Getting started with flowStats

F. Hahne, N. Gopalakrishnan

October 26, 2021

Abstract

flowStats is a collection of algorithms for the statistical analysis of flow cytometry data. So far, the focus is on automated gating and normalization.

1 Introduction

Since flowStats is more a collection of algorithms, writing a coherent Vignette is somewhat difficult. Instead, we will present a hypothetical data analysis process that also makes heavy use of the functionality provided by flowWorkspace, primarily through `GatingSet`s.

We start by loading the `ITN` data set

```
> library(flowCore)
> library(flowStats)
> library(flowWorkspace)
> library(ggcyto)
> data(ITN)
```

The data was acquired from blood samples by 3 groups of patients, each group containing 5 samples. Each *flowFrame* includes, in addition to FSC and SSC, 5 fluoresence parameters: CD3, CD4, CD8, CD69 and HLADR.

First we need to tranform all the fluorescence channels. This is a good point to start using a *GatingSet* object to keep track of our progress.

```
> require(scales)
> gs <- GatingSet(ITN)
> trans.func <- asinh
> inv.func <- sinh
> trans.obj <- trans_new("myAsinh", trans.func, inv.func)
> transList <- transformerList(colnames(ITN)[3:7], trans.obj)
> gs <- transform(gs, transList)
```

In an initial analysis step we first want to identify and subset all T-cells. This can be achieved by gating in the CD3 and SSC dimensions. However, there are several other sub-populations and we need to either specify our selection further or segment the individual sub-populations. One solution for the latter aproach is to use the mixture modelling infrastructure provided by the flowClust package. However, since we are only interested in one single sub-population, the

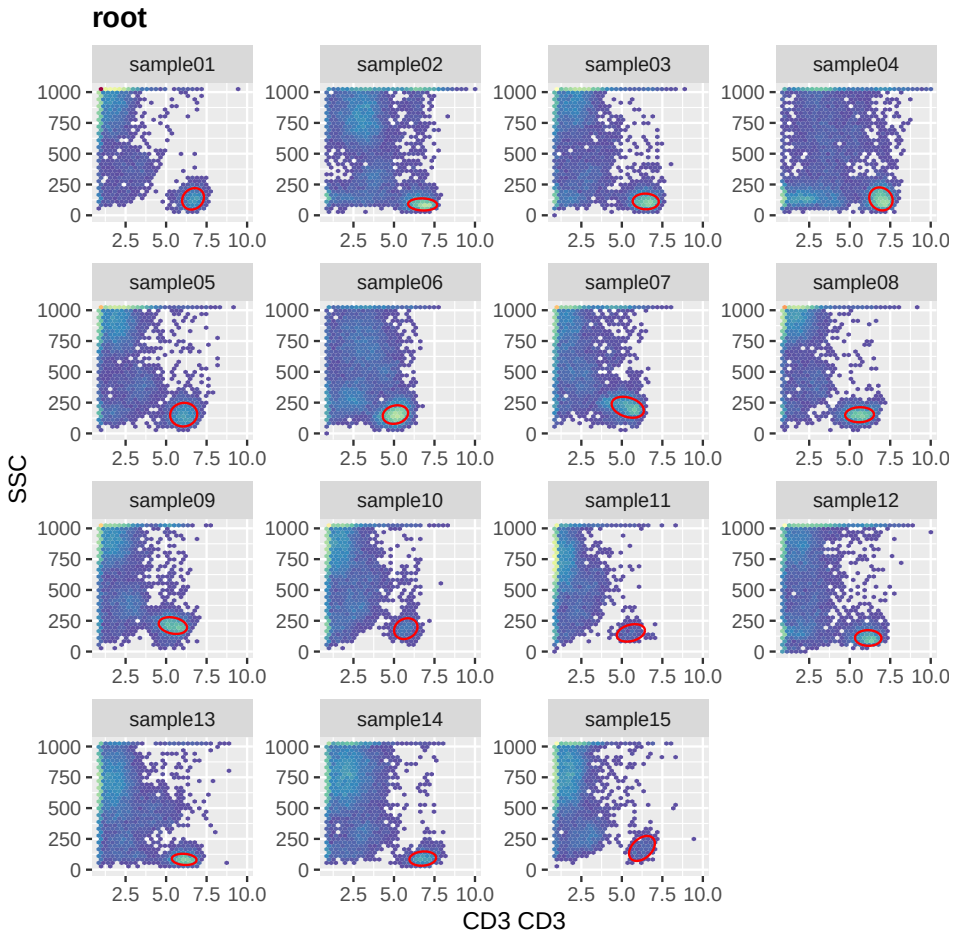
T-cell, it is much faster and easier to use the `lymphGate` function in the flowStats package. The idea here is to first do a rough preselection in the two-dimensional projection of the data based on expert knowledge or prior experience and subsequently to fit a *norm2Filter* to this subset. The function also allows us to derive the pre-selection through back-gating: we know that CD4 positive cells are a subset of T-cells, so by estimating CD4 positive cells first we can get a rough idea on where to find the T-cells in the CD3 SSC projection.

```
> lg <- lymphGate(gs_cyto_data(gs), channels=c("CD3", "SSC"),
+                 preselection="CD4", filterId="TCells",
+                 scale=2.5)
> gs_pop_add(gs, lg)
```

```
[1] 2
```

```
> recompute(gs)
```

