flowCore: data structures package for the analysis of flow cytometry data

N. Le Meur   F. Hahne   D. Sarkar   B. Ellis   P. Haaland

March 28, 2008

1 Introduction

Traditionally, flow cytometry (FCM) has been a tube-based technique limited to small-scale laboratory studies. High throughput methods have recently been developed and are now used in both basic and clinical research. The large amount of information generated by such high throughput technologies must be stored, managed, and needs to be summarized in order to make it accessible to the researcher. The open source statistical software R in conjunction with the Bioconductor Project can play an important role in this new paradigm by providing a unified research platform which bioinformaticians, computer scientists, and statisticians can use to develop standard or novel methods.

In this lab we will learn how to use the **flowCore** package. Is has been developed to handle the main steps of importing, storing, assessing and preprocessing data from FCM experiments. During this practical session, we will learn how to access and manipulate flow cytometry data, focusing on the data structures to store, transform and filter (gate) FCM data. We will also use some quality assessment and visualization tools that are built upon the **flowCore** package and its object model, specifically those in the **flowViz** package.

2 Reading and exploring the data

2.1 Reading single files: the *flowFrame* object

Flow cytometry data files are read into the R environment *via* the `read.FCS` function. FCS files (version 2.0 and 3.0) and LMD (List Mode Data) extensions are currently supported. The result of this operation is an object of class *flowFrame*, the basic unit of storage for flow data in **flowCore**.

Exercise 1
Read in a file using the `read.FCS` *function as object* `fcs1`*. You should have sample FCS files available in the* `data/` *folder in your wokring directory. You could also import your own files if you want to, but we will be using the files in* `data/` *for later examples.*

Solution:

```
> library(flowCore)
> myfile <- "data/a01"
> fcs1 <- read.FCS(myfile)
> fcs1

flowFrame object 'A01' with 2205 cells and 8 observables:
     name               desc range minRange maxRange
$P1 FSC-H         FSC-Height  1024        0     1023
$P2 SSC-H         SSC-Height  1024        0     1023
$P3 FL1-H          CD15 FITC  1024        1    10000
$P4 FL2-H            CD45 PE  1024        1    10000
$P5 FL3-H         CD14 PerCP  1024        1    10000
$P6 FL2-A               <NA>  1024        0     1023
$P7 FL4-H           CD33 APC  1024        1    10000
$P8  Time Time (102.40 sec.)  1024        0     1023

slot 'description' has 150 elements
```

The primary elements of a *flowFrame* object are the `exprs` and `parameters` slots, which contain the event-level information (as a numeric matrix), and metadata describing the measurement channels, respectively. The `parameters` slot holds an *AnnotatedDataFrame* object that contains annotated information derived from an FCS file's `$P<n>` keywords about the detectors and stains used. Also, you can find out about the dynamic range of the measurement instrument for each channel. The `range` method of a *flowFame* will access the `parameters` slot, rather then returning the actual range of the raw data. Additional information regarding the parameters can be provided by the user and you should consult the documentation for *AnnotatedDataFrame* for details. Remember that you can get a table of values for an *AnnotatedDataFrame* using the `pData` function, and descriptions of each variable via the `varMetadata` function.

Exercise 2
Use the `exprs` *and* `parameters` *methods to explore your data. These are accessor methods, and you will soon find out that such mehtods exist for most of the features in* ***flowCore****'s classes. (Hint: you might want to subset the raw data matrix or use the function* `head` *to avoid all events being displayed, which might take quite a while). Finally apply the* `summary` *method on the flowFrame to get the summary statistics of the measurements in each channel.*

Solution:

```
> exprs(fcs1)[1:3,]
> head(fcs1)
> parameters(fcs1)
> pData(parameters(fcs1))
> summary(fcs1)
```

The `keyword` method allows access to the raw FCS keywords, which are a mix of standard entries such as "SAMPLE ID", vendor specific keywords, and user-defined keywords, that add more information about an experiment. For example, when acquiring flow data, the amplification information for parameters (channels) is stored in the amplification keyword mentioned before, which is part of the standard FCS definition, in the form $a \times 10^{x/R}$. Use either `keyword` without any additional arguments or, alternatively, the `description` function to get a list of all available keywords. The latter also lets you hide the FCS-internal keywords that start with a $ sign via the `hideInternal` argument. You can add you own keywords as a named list using one of the respective replacement functions.

Exercise 3
Use the `keyword` function to explore the amplification parameter (these have names of the form `$P<n>E`). Add your own keyword "foo" with the value "bar" using `keyword`'s replacement method. There is one keyword that encodes the patient ID. Can you find it?

Solution:

```
> names(description(fcs1, hideInternal=TRUE))
> keyword(fcs1, c("$P1E", "$P2E", "$P3E", "$P4E"))
> keyword(fcs1) <- list(foo="bar")
> keyword(fcs1, "foo")
> keyword(fcs1, "&10Patient ID")
```

The `read.FCS` function has a `transformation` argument. The default `linearize` transformation option will convert the value of these channels to have a `"$P<n>E"` of `"0,0"`. The `scale` option will both linearize values as well as ensure that they are in the range $[0, 1]$, which is the proposed method of data storage for the ACS1.0/FCS4.0 specification (Spidlen et al., 2006).

Exercise 4
Read in a file using the `read.FCS` function with different options for the `transformation` argument. Compare the results.

Solution:

```
> fcs1 <- read.FCS(myfile, transformation = "linearize")
> summary(fcs1)
> fcs2 <- read.FCS(myfile, transformation = "scale")
> summary(fcs2)
```

Another argument of interest in `read.FCS` is `alter.names`, which will convert the parameter names into more "R friendly" equivalents, e.g., by replacing "FSC-H" with "FCS.H":

```
> read.FCS(file.name, alter.names = TRUE)
```

For many of the downstream applications this is very useful, since you don't need to quote variable names.

2.2 Visualizing a *flowFrame*

flowCore implements some basic plotting facilities using the standard `plot` function. The default plot for a *flowFrame* without further arguments is a simple pairs density plot containing all available channels.

```
> plot(fcs1)
```

While such a plot can give a quick overview, it is not really helpful to inspect the details of the data. You can create individual bivariate density plots (or scatter plots if you prefer to look at those) by supplying the names of the parameters to plot as an additional argument.

```
> plot(fcs1, c("FSC-H", "SSC-H"))
```

Exercise 5
Create a similar plot of two other parameters. Take a look at the optional `smooth` argument. What does it do?

