Computational Analysis of Flow Cytometry Data using R / Bioconductor BioC 2011

Greg Finak[1]

[1] Fred Hutchinson Cancer Research Center

July 29, 2011

Introduction

The purpose of this workshop is to demonstrate advances in the development of R packages that support the gating and analysis of complex flow cytometry datasets.

Current Software Development Status

1. flowCore and related packages are stable, robust, and highly functional.
2. There is a new package flowWorkspace for communicating between R and FlowJo.

These capabilities now make it feasible to analyze complex flow datasets and directly compare the results to results obtained from FlowJo.

In this workshop, we'll demonstrate the use of these new tools.
The data and code are available at the Bioconductor website.

The Importance of the R/FlowJo Dialog

R and FlowJo provide two different, equally important roles data analysis.

Benefits of Collaboration

1. FlowJo enables the cytometrist to develop an analysis that captures biological meaning.
2. R enables the use of modern machine learning methods and objective, numerical approaches.
3. The informatician needs to understand what the cytometrist did to ensure the correct methods are used.
4. The cytometrist needs to review and validate the results of the R analysis.

The new possibilities for collaboration will increase the effectiveness of both approaches.

A 96-Well Plate Flow Assay

We will do a comparative analysis of a flow cytometry assay done on a 96-well plate.

Two Ways to Use flowCore

1. Import a flowJo workspace with gates, transforms, and compensation matrices
2. Analyze from scratch within BioConductor

The aim of the assay is to identify wells with cells that are positive for pairwise combinations of three markers.

We'll walk through each analysis then compare the results.

Setting up the R session

First we set up the environment by telling R where to find FCS files and where to execute code.

Setup

```
> #Set the path to the FCS files in the package
+ fcspath<-system.file("extdata",package="BioC2011FlowWorkshop")
> #load libraries
> if(require(Rmpi))
+   require(multicore)
> require(flowWorkspace)
> require(flowStats)
> require(flowViz)
> ncdf4<-require(ncdf4)
```

Requirements

Software:

- Development version of R (2.14) and Bioconductor (2.9)
- **flowWorkspace (0.99.14) and flowWorkspaceData (0.99.4)**

Download urls

- `http://bioconductor.org/packages/2.9/data/experiment/html/flowWorkspaceData.html`
- `http://bioconductor.org/packages/2.9/html/flowWorkspace.html`

Installation from Bioconductor

Ensure Packages are up to date

```
> update.packages();
```

Install flowWorkspace and flowWorkspaceData from Bioconductor repository

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("flowWorkspace","flowWorkspaceData",
  dependencies=TRUE);
```

Opening and Importing flowJo Workspaces

Parsing a flowJo Workspace

```
> xmlfile <- system.file("extdata", package = "BasicFlowWorkshop",
+     mustWork = TRUE)
> xmlfile <- list.files(pattern = "BDWorkspace.xml",
+     xmlfile, full = T)
> ws <- openWorkspace(xmlfile)
> G <- try(parseWorkspace(ws, execute = TRUE, isNcdf = FALSE,
+     path = fcspath, name = "samples"))
```

`openWorkspace()` will open an XML flowJo workspace file. You can get more information about it with:

```
> show(ws)
```

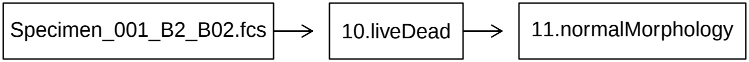
`parseWorkspace()` will parse the flowJo workspace file Additionally we tell it to execute the compensation,transformation, and gating, don't use netCDF, where are the FCS files, and the name of the group of samples to import.

GatingSet and GatingHierarchy

```
> G[1:2]

A GatingSet with  2  samples
1 .  FCS File:  Specimen_001_B2_B02.fcs
GatingHierarchy with  3  gates
2 .  FCS File:  Specimen_001_B1_B01.fcs
GatingHierarchy with  3  gates

