The flowPhyto Package
(Version 1.12.1)

Francois Ribalet   David Schruth
ribalet@u.washington.edu

October 1, 2013

1   Licensing

This package is licensed under the Artistic License v2.0: it is therefore free to use and redistribute, however, we, the copyright holders, wish to maintain primary artistic control over any further development. Please be sure to cite us if you use this package in work leading to publication.

1. Ribalet, F., Schruth, D., Armbrust, E.V. flowPhyto: enabling automated analysis of microscopic algae from continuous flow cytometric data. 2011 *Bioinformatics*, doi: 10.1093/bioinformatics/btr003.

2   Installation

2.1   Unix/Linux/Mac

Building the *flowPhyto* package from source requires that you have a C compiler, and all of the prerequisites for the underlying flowCore package: namely the GNU Scientific library (GSL), and the Basic Linear Algebra Subprograms (BLAS). After these prerequisites are taken care of, the package is ready to install via:

```
R CMD INSTALL flowPhyto_x.y.z.tar.gz
```

After a successful installation the package can be loaded in the normal way: by starting R and invoking the `library` command like so:

```
> library(flowPhyto)
```

2.2   Windows

The *flowPhyto* package is compatible with the Windows version of R and the same prerequisites apply. However, the `pipeline` function and the downstream file-based functions which deploy the four analysis steps to a cluster are not currently supported.

3 Introduction

Flow cytometry is a widely used technique among biologists to study the abundances of populations of microscopic algae living in aquatic environments. A new generation of high-frequency flow cytometer, known as SeaFlow, collects up to several hundred samples per day and can run continuously for several weeks (see Ribalet *et al.*, 2010 for more details). Automated computational methods are needed to analyze the different phytoplankton populations present in each sample. Here we describe the *flowPhyto* R package which performs aggregate statistics on virtually unlimited collections of raw flow cytometry files in a memory efficient, parallelized fashion.

4 The SeaFlow Respository

SeaFlow data are stored in a custom binary file (EVT file) created every 3 minutes and consist of eight 16-bit integer channels namely:

```
> CHANNEL.CLMNS

[1] "fsc_small" "fsc_perp"  "fsc_big"    "pe"
[5] "chl_small" "chl_big"
```

The SeaFlow repository is composed of julian day labeled directories, each containing chronologically-ordered EVT files. The following code shows how to read one of these files into memory:

```
> evt.file.path <- system.file("extdata","seaflow_cruise","2011_001", "2.evt",
+                               package="flowPhyto")
> evt <- readSeaflow(evt.file.path)
```

5 Core Functions

5.1 OPP Filtration

Unlike a traditional flow cytometer, SeaFlow directly analyzes a raw stream of seawater using two detectors that determine the position of a particle in the focal region of the instrument optical system (Swalwell *et al.*, 2009). The `filter` function selects optimally positioned particles (OPP) in each EVT file that are used to distinguish the different phytoplankton populations.

```
> opp <- filter(evt, notch=1.1)
```

5.2 Cluster Based Classification

Because the characteristics of each phytoplankton population vary according to environmental conditions and instrument settings, a table of customizable parameters (pop.def.tab) is used to define the pre-gating regions and statistical priors of phytoplankton population clusters.

```
> opp.path <- system.file("extdata","seaflow_cruise","2011_001", "2.evt.opp",
+                                     package="flowPhyto")
> pop.def.path <- system.file("extdata","seaflow_cruise","pop.def.tab",
+                                     package="flowPhyto")
> opp <- readSeaflow(opp.path)
> def <- readPopDef(pop.def.path)
> def
            abrev              title  xmin  ymin  xmax  ymax
beads       beads               Beads 10000 30000 65000 65000
synecho   synecho      Synechococcus  7000  7000 35000 35000
crypto     crypto Cryptophyte-like 30000 30000 65000 65000
diatoms   diatoms      Pennates-like 20000 20000 65000 65000
ultra       ultra      Ultraplankton 25000 30000 40000 45000
nano         nano        Nanoplankton 40000 20000 65000 65000
pico         pico        Picoplankton 10000 15000 30000 35000
unknown   unknown             Unknown 40000     0 65000 20000
                  color      xvar         yvar u.co     lim
beads             black chl_small           pe 0.05   15000
synecho            tan2        pe chl_small 0.25  -10000
crypto          tomato3        pe chl_small 0.75   -1000
diatoms            gold fsc_small   chl_big 0.75   -5000
ultra       palegreen3 fsc_small chl_small 0.50      NA
nano          darkcyan fsc_small   chl_big 0.75      NA
pico      lightseagreen fsc_small chl_small 0.75      NA
unknown            grey fsc_small   chl_big 0.75      NA
```