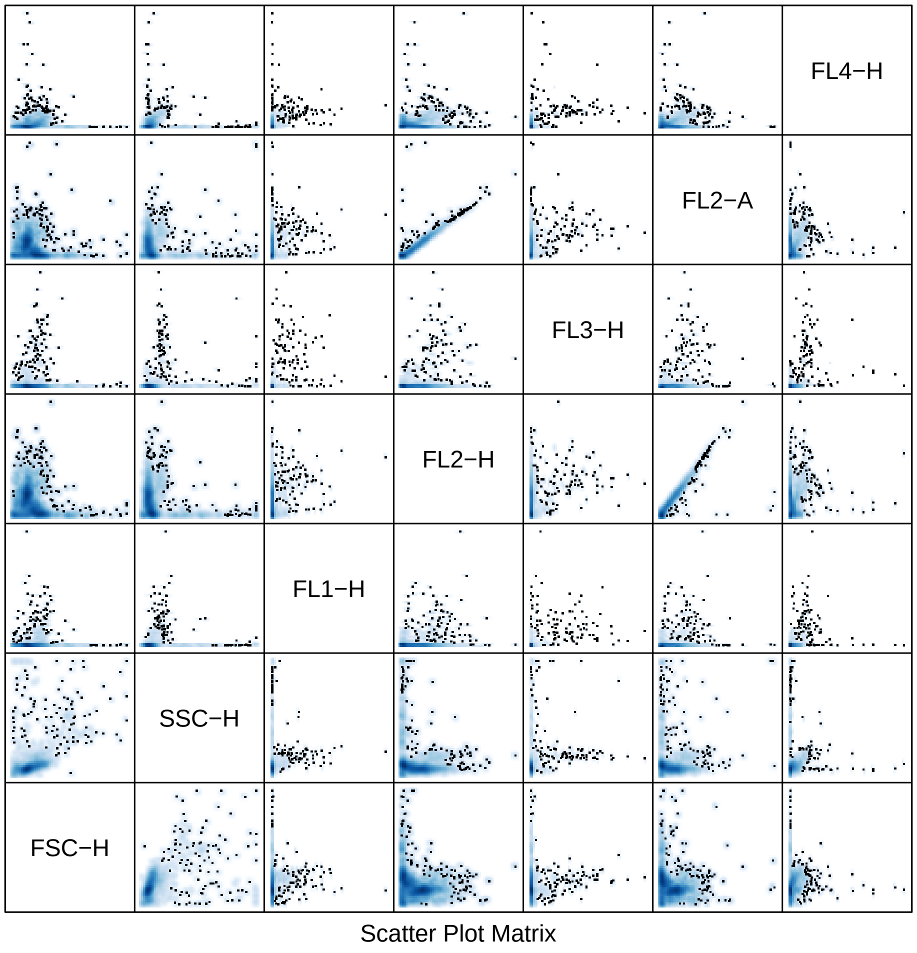

Figure 1: A pairs plot of all parameters in a *flowFrame*

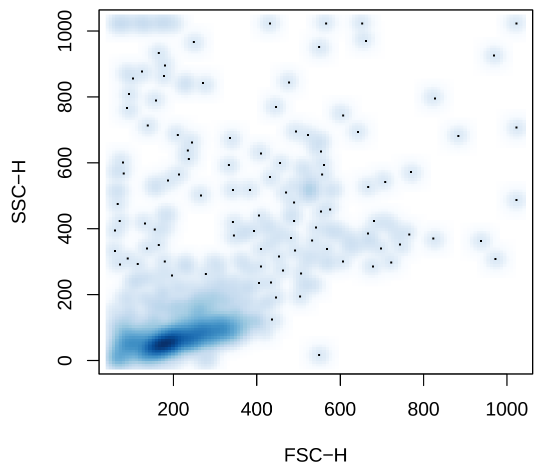

Figure 2: A bivariate density plot of the FSC and SSC parameters in a *flowFrame*

Solution:

```
> plot(fcs1, c("FL3-H", "FL4-H"))
> plot(fcs1, c("FL3-H", "FL4-H"), smooth=FALSE)
```

Univariate histograms can be produced by supplying one parameter name.

```
> plot(fcs1, c("SSC-H"), breaks=256)
```

More sophisticated visualization of FCM data is implemented in the **flowViz** package, which we will talk about after introducing the *flowSet* class.

2.3 The *flowSet* Class

Most experiments generate multiple FCS files, which can be organized using the *flowSet* class. This class provides a mechanism for efficiently hosting multiple *flowFrame* objects with minimal copying, reducing memory requirements, as well as ensuring that experimental metadata stays properly associated to the appropriate *flowFrame* objects. A *flowSet* object can be created by reading in a set of FCS files using the `read.flowSet` function. You should take a look at its documentation to find out about the multiple possibilities of data import. As an example, we will read a subset of data from Rizzieri et al. (2007) (available to you in the `data/` directory), and we identify FCS files by the pattern of their file names:

```
> ffiles <- list.files("data", pattern = "^[a-h]")
> fset <- read.flowSet(ffiles, path = "data")
> fset

A flowSet with 96 experiments.

  column names:
  FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time
```

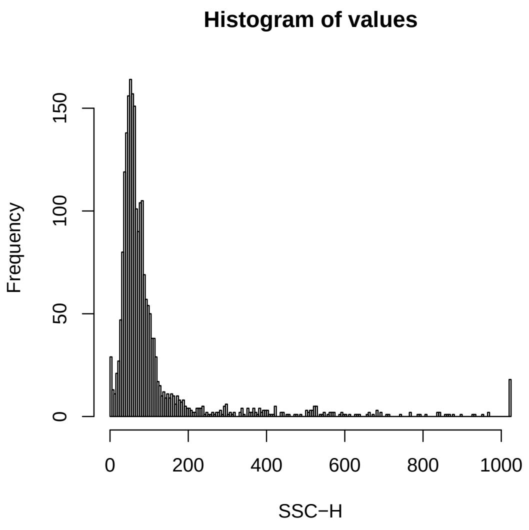

Figure 3: A histogram of SSC parameter in a *flowFrame*

The whole dataset originated from a collection of weekly peripheral blood samples obtained from several patients following allogenic blood and marrow transplant. The goal of this study was to identify cellular markers that would predict the development of Graft *vs* Host Disease (GvHD). Samples were taken at various time points before and after transplantation. At each time point, every patient blood sample was divided into eight aliquots. Each aliquot was labeled with four different fluorescent markers and their fluorescent intensity determined, in addition to the usual Forward and Side scatter measurements. In this lab we will focus on the measurements from 1 patient, which corresponds to a single 96-well microtitre plate.

