QUALIFIER: Quality Assessment of Gated Flow Cytometry Data

Mike Jiang,Greg Finak

July 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 QUALIFIER 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

`parseWorkspace` function from flowWorkspace package is used to parse the flowJo workspace. Since we are only interested in the gating strategy (gates defined in the template),`execute` can be set as FALSE to skip the actual gating process. `useInternal` is set as TRUE to use internal structure(c++) for the faster parsing.

```
> ws<-openWorkspace("~/QA_MFI_RBC_bounary_eventsV3.xml")
> GT<-parseWorkspace(ws,execute=FALSE,useInternal=TRUE)
```

2 Apply the gating template to the data

Then gating hierarchy `gh_template` containing the actual template gates is extracted from the result `GatingSet` "`GT`".

```
> gh_template<-GT[[1]]
```

The `GatingSet` method here is the contructor that takes a `gatingHierarchy` object as the template and and a list of FCS file names (`filenames`) that need the quality assurance. The result "G" is the `GatingSet` that contains the gated data and some of the cell population statistics that can be viewed by `getPopStats` method.

```
> ##datapath is the path where FCS files stores
> G<-GatingSet(gh_template,filenames,path=datapath)
> getPopStats(G[[1]])
```

Optionally,`isNcdf` can be set as TRUE to support netCDF storage of large flow datasets when memory resource is limited.

3 Calculating the statistics

After flow data is gated,the statistics of the gated data need to be extracted and saved before the QA checks. `db` is an environment that serves as a container storing all the QA data and results. Firsly ,`initDB` function initializes and creates the data structures for QA. Then `qaPreprocess` is a convenient wrapper that calls underlining routines (`getQAStats`,`saveToDB`) to calculates/extracts statistics of gated cell populations and save them along with the gating set and the extra FCS meta data.

```
> db<-new.env()
> initDB(db)
> qaPreprocess(db=db,gs=G
+                          ,metaFile=metaFile
+                          ,fcs.colname="FCS_Files"
+                          ,date.colname=c("RecdDt","AnalysisDt")
+         )
```

`fcs.colname` and `date.colname` are the arguments to identify the columns in the meta data (the flat table stored as a csv) that specifies the FCS filenames and the dates (to be formatted as to "%m/%d/%y"). `stats` in the data environment is the data frame that stores the statitics.

```
> head(get("stats",envir=db))
```

Once the preprocessing steps are finished,the data are ready for quality assessments.

4 Defining qaTasks

`read.qaTask` reads external csv spreadsheet that contains the descriptions of each QA task and creates a list of *qaTask* objects.

```
> checkListFile<-file.path(system.file("data",package="QUALIFIER"),"qaCheckList.csv.gz")
> qaTask.list<-read.qaTask(db,checkListFile=checkListFile)

[1] "7 qaTask created ahd saved in db!"

> qaTask.list[1:2]

$MFIOverTime
qaTask: MFIOverTime
Level : Assay
```

```
Description : Fluorescence stability over time
population:  MFI
Default formula :MFI ~ RecdDt | channel * stain
<environment: 0x7f209975ee28>
Plot type:  xyplot
$horiz
[1] FALSE


$NumberOfEvents
qaTask: NumberOfEvents
Level : Tube
Description : Number of Events Collected
population:  root
Default formula :count ~ RecdDt | Tube
<environment: 0x7f2087ac2778>
Plot type:  xyplot
$horiz
[1] FALSE
```

The *qaTask* can also created individually by the contructor `makeQaTask`.

5 Quality assessment and visualiztion

`qaCheck` and `plot` are the two main methods to perform the 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*,