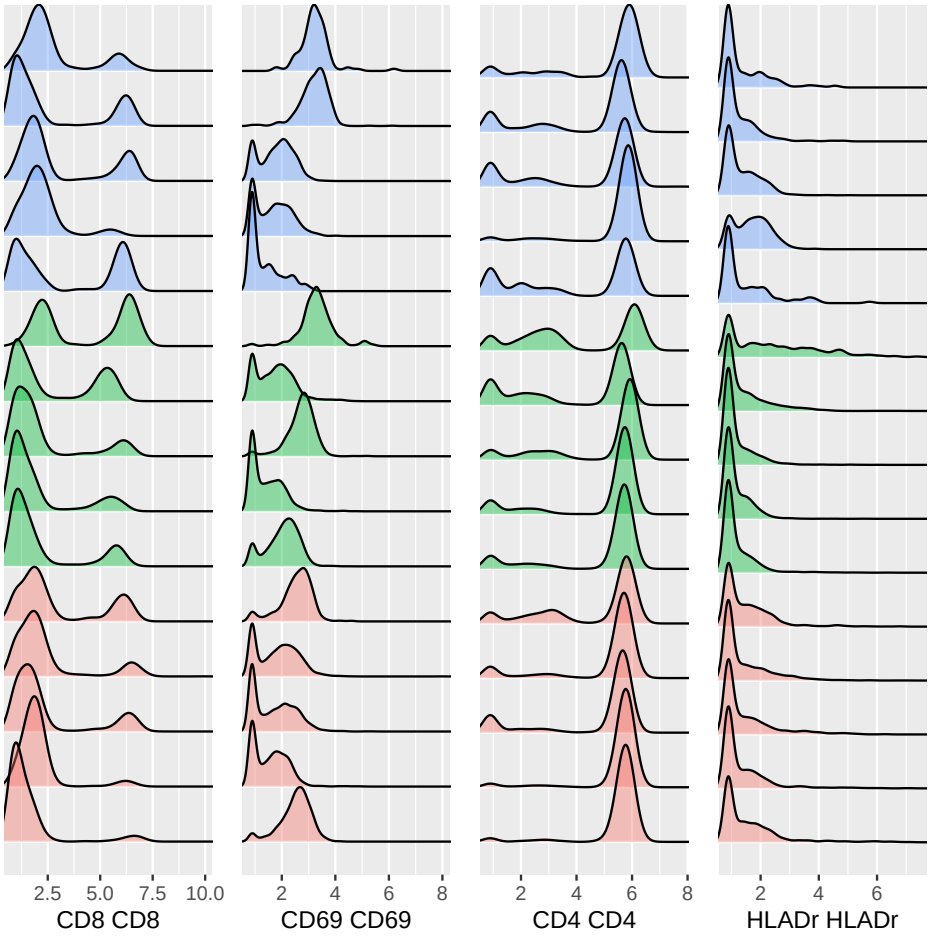
```
> ggcyto(gs, aes(x = CD3, y = SSC)) + geom_hex(bins = 32) + geom_gate("TCells")
```



In the next step we want to separate T-helper and NK cells using the CD4 and CD8 stains. A convenient way of doing this is to apply a *quadGate*, assuming that both CD4 and CD8 are

binary markers (cells are either positive or negative for CD4 and CD8). Often investigators use negative samples to derive a split point between the postive and negative populations, and apply this constant gate on all their samples. This will only work if there are no unforseen shifts in the fluorescence itensities between samples which are purely caused by technical variation rather than biological phenotype. Let's take a look at this variation for the T-cell subset and all 4 remaining fluorescense channels:

```
> require(ggridges)
> require(gridExtra)
> pars <- colnames(gs)[c(3,4,5,7)]
> data <- gs_pop_get_data(gs, "TCells")
> plots <- lapply(pars, function(par)
+   as.ggplot(ggcyto(data, aes(x = !!par, fill = GroupID)) +
+     geom_density_ridges(mapping = aes(y = name), alpha = 0.4) +
+     theme(axis.title.y = element_blank(),
+           axis.text.y = element_blank(),
+           axis.ticks.y = element_blank()) +
+     facet_null()
+   ))
> do.call("grid.arrange", c(plots, nrow=1))
```



Indeed the data, especially for CD4 and CD8, don't align well. At this point we could decide to compute the *quadGates* for each sample separately. Alternatively, we can try to normalize the data and then compute a common gate. The `warpSet` function can be used to normalize data according to a set of landmarks, which essentially are the peaks or high-density areas in the density estimates shown before. The ideas here are simple:

- High density areas represent particular sub-types of cells.
- Markers are binary. Cells are either positive or negative for a particular marker.
- Peaks should aline if the above statements are true.

The algorithm in `warpSet` performs the following steps:

1. Identify landmarks for each parameter using a `curv1Filter`
2. Estimate the most likely total number (k) of landmarks
3. Perform k-means clustering to classify landmarks
4. Estimate warping functions for each sample and parameter that best align the landmarks, given the underlying data. This step uses functionality from the `fda` package.

5. Transform the data using the warping functions.

The algorithm should be robust to missing peaks in some of the samples, however the classification in step 3 becomes harder since it is not clear which cell population it represents.