2.4 Working with experimental meta-data

Like most Bioconductor organizational classes, the *flowSet* has an associated *AnnotatedDataFrame* that provides metadata not contained within the *flowFrame* objects themselves. This data frame is accessed and modified via the usual `phenoData` and `phenoData<-` methods. Our data corresponds to one particular transplant patient. The associated phenotypic data (aliquots and time in days past transplant) are available in the file `data/phenodata.txt`. You might want to inspect this file using a spreadsheet software like Excel to find out about its structure. We can read it in and create an *AnnotatedDataFrame* object using

```
> pdata <- read.AnnotatedDataFrame("data/phenodata.txt")
> phenoData(fset) <- pdata
> fset
```

```
A flowSet with 96 experiments.

An object of class "AnnotatedDataFrame"
  rowNames: a01, a02, ..., h12  (96 total)
  varLabels and varMetadata description:
    aliquot:  Sample aliquot
    time:  Days Past Transplant
  column names:
  FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time
```

Each frame has a unique identifier which can be accessed using the function `sampleNames`.

```
> sampleNames(fset)
```

```
 [1] "a01" "a02" "a03" "a04" "a05" "a06" "a07" "a08" "a09" "a10" "a11" "a12"
[13] "b01" "b02" "b03" "b04" "b05" "b06" "b07" "b08" "b09" "b10" "b11" "b12"
[25] "c01" "c02" "c03" "c04" "c05" "c06" "c07" "c08" "c09" "c10" "c11" "c12"
[37] "d01" "d02" "d03" "d04" "d05" "d06" "d07" "d08" "d09" "d10" "d11" "d12"
[49] "e01" "e02" "e03" "e04" "e05" "e06" "e07" "e08" "e09" "e10" "e11" "e12"
[61] "f01" "f02" "f03" "f04" "f05" "f06" "f07" "f08" "f09" "f10" "f11" "f12"
[73] "g01" "g02" "g03" "g04" "g05" "g06" "g07" "g08" "g09" "g10" "g11" "g12"
[85] "h01" "h02" "h03" "h04" "h05" "h06" "h07" "h08" "h09" "h10" "h11" "h12"
```

Exercise 6
It would be useful to add the patient ID you found before in the FCS keywords and which is the same for all samples in the flowSet along with the well ID (which is encoded in the sample names) to the metadata. (Hint: use the replacement functions for `pData` *and* `varMetadata`*). Note that you first have to extract the AnnotatedDataFrame from the flowSet to be able to edit it. After you are done, you should get something like*

```
> fset

A flowSet with 96 experiments.

An object of class "AnnotatedDataFrame"
  rowNames: a01, a02, ..., h12  (96 total)
  varLabels and varMetadata description:
    aliquot:  Sample aliquot
    time:  Days Past Transplant
    patient: Patient ID
    well: Well ID

  column names:
  FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time
```

Solution:

```
> pd <- phenoData(fset)
> pData(pd)$patient <- keyword(fcs1, "&10Patient ID")
> pData(pd)$well <- sampleNames(fset)
> varMetadata(pd)[c("patient", "well"), "labelDescription"] <- c("Patient ID",
                                                                "Well ID")
> phenoData(fset) <- pd
```

2.5 Manipulating a *flowSet*

It is possible to extract a *flowFrame* from a *flowSet* object using a `list`-like syntax:

```
> fset$"a01"

flowFrame object 'A01' with 2205 cells and 8 observables:
     name               desc range minRange maxRange
$P1 FSC-H         FSC-Height  1024        0     1023
$P2 SSC-H         SSC-Height  1024        0     1023
$P3 FL1-H          CD15 FITC  1024        1    10000
$P4 FL2-H            CD45 PE  1024        1    10000
$P5 FL3-H         CD14 PerCP  1024        1    10000
$P6 FL2-A               <NA>  1024        0     1023
$P7 FL4-H           CD33 APC  1024        1    10000
$P8  Time Time (102.40 sec.)  1024        0     1023

slot 'description' has 150 elements

> fset[[1]]

flowFrame object 'A01' with 2205 cells and 8 observables:
     name               desc range minRange maxRange
$P1 FSC-H         FSC-Height  1024        0     1023
$P2 SSC-H         SSC-Height  1024        0     1023
$P3 FL1-H          CD15 FITC  1024        1    10000
$P4 FL2-H            CD45 PE  1024        1    10000
$P5 FL3-H         CD14 PerCP  1024        1    10000
$P6 FL2-A               <NA>  1024        0     1023
$P7 FL4-H           CD33 APC  1024        1    10000
$P8  Time Time (102.40 sec.)  1024        0     1023

slot 'description' has 150 elements
```

Again analogous to lists, we can subset a *flowSet* via the single colon operator:

```
> fset[1:3]

A flowSet with 3 experiments.

An object of class "AnnotatedDataFrame"
  rowNames: a01, a02, a03
  varLabels and varMetadata description:
    aliquot:  Sample aliquot
    time:  Days Past Transplant
    patient: Patient ID
    well: Well ID

  column names:
  FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time

> fset[c("a01", "b03")]
```

```
A flowSet with 2 experiments.

An object of class "AnnotatedDataFrame"
  rowNames: a01, b03
  varLabels and varMetadata description:
    aliquot:  Sample aliquot
    time:  Days Past Transplant
    patient: Patient ID
    well: Well ID

  column names:
  FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time
```

There is also an iterator method `fsApply` that can be used to apply an arbitrary function on all components of a *flowSet*. It behaves much like `lapply`, except that by default, if all of the return values are *flowFrame* objects, `fsApply` will create a new *flowSet* object to hold them. Moreover, the user can control whether the function should operate on the individual *flowFrames* or on the raw data matrix of each frame.

Exercise 7
Compute the interquartile range (using `IQR`) for each parameter of each flowFrame in `fset`. Hint: look at `help(fsApply)` and `help(each_col)`.

