flowFP: Fingerprinting for Flow Cytometry

H. Holyst and W. Rogers

October 17, 2016

Abstract

Background A new software package called flowFP for the analysis of flow cytometry data is introduced. The package, which is tightly integrated with other Bioconductor software for analysis of flow cytometry, provides tools to transform raw flow cytometry data into a form suitable for direct input into conventional statistical analysis and empirical modeling software tools. The approach of flowFP is to generate a description of the multivariate probability distribution function of flow cytometry data in the form of a "fingerprint". As such, it is independent of a presumptive functional form for the distribution, in contrast with model-based methods such as Gaussian Mixture Modeling. flowFP is computationally efficient, and able to handle extremely large flow cytometry data sets of arbitrary dimensionality. Algorithms and software implementation of the package are described.

Methods flowFP implements several S4 classes and methods, and rests upon the flowCore foundation classes for flow cytometry data.

Results Use of the software is exemplified with applications to data quality control and to the automated classification of Acute Myeloid Leukemia.

keywords Flow cytometry, fingerprinting, high throughtput, software

1 Introduction

Flow cytometry (FC) produces multi-dimensional biological information at the level of the cellular compartment, and over very large numbers of cells. As such it is ideally suited for a wide variety of investigations for which cellular context and large sample observations are important. In recent years the technology of FC has undergone appreciable development (**??**) with the introduction of digital signal processing electronics (**?**), multiple lasers, increasing numbers of fluorescence detectors, and robotic automation, both in sample preparation (**?**) and in instrumental data collection (**?**). The recent development of new reagents (**?**) that enable increasing assay complexity has also been rapid and accelerating. Given the scope and pace of these developments, the bottleneck in many FC experiments has shifted from the wet laboratory to the computer laboratory; that is to say, data analysis(**?**).

FC data are typically analyzed using graphically-driven approaches. Subsets of cells (events) are delineated usually in one- or two-dimensional histograms or "dotplots" in a procedure termed "gating". The gating process is often applied in a sequential fashion, with the numbers

of events inside successive gates falling monotonically from step to step. Subsets determined via gating are typically then quantified with respect to their expression patterns in the dimensions of multiparameter space not utilized for gating, often by simply counting proportions of the subsets that are positive or negative for each of the markers of interest for that subset. Several commercially available software packages have been extensively optimized to support this kind of visually-guided analysis workflow, for example, FlowJo (Treestar Inc, Ashland, OR), WinList (Verity Software House, Topsham, ME) and FCSExpress (De Novo Software, Los Angeles, CA).

Despite near-universality of this data analysis approach within the flow cytometry community, the procedure has three main drawbacks. First, the choice of gates is often subjective, particularly in the not-unusual situation where the gating distribution is broad and smooth. This leads to an inherent lack of reproducibility from sample-to-sample, or even for the re-analysis of the same sample, as well as presenting audit trail difficulties that compromise verifiability of results. Second, because gates are specified by manually drawing regions on a graph using a computer mouse, the process is very labor-intensive and time-consuming, and in most cases takes many times longer than the actual acquisition of the data. Finally, because gating and analytical regions are determined by the data analyst based on his or her experience, there may be interesting and informative features that exist within the full, un-gated multivariate distribution of events but that nevertheless escape detection in this analysis paradigm.

flowFP is designed to address these limitations in conventional approaches to the analysis of FC data. The broad aim of flowFP is to directly transform raw FC list-mode data into a form suitable for direct input to other statistical analysis and empirical modeling tools. Thus, it is useful to think of flowFP as an intermediate step between the acquisition of high- throughput FC data on the one hand, and empirical modeling, machine learning, and knowledge discovery on the other.

2 Algorithm Description

flowFP implements and integrates ideas put forth in (**?**) and (**?**). FlowFP utilizes the Probability Binning (PB) algorithm (**?**) to subdivide multivariate space into hyper-rectangular regions that contain nearly equal numbers of events. Regions (bins) are determined by (a) finding the parameter whose variance is highest, (b) dividing the population at the median of this parameter which results in two bins, each with half of the events, and (c) repeating this process for each subset in turn. Thus, at the first level of binning the population is divided into two bins. At the second level, each of the two "parent" bins is divided into two "daughter" bins, and so forth. The final number of bins n is determined by the number of levels l of recursive subdivision, such that $n = 2^l$.

This binning procedure is typically carried out for a collection of samples (instances), called a "training set". The result of this process is in essence a description of the subdivision of a multiparametric space into sub-regions, and is thus termed a "model" of the space (not to be confused with modeling approaches that fit data to a parameterized model or set of models).

The model is then applied to another set of samples (which may or may not include instances from the training set). This operation results in a feature vector of event counts in each bin of the model for each instance in the set. These feature vectors are, in the context of a specific model, a unique description of the multivariate probability distribution function for each instance in the set of samples, and thus are aptly referred to as "fingerprints".

Although flowFP generates bins using the PB algorithm, the way it utilizes the resulting fingerprints is similar to the methods described in (**?**). Each element of a fingerprint represents the number of events in a particular sub-region of the model. Although it may not be known *a priori* which of these regions are informative with respect to an experimental question, it is possible to determine this by using appropriate statistical tests, along with corrections for multiple comparisons, to ascertain which regions (if any) are differentially populated in two or more groups of samples. If we regard the number of events in a bin as one of n features describing an instance, then the statistical determination of informative sub-regions is clearly seen to be a feature selection procedure.

Fingerprint features are useful in two distinct modes. First, all or a selected subset of features can be used in clustering or classification approaches to predict the class of an instance based on its similarity to groups of instances. Second, the events within selected, highly informative bins can be visualized within their broader multivariate context in order to interpret the output of the modeling process. This step is crucial in that it provides a means to develop new hypotheses for FC-derived biomarkers within the context of existing reagent panels.