While `warpSet` can be used directly on a `flowSet`, the `normalize` method was written to integrate with `GatingSet` objects. We must first create the gate whose channels we would like to normalize, which in this case will be done using the `quadrantGate` function to automatically estimate a `quadGate` in the CD4 and CD8 channels.

```
> qgate <- quadrantGate(gs_pop_get_data(gs, "TCells"), stains=c("CD4", "CD8"),
+                       filterId="CD4CD8", sd=3)
> gs_pop_add(gs, qgate, parent = "TCells")

[1] 3 4 5 6

> recompute(gs)
```

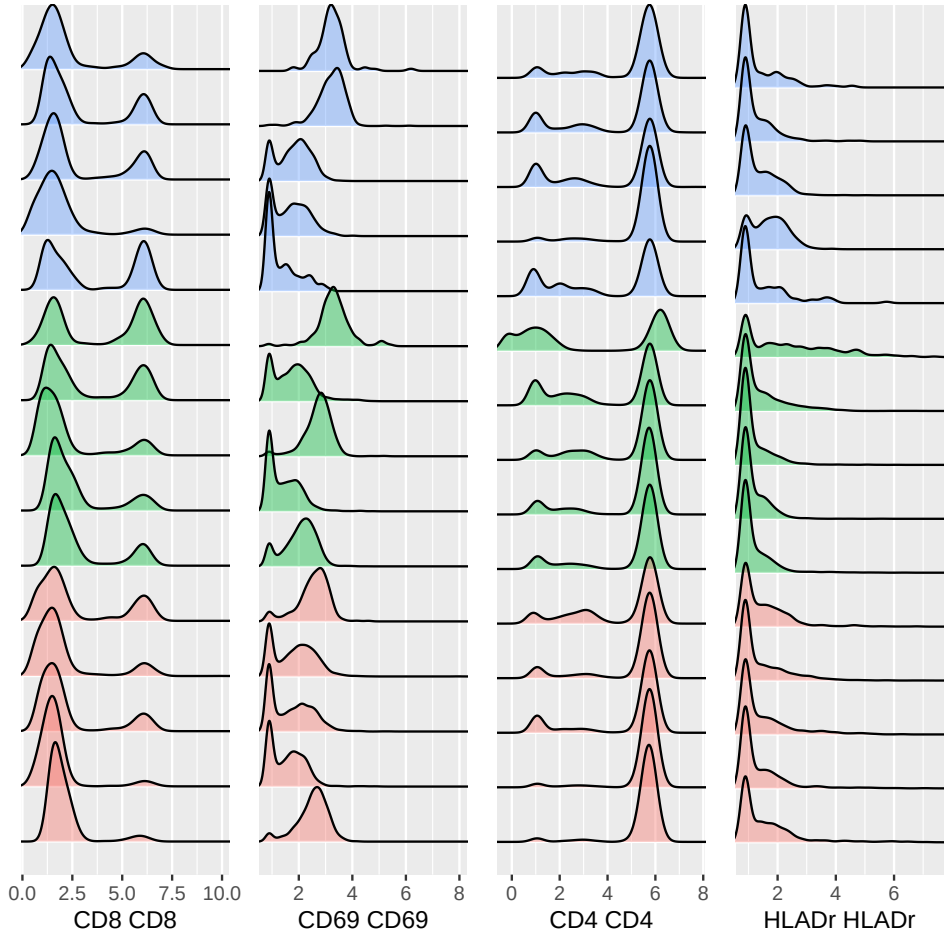
We can now normalize the channels relevant to this gate using `normalize`, so that we can successfully apply a single `quadGate` to all of the samples.

```
> gs <- normalize(gs, populations=c("CD4+CD8+", "CD4+CD8-", "CD4-CD8+", "CD4-CD8-"),
+                     dims=c("CD4", "CD8"), minCountThreshold = 50)

Estimating landmarks for channel CD4 ...
Estimating landmarks for channel CD8 ...
Registering curves for parameter CD4 ...
Registering curves for parameter CD8 ...
```