Solution:

```
> fsApply(fset, each_col, IQR)
```

which is equivalent to the less readable

```
> fsApply(fset, function(x) apply(x, 2, IQR), use.exprs = TRUE)
```

3 Transformations

flowCore features two methods of transforming parameters within a *flowFrame*. One option is to use it in a fashion similar to R's `transform` function:

```
> summary(transform(fset[[1]], log.FL1.H = log(`FL1-H`),
                    "FL2-H"=log(`FL2-H`)))

         FSC-H  SSC-H    FL1-H FL2-H   FL3-H FL2-A   FL4-H Time log.FL1.H
Min.      60.0    0.0    1.000 0.000   1.000   0.0   1.000 11.0   0.00000
1st Qu.  159.0   48.0    1.046 3.565   1.000   6.0   1.000 40.0   0.04502
Median   196.0   65.0    2.644 5.078   1.383  36.0   5.289 57.0   0.97240
Mean     220.8  108.9   57.540 4.591   7.367  48.7  16.240 51.9   1.52100
3rd Qu.  264.0   97.0    7.055 5.771   2.460  75.0  20.780 66.0   1.95400
Max.    1023.0 1023.0 3782.000 7.401 326.700 516.0 503.300 80.0   8.23800
```

This returns a new *flowFrame* (or *flowSet*) with additional (or modified) parameters. The other method is to create a *transformList* object that represents an abstract transformation that can be subsequently applied to a *flowFrame* or *flowSet*. This allows for a more convenient inline notation, but for the sake of code readability we encourage using the first method.

```
> summary(transform("FL1-H" = log) %on% fset[[1]])

         FSC-H  SSC-H   FL1-H   FL2-H   FL3-H FL2-A   FL4-H Time
Min.      60.0    0.0 0.00000    1.00   1.000   0.0   1.000 11.0
1st Qu.  159.0   48.0 0.04502   35.35   1.000   6.0   1.000 40.0
Median   196.0   65.0 0.97240  160.40   1.383  36.0   5.289 57.0
Mean     220.8  108.9 1.52100  210.10   7.367  48.7  16.240 51.9
3rd Qu.  264.0   97.0 1.95400  320.90   2.460  75.0  20.780 66.0
Max.    1023.0 1023.0 8.23800 1637.00 326.700 516.0 503.300 80.0
```

Though any function can be used as a transform in both methods, **flowCore** provides a number of parameterized transform generators that correspond to the transforms commonly found in flow cytometry and defined in the Transformation Markup Language (Transformation-ML). A list can be seen by typing `help("transform-class")`. Transformations can be very useful for plotting (and for fitting the data driven filters we will see below). For example, see Figure 4, which compares

```
> library(flowViz)
> splom(fset$h08[, 1:5])
```

and

```
> splom(transform("FSC-H" = asinh, "SSC-H" = asinh,
                  "FL1-H" = asinh, "FL2-H" = asinh,
                  "FL3-H" = asinh) %on% fset$h08[, 1:5])
```

To produce these plots we used the function `splom` from the **flowViz** package, and later in this tutorial we will see more examples of that.

A tranformation can be applied to a complete *flowSet* just as it was applied to a *flowFrame* above.

Exercise 8
Create a new flowSet object called `fset.trans` *that has your favorite transformation applied to all the channels. We will use this in subsequent examples.*

Solution:

```
> fset.trans <- transform(fset, "FL1-H"=asinh(`FL1-H`),
                          "FL2-H"=asinh(`FL2-H`),
                          "FL3-H"=asinh(`FL3-H`),
                          "FL2-A"=asinh(`FL2-A`),
                          "FL4-H"=asinh(`FL4-H`))
```

4 Quality assessment

flowViz provides several graphical methods that, among basic visual inspection of FCM data, can be used for quality assessment. Much more sophisticated QA methods can be found in the **flowQ** package, but this is beyond the scope of this introduction. One simple plot, the default `xy.plot` for a *flowFrame*, displays the fluorescence intensities over time (Figure 5):

```
> xyplot(fset.trans[[1]])
```

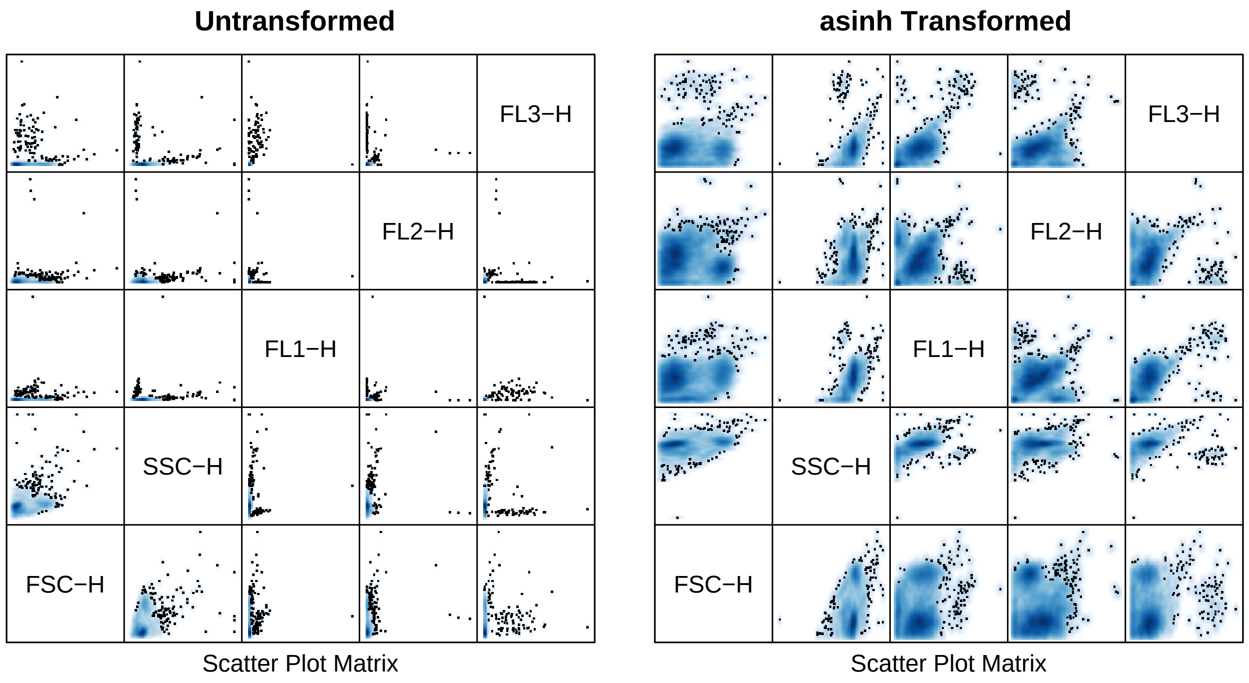

Figure 4: Pairwise scatter plots of untransformed and transformed parameters

We can see that for this particular sample, the measurements are a bit odd at the beginning of the run but seem to level off after some time. This is not uncommon for flow cytometry data as the fluidics often need some time to stabilize.