> plot(G[[1]])
```



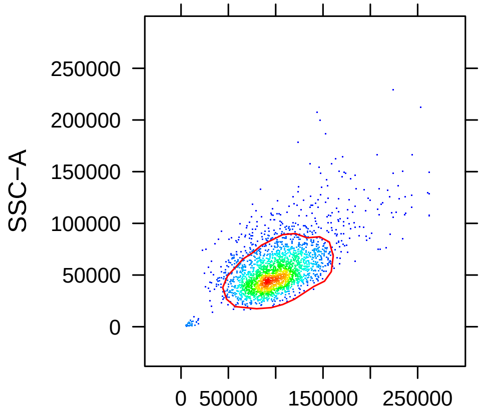
G is a `GatingSet`, which is a list of `GatingHierarchy` objects. Each describes a sample in the flowJo workspace and how it is compensated, transformed,and gated.

Exploring the Data

We can plot individual gates for samples (`GatingHierarchy`) in a `GatingSet`.

plotGate

```
> print(plotGate(G[[1]], getNodes(G[[1]])[3]))
```



Extracting Population Statistics

getPopStats()

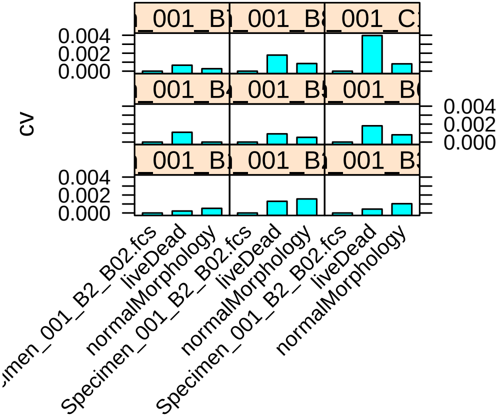
```
> require(xtable)
> getPopStats(G[[5]])
> stats <- do.call(cbind, lapply(G, function(x) getPopStats(x)[,
+     1]))
> rownames(stats) <- rownames(getPopStats(G[[1]]))
> colnames(stats) <- getSamples(G)
> print(xtable(t(stats)[1:10, ]), size = "tiny")
```

	Specimen_001_B2_B02.fcs	/liveDead	/liveDead/normalMorphology
Specimen_001_B2_B02.fcs	1.00	0.95	0.84
Specimen_001_B1_B01.fcs	1.00	0.95	0.84
Specimen_001_B3_B03.fcs	1.00	0.95	0.84
Specimen_001_B4_B04.fcs	1.00	0.94	0.83
Specimen_001_B5_B05.fcs	1.00	0.91	0.87
Specimen_001_B6_B06.fcs	1.00	0.89	0.84
Specimen_001_B7_B07.fcs	1.00	0.91	0.82
Specimen_001_B8_B08.fcs	1.00	0.89	0.80
Specimen_001_C1_C01.fcs	1.00	0.95	0.77
Specimen_001_C2_C02.fcs	1.00	0.94	0.78

Some Quality Control after Importing a Workspace

plotPopCV

```
> #Plot the coefficient of variation for the first
> #9 samples (to save space)
+ print(plotPopCV(G[1:9],rot=45));
```



Some Housekeeping

Set Metadata

```
> FITC.WELL = "B02"
> PE.WELL = "B03"
> APC.WELL = "B04"
> pData(G) <- data.frame(WELL = unlist(lapply(G,
+     function(x) keyword(x, "WELL ID")[[1]])))
```

- Specify the FITC, PE, and APC isotype control wells.
- Ensure metadata for the `GatingSet` is up to date.
- Extract the ‘`WELL ID`‘ keyword from each sample.

Calculate Cutoffs from Isotype Controls

We've imported gates, transformations and compensation matrices. We can just extract the data from the GatingHierarchy at position 3.

Extracting Gated Data

