flowQA: A package that provides automated flow data quality assessment based on gated cell populations

Mike Jiang,Greg Finak

February 16, 2012

Abstract

Background The current flowQ package does the quality assessment on the ungated FCM data.However,there is need to identify deviant samples by monitoring the consistencies of the underlying statistical properties of different gated cell populations(such as white blood cells,lymphacytes,monocytes etc). The current package was also not designed for dealing with large datasets. To meet these needs, We developed flowQA package using the gating template created in flowJo and performing QA checks on different gated populations. It divides the data preprocessing from the actual outlier detection process so that the statistics are calcuated all at once and the outlier detections and visualization can be done more efficiently and interactively. ncdfFlow is used to solve the memory limit issue for large datasets.

keywords Flow cytometry,Quality Assessment, high throughput,svg,flowWorkspace,ncdfFlowSet

1 Parsing the QA gating template

Optionally, parsing xml workspace can be done in parallel mode. It can speed up the process for large datasets.

```
> library(Rmpi)
> library(multicore)
> library(ncdfFlow)
```

ncdfFlow also needs to be loaded in order to support netCDF storage for flow data that will solve the limitation of memory issue,

`parseWorkspace` function from flowWorkspace package is used to parse the flowJo workspace. If Rmpi and multicore package are loaded and it will automatically switch to the parallel mode and `nslaves` arugment is used to specify the number of computing nodes used.

```
> ws<-openWorkspace("~/QA_MFI_RBC_bounary_eventsV3.xml")
> G<-parseWorkspace(ws,execute=TRUE,isNcdf=TRUE,nslaves=6)
> saveNcdf("G","gatingHierarchy")
> save(G,file="gatingHierarchy/GS.Rda")
```

The result G is a `GatingSet` containing multiple `GatingHierarchy` within which gated cell populations are stored. Note that this step is most time consuming especially for large datasets. So it is convienient to save the gatingset once the parsing is done so that it be loaded directly from disk later on for the further processing

2 Calculating the statistics

This is the second preprocessing step followed by parsing gating template from flowJo workspace. Firstly,we need to save the gating hierarchies are calculated and the sample annoation data (containing all the meta information about the FCS files and samples) into a global environment.

```
> anno<-read.csv("~/FCS_File_mapping.csv")###read annotation data
> db<-new.env()
> saveToDB(db,G,anno)
```

Then statistics of each gated population is extracted and saved in db.Again,`getQAStats` can be speeded up by running in parallel mode. It uses parallel package and automatically detection the number of computing nodes available. Optionally `nslaves` arugment can also be provided to manually specify the computing node.

```
> library(parallel)
> getQAStats(db)
> ls(db)
> db$statsOfGS[1:5,]
```

It is recommened to save all the preprocessed data to avoid the efforts of recomputing from the beginning:

```
> save(db,file="ITN029.rda")#save stats
```

Once this is done,the more interactive quality assessment task can be performed based on the statistics extracted for each gated population.

3 Defining qaTasks

We provide a function to create a list of *qaTask* objects by reading external csv spreadsheet containing descriptions of each QA task:

```
> data("ITN029")
> checkListFile<-file.path(system.file("data",package="flowQA"),"qaCheckList.csv")
> qaTask.list<-makeQaTask(db,checkListFile)
> qaTask.list[1:2]

$MFIOverTime
qaTask: MFIOverTime
Level : Assay
Description : Fluorescence stability over time
Plot type:  xyplot
Gated node:  MFI
Default formula :MFI ~ RecdDt | channel * stain
<environment: 0x4d7f6670>
```

```
$NumberOfEvents
qaTask: NumberOfEvents
Level : Tube
Description : Number of Events Collected
Plot type:  xyplot
Gated node:  Total
Default formula :count ~ RecdDt | Tube
<environment: 0x4d79f568>
```

This is a convenient way to construct multiple *qaTask*s. Users can also create the individual *qaTask* by using `new` method.

