The *qcmetrics* infrastructure for quality control and automatic reporting

Laurent Gatto[1]

[1] lg390@cam.ac.uk

Computational Proteomics Unit[2]
University of Cambridge, UK

[2] http://cpu.sysbiol.cam.ac.uk

May 2, 2019

Abstract

The *qcmetrics* package is a framework that provides simple data containers for quality metrics and support for automatic report generation. This document briefly illustrates the core data structures and then demonstrates the generation and automation of quality control reports for microarray and proteomics data.

Keywords: Bioinformatics, Quality control, reporting, visualisation

Contents

1 Introduction . . . 2

2 The QC classes . . . 3

2.1 The `QcMetric` class . . . 4

2.2 The `QcMetrics` class . . . 5

3 Creating QC pipelines . . . 6

3.1 Microarray degradation . . . 6

3.2 A wrapper function . . . 10

3.3 Proteomics raw data . . . 11

3.4 Processed ^{15}N labelling data . . . 16

4 Report generation . . . 23

4.1 Custom reports . . . 23

4.2 New report types . . . 27

5 QC packages . . . 27

5.1 A simple RNA degradation package . . . 27

5.2 A QC pipeline repository . . . 28

6 Conclusions . . . 28

1 Introduction

Quality control (QC) is an essential step in any analytical process. Data of poor quality can at best lead to the absence of positive results or, much worse, false positives that stem from uncaught faulty and noisy data and much wasted resources in pursuing red herrings.

Quality is often a relative concept that depends on the nature of the biological sample, the experimental settings, the analytical process and other factors. Research and development in the area of QC has generally lead to two types of work being disseminated. Firstly, the comparison of samples of variable quality

and the identification of metrics that correlate with the quality of the data. These quality metrics could then, in later experiments, be used to assess their quality. Secondly, the design of domain-specific software to facilitate the collection, visualisation and interpretation of various QC metrics is also an area that has seen much development. QC is a prime example where standardisation and automation are of great benefit. While a great variety of QC metrics, software and pipelines have been described for any assay commonly used in modern biology, we present here a different tool for QC, whose main features are flexibility and versatility. The *qcmetrics* package is a general framework for QC that can accommodate any type of data. It provides a flexible framework to implement QC items that store relevant QC metrics with a specific visualisation mechanism. These individual items can be bundled into higher level QC containers that can be readily used to generate reports in various formats. As a result, it becomes easy to develop complete custom pipelines from scratch and automate the generation of reports. The pipelines can be easily updated to accommodate new QC items of better visualisation techniques.

Section 2 provides an overview of the framework. In section 3, we use microarray (subsection 3.1) and proteomics data (subsection 3.3) to demonstrate the elaboration of QC pipelines: how to create individual QC objects, how to bundle them to create sets of QC metrics and how to generate reports in multiple formats. We also show how the above steps can be fully automated through simple wrapper functions in section 3.2. Although kept simple in the interest of time and space, these examples are meaningful and relevant. In section 4, we provide more detail about the report generation process, how reports can be customised and how new exports can be contributed. We proceed in section 5 to the consolidation of QC pipelines using *R*and elaborate on the development of dedicated QC packages with *qcmetrics*.

2 The QC classes

The package provides two types of QC containers. The `QcMetric` class stores data and visualisation functions for single metrics. Several such metrics can be bundled into `QcMetrics` instances, that can be used as input for automated report generation. Below, we will provide a quick overview of how to create respective `QcMetric` and `QcMetrics` instances. More details are available in the corresponding documentations.

2.1 The `QcMetric` class

A QC metric is composed of a description (`name` in the code chunk below), some QC data (`qcdata`) and a `status` that defines if the metric is deemed of acceptable quality (coded as `TRUE`), bad quality (coded as `FALSE`) or not yet evaluated (coded as `NA`). Individual metrics can be displayed as a short textual summary or plotted. To do the former, one can use the default `show` method.

```
library("qcmetrics")
qc <- QcMetric(name = "A test metric")
qcdata(qc, "x") <- rnorm(100)
qcdata(qc) ## all available qcdata

## [1] "x"

summary(qcdata(qc, "x")) ## get x

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.2147 -0.4942  0.1139  0.1089  0.6915  2.4016

show(qc) ## or just qc

## Object of class "QcMetric"
##  Name: A test metric
##  Status: NA
##  Data: x

status(qc) <- TRUE
qc

## Object of class "QcMetric"
##  Name: A test metric
##  Status: TRUE
##  Data: x
```

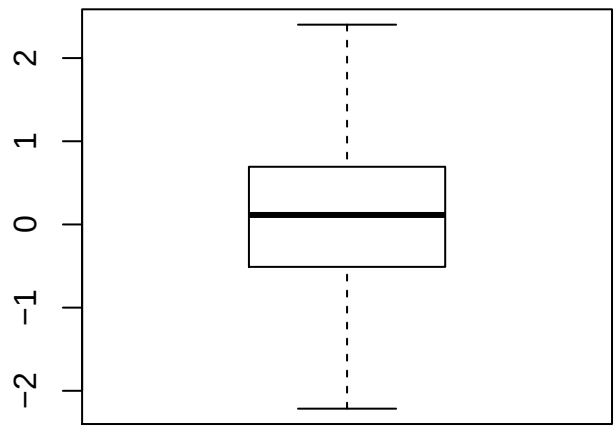
Plotting `QcMetric` instances requires to implement a plotting method that is relevant to the data at hand. We can use a `plot` replacement method to define our custom function. The code inside the `plot` uses `qcdata` to extract the relevant QC data from `object` that is then passed as argument to `plot` and uses the adequate visualisation to present the QC data.

```
plot(qc)

## Warning in x@plot(x, ...):  No specific plot function defined

plot(qc) <-
    function(object, ... ) boxplot(qcdata(object, "x"), ...)
```

```
plot(qc)
```



2.2 The `QcMetrics` class

A `QcMetrics` object is essentially just a list of individual `QcMetric` instances. It is also possible to set a list of metadata variables to describe the source of the QC metrics. The metadata can be passed as an `QcMetadata` object (the way it is stored in the `QcMetrics` instance) or directly as a named `list`. The `QcMetadata` is itself a `list` and can be accessed and set with `metadata` or `mdata`. When accessed, it is returned and displayed as a `list`.

```
qcm <- QcMetrics(qcdata = list(qc))
qcm

## Object of class "QcMetrics"
##  containing 1 QC metrics.
##  and no metadata variables.

metadata(qcm) <- list(author = "Prof. Who",
                      lab = "Big lab")
qcm

## Object of class "QcMetrics"
##  containing 1 QC metrics.
##  and 2 metadata variables.

mdata(qcm)

## $author
## [1] "Prof. Who"
##
```

```
## $lab
## [1] "Big lab"
```

The metadata can be updated with the same interface. If new named items are passed, the metadata is updated by addition of the new elements. If a named item is already present, its value gets updated.

```
metadata(qcm) <- list(author = "Prof. Who",
                      lab = "Cabin lab",
                      University = "Universe-ity")
mdata(qcm)
```

```
## $author
## [1] "Prof. Who"
##
## $lab
## [1] "Cabin lab"
##
## $University
## [1] "Universe-ity"
```

The `QcMetrics` can then be passed to the `qcReport` method to generate reports, as described in more details below.