```
> mydata <- getData(G[[which(pData(G) == FITC.WELL)]],
+     3)
> fitc.cutoff <- quantile(exprs(mydata[, "<FITC-A>"]),
+     0.99)
> mydata <- getData(G[[which(pData(G) == PE.WELL)]],
+     3)
> pe.cutoff <- quantile(exprs(mydata[, "<PE-A>"]),
+     0.99)
> mydata <- getData(G[[which(pData(G) == APC.WELL)]],
+     3)
> apc.cutoff <- quantile(exprs(mydata[, "<APC-A>"]),
+     0.99)
```

Then compute the 99% cutoff for each isotype control channel/well.

Get Non–Control Wells

```
> d <- getData(G[which(pData(G) != FITC.WELL & pData(G) !=
+     PE.WELL & pData(G) != APC.WELL)], 3)[, c("<FITC-A>",
+     "<PE-A>", "<APC-A>")]
```

Adjust Metadata

```
> phenodata <- pData(G[which(pData(G) != FITC.WELL &
+     pData(G) != PE.WELL & pData(G) != APC.WELL)])
> sampleNames(d) <- rownames(phenodata)
> pData(d) <- data.frame(phenodata, Name = rownames(pData(d)))
```

Compute Positive Cell Counts

```
> tables <- unlist(fsApply(d, function(x) {
+     df <- data.frame(t(t(exprs(x)) > c(fitc.cutoff,
+         pe.cutoff, apc.cutoff)))
+     for (i in 1:ncol(df)) {
+         df[, i] <- factor(df[, i], levels = c("FALSE",
+             "TRUE"))
+     }
+     colnames(df) <- c("FITC", "PE", "APC")
+     tbl <- table(df)
+     list(tbl)
+ }), recursive = F)
```

Cross-Tabulation

```
> print(tables[[10]])

, , APC = FALSE
       PE
FITC    FALSE TRUE
  FALSE   236  244
  TRUE     27   30

, , APC = TRUE

       PE
FITC    FALSE TRUE
  FALSE   252  835
  TRUE     36  151
```

Calculate Proportions from Counts

Organize the Results

```
> #Calculate % positive cells for each combination of FITC/PE/APC
+ res<-do.call(rbind,
+  (lapply(tables,
+  function(x)as.vector(x/sum(x))
+  ))
> #Define groups by well and by FITC/PE/APC combo
> res<-data.frame(proportion=as.vector(res),combo=gl(8,81),well=
+  factor(as.numeric(as.vector(t(matrix(gl(81,8),nrow=8))))))
> #Relabel the combos to +++ through ---
> res[,2]<-factor(c("---","+--","-+-","++-","--+","+-+",
+  "-++","+++"))[res[,2]]
```

Calculate proportions as fraction of the total cells (after gating).
Add variables identifying:

- common measurements from each well
- common combinations across wells.

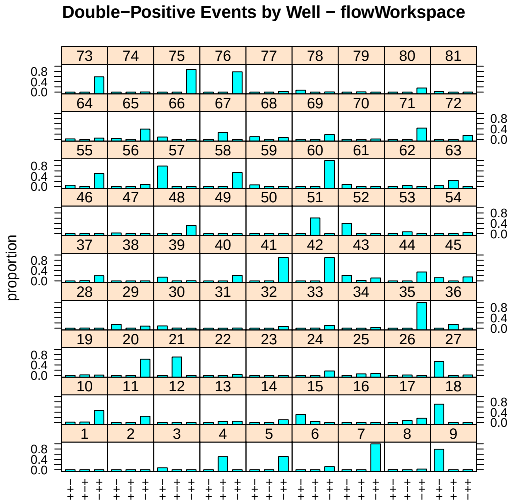
Tabulated data

```
> print(res[sample(1:648, 15), ])

     proportion combo well
364 0.3281165677   --+   40
114 0.0005583473   +--   33
542 0.0817166373   -++   56
299 0.0000000000   ++-   56
591 0.0235798499   +++   24
579 0.0000000000   +++   12
98  0.0076880835   +--   17
617 0.9828729282   +++   50
586 0.0060273973   +++   19
382 0.0647284696   --+   58
406 0.0000000000   +-+    1
135 0.0005420054   +--   54
393 0.5214592275   --+   69
148 0.1708378672   +--   67
548 0.0083989501   -++   62
```

Plotting Results