3 Fingerprint Representation

FlowFP is one of a growing number of Bioconductor packages integrated within the framework provided by flowCore and is thus able to interoperate with other flowCore-compliant tools as well as with the full range of downstream statistical analysis and machine learning tools available in R. This integration enables flexible creation of powerful high-throughput analysis procedures for large FC data sets.

FlowFP uses the S4 object-oriented facility of R. Computationally intensive parts are written in the C programming language for efficiency. FlowFP is built around a set of three S4 classes, each with a constructor of the same name as the class name. In addition there are a number of methods for accessing, manipulating and visualizing the data in each of the classes.

3.1 The *flowFPModel* Class

flowFPModel is the fundamental class for the flowFP package. The `flowFPModel` constructor takes a collection of one or more list-mode instances which are represented in the flowCore framework as a *flowFrame* (for a single instance) or a *flowSet* (for a collection of instances), respectively (henceforth we shall refer to *flowFrame*s and *flowSet*s, the original list-mode data being implied). In addition to the required argument, `flowFPModel` has optional arguments

that allow control over the number of levels of recursive subdivision and the set of parameters to be considered in the binning process. By default all parameters in the input *flowSet* are considered, but if this argument is provided, any parameters not listed are ignored. The constructor emits an object of type *flowFPModel*, which encapsulates a complete representation of the binning process that is used later to construct fingerprints.

To see how this works, let's build a *flowFPModel* for a small data set. *fs1* is a *flowSet* comprised of 7 *flowFrame*s, one for each tube in a sample. The tubes are stained with different antibody conjugates, but CD45-ECD is common to all of the tubes in order to support gating of the entire panel from one of the tubes using CD45 vs. SSC.

```
> data(fs1)
> fs1

A flowSet with 7 experiments.

  column names:
  FS Lin SS Log FL1 Log FL2 Log FL3 Log FL4 Log FL5 Log
```

One of the tubes (the first one) looks like this:

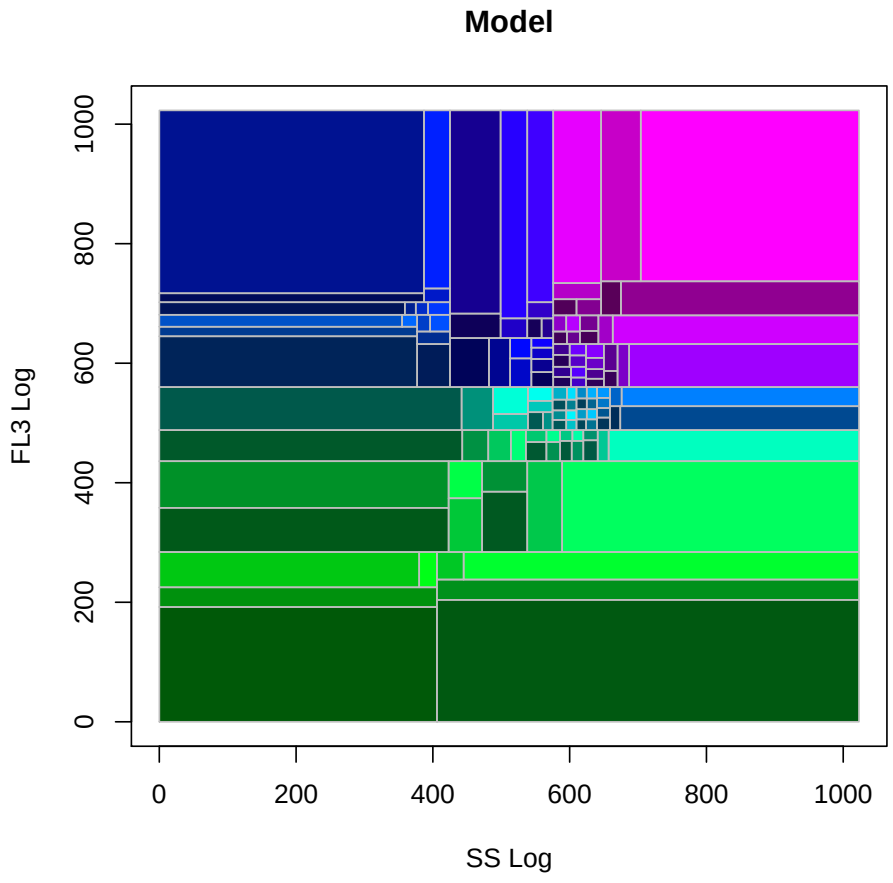
```
> fs1[[1]]

flowFrame object 'FI05_000942_001.LMD'
with 30000 cells and 7 observables:
       name      desc range minRange maxRange
$P1  FS Lin    FS Lin  1024   0.0000     1023
$P2  SS Log    SS Log  1024   0.1024     1023
$P3 FL1 Log IgG1-FITC  1024   0.0000     1023
$P4 FL2 Log   IgG1-PE  1024   0.0000     1023
$P5 FL3 Log  CD45-ECD  1024   0.0000     1023
$P6 FL4 Log  IgG1-PC5  1024   0.0000     1023
$P7 FL5 Log  IgG1-PC7  1024   0.0000     1023
166 keywords are stored in the 'description' slot
```

Let's construct a model, using SSC (parameter 2) and CD45 (parameter 5). We will specify the number of recursions to be 7, resulting in $2^7 = 128$ bins in the model:

```
> mod <-  flowFPModel(fs1, name="CD45/SS Model", parameters=c(2,5), nRecursions=7)
> show(mod)

A flowFPModel:

  Name = CD45/SS Model
  nRecursions (max) = 7 (7)
```

```
  Dequantize = TRUE
  Parameters Considered:
    SS Log, FL3 Log
  Parameters Used:
    SS Log, FL3 Log
  Training Set:
    FI05_000942_001.LMD
    FI05_000942_002.LMD
    FI05_000942_003.LMD
    FI05_000942_004.LMD
    FI05_000942_005.LMD
    FI05_000942_006.LMD
    FI05_000942_007.LMD
```

```
> plot(mod)
```