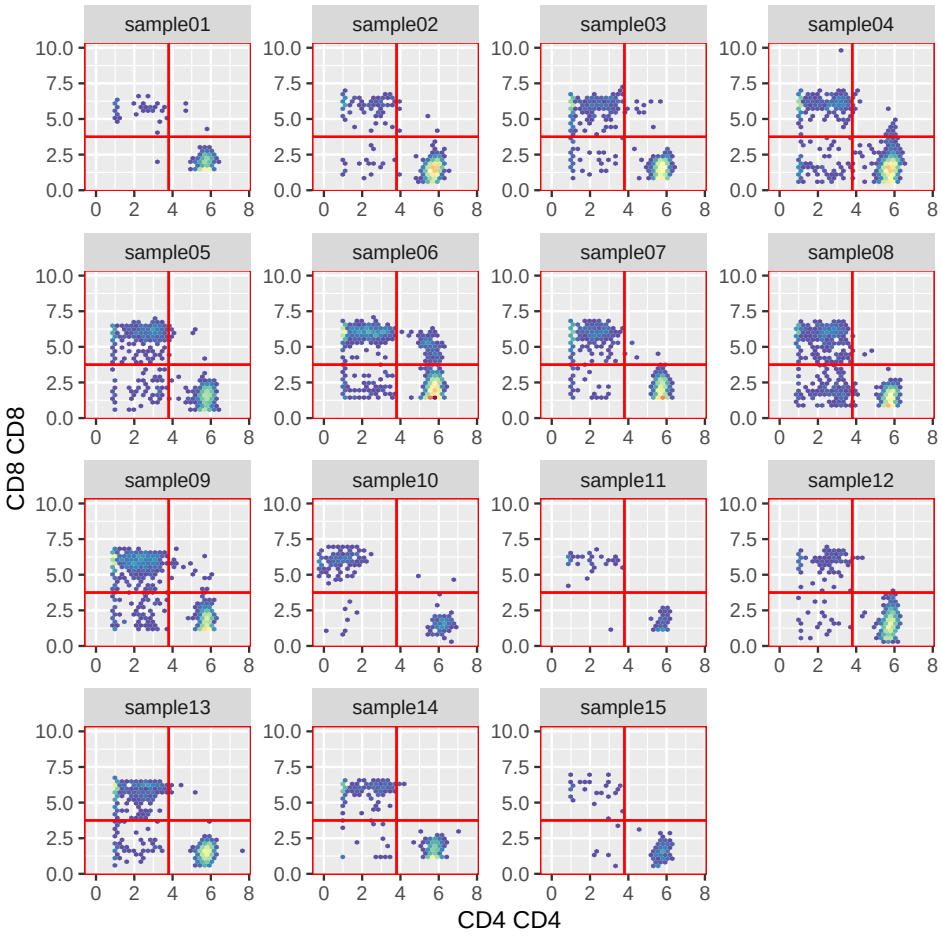
The normalization aligns the peaks in the selected channels across all samples.

```
> data <- gs_pop_get_data(gs, "TCells")
> plots <- lapply(pars, function(par)
+   as.ggplot(ggcyto(data, aes(x = !!par, fill = GroupID)) +
+     geom_density_ridges(mapping = aes(y = name), alpha = 0.4) +
+     theme(axis.title.y = element_blank(),
+           axis.text.y = element_blank(),
+           axis.ticks.y = element_blank()) +
+     facet_null()
+   ))
> do.call("grid.arrange", c(plots, nrow=1))
```



This ensures that the single `quadGate` is appropriately placed for each sample.

```
> ggcyto(gs_pop_get_data(gs, "TCells"), aes(x=CD4, y=CD8)) +
+   geom_hex(bins=32) +
+   geom_gate(gs_pop_get_gate(gs, "CD4-CD8-")) +
+   geom_gate(gs_pop_get_gate(gs, "CD4-CD8+")) +
+   geom_gate(gs_pop_get_gate(gs, "CD4+CD8-")) +
+   geom_gate(gs_pop_get_gate(gs, "CD4+CD8+"))
```



In a final step we might be interested in finding the proportion of activated T-helper cells by means of the CD69 stain. The `rangeGate` function is helpful in separating positive and negative peaks in 1D.

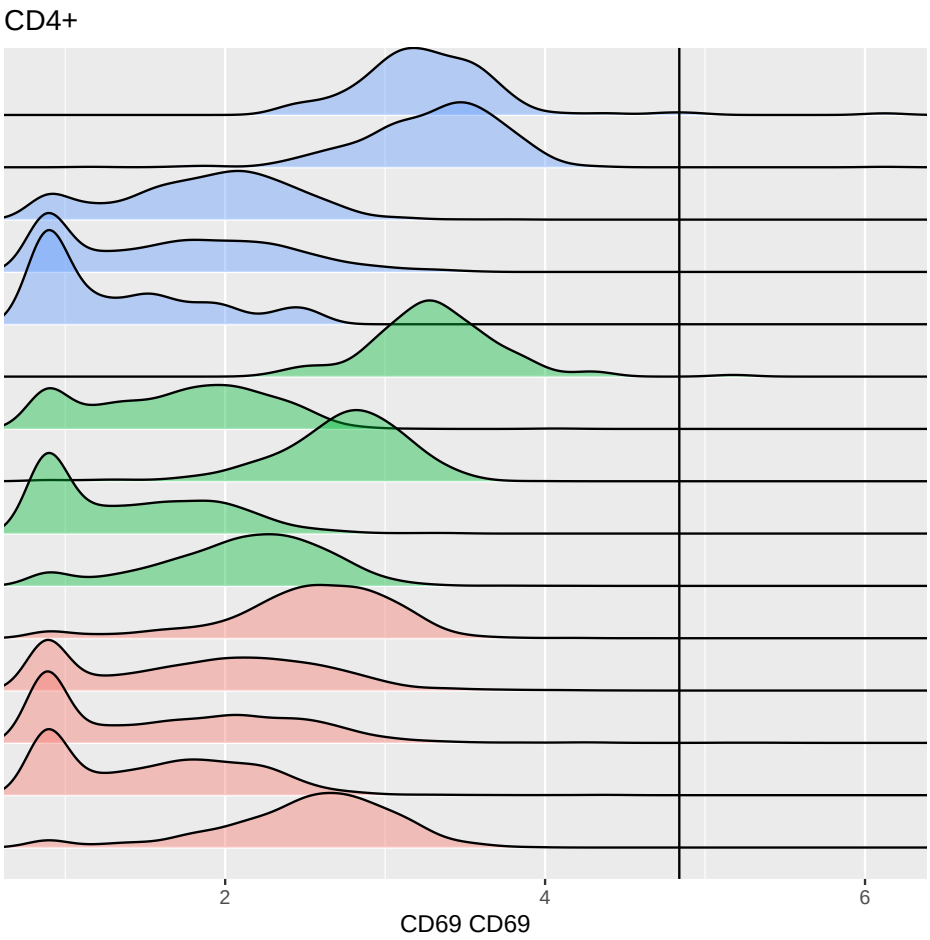
```
> CD69rg <- rangeGate(gs_cyto_data(gs), stain="CD69",
+                     alpha=0.75, filterId="CD4+CD8-CD69", sd=2.5)
> gs_pop_add(gs, CD69rg, parent="CD4+CD8-")

[1] 7

> # gs_pop_add(gs, CD69rg, parent="CD4+CD8-", name = "CD4+CD8-CD69-", negated=TRUE)
> recompute(gs)

> ggcyto(gs_pop_get_data(gs, "CD4+CD8-"), aes(x=CD69, fill = GroupID)) +
+     geom_density_ridges(mapping = aes(y = name), alpha = 0.4) +
+     theme(axis.title.y = element_blank(),
+           axis.text.y = element_blank(),
+           axis.ticks.y = element_blank()) +
+     geom_vline(xintercept = CD69rg@min) +
```

```
+     facet_null() +
+     ggtitle("CD4+")
```