Exercise 9
Try this for a few other samples. Can you find other samples that also look odd? How could this be explained?

In our example, the distribution of Forward Scatter and Side Scatter can also be used to detect unusual samples (Figure 6) [1]. **flowViz** is based on the **lattice** package, and the notation in the following code chunks makes use of its somewhat unfamiliar formula interface. The interested user is refered to the documentation of the **lattice** package to learn about the details. For now, it is sufficient to understand that the interface allows to arange multiple plots according to one or several factors, and that is exactly what we need in order to visualize a whole *flowSet* or a subset of several *flowFrames*.

```
> qqmath(~`FSC-H` | factor(time), fset.trans, groups = aliquot,
         f.value = ppoints(500), type = "l")
```

Exercise 10
Which aliquot is the odd one out for time 46?

Solution:

[1]The other channels can not, since they have different dyes associated with them for different aliquots

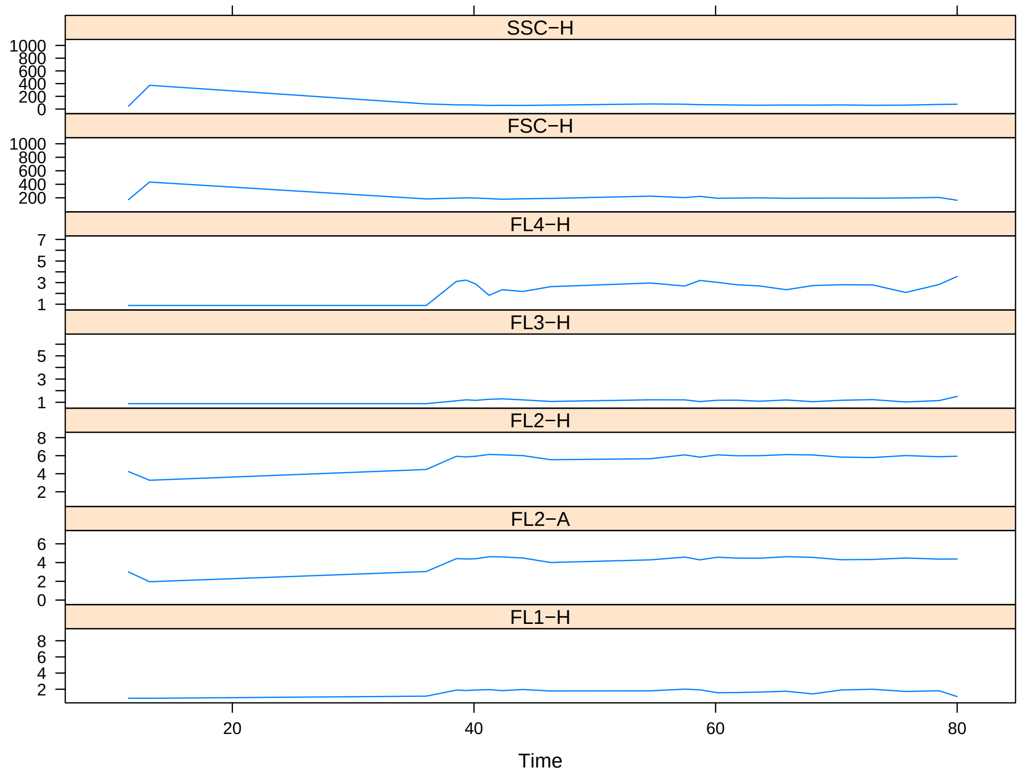

Figure 5: Parameter values over time in a *flowFrame*

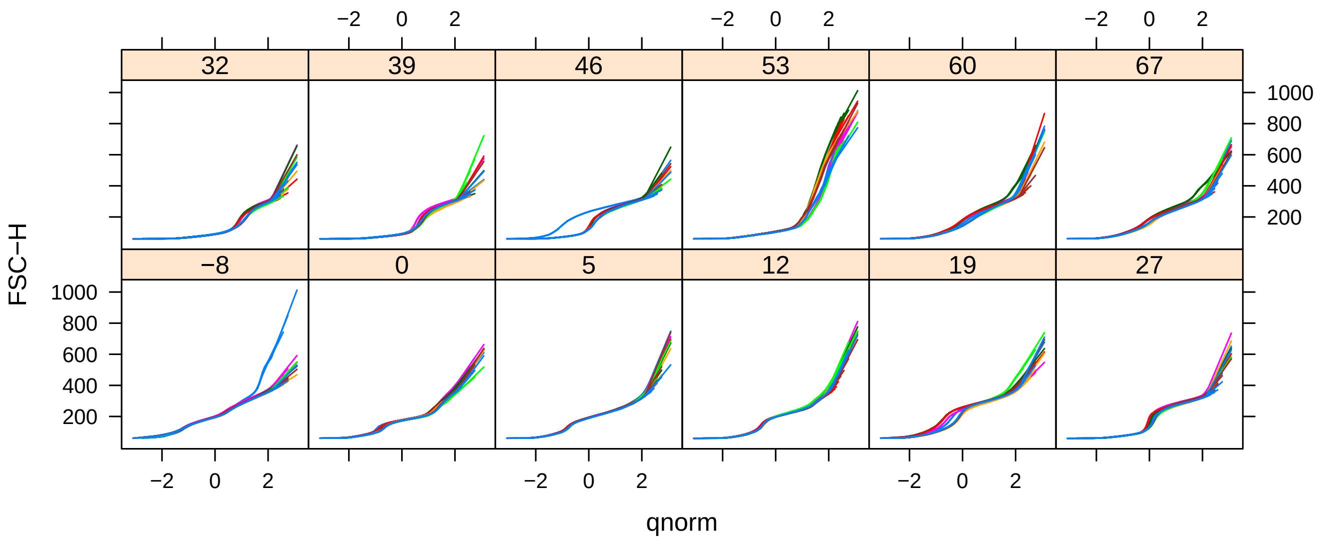

Figure 6: Normal Q-Q plot of Forward Scatter in all frames in a *flowSet*

```
> qqmath(~`FSC-H` | aliquot, fset.trans, f.value = ppoints(500),
         type = c("l", "g"), subset = (time == 46))
```