Usage and argument descriptions for `flowFPModel` are as follows:

Usage:

`flowFPModel(obj, name="Default Model", parameters=NULL, nRecursions="auto", dequantize=TRUE, sampleSize=NULL, verbose=FALSE)`

obj	Training data for model, either a *flowFrame* or *flowSet*.
parameters	A vector of parameters to be considered during model construction. You may provide these as a vector of parameter indices (as shown above) or as a character vector. For example, parameters = c("SS Log", "FL3 Log") would yield the same result as shown in the example.
nRecursions	Number of times the FCS training data will be sub divided. Each recursion doubles the number of bins, so that $n_{bins} = 2^{nRecursions}$. A warning will be generated if the number of expected events in each bin is < 1. (e.g. if your training set had 1000 events, and you specified $nRecursions = 10$.)
dequantize	If TRUE, all of the event values in the training set will be made unique by adding a tiny value (proportional to the ordinal position of each event) to the data.
sampleSize	Used to specify the per-*flowFrame* sample size of the data to use in model generation. If *NULL*, all of the data in x is used. Setting this to a smaller number will speed up processing, at the cost of accuracy.
name	A descriptive name assigned to the model.
verbose	If TRUE, prints out information as it constructs the model. Useful for debugging.

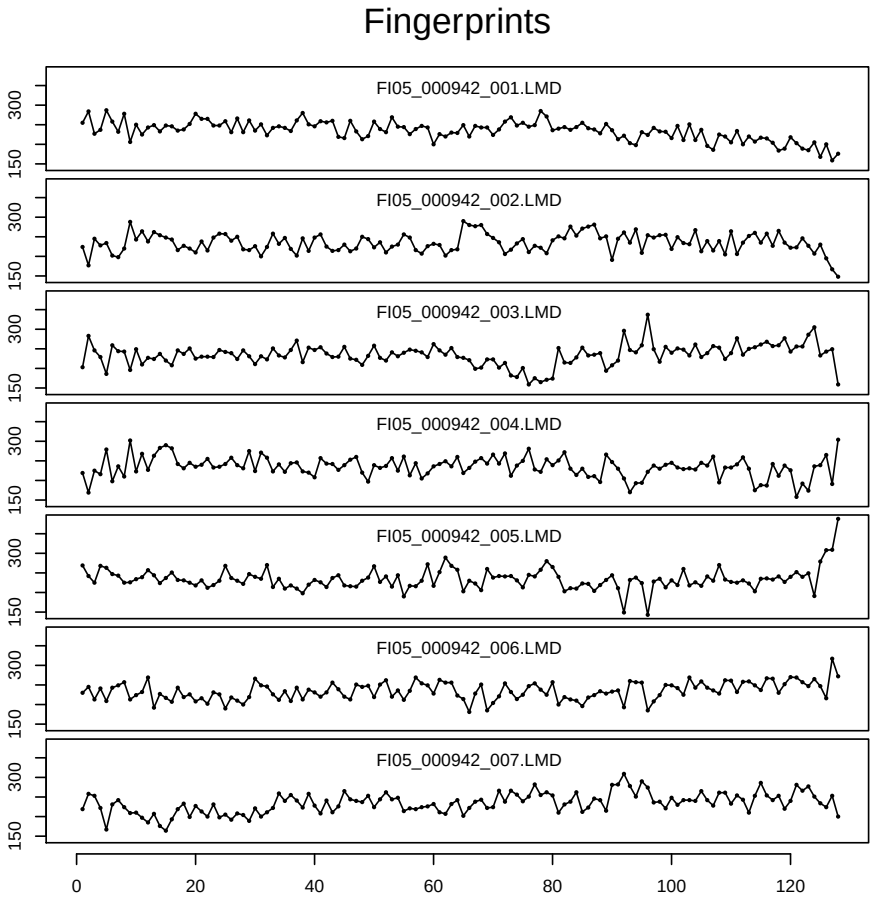
3.2 The *flowFP* Class

The `flowFP` constructor takes a *flowFrame* or a *flowSet* as its only required argument, and an optional *flowFPModel*. If no *flowFPModel* is supplied, `flowFP` computes a model (by calling `flowFPModel` internally). Regardless the source of the model, `flowFP` applies the model to each of the instances in its input. The resulting *flowFP* object extends the *flowFPModel* class and contains two additional important slots to store a matrix of counts and a list of tags. The counts matrix has dimensions $m \times n$, where m is the number of instances in the input *flowSet* (or one if a *flowFrame* is provided), and n is the number of features in the model. The tags slot is a list of m vectors, each of which has e elements, where e is the number of events in the corresponding frame in the input *flowSet*. The value for each element of the tag vector represents the bin number into which the corresponding event fell during the fingerprinting procedure. This is useful for visualization or gating based on fingerprints, as will be illustrated below.

A set of fingerprints is obtained by applying the model to a *flowSet*. In this case we will apply the model derived from *fs1* to the same *flowSet*:

```
> fp <- flowFP (fs1, mod)
> show(fp)

A flowFP containing 7 instances with 128 features.
```

```
[1] "FI05_000942_001.LMD" "FI05_000942_002.LMD" "FI05_000942_003.LMD"
[4] "FI05_000942_004.LMD" "FI05_000942_005.LMD" "FI05_000942_006.LMD"
[7] "FI05_000942_007.LMD"

Extends A flowFPModel:

  Name = CD45/SS Model
  nRecursions (max) = 7 (7)
  Dequantize = TRUE
  Parameters Considered:
    SS Log, FL3 Log
  Parameters Used:
    SS Log, FL3 Log
  Training Set:
    FI05_000942_001.LMD
    FI05_000942_002.LMD
    FI05_000942_003.LMD
    FI05_000942_004.LMD
    FI05_000942_005.LMD
    FI05_000942_006.LMD
    FI05_000942_007.LMD

> plot (fp, type="stack")
```



