NacoStringQCPro

Dorothee Nickles, Thomas Sandmann, Robert Ziman, Richard Bourgon

Modified: April 10, 2017. Compiled: April 24, 2017

Contents

1 Scope

The *NanoStringQCPro* package facilitates preprocessing and quality control of NanoString® gene expression data. It provides functions for creating an *RccSet* (an extension of the standard *ExpressionSet* class) from the files produced by an nCounter® mRNA Gene Expression assay, performing NanoString®-recommended background correction and normalization, and generating a QC report with metrics that can be used to identify outlier samples and poorly performing probes (i.e. probes with signals within noise range). The overall workflow is outlined in Figure 1 (see next page).

2 Quick start

The package's `newRccSet()` function is the main wrapper function for reading in nCounter® mRNA Gene Expression count data and annotations:

```
> library(NanoStringQCPro)

> exampleDataDir <- system.file("extdata", package="NanoStringQCPro")
> rccDir <- file.path(exampleDataDir, "RCC")
> example_rccSet <- newRccSet(
+      rccFiles               = dir(rccDir, full.names=TRUE)
+    #,rccCollectorToolExport = file.path(exampleDataDir, "nSolver", "RCC_collector_tool_export.csv")
+     ,rlf                    = file.path(exampleDataDir, "RLF", "NQCP_example.rlf")
+     ,cdrDesignData          = file.path(exampleDataDir, "CDR", "CDR-DesignData.csv")
+     ,extraPdata             = file.path(exampleDataDir, "extraPdata", "SampleType.txt")
+     ,blankLabel             = "blank"
+     ,experimentData.name    = "Dorothee Nickles"
+     ,experimentData.lab     = "Richard Bourgon"
+     ,experimentData.contact = "nickles.dorothee@gene.com"
+     ,experimentData.title   = "NanoStringQCPro example dataset"
+     ,experimentData.abstract= "Example data for the NanoStringQCPro package"
+ )
>
> # Reading RCC files...
> # checkRccSet() messages:
> #   The following panel housekeeping genes were found: RBCK1, USP19
```

Raw count data is stored in a series of Reporter Code Count (`.RCC`) files where each file holds the data for one sample. The `rccFiles` argument should be assigned a vector of paths to these files. Alternatively, the `.CSV` file produced using the RCC Collector Tool Format Export feature of NanoString® nSolver™ Analysis Software can be imported instead via the `rccCollectorToolExport` argument (commented out in the example above), but we recommend importing the `.RCC` files directly if they are available. Details for each probe in the codeset used

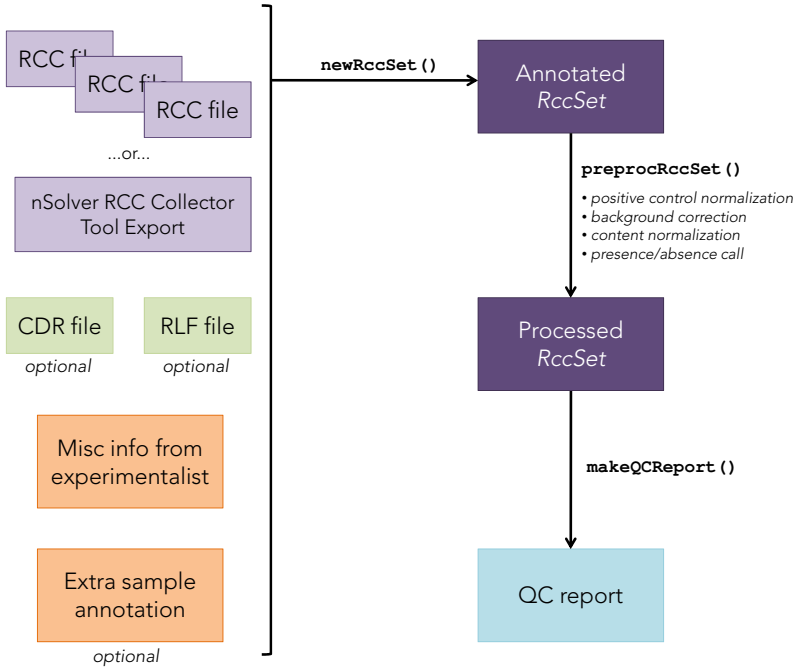

Figure 1: Overview of the *NanoStringQCPro* work flow.

in the experiment are specified in a Reporter Library File (`.RLF`): a path to this file must always be specified via the `rlf` argument. Optional additional details for each probe are usually provided by NanoString® in the "Design Data" tab of the Codeset Design Report (CDR) — an Excel spreadsheet that accompanies each codeset order. To import these details, an extract of the "Design Data" tab should be saved as a `.CSV` file (see `extdata/CDR` for an example) and then its path should be specified in the `cdrDesignData` argument.

The function has a few optional arguments that are described in more detail in its man page and in **Sample annotation** and **Codeset annotation** in the following section. For examples of the various input files, see the `extdata` directory included with the package and the **Example files** section at the end of this vignette.