"proportion": 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
population:  WBC_perct
Default formula :proportion ~ RecdDt | Tube
<environment: 0x7f1f854c2d88>
Plot type:  xyplot
$horiz
[1] FALSE
```

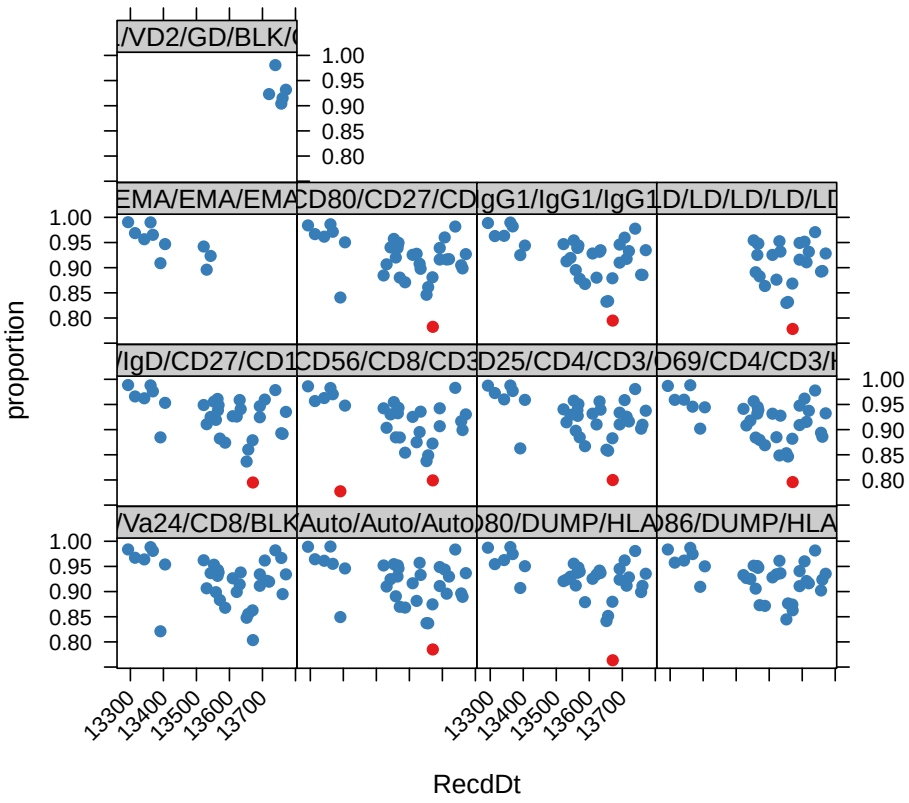
According to the formula stored in *qaTask*, it uses the statistical property "proportion" and groups the data by "Tube"(or staining panel). "RecdDt" is reserved for plotting purpose. Cell population is defined as "WBC_perct"

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

`qaCheck` reads all the necessary information about the gated data from *qaTask* object. Users only needs to specificy 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.

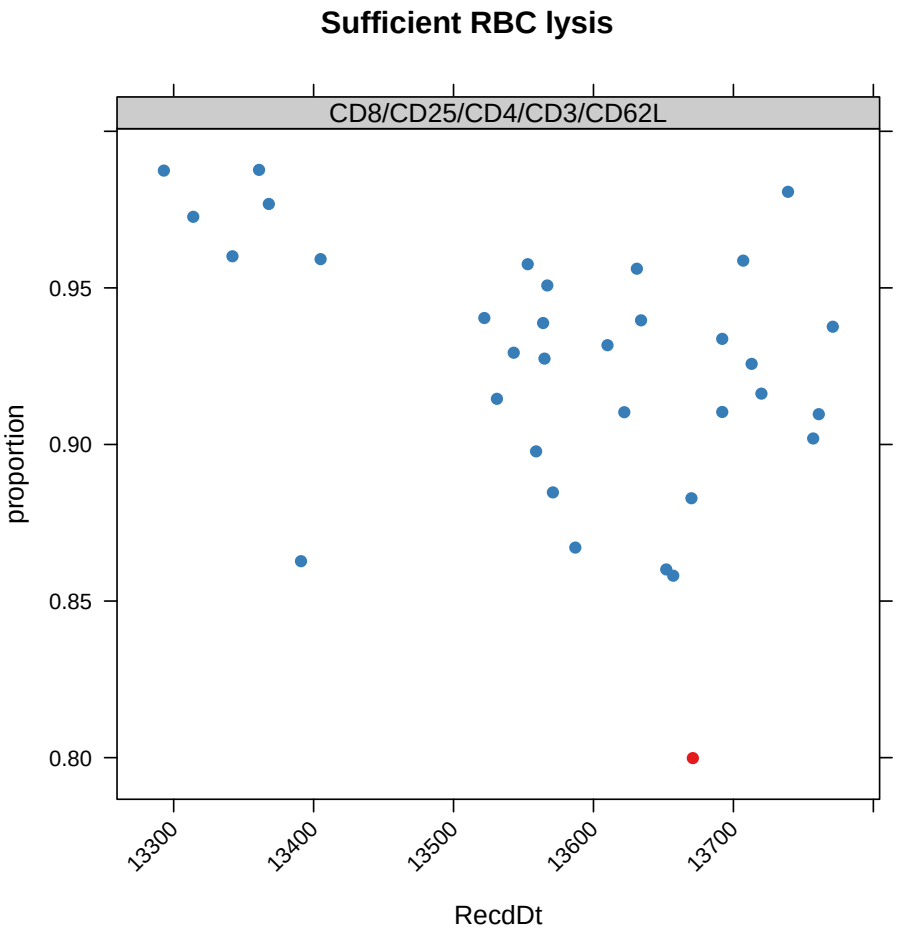
After the outliers are called, the results can be plotted by `plot` method.

