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

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

August 6, 2007

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. This package is being 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 using the flowCore package, 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

FCS 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 is a *flowFrame*, the basic unit of storage in flowCore.

Exercise 1
Read in a file using the read.FCS() function. You should have sample FCS files available in the `data/` *folder. You could also use your own files, but we will be using the files in* `data/` *for later examples.*

Solution:

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

flowFrame object with 2205 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A>
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 matrix) and column meta-data respectively. The `parameters` slot is an *AnnotatedDataFrame*

object that contains information derived from an FCS file's "$P<n>" keywords, which describe the detector and stain information.

Exercise 2
Use the exprs() and parameters() functions to explore your data (hint: use some subsetting with the exprs() or all single event will be displayed). Finally apply the summary() method on the flowFrame to get the range of the measurements in each channel.

Solution:

```
> exprs(fcs1)[1:10,]
> 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 saving the measurements, some of the parameters (channels) can be stored in the form $a \times 10^{x/R}$.

Exercise 3
Use the keyword() function to explore the amplification parameter (these have names of the form "$P<n>E").

Solution:

```
> keyword(fcs1, c("$P1E", "$P2E", "$P3E", "$P4E"))
```

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 output values are contained in $[0, 1]$, 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 using 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 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)
```

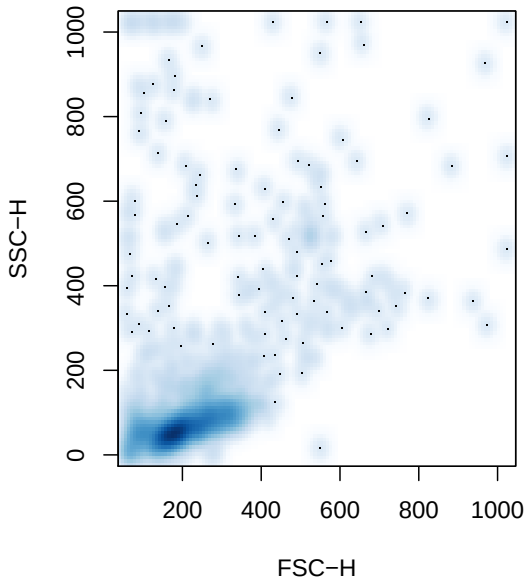

Figure 1: A plot of the first two parameters in a *flowFrame*

2.2 Visualizing a *flowFrame*

flowCore implements some basic plotting facilities using the standard plot() function. The basic plot provides a simple bivariate density plot of the first two parameters

```
> plot(fcs1)
```

Exercise 5
Create a similar plot of two other parameters.

Solution:

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

Univariate histograms can be produced by supplying one parameter name.

```
> plot(fcs1, "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 meta-data 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. As an example, we will read a subset of data from Rizzieri et al. (2007) (available to you in the `data/` directory):

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

A flowSet with 96 experiments.

  rowNames: a01, a02, ..., h12 (96 total)
  varLabels and varMetadata:
    name: Name

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

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.

2.4 Working with experimental meta-data

Like most Bioconductor organizational classes, the *flowSet* has an associated *AnnotatedDataFrame* that provides meta-data 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`. We can read it in and create an *AnnotatedDataFrame* object using

```
> pdata <- read.table("data/phenodata.txt", header = TRUE)
> pdata <- as(pdata, "AnnotatedDataFrame")
> phenoData(fset) <- pdata
> fset

A flowSet with 96 experiments.

  rowNames: a01, a02, ..., h12 (96 total)
  varLabels and varMetadata:
    aliquot: NA
    time: NA

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

Exercise 6
Add meaningful values for the `labelDescription` *column in the phenodata of* `fset` *(hint: try looking at* `varMetadata(phenoData(fset))`*). After you are done, you should get something like*

```
> fset
```

```
A flowSet with 96 experiments.

  rowNames: a01, a02, ..., h12 (96 total)
  varLabels and varMetadata:
    aliquot: Aliquot
    time: Days Past Transplant

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

Solution:

```
> varMetadata(phenoData(fset))[, "labelDescription"] <-
      c("Aliquot", "Days Past Transplant")
```

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 with 2205 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A> <F
slot 'description' has 150 elements

> fset[[1]]

flowFrame object with 2205 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A> <F
slot 'description' has 150 elements
```

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 of the are *flowFrame* objects, fsApply() will create a new *flowSet* object to hold them.