4 Quality assessment and visualiztion

The package provides two important methods:`qaCheck` and `plot` to perform quality assessment and visualize the QA results. They both use the information stored in `qaTask` object and the `formula`, which is given either explicitly by the argument or implicitly by the `qaTask` object. It is generally of the form y $\sim$ x | g1 * g2 * ... ,y is the statistics to be checked in this QA, It must be one of the four types:

"MFI": Median Fluorescence Intensity of the cell population specified by *qaTask*,

"percent": the percentage of the cell population specified by *qaTask* in the parent population,

"count": the number of events of the cell population specified by *qaTask*,

"spike": the variance of intensity over time of each channel ,which indicating the stability of the fluorescence intensity.

x specifies the variable plotted on x-axis (such as date) in `plot` method.

g1,g2,.... are the conditioning variables, which divide the data into subgroups and apply the outlier detection whitin each individual groups or plot them in different panels. They may also be omitted,in which case the outliers detection is peformed in the entire dataset.

For example, RBC Lysis efficiency(percentage of WBC population) check is defined by *qaTask* .

```
> qaTask.list[["RBCLysis"]]
```

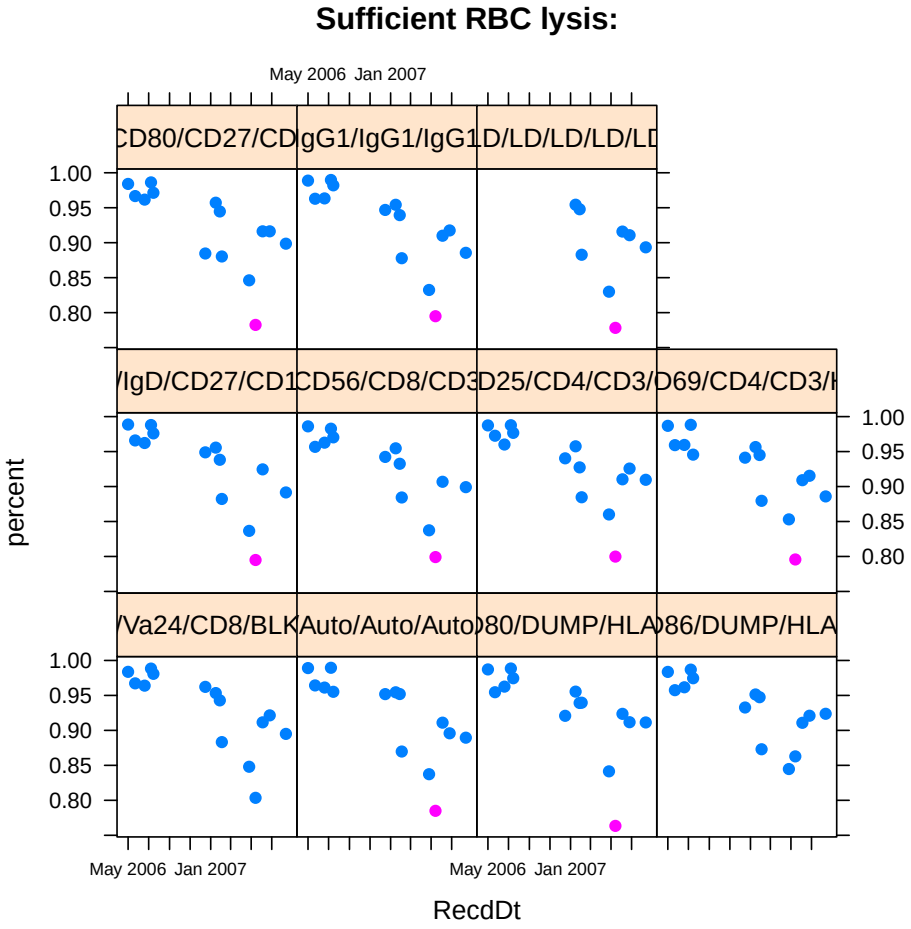
```
qaTask: RBCLysis
Level : Tube
Description : Sufficient RBC lysis
Plot type:  xyplot
Gated node:  WBC_perct
Default formula :percent ~ RecdDt | Tube
<environment: 0x4d6a7488>
```

According to the formula stored in *qaTask*, it uses the statistical property "percent" and groups the data by "Tube"(or staining panel). "RecdDt" is reserved for plotting purpose. Cell population is defined as "WBC_perct"

```
> qaCheck(qaTask.list[["RBCLysis"]],outlierfunc=outlier.cutoff,lBound=0.8)
```

