Introduction to scPower

Matthias Heinig, Katharina Schmid

30 March, 2020

Contents

scPower is a R package for design and power analysis of single cell transcriptomics experiments for differential expression and eQTL analysis. A detection of cell-type specific DE and eQTL genes is possible with the help of single cell RNA-seq. It enables the user to calculate the power for a given experimental setup and to choose for a restricted budget the optimal combination of experimental parameters which maximizes the power. Necessary experimental priors, e.g. effect sizes and expression distributions, can be taken from example data sets, saved in the package, or estimated from new data sets.

The tool was evaluated with data from different tissues and single cell technologies, based on UMI counts and read counts.

A detailed description of all methods and citation of all used tools and data sets can be found in the associated paper "Design and power analysis for multi-sample single cell genomics experiments". An explanation how the plots in the paper were generated can be found in the second vignette "reproduce-paper-plots".

In the first part of this tutorial, the estimation of power and optimal design with the example priors of scPower is shown, in the second part, the fitting of new priors using own data is described. Additionally, in the third part, the power to detect a sufficent number of cells of a specific cell type is explained.

We will show the analysis for an eQTL study, for a DE study it works completely analogously.

```
library(scPower)
library(reshape2)
library(ggplot2)
#> Warning: package 'ggplot2' was built under R version 3.5.2
```

Part 1: Power estimation and experimental design selection with example data sets

In the package, there exist two versions of power estimation, one for droplet based methods using UMI counts, such as 10X Genomics and drop-seq, and one for read based methods, such as Smart-seq2. They differ in the modelling of experimental costs and of multiplet rates (overloading possible for 10X Genomics). Furthermore, the UMI based methods need an additional layer of modelling between reads and UMI counts.

Power estimation

We model the overall detection power as the product of the expression probability of a gene and the eQTL power of a gene. Both are influenced by the three main experimental parameters:

- number of measured samples
- number of measured cells per sample
- read depth

Exemplarily, we want to see the power, which can be reached for a specific combination of the three parameters, when we want to analyze eQTLs in the cell type Monocyte. The experimental parameters are a sample size of 100, 1,500 cells per sample and a read depth to 25,000.

The expected effect sizes and the expression ranks of the eQTL genes have a large influence on the power. To get realistic cell type specific estimates, we use the results of eQTL studies performed with FACS sorted bulk RNA-seq. In this example calculation, a reference study of scPower is used, containing data from the Blueprint project, which estimated eQTLs in FACS sorted Monocytes.

Another important influence on the detection power is how many genes are found in the data set in general. Single cell RNA-seq is typically sparser than bulk RNA-seq and a fraction of eQTL genes can be already missed because no counts are detected for them. So in total, the overall detection power depends on the eQTL power and the expression probability of the genes.

For the expression probability, it needs to be defined, how often the target cells appear in the sample, i.e. the cell type frequency, and how the expression distribution of the genes in this cell type looks like. We assume a cell type frequency of 20%. Expression distribution fits from an example PBMC data set are available in the package, including one for “CD14+ Monocytes”, our cell type of interest. The expression distribution fits describe the mean and dispersion parameters of all genes in the cell type. The mean distribution is modelled as a mixed distribution of with a zero component and two left censored gamma components parameterized over the UMI counts, for which the data frame gamma.mixed.fits is necessary. The relationship between gene expression means and dispersion is modelled using DESeq fits, an example fit is available in the data frame disp.fun.param.

To estimate the mean number of UMI counts for a given read depth, the relationship between read depth and UMI counts is modelled logarithmic. An example fit is available in the data frame read.umi.fit. Additionally, the mapping efficiency of the reads is important, we assume a efficiency of 80%.

The definition of expression can be set user-specific to be above a specific count threshold in a certain fraction of all samples. We will set the defintion of an expressed gene to have more than 3 counts in more than 50% of the samples.

In the following, different example scenarios for different single cell technologies are shown.

Case 1: Droplet-based method with flexible overloading

For droplet-based methods, the so called overloading is possible. Loading more cells on a lane than recommended increases the multiplet rate, but reduces the cost.

We modelled the multiplet rate linear increasing with the number of cells loaded per lane, using data from the 10X Genomics user guide V3. A doublet gets on average more reads than a singlet, for this, a model of Satija lab is used (https://satijalab.org/costpercell). The multiplet factor describes the ratio of reads in multiplets compared to reads in singlets.

In this model, we permit overloading of the lane, setting only the number of samples per lane to a specific number.

```
power<-power.general.withDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                  ct.freq=0.2,type="eqtl",
                                  ref.study=scPower::eqtl.ref.study,
                                  ref.study.name="Blueprint (Monocytes)",
                                  samplesPerLane=4,
                                  read.umi.fit = scPower::read.umi.fit,
                                  gamma.mixed.fits = scPower::gamma.mixed.fits,
                                  ct="CD14+ Monocytes",
                                  disp.fun.param=scPower::disp.fun.param,
                                  mappingEfficiency = 0.8,
                                  min.UMI.counts = 3,
                                  perc.indiv.expr = 0.5)
#> Loading required package: pwr

print(power)
#>                    name powerDetect exp.probs     power sampleSize totalCells
#> 1 Blueprint (Monocytes)   0.3504128 0.5556629 0.6348777        100       1500
#>   usableCells multipletFraction ctCells readDepth readDepthSinglet
#> 1        1431           0.04602     286     25000         24091.28
#>   mappedReadDepth expressedGenes
#> 1        19273.02           8094
```

The output shows an overall detection power for this parameter combination of 35.0%, which is the product of the expression probability of 55.6% and the eQTL power of 63.5%. Of in total 1,500 cells, 1,431 are "usable", which means that they are singlets (estimating a multiplet fraction of 5%). Of this, 286 cells are estimated to be cells of the correct cell type "CD14+ Monocytes". The read depth declines with more multiplets, resulting in an average read depth of 24,091 for a singlet and an average mapped read depths of 19,273. 8094 genes are estimated to be expressed, i.e. to have more than 3 counts in more than 50% of the 100 samples.

Case 2: Droplet-based method with restricted overloading

We implemented a second variant of the power function, where instead of the number of samples per lane the number of cells per lane is set. This restricts the overloading and gives the possibility to set the mutliplet rate. For example, setting the cells per lane to 20,000 results in a multiplet rate of 15%.

```
power<-power.general.restrictedDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                        ct.freq=0.2,type="eqtl",
                                        ref.study=scPower::eqtl.ref.study,
                                        ref.study.name="Blueprint (Monocytes)",
                                        cellsPerLane=20000,
                                        read.umi.fit = scPower::read.umi.fit,
                                        gamma.mixed.fits = scPower::gamma.mixed.fits,
                                        ct="CD14+ Monocytes",
```

```
                                        disp.fun.param=scPower::disp.fun.param,
                                        mappingEfficiency = 0.8,
                                        min.UMI.counts = 3,
                                        perc.indiv.expr = 0.5)

print(power)
#>                    name powerDetect exp.probs     power sampleSize totalCells
#> 1 Blueprint (Monocytes)   0.3378688  0.534071 0.6369226        100       1500
#>   usableCells multipletFraction ctCells readDepth readDepthSinglet
#> 1        1276          0.149565     255     25000         22272.64
#>   mappedReadDepth expressedGenes
#> 1        17818.11           7774
```

In this case, the power decreases slightly compared to case 1, but the overall experimental cost is lower.