To get a quick comparison of the data in a *flowSet* for a single parameter, stacked density plots have shown to be very useful. The following code produces the plot in 7.

```
> densityplot(factor(sampleNames(fset.trans))~ `FSC-H`, fset.trans,
              subset=(aliquot == "A"),
              scales = list(y = list(draw = F)))
```

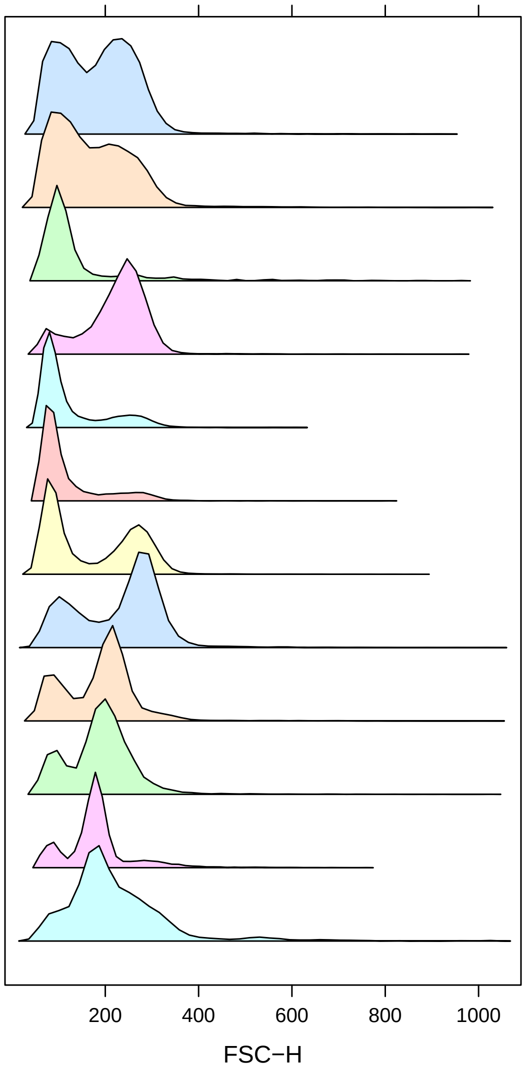

Figure 7: Stacked density plots for all frames in a *flowSet*

5 Filtering

The most common task in the analysis of flow cytometry data is usually some form of filtering operation, either to obtain summary statistics about the number of events that meet a certain criteria or to perform further analysis on a subset of the data.

5.1 Standard Filters

Most filtering operations are a composition of one or more common filtering operations. Like transformations, **flowCore** includes a number of built-in common flow cytometry filters. The simplest of these filters are the geometric filters, which correspond to those typically found in interactive flow cytometry software:

rectangleGate Describes a cubic shape in one or more dimensions–a rectangle in one dimension is simply an interval gate.

polygonGate Describes an arbitrary two dimensional polygonal gate.

polytopeGate Describes a region that is the convex hull of the given points. This gate can exist in dimensions higher than 2, unlike the `polygonGate`.

ellipsoidGate Describes an ellipsoidal region in two or more dimensions

These gates are all specified in more or less the same manner. For example,

```
> rectgate <- rectangleGate(filterId="Rectangle",
                             "FSC-H" = c(150, 450), "SSC-H" = c(25, 350))
> rectgate
```

```
Rectangular gate with dimensions:
        FSC-H: (150,450)
        SSC-H: (25,350)
```

creates a rectangular gate object, which can be applied to a *flowFrame* or *flowSet*.

Much more intersting are the additional data-driven filters not usually found in flow cytometry software:

norm2Filter A robust method for finding a region that most resembles a bivariate Normal distribution.

kmeansFilter Identifies populations based on a one dimensional k-means clustering operation. Allows the specification of multiple populations.

curv1Filter Identifies populations based on density estimation in one dimension. Allows the specification of multiple populations.

curv2Filter Identifies populations based on density estimation in two dimensions. Allows the specification of multiple populations.

For example,

```
> bvnormfilt <- norm2Filter(filterId = "BVNorm", "FSC-H", "SSC-H", scale=2)
> bvnormfilt
```

```
A filter named 'BVNorm'
```

can be used to gate for lymphocytes based on their FSC and SSC properties.

```
> curvfilt <- curv2Filter(filterId="curvature", "FL1-H", "FL3-H",
                          bwFac=2)
> curvfilt

A curv2 filter named 'curvature' with settings:
  bwFac=2
  gridsize=151,151
```

will identify high density region of significant curvature in the FL1-H and FL3-H channels and

```
> kmfilt <- kmeansFilter("kmfilt", "FSC-H" = c("Low", "High"))
> kmfilt

A k-means filter named ' kmfilt ':  FSC-H  ( Low,High )
```

will separate the data in the two groups "High" and "Low" based on the FSC signal. In all these cases we have constructed an abstract filter object which we can now apply to a particular data set to collect simple summary statistics on the number and proportion of events considered to be contained within the gate or filter. This is done using the `filter` method, and the output of this operation is an object of class *filterResult*. A filter can be applied to an individual *flowFrame* as well as an entire *flowSet*, in which case it returns a list of *filterResult* objects:

```
> rectgate.results <- filter(fset.trans, rectgate)
> bvnormfilt.results <- filter(fset.trans, bvnormfilt)
> kmfilt.results <- filter(fset.trans, kmfilt)
> summary(rectgate.results[[1]])
```

Exercise 11
Produce a summary of `kmfilt.results[[1]]`*. How can you obtain a summary for more than one frame at a time?*