```
> 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')
```

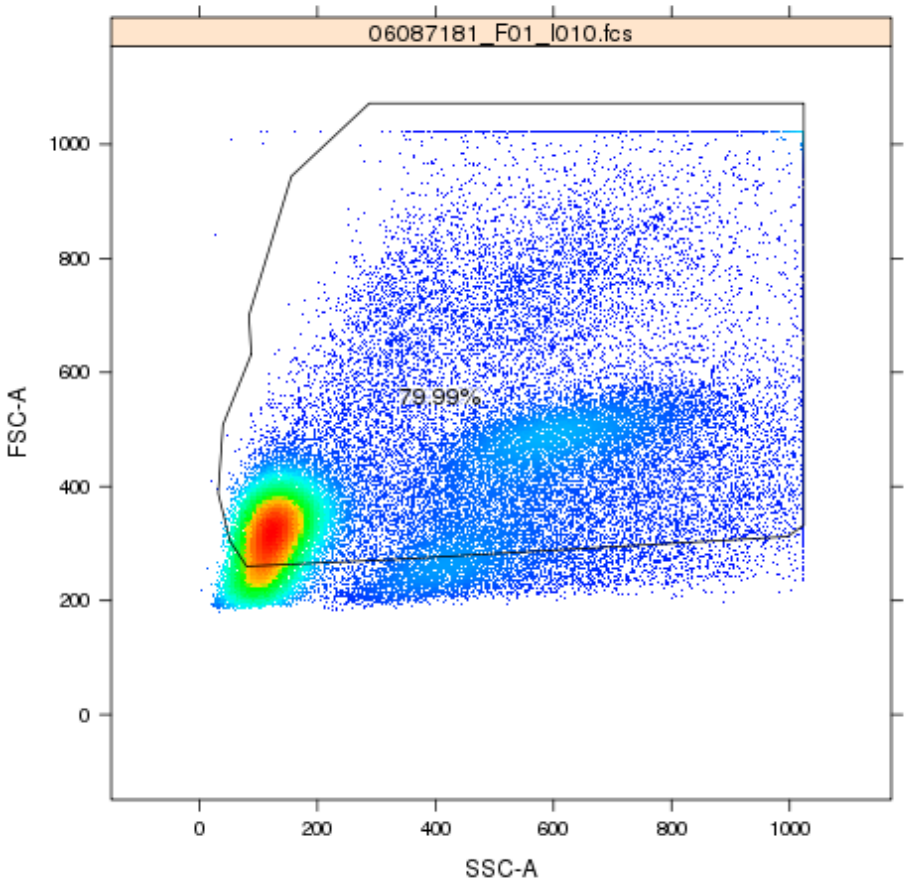


`clearCheck` is the method to removes the outlier results detected by the previous `qaCheck` call for the specific qaTask.

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

With `scatterPlot` flag set as true and `subset` properly specified plot method can generate scatter plots for the selected FCS files,