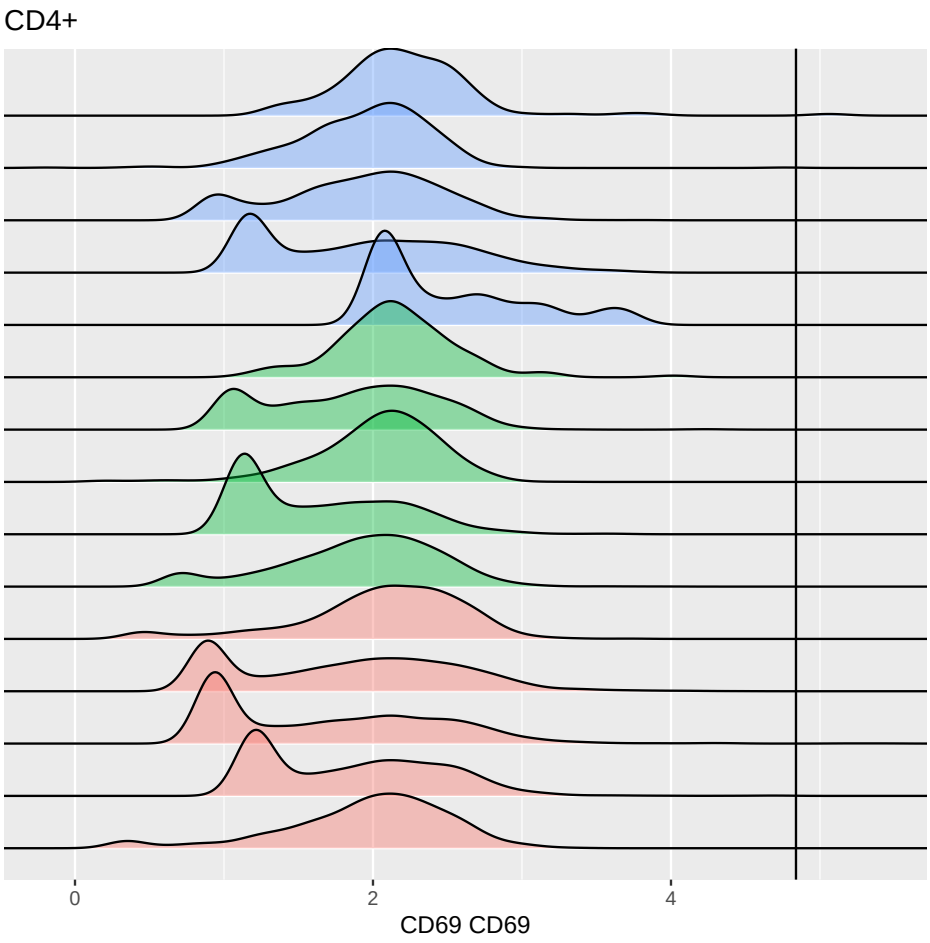


This channel could also benefit from normalization before determining a single `rangeGate` to apply to all samples.

```
> gs <- normalize(gs, populations=c("CD4+CD8-CD69"),
+                     dims=c("CD69"), minCountThreshold = 50)
```

```
Estimating landmarks for channel CD69 ...
Registering curves for parameter CD69 ...
```

```
> ggcyto(gs_pop_get_data(gs, "CD4+CD8-"), aes(x=CD69, fill = GroupID)) +
+     geom_density_ridges(mapping = aes(y = name), alpha = 0.4) +
+     theme(axis.title.y = element_blank(),
+           axis.text.y = element_blank(),
+           axis.ticks.y = element_blank()) +
+     geom_vline(xintercept = CD69rg@min) +
+     facet_null() +
+     ggtitle("CD4+")
```



2 Probability Binning

A probability binning algorithm for quantitating multivariate distribution differences was described by Roederer et al. The algorithm identifies the flow parameter in a flowFrame with the largest variance and divides the events in the flowFrame into two subgroups based on the median of the parameter. This process continues until the number of events in each subgroup is less than a user specified threshold.