Usage and argument descriptions for `flowFP` are as follows:

Usage:

```
flowFP(obj, model=NULL, sampleClasses=NULL, verbose=FALSE, ...)
```

fcs	A *flowFrame* or *flowSet* for which fingerprint(s) are desired.
model	A model generated with the `flowFPModel` constructor, or *NULL*. If *NULL*, a default model will be silently generated from all instances in x.
sampleClasses	An optional character vector describing modeling classes. If supplied, there must be exactly one element for each *flowFrame* in the *flowSet* in x (see Details).
verbose	If TRUE, prints out information as it constructs the fingerprint and possibly the model. Useful for debugging.
...	If model is *NULL*, additional arguments are passed on to the model constructor. see `flowFPModel` for details.

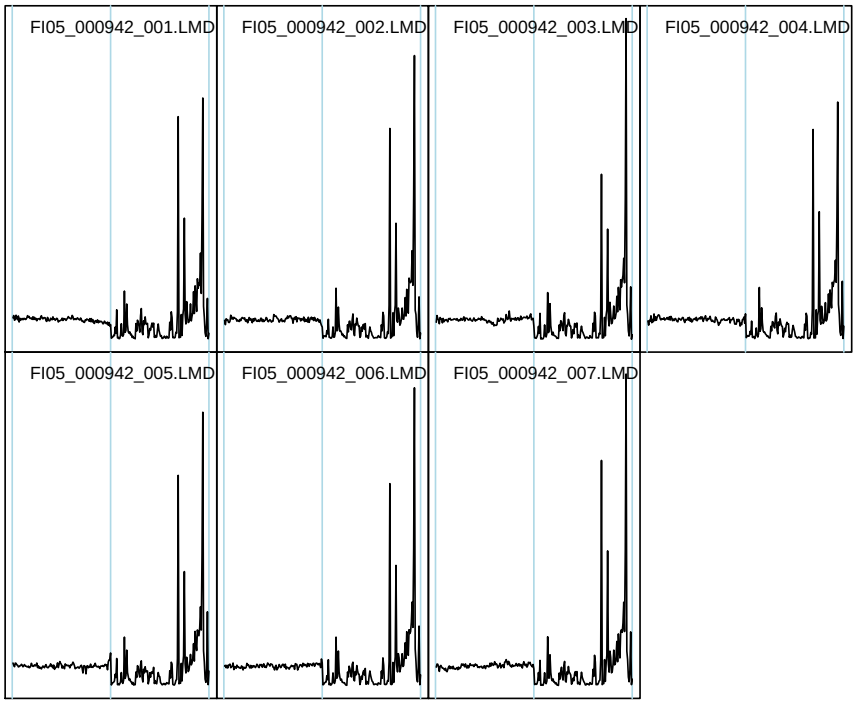
3.3 The *flowFPPlex* Class

The *flowFPPlex* is a container object which facilitates combining, processing and visualizing large collections of flowFP objects which are all derived from the same set of instances. The `flowFPPlex` constructor takes a list of *flowFP* objects. The *flowFPPlex* manages the logical association of a set of *flowFP* descriptions. In particular, it extends the counts matrices of its members "horizontally" so as to create a unified representation of the entire collection of fingerprints.

For example, let's load data for another sample, similar to *fs1*. We will then use both *flowSet*s as model bases, and fingerprint *fs1* with respect to both of them. Then we'll load both sets of fingerprints into a *flowFPPlex* and visualize the result:

```
> data(fs2)
> mod1 <-  flowFPModel(fs1, name="CD45/SS Model vs fs1", parameters=c("SS Log", "FL3 Log"), n
> mod2 <-  flowFPModel(fs2, name="CD45/SS Model vs fs2", parameters=c("SS Log", "FL3 Log"), n
> fp1_1 <- flowFP (fs1, mod1)
> fp1_2 <- flowFP (fs1, mod2)
> plex <- flowFPPlex(c(fp1_1, fp1_2))
> plot (plex, type='grid', vert_scale=10)
```

