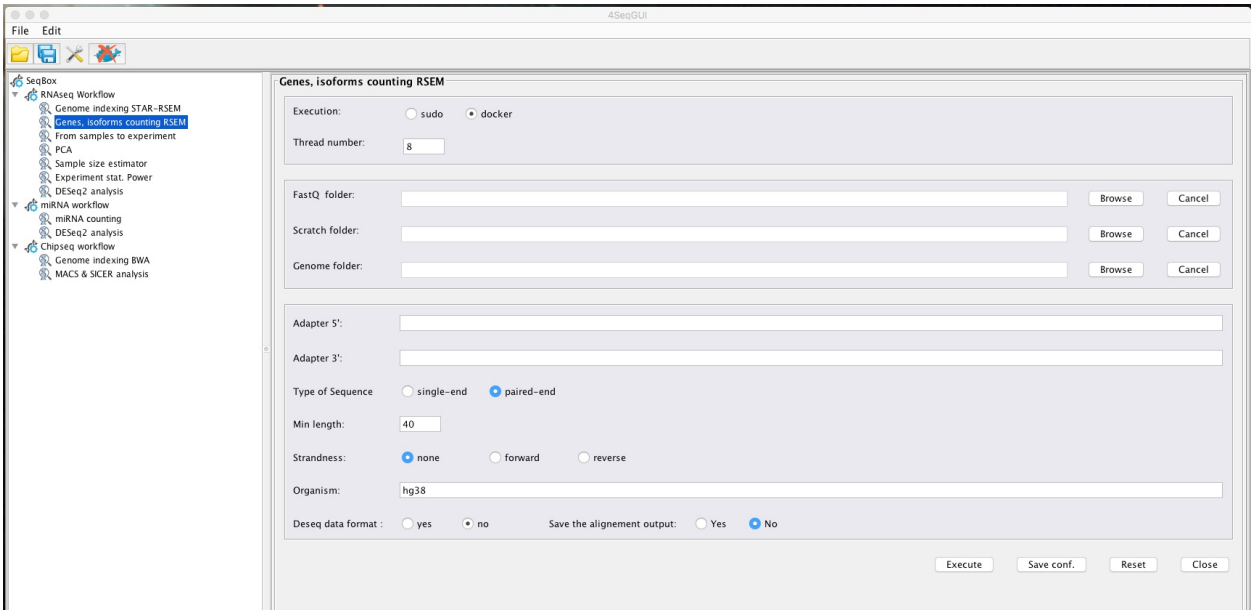

Figure 6: Gene, Isoform counting

```
seq.type="se", threads=8,  min.length=40,
genome.folder="/data/scratch/mm10star", strandness="none", save.bam=FALSE,
org="mm10", annotation.type="gtfENSEMBL")
```

User needs to create the **fastq.folder**, where the fastq.gz file(s) for the sample under analysis are located. The **scratch.folder** is the location where temporary data are created. The results will be then saved in the **fastq.folder**.

User needs to provide also the sequence of the sequencing adapters, **adapter5** and **adapter3** parameters. In case Illumina platform the adapters sequences can be easily recovered **here**.

seq.type indicates if single-end (se) or pair-end (pe) data are provided, **threads** indicates the max number of cores used by *skewer* and *STAR*, all the other steps are done on a single core.

The **min.length** refers to the minimal length that a reads should have after adapters trimming. Since today the average read length for a RNAseq experiment is 50 or 75 nts would be better to bring to 40 nts the min.length parameter to increase the precision in assigning the correct position on the genome.

The **genome.folder** parameter refers to the location of the genomic index generated by STAR using the *docker4seq* function **rsemstarIndex**, see above paragraph.

strandness, is a parameter referring to the kit used for the library prep. If the kit does not provide strand information it is set to "none", if provides strand information is set to "forward" for Illumina stranded kit and it set to "reverse" for Illumina ACCESS kit. **save.bam** set to TRUE indicates that genomic bam file and transcriotomic bam files are also saved at the end of the analysis. **annotation.type** refers to the type of available gene-level annotation. At the present time is only available ENSEMBL annotation defined by the gtf downloaded during the creation of the indexed genome files, see paragraph *at the end*Creating a STAR index file for mRNAseq*.

Salmon, reference free alignment

Recently Zhang and coworkers (BMC Genomics 2017, 18,583) compared, at transcript level, alignment-dependent tools (Salmon_aln, eXpress, RSEM and TIGAR2) and aligner-free methods (Salmon, Kallisto Sailfish). In their paper, STAR was used as mapping tool for all alignment-dependent tools. In terms of isoform quantification, the authors indicated that there is strong concordance among quantification results from RSEM, Salmon, Salmon_aln, Kallisto and Sailfish ($R2 > 0.89$), suggesting that the impact of mappers on isoform quantification is small. Furthermore, the paper of Teng and coworkers (Genome Biology 2016, 17,74) reported that,in term of gene-level quantification, differences between alignment-dependent tools and aligner-free methods are shrinking with respect to transcripts level analysis. On the basis of the above papers it seems that from the quantification point of view the difference between alignment free and alignment-dependent tools is very limited. However, aligner-free methods have low memory requirements and we added Salmon in docker4seq.

Creating a Salmon quasi-reference file for mRNAseq

The quasi-reference can be created using cDNA fasta files available at ENSEMBL. The corresponding genomics GTF is required for the gene level annotation.

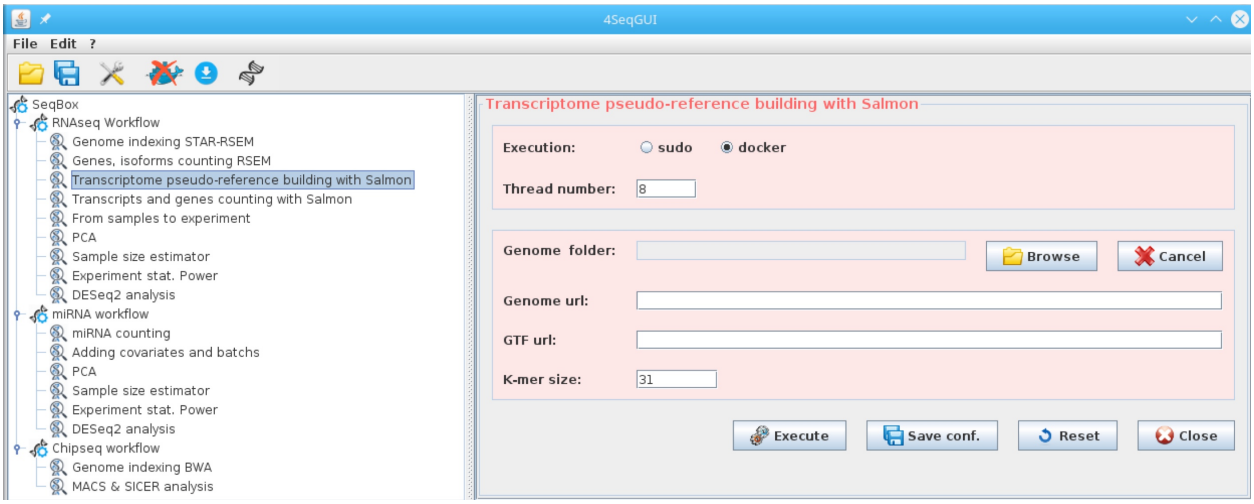

Figure 7: Salmon reference

```
#running salmonIndex human
salmonIndex(group="docker", index.folder=getwd(),
       ensembl.urltranscriptome="ftp://ftp.ensembl.org/pub/release-90/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh38.cdna.all.fa.gz
       ensembl.urlgtf="ftp://ftp.ensembl.org/pub/release-90/gtf/homo_sapiens/Homo_sapiens.GRCh38.90.gtf.gz",
       k=31)
```

Quantifying isoforms and genes with Salmon

Salmon quantification function *wrapperSalmon* has the same structure of *rnaseqCounts*. It performs adapters trimming, transcripts quantification, genes quantification and annotation. The output of wrapperSalmon is identical to the output of rnaseqCounts for what concern the files **isoforms.results** and **gtf_annotated_genes.results**, which can be used by *samples2experiment* to generate the tables for differential expression analysis.

```
system("wget http://130.192.119.59/public/test_R1.fastq.gz")
system("wget http://130.192.119.59/public/test_R2.fastq.gz")

#running salmonCounts
wrapperSalmon(group="docker", scratch.folder="/data/scratch/",
        fastq.folder=getwd(), index.folder="/data/genome/salmonhg38/",
        threads=8, seq.type="pe", adapter5="AGATCGGAAGAGCACACGTCTGAACTCCAGTCA",
        adapter3="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT", min.length=40, strandness="none")
```

IMPORTANT: Salmon produces only isoforms-level counts, thus we quantified gene-level counts as described by RSEM:

- Gene counts are gived by the sum of the counts associated to all transcripts associated to the gene locus.
- Gene-length is given by the mean length of the transcripts having counts greater than 0 or by the mean of all the transcripts annotated in the gene locus in case all transcripts have counts equal to 0.

However, there are some differences between the information present in RSEM and Salmon annotation:

	RSEM	SALMON
Annotated transcripts	197898	164819
Annotated genes	57955	34912
Biotypes	43	24

The biotypes lacking in Salmon are 20, because the pseudo-reference is build using ENSEMBL cDNA fasta file:

processed_transcript, antisense, lincRNA, sense_intronic, sense_overlapping, 3prime_overlapping_ncRNA, bidirectional_promoter_lncRNA, snRNA, miRNA, misc_RNA, snoRNA, rRNA, ribozyme, TEC, scRNA, scaRNA, vaultRNA, sRNA, macro_lncRNA, non_coding.

Furthermore, the data generated by SALMON do not overlap perfectly to RSEM output.