Experimental cost calculation

The experimental costs are the sum of the library preparation cost and the sequencing costs. They dependent on the three main experimental parameters (sample size, cells per sample and read depth). The library prepration costs for 10X are determined by the cost of a 10X kit, the sequencing cost by the cost of a flow cell and the reads that can be sequenced with a flow cell. Example values are chosen for all three to calculate the experimental costs for case 1 and case 2.

```
costKit<-5600
costFlowCell<-14032
readsPerFlowcell<-4100*10^6

#Experimental cost for case 1
cost<-budgetCalculation(nSamples=100,nCells=1500,readDepth=25000,
                        costKit=costKit,samplesPerLane=4,
                        costFlowCell=costFlowCell,readsPerFlowcell = readsPerFlowcell)
print(paste("Costs for case 1:",round(cost)))
#> [1] "Costs for case 1: 36167"

#Experimental cost for case 2
cost<-budgetCalculation.restrictedDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                           costKit=costKit,cellsPerLane=20000,
                                           costFlowCell=costFlowCell,
                                           readsPerFlowcell = readsPerFlowcell)
print(paste("Costs for case 2:",round(cost)))
#> [1] "Costs for case 2: 20014"
```

Case 3: Droplet-based method with constant multiplet rate

If data is missing to model the overloading of the lane, a constant multiplet rate can be used instead. This is influenced by the parameter multipletRateGrowth, which is either "linear" (for a linear increase with cells per lane during overloading) or "constant". The parameter multipletRate will be interpreted correspondingly.

```
power<-power.general.restrictedDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                        ct.freq=0.2,type="eqtl",
                                        ref.study=scPower::eqtl.ref.study,
                                        ref.study.name="Blueprint (Monocytes)",
                                        cellsPerLane=8000,
```

```
                                        read.umi.fit = scPower::read.umi.fit,
                                        gamma.mixed.fits = scPower::gamma.mixed.fits,
                                        ct="CD14+ Monocytes",
                                        disp.fun.param=scPower::disp.fun.param,
                                        mappingEfficiency = 0.8,
                                        min.UMI.counts = 3,
                                        perc.indiv.expr = 0.5,
                                        multipletRateGrowth = "constant",
                                        multipletRate=0.05)

print(power)
#>                    name powerDetect exp.probs     power sampleSize totalCells
#> 1 Blueprint (Monocytes)   0.3500381  0.555008 0.6349403        100       1500
#>   usableCells multipletFraction ctCells readDepth readDepthSinglet
#> 1        1425              0.05     285     25000         24015.37
#>   mappedReadDepth expressedGenes
#> 1         19212.3           8084
```

Case 4: Smart-seq2

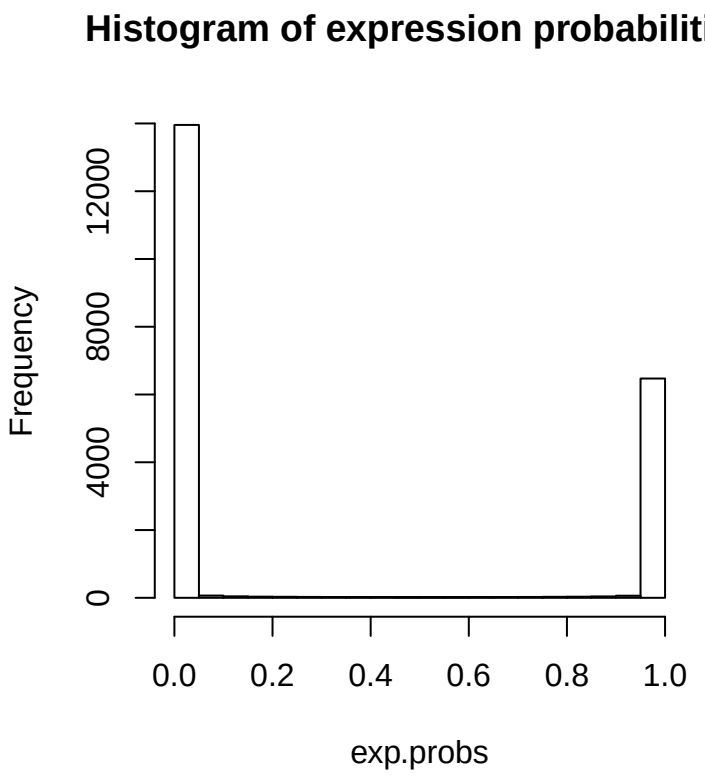
Read based technologies, such as Smart-seq2, need to be modelled slightly differently. The fit between reads and UMI is not required, as the gamma mixed curves are directly parameterized over the read depth. Instead, specific gene curve fits with Smart-seq data are used. As an additional prior, the gene length of the eQTL/DE genes is taken here, which can also be gained from the FACS sorted bulk eQTL/DE studies. As example case, a pancreas Smart-seq data set is used here together with a DE study performed in pancreas tissue.

```
power<-power.smartseq(nSamples=100,nCells=1500,readDepth=25000,
                      ct.freq=0.2,type="de",
                      ref.study=scPower::de.ref.study,
                      ref.study.name="Pancreas_alphabeta",
                      gamma.mixed.fits = scPower::gamma.mixed.fits.smart,
                      ct="alpha",
                      disp.linear.fit = scPower::disp.fun.param.smart,
                      mappingEfficiency = 0.8,
                      min.norm.count = 3,
                      perc.indiv.expr = 0.5)
#> Loading required package: MKmisc
#> Warning: package 'MKmisc' was built under R version 3.5.2

print(power)
#>                  name powerDetect exp.probs     power sampleSize totalCells
#> 1 Pancreas_alphabeta   0.5465679  0.547559 0.6611329        100       1500
#>   usableCells multipletFraction ctCells readDepth readDepthSinglet
#> 1        1500                 0     300     25000            25000
#>   mappedReadDepth expressedGenes
#> 1           20000          12232
```

Expression probabilities

As mentioned before, the detection power is the product of the expression probabilities and the power. The expression probabilities can also be calculated independently, using the function estimate.exp.prob.param. However, in this function doublets and cell type frequencies are not taken into account, but instead directly the number of cells per cell type and individual need to be entered.

```
exp.probs<-scPower::estimate.exp.prob.param(nSamples=100,readDepth=25000,nCellsCt=150,
                                            read.umi.fit = scPower::read.umi.fit,
                                            gamma.mixed.fits = scPower::gamma.mixed.fits,
                                            ct="CD14+ Monocytes",
                                            disp.fun.param=scPower::disp.fun.param,
                                            min.counts = 3,
                                            perc.indiv = 0.5)

hist(exp.probs,main="Histogram of expression probabilities")
```




```
print(paste("Expected number of expressed genes:",round(sum(exp.probs))))
#> [1] "Expected number of expressed genes: 6754"
```

Selection of optimal parameter combination for a restricted budget

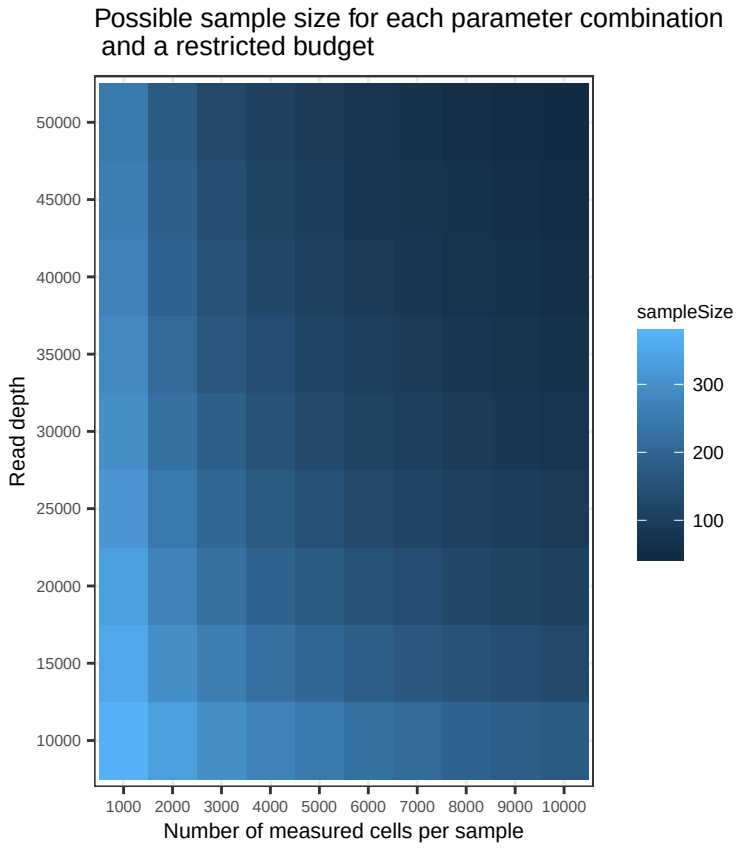
scPower gives the user the opportunity to select the best parameter combination for a restricted budget. In the following example, we optimize the experimental design for a budget of 100,000. For two of the three parameters, sample size, cells per individual and read depth, vectors with potential values are required, the third variable is determined uniquely given the other two and the overall budget. Which of the variables is left out, can be freely chosen. In this example we set the number of cells per individual and read depths in the parameters readDepthRange and cellPersRange, and leave out the sample size, which will be determined by the algorithm