For comparison across multiple samples, probability binning algorithm can be applied to a control dataset to obtain the position of bins and the same bins can be applied to the experimental dataset. The number of events in the control and sample bins can then be compared using the Pearsons chi-square test or the probability binning metric defined by Roederer et al.

Although probability binning can be applied simultaneously to all parameters in a flowFrame with bins in n dimensional hyperspace, we proceed with a two dimenstional example from our previous discussion involving CD4 and CD8 populations. This helps to simplify the demonstration of the method and interpretation of results.

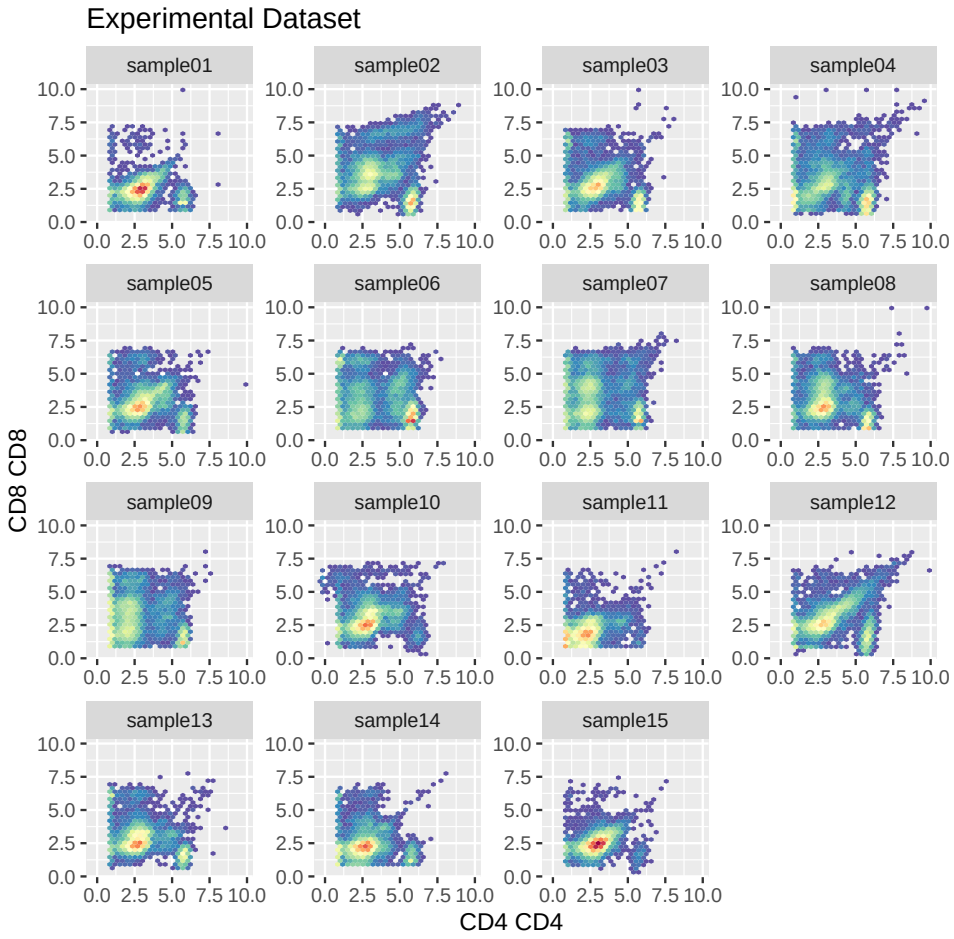
From the workflow object containing the warped data, we extract our data frame of interest.

We try to compare the panels using probability binning to identify patients with CD4, CD8 populations different from a control flowFrame that we create using the data from all the patients.

```
> dat <- gs_cyto_data(gs)
>
```

The dat is visualized below

```
> autoplot(dat, "CD4", "CD8") +
+   ggtitle("Experimental Dataset")
```



The control dataset is created by combining all the flowFrames in the flowSet. The flowFrame is then subsetted after applying a sampleFilter so that the control flowSet created has approximately the same number of events as the other flowSets in our example.

```
> datComb <- as(dat,"flowFrame")
> subCount <- nrow(exprs(datComb))/length(dat)
>         sf <- sampleFilter(filterId="mySampleFilter", size=subCount)
>         fres <- filter(datComb, sf)
```