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

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`)))
```

```
         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    1.00   1.000   0.0   1.000 11.0   0.00000
1st Qu.  159.0   48.0    1.046   35.35   1.000   6.0   1.000 40.0   0.04502
Median   196.0   65.0    2.644  160.40   1.383  36.0   5.289 57.0   0.97240
Mean     220.8  108.9   57.540  210.10   7.367  48.7  16.240 51.9   1.52100
3rd Qu.  264.0   97.0    7.055  320.90   2.460  75.0  20.780 66.0   1.95400
Max.    1023.0 1023.0 3782.000 1637.00 326.700 516.0 503.300 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*.

```
> 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 3, which compares

```
> 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])
```

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:

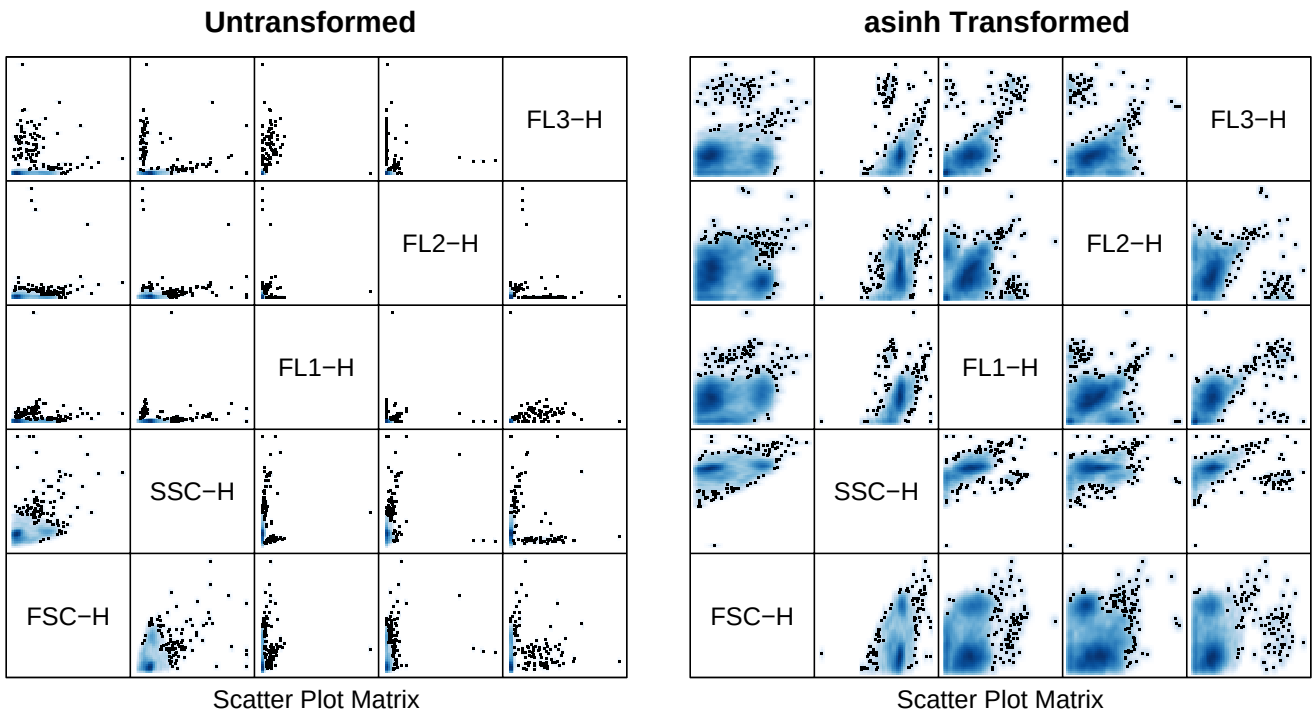

Figure 2: Pairwise scatter plots of untransformed and transformed parameters

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

4 Quality assessment

flowViz provides several graphical methods that can be used for quality assessment of FCM data. One simple plot looks at fluorescence values over time (Figure 4):