To calculate the budget, again costs per 10X kit and flow cells and reads per flow cell need to be defined. The other parameters are the ones that were already necessary for the power calculation in the section before, such as gamma and dispersion fits.

```
opt.design<-optimize.constant.budget(totalBudget=100000,type="eqtl",
```

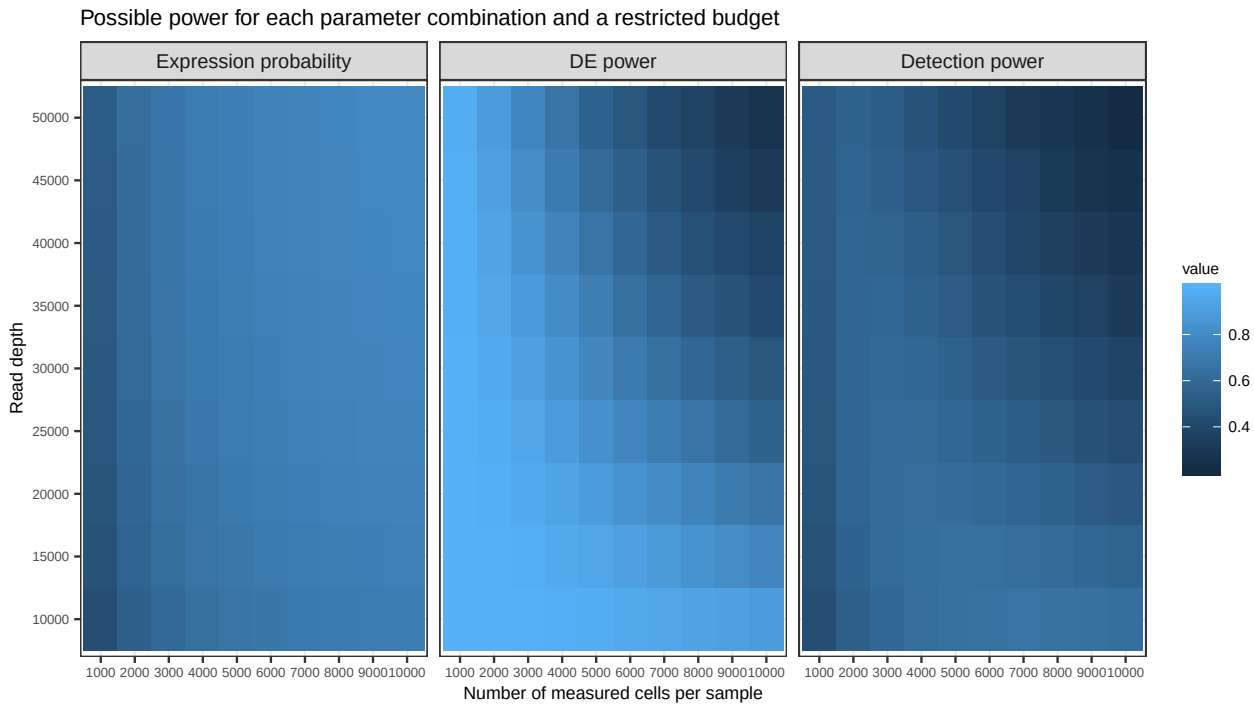
```
                                    ct="CD14+ Monocytes",ct.freq=0.2,
                                    costKit=5600,
                                    costFlowCell=14032,
                                    readsPerFlowcell = 4100*10^6,
                                    ref.study=scPower::eqtl.ref.study,
                                    ref.study.name="Blueprint (Monocytes)",
                                    samplesPerLane=4,
                                    read.umi.fit = scPower::read.umi.fit,
                                    gamma.mixed.fits = scPower::gamma.mixed.fits,
                                    disp.fun.param=scPower::disp.fun.param,
                                    nSamplesRange=NULL,
                                    nCellsRange=seq(1000,10000,by=1000),
                                    readDepthRange=seq(10000,50000,by=5000),
                                    mappingEfficiency = 0.8)

#First lines of result data frame
head(opt.design)
#>   name powerDetect exp.probs     power sampleSize totalCells usableCells
#> 1    1   0.4371104 0.4374897 0.9991525        372       1000         969
#> 2    1   0.5454649 0.5470747 0.9970108        330       2000        1877
#> 3    1   0.5947518 0.5991505 0.9925591        296       3000        2724
#> 4    1   0.6272078 0.6361099 0.9857326        270       4000        3509
#> 5    1   0.6449472 0.6612685 0.9749007        246       5000        4233
#> 6    1   0.6526992 0.6787536 0.9607818        226       6000        4896
#>   multipletFraction ctCells readDepth readDepthSinglet mappedReadDepth
#> 1           0.03068     194     10000         9752.102        7801.681
#> 2           0.06136     375     10000         9519.911        7615.929
#> 3           0.09204     545     10000         9298.520        7438.816
#> 4           0.12272     702     10000         9085.499        7268.399
#> 5           0.15340     847     10000         8882.667        7106.134
#> 6           0.18408     979     10000         8689.003        6951.203
#>   expressedGenes
#> 1           6444
#> 2           7958
#> 3           8749
#> 4           9251
#> 5           9606
#> 6           9867

#Optimal experimental design combination
print(opt.design[which.max(opt.design$powerDetect),])
#>   name powerDetect exp.probs     power sampleSize totalCells usableCells
#> 7    1   0.6547746 0.6921425 0.9448369        210       7000        5497
#>   multipletFraction ctCells readDepth readDepthSinglet mappedReadDepth
#> 7           0.21476    1099     10000         8502.926        6802.341
#>   expressedGenes
#> 7          10067

#Plot results
power.DE.study<-melt(opt.design,id.vars=c("name","sampleSize","totalCells","usableCells",
                                          "multipletFraction","ctCells","readDepth",
                                          "readDepthSinglet","mappedReadDepth",
                                          "expressedGenes"))
```

```
power.DE.study$variable<-factor(power.DE.study$variable,levels=c("exp.probs","power","powerDetect"))

var.labs<-c("Expression probability", "DE power",
            "Detection power")
names(var.labs)<-c("exp.probs","power","powerDetect")

power.DE.study$totalCells<-as.factor(power.DE.study$totalCells)
power.DE.study$readDepth<-as.factor(power.DE.study$readDepth)
g<-ggplot(power.DE.study,aes(x=totalCells,y=readDepth,fill=sampleSize))+
  geom_tile()+
  xlab("Number of measured cells per sample")+ylab("Read depth")+
  ggtitle("Possible sample size for each parameter combination \n and a restricted budget")+
  theme_bw()+
  theme(plot.title = element_text(size=10),
        axis.title=element_text(size=8),
        axis.text=element_text(size=6),
        legend.title=element_text(size=7),
        legend.text=element_text(size=7))

print(g)
```




```
g<-ggplot(power.DE.study,aes(x=totalCells,y=readDepth,fill=value))+
  geom_tile()+facet_wrap(~variable,ncol=3,labeller = labeller(variable=var.labs))+
  xlab("Number of measured cells per sample")+ylab("Read depth")+
  ggtitle("Possible power for each parameter combination and a restricted budget")+
  theme_bw()+
  theme(plot.title = element_text(size=10),
```

```
        axis.title=element_text(size=8),
        axis.text=element_text(size=6),
        legend.title=element_text(size=7),
        legend.text=element_text(size=7))

print(g)
```



The function returns for each combination of read depth and cells per individual, how many samples can be measured with the given budget and which detection power can be reached. The combination in the data frame with the maximal detection power is the optimal experimental design.