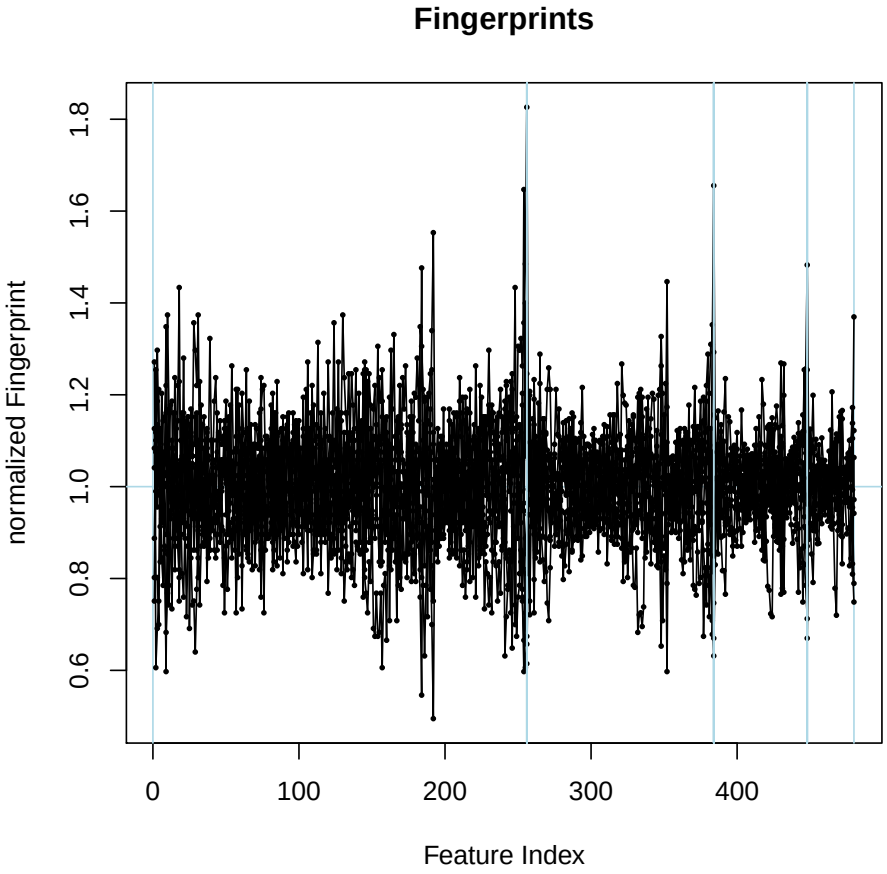
Fingerprints



In the figure, the light blue vertical lines show the division of the fingerprints resulting from the two models. Notice that, as one might expect, when fingerprinting *fs1* against a model constructed from itself, the deviations from the norm are small, whereas when fingerprinting *fs1* against a model constructed from a different *flowSet*, the deviations in the fingerprint values are large.

We might also wish to use the *flowFPPlex* to facilitate exploration of the effect on fingerprints due to variation of the number of levels of recursion.

```
> fp <- flowFP (fs1, param=c("SS Log", "FL3 Log"), nRecursions=8)
> plex <- flowFPPlex()
> for (levels in 8:5) {
+         nRecursions(fp) <- levels
+         plex <- append (plex, fp)
+ }
> plot (plex, type="tangle", transformation="norm")
```



Notice that as the fingerprint resolution (determined by the number of recursions) is reduced from left to right, the number of features in each fingerprint falls, but the size of the variations from the norm also falls. Evidently, the more local the modeling, the larger the variations from instance to instance we can expect.

Also notice a couple of other features we illustrate in this example. First, we only computed the model once, but we can change the effective number of recursions (and thus the resolution of the fingerprints) to any integer less than that at which the model was computed, using the accessor function `nRecursions`. Second, we can initialize an empty *flowFPPlex*, and then use the function `append` to add *flowFP*s one at a time.

Usage and argument descriptions for `flowFPPlex` are as follows:

Usage:

```
flowFPPlex(fingerprints=NULL)
```

fingerprints	List of *flowFP*s.

3.4 Generic functions

A number of other methods have been provided to facilitate interaction with and analysis of fingerprinting results. Chief among these are visualization methods that aid in the understanding and interpretation of fingerprinting results. They are provided as overloads to the generic `plot` function. In addition, a few other accessor methods deserve special mention.

nRecursions(obj). This generic function returns the number of levels of recursive subdivision of its argument. FlowFP, flowFPPlex and flowFPModel all implement the method. Furthermore, the flowFP class implements the "set" method. This enables the user to compute a model at some fairly high resolution, and then to derive fingerprints at that resolution or any lower resolution without re-computing the model. This is possible because fingerprinting is recursive, so that given any high-resolution model, all models of lower resolution can be derived from it.

counts(obj). This generic function returns a matrix of the number of events per instance and per bin. FlowFP and flowFPPlex classes implement this method, facilitating creation of fingerprint matrices suitable for processing by downstream methods outside of the flowFP package. The method has an optional argument "transformation" that can take on values "raw" (returns the actual event counts for each bin), "normalize" (normalizes by dividing raw counts by the expected number of events), or "log2norm" (like normalize except that it further takes the log2 of the result).

sampleNames(obj) and sampleClasses(obj). These generic functions set or get sample identifiers for objects of class flowFP or flowFPPlex. By default, for flowFPs, sample names are derived from the flowSet. However they can be overridden by the set method, providing flexibility to handle cases where the sample names in a flowSet are not appropriate. When adding fingerprints to a flowFPPlex, sample names, and if present sample classes, are compared, and the join operation is not permitted unless names and classes among all fingerprints in the flowFPPlex are identical.

parameters(obj). This generic function returns the light scatter and/or fluorescence parameters involved in binning, either for a flowFPModel or a flowFP. The function is able to report both the parameters that were considered for binning as well as those that actually participating (i.e. ones that were subdivided at during recursive subdivision).

tags(fp). This generic function returns the tags slot of a flowFP object. This is useful for visualization and gating operations.

binBoundary(obj). This generic function reports a list of multivariate rectangles corresponding to the limits of the bins. FlowFP and flowFPModel classes both implement this

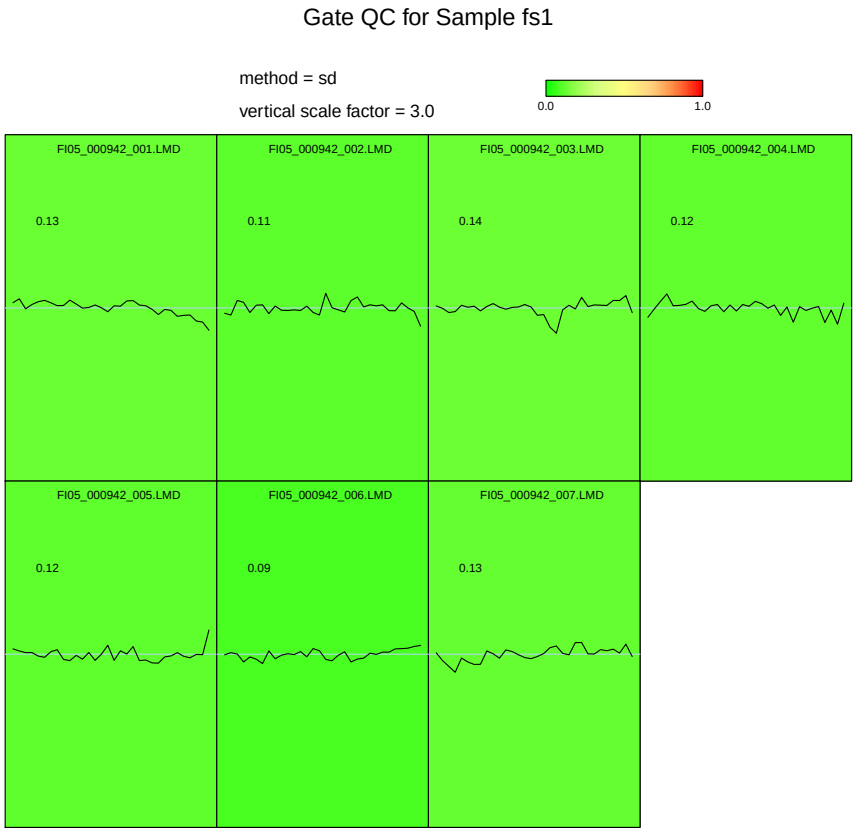
method. This information is also useful for visualization and gating operations.