```
> print(barchart(proportion ~ combo | well, data = res,
+     scales = list(x = list(rot = 90)), subset = as.numeric(res[,
+         2]) %in% c(2, 3, 5), main = "Double-Positive Events by Well - flowWor
```



That Was Easy..

Because we imported the workspace from flowJo we had little to do in the way of gating or other data pre-processing.

Now let's repeat the analysis but do the compensation and gating in `flowCore`.

Reading in a flowSet

read.flowSet

```
> dat <- read.flowSet(files = list.files(fcspath,
+     pattern = "fcs", full = T))
```

`read.flowSet` will load in a collection of FCS files that share a **common** set of parameters.

- similar to a `GatingSet`
- no gating, compensation, or transformation information

Housekeeping

Update Metadata, etc..

```
> pData(dat) <- data.frame(pData(dat)[, 1], well = keyword(dat,
+     "WELL ID"))
> varMetadata(phenoData(dat))$labelDescription <- c("Name",
+     "Well")
> compWells = c("A08", "A09", "A10", "A11", "A12")
> ldWell = "A09"
> ldChan = "APC-Cy7-A"
> unstainedWell = 1
> namePattern = "FITC-A|PE-A|APC-A|APC-Cy7-A|FSC-A|SSC-A"
```

- Same as before, update the `phenoData` with `WELL ID` information.
- Identify the compensation control wells

Calculate the Spillover Matrix

Spillover

```
> comps <- dat[which(pData(dat)$WELL.ID %in% compWells)]
> spill <- spillover(x = comps, unstained = unstainedWell,
+     patt = namePattern, fsc = "FSC-A", ssc = "SSC-A",
+     useNormFilt = TRUE)
```

flowCore has a handy method, `spillover`, that can automatically compute the spillover matrix for you.

- comps is a flowSet of compensation wells
- unstained identifies which well is the unstained control
- patt is a regular expression to select the desired columns (defined on previous slide)
- fsc and ssc - obvious
- useNormFilt - use autogating to select a homogeneous population

Compensation

using workFlow()

```
> dat <- dat[which(!pData(dat)$WELL.ID %in% compWells)]
> wf <- workFlow(dat)
> add(wf, compensation(spill, compensationId = "Plate Defined"))
```

Workflows help organize an analysis in a coherent way.

- `workFlow()` will create a workflow object and initialize it with some raw data.
- `add()` will add actions to a workflow.
    - compensation, transformation, gating.

Note

`workFlows` apply the same action to all samples in a flowSet `GatingSet` and `GatingHierarchy` allow different actions for each sample.

Transformation

Logicle / Biexponential Transformation

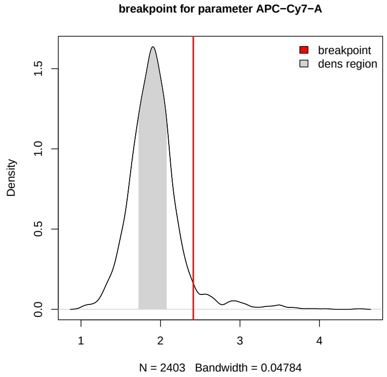
```
> tr <- estimateLogicle(flowFrame(fsApply(dat, function(x) exprs(x))),
+     colnames(dat)[4:7])
> identifier(tr) <- "commonLogicle"
> add(wf, tr, parent = "Plate Defined")
```

- combine all wells
- estimate the parameters of the logicle from the data
- add to the workflow (apply the transformation)
- `parent` specifies what data to apply the action.

Gating Live / Dead Cells

rangeGate() Example