Panel A indicates that although there is a limited correlation between SALMON and RSEM expected counts (black) at transcript-level, **R2=0.35**, some transcripts are detected with more counts by SALMON, see vertical spikes. SALMON TPMs are generally higher of those estimated in RSEM (Panel A, red). This behavior is still present at gene-level (Panel B) and the R2 for the expected counts is **0.56**. The correlation between SALMON and RSEM at counts level (transcripts/genes) is always better of that observed at FPKM or TPM level. It is notable that length estimation between SALMON and RSEM is well correlated only for transcripts/genes bigger than 100 nts (Panel C, D).

Sample quantification output files

The mRNAseq workflow produces the following output files:

```
+ XXXXX-trimmed.log, containing the information related to the adapters trimming
+ gtf_annotated_genes.results, the output of RSEM gene quantification with gene-level annotation
```

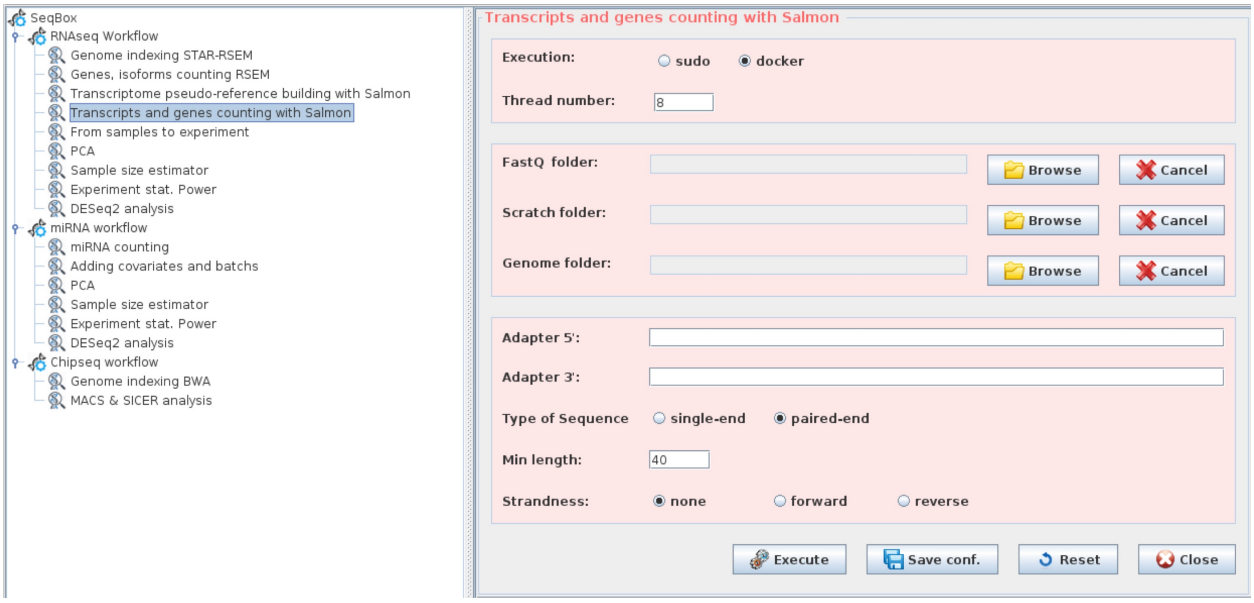

Figure 8: Salmon qunatification

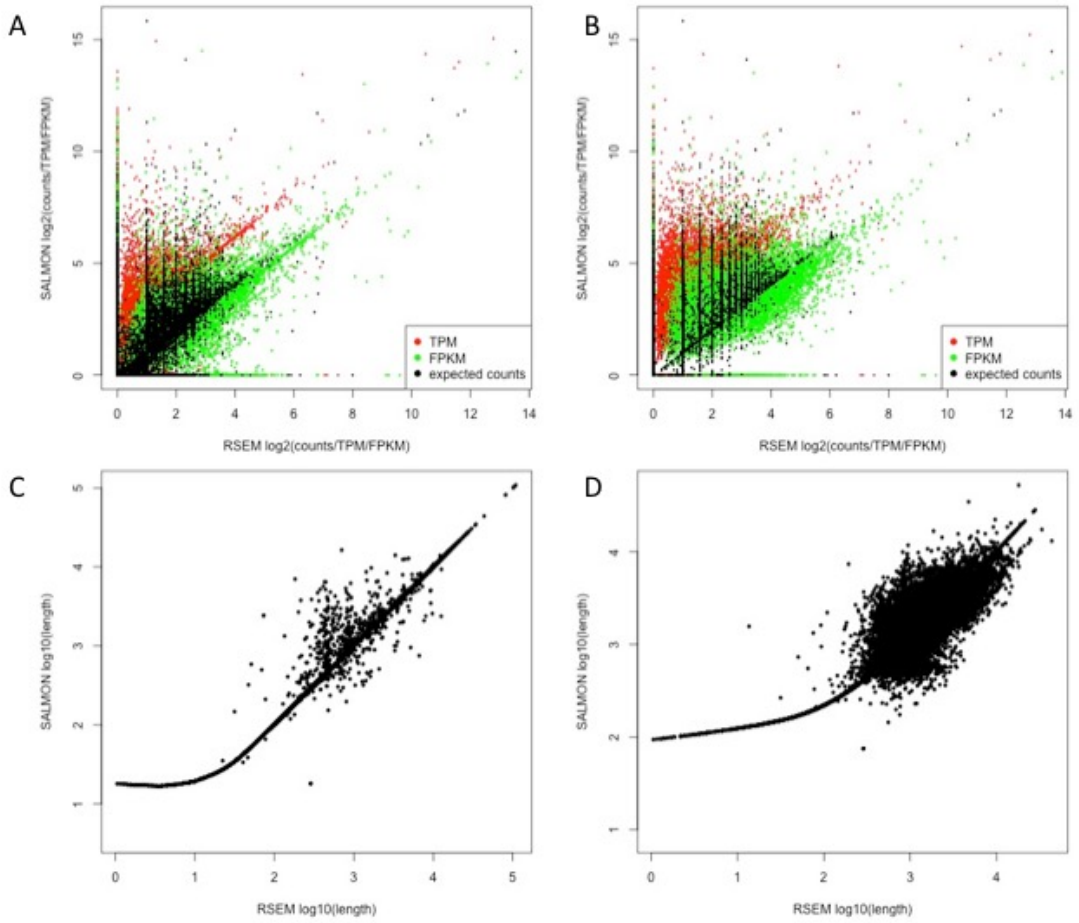

Figure 9: RSEM versus SALMON

annotation.gene_id	annotation.gene_biotype	annotation.gene_name	annotation.source	transcript_id.s.	length	effective_length	expected_count	TPM	FPKM
ENSMUSG00000000001	protein_coding	Gnai3	ensembl_havana	ENSMUST00000000001	3262	3213.06	67	48.66	36.48
ENSMUSG00000000003	protein_coding	Pbsn	ensembl_havana	ENSMUST00000000003,ENSMUST00000114041	799.5	750.56	0	0	0
ENSMUSG00000000028	protein_coding	Cdc45	ensembl_havana	ENSMUST00000000028,ENSMUST00000096990,ENSMUST00000115585	1874.36	1825.42	43	54.97	41.21
ENSMUSG00000000031	lincRNA	H19	ensembl_havana	ENSMUST00000132294,ENSMUST00000136359,ENSMUST00000140716,ENSMUST00000149974,ENSMUST00000152754	817	768.06	1	3.04	2.28
ENSMUSG00000000037	protein_coding	Scml2	ensembl_havana	ENSMUST00000019101,ENSMUST00000074802,ENSMUST00000077375,ENSMUST00000087090,ENSMUST00000101113,ENSMUST00000112345,ENSMUST00000124775	3297.14	3248.21	0	0	0
ENSMUSG00000000049	protein_coding	Apoh	ensembl_havana	ENSMUST00000000049,ENSMUST00000133383,ENSMUST00000146050,ENSMUST00000152958	665.5	616.56	0	0	0
ENSMUSG00000000056	protein_coding	Narf	ensembl_havana	ENSMUST00000103015,ENSMUST00000151088,ENSMUST00000154047	4395	4346.06	24	12.89	9.66
ENSMUSG00000000058	protein_coding	Cav2	ensembl_havana	ENSMUST00000000058,ENSMUST00000115459,ENSMUST00000115462	2733	2684.06	38	33.04	24.77

Figure 10: gtf_annotated_genes.results

	sa_Cov.1	sb_Cov.1	sc_Cov.1	sd_Cov.1	se_Cov.1	sf_Cov.1	sg_Cov.1	ra_Cov.2	rb_Cov.2	rc_Cov.2	rd_Cov.2	re_Cov.2	rf_Cov.2	rg_Cov.2
TSPAN6:ENSG00000000003	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TNMD:ENSG00000000005	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DPM1:ENSG00000000419	161	205	163	56	91	58	225	179	118	222	161	145	187	253

Figure 11: counts table with covariates

In case also a batch is present also this is added to the sample name through a further underscore, e.g. mysample1__Cov1__batch1, mysample2__Cov__batch2.

The addition of the covariates to the various samples can be done using the **4seqGUI** using the button: *From samples to experiment*.

From samples to experiments by line command

```
#test example
system("wget http://130.192.119.59/public/test.samples2experiment.zip")
unzip("test.samples2experiment.zip")
setwd("test.samples2experiment")
library(docker4seq)
sample2experiment(sample.folders=c("./e1g","./e2g","./e3g",
"./p1g", "./p2g", "./p3g"),
covariates=c("Cov.1","Cov.1","Cov.1","Cov.2","Cov.2","Cov.2"),
bio.type="protein_coding", output.prefix=".")
```

User needs to provide the paths of the samples, **sample.folder** parameter, a vector of the covariates, **covariates**, and the biotype(s) of interest, **bio.type** parameter. The parameter **output.prefix** refers to the path where the output will be created, as default this is the current R working folder.