4 Fingerprinting for Gating Quality Control

As alluded to in Section 3.1, a common practice, especially in some clinical settings, is the collection of data in several aliquots, each stained with different reagent cocktails in order to see all of the markers of interest, but including in all of the tubes at least one common marker. Using parameters common to all of the tubes (CD45 and SSC are frequently used for this purpose (**?**)) subsets of cells can be delineated by drawing gates on one tube and then applying the gates to all of the tubes. This saves time, but relies on the assumption that the probability distribution is stationary over all of the tubes. If this assumption is not valid, subsetting errors will occur, but may not be readily apparent without careful study of the gating plots.

Using flowFP, in order to rapidly detect consistency of CD45 vs. SSC distributions without the need to look at dotplots, we can fingerprint a collection of tubes and look for outliers.

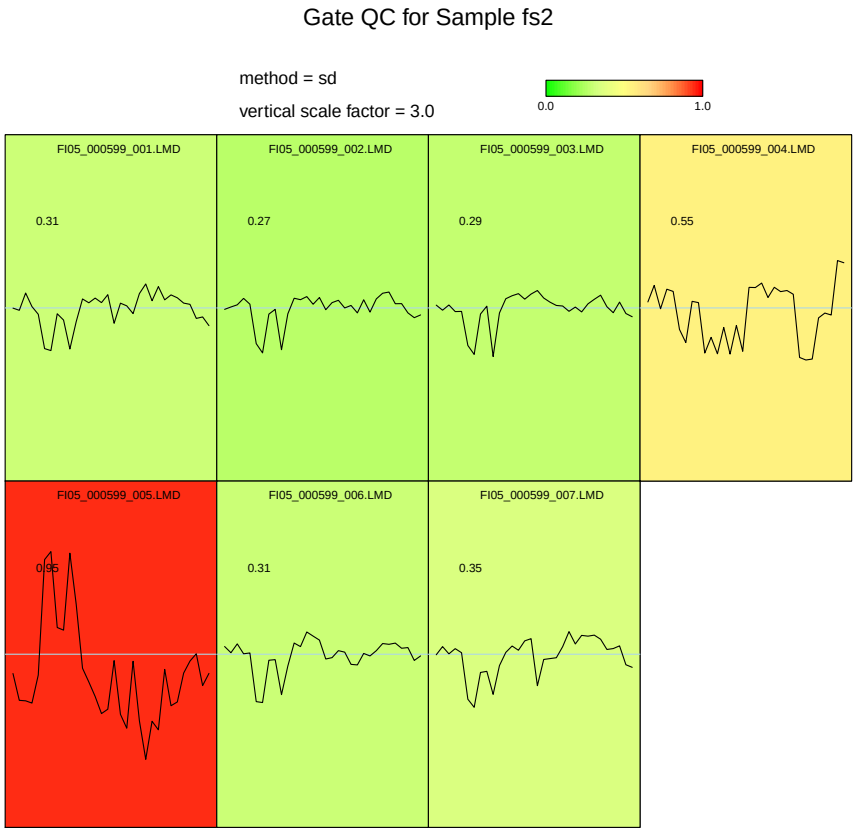
```
> fp1 <- flowFP (fs1, parameters=c("SS Log", "FL3 Log"), name="self model: fs1", nRecursions=
> plot (fp1, type="qc", main="Gate QC for Sample fs1")
```



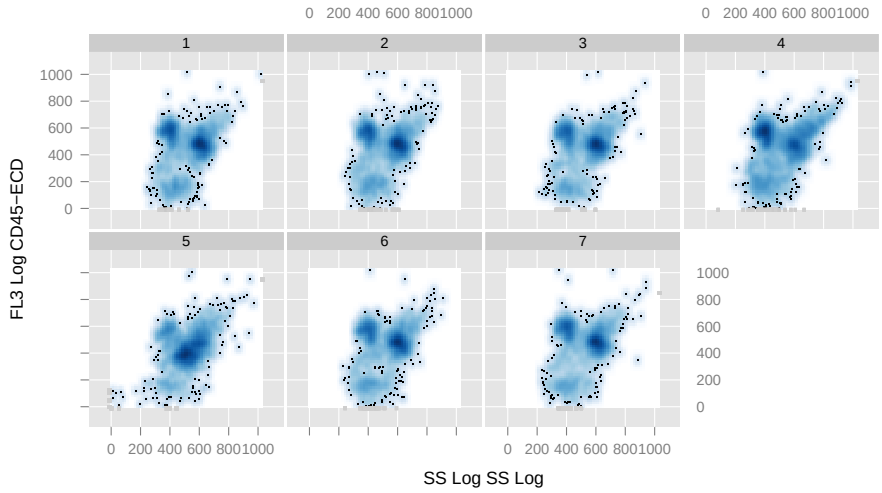
In this plot the fingerprints for each of the 7 tubes is shown in a grid. The color of the grid square for a tube indicates the standard deviation of the normalized and log-transformed fingerprint feature vector for that tube according to the color scale at the top of the figure. The standard deviation value is printed in the grid square.