3 Creating QC pipelines

3.1 Microarray degradation

We will use the `refA` Affymetrix arrays from the *MAQCsubsetAFX* package as an example data set and investigate the RNA degradation using the `AffyRNAdeg` from *affy* [1] and the actin and GAPDH $\frac{3'}{5'}$ ratios, as calculated in the *yaqcaffy* package [2]. The first code chunk demonstrate how to load the data and compute the QC data[3].

```
library("MAQCsubsetAFX")
data(refA)
library("affy")
deg <- AffyRNAdeg(refA)
library("yaqcaffy")
yqc <- yaqc(refA)
```

[3]The pre-computed objects can be directly loaded with `load(system.file("extdata/ package = "qc metrics"))` and `load(system.file("extdata/ package = "qcmet rics"))`.

We then create two `QcMetric` instances, one for each of our quality metrics.

```
qc1 <- QcMetric(name = "Affy RNA degradation slopes")
qcdata(qc1, "deg") <- deg
plot(qc1) <- function(object, ...) {
    x <- qcdata(object, "deg")
    nms <- x$sample.names
    plotAffyRNAdeg(x, col = 1:length(nms), ...)
    legend("topleft", nms, lty = 1, cex = 0.8,
           col = 1:length(nms), bty = "n")
}
status(qc1) <- TRUE
qc1
```

```
## Object of class "QcMetric"
##  Name: Affy RNA degradation slopes
##  Status: TRUE
##  Data: deg
```

```
qc2 <- QcMetric(name = "Affy RNA degradation ratios")
qcdata(qc2, "yqc") <- yqc
plot(qc2) <- function(object, ...) {
    par(mfrow = c(1, 2))
    yaqcaffy:::.plotQCRatios(qcdata(object, "yqc"), "all", ...)
}
status(qc2) <- FALSE
qc2
```

```
## Object of class "QcMetric"
##  Name: Affy RNA degradation ratios
##  Status: FALSE
##  Data: yqc
```

Then, we combine the individual QC items into a `QcMetrics` instance.

```
maqcm <- QcMetrics(qcdata = list(qc1, qc2))
maqcm
```

```
## Object of class "QcMetrics"
##  containing 2 QC metrics.
##  and no metadata variables.
```

With our `QcMetrics` data, we can easily generate quality reports in several different formats. Below, we create a `pdf` report, which is the default type. Using `type = "html"` would generate the equivalent report in `html` format. See `?qcReport` for more details.

```
qcReport(maqcm, reportname = "rnadeg", type = "pdf")
```

The resulting report is shown below. Each `QcMetric` item generates a section named according to the object's name. A final summary section shows a table with all the QC items and their status. The report concludes with a detailed session information section.

In addition to the report, it is of course advised to store the actual `QcMetrics` object. This is most easily done with the *R* `save`/`load` and `saveRDS`/`readRDS` functions. As the data and visualisation methods are stored together, it is possible to reproduce the figures from the report or further explore the data at a later stage.

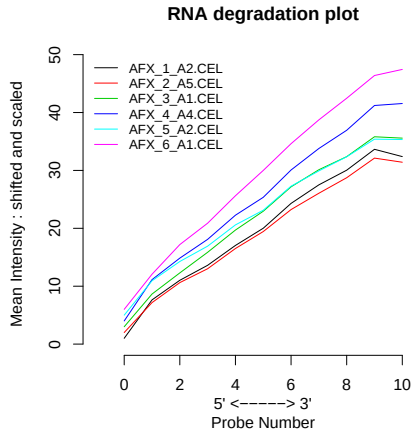
Quality control report generated with `qcmetrics`

biocbuild

May 2, 2019

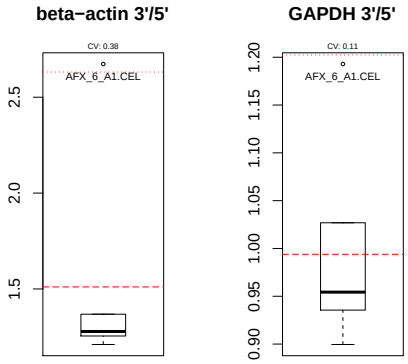
1 Affy RNA degradation slopes

```
## Object of class "QcMetric"
##  Name: Affy RNA degradation slopes
##  Status: TRUE
##  Data: deg
```



2 Affy RNA degradation ratios

```
## Object of class "QcMetric"
##  Name: Affy RNA degradation ratios
##  Status: FALSE
##  Data: yqc
```



3 QC summary

	name	status
1	Affy RNA degradation slopes	TRUE
2	Affy RNA degradation ratios	FALSE

4 Session information

- R version 3.6.0 (2019-04-26), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.46.0, Biobase 2.44.0, BiocGenerics 0.30.0, BiocStyle 2.12.0, IRanges 2.18.0, MAQCsubsetAFX 1.21.0, MSnbase 2.10.0, ProtGenerics 1.16.0, Rcpp 1.0.1, RforProteomics 1.21.0, S4Vectors 0.22.0, affy 1.62.0, gcrma 2.56.0, genefilter 1.66.0, knitr 1.22, mzR 2.18.0, qcmetrics 1.22.0, simpleaffy 2.60.0, xtable 1.8-4, yaqcaffy 1.44.0
- Loaded via a namespace (and not attached): BiocManager 1.30.4, BiocParallel 1.18.0, Biostrings 2.52.0, DBI 1.0.0, MALDIquant 1.19.2, MASS 7.3-51.4, Matrix 1.2-17, Nozzle.R1 1.1-1, R.methodsS3 1.7.1, R.oo 1.22.0, R.utils 2.8.0, R6 2.4.0, RBGL 1.60.0, RCurl 1.95-4.12, RSQLite 2.1.1, RUnit 0.4.32, XML 3.98-1.19, XVector 0.24.0, affyio 1.54.0, annotate 1.62.0, assertthat 0.2.1, biocViews 1.52.0, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, bookdown 0.9, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.0, crayon 1.3.4, digest 0.6.18, doParallel 1.0.14, dplyr 0.8.0.1, evaluate 0.13, foreach 1.4.4, ggplot2 3.1.1, glue 1.3.1, graph 1.62.0, grid 3.6.0, gtable 0.3.0, highr 0.8, htmltools 0.3.6, httpuv 1.5.1, impute 1.58.0, iterators 1.0.10, later 0.8.0, lattice 0.20-38, lazyeval 0.2.2, limma 3.40.0, magrittr 1.5, memoise 1.1.0, mime 0.6, munsell 0.5.0, mzID 1.22.0, ncdf4 1.16.1, pander 0.6.3, pcaMethods 1.76.0, pillar 1.3.1, pkgconfig 2.0.2, plyr 1.8.4, preprocessCore 1.46.0, promises 1.0.1, purrr 0.3.2, rlang 0.3.4, rmarkdown 1.12, scales 1.0.0, shiny 1.3.2, splines 3.6.0, stringi 1.4.3, stringr 1.4.0, survival 2.44-1.1, tibble 2.1.1, tidyselect 0.2.5, tools 3.6.0, vsn 3.52.0, xfun 0.6, yaml 2.2.0, zlibbioc 1.30.0