```
> xyplot(fset.trans[[3]])
```

Exercise 9
Try this for a few other samples.

In our example, the distribution of Forward Scatter and Side Scatter can also be used to detect unusual samples (Figure 4): [1]

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

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

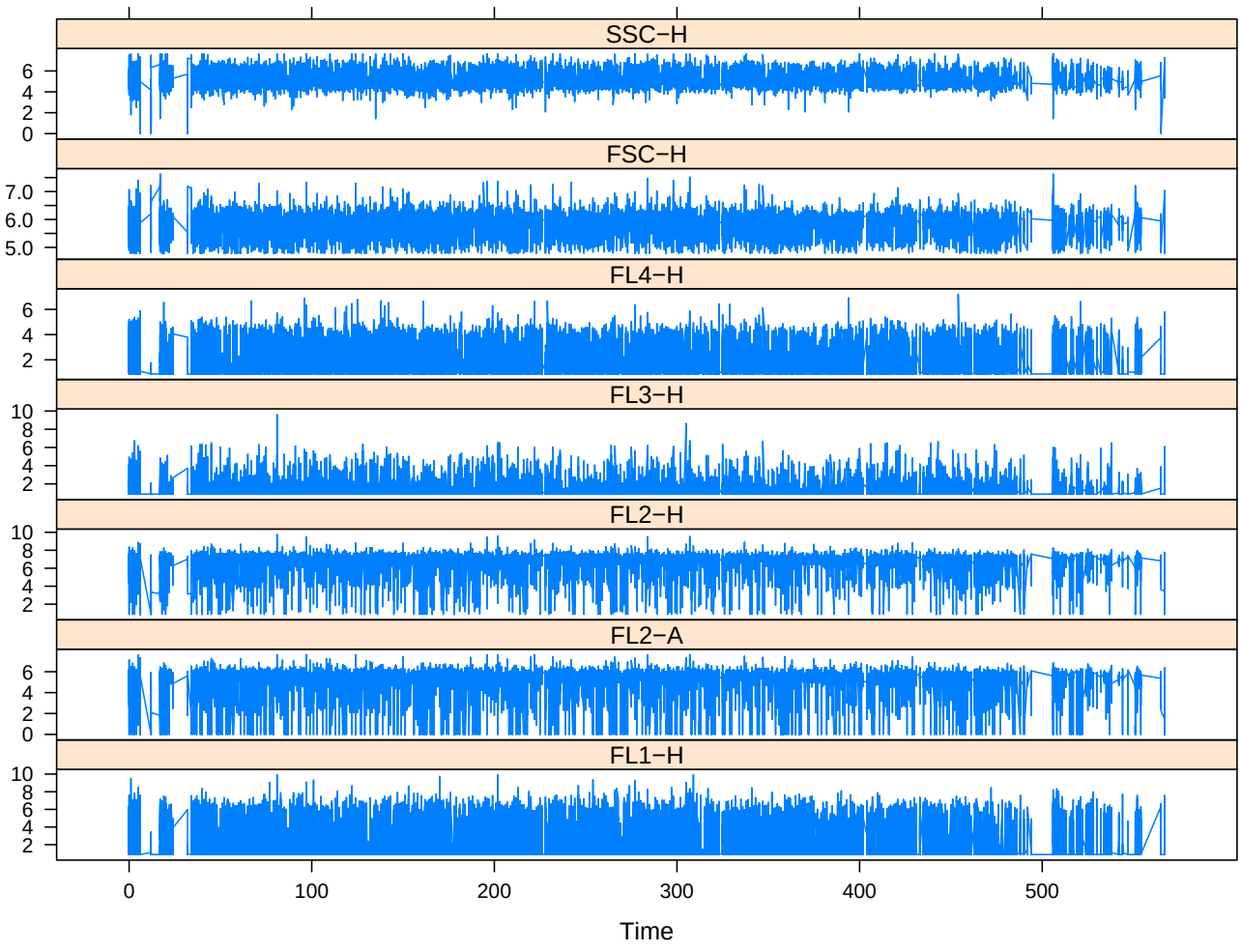

Figure 3: Parameter values over time in a *flowFrame*

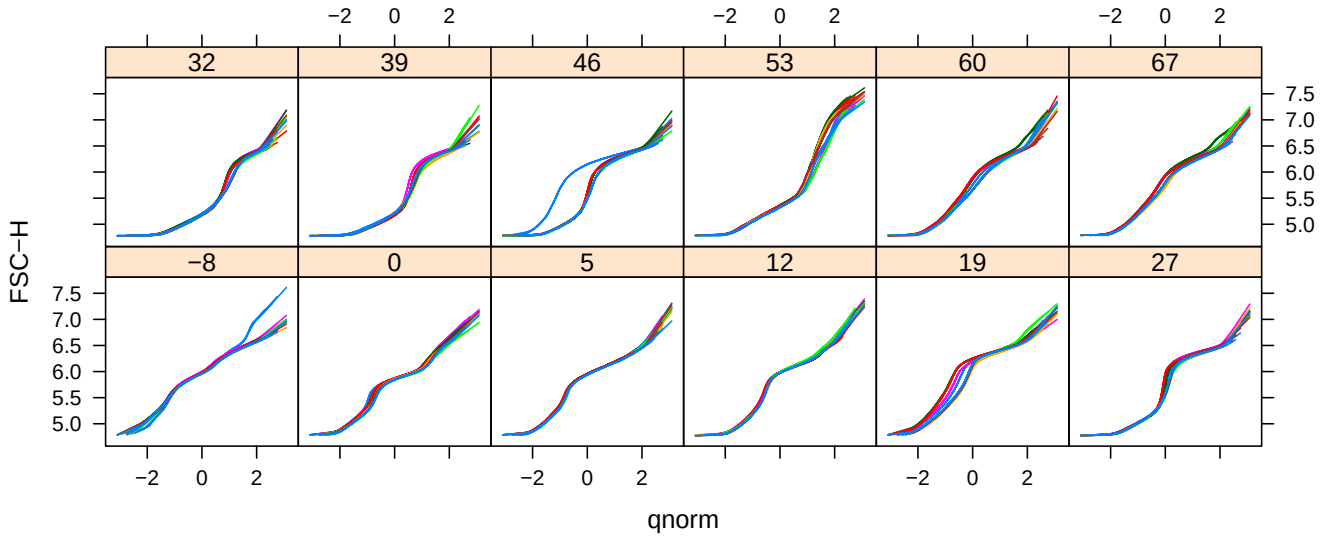

Figure 4: Normal Q-Q plot of Forward Scatter in all frames in a *flowSet*

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

Solution:

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

5 Filtering

The most common task in the analysis of flow cytometry data is usual 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 described in more or less the same manner. For example,