```
> plot(qaTask.list[["RBCLysis"]],subset=name=='06087181_F01_I010.fcs',scatterPlot=TRUE)
```



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.

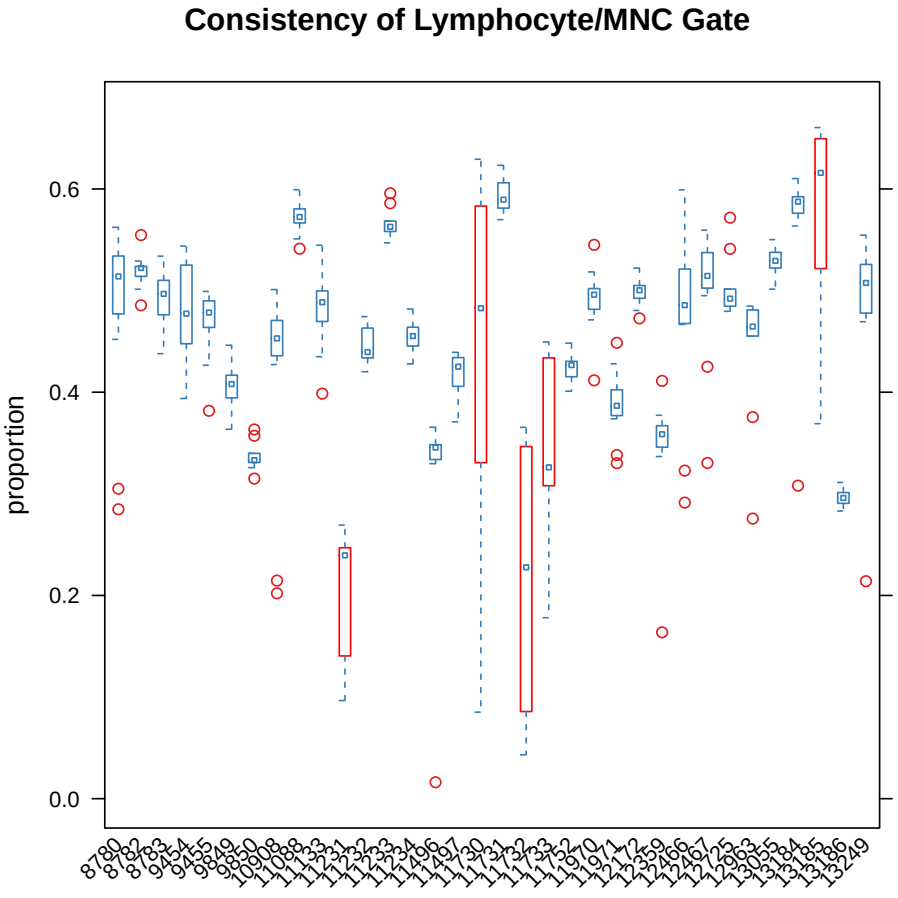
```
> qaTask.list[["MNC"]]

qaTask: MNC
Level : Assay
Description : Consistency of Lymphocyte/MNC Gate
population:  MNC
Default formula :proportion ~ coresampleid
<environment: 0x7f1ecc85edf8>
Plot type:  bwplot
$horiz
[1] FALSE
```

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=1.5)
```

Interquartile Range based outlier detection function is used to detect outliers

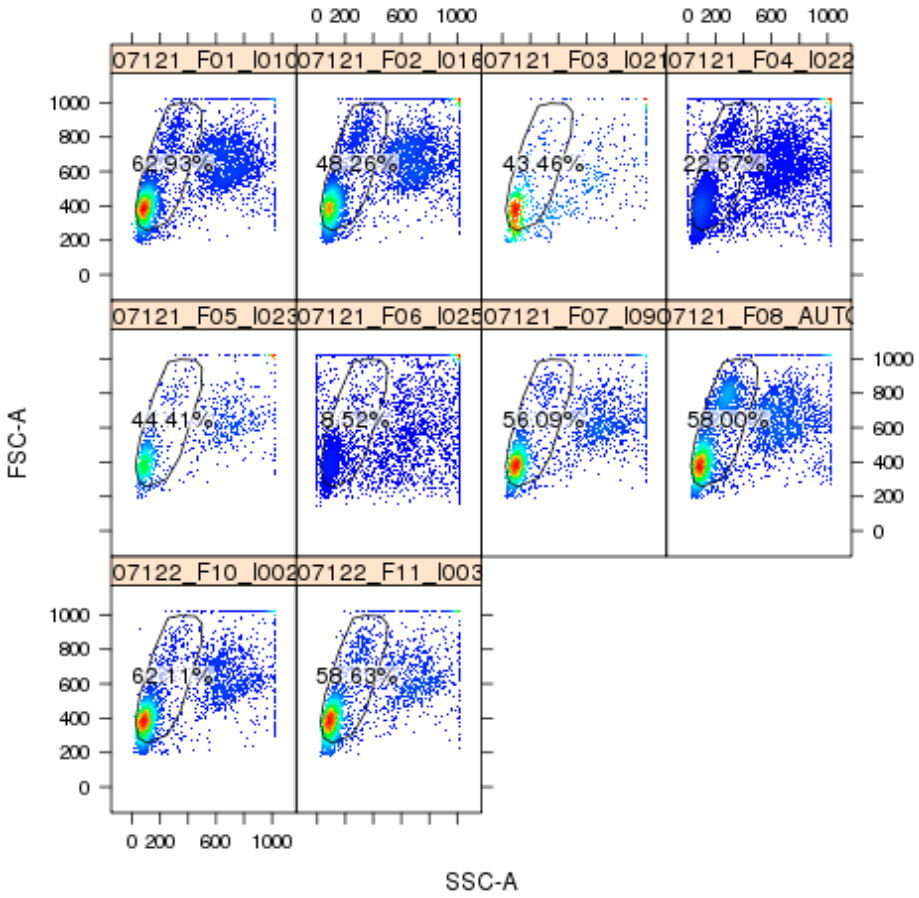
```
> plot(qaTask.list[["MNC"]],proportion~factor(coresampleid))
```



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. The formula supplied here in the plot method can overwrite the one stored in the qaTask object thus change the way of viewing the data.