3.2 A wrapper function

Once an appropriate set of quality metrics has been identified, the generation of the `QcMetrics` instances can be wrapped up for automation.

```
rnadeg

## function (input, status, type, reportname = "rnadegradation")
## {
##     requireNamespace("affy")
##     requireNamespace("yaqcaffy")
##     if (is.character(input))
##         input <- affy::ReadAffy(input)
##     qc1 <- QcMetric(name = "Affy RNA degradation slopes")
##     qcdata(qc1, "deg") <- affy::AffyRNAdeg(input)
##     plot(qc1) <- function(object) {
##         x <- qcdata(object, "deg")
##         nms <- x$sample.names
##         affy::plotAffyRNAdeg(x, cols = 1:length(nms))
##         legend("topleft", nms, lty = 1, cex = 0.8, col = 1:length(nms),
##             bty = "n")
##     }
##     if (!missing(status))
##         status(qc1) <- status[1]
##     qc2 <- QcMetric(name = "Affy RNA degradation ratios")
##     qcdata(qc2, "yqc") <- yaqcaffy::yaqc(input)
##     plot(qc2) <- function(object) {
##         par(mfrow = c(1, 2))
##         yaqcaffy:::.plotQCRatios(qcdata(object, "yqc"), "all")
##     }
##     if (!missing(status))
##         status(qc2) <- status[2]
##     qcm <- QcMetrics(qcdata = list(qc1, qc2))
##     if (!missing(type))
##         qcReport(qcm, reportname, type = type, title = "Affymetrix RNA degradation repo
##     invisible(qcm)
## }
## <bytecode: 0x555ce85135f0>
## <environment: namespace:qcmetrics>
```

It is now possible to generate a `QcMetrics` object from a set of CEL files or directly from an `affybatch` object. The `status` argument allows to directly set the statuses of the individual QC items; these can also be set later, as illustrated below. If a report type is specified, the corresponding report is generated.

```
maqcm <- rnadeg(refA)
```

```
status(maqcm)

## [1] NA NA

## check the QC data
(status(maqcm) <- c(TRUE, FALSE))

## [1]  TRUE FALSE
```

The report can be generated manually with `qcReport(maqcm)` or directly with the wrapper function as follows:

```
maqcm <- rnadeg(refA, type = "pdf")
```

3.3 Proteomics raw data

To illustrate a simple QC analysis for proteomics data, we will download data set `PXD00001` from the ProteomeXchange repository in the `mzXML` format [3]. The MS^2 spectra from that mass-spectrometry run are then read into *R*[4] and stored as an `MSnExp` experiment using the `readMSData` function from the *MSnbase* package [4].

[4] In the interest of time, this code chunk has been pre-computed and a subset (1 in 3) of the `exp` instance is distributed with the package. The data is loaded with `load(system.file("extdata/ package = "qcmet rics"))`.

```
library("RforProteomics")
msfile <- getPXD000001mzXML()
library("MSnbase")
exp <- readMSData(msfile, verbose = FALSE)
```

The `QcMetrics` will consist of 3 items, namely a chromatogram constructed with the MS^2 spectra precursor's intensities, a figure illustrating the precursor charges in the MS space and an $\frac{m}{z}$ delta plot illustrating the suitability of MS^2 spectra for identification (see `?plotMzDelta` or [5]).

```
qc1 <- QcMetric(name = "Chromatogram")
x <- rtime(exp)
y <- precursorIntensity(exp)
o <- order(x)
```

```
qcdata(qc1, "x") <- x[o]
qcdata(qc1, "y") <- y[o]
plot(qc1) <- function(object, ...)
    plot(qcdata(object, "x"),
         qcdata(object, "y"),
         col = "darkgrey", type ="l",
         xlab = "retention time",
         ylab = "precursor intensity")
```

```
qc2 <- QcMetric(name = "MS space")
qcdata(qc2, "p2d") <- plot2d(exp, z = "charge", plot = FALSE)
plot(qc2) <- function(object) {
    require("ggplot2")
    print(qcdata(object, "p2d"))
}
```

```
qc3 <- QcMetric(name = "m/z delta plot")
qcdata(qc3, "pmz") <- plotMzDelta(exp, plot = FALSE,
                                  verbose = FALSE)
plot(qc3) <- function(object)
    suppressWarnings(print(qcdata(object, "pmz")))
```

Note that we do not store the raw data in any of the above instances, but always pre-compute the necessary data or plots that are then stored as `qcdata`. If the raw data was to be needed in multiple `QcMetric` instances, we could re-use the same `qcdata` *environment* to avoid unnecessary copies using `qcdata(qc2) <- qcenv(qc1)` and implement different views through custom `plot` methods.

Let's now combine the three items into a `QcMetrics` object, decorate it with custom metadata using the MIAPE information from the `MSnExp` object and generate a report.

```
protqcm <- QcMetrics(qcdata = list(qc1, qc2, qc3))
metadata(protqcm) <- list(
    data = "PXD000001",
    instrument = experimentData(exp)@instrumentModel,
    source = experimentData(exp)@ionSource,
    analyser = experimentData(exp)@analyser,
    detector = experimentData(exp)@detectorType,
    manufacurer = experimentData(exp)@instrumentManufacturer)
```

The status column of the summary table is empty as we have not set the QC items statuses yet.

```
qcReport(protqcm, reportname = "protqc")
```

Quality control report generated with `qcmetrics`

biocbuild

May 2, 2019

1 Metadata

data PXD000001

instrument LTQ Orbitrap Velos

source nanoelectrospray

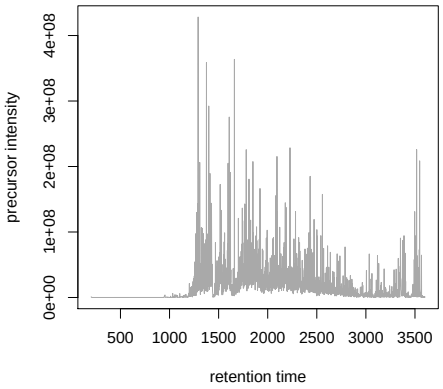
analyser orbitrap

detector inductive detector

manufacurer Thermo Scientific

2 Chromatogram

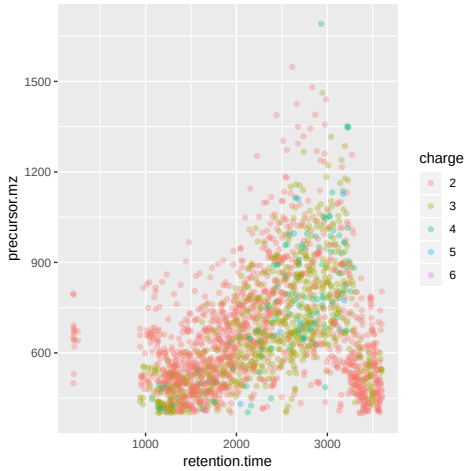
```
## Object of class "QcMetric"
##  Name: Chromatogram
##  Status: NA
##  Data: x y
```