From samples to experiments output files

The samples to experiments produces the following output files:

```
+ _counts.txt: gene-level raw counts table for differential expression analysis
+ _isoforms_counts.txt: isoform-level raw counts table for differential expression analysis
+ _isoforms_log2TPM.txt: isoform-level log2TPM for visualization purposes
+ _log2TPM.txt: gene-level log2TPM for visualization purposes
+ _isoforms_log2FPKM.txt: isoform-level log2FPKM for visualization purposes
+ _log2FPKM.txt: gene-level log2FPKM for visualization purposes
+ XXXXX.Rout: logs of the execution
```

Visualizing experiment data with PCA

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component accounts for as much of the variability in the data as possible, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. **4SeqGUI** provides an interface to the generation experiment samples PCA

The plot is saved in **pca.pdf** in the selected folder.

	sa_Cov.1_1	sb_Cov.1_2	sc_Cov.1_3	sd_Cov.1_4	se_Cov.1_5	sf_Cov.1_6	sg_Cov.1_7	ra_Cov.2_1	rb_Cov.2_2	rc_Cov.2_3	rd_Cov.2_4	re_Cov.2_5	rf_Cov.2_6	rg_Cov.2_7
TSPAN6:ENSG00000000003	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TNMD:ENSG00000000005	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DPM1:ENSG00000000419	161	205	163	56	91	58	225	179	118	222	161	145	187	253

Figure 12: counts table with covariates and batch

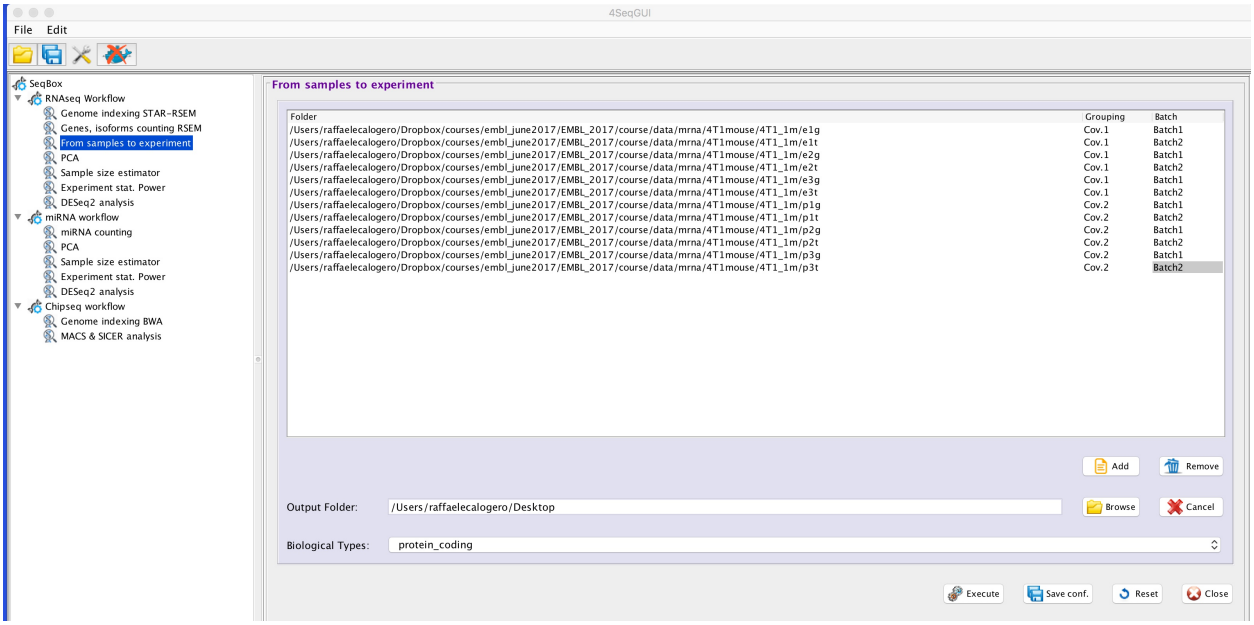

Figure 13: generating a table with covariates

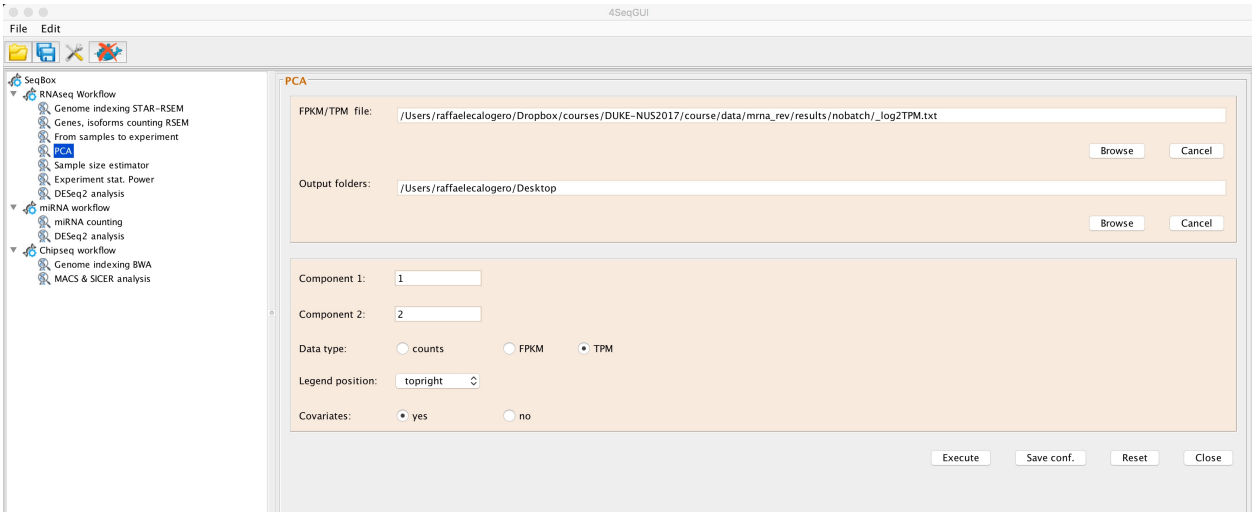

Figure 14: PCA

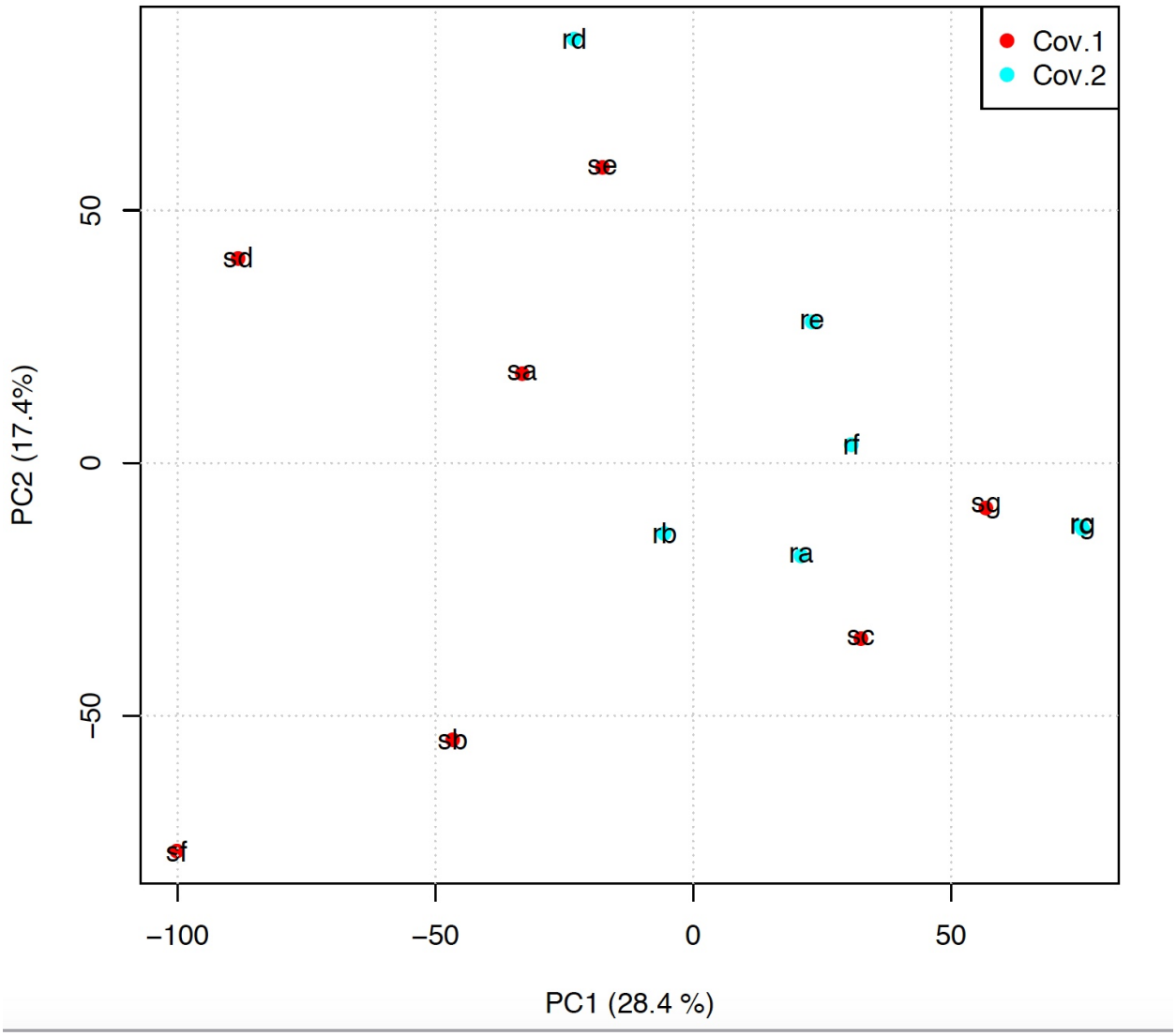

Figure 15: pca.pdf

PCA by line command

```
#test example
system("wget 130.192.119.59/public/test.analysis.zip")
unzip("test.analysis.zip")
setwd("test.analysis")
library(docker4seq)
pca(experiment.table="_log2FPKM.txt", type="FPKM", legend.position="topleft", covariatesInNames=FALSE, principal.components=c(1,
```

User needs to provide the paths of experiment table, **experiment.table** parameter, i.e. the file generated using the samples2experiment function. The **type** parameter indicates if FPKM, TPM or counts are used for the PCA generation. The parameter **legend.position** defines where to locate the covariates legend. The parameter **covariatesInNames** indicates if the header of the experiment table contains or not covariate information. The parameter **principal.components** indicates which principal components should be plotted. **output.folder** indicates where to save the pca.pdf file.