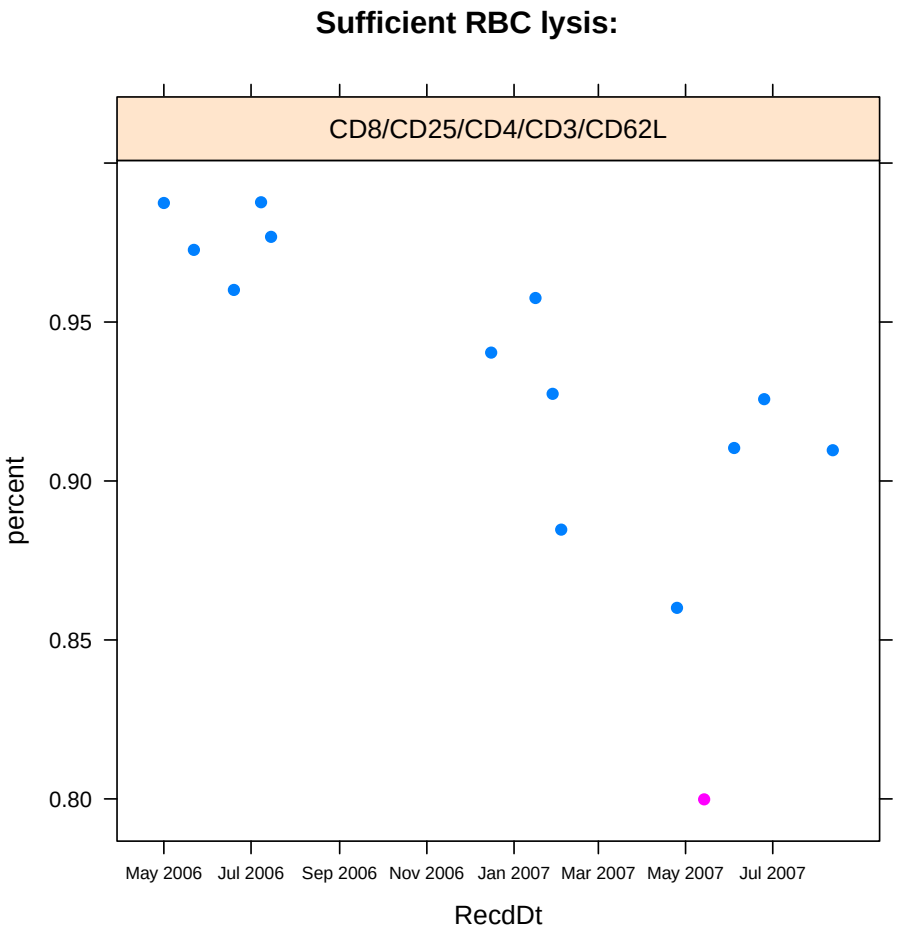
As we see,`qaCheck` reads all the necessary information about the gated data from *qaTask* object. The only thing needs to be specified by end users is how the outliers are called. This is done by providing an outlier detection function `outlierfunc` that takes a numeric vector as input and returns a logical vector as the output. Here "outlier.cutoff" is used and threshold "lBound"("less than",using uBound for "larger than") is specified.

```
> plot(qaTask.list[["RBCLysis"]])
```



By default all the data are plotted,argument "subset" can be used to visualize a small subset.

```
> plot(qaTask.list[["RBCLysis"]],subset="Tube=='CD8/CD25/CD4/CD3/CD62L'")
```



x term in the formula is normally ignored in `qaCheck`.However,when "plotType" of the `qaTask` is "bwplot", it is used as the conditioning variable that divides the data into subgroups within which the `outlierfunc` is applied.

```
> qaTask.list[["MNC"]]
```