Instead of the sample size, the flexible parameter can also be the number of cells per individual or the read depth. We set here the sample size to values between 50 and 300 and leave one of the other two parameters out.

```
#Infer cells per sample
opt.design<-optimize.constant.budget(totalBudget=100000,type="eqtl",
                                     ct="CD14+ Monocytes",ct.freq=0.2,
                                     costKit=5600,
                                     costFlowCell=14032,
                                     readsPerFlowcell = 4100*10^6,
                                     ref.study=scPower::eqtl.ref.study,
                                     ref.study.name="Blueprint (Monocytes)",
                                     samplesPerLane=4,
                                     read.umi.fit = scPower::read.umi.fit,
                                     gamma.mixed.fits = scPower::gamma.mixed.fits,
                                     disp.fun.param=scPower::disp.fun.param,
                                     nSamplesRange=seq(100,300,by=25),
                                     nCellsRange=NULL,
                                     readDepthRange=seq(10000,50000,by=5000),
                                     mappingEfficiency = 0.8)
```

```
#Optimal experimental design combination
print(opt.design[which.max(opt.design$powerDetect),])
#>   name powerDetect exp.probs     power sampleSize totalCells usableCells
#> 6    1   0.6548442  0.681604 0.9598763        225       6168        5001
#>   multipletFraction ctCells readDepth readDepthSinglet mappedReadDepth
#> 6         0.1892342    1000     10000         8656.915        6925.532
#>   expressedGenes
#> 6           9910

#Infer read depth
opt.design<-optimize.constant.budget(totalBudget=100000,type="eqtl",
                                     ct="CD14+ Monocytes",ct.freq=0.2,
                                     costKit=5600,
                                     costFlowCell=14032,
                                     readsPerFlowcell = 4100*10^6,
                                     ref.study=scPower::eqtl.ref.study,
                                     ref.study.name="Blueprint (Monocytes)",
                                     samplesPerLane=4,
                                     read.umi.fit = scPower::read.umi.fit,
                                     gamma.mixed.fits = scPower::gamma.mixed.fits,
                                     disp.fun.param=scPower::disp.fun.param,
                                     nSamplesRange=seq(100,300,by=25),
                                     nCellsRange=seq(1000,10000,by=1000),
                                     readDepthRange=NULL,
                                     mappingEfficiency = 0.8)


#Optimal experimental design combination
print(opt.design[which.max(opt.design$powerDetect),])
#>    name powerDetect exp.probs     power sampleSize totalCells usableCells
#> 78    1   0.6603405 0.6874044 0.9597533        225       9000        6515
#>    multipletFraction ctCells readDepth readDepthSinglet mappedReadDepth
#> 78           0.27612    1303      6853         5587.849        4470.279
#>    expressedGenes
#> 78           9996
```

Similar to the power calculation, variants of the function exist to model a restricted number of cells per lane (optimize.constant.budget.restrictedDoublets) and to model Smart-seq data (optimize.constant.budget.smartseq). Furthermore, there is the option to calculate the library preparation cost per cell instead of per kit (optimize.constant.budget.libPrepCell).

Part 2: Generation of new priors from a data set

Additionally to the priors given by the package, scPower offers the option to generate new priors from your own data set. Two types of priors are necessary:

- the cell type specific expression distribution of all genes for the expression probability curves, which can be fitted from single cell RNA-seq data of the same technology (e.g. data from a pilot study). The distribution include the read-UMI fit to estimate the UMI counts per cell, the gamma mixed fits for the mean values of each gene and the mean-dispersion fits for the dispersion values of each gene.
- effect sizes and expression ranks of the DE/eQTL genes, which can be taken from any kind of study or simulated. We recommend studies with FACS sorted bulk RNA-seq to get a realistic cell type specific estimations of the priors.

Expression probability curves

A small example data set of a UMI count matrix is provided in the package to show the fitting of the expression probability curves. It contains data of one cell type, as each cell type needs to be fitted separately.

The data set is a list of four raw count matrices, the first one is the original count matrix and 3 subsampled matrices to evaluate the effect of the read depth. The subsampling was performed on the reads using the tool fastq-sample from fastq-tools version 0.8 and then the mapping repeated with cellranger.

```
data(countMatrixExample)

#Dimensions of the three count matrices
sapply(count.matrix.example,dim)
#>      complete subsampled75 subsampled50 subsampled25
#> [1,]    32738        32738        32738        32738
#> [2,]      410          410          410          409
```

Counting observed expressed genes

The number of expressed genes in the data set can be estimated by reformating the 2d count matrix into a 3d pseudobulk matrix, using the function "create.pseudobulk". Therefore, an annotation data frame is required with individual and cell type annotations for each cells. The rows of the annotaiton data frame must match the columns in the count matrix (same ordering of cells). After the pseudobulk matrix is created, the expressed genes are extracted using the function "calculate.gene.counts". Here, we use a threshold of more than 3 counts in more 50% of the individuals.

```
expressed.genes.df<-NULL
#Iterate over each count matrix
for(name in names(count.matrix.example)){

  count.matrix<-count.matrix.example[[name]]

  #Create an annotation file (here containing only one cell type, but can be more)
  annot.df<-data.frame(individual=paste0("S",rep(1:14,length.out=ncol(count.matrix))),
                       cell.type=rep("default_ct",ncol(count.matrix)))
  #Reformat count matrix into pseudobulk matrix
  pseudo.bulk<-create.pseudobulk(count.matrix,annot.df)
  #Calculate expressed genes in the pseudobulk matrix
  expressed.genes<-calculate.gene.counts(pseudo.bulk,min.counts=3, perc.indiv=0.5)
  #Get the number of expressed genes
  num.expressed.genes<-nrow(expressed.genes)

  #Save expressed genes
  expressed.genes.df<-rbind(expressed.genes.df,
                            data.frame(matrix=name,
                                       num.cells=ncol(count.matrix),
                                       expressed.genes=num.expressed.genes))
}

print(expressed.genes.df)
#>         matrix num.cells expressed.genes
#> 1     complete       410            2076
#> 2 subsampled75       410            1962
#> 3 subsampled50       410            1773
#> 4 subsampled25       409            1367
```

As expected, the subsampling of the count matrix leads to an reduction of the number of expressed genes.

Estimation of negative binomial paramters for each gene

As a first step for fitting the expression distribution, the negative binomial fit for each gene, i.e. the mean and disperion parameter, is estimated with "nbinom.estimation". The function uses DESeq for library normalization and parameter estimation. For the normalization, the standard normalization of DEseq can be used or a variant of it called poscounts. It can be set in the parameter "sizeFactors" of the function. Default is "standard", but for very sparse data, "poscounts" might be the better option. For too sparse data, "standard" is not working and will give an error message.