The values in parentesis on x and y axes are the amount of variance explained by each principal component.

IMPORTANT: The above analysis is suitable also for miRNAseq data

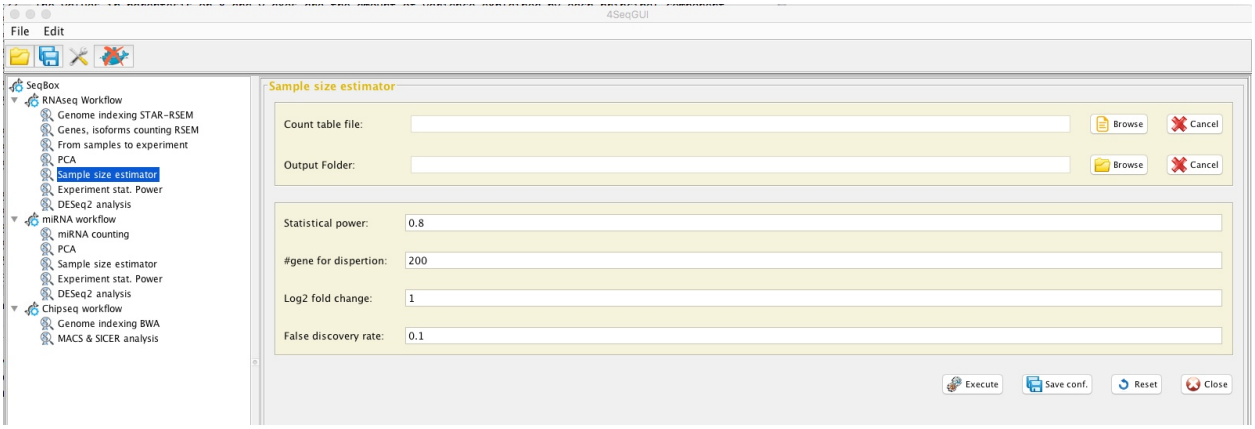

Figure 16: sample size estimation

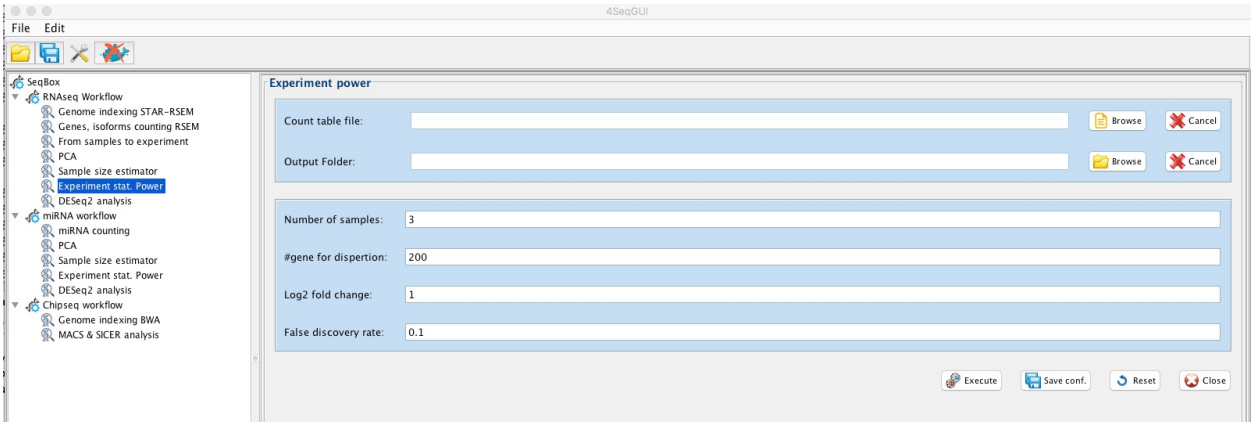

Figure 17: stat power estimation

Evaluating sample size and experiment power

Sample size estimation is an important issue in the design of RNA sequencing experiments. Furthermore, experiment power provides an indication of which is the fraction of differentially expressed genes that can be detected given a specific number of samples and differential expression detection thresholds. RnaSeqSampleSize Bioconductor package provides the possibility to calculate, from a pilot experiment, the statistical power and to define the optimal sample size. We have implemented wrapper functions to RnaSeqSampleSize sample size and experiment power estimation.

4SeqGUI provides an interface to sample size estimation and to statistical power estimation.

Sample size estimation by line command

```
#test example
system("wget 130.192.119.59/public/test.analysis.zip")
unzip("test.analysis.zip")
setwd("test.analysis")
library(docker4seq)
sampleSize(group="docker", filename="_counts.txt", power=0.80, FDR=0.1, genes4dispersion=200, log2fold.change=1)
```

The requested parameters are the path to the counts experiment table generated by **samples2experiment** function. The param **power** indicates the expected fraction of differentially expressed gene, e.g 0.80. **FDR** and **log2fold.change** are the two thresholds used to define the set of differentially expressed genes of interest.

The output file is **sample_size_evaluation.txt** is saved in the R working folder, below an example of the file content:

IMPORTANT: The above analysis is suitable also for miRNAseq data

Experiment statistical power estimation by line command

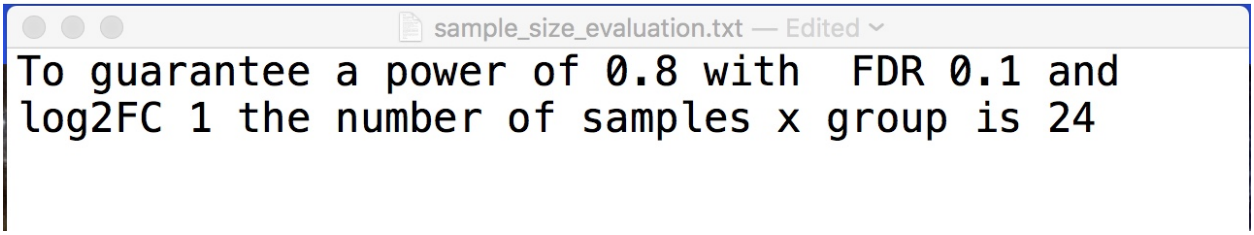

Figure 18: sample_size_evaluation.txt

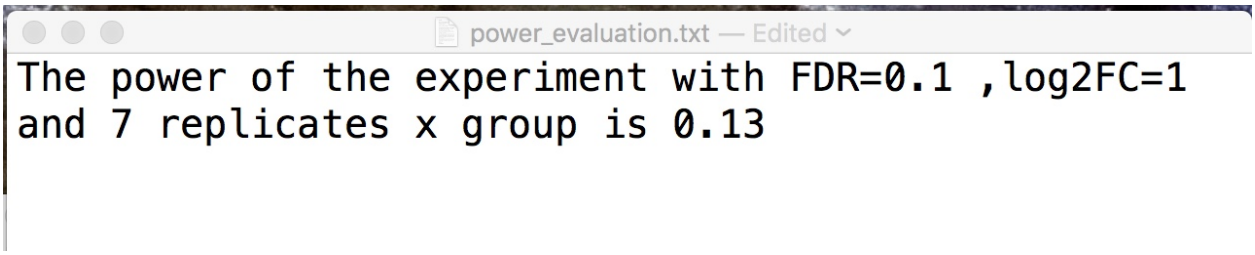

Figure 19: power_estimation.txt

```
#test example
system("wget 130.192.119.59/public/test.analysis.zip")
unzip("test.analysis.zip")
setwd("test.analysis")
library(docker4seq)
experimentPower(group="docker", filename="_counts.txt",replicatesXgroup=7, FDR=0.1, genes4dispersion=200, log2fold.change=1)
```

The requested parameters are the path to the counts experiment table generated by **samples2experiment** function. The param **replicatesXgroup** indicates the number of sample associated with each of the two covariates. **FDR** and **log2fold.change** are the two thresholds used to define the set of differentially expressed genes of interest. **genes4dispersion** indicates the number of genes used in the estimation of read counts and dispersion distribution.

The output file is **power_estimation.txt** is saved in the R working folder, below an example of the file content:

IMPORTANT: The above analysis is suitable also for miRNAseq data

Differential expression analysis with DESeq2

A basic task in the analysis of count data from RNA-seq is the detection of differentially expressed genes. **4SeqGUI** provides an interface to DESeq2 to simplify differential expression analysis.

The output files are:

DEfull.txt containing the full set of results generated by DESeq2

DEfiltered_log2fc_X_fdr_Y.Y.txt containing the set of differentially expressed genes passing the indicated thresholds

genes4david.txt a file containing only the gene symbols to be used as input for DAVID or ENRICHR

log2normalized_counts.txt, log2 library size normalized counts, calculated by DESeq2, that can be used for visualization purposes.