Above we can see the default population definition table with the two dimentional pregating ranges and the parameters passed to the statistical clustering methods of the *flowClust* package (Lo *et al.* 2009).

Below, the `classify` function uses these pre-defined parameters and inputs one or more OPP files (3 by default) to classify individual phytoplankton cells into different populations.

```
> pop <- classify(x=opp, pop.def= def, func=2 )

[1] "Clustering 8 populations defined in pop.def table..."

> table(pop$pop)

      0       1       2   beads  crypto    nano synecho
    219    3810     178     415      11      39      52
  ultra unknown
    209      67
```

The `plotCytogram` function outputs a series of customizable 2-D cytograms to visualize the phytoplankton populations identified by the `classify` function.

```
> plotCytogram(pop, "fsc_small","chl_small", pop.def= def, add.legend=TRUE, cex=1)
>
```
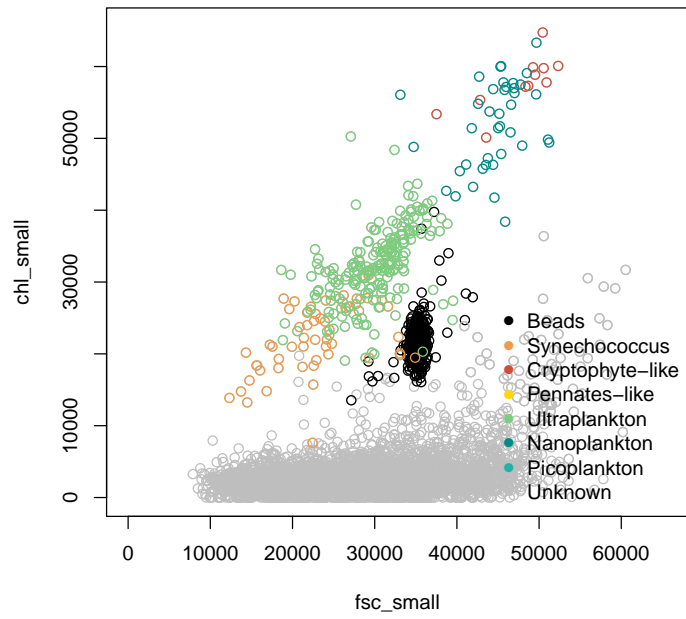
Figure 1: The above 2-D Cytogram depicts the phytoplankton population present in the sample.

5.3   Consensus and Census

`classify` outputs vector files (consensus.vct) that contain the population identification of the cells. `classify` is run in single file increments to provide multiple passes over a single cell and strengthen the clustering analysis. During the `census` step, these multiple-pass vector files are collapsed into one consensus vector, which represents the most likely population classification of the different phytoplankton cells. In addition, `census` produces a one-row census tab file that contains the number of cells per population for each file. The concatenation of these census tab files is used to create a per-population resampling scheme that calculates the number of OPP files necessary so a sufficient number of cells (500 by default) is present in the resampled population.

```
> vct.paths <- sapply(c(1,439,440), function(i)
+                 system.file("extdata","seaflow_cruise","2011_001",
+                       paste("1.evt.opp.",i,'-class.vct',sep=''),
+                             package="flowPhyto"))
> mat <- do.call(cbind,lapply(vct.paths, read.delim))
> consen.df <- consensus(mtrx=mat)
> table(consen.df$pop)

  beads    nano    pico synecho   ultra       x
     52      25      31      74     174    4644

> aggregate(consen.df$support,list(consen.df$pop), mean)

  Group.1        x
1   beads 2.923077
2    nano 2.960000
3    pico 2.774194
4 synecho 2.959459
5   ultra 2.977011
6       x 2.986865
```

Above is a table of cross tabulated sums per population of the generated consensus vector and a corresponding table of the average 'support' counts. The support column in the output of `consensus` keeps track of the number of the multiple-pass classification vectors that called an event as this population.