The function returns a list with three elements: the normalized mean values, the dispersion values and the parameters of the mean-dispersion function fitted from DESeq. We will save the mean values and the parameters of the mean-dispersion function for further processing.

```
norm.mean.values<-NULL
disp.param<-NULL
for(name in names(count.matrix.example)){
  temp<-nbinom.estimation(count.matrix.example[[name]])

  #Save the normalized mean values
  norm.mean.values.temp<-temp[[1]]
  norm.mean.values.temp$matrix<-name
  norm.mean.values<-rbind(norm.mean.values,norm.mean.values.temp)

  #Save the parameter of the mean-dispersion function
  disp.param.temp<-temp[[3]]
  disp.param.temp$matrix<-name
  disp.param<-rbind(disp.param,disp.param.temp)
}
#> Loading required package: DESeq
#> Warning: package 'DESeq' was built under R version 3.5.2
#> Loading required package: BiocGenerics
#> Warning: package 'BiocGenerics' was built under R version 3.5.1
#> Loading required package: parallel
#>
#> Attaching package: 'BiocGenerics'
#> The following objects are masked from 'package:parallel':
#>
#>     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
#>     clusterExport, clusterMap, parApply, parCapply, parLapply,
#>     parLapplyLB, parRapply, parSapply, parSapplyLB
#> The following objects are masked from 'package:stats':
#>
#>     IQR, mad, sd, var, xtabs
#> The following objects are masked from 'package:base':
#>
#>     anyDuplicated, append, as.data.frame, basename, cbind, colMeans,
#>     colnames, colSums, dirname, do.call, duplicated, eval, evalq,
#>     Filter, Find, get, grep, grepl, intersect, is.unsorted, lapply,
#>     lengths, Map, mapply, match, mget, order, paste, pmax, pmax.int,
#>     pmin, pmin.int, Position, rank, rbind, Reduce, rowMeans, rownames,
#>     rowSums, sapply, setdiff, sort, table, tapply, union, unique,
#>     unsplit, which, which.max, which.min
#> Loading required package: Biobase
```

```
#> Warning: package 'Biobase' was built under R version 3.5.1
#> Welcome to Bioconductor
#>
#>     Vignettes contain introductory material; view with
#>     'browseVignettes()'. To cite Bioconductor, see
#>     'citation("Biobase")', and for packages 'citation("pkgname")'.
#> Loading required package: locfit
#> locfit 1.5-9.1    2013-03-22
#> Loading required package: lattice
#>     Welcome to 'DESeq'. For improved performance, usability and
#>     functionality, please consider migrating to 'DESeq2'.

#First rows of the data frame with normalized mean values
head(norm.mean.values)
#>                             gene       mean   matrix
#> ENSG00000243485 ENSG00000243485 0.00000000 complete
#> ENSG00000237613 ENSG00000237613 0.00000000 complete
#> ENSG00000186092 ENSG00000186092 0.00000000 complete
#> ENSG00000238009 ENSG00000238009 0.00343702 complete
#> ENSG00000239945 ENSG00000239945 0.00000000 complete
#> ENSG00000237683 ENSG00000237683 0.00323091 complete

#Parameter of the mean - dispersion function
print(disp.param)
#>             asymptDisp extraPois       matrix
#> asymptDisp  0.07182929 0.1783764     complete
#> asymptDisp1 0.07198079 0.1753460 subsampled75
#> asymptDisp2 0.06526182 0.1785742 subsampled50
#> asymptDisp3 0.05327572 0.1784178 subsampled25
```

Estimation of a gamma mixed distribution over all means

A mixed distribution of a zero component and two left zensored gamma components is fitted for each mean value vector (one fit per count matrix). The count matrix contains often a large number of genes with a mean of zero. To reduce the matrix size and faciliate so the fit, the parameter of "num.genes.kept" regulates the number of mean values for the fit (it removes only genes with a mean of zero). The number should however never be smaller than the number of positive means values. As a fraction of the zero values can be expressed genes below the detection threshold of the experiment, the gamma distributions are modelled as left censored. If no censoring point is set in the variable "censoredPoint", the smallest positive value in the data set is taken as the censoring point. For our models, we use 1/num_cells_celltype as the censoring point.

```
gamma.fits<-NULL
for(name in names(count.matrix.example)){

  #Number of cells per cell type as censoring point
  censoredPoint<- 1 / ncol(count.matrix.example[[name]])

  norm.mean.values.temp<-norm.mean.values[norm.mean.values$matrix==name,]
  gamma.fit.temp<-mixed.gamma.estimation(norm.mean.values.temp$mean,
                                         num.genes.kept = 21000,
                                         censoredPoint = censoredPoint)
  gamma.fit.temp$matrix<-name
  gamma.fits<-rbind(gamma.fits,gamma.fit.temp)
```

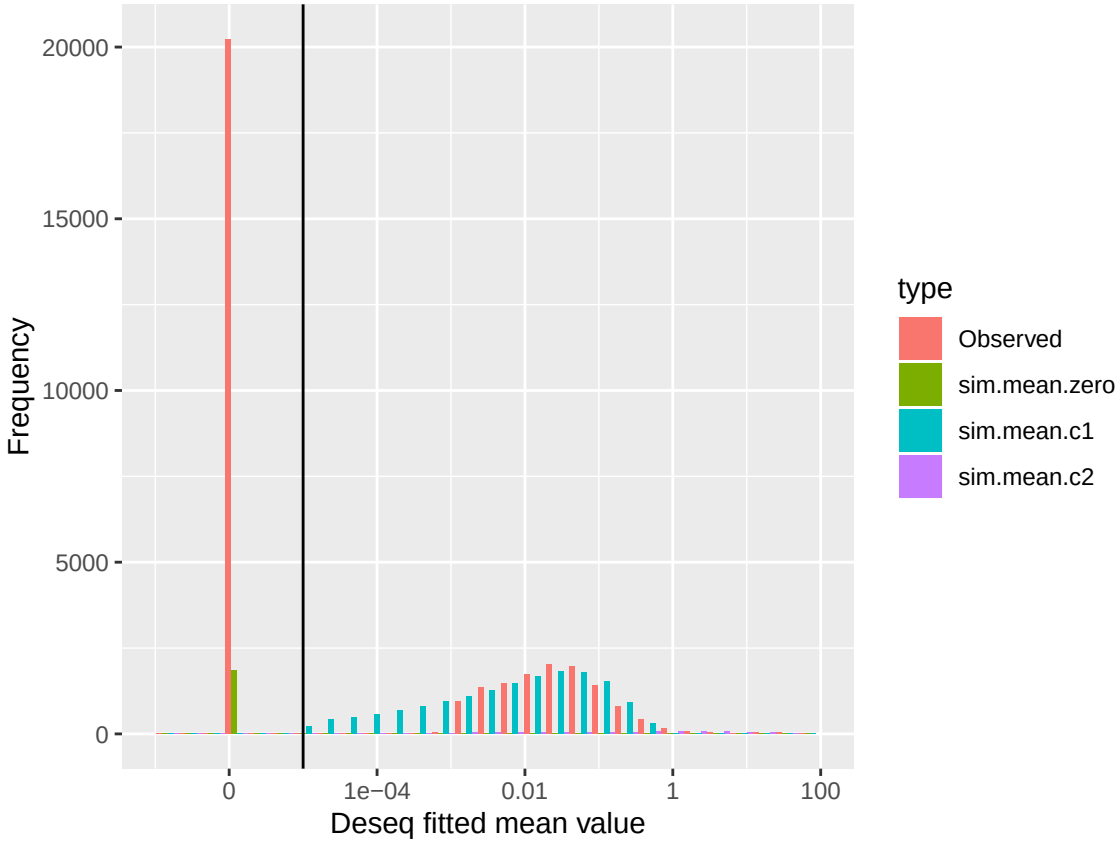
```
}
#> Warning in em(mean.vals, ncomp = 3, prop = c(zero.prop, 1 - (zero.prop + :
#> Problem in the EM algorithm: likelihood is decreasing!

print(gamma.fits)
#>                p1        p2        s1        s2       r1         r2
#> shape  0.08758294 0.8614068 0.2286524 0.1125516 5.530817 0.04105659
#> shape1 0.09049283 0.8585065 0.2273825 0.1130025 5.869807 0.04355567
#> shape2 0.09408661 0.8549195 0.2252026 0.1134584 6.514028 0.04837719
#> shape3 0.10217340 0.8468386 0.2224447 0.1123769 8.464049 0.06310104
#>              matrix
#> shape      complete
#> shape1 subsampled75
#> shape2 subsampled50
#> shape3 subsampled25
```

