An Introduction to SingleCellAssay

Andrew McDavid and Greg Finak

February 12, 2013

1 Philosophy

`SingleCellAssay`is an R/Bioconductor package for managing and analyzing Fluidigm single–cell gene expression data as well as data from other types of single–cell assays. Our goal is to support assays that have multiple *features* (genes, markers, etc) per *well* (cell, etc) in a flexible manner. Assays are assumed to be mostly *complete* in the sense that most *wells* contain measurements for all features.

2 Implementation Details

Here we provide some background on the implementation of the package.

There are several fundamental new object types provided by the package. The `SCA` is a virtual class that is inherited by `SingleCellAssay` and `FluidigmAssay`. The latter also inherits from `SingleCellAssay`. New types of single cell assays can be incorportated by extending `SingleCellAssay` or `SCA`, as appropriate (depending on the structure of the data).

The correspondence between data files and the internal representation in the package for `features`, `cells` or `wells`, and `phenotypic data` is provided by the `Mapping` class which simply provides a mapping of internal keys (i.e. `primerid, idvars, featurevars, cellvars, phenovars`) and values (one or more column names in the assay data files). These key–value maps allow the constructors to correctly map feature–level, cell–level, and sample–level information to each measured cell. The

The `Mapping` class makes this very flexible and amenable to many non–standard tabular data representations, and indepdendent of whatever names a user may have given different variables in the data set.

Some mappings are required to construct a valid `SingleCellAssay` or `FluidigmAssay` object. The required mappings and default maps are present for each single–cell assay class in the package (i.e. `SingleCellAssay:::FluidigmMap, SingleCellAssay:::FluidigmMapNames, SingleCellAssay:::SingleCellAssayMap`, and `SingleCellAssay:::SingleCellAssayMapNames`).

Sets of single cell assays are stored in the `SCASet` class. A constructor for SCASet is provided to construct an SCASet directly from a data frame. Alternatively, a SingleCellAssay or derived class can be `split` on an arbitray variable to produce an SCASet.

2.1 Extending SingleCellAssay Classes

Any extension of the package to new types of single–cell assays should implement a new `class` for the assay, provide a `constructor` with the same name as the class, and provide a set of default required `map names` and an empty `Mapping`, which would then be filled in by the constructor.

The function `SingleCellAssayValidity` function can be used as a validity checking method for classes inheriting from `SingleCellAssay`. It will check that the required map names are provided, and that their values are present in the data.

On construction of a `SingleCellAssay` object, the package tests for completeness, and will fill in the missing data (with NA) if it is not, so assays with lots of missing data can make reading marginally slower.

2.2 Internals

Internally (within a `SingleCellAssay`, we store everything as a `data.frame` with names of special columns kept in the `@mapping` slot that contains a `Mapping` object. The data is stored in long-melted format, in feature-major order, so while not especially fast or space-efficient, it is rather intended to be very flexible.

Each well, feature TODO: , and unit (phenotype) has various measured covariates. These are kept in `AnnotatedDataframes` in slots in the object, which are generated from the base `data.frame`, if provided. TODO: If not provided, then they can be added after object creation.

2.3 Statistical Testing

Apart from reading and storing single–cell assay data, the package also provides functionality for significance testing of differential expression using a combined binomial and normal–theory likelihood ratio test, as well as filtering of individual outlier wells. These methods are described in the paper.

3 Examples

With the cursory background out of the way, we'll proceed with some examples to help understand how the package is used.

3.1 Reading Data

Data can be imported in a Fluidigm instrument-specific format (the details of which are undocumented, and likely subject-to-change) or some derived, annotated format, or in "long" (melted) format, in which each row is a measurement, so if there are N wells and M cells, then the `data.frame` should contain $N \times M$ rows. The use of key–value mappings makes the reading of various input formats very flexible, provided that they contain the minimal required information expected by the package.

For example, the following data set was provided in as a comma-separated value file. It has the cycle threshold (ct) recorded. Non-detected genes are recorded as NAs. For the Fluidigm/qPCR single cell expression functions to work as expected, we must use the *expression threshold*, defined as $E_t = c_{\max} - ct$, which is proportional to the log-expression.