3 MS space

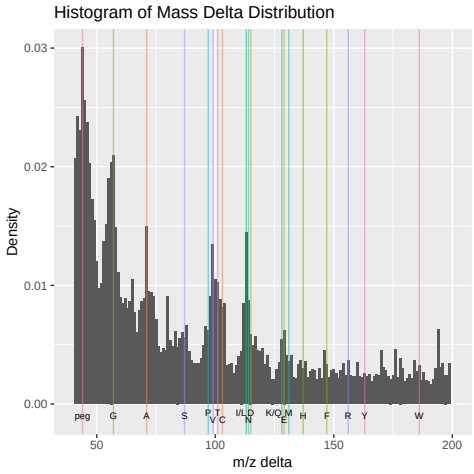
```
## Object of class "QcMetric"
##  Name: MS space
##  Status: NA
##  Data: p2d
```

```
## Loading required package:  ggplot2
```



4 m/z delta plot

```
## Object of class "QcMetric"
##  Name: m/z delta plot
##  Status: NA
##  Data: pmz
```



5 QC summary

	name	status
1	Chromatogram	
2	MS space	
3	m/z delta plot	

6 Session information

- R version 3.6.0 (2019-04-26), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.46.0, Biobase 2.44.0, BiocGenerics 0.30.0, BiocStyle 2.12.0, IRanges 2.18.0, MAQCsubsetAFX 1.21.0, MSnbase 2.10.0, ProtGenerics 1.16.0, Rcpp 1.0.1, RforProteomics 1.21.0, S4Vectors 0.22.0, affy 1.62.0, gcrma 2.56.0, genefilter 1.66.0, ggplot2 3.1.1, knitr 1.22, mzR 2.18.0, qcmetrics 1.22.0, simpleaffy 2.60.0, xtable 1.8-4, yaqcaffy 1.44.0
- Loaded via a namespace (and not attached): BiocManager 1.30.4, BiocParallel 1.18.0, Biostrings 2.52.0, DBI 1.0.0, MALDIquant 1.19.2, MASS 7.3-51.4, Matrix 1.2-17, Nozzle.R1 1.1-1, R.methodsS3 1.7.1, R.oo 1.22.0, R.utils 2.8.0, R6 2.4.0, RBGL 1.60.0, RCurl 1.95-4.12, RSQLite 2.1.1, RUnit 0.4.32, XML 3.98-1.19, XVector 0.24.0, affyio 1.54.0, annotate 1.62.0, assertthat 0.2.1, biocViews 1.52.0, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, bookdown 0.9, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.0, crayon 1.3.4, digest 0.6.18, doParallel 1.0.14, dplyr 0.8.0.1, evaluate 0.13, foreach 1.4.4, glue 1.3.1, graph 1.62.0, grid 3.6.0, gtable 0.3.0, highr 0.8, htmltools 0.3.6, httpuv 1.5.1, impute 1.58.0, iterators 1.0.10, labeling 0.3, later 0.8.0, lattice 0.20-38, lazyeval 0.2.2, limma 3.40.0, magrittr 1.5, memoise 1.1.0, mime 0.6, munsell 0.5.0, mzID 1.22.0, ncdf4 1.16.1, pander 0.6.3, pcaMethods 1.76.0, pillar 1.3.1, pkgconfig 2.0.2, plyr 1.8.4, preprocessCore 1.46.0, promises 1.0.1, purrr 0.3.2, rlang 0.3.4, rmarkdown 1.12, scales 1.0.0, shiny 1.3.2, splines 3.6.0, stringi 1.4.3, stringr 1.4.0, survival 2.44-1.1, tibble 2.1.1, tidyselect 0.2.5, tools 3.6.0, vsn 3.52.0, withr 2.1.2, xfun 0.6, yaml 2.2.0, zlibbioc 1.30.0

3.4 Processed ^{15}N labelling data

In this section, we describe a set of ^{15}N metabolic labelling QC metrics [6]. The data is a phospho-enriched ^{15}N labelled *Arabidopsis thaliana* sample prepared as described in [7]. The data was processed with in-house tools and is available as an `MSnSet` instance. Briefly, MS^2 spectra were search with the Mascot engine and identification scores adjusted with Mascot Percolator. Heavy and light pairs were then searched in the survey scans and ^{15}N incorporation was estimated based on the peptide sequence and the isotopic envelope of the heavy member of the pair (the `inc` feature variable). Heavy and light peptides isotopic envelope areas were finally integrated to obtain unlabelled and ^{15}N quantitation data. The `psm` object provides such data for PSMs (peptide spectrum matches) with a posterior error probability $<$0.05 that can be uniquely matched to proteins.

We first load the *MSnbase* package (required to support the `MSnSet` data structure) and example data that is distributed with the *qcmetrics* package. We will make use of the *ggplot2* plotting package.

```
library("ggplot2")
library("MSnbase")
data(n15psm)
psm
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 1772 features, 2 samples
##   element names: exprs
## protocolData: none
## phenoData: none
## featureData
##   featureNames: 3 5 ... 4499 (1772 total)
##   fvarLabels: Protein_Accession
##     Protein_Description ... inc (21 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
##   pubMedIds: 23681576
## Annotation:
## - - - Processing information - - -
## Subset [22540,2][1999,2] Tue Sep 17 01:34:09 2013
## Removed features with more than 0 NAs: Tue Sep 17 01:34:09 2013
## Dropped featureData's levels Tue Sep 17 01:34:09 2013
##  MSnbase version: 1.9.7
```

The first QC item examines the ^{15}N incorporation rate, available in the `inc` feature variable. We also defined a median incorporation rate threshold `tr` equal to 97.5 that is used to set the QC status.

```
## incorporation rate QC metric
qcinc <- QcMetric(name = "15N incorporation rate")
qcdata(qcinc, "inc") <- fData(psm)$inc
qcdata(qcinc, "tr") <- 97.5
status(qcinc) <- median(qcdata(qcinc, "inc")) > qcdata(qcinc, "tr")
```

Next, we implement a custom `show` method, that prints 5 summary values of the variable's distribution.

```
show(qcinc) <- function(object) {
    qcshow(object, qcdata = FALSE)
    cat(" QC threshold:", qcdata(object, "tr"), "\n")
    cat(" Incorporation rate\n")
    print(summary(qcdata(object, "inc")))
    invisible(NULL)
}
```

We then define the metric's `plot` function that represent the distribution of the PSM's incorporation rates as a boxplot, shows all the individual rates as jittered dots and represents the `tr` threshold as a dotted red line.

```
plot(qcinc) <- function(object) {
    inc <- qcdata(object, "inc")
    tr <- qcdata(object, "tr")
    lab <- "Incorporation rate"
    dd <- data.frame(inc = qcdata(qcinc, "inc"))
    p <- ggplot(dd, aes(factor(""), inc)) +
        geom_jitter(colour = "#4582B370", size = 3) +
    geom_boxplot(fill = "#FFFFFFD0", colour = "#000000",
                 outlier.size = 0) +
    geom_hline(yintercept = tr, colour = "red",
               linetype = "dotted", size = 1) +
    labs(x = "", y = "Incorporation rate")
    p
}
```

^{15}N experiments of good quality are characterised by high incorporation rates, which allow to deconvolute the heavy and light peptide isotopic envelopes and accurate quantification.