Comparison of gamma mixed fits with original means

The fit can be evaluated by simulating mean values using this gamma distribution. In general, the package provides two options for the sampling the values, either random (sample.mean.values.random) or using the quantiles of the gamma mixed distribution (sample.mean.values.quantiles).

```
#Check fit for the full matrix
sim.mean.vals<-sample.mean.values.quantiles(gamma.fits[gamma.fits$matrix=="complete",],
                                            nGenes=21000)

#Calculate from which component each value in the simulation originates (ZI + LZG1 + LZG2)
nGenes<-21000
nZeros<-round(nGenes*gamma.fits$p1[gamma.fits$matrix=="complete"])
nGamma1<-round(nGenes*gamma.fits$p2[gamma.fits$matrix=="complete"])
nGamma2<-nGenes-nZeros-nGamma1

plot.values<-data.frame(means=c(norm.mean.values$mean[norm.mean.values$matrix=="complete"],
                                sim.mean.vals),
                        type=c(rep("Observed",sum(norm.mean.values$matrix=="complete")),
                               rep("sim.mean.zero",nZeros),
                               rep("sim.mean.c1",nGamma1),
                               rep("sim.mean.c2",nGamma2)))

#Set zero values to a very small value to be able to plot on logarithmic scale
lower_dist<-1e-5
zero_pos<-1e-6

plot.values<-plot.values[plot.values$mean==0 | plot.values$mean>=lower_dist,]
plot.values$mean[plot.values$mean==0]<-zero_pos

#Order estimated means
plot.values$type<-factor(plot.values$type,
                         levels=c("Observed","sim.mean.zero","sim.mean.c1","sim.mean.c2"))

ggplot(data=plot.values,aes(x=mean, fill=type))+
  geom_histogram(position="dodge")+
  scale_x_log10(limits = c(1e-7,1e2), breaks=c(1e-6,1e-4,1e-2,1e0,1e2),
```

```
                labels=c(0,c(1e-4,1e-2,1e0,1e2)))+geom_vline(xintercept=1e-5)+
  xlab("Deseq fitted mean value")+ylab("Frequency")
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
#> Warning: Removed 1 rows containing non-finite values (stat_bin).
#> Warning: Removed 5 rows containing missing values (geom_bar).
```

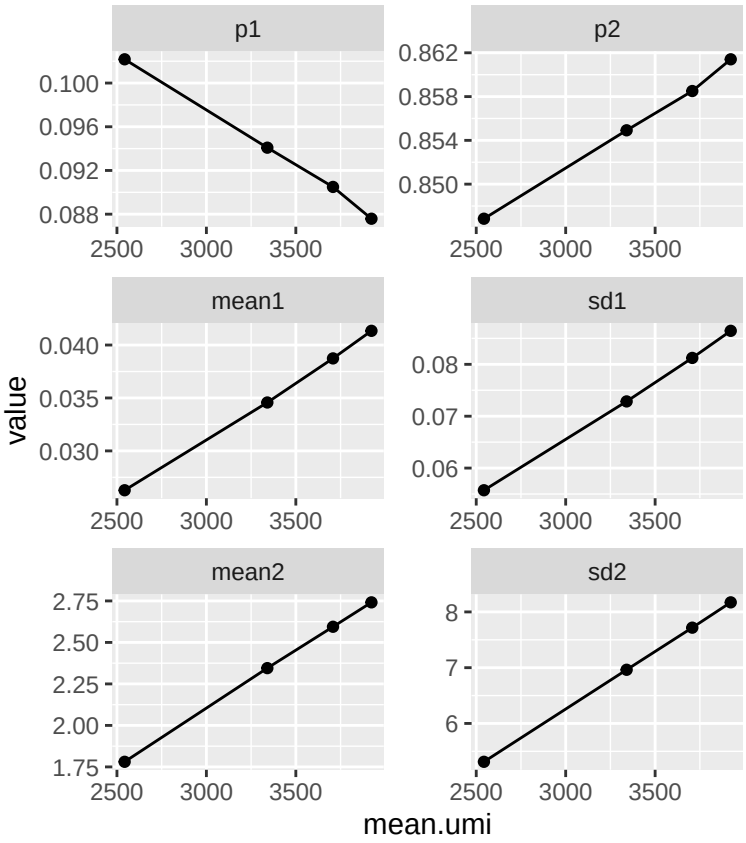


Parameterization of the parameters of the gamma fits by the mean UMI counts per cell

The gamma fits over all matrices are parameterized by the mean UMI counts per cell. The mean UMI counts are calculated from the original count matrices using the function "meanUMI.calculation". For the fits, the parameterization of the mixtures need to be converted from rate-shape parameters to mean-standard deviation parameters by "convert.gamma.parameters".

```
#Estimate the mean umi values per cell for each matrix
umi.values<-NULL
for(name in names(count.matrix.example)){
  mean.umi<-meanUMI.calculation(count.matrix.example[[name]])
  umi.values<-rbind(umi.values,data.frame(mean.umi,matrix=name))
}

print(umi.values)
#>   mean.umi       matrix
#> 1 3922.559     complete
#> 2 3707.690 subsampled75
#> 3 3340.644 subsampled50
#> 4 2543.046 subsampled25
```

```
gamma.fits<-merge(gamma.fits,umi.values,by="matrix")

#Convert the gamma fits from the shape-rate parametrization to the mean-sd parametrization
gamma.fits<-convert.gamma.parameters(gamma.fits)

#Visualize the linear relationship between gamma parameters and UMI values in plots
plot.values<-melt(gamma.fits,id.vars=c("matrix","mean.umi"))
plot.values<-plot.values[plot.values$variable %in% c("mean1","mean2","sd1","sd2","p1","p2"),]
ggplot(plot.values,aes(x=mean.umi,y=value))+
  geom_point()+geom_line()+
  facet_wrap(~variable,ncol=2,scales="free")
```




```
#Fit relationship between gamma parameters and UMI values
gamma.linear.fit.new<-umi.gamma.relation(gamma.fits)
print(gamma.linear.fit.new)
#>   parameter     intercept       meanUMI
#> 1        p1  0.1287148420 -1.039842e-05
#> 2     mean1 -0.0014386996  1.085450e-05
#> 3     mean2  0.0116320294  6.967235e-04
#> 4       sd1 -0.0008401048  2.217179e-05
#> 5       sd2  0.0477075538  2.069985e-03
#> 6        p3  0.0509972852  0.000000e+00
```

Estimation of median dispersion function for each cell type

For the dispersion parameter, no relation with the UMI counts was found. Therefore, simply the median

value over all subsampling runs is taken for each parameter of the mean-dispersion function.

```
disp.fun.general.new<-dispersion.function.estimation(disp.param)
print(disp.fun.general.new)
#>   asymptDisp extraPois
#> 1 0.06854555 0.1783971
```