DESeq2 by line command

```
#test example
system("wget 130.192.119.59/public/test.analysis.zip")
unzip("test.analysis.zip")
setwd("test.analysis")
library(docker4seq)
wrapperDeseq2(output.folder=getwd(), group="docker",
      experiment.table="_counts.txt", log2fc=1, fdr=0.1,
      ref.covar="Cov.1", type="gene", batch=FALSE)
```

User has to provide experiment table, **experiment.table** param, i.e. the counts table generated with **samples2experiment** function, the thresholds for the differential expression analysis, **log2fc** and **fdr** params, the reference covariate, **ref.covar** param, i.e. the covariate that is used as reference for differential expression detection, the **type** param, which refers to the type of

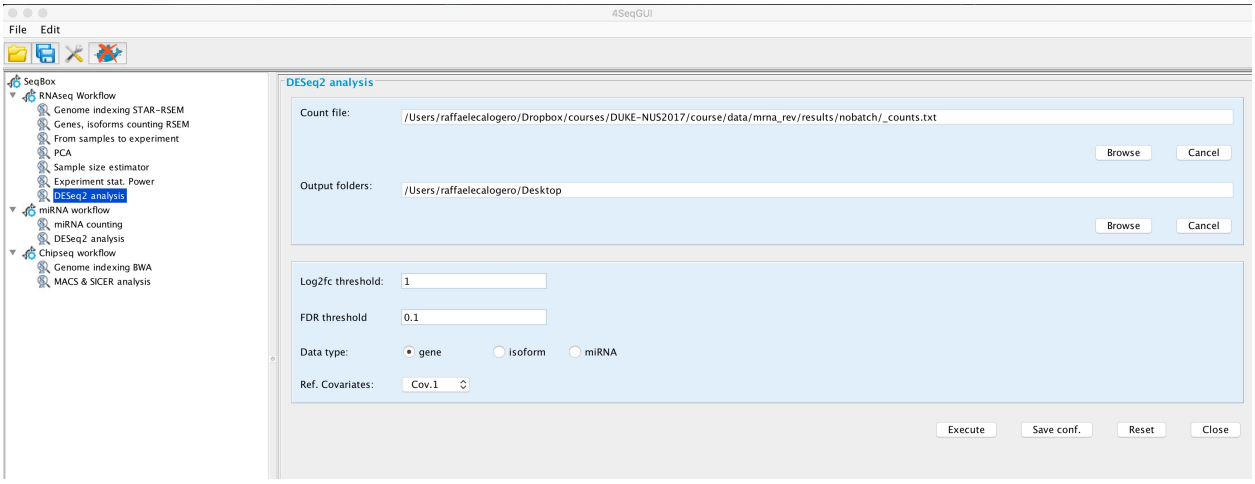

Figure 20: DESeq2

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
TSPAN6:ENSG00000000003	0	NA	NA	NA	NA	NA
TNMD:ENSG00000000005	0	NA	NA	NA	NA	NA
DPM1:ENSG00000000419	151.2813052	0.229109109	0.281353207	0.814311347	0.415466611	0.654910102
SCYL3:ENSG00000000457	9.579249027	-0.409918944	0.370807831	-1.105475425	0.268953637	0.521645453
C1orf112:ENSG00000000460	41.97662811	-0.24214877	0.248599391	-0.974052143	0.33003065	0.582892889
FGR:ENSG00000000938	0.404790498	-0.377526098	0.396378555	-0.952438253	0.340874767	NA
CFH:ENSG00000000971	329.6621973	0.083214521	0.556259849	0.14959649	0.881082977	0.947442296
FUCA2:ENSG00000001036	13.75729193	-0.247735458	0.560111923	-0.442296347	0.658274774	0.830452815
GCLC:ENSG00000001084	60.38724968	0.309693536	0.380151483	0.814658234	0.415267967	0.654779367

Figure 21: DEfull.txt

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
CFLAR:ENSG00000003402	39.9725599	-1.3390809	0.32914578	-4.06835204	4.73E-05	0.00243639
WDR54:ENSG00000005448	41.1342173	-1.0498896	0.31243125	-3.360386	0.00077834	0.01470356
KMT2E:ENSG00000005483	142.828476	-1.20951634	0.31531729	-3.83587064	0.00012512	0.00441174
TRAPPC6A:ENSG00000007255	7.27624305	-1.25031688	0.51531214	-2.42632917	0.01525243	0.09759974
RPUSD1:ENSG00000007376	3.22069221	1.28290362	0.510833	2.51139536	0.01202549	0.08408484
LUC7L:ENSG00000007392	22.4784933	1.07737824	0.30939729	3.4821838	0.00049734	0.01090749
SYN1:ENSG00000008056	68.2773928	1.65920689	0.52085936	3.18551807	0.00144495	0.02172925
IDS:ENSG00000010404	37.2144825	-1.19958366	0.2985641	-4.01784293	5.87E-05	0.00267413
CALCOCO1:ENSG00000012822	4.22352463	-1.74891951	0.52455504	-3.33410102	0.00085576	0.01552904

Figure 22: DEfiltered_log2fc_1_fdr_0.1.txt

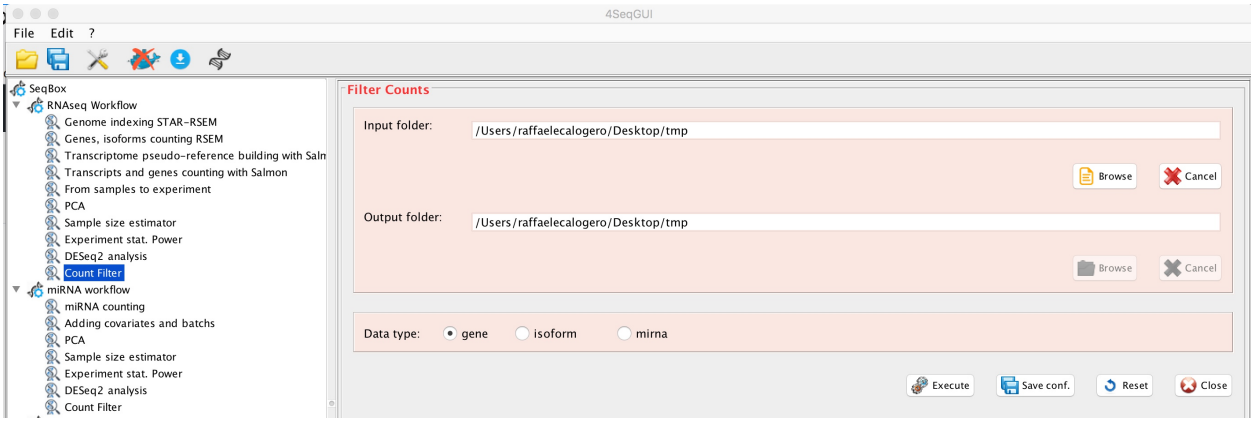

Figure 23: Count Filter

experiment table in use: *gene*, *isoform*, *mirna*, **batch** parameter that indicates, if it is set to **TRUE** that the header of the experiment table also contains the extra information for the batch effect (see above).

IMPORTANT: the above analysis can be also applied to miRNAseq data.

Subsetting counts/FPKM/TPM with differentially expressed genes

The function *filterCounts* allows to generate counts/FPKM/TPM tables that contains only differentially expressed genes. These tables can be used for visualization purposes, e.g. hetmaps generation with **ClustVis**

filterCounts

```
system("wget 130.192.119.59/public/test.analysis.zip")
unzip("test.analysis.zip")
setwd("test.analysis")
library(docker4seq)
     wrapperDeseq2(output.folder=getwd(), group="docker", experiment.table="_counts.txt", log2fc=1,
     fdr=0.1, ref.covar="Cov.1", type="gene", batch=FALSE))

    filterCounts(data.folder=getwd(), type="gene")
```

IMPORTANT: the above analysis can be also applied to miRNAseq data, using for the type parameter **mirna**. In this case also mean centered CPMs are calculated (**NOTE: DE filtered CPM are log2 transformed!**).

The outputs of **filterCounts** function start with **DEfiltered**. Mean centered data indicates that for each gene count is divided by the mean of that gene over all samples. This representation is more convenient to observe changes between experimental groups.

miRNAseq workflow

The miRNAseq workflow can be run using **4SeqGUI** graphical interface:

The miRNAseq docker container executes the following steps:

The full workflow is described in Cordero et al. Plos ONE 2012. In brief, fastq files are trimmed using cutadapt and the trimmed reads are mapped on miRNA precursors, i.e. harpin.fa file, from miRBase using SHRIMP. Using the location of the mature miRNAs in the precursor, countOverlaps function, from the Bioconductor package GenomicRanges is used to quantify the reads mapping on mature miRNAs.