Below, we load the package and the data, then compute the expression threshold from the ct, and construct a `FluidigmAssay`.

```
library(SingleCellAssay)

##
## Attaching package:  'SingleCellAssay'
## The following object is masked from 'package:stats':
##
##     filter

require(plyr)

## Loading required package:  plyr

data(vbeta)
colnames(vbeta)

##  [1] "Sample.ID"         "Subject.ID"        "Experiment.Number"
##  [4] "Chip.Number"       "Stim.Condition"    "Time"
##  [7] "Population"        "Number.of.Cells"   "Well"
## [10] "Gene"              "Ct"

vbeta <- computeEtFromCt(vbeta)
vbeta.fa <- FluidigmAssay(vbeta, idvars = c("Subject.ID", "Chip.Number", "Well"),
    primerid = "Gene", measurement = "Et", ncells = "Number.of.Cells", geneid = "Gene",
    cellvars = c("Number.of.Cells", "Population"), phenovars = c("Stim.Condition",
        "Time"), id = "vbeta all")
show(vbeta.fa)

## FluidigmAssay  id:  vbeta all
##  456  wells;  75  features
```

We see that the variable `vbeta` is a `data.frame` from which we construct the `FluidigmAssay` object. The `idvars` is the set of column(s) in `vbeta` that uniquely identify a well (globally), the `primerid` is a column(s) that specify the feature measured at this well. The `measurement` gives the column name containing the log-expression measurement, `ncells` contains the number of cells (or other normalizing factor) for the well. `geneid`, `cellvars`, `phenovars` all specify additional columns to be included in the `featureData`, `phenoData` and `cellData` (TODO: wellData). The output is a `FluidigmAssay` object with 456 wells and 75 features.

We can access the feature–level metadata and the cell–level metadata using the `fData` and `cData` accessors.

```
head(fData(vbeta.fa), 3)

##          Gene
## B3GAT1 B3GAT1
## BAX       BAX
## BCL2     BCL2

head(cData(vbeta.fa), 3)

##                                  Number.of.Cells            Population
## 679a575f85011d3258fa212e9b16a29c               1 CD154+VbetaResponsive
## a5a844d4b0814eabb2d69505c1b4b96c               1 CD154+VbetaResponsive
## c98379705c703464173e09ab0cb7a401               1 CD154+VbetaResponsive
##                                  Subject.ID Chip.Number Well
## 679a575f85011d3258fa212e9b16a29c      Sub01           1  A01
## a5a844d4b0814eabb2d69505c1b4b96c      Sub01           1  A02
## c98379705c703464173e09ab0cb7a401      Sub01           1  A03
##                                  Stim.Condition Time
## 679a575f85011d3258fa212e9b16a29c      Stim(SEB)   12
## a5a844d4b0814eabb2d69505c1b4b96c      Stim(SEB)   12
## c98379705c703464173e09ab0cb7a401      Stim(SEB)   12
```

We see this gives us the set of genes measured in the assay, or the cell-level metadata (i.e. the number of cells measured in the well, the population this cell belongs to, the subject it came from, the chip it was run on, the well id, the stimulation it was subjected to, and the timepoint for the experiment this cell was part of). The wellKey is a hash of idvars columns, helping to ensure consistency when splitting and merging `SingleCellAssay`objects. TODO: Some of this "cell–level" information could arguably be part of the `@phenoData` slot of the object. This functionality is forthcoming but doesn't limit what can be done with the package at this stage.

4 Subsetting, splitting, combining

It's possible to subset `SingleCellAssay`objects by wells and features. Double square brackets ("[[") and `subset` will subset by wells on the first index and by features on the second index. Both integer and boolean indices may be used, as well as character vectors naming the cellKey or the feature (via the primerid).

```
sub1 <- vbeta.fa[[1:10]]
show(sub1)

## FluidigmAssay  id:  vbeta all
##  10  wells;  75  features

sub2 <- subset(vbeta.fa, Well == "A01")
show(sub2)

## FluidigmAssay  id:  vbeta all
##  5  wells;  75  features

sub3 <- vbeta.fa[[1:10, 6:10]]
show(sub3)
```

```
## FluidigmAssay  id:  vbeta all
##  10  wells;  5  features

cellData(sub3)

## An object of class 'AnnotatedDataFrame'
##   rowNames: 679a575f85011d3258fa212e9b16a29c
##     a5a844d4b0814eabb2d69505c1b4b96c ...
##     573145170fe2851b049af00470be106f (10 total)
##   varLabels: Number.of.Cells Population ... Time (7 total)
##   varMetadata: labelDescription

featureData(sub3)

## An object of class 'AnnotatedDataFrame'
##   rowNames: CCL4 CCL5 ... CCR5 (5 total)
##   varLabels: Gene
##   varMetadata: labelDescription
```

The cellData and featureData `AnnotatedDataFrames` are subset accordingly as well.

A `SingleCellAssay`may be split into a list of `SingleCellAssay`, which is known as an `SCASet`. The split method takes an argument which names the column (factor) on which to split the data. Each level of the factor will be placed in its own `SingleCellAssay`within the SCASet.

```
sp1 <- split(vbeta.fa, "Subject.ID")
show(sp1)

## SCASet of size  2
## Samples  Sub01, Sub02
```

The splitting variable can either be a character vector naming column(s) of the `SingleCellAssay`, or may be a `factor` or `list` of `factor`s.