Again,with `scatterPlot` and `subset` arguments, scatter plots can be generated for the selected FCS files or sample groups,

```
> plot(qaTask.list[["MNC"]]
+                 ,scatterPlot=TRUE
+                 ,subset=coresampleid==11730)
```



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

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

qaTask: BoundaryEvents
Level : Channel
Description : Off-scale Boundary Events
population:  margin
Default formula :proportion ~ RecdDt | channel
<environment: 0x7f1f6fd05b50>
Plot type:  xyplot
$horiz
[1] FALSE
```

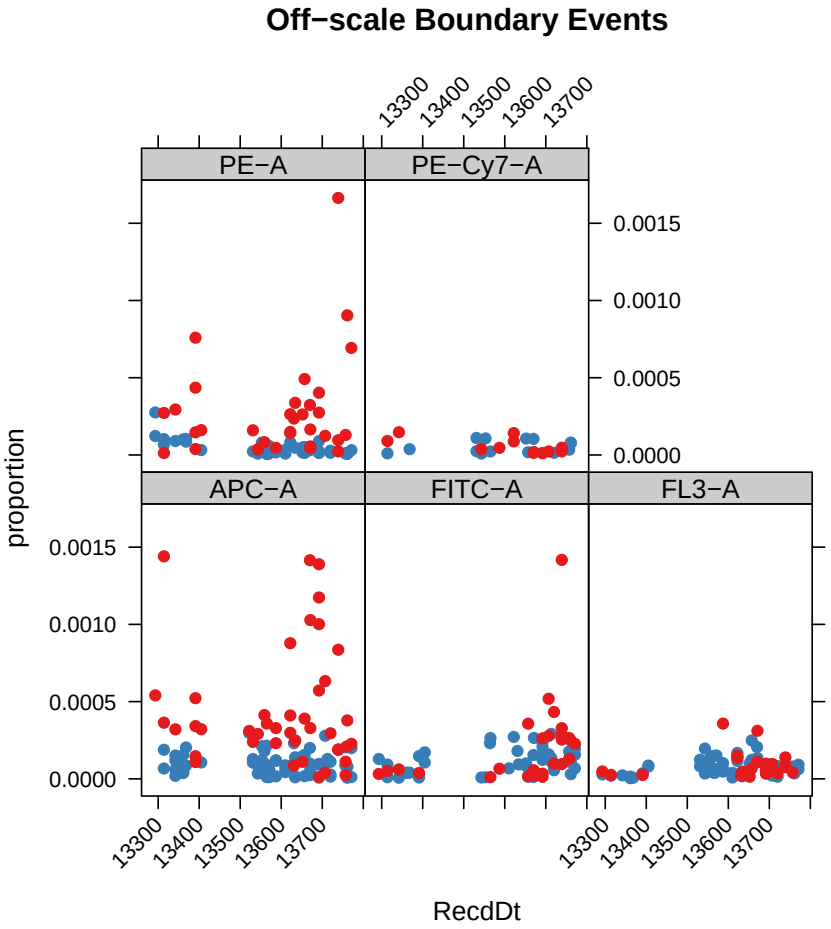
Here the default formula only extracts the ”proportion” from each individual channel. In order to check the total percentage of boundary events of all channels for each fcs file, we can write a new formula by applying aggregation function ”sum” to ”proportion” and group the data by fcs file (”name” in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

```
> qaCheck(qaTask.list[["BoundaryEvents"]]
+                 ,sum(proportion) ~ RecdDt | name
```