All the parameters needed to run the miRNAseq workflow can be setup using 4SeqGUI

A detailed description of the parameters is given below.

miRNAseq workflow by line command

The miRNAseq workflow can be also executed using R and it is completely embedded in a unique function:

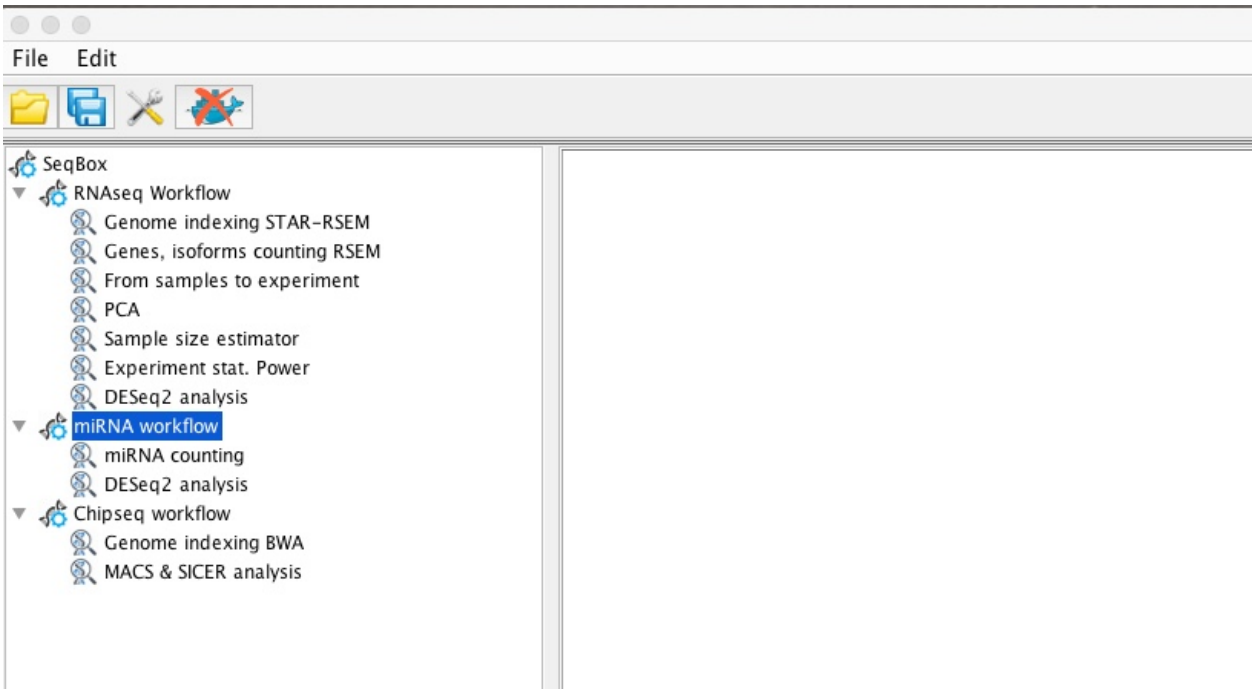

Figure 24: miRNAseq workflow

```
#test example
system("wget 130.192.119.59/public/test.mirnaCounts.zip")
unzip("test.mirnaCounts.zip")
setwd("test.mirnaCounts")
library(docker4seq)
mirnaCounts(group="docker",fastq.folder=getwd(), scratch.folder="/data/scratch",
            mirbase.id="hsa",download.status=FALSE, adapter.type="NEB", trimmed.fastq=FALSE)
```

User has to create the **fastq.folder**, where the fastq.gz files for all miRNAs under analysis are located. The **scratch.folder** is the location where temporary data are created. The results will be then saved in the **fastq.folder**. User has to provide also the identifier of the miRBase organism, e.g. **hsa** for Homo sapiens, **mmu** for Mus musculus. If the **download.status** is set to FALSE, mirnaCounts uses miRBase release 21, if it is set to TRUE the lastest version of precursor and mature miRNAs will be downloaded from miRBase. Users need to provide the name of the producer of the miRNA library prep kit to identify which adapters need to be provided to cutadapt, **adapter.type** parameter. The available adapters are NEB and Illumina, but, upon request, we can add other adapters. Finally, if the **trimmed.fastq** is set to FALSE the trimmed fastq are not saved at the end of the analysis.

miRNAseq workflow output files

The miRNAseq workflow produces the following output files:

```
+ README: A file describing the content of the data folder
+ all.counts.txt: miRNAs raw counts, to be used for differential expression analysis
+ trimmimg.log: adapters trimming statistics
+ shrimp.log: mapping statistics
+ all.counts.Rda: miRNAs raw counts ready to be loaded in R.
+ analysis.log: logs of the full analysis pipeline
```

Adding covariates and batches to mirnaCounts output: all.counts.txt

4SeqGUI provides an interface to add covariates and batches to all.counts.txt

The function **mirnaCovar** add to the header of all.counts.txt covariates and batches or covariates only.

```
#test example
system("wget 130.192.119.59/public/test.mirna.analysis.zip")
unzip("test.mirna.analysis.zip")
setwd("test.mirna.analysis")
library(docker4seq)
```