The second metric inspects the $\log_2$ fold-changes of the PSMs, unique peptides with modifications, unique peptide sequences (not taking modifications into account) and proteins. These respective data sets are computed with the `combineFeatures` function (see `?combineFeatures` for details).

```
fData(psm)$modseq <- ## pep seq + PTM
    paste(fData(psm)$Peptide_Sequence,
          fData(psm)$Variable_Modifications, sep = "+")
pep <- combineFeatures(psm,
                       as.character(fData(psm)$Peptide_Sequence),
                       "median", verbose = FALSE)
modpep <- combineFeatures(psm,
                          fData(psm)$modseq,
                          "median", verbose = FALSE)
prot <- combineFeatures(psm,
                        as.character(fData(psm)$Protein_Accession),
                        "median", verbose = FALSE)
```

The $\log_2$ fold-changes for all the features are then computed and stored as QC data of our next QC item. We also store a pair of values `explfc` that defined an interval in which we expect our median PSM $\log_2$ fold-change to be.

```
## calculate log fold-change
qclfc <- QcMetric(name = "Log2 fold-changes")
qcdata(qclfc, "lfc.psm") <-
    log2(exprs(psm)[,"unlabelled"] / exprs(psm)[, "N15"])
qcdata(qclfc, "lfc.pep") <-
    log2(exprs(pep)[,"unlabelled"] / exprs(pep)[, "N15"])
qcdata(qclfc, "lfc.modpep") <-
    log2(exprs(modpep)[,"unlabelled"] / exprs(modpep)[, "N15"])
qcdata(qclfc, "lfc.prot") <-
    log2(exprs(prot)[,"unlabelled"] / exprs(prot)[, "N15"])
qcdata(qclfc, "explfc") <- c(-0.5, 0.5)

status(qclfc) <-
    median(qcdata(qclfc, "lfc.psm")) > qcdata(qclfc, "explfc")[1] &
    median(qcdata(qclfc, "lfc.psm")) < qcdata(qclfc, "explfc")[2]
```

As previously, we provide a custom `show` method that displays summary values for the four fold-changes. The `plot` function illustrates the respective $\log_2$ fold-change densities and the expected median PSM fold-change range (red rectangle). The expected 0 $\log_2$ fold-change is shown as a dotted black vertical line and the observed median PSM value is shown as a blue dashed line.

```
show(qclfc) <- function(object) {
    qcshow(object, qcdata = FALSE) ## default
    cat(" QC thresholds:", qcdata(object, "explfc"), "\n")
    cat(" * PSM log2 fold-changes\n")
    print(summary(qcdata(object, "lfc.psm")))
    cat(" * Modified peptide log2 fold-changes\n")
    print(summary(qcdata(object, "lfc.modpep")))
    cat(" * Peptide log2 fold-changes\n")
    print(summary(qcdata(object, "lfc.pep")))
    cat(" * Protein log2 fold-changes\n")
    print(summary(qcdata(object, "lfc.prot")))
    invisible(NULL)
}
plot(qclfc) <- function(object) {
    x <- qcdata(object, "explfc")
    plot(density(qcdata(object, "lfc.psm")),
         main = "", sub = "", col = "red",
         ylab = "", lwd = 2,
         xlab = expression(log[2]~fold-change))
    lines(density(qcdata(object, "lfc.modpep")),
          col = "steelblue", lwd = 2)
    lines(density(qcdata(object, "lfc.pep")),
          col = "blue", lwd = 2)
    lines(density(qcdata(object, "lfc.prot")),
          col = "orange")
    abline(h = 0, col = "grey")
    abline(v = 0, lty = "dotted")
    rect(x[1], -1, x[2], 1, col = "#EE000030",
         border = NA)
    abline(v = median(qcdata(object, "lfc.psm")),
           lty = "dashed", col = "blue")
    legend("topright",
           c("PSM", "Peptides", "Modified peptides", "Proteins"),
           col = c("red", "steelblue", "blue", "orange"), lwd = 2,
           bty = "n")
}
```

A good quality experiment is expected to have a tight distribution centred around 0. Major deviations would indicate incomplete incorporation, errors in the respective amounts of light and heavy material used, and a wide distribution would reflect large variability in the data.

Our last QC item inspects the number of features that have been identified in the experiment. We also investigate how many peptides (with or without considering the modification) have been observed at the PSM level and the number of unique peptides per protein. Here, we do not specify any expected values as the number of observed features is experiment specific; the QC status is left as `NA`.

```
## number of features
qcnb <- QcMetric(name = "Number of features")
qcdata(qcnb, "count") <- c(
    PSM = nrow(psm),
    ModPep = nrow(modpep),
    Pep = nrow(pep),
    Prot = nrow(prot))
qcdata(qcnb, "peptab") <-
    table(fData(psm)$Peptide_Sequence)
qcdata(qcnb, "modpeptab") <-
    table(fData(psm)$modseq)
qcdata(qcnb, "upep.per.prot") <-
    fData(psm)$Number_Of_Unique_Peptides
```

The counts are displayed by the new `show` and plotted as bar charts by the `plot` methods.

```
show(qcnb) <- function(object) {
    qcshow(object, qcdata = FALSE)
    print(qcdata(object, "count"))
}
plot(qcnb) <- function(object) {
    par(mar = c(5, 4, 2, 1))
    layout(matrix(c(1, 2, 1, 3, 1, 4), ncol = 3))
    barplot(qcdata(object, "count"), horiz = TRUE, las = 2)
    barplot(table(qcdata(object, "modpeptab")),
            xlab = "Modified peptides")
    barplot(table(qcdata(object, "peptab")),
            xlab = "Peptides")
    barplot(table(qcdata(object, "upep.per.prot")),
            xlab = "Unique peptides per protein ")
}
```

In the code chunk below, we combine the 3 QC items into a `QcMetrics` instance and generate a report using meta data extracted from the `psm` `MSnSet` instance.

```
n15qcm <- QcMetrics(qcdata = list(qcinc, qclfc, qcnb))
qcReport(n15qcm, reportname = "n15qcreport",
         title = expinfo(experimentData(psm))["title"],
         author = expinfo(experimentData(psm))["contact"],
         clean = FALSE)

## Report written to n15qcreport.pdf
```

We provide with the package the `n15qc` wrapper function that automates the above pipeline. The names of the feature variable columns and the thresholds for the two first QC items are provided as arguments. In case no report name is given, a custom title with date and time is used, to avoid overwriting existing reports.