```
> rectgate <- rectangleGate(filterId="Rectangle",
                            "SSC-H" = c(2.5, 6.5), "FSC-H" = c(4, 5.75))
```

This creates an abstract filter object, which can be applied to a *flowFrame* or *flowSet*. There are also some 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.

For example,

```
> kmfilt <- kmeansFilter("kmfilt", "FSC-H" = c("Low", "High"))
> bvnormfilt <- norm2Filter(filterId = "BVNorm", "FSC-H", "SSC-H", scale=2)
> bvnormfilt

A filter named 'BVNorm'
```

In all these cases, we have constructed an abstract filter. We can now apply it 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. 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)
> kmfilt.results <- filter(fset.trans, kmfilt)
> bvnormfilt.results <- filter(fset.trans, bvnormfilt)
> summary(rectgate.results[[1]])

Rectangle: 466 of 2205 (21.13%)
```

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: 1198 of 2205 (54.33%)
High: 1007 of 2205 (45.67%)

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

$a01
Low: 1198 of 2205 (54.33%)
High: 1007 of 2205 (45.67%)

$a02
Low: 2434 of 13350 (18.23%)
High: 10916 of 13350 (81.77%)

$a03
Low: 2701 of 11610 (23.26%)
High: 8909 of 11610 (76.74%)

$a04
Low: 3977 of 13830 (28.76%)
High: 9853 of 13830 (71.24%)
```

5.2 Subsetting and splitting

To subset or split a *flowFrame* or *flowSet*, we use the Subset() and split() methods respectively. Subset() is used for logical (`TRUE`/`FALSE`) filters, while split() is used for filters that detect multiple populations. 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)
> fset.trans[[1]]

flowFrame object with 2205 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A> <F
slot 'description' has 150 elements

> smaller[[1]]

flowFrame object with 1636 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A> <F
slot 'description' has 150 elements
```

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 with 948 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A>
slot 'description' has 150 elements

$High
flowFrame object with 688 cells and 8 observables:
<FSC-H> FSC-Height <SSC-H> SSC-Height <FL1-H> CD15 FITC <FL2-H> CD45 PE <FL3-H> CD14 PerCP <FL2-A>
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, e.g., if there are more than one population (Figure 5.3 a):

```
> xyplot(`FSC-H` ~ `SSC-H`, fset.trans, subset = (aliquot == "H" & time == 46),
         filter = bvnormfilt)
```

However, we might use the `kmfilt` filter to divide the sample up initially (Figure 5.3 b):

```
> xyplot(`FSC-H` ~ `SSC-H`, fset.trans, subset = (aliquot == "H" & time == 46),
         smooth = FALSE, pch = ".", cex = 2, filter = kmfilt)
```