It's possible to combine `SingleCellAssay`objects with the `combine` method.

```
combine(x = sp1[[1]], y = sp1[[2]])

## FluidigmAssay  id:  Sub01
##  456  wells;  75  features
```

TODO: Combining the contents of an SCASet would also be useful

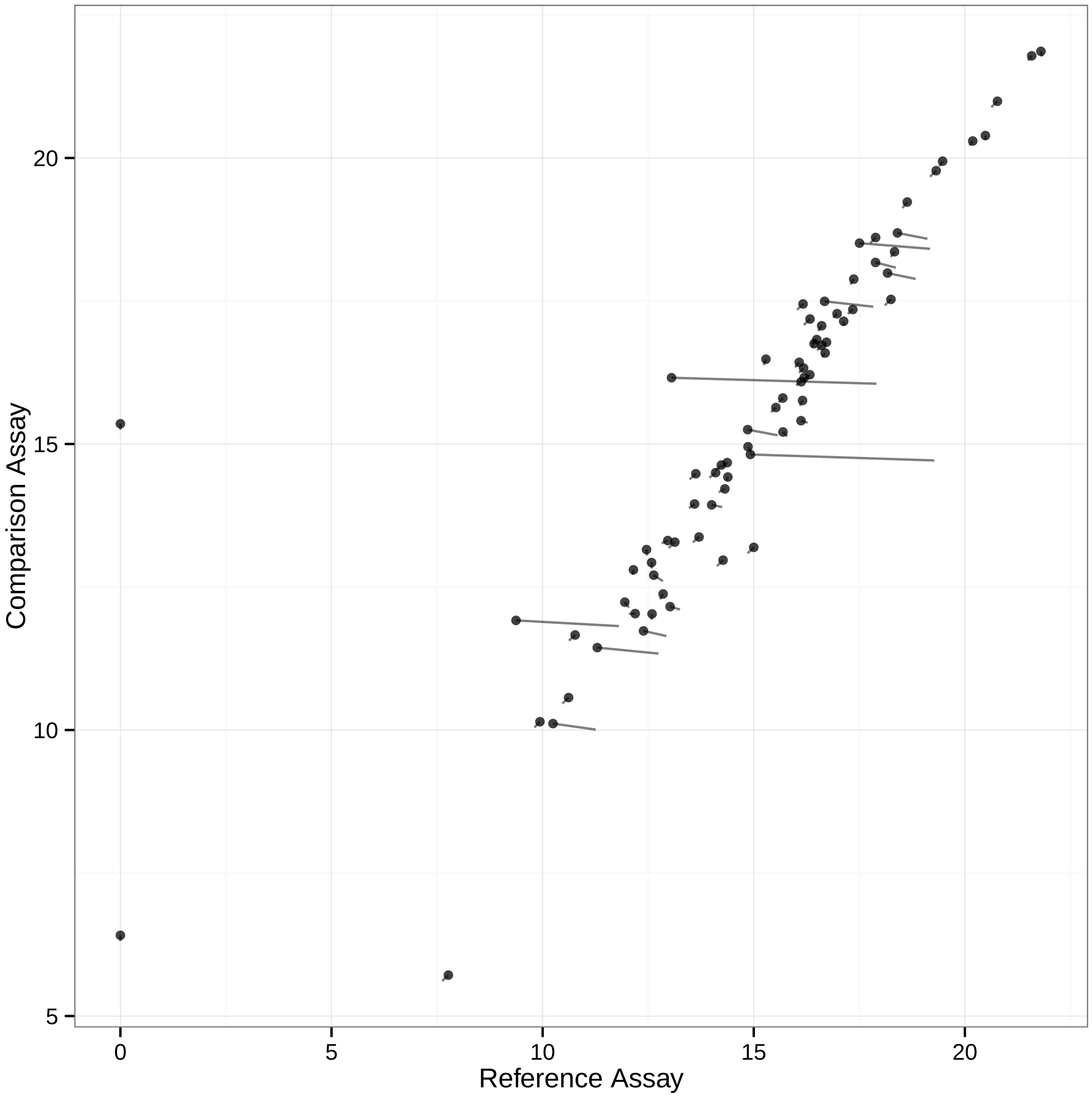
4.1 Filtering and Significance Testing

We can filter and perform some significance tests on the `SingleCellAssay`. We may want to filter any wells with at least two outlier cells where the discrete and continuous parts of the signal are at least 9 standard deviations from the mean. This is a very conservative filtering criteria. We'll group the filtering by the number of cells.

We'll split the assay by the number of cells and look at the concordance plot after filtering.

```
vbeta.split <- split(vbeta.fa, "Number.of.Cells")
# see default parameters for plotSCAConcordance
plotSCAConcordance(vbeta.split[[1]], vbeta.split[[2]], filterCriteria = list(nOutlier = 1,
    sigmaContinuous = 9, sigmaProportion = 9))

## Sum of Squares before Filtering:  14.95
##  After filtering:  12.4
##  Difference:  2.54
```


Comparison Assay
20
15
10
5
0
5
10
15
20
Reference Assay