Annotation of cell type for all fitted data frames

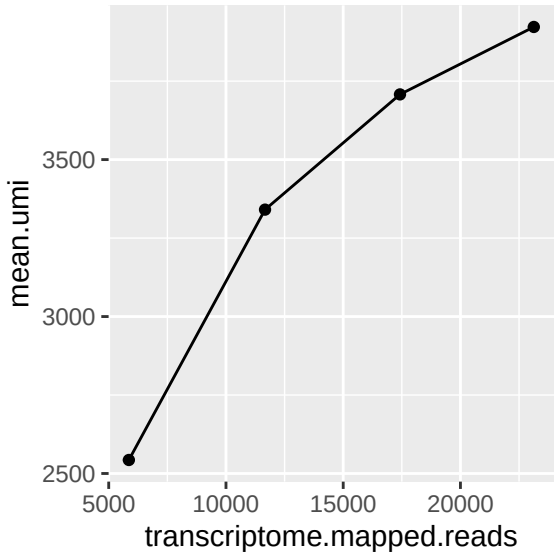
Multiple fits, e.g. from different cell types or other cell states, can be combined in the gamma fit and disperion fit data frames. To extract the correct fits for a specific cell type in the power calculation later, an additional column ct is required.

```
gamma.linear.fit.new$ct<-"New_ct"
disp.fun.general.new$ct<-"New_ct"
```

Fitting a functions for UMI counts dependent on read depth

As a last point, the relationship between reads and UMIs is fitted logarithmically. Therefore, the number of mapped reads is necessary, which can be gained from summary statistics after the mapping (e.g. in the cellranger summary statistics).

```
#Number of mapped reads taken from cellranger summary statistics
mapped.reads<-data.frame(matrix=c("complete","subsampled75","subsampled50","subsampled25"),
                         transcriptome.mapped.reads=c(23130,17422,11666,5859))

#Plot relationship between mean reads per cell and mean UMI per cell
read.umis<-merge(umi.values,mapped.reads,by="matrix")
print(read.umis)
#>         matrix mean.umi transcriptome.mapped.reads
#> 1     complete 3922.559                      23130
#> 2 subsampled25 2543.046                       5859
#> 3 subsampled50 3340.644                      11666
#> 4 subsampled75 3707.690                      17422
ggplot(read.umis,aes(x=transcriptome.mapped.reads,y=mean.umi))+
  geom_point()+geom_line()
```



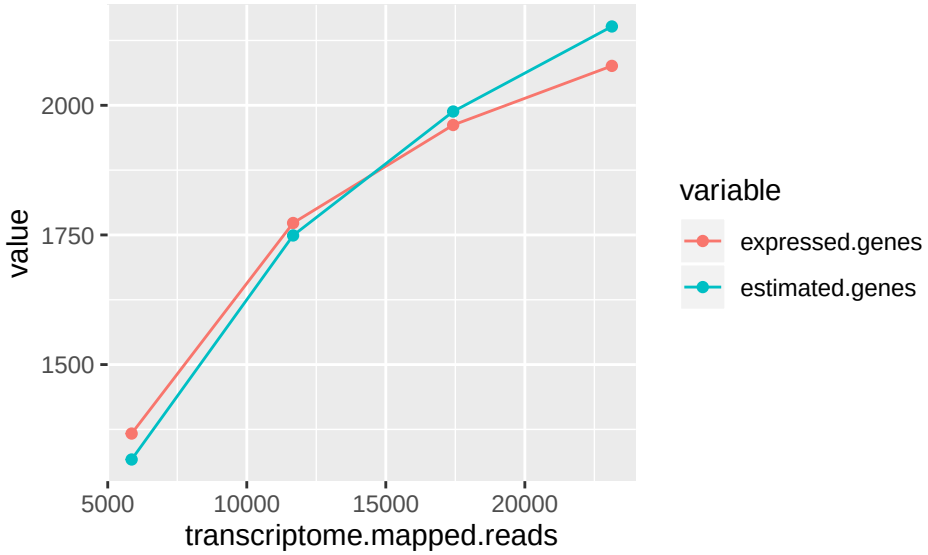
```
#Fit relationship between mean reads per cell and mean UMI per cell
read.umi.fit.new<-umi.read.relation(read.umis)
print(read.umi.fit.new)
#>             intercept    reads
#> (Intercept) -6223.204 1014.588
```

Validation of expression probability model

In the section "Counting observed expressed genes", the number of expressed genes in our example count matrices were calculated and saved in the data frame "expressed.genes.df". To validate our model, we try to predict the same number of expressed genes using our fitted model.

```
#Merge the observed numbers of expressed genes with the read depth
expressed.genes.df<-merge(expressed.genes.df,mapped.reads,
                          by="matrix")

#Get the number of cells per cell type and individual
expressed.genes.df$cells.indiv<-expressed.genes.df$num.cells/14
expressed.genes.df$estimated.genes<-NA
for(i in 1:nrow(expressed.genes.df)){

  #Vector with the expression probability for each gene
  expr.prob<-estimate.exp.prob.param(nSamples=14,
                                     readDepth=expressed.genes.df$
                                       transcriptome.mapped.reads[i],
                                     nCellsCt=expressed.genes.df$cells.indiv[i],
                                     read.umi.fit = read.umi.fit.new,
                                     gamma.mixed.fits = gamma.linear.fit.new,
                                     ct="New_ct",
                                     disp.fun.param=disp.fun.general.new,
                                     min.counts = 3,
                                     perc.indiv = 0.5)

  #Expected number of expressed genes
  expressed.genes.df$estimated.genes[i]<-round(sum(expr.prob))

}

print(expressed.genes.df)
#>          matrix num.cells expressed.genes transcriptome.mapped.reads cells.indiv
#> 1     complete       410            2076                       23130    29.28571
#> 2 subsampled25       409            1367                        5859    29.21429
#> 3 subsampled50       410            1773                       11666    29.28571
#> 4 subsampled75       410            1962                       17422    29.28571
#>   estimated.genes
#> 1            2152
#> 2            1317
#> 3            1749
#> 4            1988

plot.expressed.genes.df<-reshape2::melt(expressed.genes.df,
                                        id.vars=c("matrix","num.cells","cells.indiv",
                                                  "transcriptome.mapped.reads"))
```

```
ggplot(plot.expressed.genes.df,aes(x=transcriptome.mapped.reads,y=value,
                                   color=variable))+
  geom_point()+geom_line()
```



The estimated numbers of expressed genes match already in the small example data set quite well the observed numbers. A larger data set with more observations improves of course the fit in general. For a proper validation, the fits need to be validated not only on the training data set used for the fits, but also on an independent test data set. This is performed in our publication.

Use of expression probability model for power calculation

The fitted values can adapt the power calculation to the specific conditions of the experiment, e.g. a specific cell type of interest. The new data frames are added as parameters instead of the standard example fits of the package. The cell type is set to "New_ct", as saved in the dispersion and gamma fit data frames above.

```
power<-power.general.restrictedDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                        ct.freq=0.2,type="eqtl",
                                        ref.study=scPower::eqtl.ref.study,
                                        ref.study.name="Blueprint (Monocytes)",
                                        cellsPerLane=20000,
                                        read.umi.fit = read.umi.fit.new,
                                        gamma.mixed.fits = gamma.linear.fit.new,
                                        ct="New_ct",
                                        disp.fun.param=disp.fun.general.new,
                                        mappingEfficiency = 0.8,
                                        min.UMI.counts = 3,
                                        perc.indiv.expr = 0.5)

print(power)
#>                  name powerDetect exp.probs     power sampleSize totalCells
#> 1 Blueprint (Monocytes)   0.3233566 0.5105688 0.6390572        100       1500
#>   usableCells multipletFraction ctCells readDepth readDepthSinglet
#> 1        1276          0.149565     255     25000         22272.64
#>   mappedReadDepth expressedGenes
#> 1        17818.11           7454
```

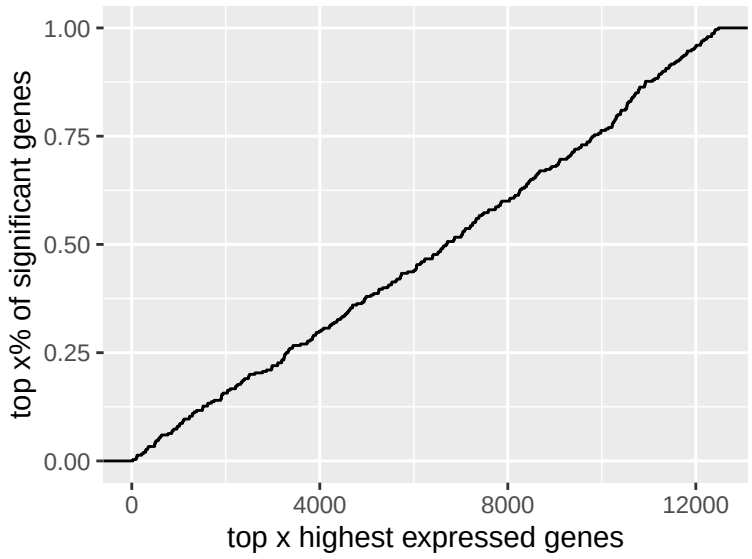
Effect sizes and expression ranks

To use other effect size and expression rank priors either data from other DE / eQTL studies or simulated data can be taken.