Compare the above population count cross tabulation with the output of census below.

```
> census(v=pop$pop, pop.def=def)

  beads synecho  crypto diatoms   ultra    nano    pico
    415      52      11       0     209      39       0
unknown       x
     67       0
```

5.4 Aggregate Statistics

The `summarize` function performs per-population aggregate statistics (cell concentration and the mean and standard deviation of the different channels) using the resampling scheme.

```
> filter.df <- readSeaflow(opp.path, add.yearday.file=TRUE)
> classed <- cbind.data.frame(filter.df, consen.df)
> names(opp.path) <- getFileNumber(opp.path)
> class.jn <- joinSDS(classed, opp.path)
> nrow.opp <- sapply(opp.path, function(p) readSeaflow(          p , count.only=TRUE))
> nrow.evt <- sapply(opp.path, function(p) readSeaflow(sub('.opp','',p), count.only=TRUE))
> class.jn$opp <- rep(nrow.opp, times=nrow.opp)
> class.jn$evt <- rep(nrow.evt, times=nrow.opp)
> summarize(class.jn,  opp.paths.str=opp.path)
```

	day	file	pop
x	2011_001	2	x
ultra	2011_001	2	ultra
synecho	2011_001	2	synecho
beads	2011_001	2	beads
pico	2011_001	2	pico
nano	2011_001	2	nano

	resam
x	/tmp/Rtmpyp4at3/Rinst54bf594aff89/flowPhyto/extdata/seaflow_cruise/2011_001/2.evt.op
ultra	/tmp/Rtmpyp4at3/Rinst54bf594aff89/flowPhyto/extdata/seaflow_cruise/2011_001/2.evt.op
synecho	/tmp/Rtmpyp4at3/Rinst54bf594aff89/flowPhyto/extdata/seaflow_cruise/2011_001/2.evt.op
beads	/tmp/Rtmpyp4at3/Rinst54bf594aff89/flowPhyto/extdata/seaflow_cruise/2011_001/2.evt.op
pico	/tmp/Rtmpyp4at3/Rinst54bf594aff89/flowPhyto/extdata/seaflow_cruise/2011_001/2.evt.op
nano	/tmp/Rtmpyp4at3/Rinst54bf594aff89/flowPhyto/extdata/seaflow_cruise/2011_001/2.evt.op

	time	lat	long	flow
x	2009-11-09 00:11:24	48.02425	-122.6206	2473.903
ultra	2009-11-09 00:11:24	48.02425	-122.6206	2473.903
synecho	2009-11-09 00:11:24	48.02425	-122.6206	2473.903
beads	2009-11-09 00:11:24	48.02425	-122.6206	2473.903
pico	2009-11-09 00:11:24	48.02425	-122.6206	2473.903
nano	2009-11-09 00:11:24	48.02425	-122.6206	2473.903

	bluk_red	salinity	temperature	event_rate
x	20.538	NaN	NaN	3171
ultra	20.538	NaN	NaN	3171
synecho	20.538	NaN	NaN	3171
beads	20.538	NaN	NaN	3171
pico	20.538	NaN	NaN	3171
nano	20.538	NaN	NaN	3171