The following figure shows an example where the gate deviation is large. Note the fact that tube 4 and especially tube 5 are the outliers. Also note how easy it is to spot this problem.

```
> fp2 <- flowFP (fs2, parameters=c("SS Log", "FL3 Log"), name="self model: fs2", nRecursions=
> plot (fp2, type="qc", main="Gate QC for Sample fs2")
```



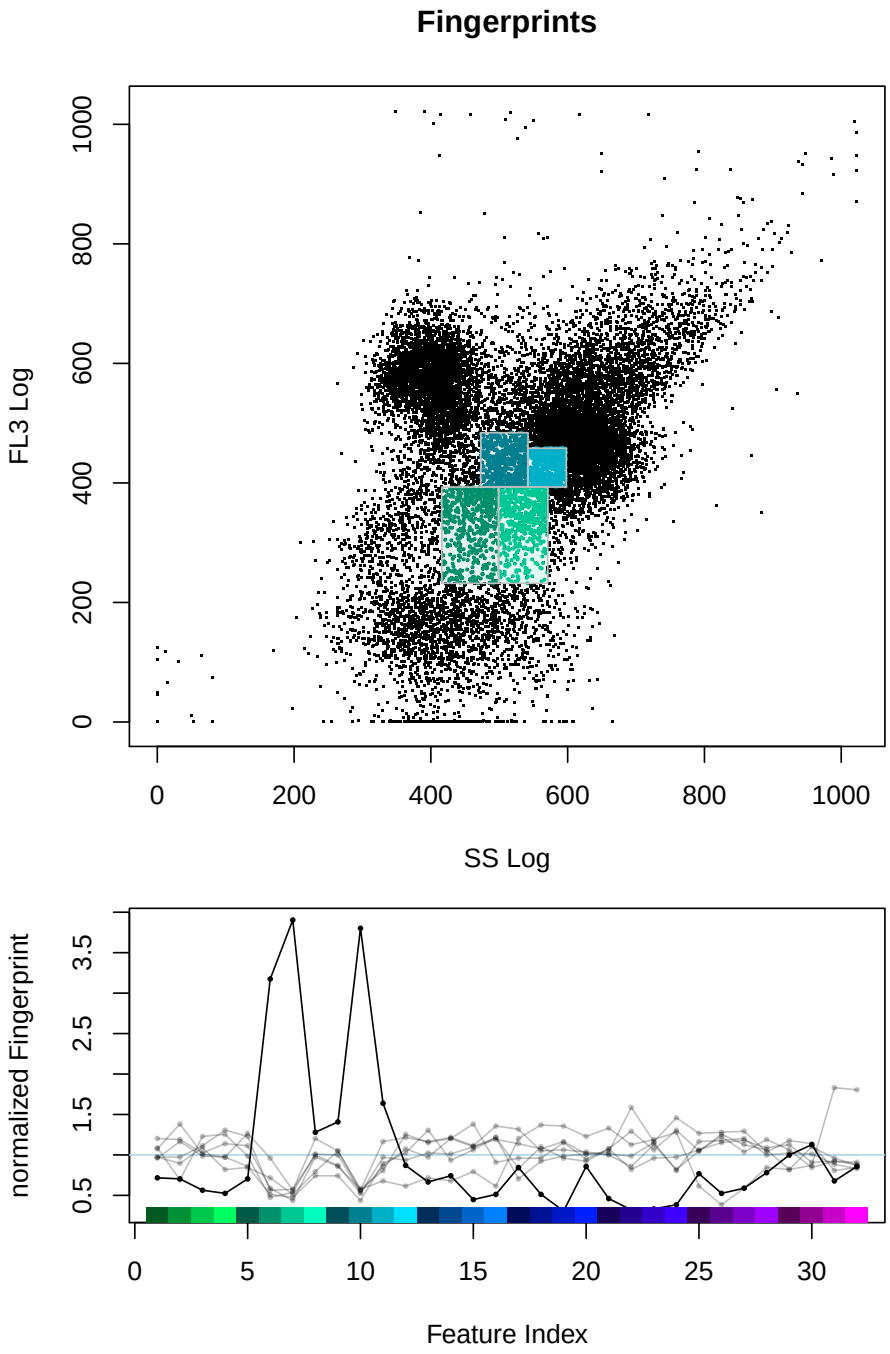

```
> xyplot (`FL3 Log` ~ `SS Log` | Tube, data=fs2)
```



In the above flowViz plot it is certainly possible to spot the inconsistency, but it's not so

easy as in the fingerprint-based QC picture. On the other hand ...