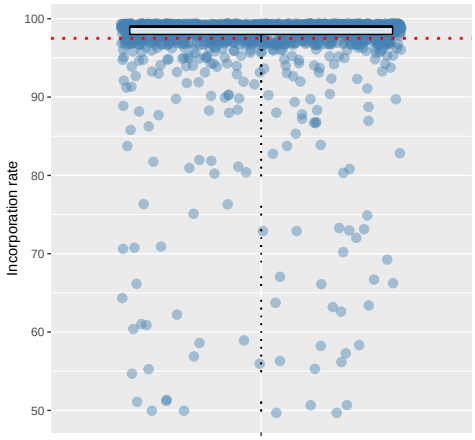
15N labelling experiment

Arnoud Groen

May 2, 2019

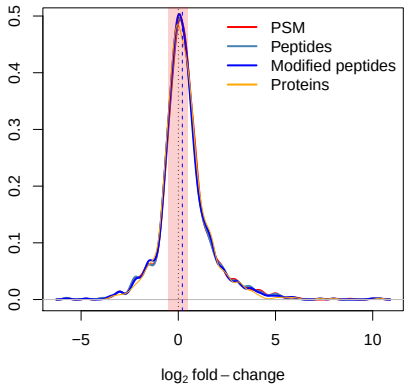
1 15N incorporation rate

```
## Object of class "QcMetric"
##  Name: 15N incorporation rate
##  Status: TRUE
##  QC threshold: 97.5
##  Incorporation rate
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   50.00   98.00   99.00   97.04   99.00   99.00
```



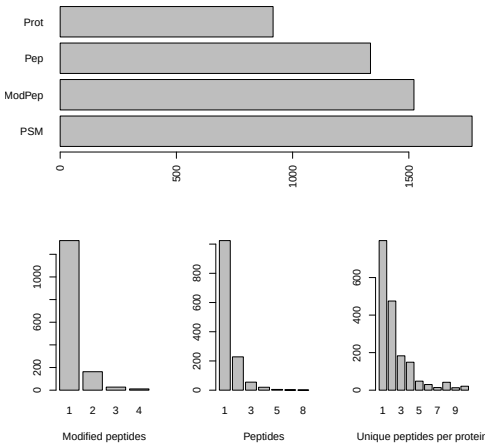
2 Log2 fold-changes

```
## Object of class "QcMetric"
##  Name: Log2 fold-changes
##  Status: TRUE
##  QC thresholds: -0.5 0.5
##  * PSM log2 fold-changes
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.7641 -0.3164  0.2087  0.3536  0.8242 10.3712
##  * Modified peptide log2 fold-changes
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.7641 -0.3306  0.1946  0.3393  0.8001 10.3712
##  * Peptide log2 fold-changes
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.7268 -0.3285  0.1854  0.3317  0.7934 10.3712
##  * Protein log2 fold-changes
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.4621 -0.3273  0.1942  0.3344  0.7902 10.3712
```



3 Number of features

```
## Object of class "QcMetric"
##  Name: Number of features
##  Status: NA
##    PSM ModPep    Pep   Prot
##   1772   1522   1335    916
```



4 QC summary

	name	status
1	15N incorporation rate	TRUE
2	Log2 fold-changes	TRUE
3	Number of features	

5 Session information

- R version 3.6.0 (2019-04-26), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.46.0, Biobase 2.44.0, BiocGenerics 0.30.0, BiocStyle 2.12.0, IRanges 2.18.0, MAQCsubsetAFX 1.21.0, MSnbase 2.10.0, ProtGenerics 1.16.0, Rcpp 1.0.1, RforProteomics 1.21.0, S4Vectors 0.22.0, affy 1.62.0, gcrma 2.56.0, genefilter 1.66.0, ggplot2 3.1.1, knitr 1.22, mzR 2.18.0, qcmetrics 1.22.0, simpleaffy 2.60.0, xtable 1.8-4, yaqcaffy 1.44.0
- Loaded via a namespace (and not attached): BiocManager 1.30.4, BiocParallel 1.18.0, Biostrings 2.52.0, DBI 1.0.0, MALDIquant 1.19.2, MASS 7.3-51.4, Matrix 1.2-17, Nozzle.R1 1.1-1, R.methodsS3 1.7.1, R.oo 1.22.0, R.utils 2.8.0, R6 2.4.0, RBGL 1.60.0, RCurl 1.95-4.12, RSQLite 2.1.1, RUnit 0.4.32, XML 3.98-1.19, XVector 0.24.0, affyio 1.54.0, annotate 1.62.0, assertthat 0.2.1, biocViews 1.52.0, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, bookdown 0.9, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.0, crayon 1.3.4, digest 0.6.18, doParallel 1.0.14, dplyr 0.8.0.1, evaluate 0.13, foreach 1.4.4, glue 1.3.1, graph 1.62.0, grid 3.6.0, gtable 0.3.0, highr 0.8, htmltools 0.3.6, httpuv 1.5.1, impute 1.58.0, iterators 1.0.10, labeling 0.3, later 0.8.0, lattice 0.20-38, lazyeval 0.2.2, limma 3.40.0, magrittr 1.5, memoise 1.1.0, mime 0.6, munsell 0.5.0, mzID 1.22.0, ncdf4 1.16.1, pander 0.6.3, pcaMethods 1.76.0, pillar 1.3.1, pkgconfig 2.0.2, plyr 1.8.4, preprocessCore 1.46.0, promises 1.0.1, purrr 0.3.2, rlang 0.3.4, rmarkdown 1.12, scales 1.0.0, shiny 1.3.2, splines 3.6.0, stringi 1.4.3, stringr 1.4.0, survival 2.44-1.1, tibble 2.1.1, tidyselect 0.2.5, tools 3.6.0, vsn 3.52.0, withr 2.1.2, xfun 0.6, yaml 2.2.0, zlibbioc 1.30.0

4 Report generation

The report generation is handled by dedicated packages, in particular *knitr* [8] and *markdown* [9].

4.1 Custom reports

Templates

It is possible to customise reports for any of the existing types. The generation of the `pdf` report is based on a `tex` template, `knitr-template.Rnw`, that is available with the package[5]. The `qcReport` method accepts the path to a custom `template` as argument.

[5] You can find it with `system.file("templates", "knitr-template.Rnw", package = "qcmetrics")`.

The template corresponds to a LaTeX preamble with the inclusion of two variables that are passed to the `qcReport` and used to customise the template: the author's name and the title of the report. The former is defaulted to the system username with `Sys.getenv("USER")` and the later is a simple character. The `qcReport` function also automatically generates summary and session information sections. The core of the QC report, i.e the sections corresponding the the individual `QcMetric` instances bundled in a `QcMetrics` input (described in more details below) is then inserted into the template and weaved, or more specifically `knit`'ted into a `tex` document that is (if `type=pdf`) compiled into a `pdf` document.

The generation of the `html` report is enabled by the creation of a *R*markdown file (`Rmd`) that is then converted with *knitr* and *markdown* into `html`. The `Rmd` syntax being much simpler, no `Rmd` template is needed. It is possible to customise the final `html` output by providing a `css` definition as `template` argument when calling `qcReport`.

Initial support for the *Nozzle.R1* package [10] is available with type `nozzle`.