```
+                   ,outlierfunc=outlier.cutoff
+                   ,uBound=0.0003
+                   )
```

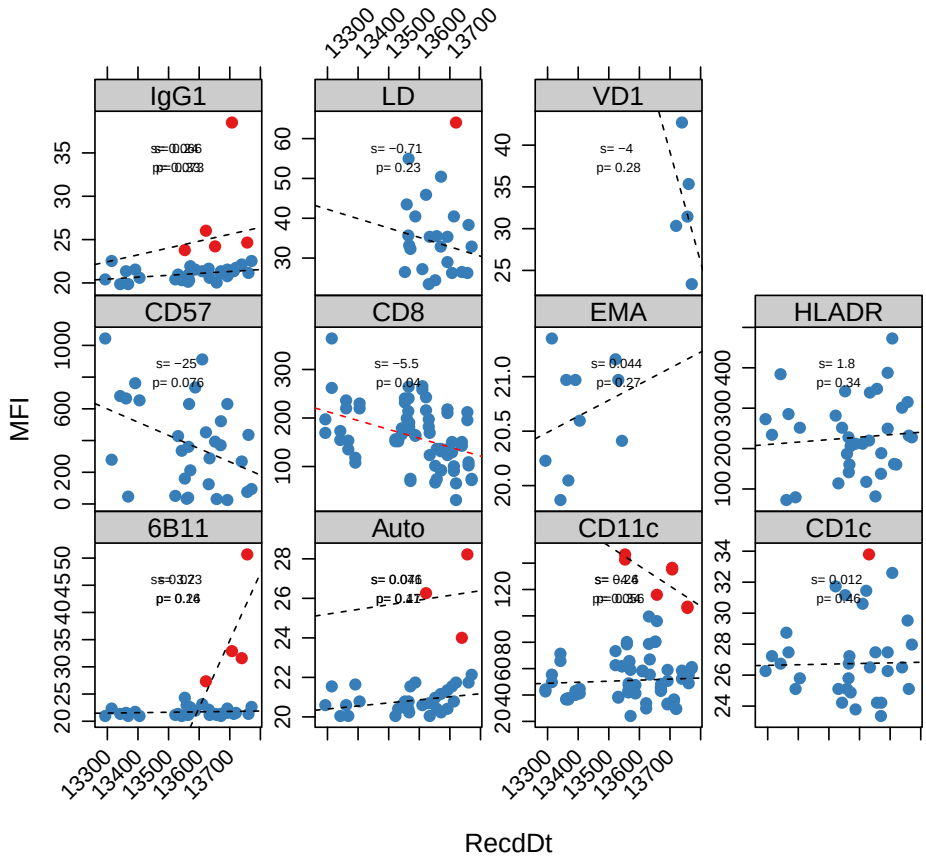
And we still can visualize the results chanel by chanel.

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



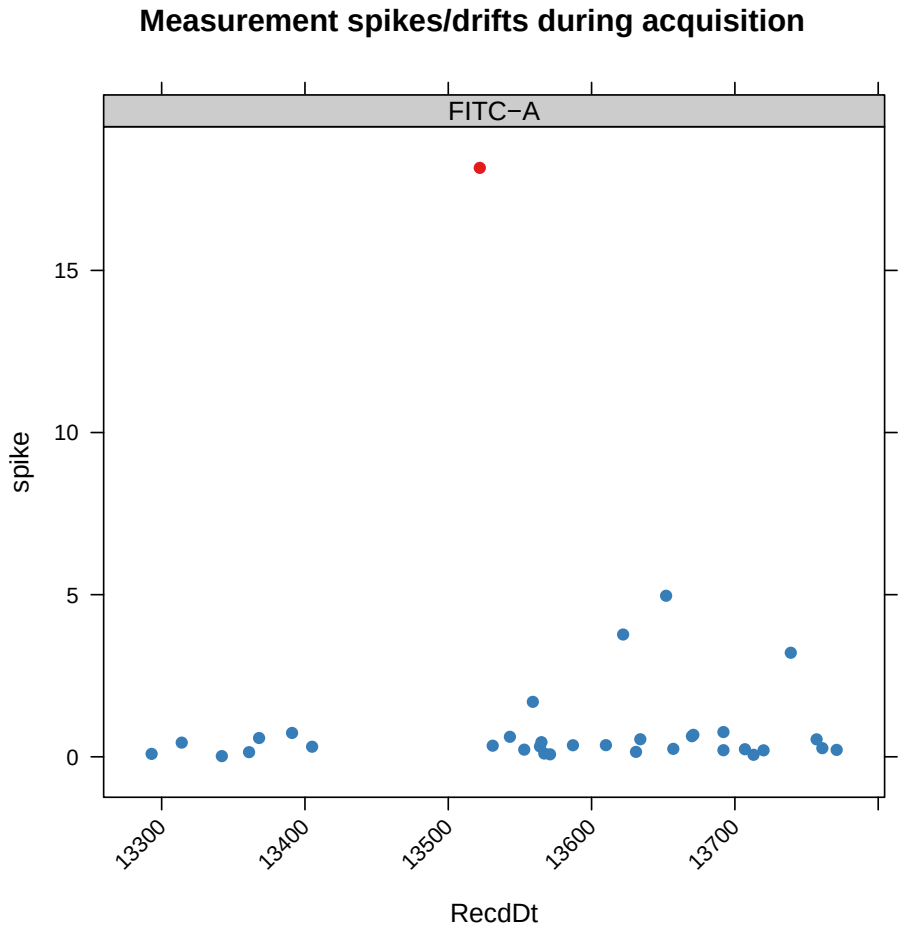
Another three examples: QA check of Fluorescence stability overtime using t-distribution based outlier detection function.