3 RccSet loaded with NanoString® data

The object returned by `newRccSet()` is an *RccSet*. This class is a light extension of the standard *ExpressionSet* class, and it enables *NanoStringQCPro* to more rigorously make use of the base *ExpressionSet* data structure in a manner suitable for the preprocessing and quality control of NanoString® data.

```
> str(max.level=2, example_rccSet)

Formal class 'RccSet' [package "NanoStringQCPro"] with 7 slots
  ..@ experimentData   :Formal class 'MIAME' [package "Biobase"] with 13 slots
  ..@ assayData        :<environment: 0x7fdcb4258640>
  ..@ phenoData        :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
  ..@ featureData      :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
  ..@ annotation       : chr "NQCP_example"
  ..@ protocolData     :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
  ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slot
```

The `experimentData` slot contains information about the experiment: its title and an abstract, and the name, lab, and contact information of the people who generated the data. `assayData` is a pointer to an environment that contains a matrix called `exprs` which initially contains the raw expression data imported from the `.RCC` files. (Note: a pseudo-count of 1 is always added to all measurements to enable subsequent log transformation of the data in cases where zero-counts are present.) After preprocessing, this matrix will contain background-corrected and normalized data and there will be additional matrices in `assayData` for each preprocessing step — see sections below. `phenoData` is an *AnnotatedDataFrame* with information about the samples, and `featureData` is a similar *AnnotatedDataFrame* with information about the probes. The elements of `phenoData` correspond to the columns of the `exprs` matrix while those of `featureData` correspond to its rows. The `annotation` slot shows the name of the codeset. The `protocolData` slot is currently unused.

Note that *Biobase* provides accessor functions for most key content: `experimentData()`, `assayData()`, `exprs()`, `phenoData()` (or `pData()` for the same content in a standard data frame), `featureData()` (or `fData()`), and `annotation()`. For more information about any of these functions or the *ExpressionSet* class overall, see the *Biobase* documentation. For more information on the additional methods provided by the *RccSet* class, see the package's man pages.

3.1 Sample annotation

Any descriptions or details about the samples that are already present in the `.RCC` files will be extracted by `newRccSet()` and stored in the `phenoData` slot. Such details are often limited and insufficient for downstream work, so additional sample annotation can be added via `.CSV` files specified in a vector of paths passed to the function's `extraPdata` argument. Each of these annotation files should be comma-separated and should

contain a column labeled `FileName` whose values are the exact `.RCC` file names. The `FileName` column is used as the primary key for merging the annotation columns into `phenoData`, and the function will return an error if it contains any missing or additional `.RCC` filenames. One particularly important bit of sample information that can be added via one of these annotation files is the `SampleType` column used primarily by the preprocessing and QC report functions to identify blank samples (i.e. water runs). If this column isn't specified as such, it will be assigned with default value `NA` for all samples. (Note also that the '`blankLabel`' argument to `newRccSet()` should be the same as the value used in the annotation file.)

3.2 Codeset annotation

Three feature annotation columns (`CodeClass`, `GeneName`, `Accession`) are extracted from the `.RCC` files and stored in the `featureData` slot. Their concatenation ("`<CodeClass>_<GeneName>_<Accession>`") is used for the `featureData` row names and thus serves as a primary key for features; probe details in the `.RLF` file are merged into `featureData` using this key. An additional column named `SpikeInInput` is added to record the RNA "spike in" input concentrations for the control probes in each codeset. These probes are identified by "Positive" and "Negative" values in the `CodeClass` column, and their concentrations are parsed from the parenthesized label in the `GeneName` (128, 32, 8, 2, 0.5, and 0.125 fM for positive control probes and 0 for all negative control probes in the mRNA Gene Expression assay). If `addEgAnnotions == TRUE`, then `featureData` is further annotated with EntrezGene identifiers and HGNC symbols by doing look ups in the *org.Hs.eg.db* package using the RefSeq accessions indicated in the `.RLF`. Note that `NA` values will be assigned as the annotations for any accessions not found in *org.Hs.eg.db*, and the values for other accessions will depend on the specific version of *org.Hs.eg.db* that you have installed.

The CDR for a given codeset is provided as an Excel file with multiple tabs containing information about the probe design history and the overall order. For the annotation of a codeset, only the "Design Data" tab is relevant: it has information such as probe identifiers and target sequences for all probes in the codeset. If the CDR is available, the "Design Data" tab can be saved as a comma delimited `.CSV` file, and the `cdrDesignData` argument of `newRccSet()` can be pointed at this file. The function will then add the relevant columns from the file to `featureData`. See the `buildCodesetAnnotation()` man page for more details.