`QcMetric` sections

The generation of the sections for `QcMetric` instances is controlled by a function passed to the `qcto` argument. This function takes care of transforming an instance of class `QcMetric` into a `character` that can be inserted into the report. For the `tex` and `pdf` reports, `Qc2Tex` is used; the `Rmd` and `html` reports make use of `Qc2Rmd`. These functions take an instance of class `QcMetrics` and the index of the `QcMetric` to be converted.

```
qcmetrics:::Qc2Tex
```

```
## function (object, i)
## {
##     c(paste0("\\section{", name(object[[i]]), "}"), paste0("<<",
##         name(object[[i]]), ", echo=FALSE>>="), paste0("show(object[[",
##         i, "]])"), "@\n", "\\begin{figure}[!hbt]", "<<dev='pdf', echo=FALSE, fig.width=5
##         paste0("plot(object[[", i, "]])"), "@", "\\end{figure}",
##         "\\clearpage")
## }
## <bytecode: 0x555cec899a18>
## <environment: namespace:qcmetrics>
```

```
qcmetrics:::Qc2Tex(maqcm, 1)
```

```
##  [1] "\\section{Affy RNA degradation slopes}"
##  [2] "<<Affy RNA degradation slopes, echo=FALSE>>="
##  [3] "show(object[[1]])"
##  [4] "@\n"
##  [5] "\\begin{figure}[!hbt]"
##  [6] "<<dev='pdf', echo=FALSE, fig.width=5, fig.height=5, fig.align='center'>>="
##  [7] "plot(object[[1]])"
##  [8] "@"
##  [9] "\\end{figure}"
## [10] "\\clearpage"
```

Let's investigate how to customise these sections depending on the `QcMetric` status, the goal being to highlight positive QC results (i.e. when the status is `TRUE`) with ● (or ☺), negative results with ● (or ☹) and use ○ if status is `NA` after the section title[6].

Below, we see that different section headers are composed based on the value of `status(object[[i]])` by appending the appropriate LaTeX symbol.

[6] The respective symbols are `CIRCLE`, `smiley`, `frownie` and `Circle` from the LaTeX package `wasysym`.

```
Qc2Tex2
```

```
## function (object, i)
## {
##     nm <- name(object[[i]])
##     if (is.na(status(object[[i]]))) {
##         symb <- "$\\Circle$"
##     }
##     else if (status(object[[i]])) {
##         symb <- "{\\color{green} $\\CIRCLE$}"
```

```
##     }
##     else {
##         symb <- "{\\color{red} $\\CIRCLE$}"
##     }
##     sec <- paste0("\\section{", nm, "\\hspace{2mm}", symb, "}")
##     cont <- c(paste0("<<", name(object[[i]]), ", echo=FALSE>>="),
##         paste0("show(object[[", i, "]])"), "@\n", "\\begin{figure}[!hbt]",
##         "<<dev='pdf', echo=FALSE, fig.width=5, fig.height=5, fig.align='center'>>=",
##         paste0("plot(object[[", i, "]])"), "@", "\\end{figure}",
##         "\\clearpage")
##     c(sec, cont)
## }
## <bytecode: 0x555cdc1e7d78>
## <environment: namespace:qcmetrics>
```

To use this specific sectioning code, we pass our new function as `qcto` when generating the report. To generate smiley labels, use `Qc2Tex3`.

```
qcReport(maqcm, reportname = "rnadeg2", qcto = Qc2Tex2)
```

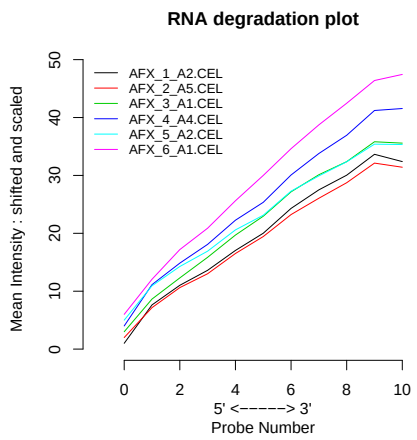
Quality control report generated with `qcmetrics`

biocbuild

May 2, 2019

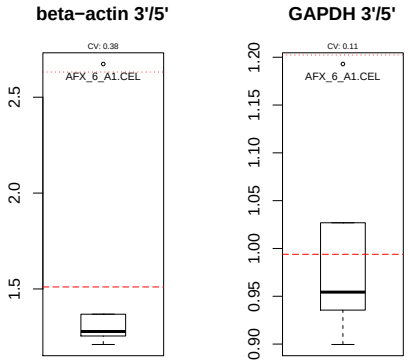
1 Affy RNA degradation slopes

```
## Object of class "QcMetric"
##  Name: Affy RNA degradation slopes
##  Status: TRUE
##  Data: deg
```



2 Affy RNA degradation ratios

```
## Object of class "QcMetric"
##  Name: Affy RNA degradation ratios
##  Status: FALSE
##  Data: yqc
```



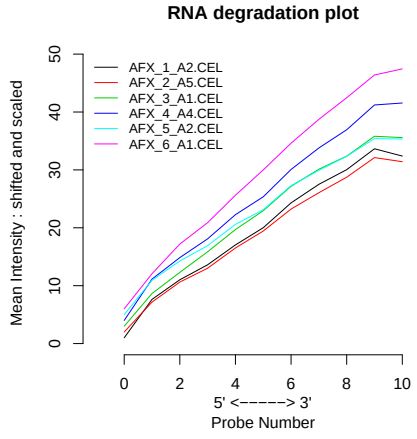
Quality control report generated with `qcmetrics`

biocbuild

May 2, 2019

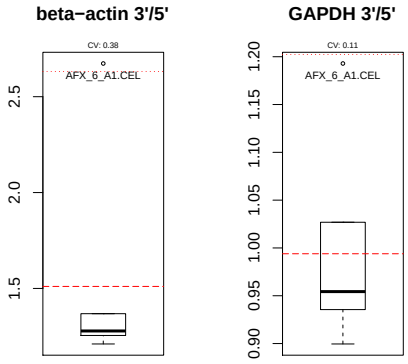
1 Affy RNA degradation slopes


```
## Object of class "QcMetric"
##  Name: Affy RNA degradation slopes
##  Status: TRUE
##  Data: deg
```

2 Affy RNA degradation ratios


```
## Object of class "QcMetric"
##  Name: Affy RNA degradation ratios
##  Status: FALSE
##  Data: yqc
```

4.2 New report types

A reporting function is a function that

- Converts the appropriate QC item sections (for example the `Qc2Tex2` function described above)
- Optionally includes the QC item sections into addition header and footer, either by writing these directly or by inserting the sections into an appropriate template. The reporting functions that are available in *qcmetrics* can be found in `?qcReport`: `reporting_tex` for type `tex`, `reporting_pdf` for type `pdf`, . . . These functions should use the same arguments as `qcReport` insofar as possible.
- Once written to a report source file, the final report type is generated. `knit` is used to convert the `Rnw` source to `tex` which is compiled into `pdf` using `tools::texi2pdf`. The `Rmd` content is directly written into a file which is knitted and converted to `html` using `knit2html` (which call `markdownTOHTML`).

New `reporting_abc` functions can be called directly or passed to `qcReport` using the `reporter` argument.

5 QC packages

5.1 A simple RNA degradation package

While the examples presented in section 3 and in particular the wrapper function in section 3.2 are flexible and fast ways to design QC pipeline prototypes, a more robust mechanism is desirable for production pipelines. The *R*packaging mechanism is ideally suited for this as it provides versioning, documentation, unit testing and easy distribution and installation facilities.