Solution:

```
> summary(kmfilt.results[[1]])

Low: 1765 of 2205 (80.05%)
High: 440 of 2205 (19.95%)

> lapply(head(kmfilt.results, 4), summary)

$a01
Low: 1765 of 2205 (80.05%)
High: 440 of 2205 (19.95%)

$a02
Low: 11743 of 13350 (87.96%)
High: 1607 of 13350 (12.04%)

$a03
Low: 4600 of 11610 (39.62%)
High: 7010 of 11610 (60.38%)

$a04
Low: 4677 of 13830 (33.82%)
High: 9153 of 13830 (66.18%)
```

5.2 Subsetting and splitting

To subset or split a *flowFrame* or *flowSet*, we use the `Subset` and `split` methods, respectively. `Subset` can be used for logical (`TRUE`/`FALSE`) filters, while `split` has to used for filters that detect multiple populations. Note that you can also use `split` in combination with logical filter in case you want to keep both the populations within and out of a gate for further analysis. For example, the `bvnormfilt` filter tries to detect a subregion that looks like a bivariate normal distribution. If we wished to deal only with that subpopulation, we might use `Subset` as follows:

```
> smaller <- Subset(fset.trans, bvnormfilt)
> nrow(fset.trans[[1]])

[1] 2205

> nrow(smaller[[1]])

[1] 1660
```

Note how the `smaller` *flowFrame* objects contain fewer events.

Exercise 12
Use `split` to obtain subpopulations of `smaller[[1]]` defined by the `kmfilt` gate. In what form does `split` return the results?

Solution:

```
> split(smaller[[1]], kmfilt)

$Low
flowFrame object 'A01' with 1074 cells and 8 observables:
     name              desc range  minRange    maxRange
$P1 FSC-H         FSC-Height  1024 0.0000000 1023.000000
$P2 SSC-H         SSC-Height  1024 0.0000000 1023.000000
$P3 FL1-H          CD15 FITC  1024 0.8813736    9.903488
$P4 FL2-H            CD45 PE  1024 0.8813736    9.903488
$P5 FL3-H         CD14 PerCP  1024 0.8813736    9.903488
$P6 FL2-A               <NA>  1024 0.0000000    7.623642
$P7 FL4-H           CD33 APC  1024 0.8813736    9.903488
$P8  Time Time (102.40 sec.)  1024 0.0000000 1023.000000

slot 'description' has 150 elements

$High
flowFrame object 'A01' with 586 cells and 8 observables:
     name              desc range  minRange    maxRange
$P1 FSC-H         FSC-Height  1024 0.0000000 1023.000000
$P2 SSC-H         SSC-Height  1024 0.0000000 1023.000000
$P3 FL1-H          CD15 FITC  1024 0.8813736    9.903488
$P4 FL2-H            CD45 PE  1024 0.8813736    9.903488
$P5 FL3-H         CD14 PerCP  1024 0.8813736    9.903488
$P6 FL2-A               <NA>  1024 0.0000000    7.623642
$P7 FL4-H           CD33 APC  1024 0.8813736    9.903488
$P8  Time Time (102.40 sec.)  1024 0.0000000 1023.000000

slot 'description' has 150 elements
```

5.3 Combinations of filters

Of course, most filtering operations consist of more than one gate. To combine gates and filters we use the standard R Boolean operators: `&`, `|` and `!` to construct intersections, unions and complements respectively:

```
> rectgate & bvnormfilt

A filter named 'Rectangle and BVNorm'

> rectgate | bvnormfilt

A filter named 'Rectangle or BVNorm'

> !bvnormfilt

A filter named 'not BVNorm'
```

Data driven filters such as `norm2Filter` do not always work out of the box; for instance if there is more than one population, fitting a single distribution doesn't make much sense (Figure 8 a). Note that in the next code chunk we again use the plotting functionality provided by the **flowViz** package, and that we passed the *filter* object on to `xyplot` in order to draw its boundaries on the plot.

```
> xyplot(`SSC-H` ~ `FSC-H` | factor(aliquot), fset.trans, subset = (time == 19),
         filter = bvnormfilt)
```

The mass of cells close to zero in some of the samples seems to attract the `norm2Filter`. We might use a filter that is better suited for mixtures of multiple populations or, alternatively, do some rough preselection like in the next code chunk, using the `rectgate` filter defined before (Figure 8 b). This time, we pass the *filterResult* to `xyplot`. Note how plotting is considerably faster, because we don't need to re-evaluate the filter.

```
> sub <- Subset(fset.trans, rectgate)
> bvnormfilt.results.new <- filter(sub, bvnormfilt)
> xyplot(`SSC-H` ~ `FSC-H` | factor(aliquot), fset.trans, subset = (time == 19),
          filter = bvnormfilt.results.new)
```