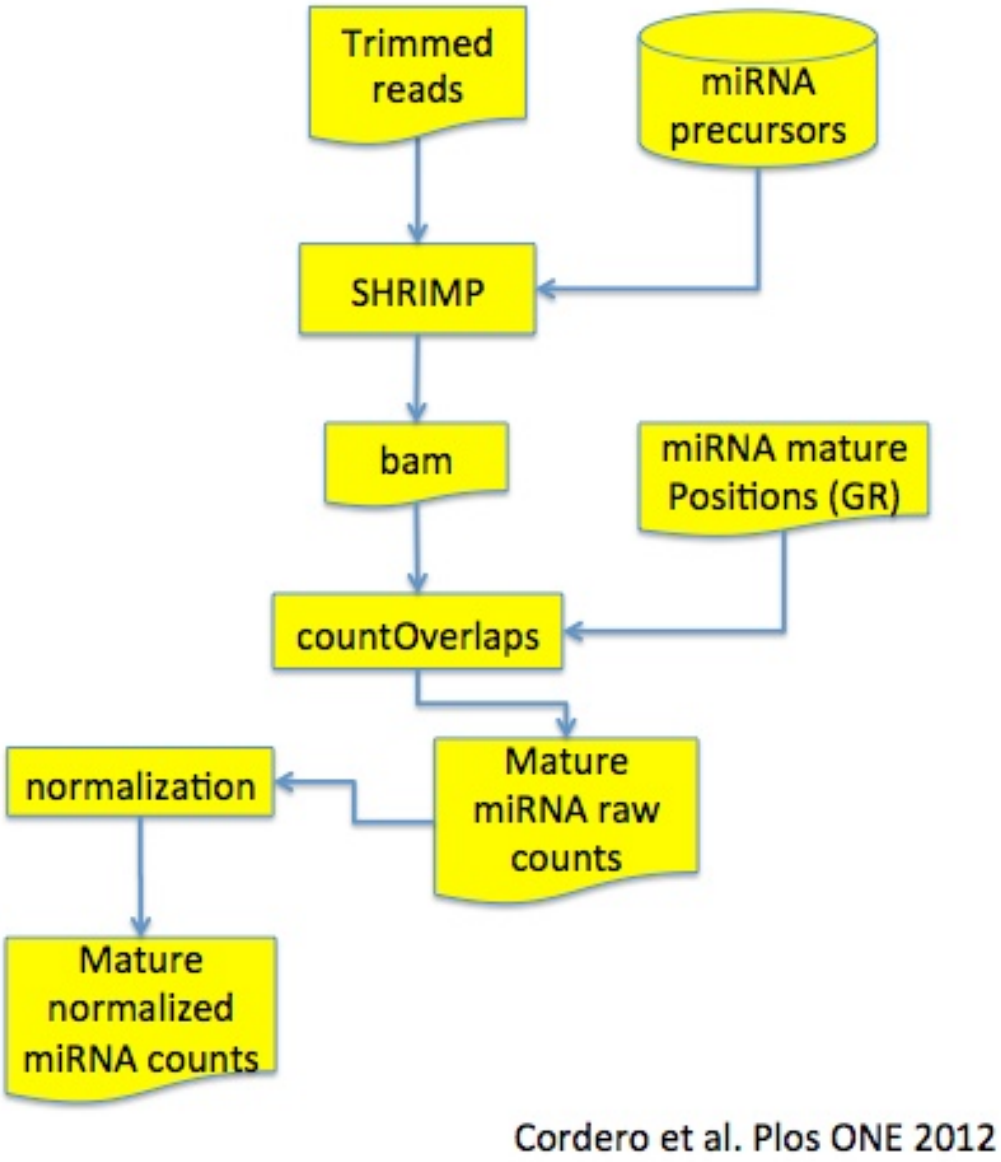

Figure 25: miRNAseq workflow

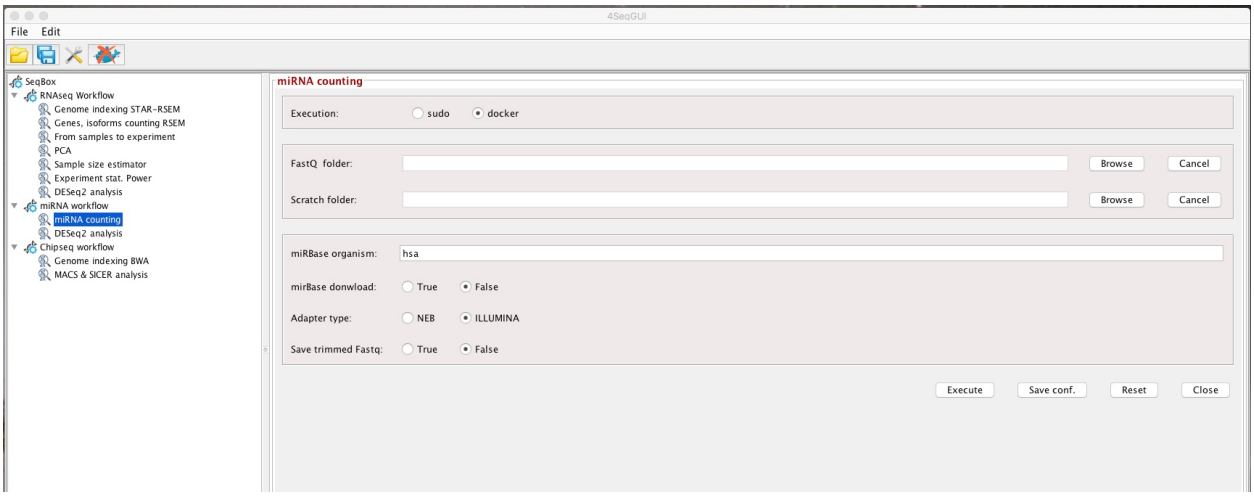

Figure 26: miRNAseq parameters

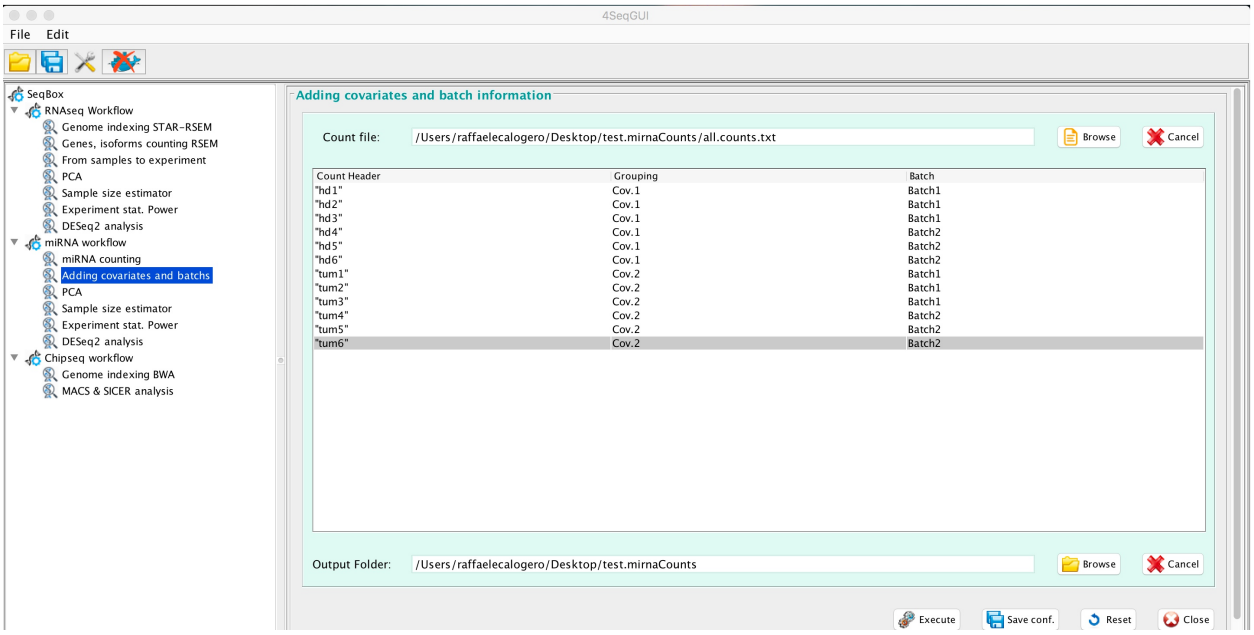

Figure 27: miRNAseq covariates and batches

Figure 28: ChIPseq workflow

```
mirnaCovar(experiment.folder=paste(getwd(), "all.counts.txt", sep="/"),
     covariates=c("Cov.1", "Cov.1", "Cov.1", "Cov.1", "Cov.1", "Cov.1",
                  "Cov.2", "Cov.2", "Cov.2", "Cov.2", "Cov.2", "Cov.2"),
     batches=c("bath.1", "bath.1", "bath.2", "bath.2", "batch.1", "batch.1",
               "batch.2", "batch.2","batch.1", "batch.1","bath.2", "bath.2"), output.folder=getwd())
```

The output of **mirnaCovar**, i.e. w_covar_batch_all.counts.txt, is compliant with PCA, Sample size estimator, Experiment stat. power and DEseq2 analysis.

chipseq workflow

The chipseq workflow can be run using **4SeqGUI** graphical interface:

The ChIPseq is made of two main steps:

- Creating a genome index for BWA (see end of this paragraph)
- Running MACS or SICER analysis

Creating a BWA index file for Chipseq

The index can be easily created using the graphical interface:

```
bwaIndexUcsc(group="sudo",genome.folder="/sto2/data/scratch/mm10bwa", uscs.urlgenome=
"http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/chromFa.tar.gz",
gatk=FALSE)
```

In brief, **bwaIndexUcsc** uses UCSC genomic data. User has to provide the URL (**uscs.urlgenome**) for the file chromFa.tar.gz related to the organism of interest and the path to the folder where the index will be generated (**genome.folder**). The parameter **gatk** has to be set to FALSE, it is not required for ChIPseq genomic index creation.

Precompiled index folders are available:

- mm10bwa

Calling peaks and annotating

All the parameters needed to run MACS or SICER can be setup using 4SeqGUI

A detailed description of the parameters is given below.

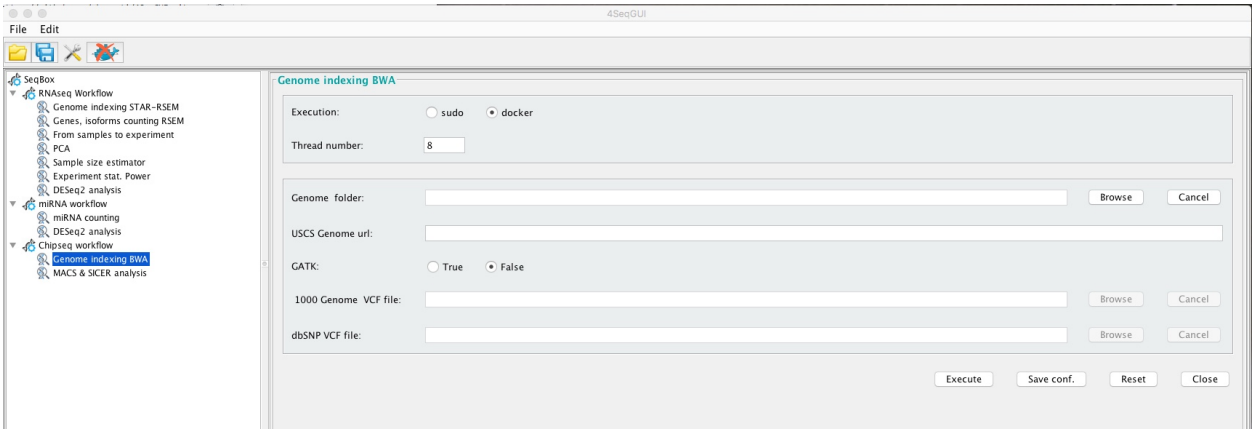

Figure 29: Creating a BWA index with Genome indexing BWA

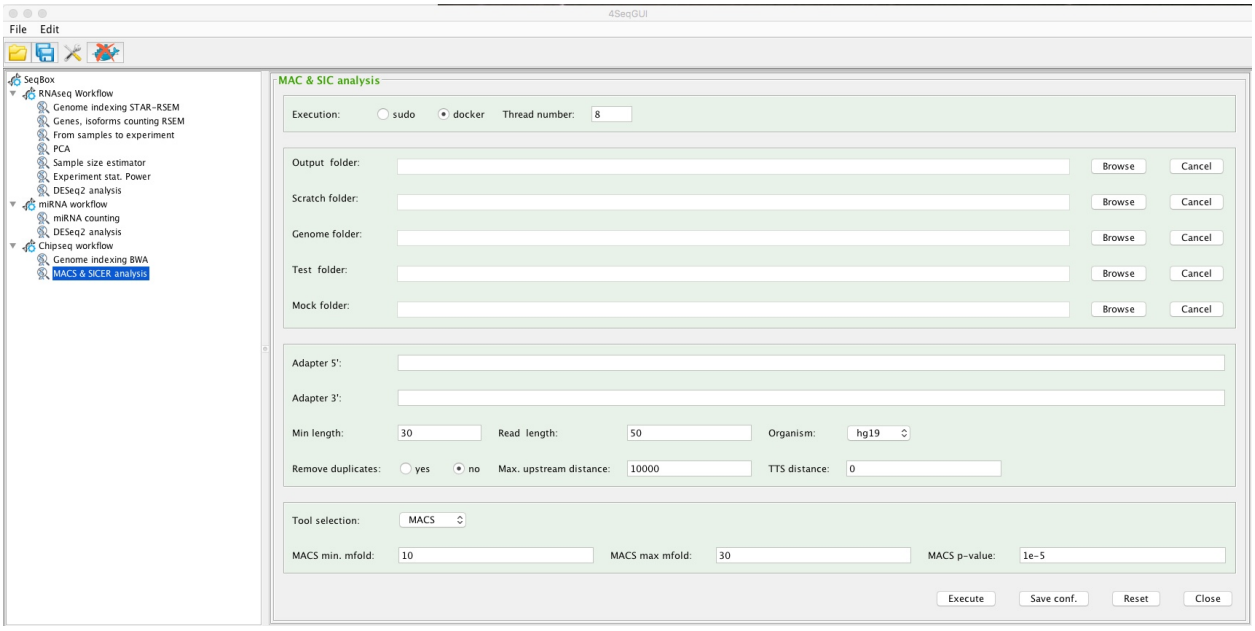

Figure 30: MACS and SICER analysis

Chipseq workflow by line command

The chipseq workflow can be also executed using R and it is completely embedded in a unique function:

```
system("wget 130.192.119.59/public/test.chipseqCounts.zip")
unzip("test.chipseqCounts.zip")
setwd("test.chipseqCounts")
library(docker4seq)
chipseqCounts(group = "docker", output.folder = "./prdm51.igg",
  mock.folder="./igg", test.folder="./prdm51", scratch.folder=getwd(),
  adapter5 = "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA",
  adapter3 = "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT",
  threads = 8, min.length = 30, genome.folder,
  mock.id = "igg", test.id = "tf", genome, read.size = 50,
  tool = "macs", macs.min.mfold = 10, macs.max.mfold = 30,
  macs.pval = "1e-5", sicer.wsize = 200, sicer.gsize = 200,
  sicer.fdr = 0.1, tss.distance = 0, max.upstream.distance = 10000,
  remove.duplicates = "N")
```

Specifically user needs to create three folders:

+ mock.folder, where the fastq.gz file for the control sample is located. For control sample we refer to ChIP with IgG only or i
+ test.folder, where the fastq.gz file for the ChIP of the sample to be analysed.
+ output.folder, where the R script embedding the above script is located.