	fluorescence	evt	opp	n	conc	fsc_small
x	5.31	4975	5000	4644	0.6226	30064.1
ultra	5.31	4975	5000	174	0.0233	29088.8

```
synecho          5.31 4975 5000   74 0.0099    29839.0
beads            5.31 4975 5000   52 0.0070    30441.9
pico             5.31 4975 5000   31 0.0042    32203.9
nano             5.31 4975 5000   25 0.0034    29994.3
        fsc_perp fsc_big      pe chl_small chl_big
x        29077.6       0 4986.8    6570.8  4254.3
ultra    28021.6       0 3277.9    6069.3  4224.1
synecho  28962.3       0 2763.4    4327.7  2572.5
beads    29857.3       0 4688.8    8068.7  5332.3
pico     31374.5       0 2191.4    7863.6  5024.5
nano     28703.6       0 4505.8    7090.7  4947.2
        fsc_small_sd fsc_perp_sd fsc_big_sd   pe_sd
x             9007.5      9217.8          0 12190.2
ultra         8937.5      9093.0          0  9397.1
synecho       9141.5      9086.8          0  8462.3
beads         9867.3      9492.3          0 12000.4
pico          8736.3      8945.1          0  4741.4
nano         11245.0     10762.6          0 12565.3
        chl_small_sd chl_big_sd
x             9495.7     6413.9
ultra         9673.5     6986.5
synecho       5564.9     4582.5
beads        11935.4     8221.6
pico         12587.0     8796.2
nano         10168.3     6885.9
```

The summarize function associates the corresponding acquisition time and location (latitude and longitude). It outputs a summary table of the entire set of SeaFlow data.

The plotStatMap creates customizable plots of the geo-referenced data created by summarize. A combination of the different parameters per population or a single parameter over different populations can be selected depending on the purpose of the analysis.

```
> stat.tab <- system.file("extdata","seaflow_cruise","stats.tab",
+                                          package="flowPhyto")
> stats <- read.delim(stat.tab)
> plotStatMap(df=stats, pop='ultra', z.param='conc', margin=0.2, zlab=expression(paste('Cell
+ main="Cell concentration of Ultra-plankton population")
> mtext(line=1, side=4, "cell concentration 10^6 cells / L")
>
```
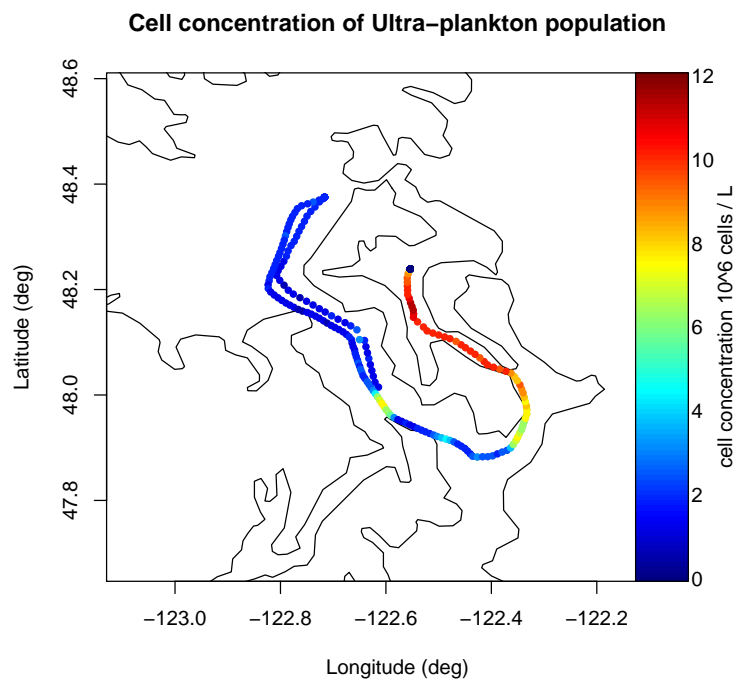
Figure 2: Ultra-plankton concentration for the Puget Sound in November, 2009

6 File-Based Functions, Input Validation, and The Pipeline

Each of the above core functions has a file based analog that takes one (or several) paths as it's main input parameter and outputs one or many files. For examples of these please check the man pages for individual file functions.

6.1 Directory Initializtion

First you'll need transfer the data to a location where there is plenty of extra disk space (around 25 percent more space than the raw EVT files alone). Then you'll want to make sure the directory has write access for the user who will be running the pipeline. Changing your working directory to the output directory is also recommended as many of the cruise specific job files get written there by default. Additional steps such as creating a log or plot sub directory in the repository or a new record in your cruise database (if you plan on uploading the resulting statistics or sds information to your database) may be desiarable as well.