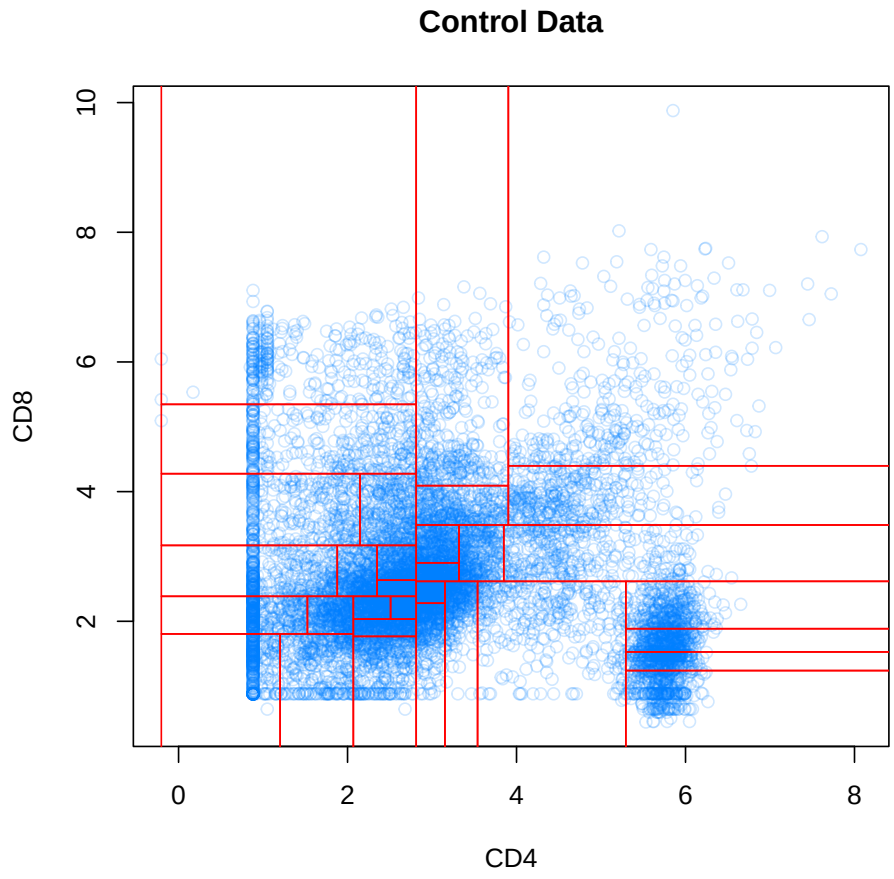
```
>          ctrlData <- Subset(datComb, fres)
>          ctrlData <- ctrlData[,-ncol(ctrlData)] ##remove the  column name "original"
>
```

The probability binning algorithm can then applied to the control data. The terminating condition for the algorithm is set so that the number of events in each bin is approximately 5 percent of the total number of events in the control data.

```
> minRow=subCount*0.05
> refBins<-proBin(ctrlData,minRow,channels=c("CD4","CD8"))
>
```

The binned control Data can be visualized using the `plotBins` function. Areas in the scatter plot with a large number of data points have a higher density of bins. Each bin also has approximately same number of events.

```
> plotBins(refBins,ctrlData,channels=c("CD4","CD8"),title="Control Data")
>
```



The same bin positions from the control data set are then applied to each flowFrame in our sample Data set.

```
> sampBins <- fsApply(dat,function(x){
+                     binByRef(refBins,x)
+                     })
```

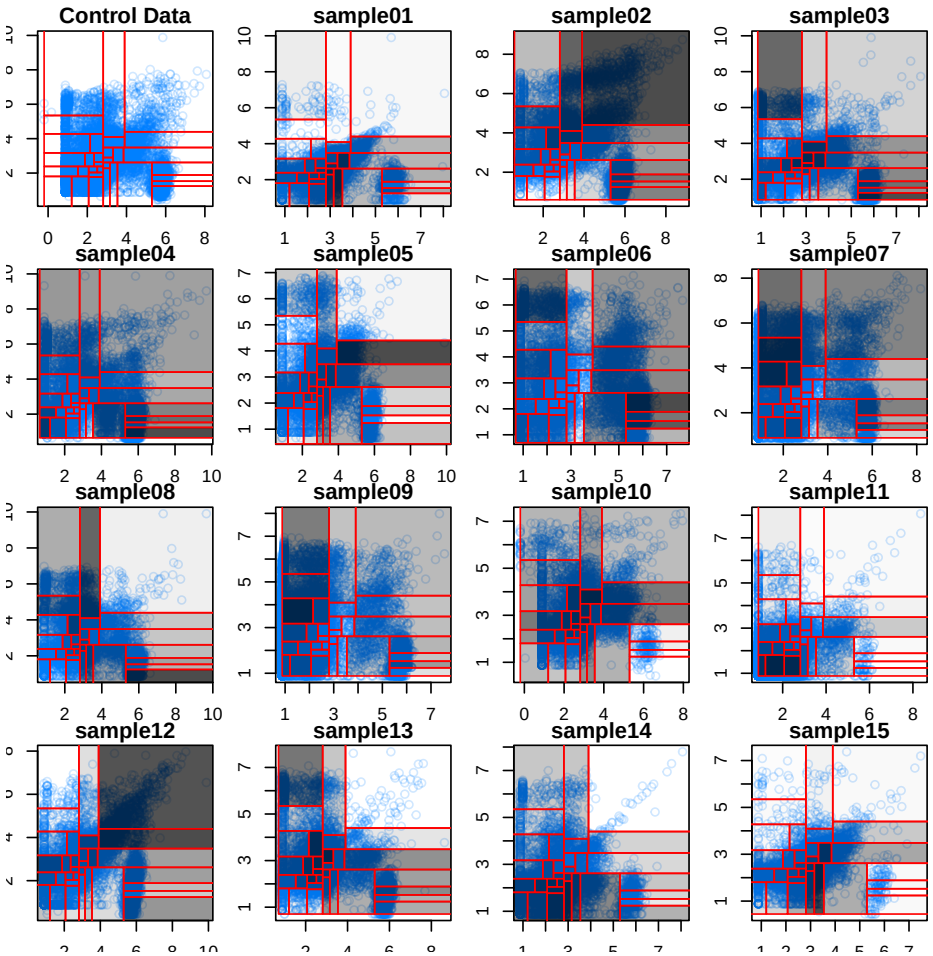
For each patient, the number events in the control and sample bins can be compared using the `calcPearsonChi` or using Roederers probability binning metric.

```
> pearsonStat <- lapply(sampBins,function(x){
+                       calcPearsonChi(refBins,x)
+                      })

> sCount <- fsApply(dat,nrow)
> pBStat <-lapply(seq_along(sampBins),function(x){
+                 calcPBChiSquare(refBins,sampBins[[x]],subCount,sCount[x])
+                 })
```