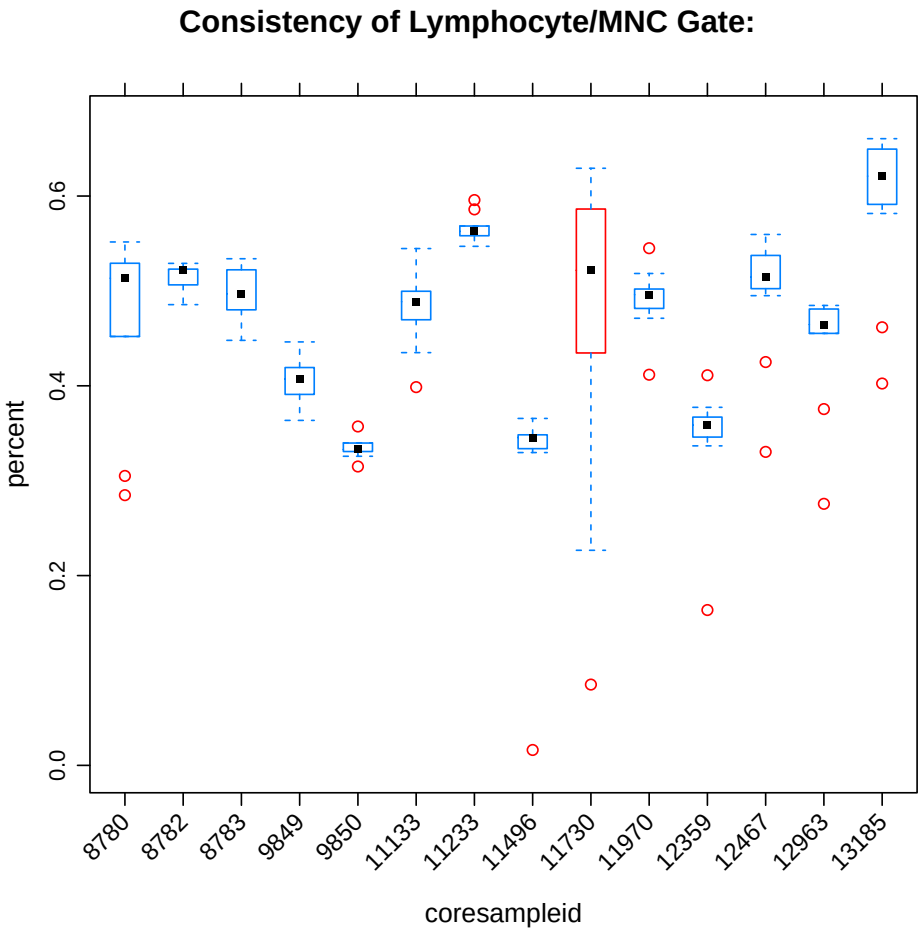
```
qaTask: MNC
Level : Assay
Description : Consistency of Lymphocyte/MNC Gate
Plot type:  bwplot
Gated node:  MNC
Default formula :percent ~ coresampleid
<environment: 0x4d73a850>
```

This qaTask detects the significant variance of MNC cell population percentage among aliquots,which have the same "coresampleid". Plot type of this object tells the method to group data by "coresampleid".

```
> qaCheck(qaTask.list[["MNC"]],z.cutoff=2)
```

Interquartile Range based outlier detection function is used to detect outliers

```
> plot(qaTask.list[["MNC"]])
```



The red circles in the boxplot indicate the possible outlier samples and the box of red color indicates the entire sample group has significant variance and is marked as the group outlier.

We can also apply simple aggregation to the statisics through the formula.