6.2 SDS and pop.def.tab validation

One of the more important pre-processing steps to make can be with validating the SDS file before running the pipeline. One should both check for evidence of parsing errors that may have crept into the ship's data stream. The following code demonstrates one way this could be done, namely, longitude and latitude checking:

```
> path <- system.file("extdata","seaflow_cruise",package="flowPhyto")
> sds <- combineSdsFiles(path)
> plot(sds$LON, sds$LAT)
```

Additionally any externally defined population definition table should be validated using the following function.

```
> validatePopDef(readPopDef(pop.def.path))
```

```
[1] TRUE
```

An external pop.def can be specified by placing a file named 'pop.def.tab' in the cruise's directory. The parameter names and data types should match those found in the POP.DEF object. If such a file is not present, one will get created automatically from the dataframe hard coded into Define.R.

6.3 Running the Pipeline

The pipeline itself is merely a cluster deployment function which executes, in concerted batches, each of the file-based wrapper functions for the 4 main

analysis steps. Many of the sub-function specific parameters can also be passed through from this upper level function. The following example copies the very small bundled example data set to the present working directory and runs the pipeline for just step 4 which calculate statistics on a repository that has already undergone analysis steps 1 through 3.

```
> repository.dir <- '.'
> output.path <- paste(repository.dir,'/','seaflow_cruise',sep='')
> seaflow.path <- system.file("extdata", 'seaflow_cruise', package="flowPhyto")
> file.copy(from=seaflow.path, to=repository.dir, recursive=TRUE)

[1] TRUE

> pipeline(repo= repository.dir, cruise.name='seaflow_cruise', steps=4, parallel=FALSE, subm
> unlink(output.path, recursive=TRUE)
```

The most important parameters to set when calling pipeline are 'repo' which should be set to the location of your repository, and 'parallel' which tells the function whether or not to run in serial or parallel. Currently parallel jobs are simply submitted a via a cluster submission command such as 'qsub' (for Torque/SGE) or 'mosrun' (for MOSIX) as specified by the 'submit.cmd' option. For the purposes of this brief example 'parallel' has been set to FALSE but should almost always, where possible, be set to TRUE (the default) when running the pipeline over realistically sized, day or more long data sets. Additionally, the submit.cmd parameter was set to use qsub as a non-functional example. (Normally parallel=FALSE and submit.cmd would not be used together). Future plans for parallization include replacement of the above 'R CMD BATCH' and 'submit.cmd' based parallelization with a PVM/MPI based *snow* package implimentation.

6.4   Cleanup

There are two useful functions that can help to clean up the aftermath of all of the pipelined R CMD BATCH calls. The `cleanupLogs` function deletes log files depending on their error status. The `clearOutput` removes any output files from specified steps to clear the way for a re-run of the pipeline.

7   Example Dataset

The examples bundled with this dataset have been artifically reduced both in size and in number to make the package as light weight as possible. For a more realistic example you can visit our website `http://seaflow.ocean.washington.edu` to download a copy of the day-long 2009 Puget Sound cruise.

References

Lo,K. *et al.* (2009) flowclust: a bioconductor package for automated gating of flow cytometry data. *BMC Bioinformatics*, **10**(1):145.

Ribalet,F. *et al.* (2010) Unveiling a phytoplankton hotspot at a narrow boundary between coastal and offshore waters. *Proc. Nat. Acad. Sci.*, **107**(38):16571–16576.

Ribalet, F., Schruth, D., Armbrust, E.V. flowPhyto: enabling automated analysis of microscopic algae from continuous flow cytometric data. *Bioinformatics*, submitted.

Spidlen,J. *et al.* (2010) Data file standard for flow cytometry, version fcs 3.1. *Cytom. Part A*, **77**A(1):97–100.

Swalwell,J. *et al.* (2009) Virtual-core flow cytometry. *Cytom. Part A*, **75**A(11):960–965.