For each sample, the results can be visualized using the `plotBins` function. The residuals from Roeders probability binning metric or the Pearsons chi square test can be used to shade bins to highlight bins in each sample that differ the most from the control sample.

```
> par(mfrow=c(4,4),mar=c(1.5,1.5,1.5,1.5))
> plotBins(refBins,ctrlData,channels=c("CD4","CD8"),title="Control Data")
> patNames <-sampleNames(dat)
> tm<-lapply(seq_len(length(dat)),function(x){
+                 plotBins(refBins,dat[[x]],channels=c("CD4","CD8"),
+                         title=patNames[x],
+                         residuals=pearsonStat[[x]]$residuals[2,],
+                         shadeFactor=0.7)
+
+                 }
+       )
```



The patient with CD4/CD8 populations most different from that of the control group can be identified from the magnitue of Pearson-chi square statistic(or Probability binning statistic).

	chi_Square_Statistic	pBin_Statistic
sample01	2536.70	318.22
sample02	2930.08	368.18
sample03	704.18	85.49
sample04	1749.80	218.29
sample05	696.63	84.53
sample06	3198.08	402.22
sample07	2093.41	261.93
sample08	832.11	101.74
sample09	3022.76	379.95
sample10	1474.75	183.36
sample11	5608.97	708.40
sample12	1204.51	149.04
sample13	1041.89	128.38
sample14	1782.30	222.42
sample15	3813.09	480.33

- R version 4.1.1 Patched (2021-08-22 r80813), `x86_64-apple-darwin17.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Running under: `macOS Mojave 10.14.6`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: BH 1.75.0-0, RcppArmadillo 0.10.7.0.0, flowCore 2.6.0, flowStats 4.6.0, flowWorkspace 4.6.0, ggcyto 1.22.0, ggplot2 3.3.5, ggridges 0.5.3, gridExtra 2.3, ncdfFlow 2.40.0, scales 1.1.1, xtable 1.8-4
- Loaded via a namespace (and not attached): Biobase 2.54.0, BiocGenerics 0.40.0, DBI 1.1.1, DEoptimR 1.0-9, IDPmisc 1.1.20, KernSmooth 2.23-20, MASS 7.3-54, Matrix 1.3-4, R6 2.5.1, RColorBrewer 1.1-2, RCurl 1.98-1.5, RProtoBufLib 2.6.0, Rcpp 1.0.7, RcppParallel 5.1.4, Rgraphviz 2.38.0, S4Vectors 0.32.0, XML 3.99-0.8, assertthat 0.2.1, aws.s3 0.3.21, aws.signature 0.6.0, base64enc 0.1-3, bitops 1.0-7, cluster 2.1.2, colorspace 2.0-2, compiler 4.1.1, crayon 1.4.1, curl 4.3.2, cytolib 2.6.0, data.table 1.14.2, deSolve 1.30, digest 0.6.28, dplyr 1.0.7, ellipsis 0.3.2, fansi 0.5.0, farver 2.1.0, fda 5.4.0, fds 1.8, flowViz 1.58.0, generics 0.1.1, glue 1.4.2, graph 1.72.0, grid 4.1.1, gtable 0.3.0, hdrcde 3.4, hexbin 1.28.2, httr 1.4.2, jpeg 0.1-9, ks 1.13.2, labeling 0.4.2, lattice 0.20-45, latticeExtra 0.6-29, lifecycle 1.0.1, magrittr 2.0.1, matrixStats 0.61.0, mclust 5.4.7, munsell 0.5.0, mvtnorm 1.1-3, pcaPP 1.9-74, pillar 1.6.4, pkgconfig 2.0.3, plyr 1.8.6, png 0.1-7, pracma 2.3.3, purrr 0.3.4, rainbow 3.6, rlang 0.4.12, robustbase 0.93-9, rrcov 1.6-0, splines 4.1.1, stats4 4.1.1, tibble 3.1.5, tidyselect 1.1.1, tools 4.1.1, utf8 1.2.2, vctrs 0.3.8, withr 2.4.2, xml2 1.3.2, zlibbioc 1.40.0