The **scratch.folder** can be the same as the **output.folder**. However, if the system has a high speed disk for temporary calculation, e.g. a SSD disk, the location of the scratch.folder on the SSD will reduce significantly the computing time.

User needs to provide also the sequence of the sequencing adapters, **adapter5** and **adapter3** parameters. In case Illumina platform the adapters sequences can be easily recovered here.

Threads indicates the max number of cores used by *skewer* and *bwa*, all the other steps are done on a single core. The **min.length** refers to the minimal length that a reads should have after adapters trimming. Since today the average read length for a ChIP experiment is 50 or 75 nts would be better to bring to 40 nts the min.length parameter to increase the precision in assigning the correct position on the genome.

The **genome.folder** parameter refers to the location of the genomic index generated by bwa using the *docker4seq* function **bwaIndexUcsc**.

mock.id and **test.id** identify the type of sample and are assigned to the ID parameter in the RG field of the bam file.

genome is the parameter referring to the annotation used to associate ChIP peaks with genes. In the present implementation hg38, hg19 for human and mm10 and mm9 for mouse annotations are available.

read.size is a parameter requested by MACS and SICER for their analysis. **macs.min.mfold**, **macs.max.mfold**, **macs.pval** are the default parameters requested for peaks definition for more info please refer to the documentation of MACS 1.4. **sicer.wsize**, **sicer.gsize**, **sicer.fdr** are the default parameters requested for peaks definition for more info please refer to the documentation of SICER 1.1. **IMPORTANT**: The optimal value for **sicer.gsize** in case of H3K4Me3 ChIP is 200 and in case of ChIP H3K27Me3 is 600.

tss.distance and **max.upstream.distance** are parameters required by ChIPseqAnno, which is the Bioconductor package used to assign the peaks to specific genes. Specifically max.upstream.distance refers to the max distance in nts that allows the association of a peak to a specific gene.

remove.duplicates is the parameter that indicates if duplicates have to be removed or not. It has two options: **N** duplicates are not removed, **Y** duplicates are removed.

Chipseq workflow output files

The chipseq workflow produces the following output files:

+ README: A file describing the content of the data folder
+ mypeaks.xls: All detected peaks alongside the nearest gene and its annotation
+ mytreat.counts: The total reads count for the provided treatment file
+ mycontrol.counts: The total reads count for the provided control/background file
+ peak_report.xls: Aggregate information regarding the peak and their position relative to the nearest gene
+ chromosome_distribution.pdf: Barplot of the distribution of the peaks on the chromosomes
+ relative_position_distribution.pdf: Barplot of the distribution of the peaks positions relative to their nearest gene
+ peak_width_distribution.pdf: Histogram of the distribution of the width of the peaks
+ distance_from_nearest_gene_distribution.pdf: Histogram of the distribution of the distance of each peak from its nearest gene
+ cumulative_coverage_total.pdf: Cumulative normalized gene coverage
+ cumulative_coverage_chrN.pdf: Cumulative normalized gene coverage for the specific chromosome
+ mycontrol_sorted.bw: bigWig file for UCSC Genome Browser visualization
+ mytreat_sorted.bw: bigWig file for UCSC Genome Browser visualization

PDX Exomeseq workflow

Patient derived tumor xenografts (PDTX) are created when cancerous tissue from a patient's primary tumor is implanted directly into an immunodeficient mouse. PDTX models are providing solutions to the challenges that researchers face in cancer drug research such as positive tumor responses in mouse models but not translating over when the study is implemented in humans. As a result, PDTX cancer models are popular models to use in cancer drug research. Exome sequencing is an importnat step of the PCX characterization. IN PDX we have the human tumor mixex with the mouse stroma and it is necessary to remove the mouse information from the exome data. One way of removing the mouse component is the use of xenome, which performs fast, accurate and specific classification of xenograft-derived sequence read data gossamer.

Creating Xenome index files

The index can be easily created using the the following function:

```
xenomeIndex(group="docker",xenome.folder="/data/scratch/hg19.mm10",
    hg.urlgenome="http://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz",
    mm.urlgenome="http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/chromFa.tar.gz", threads=8)
```

In brief, **xenomeIndex** uses UCSC genomic data. User has to provide the **hg.urlgenome** for the human chromFa.tar.gz, the **mm.urlgenome** for the mouse chromFa.tar.gz and the path to the folder where the index will be generated (**xenome.folder**). The parameter **threads** indicates how many threads the user would like to use.

Running Xenome

Xenome classification can be run using the following code:

```
system("wget http://130.192.119.59/public/mcf7_mouse_1m_R1.fastq.gz")
system("wget http://130.192.119.59/public/mcf7_mouse_1m_R2.fastq.gz")
#running xenome
xenome(group="docker",fastq.folder=getwd(), scratch.folder="/data/scratch",
        xenome.folder="/data/scratch/hg19.mm10", seq.type="pe",
        threads=8)
```

In brief, **xenome.folder** is the location of the output generated by **xenomeIndex**. The **scratch.folder** is the folder where temporary data are created. **seq.type** indicates if it is a pair-end **pe** or a single-end **se** sequence. The parameter **threads** indicates how many threads the user would like to use. **fastq.folder** is the folder were input fastq are located and where the output of xenome will be located.

PDX data preprocessing HowTo

PDX data preprocessing is done stating from the fastq.gz file of a sample present in a user defined folder **fastq.folder**. Data preprocessing is done by **wrapperPdx** with embeds:

- **xenome** for mouse stromal data removal,
- **skewer** for adapter trimming,
- **bwa** for mapping and duplicates marking.

IMPORTANT In case the mutect v1 will be used it is necessary to use, a genome reference for **bwa** the following archive (61 GB):

```
system("wget http://130.192.119.59/public/hg19_exome.tar.gz)
```

wrapperPdx

PDX fastq preprocessing can be done with **wrapperPdx**. These are the parameters to be passed:

```
#example set made of 1 million reads of MCF7 exome data and 1 million reads of mouse genomic DNA pulled down with Illumina Nexter

system("wget http://130.192.119.59/public/mcf7_mouse_1m_R1.fastq.gz")
system("wget http://130.192.119.59/public/mcf7_mouse_1m_R2.fastq.gz")

#running wrapperPdx
wrapperPdx(group="docker",fastq.folder=getwd(), scratch.folder="/data/scratch",
     xenome.folder="/data/scratch/hg19.mm10", seq.type="pe", threads=24,
     adapter5="AGATCGGAAGAGCACACGTCTGAACTCCAGTCA",
     adapter3="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
     min.length=40, genome.folder="/data/scratch/hg19_exome", sample.id="sampleX")
```

In brief, **xenome.folder** is the location of the output generated by **xenomeIndex**. The **scratch.folder** is the folder where temporary data are created. **seq.type** indicates if it is a pair-end **pe** or a single-end **se** sequence. The parameter **threads** indicates how many threads the user would like to use. **fastq.folder** is the folder were input fastq are located and where the output of xenome will be located. User needs to provide also the sequence of the sequencing adapters, **adapter5** and **adapter3**

parameters. In case Illumina platform the adapters sequences can be easily recovered **here**. The **min.length** refers to the minimal length that a reads should have after adapters trimming. Since today the average read length for a RNAseq experiment is 50 or 75 nts would be better to bring to 40 nts the min.length parameter to increase the precision in assigning the correct position on the genome. The **genome.folder** parameter refers to the location of the genomic index **hg19_exome** required for **bwa**, see above. **Sample.id** is a character string indicating the unique id to be associated to the bam that will be created.

wrapperPdx output

The output of the function are three files:

- **xenome_folder**, which contains xeno_ambiguous_ (difficult to be classified as human or mouse), xeno_both_ (classified in both mouse and human), xeno_mm_ (mouse specific reads), xeno_neither_ (not human or mouse, possibly adapter sequences) files.
- **xeno_hs_R1.fastq.gz**, xeno_hs_R2.fastq.gz, hg19 human associated reads, produced by xenome analysis and further processed by **wrapperPdx** to generated bam, bai and stats files.
- One or two gzip fastq files ending with **trimmed-pair1.fastq.gz** and **trimmed-pair1.fastq.gz**, a log file of the trimming with the extensione **trimmed.log**
- **dedup_reads.bam**, which is sorted and duplicates marked bam file generated by **bwa**,
- **dedup_reads.bai**, which is the index of the dedup_reads.bam generated by **bwa**,
- **dedup_reads.stats**, which provides mapping statistics generated by **bwa**.

oncoSNP

The output of oncoSNP is used to associate to gene symbols the detected CN, rank 5 (see oncoSNP help). This association is done using the *oncosnpAnnotation* function. The function requirements are the folder where the ouptut of oncoSNP is saved, i.e. where the file with extension .cnvs are located, and the folder where the ENSEMBL genome.gtf is located. The analysis returns for each .cnvs a file annotation_XXXXXX.cnvs.txt, which contains the genes location, gene symbol and CN and LOH (see oncoSNP help). Please note

```
system("wget http://130.192.119.59/public/test_oncosnp.zip")
system("unzip test_oncosnp.zip")

oncosnpAnnotation(group="docker", data.folder="./test_oncosnp/oncosnp_out", genome.folder="./test_oncosnp/hg19")
```