```
> 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
+                   ,scales=list(y=c(relation="free"))
+         )
```



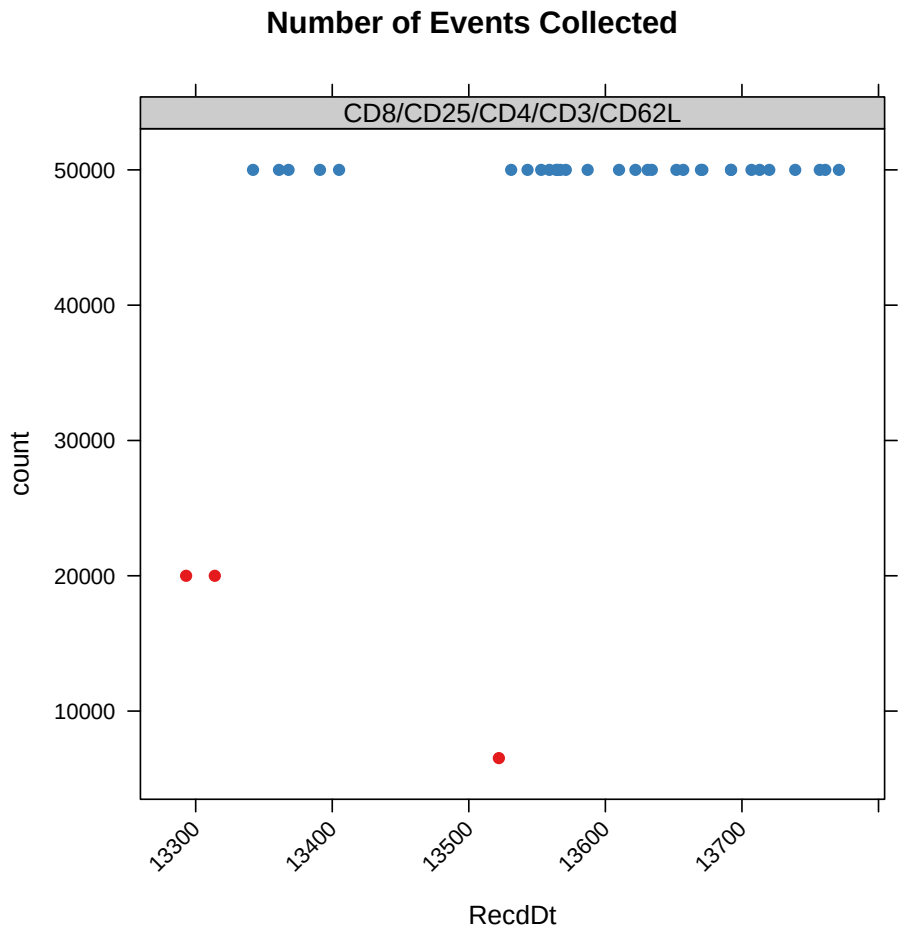
Note that the robust linear regression is applied in each group in order to capture the significant MFI change over time. The individual outliers within each group is detected based on the residue. All the lattice options (like `scales` here in this example) can be passed to control the apparance of trellis plot.

```
> 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')
+         )
>
```



When minitoring the total number of events for each tube, a pre-determined events number can be provided as the threshold to the qaCheck method.

```
> qaCheck(qaTask.list[["NumberOfEvents"]]
+                 ,formula=count ~ RecdDt | Tube
+                 ,outlierfunc=outlier.cutoff
+                 ,lBound=0.8*tubesEvents
+                 )

[1] "done!"