Exercise 13
Apply the `kmfilt` *filter to each frame and extract the* `"High"` *population (hint: use* `fsApply`*). Apply* `bvnormfilt` *to the result, and compare it to the unfiltered case. Specifically, look at the subset corresponding to Aliquot H.*

Solution:

```
> fset.high <- fsApply(fset.trans, function(x) split(x, kmfilt)$High)
> xyplot(`FSC-H` ~ `SSC-H` | factor(time), fset.high, subset = aliquot == "H",
         filter = bvnormfilt)
> xyplot(`FSC-H` ~ `SSC-H` | factor(time), fset.trans, subset = aliquot == "H",
         filter = bvnormfilt)
```

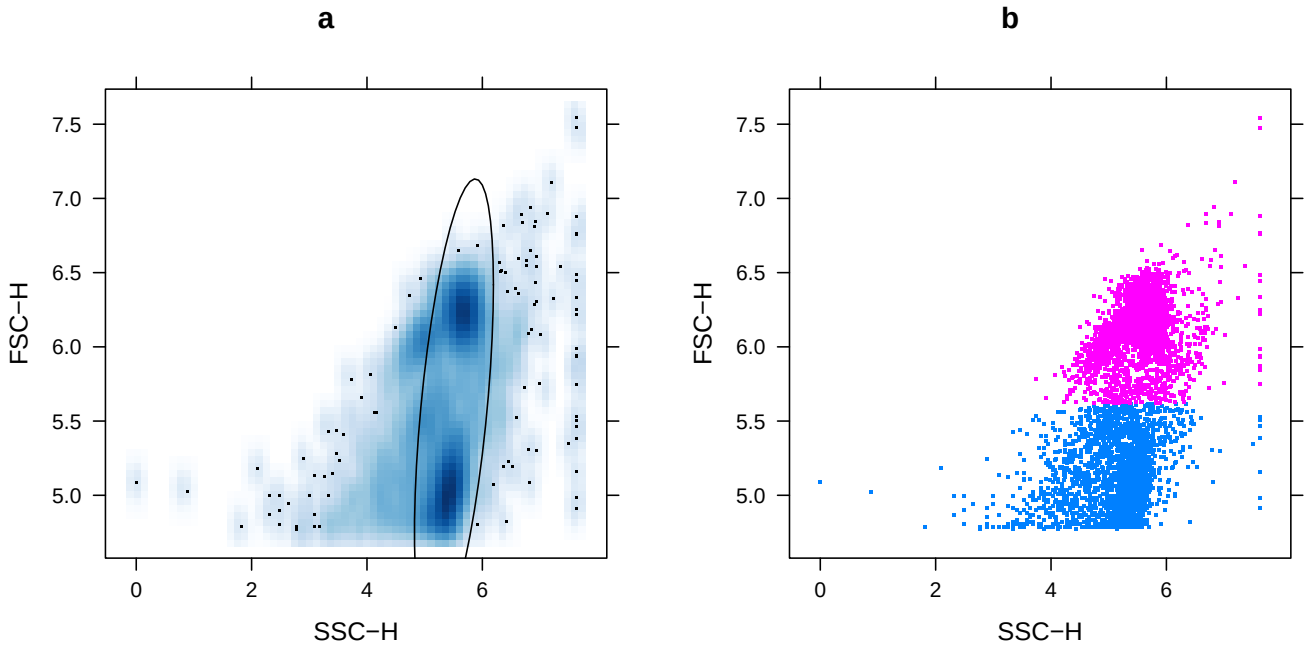

Figure 5: Scatter plot of FSC-H and SSC-H with two different data-driven filters

```
> sessionInfo()

R version 2.6.0 Under development (unstable) (2007-07-30 r42359)
i686-pc-linux-gnu

locale:
LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF-8;LC_MONETARY=en_US.UTF-8;

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

other attached packages:
[1] flowViz_1.1.4      MASS_7.2-35        geneplotter_1.13.8 lattice_0.16-2
[5] annotate_1.13.8    rgl_0.70           flowCore_1.1.8     rrcov_0.3-05
[9] Biobase_1.13.47

loaded via a namespace (and not attached):
 [1] AnnotationDbi_0.0.56 cluster_1.11.7       DBI_0.2-1
 [4] graph_1.13.2         grid_2.6.0           KernSmooth_2.22-21
 [7] latticeExtra_0.2-2   RColorBrewer_0.2-3   RSQLite_0.5-3
[10] stats4_2.6.0
```

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.