```
> rg <- rangeGate(Data(wf[["commonLogicle"]])[[1]],
+     stain = ldChan, filterId = "liveDeadGate",
+     refLine = 1, plot = T)
```



Autogating

- `rangeGate` identifies the split between populations
- Gating live (-ive staining) vs dead (+ive stating) cells

Gating Live / Dead Cells

Gate the workFlow

```
> rg <- rangeGate(Data(wf[["commonLogicle"]]), stain = ldChan,
+     filterId = "liveDeadGate", refLine = 1)
> add(wf, rg, parent = "commonLogicle")
```

- Define a rangeGate on the flowSet
- Add it to the workflow (`parent="commonLogicle"`)

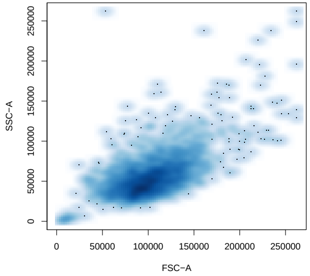
Filter Boundary Events

boundaryFilter

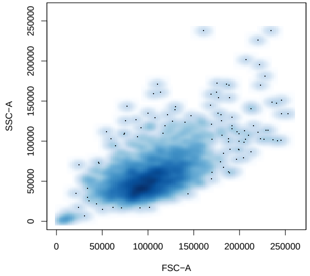
```
> bnd <- boundaryFilter(c("FSC-A", "SSC-A"), filterId = "boundary",
+     tolerance = c(10, 10))
> add(wf, bnd, parent = "liveDeadGate-")
```

Filter boundary (saturated or off–scale) events from the FSC and SSC channels.

With boundary events.



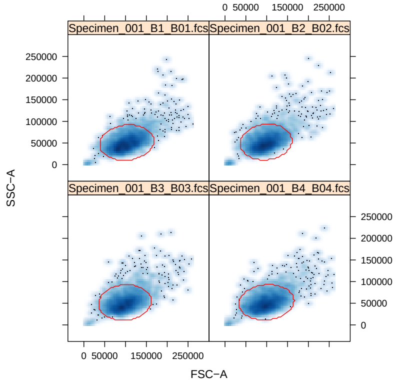
Without boundary events



Normal Morphology Gate

curv2Filter() example

```
> print(xyplot(`SSC-A` ~ `FSC-A`, Data(wf[["boundary+"]])[1:4],
+     filter = filter(Data(wf[["boundary+"]])[1:4],
+         curv2Filter("FSC-A", "SSC-A", filterId = "normalMorphology",
+             bwFac = 7, gridsize = c(151, 151)))))
```



Normal Morphology Gate

Gate Remaining Samples

```
> nmorph <- curv2Filter("FSC-A", "SSC-A", bwFac = 7,
+     gridsize = c(151, 151), filterId = "normalMorphology")
> add(wf, nmorph, parent = "boundary+")
```

Extract Isotype Cutoff Values

```
> fitc.cutoff.wf<-quantile(exprs(Data(wf[["area 1"]])[which(
+  pData(Data(wf[["area 1"]]))$WELL.ID%in%FITC.WELL)][[1]][,"FITC-A"]),
+  0.99)
> pe.cutoff.wf<-quantile(exprs(Data(wf[["area 1"]])[which(
+  pData(Data(wf[["area 1"]]))$WELL.ID%in%PE.WELL)][[1]][,"PE-A"]),
+  0.99)
> apc.cutoff.wf<-quantile(exprs(Data(wf[["area 1"]])[which(
+  pData(Data(wf[["area 1"]]))$WELL.ID%in%APC.WELL)][[1]][,"APC-A"]),
+  0.99)
```

Housekeeping

Get the non-control wells

```
> dat <- Data(wf[["area 1"]])[!pData(Data(wf[["area 1"]]))$WELL.ID %in%
+     c(FITC.WELL, APC.WELL, PE.WELL)][,
+     c("FITC-A", "PE-A", "APC-A")]
```

Reorder samples