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



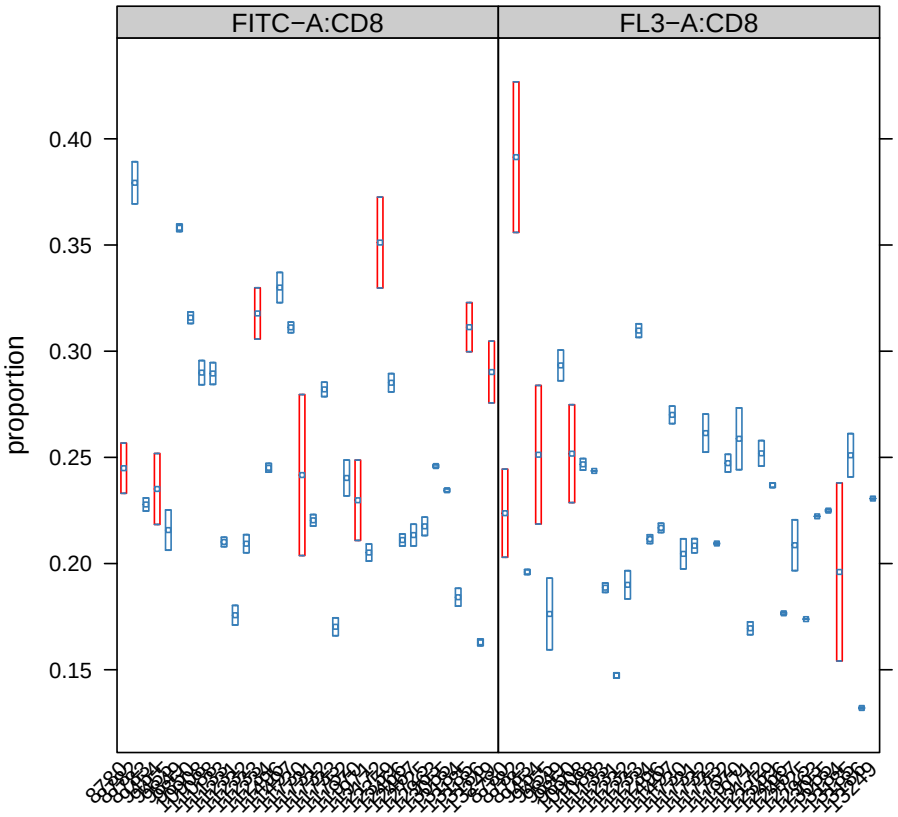
`tubesEvents` could be a one-column data frame or a named list/vector. Threshold values are stored in the column or list/vecor and conditioning values stored in rownames or names of the list/vector.

```
> tubesEvents
```

```
                           events
CD8/CD25/CD4/CD3/CD62L      50000
CD1c/IgD/CD27/CD19/IgM     150000
HLADR/CD80/CD27/CD19/CD86  150000
CD57/CD56/CD8/CD3/CD14      20000
6B11/Va24/CD8/BLK/CD4       50000
CD8/CD69/CD4/CD3/HLADR      20000
IgG1/IgG1/IgG1/IgG1/IgG1    20000
Auto/Auto/Auto/Auto/Auto    20000
CD11c/CD80/DUMP/HLADr/CD123 400000
CD11c/CD86/DUMP/HLADr/CD123 400000
LD/LD/LD/LD/LD              20000
VD1/VD2/GD/BLK/CD3          250000
```

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

Consistency of redundant Staining Across sample aliquots



6 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.

```
> qaReport(qaTask.list
+                 ,outDir="output"
+                 ,plotAll=FALSE
+                 ,subset=as.POSIXlt(RecdDt)$year==(2007-1900)
```

```
+                   )
>
```

`plotAll` is the argument to control the plotting of the individual scatter plot for each FCS file. IF TRUE, scatter plots for all the FCS files are generated. If FALSE,only the FCS files marked as outliers will be plotted. When it is set to "none", no scatter plot will be generated, which is helpful to provide a quick preview of the html report. `subset` is the filter to only generate the report on a subset of flow data.

Note that if there is need to adjust the QA plot for each indivdiual qaTask in the report, the arguments must be stored in qaTask before qaReport method is called. If there is no outlier is detected, the qaTask is not plotted by default. In order to change this setting, `htmlReport` can be set as TRUE to plot the qaTask regardlessly. This is particually useful for the task that trackes the longitudinal trend instead of the individual outliers. Besides the lattice arguments that can be configured by `qpar`(for the summary xyplot and bwplot) and `scatterPar` (for the individual FCS xyplot). xlog(ylog) flag can be set for scatter plot to transform the flow data to log scale for the proper displaying those stained channels.

```
> htmlReport(qaTask.list[["MFIOverTime"]])<-TRUE
> rFunc(qaTask.list[["MFIOverTime"]])<-rlm
> scatterPar(qaTask.list[["MFIOverTime"]])<-list(xlog=TRUE)
> scatterPar(qaTask.list[["BoundaryEvents"]])<-list(xlog=TRUE)
> scatterPar(qaTask.list[["RedundantStain"]])<-list(xlog=TRUE)
> qpar(qaTask.list[["RedundantStain"]])<-list(scales=list(x=list(relation="free")))
```

The complete example of the report can be viewed at http://mikejiang.github.com/QUALIFIER/qaReport/ne

7 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.