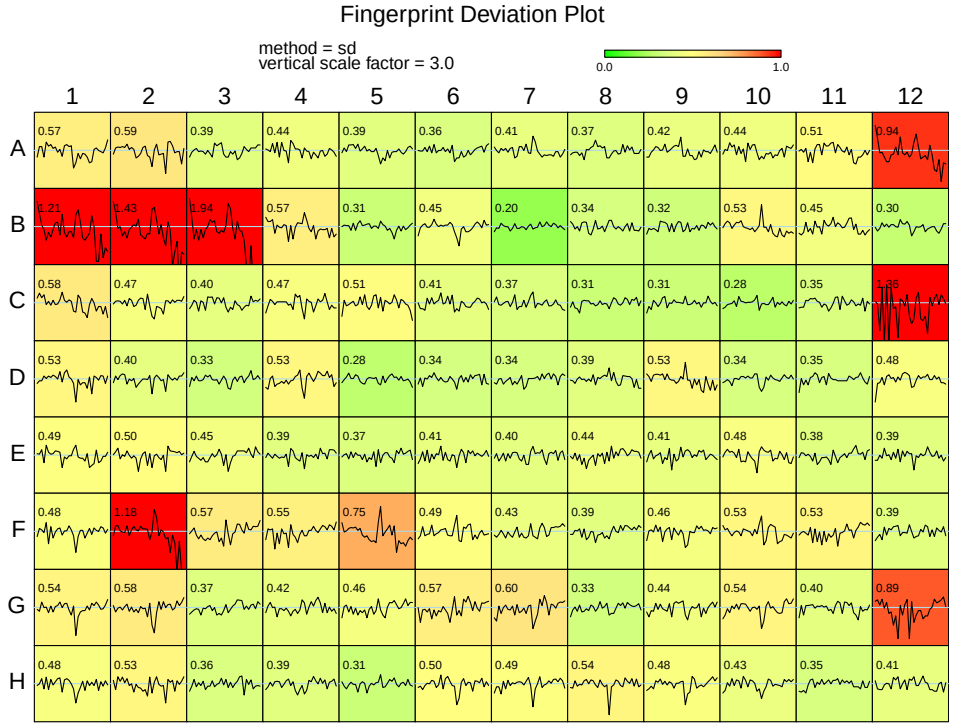
```
> plot (fp2, fs2, hi=5, showbins=c(6,7,10,11), pch=20, cex=.3, transformation='norm')
```



In this figure we can follow the fingerprint bins containing excess events in Tube 5 by way of the color map shown below the fingerprint. By comparing the bin indices 6, 7, 10 and 11 corresponding to green to blue-green colors, it's easy now to localize the place where Tube 5 differs from the rest.

Quality control for individual multi-tube samples is tedious, but not crazily impossible. 96-well plate data will drive you nuts with the need to examine gating data for each well and for many plates. Try this instead:

```
> data(plate)
> fp <- flowFP (plate, parameters=c("SSC-H","FL3-H","FL4-H"), nRecursions=5)
> plot (fp, type='plate')
```



This is a stimulation dataset, described in (**?**) (data were drastically sampled down to 1000 events per well so that they could be included in the package for illustration purposes). The original data are available at (**?**).

5 Limitations, Caveats and Comments

It is important to note that fingerprinting of FC data is not without limitations. First, we note that fingerprinting approaches are sensitive to differences in multivariate probability distributions no matter their origin. Thus, instrumental, reagent or other systematic variations may cause spurious signals as large, or larger than true biological effects. For this reason it is

important to measure and control for these effects(**?**). In fact, fingerprinting itself can be used to assess and to help control for systematic effects, as was illustrated in Section 4.

Second, because fingerprinting is, in essence, the creation of a multivariate histogram, it responds to factors that might artificially emphasize certain bins in preference to others. In particular, events may pile up on either the zero or full- scale axis for one or more parameters. This situation frequently results from values that would be negative due to compensation or background subtraction (causing pile-up on the zero axis) or at the other end of the scale, values that exceed the dynamic range of the signal detection apparatus causing pile-up at full scale. At either end this results falsely in an apparent high density of events. Fingerprinting bins are thus "attracted" to these locations, causing a distortion in the proper characterization of the true multivariate probability distribution function.

Just as scaling and transformation of data are important for visualization of multi- parameter distributions(**???**), so they are also important for fingerprinting. Data acquired using linear amplifiers such as exist in some modern instruments, or data that have been "linearized" from instruments with logarithmic amplifiers, tend to be heavily skewed to the left, since in most cases data distributions are quasi-log-normally distributed. Bins determined from such data thus have extreme variations in size. A good rule of thumb is to use a data transformation that produces the most spread-out distribution, which also is often the transformation most effective for clear visualization of the distribution.

A key limitation for fingerprinting approaches, including flowFP, relates to the number of events available for analysis. Since the objective of probability binning is to find bins containing equal numbers of events, it follows that once the number of bins is on the order of the number of events in an instance, the expected number of events per bin will be of order unity. In this case differences in bin counts will not be statistically significant. On the other hand, if the dimensionality of the data set is high, the average number of times any parameter will be divided in the binning process will be small. For example, in a dataset with 18 parameters, if we demand at least, say, 10 events per bin for statistical accuracy, about 2.6×10^6 events would be required in order that each parameter be divided on average into at least two bins. Thus, the spatial resolution of binning is limited by the number of events collected, and as the number of parameters increases, the number of events needed to maintain resolution increases geometrically.

Finally, although flowFP is computationally fast, because of the way that flow cytometric data are represented in R large datasets consume vast amounts of memory. If you need to process 96-well data for example, you will probably either need a machine with lots of memory (>4 GByte), or you will have to use some tricks, like sampling the data in order to reduce memory footprint. Fortunately, memory is cheap and 64-bit operating systems are becoming commonplace. For example, just reading in the data in (**?**) consumed 3.1 GB on a Linux 64-bit machine with 32 GB of memory. Fingerprinting required (briefly) an additional 2.5 GB for a total of 5.6 GB. However the whole process (reading in the data, computing the fingerprints, and displaying the result similar to the 96-well figure above) only took about 1 minute.

With recent technological advances, FC is now capable of operating as a true high-throughput

technique. A key enabling requirement is the need to automate data analysis for speed, much as automation in sample preparation and data acquisition have accelerated the rate of generation of data and thereby enabled high-throughput FC. This requirement inevitably drives movement away from human-drawn, visually-based gating which is the single most significant obstacle preventing a true high-throughput FC workflow. We hope you find flowFP a useful tool in your toolbox to help you achieve this goal.

6 Acknowledgements

We wish to express our deep gratitude to Jonni Moore and all of the people of the University of Pennsylvania Flow Cytometry Resource. We thank Florian Hahne, Nolwenn Le Meur and Ryan Brinkman for advice and assistance in programming in R and integration with flowCore. We are most especially grateful to Clarient, Inc. for generously making available data sets used to illustrate the utility of flowFP.