```
> dat <- dat[match(as.character(pData(d)[,
+     1]), as.character(pData(dat)[,
+     2]))]
```

For later comparison we reorder the samples so that they are in the same order as in the flowJoWorkspace.

Finally..

Organize the results

```
> #for each sample count the events that are
> #above the isotype control cutoffs
+ tables.wf<-fsApply(dat,function(x){r<-t(t(exprs(x))>
+  c(fitc.cutoff.wf,pe.cutoff.wf,apc.cutoff.wf));
+  #put them in a data frame
+  r<-data.frame(r);
+  #Label them nicely
+  for(i in 1:ncol(r)){
+  r[,i]<-factor(r[,i],levels=c("FALSE","TRUE"))
+  };
+  #tabulate counts
+  colnames(r)<-c("FITC-A","PE-A","APC-A");list(table(r))
+  })
> tables.wf<-unlist(tables.wf,recursive=F)
> #compute proportions from counts
> res.wf<-do.call(rbind,lapply(tables.wf,function(x)as.vector(
+  x/sum(x))))
> #make a nice final data frame
> res.wf<-data.frame(proportion=as.vector(res.wf),combo=gl(8,81),
+ well=factor(as.numeric(as.vector(t(matrix(gl(81,8),nrow=8))))))
> res.wf[,2]<-factor(c("---","+--","-+-","++-",
+ "--+","+-+","-++","+++"))[res.wf[,2]]
```

Data frame of output

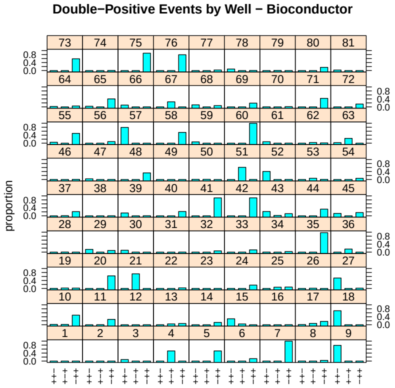
```
> print(res.wf[sample(1:dim(res.wf)[1],
+     15), ])
```

```
     proportion combo well
413 0.0000000000   +-+    8
292 0.0004909180   ++-   49
421 0.0004780115   +-+   16
481 0.0000000000   +-+   76
18  0.0163540164   ---   18
501 0.0000000000   -++   15
531 0.1622659626   -++   45
99  0.0110630111   +--   18
213 0.0000000000   -+-   51
448 0.0186732187   +-+   43
602 0.0235237638   +++   35
402 0.0000000000   --+   78
277 0.0024666996   ++-   34
618 0.3675213675   +++   51
135 0.0004880429   +--   54
```

Plot the results

barchart..

```
> print(barchart(proportion ~ combo |
+     well, data = res.wf, scales = list(x = list(rot = 90)),
+     subset = as.numeric(res.wf[, 2]) %in%
+         c(2, 3, 5), main = "Double-Positive Events by Well - Bioconductor"))
```

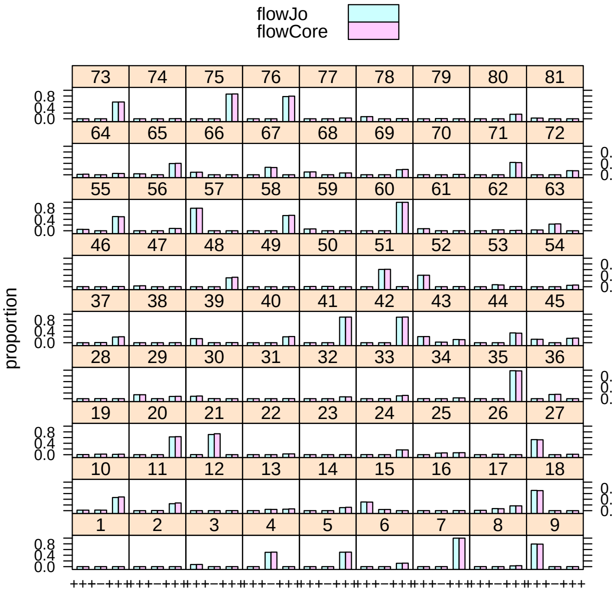


The two analyses look pretty similar! We should take a closer look..

Comparison Across Analyses