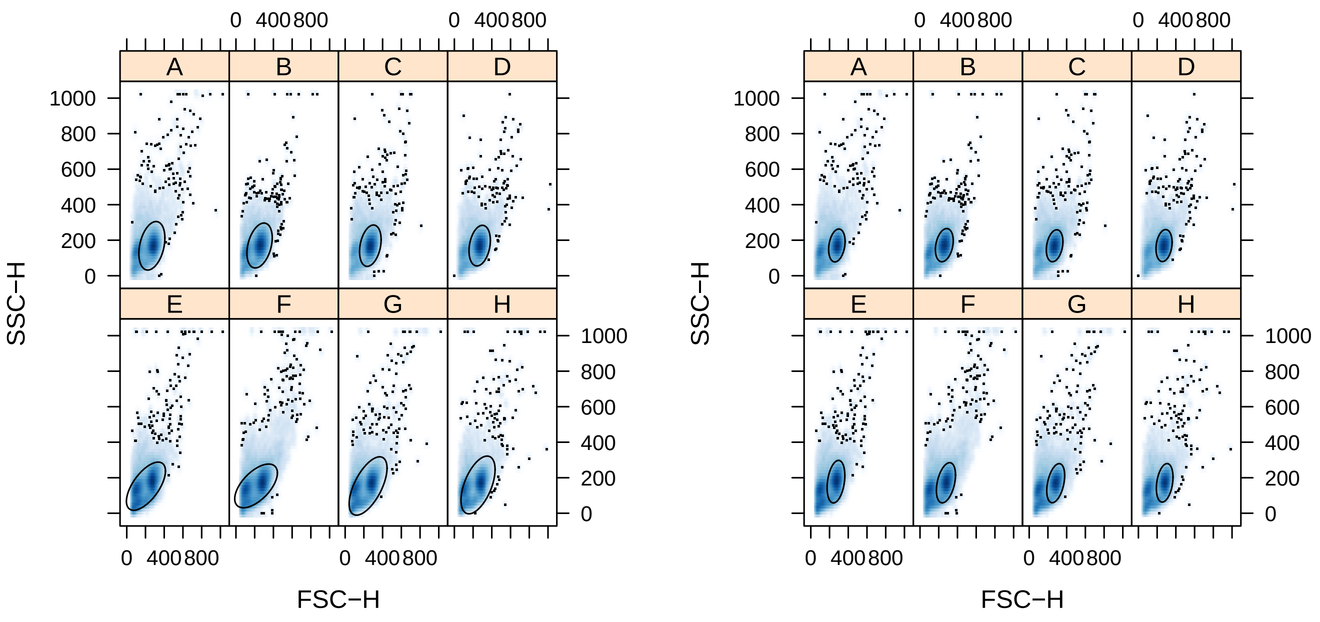

Figure 8: Scatter plot of `FSC-H` and `SSC-H` using a data-driven *norm2Filter* naively (left) and after some rough preselection(right)

Exercise 13
Apply the improved `norm2Filter` *to each frame and extract the lymphocyte population into a new flowSet* `lymphs`*.*

Solution:

```
> lymphs <- Subset(fset.trans, bvnormfilt.results.new)
```

Assuming that we have done a decent job in selecting lymphocytes, we could go on an try to find interesting populations in some of the fluorescent channels. We could for example use our *curv2Filter* to look for high density regions:

```
> xyplot(`FL3-H` ~ `FL1-H` | factor(aliquot), lymphs,
         subset = time == 46, filter = curvfilt, col="transparent")
```

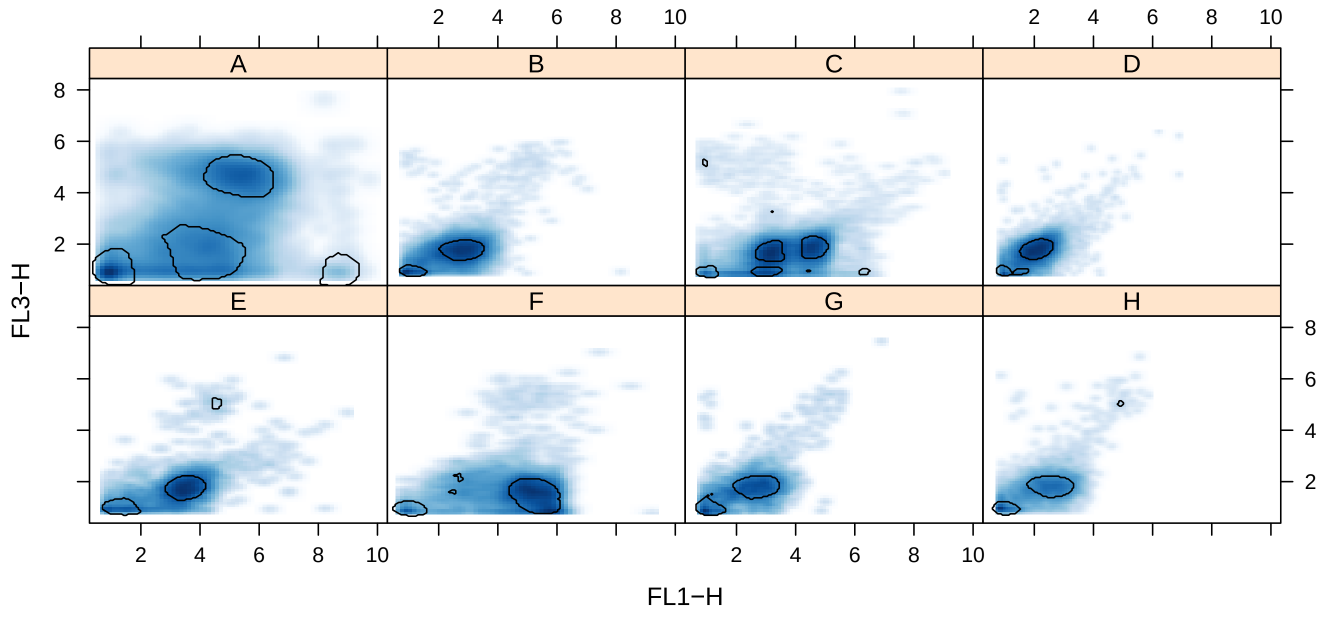

Figure 9: High density regions.

```
> sessionInfo()

R version 2.8.0 Under development (unstable) (2008-03-25 r44912)
x86_64-unknown-linux-gnu

locale:
LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=C;LC_MESSAGES=en_US;LC_PAPER=e

attached base packages:
[1] tools     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] flowViz_1.3.4        RColorBrewer_1.0-2   MASS_7.2-41
 [4] geneplotter_1.17.7   lattice_0.17-6       annotate_1.17.11
 [7] xtable_1.5-2         AnnotationDbi_1.1.21 RSQLite_0.6-7
[10] DBI_0.2-4            flowCore_1.5.13      feature_1.1-12
[13] rgl_0.76             misc3d_0.4-1         KernSmooth_2.22-22
[16] rrcov_0.4-03         robustbase_0.2-8     Biobase_1.17.13

loaded via a namespace (and not attached):
[1] grid_2.8.0         latticeExtra_0.3-1 stats4_2.8.0
```

References

D.A. Rizzieri, L.P. Koh, G.D. Long, C. Gasparetto, K.M. Sullivan, M. Horwitz, J. Chute, C. Smith, J.Z. Gong, A. Lagoo, et al. Partially Matched, Nonmyeloablative Allogeneic Transplantation: Clinical Outcomes and Immune Reconstitution. *Journal of Clinical Oncology*, 25(6):690, 2007.

J. Spidlen, R.C. Gentleman, P.D. Haaland, M. Langille, N. Le Meur N, M.F. Ochs, C. Schmitt, C.A. Smith, A.S. Treister, and R.R. Brinkman. Data standards for flow cytometry. *OMICS*, 10(2):209–214, 2006.