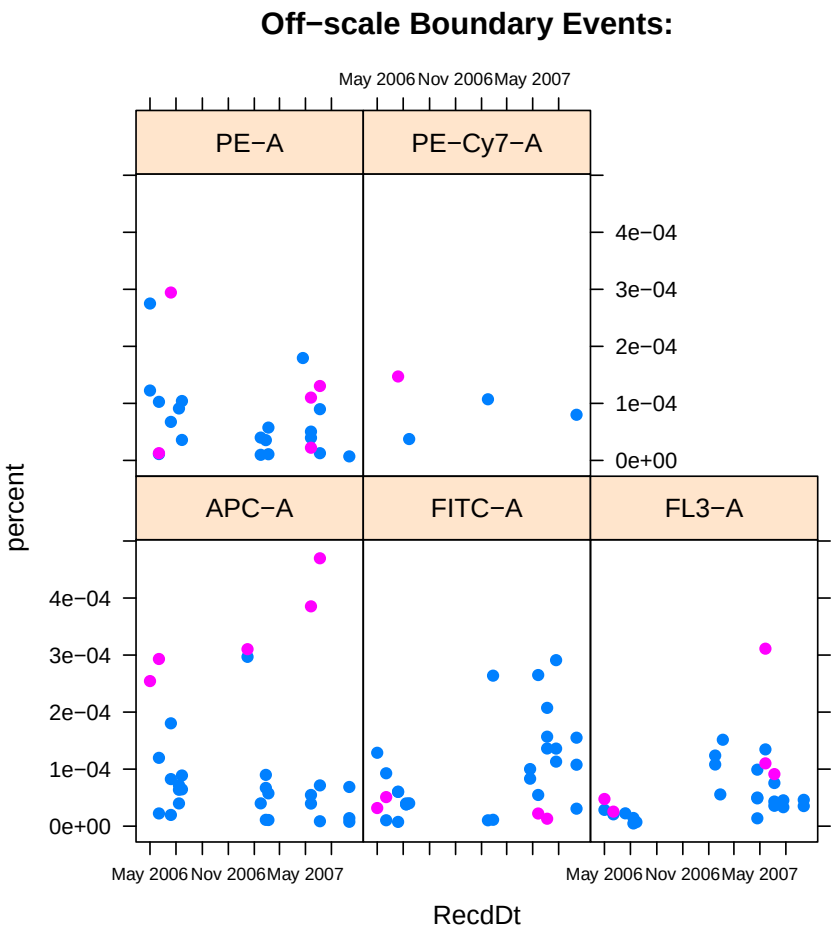
```
> qaTask.list[["BoundaryEvents"]]

qaTask: BoundaryEvents
Level : Channel
Description : Off-scale Boundary Events
Plot type:  xyplot
Gated node:  margin
Default formula :percent ~ RecdDt | channel
<environment: 0x4d5f33b8>
```

Here the default formula only extracts the "percent" from each individual channel. In order to check the total percentage of boundary events of all channels for each fcs file, we can write a new formula by applying aggregation function "sum" to "percent" and group the data by fcs file ("name" in this case).

```
> qaCheck(qaTask.list[["BoundaryEvents"]]
+                  ,sum(percent) ~ RecdDt | name
+                  ,outlierfunc=outlier.cutoff
+                  ,uBound=0.0003
+                  )
```

And we still can visualize the results chanel by chanel.

```
> plot(qaTask.list[["BoundaryEvents"]],percent ~ RecdDt | channel)
```