While the detailed description of package development is out of the scope of this document, it is of interest to provide an overview of the development of a QC package. Taking the wrapper function, it could be used the create the package structure

```
package.skeleton("RnaDegQC", list = "rnadeg")
```

The `DESCRIPTION` file would need to be updated. The packages *qcmetrics*, *affy* and *yaqcaffy* would need to be specified as dependencies in the `Imports:` line and imported in the `NAMESPACE` file. The documentation file `RnaDegQC/man/rnadeg.Rd` and the (optional) `RnaDegQC/man/RnaDegQC-packge.Rd` would need to be updated.

Alternatively, the `rnadeg` function could be modularised so that QC items would be created and returned by dedicated constructors like `makeRnaDegSlopes` and `makeRnaDegRatios`. This would provide other developers with the means to re-use some components of the pipeline by using the package.

5.2 A QC pipeline repository

The wiki on the *qcmetrics* github page[7] can be edited by any github user and will be used to cite, document and share QC functions, pipelines and packages, in particular those that make use of the *qcmetrics* infrastructure.

[7] https://github.com/lgatto/qcmetrics

6 Conclusions

*R*and Bioconductor are well suited for the analysis of high throughput biology data. They provide first class statistical routines, excellent graph capabilities and an interface of choice to import and manipulate various omics data, as demonstrated by the wealth of packages[8] that provide functionalities for QC.

[8] http://bioconductor.org/packages/release/BiocViews.html#____QualityControl

The *qcmetrics* package is different than existing *R*packages and QC systems in general. It proposes a unique domain-independent framework to design QC pipelines and is thus suited for any use case. The examples presented in this document illustrated the application of *qcmetrics* on data containing single or multiple samples or experimental runs from different technologies. It is also possible to automate the generation of QC metrics for a set of repeated (and growing) analyses of standard samples to establish *lab memory* types of QC reports, that track a set of metrics for controlled standard samples over time. It can be applied to raw data or processed data and tailored to suite precise needs. The popularisation of integrative approaches that combine multiple types of data in novel ways stresses out the need for flexible QC development.

qcmetrics is a versatile software that allows rapid and easy QC pipeline prototyping and development and supports straightforward migration to production level systems through its well defined packaging mechanism.

Acknowledgements

Many thanks to Arnoud Groen for providing the ^{15}N data and Andrzej Oles for helpful comments and suggestions about the package and this document.

Session information

All software and respective versions used to produce this document are listed below.

- R version 3.6.0 (2019-04-26), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.46.0, Biobase 2.44.0, BiocGenerics 0.30.0, BiocStyle 2.12.0, IRanges 2.18.0, MAQCsubsetAFX 1.21.0, MSnbase 2.10.0, ProtGenerics 1.16.0, Rcpp 1.0.1, RforProteomics 1.21.0, S4Vectors 0.22.0, affy 1.62.0, gcrma 2.56.0, genefilter 1.66.0, ggplot2 3.1.1, knitr 1.22, mzR 2.18.0, qcmetrics 1.22.0, simpleaffy 2.60.0, xtable 1.8-4, yaqcaffy 1.44.0
- Loaded via a namespace (and not attached): BiocManager 1.30.4, BiocParallel 1.18.0, Biostrings 2.52.0, DBI 1.0.0, MALDIquant 1.19.2, MASS 7.3-51.4, Matrix 1.2-17, Nozzle.R1 1.1-1, R.methodsS3 1.7.1, R.oo 1.22.0, R.utils 2.8.0, R6 2.4.0, RBGL 1.60.0, RCurl 1.95-4.12, RSQLite 2.1.1, RUnit 0.4.32, XML 3.98-1.19, XVector 0.24.0, affyio 1.54.0, annotate 1.62.0, assertthat 0.2.1, biocViews 1.52.0, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, bookdown 0.9, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.0, crayon 1.3.4, digest 0.6.18, doParallel 1.0.14, dplyr 0.8.0.1, evaluate 0.13,

foreach 1.4.4, glue 1.3.1, graph 1.62.0, grid 3.6.0, gtable 0.3.0, highr 0.8, htmltools 0.3.6, httpuv 1.5.1, impute 1.58.0, iterators 1.0.10, labeling 0.3, later 0.8.0, lattice 0.20-38, lazyeval 0.2.2, limma 3.40.0, magrittr 1.5, memoise 1.1.0, mime 0.6, munsell 0.5.0, mzID 1.22.0, ncdf4 1.16.1, pander 0.6.3, pcaMethods 1.76.0, pillar 1.3.1, pkgconfig 2.0.2, plyr 1.8.4, preprocessCore 1.46.0, promises 1.0.1, purrr 0.3.2, rlang 0.3.4, rmarkdown 1.12, scales 1.0.0, shiny 1.3.2, splines 3.6.0, stringi 1.4.3, stringr 1.4.0, survival 2.44-1.1, tibble 2.1.1, tidyselect 0.2.5, tools 3.6.0, vsn 3.52.0, withr 2.1.2, xfun 0.6, yaml 2.2.0, zlibbioc 1.30.0

References

[1] L Gautier, L Cope, B M Bolstad, and R A Irizarry. affy – analysis of affymetrix genechip data at the probe level. *Bioinformatics*, 20(3):307–315, 2004. doi:http://dx.doi.org/10.1093/bioinformatics/btg405.

[2] L Gatto. *yaqcaffy: Affymetrix expression data quality control and reproducibility analysis*. R package version 1.21.0.

[3] P G A Pedrioli et al. A common open representation of mass spectrometry data and its application to proteomics research. *Nat. Biotechnol.*, 22(11):1459–66, 2004. doi:10.1038/nbt1031.

[4] L Gatto and K S Lilley. MSnbase – an R/Bioconductor package for isobaric tagged mass spectrometry data visualization, processing and quantitation. *Bioinformatics*, 28(2):288–9, Jan 2012. doi:10.1093/bioinformatics/btr645.

[5] K M Foster, S Degroeve, L Gatto, M Visser, R Wang, K Griss, R Apweiler, and L Martens. A posteriori quality control for the curation and reuse of public proteomics data. *Proteomics*, 11(11):2182–94, 2011. doi:10.1002/pmic.201000602.

[6] J Krijgsveld, R F Ketting, T Mahmoudi, J Johansen, M Artal-Sanz, C P Verrijzer, R H Plasterk, and A J Heck. Metabolic labeling of c. elegans and d. melanogaster for quantitative proteomics. *Nat Biotechnol*, 21(8):927–31, Aug 2003. doi:10.1038/nbt848.

[7] A Groen, L Thomas, K Lilley, and C Marondedze. Identification and quantitation of signal molecule-dependent protein phosphorylation. *Methods Mol Biol*, 1016:121–37, 2013. doi:10.1007/978-1-62703-441-8_9.

[8] Y Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2013. ISBN 978-1482203530. URL: http://yihui.name/knitr/.

[9] JJ Allaire, J Horner, V Marti, and N Porte. *markdown: Markdown rendering for R*, 2013. R package version 0.6.3. URL: http://CRAN.R-project.org/package=markdown.

[10] N Gehlenborg. *Nozzle.R1: Nozzle Reports*, 2013. R package version 1.1-1. URL: http://CRAN.R-project.org/package=Nozzle.R1.