```
> res3 <- data.frame(rbind(res, res.wf),
+     analysis = gl(2, dim(res)[1],
+         labels = c("flowJo", "flowCore")))
> print(barchart(proportion ~ combo |
+     well, ylim = c(-0.1, 1.1), data = res3,
+     subset = which(as.numeric(res3[,
+         2]) %in% c(2, 3, 5)), group = analysis,
+     main = "Positive Wells Across Analyses",
+     auto.key = TRUE))
```

Positive Wells Across Analyses

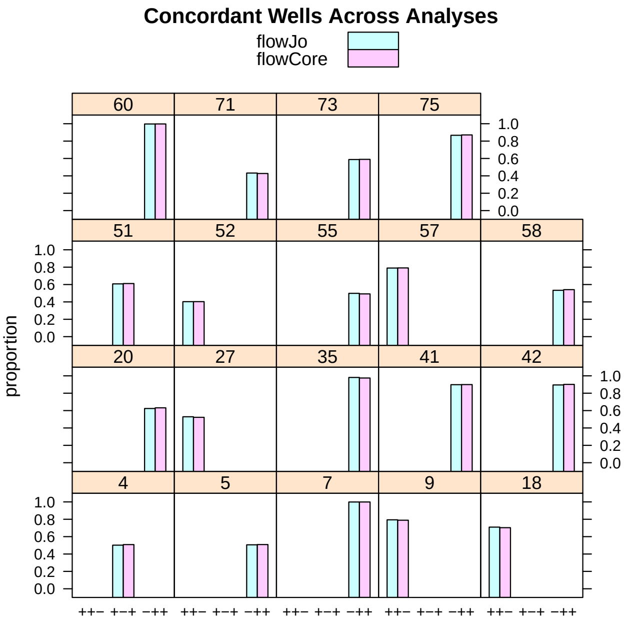


Comparison Across Analyses

Concordant Wells

```
> cv <- 0.01
> high <- 0.25
> print(barchart(proportion ~ combo |
+     well, ylim = c(-0.1, 1.1), data = res3,
+     subset = which(apply(cbind(res[,
+         1], res.wf[, 1]), 1, function(x) sd(x)/mean(x)) <
+         cv & as.numeric(res3[, 2]) %in%
+         c(2, 3, 5) & (res[, 1] > high |
+         res.wf[, 1] > high)), group = analysis,
+     main = "Concordant Wells Across Analyses",
+     auto.key = TRUE))
```

Pick out wells with C.V. $< 1\%$ and proportion $> 25\%$



Concordant Wells Across Analyses
flowJo
flowCore
proportion
60
71
73
75
51
52
55
57
58
20
27
35
41
42
4
5
7
9
18
1.0
0.8
0.6
0.4
0.2
0.0
++−
+−+
−++

In Conclusion

You should now have a basis (examples and working code) for analyzing your own data.
-Import preprocessing and gates from flowJo
-Or (for the brave) entirely within BioConductor
Other features of flowWorkspace not covered here:
-NetCDF support (requires ncdf4 package and netcdf4 C library)
Just pass `isNCDF=TRUE` to `parseWorkspace()`
-Parallel processing using `Rmpi` and `multicore` packages.
Pass `nslaves=#` to `parseWorkspace()`
More complete netCDF support coming (ncdfFlow package).
flowWorkspace constantly being improved.

Acknowledgements

rglab.org

- Raphael Gottardo
- Mike Jiang
- Mose Andre

HVTN

- Steve DeRosa

Bioconductor

- Nishant Gopalakrishnan
- Chao-Jen Wong

Funding NIH R01 EB008400 and the ITN