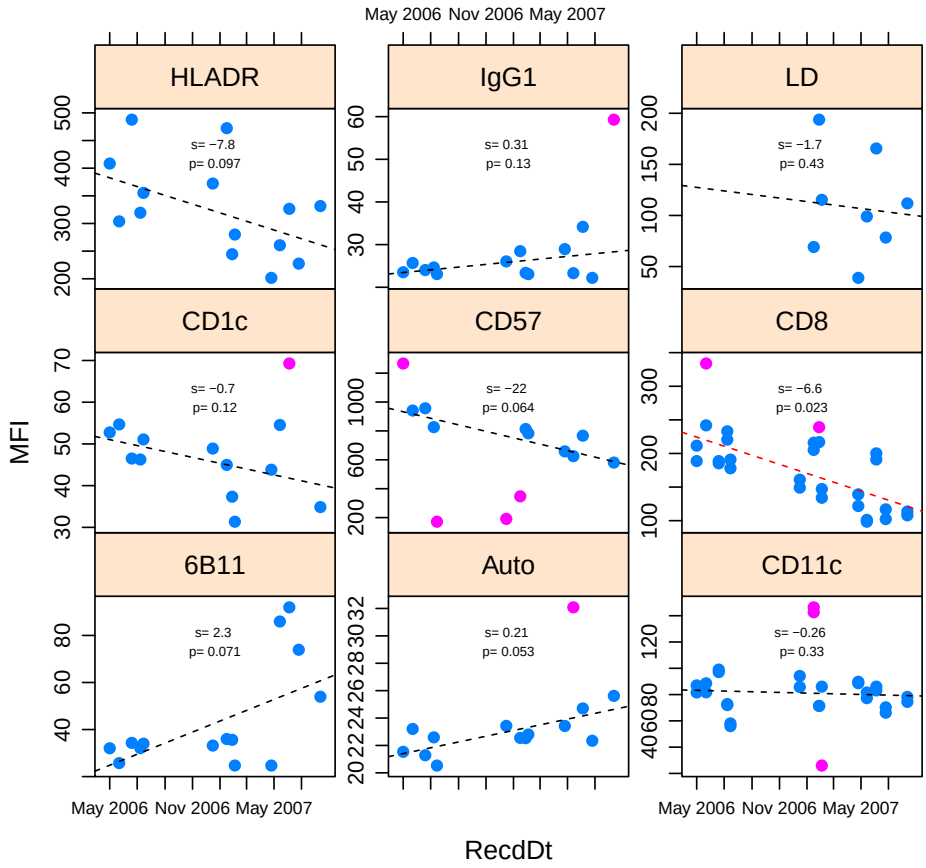


Another three examples: QA check of Fluorescence stability overtime using t-distribution based outlier detection function.

```
> qaCheck(qaTask.list[["MFIOverTime"]]
+                  ,rFunc=rlm
+                  ,z.cutoff=3
+                  )
> plot(qaTask.list[["MFIOverTime"]]
+                  ,y=MFI~RecdDt|stain
+                  ,subset="channel%in%c('FITC-A')"
```

```
+                   ,rFunc=rlm
+                   ,relation="free"
+          )
```

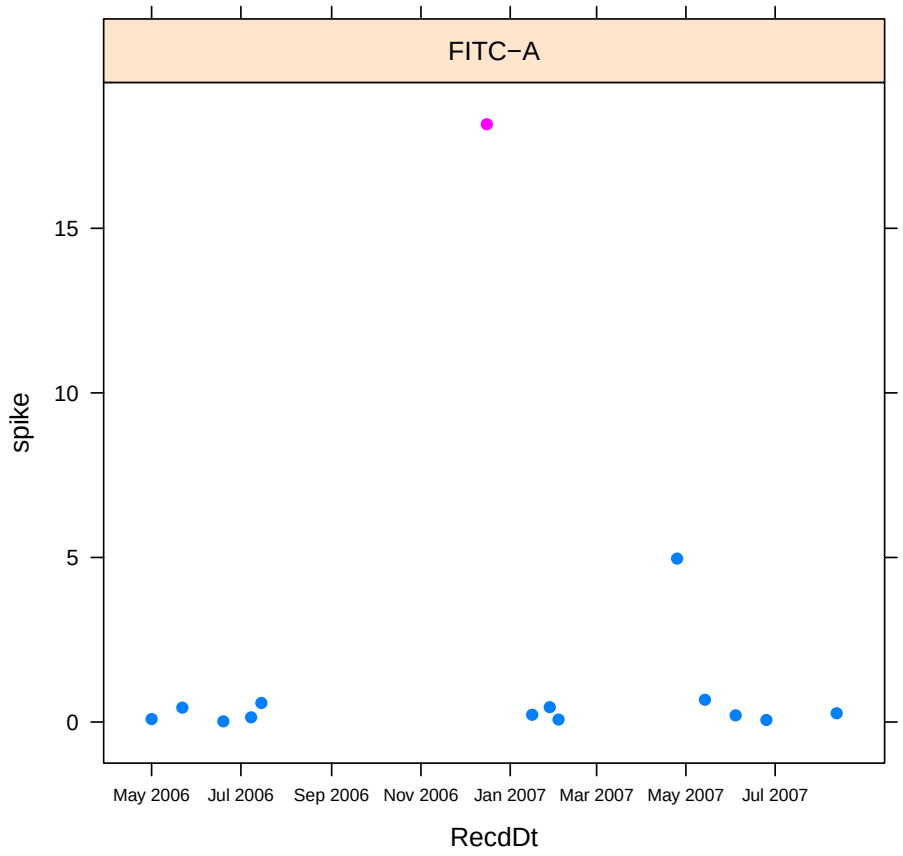
Fluorescence stability over time:



Note that the linear regression is applied in each group in order to capture the significant MFI change over time. The individual outliers within each group is detected based on the residue.