Priors from DE / eQTL studies

Effect sizes from other studies can be taken directly from the summary statistics of the study (FoldChange for DE genes and R^2 value for eQTL genes). A function of scPower extracts the gene ranks, if the normalized (!) count matrix is provided and a vector with the names of the significante DE / eQTL genes. To show the function, an example count matrix of scPower is processed in the following. For normalization, a simple normalization by counts per cell is done here. Depending on the data, other normalization strategies, e.g. incorporating also the gene length, might be more appropriate.

```
data(countMatrixExample)
#Choose one of the example matrices given by scPower
example.matrix<-count.matrix.example[["complete"]]

#To speed up computation, remove all 0 genes
example.matrix<-example.matrix[rowSums(example.matrix)>0,]

#Normalize by count per cell
example.matrix<-t(t(example.matrix)/colSums(example.matrix))

#Randomly select a fraction of the gene to be significant DE genes
sign.genes<-rownames(example.matrix)[1:300]

#Calulate gene ranks
gene.ranks<-gene.rank.calculation(example.matrix,sign.genes)

head(gene.ranks)
#>                     gene_symbol cumFraction rank
#> ENSG00000142676 ENSG00000142676 0.001919232   24
#> ENSG00000116251 ENSG00000116251 0.007357057   92
#> ENSG00000169442 ENSG00000169442 0.008556577  107
#> ENSG00000162511 ENSG00000162511 0.009036385  113
#> ENSG00000142669 ENSG00000142669 0.016873251  211
#> ENSG00000198830 ENSG00000198830 0.019912035  249

#Plot cumulative density function of significant genes
ggplot(gene.ranks, aes(rank)) + stat_ecdf(geom = "step") +
  xlab("top x highest expressed genes") +ylab("top x% of significant genes")
```



Simulated priors

If a data from a suitable study is not available or extreme cases, such as very small or very large effect sizes, shall be tested, the priors can also be simulated using functions of scPower. Gene ranks can be either sampled randomly or if an uniform distribution in a certain interval is preferred, generated with the function uniform.ranks.interval.

For eQTL genes, the R^2 value are simulated by transformation of Z-Scores using the Fisher Z Transformation. The Z-Scores are sampled from a normal distribution with parameters mean and sd.

```
#Uniform distributed gene ranks for 200 genes in the interval from 5,001-10,000
ranks<-uniform.ranks.interval(start=5001,end=10000,numGenes=200)
#Simulation of fold changes
Rsq<-effectSize.eQTL.simulation(mean=0.5,sd=0.2,numGenes=200)
#> Loading required package: HardyWeinberg
#> Warning: package 'HardyWeinberg' was built under R version 3.5.2
#> Loading required package: mice
#> Warning: package 'mice' was built under R version 3.5.2
#>
#> Attaching package: 'mice'
#> The following objects are masked from 'package:BiocGenerics':
#>
#>     cbind, rbind
#> The following objects are masked from 'package:base':
#>
#>     cbind, rbind
#> Loading required package: Rsolnp

simulated.eqtl.genes<-data.frame(ranks=ranks,
                                 Rsq=Rsq,
                                 name="Simulated")

power<-power.general.withDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                  ct.freq=0.2,type="eqtl",
                                  ref.study=simulated.eqtl.genes,
                                  ref.study.name="Simulated",
```

```
                                    samplesPerLane=4,
                                    read.umi.fit = scPower::read.umi.fit,
                                    gamma.mixed.fits = scPower::gamma.mixed.fits,
                                    ct="CD14+ Monocytes",
                                    disp.fun.param=scPower::disp.fun.param,
                                    mappingEfficiency = 0.8,
                                    min.UMI.counts = 3,
                                    perc.indiv.expr = 0.5)

print(power)
#>        name powerDetect exp.probs     power sampleSize totalCells usableCells
#> 1 Simulated   0.5696595   0.62118 0.9191417        100       1500        1431
#>   multipletFraction ctCells readDepth readDepthSinglet mappedReadDepth
#> 1           0.04602     286     25000         24091.28        19273.02
#>   expressedGenes
#> 1           8094
```

For DE genes, the log fold changes are simulated from a normal distribution with parameters mean and sd and transformed to fold changes.

```
#Uniform distributed gene ranks for 200 genes in the interval from 5,001-10,000
ranks<-uniform.ranks.interval(start=5001,end=10000,numGenes=200)
#Simulation of fold changes
foldChange<-effectSize.DE.simulation(mean=2,sd=0.5,numGenes=200)

simulated.de.genes<-data.frame(ranks=ranks,
                               FoldChange=foldChange,
                               name="Simulated")

power<-power.general.withDoublets(nSamples=100,nCells=1500,readDepth=25000,
                                  ct.freq=0.2,type="de",
                                  ref.study=simulated.de.genes,
                                  ref.study.name="Simulated",
                                  samplesPerLane=4,
                                  read.umi.fit = scPower::read.umi.fit,
                                  gamma.mixed.fits = scPower::gamma.mixed.fits,
                                  ct="CD14+ Monocytes",
                                  disp.fun.param=scPower::disp.fun.param,
                                  mappingEfficiency = 0.8,
                                  min.UMI.counts = 3,
                                  perc.indiv.expr = 0.5)

print(power)
#>        name powerDetect exp.probs     power sampleSize totalCells usableCells
#> 1 Simulated   0.6161472   0.62118 0.9920167        100       1500        1431
#>   multipletFraction ctCells readDepth readDepthSinglet mappedReadDepth
#> 1           0.04602     286     25000         24091.28        19273.02
#>   expressedGenes
#> 1           8094
```

Part 3: Power estimation to detect a sufficent number of cells in a specific cell types

To perform the cell type specific DE and eQTL analysis described in part 1 and 2, a sufficient number of cells of the target cell type needs to be detected for each individual. The package provides to functions, either to calculate the power for a specific parameter combination or to calculate the number of cells per sample to reach a certain power. In both cases, important parameters are the cell type frequency of the cell type of interest, which can be gained for example by literature research, here set to 2%. The minimal number of cells which should be detected was set to 10 and the sample size to 100.

In case of the power calculation, we assume to measure 1000 cells per individual. This leads to a detection power of:

```
scPower::power.detect.celltype(nCells=1000,min.num.cells = 10,
                               cell.type.frac=0.02,nSamples=100)
#> [1] 0.6255188
```

In case of estimating the number of cells, we set the detection threshold to 0.95, i.e. the functions returns the number of cells to reach a detection power of at least 95%. The required number of cells per person is then:

```
scPower::number.cells.detect.celltype(prob.cut=0.95,min.num.cells = 10,
                               cell.type.frac=0.02,nSamples=100)
#> [1] 1179
```