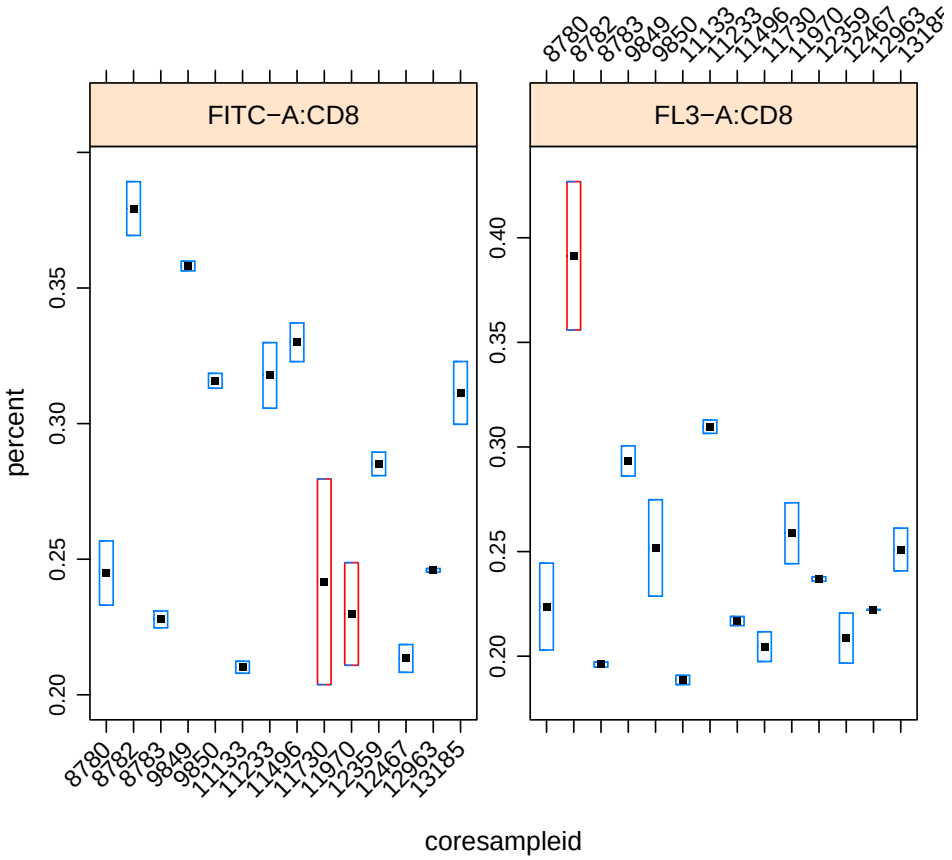
```
> qaCheck(qaTask.list[["spike"]]
+                         ,outlierfunc=outlier.t
+                         ,alpha=0.00001)
> plot(qaTask.list[["spike"]],y=spike~RecdDt|channel
+                   ,subset="Tube=='CD8/CD25/CD4/CD3/CD62L'&channel%in%c('FITC-A')"
+          )
>
```

Measurement spikes/drifts during acquisition:




```
> qaCheck(qaTask.list[["RedundantStain"]],z.cutoff=1)
> plot(qaTask.list[["RedundantStain"]]
+                 ,y=percent~coresampleid|channel:stain
+                 ,subset="stain%in%c('CD8')"
+         )
```

Consistency of redundant Staining Across sample aliquots:



5 Creating quality assessment report

Besides the interactive visualization provided by `plot` method,we also provide one routine to generate all plots in one report.This function reads the QA results calculated by `qaCheck` and the meta information of each QA task provided in spreadsheet qaCheckList and generate the summary tables and svg plots. Svg plots provide tooltips containing the detail information about each sample as and hyperlinks of densityplot for each individual FCS file.

```
> qa.report(db,outDir="~/output",plotAll=FALSE)
```

`plotAll` is the argument to control the plotting of the individual scatter plot for each FCS file. When TRUE, all the FCS files are plotted. If FALSE,only the FCSs marked as Outliers will be plotted. It can also be set to "none" meaning that no scatter plot will be generated, which provides the quick review of the html report.

6 Conclusion

By the formula-based `qaCheck` and `plot` methods,different QA tasks can be defined and performed in a generic way. And `plot` only reads the outliers detection results pre-calculated by `qaCheck`, which reduces the cost of interactive visualization.

Two kinds of lattice plots are currently supported:xyplot and bwplot(boxplot),depends on the `plotType` in `qaTask` object. When the output path is provided by `dest`, the svg plot is generated. In svg plot, each dot or box (or only the one marked as outliers) is annotated by the tooltip or hyperlink.which further points to the individual density plot of the gated population.