4 Preprocessing

NanoString® proposes several data preprocessing steps for NanoString® gene expression data:

- **Positive control normalization**. Positive control normalization can be used to adjust for all platform-associated sources of variation (e.g., automated purification, hybridization conditions, etc.). This type of normalization will not, however, account for differences in sample input — which is also an important source of systematic variation.
- **Background correction**. Background signal estimates — ideally, probe-specific estimates — are the basis for determining whether or not a specific transcript was detected in a given sample. In addition, background subtraction improves accuracy of fold change estimation across samples, especially for targets whose signal is above but still close to the background level. Global background signal estimates can be computed using negative controls. Probe-specific background signal estimates require running a small number of blanks in addition to the primary samples of interest (three or more blanks is recommended).
- **RNA content normalization**. To account for inevitable differences in total RNA input per sample, an additional step beyond positive control normalization is required. One such approach uses scaling to

make the average signal for so-called "housekeeping" genes the same across all samples. Other non-linear approaches exist as well, but those in common use for microarray or whole-transcriptome RNA-seq data typically assume that most targets show no differential expression between conditions. For NanoString® panels that include target sets that are highly focused on particular pathways or processes, this is often not a reasonable assumption, and as a consequence such approaches may not be appropriate.

For most data sets, we typically use the following approach:

(i) Positive control normalization, to first reduce technical variability.
(ii) Subtraction of background estimates. Our preferred method (described in more detail below) is identical to that of NanoString®, and it makes use of both blank samples and negative control probes. The method smoothly combines blanks and negative controls — relying wholly on the latter if no blanks are available, but largely ignoring negative controls as the number of blanks increases. We have observed strong and systematic differences in probe-specific background, and we therefore recommend including enough blanks to make the negative control impact on background estimation minor.
(iii) RNA content normalization via housekeeping or global median scaling. Note that if background subtraction is omitted, there is no need to do both positive control and RNA content normalization: the latter will override the former.

The `preprocRccSet()` function can be used to perform all recommended preprocessing steps (or any combination of them) in one go. It takes an *RccSet* and the preprocessing parameters as input, and it returns a new *RccSet* with `exprs` containing the final result of the preprocessing. Results for intermediate preprocessing steps are included in additional matrices in `assayData`, and the settings for each preprocessing step are recorded in correspondingly named elements in the `preprocessing` slot of `experimentData` (accessible via the `preproc()` function provided by *Biobase*). **Note:** Most downstream analysis is best performed on a logarithmic scale, so in the output's `assayData`, the `normData` matrix is on the log2 scale; all other matrices in `assayData` are on the natural (original) scale.

```
> norm_example_rccSet <- preprocRccSet(rccSet = example_rccSet, normMethod = "housekeeping")
> # Warning message:
> # In .local(rccSet, ...) : Less than three housekeeping features are defined
>
> ls(assayData(norm_example_rccSet))

[1] "bgCorrData"  "bgEstimates" "exprs"       "normData"    "paData"
[6] "posCtrlData"

> preproc(norm_example_rccSet)

$org.Hs.eg.db_version
[1] NA

$posCtrlData_summaryFunction
[1] "sum"

$bgEstimates_bgReference
[1] "both"

$bgEstimates_summaryFunction
[1] "median"

$bgEstimates_stringency
[1] 1

$bgEstimates_nSolverBackground.w1
[1] 2.18
```

```
$bgEstimates_nSolverBackground.shrink
[1] TRUE

$bgEstimates_inputMatrix
[1] "posCtrlData"

$paData_stringency
[1] 2

$paData_inputMatrix
[1] "posCtrlData"

$normData_method
[1] "housekeeping"

$normData_summaryFunction
[1] "median"

$normData_hkgenes
[1] "RBCK1" "USP19"

$normData_hkfeatures
[1] "Housekeeping_RBCK1_NM_006462"    "Housekeeping_USP19_XP_005264882"

$normData_inputMatrix
[1] "bgCorrData"
```

The following subsections further describe each individual preprocessing step.

4.1 Positive control normalization

The positive spike-in RNA hybridization controls for each lane may be used to estimate the overall efficiency of hybridization and recovery for each lane. To do so, a lane-specific summary (e.g., sum or, equivalently, average) of positive control counts is first calculated. The average of these per-lane values is then used as the target, and all counts for each lane are then scaled so that all lane-specific positive control summaries match the target.

The `posCtrlNorm()` function performs positive control normalization and stores the results in a matrix called `posCtrlData` in the `assayData` slot. (**Note:** this differs from the behavior of other packages dealing with *ExpressionSet* objects that modify the values in `exprs`. The implementation was chosen in order to handle certain cases where the QC report needs to perform its own preprocessing on a given *RccSet*.) You can choose mean, median, or sum (the default) as the summary of positive control counts. NanoString® advises caution with interpreting results of samples with positive control scaling factors outside the range of 0.3 to 3 since values outside this range may indicate significant under-performance of the respective lanes. A plot highlighting any outliers is provided in the QC report, and `posCtrlNorm()` records the positive control scaling factors in a column named `PosFactor` in the output's `phenoData` for additional inspection.

```
> adj_example_rccSet <- posCtrlNorm(example_rccSet, summaryFunction="sum")
> ls(assayData(adj_example_rccSet))

[1] "exprs"       "posCtrlData"

> preproc(adj_example_rccSet)
```

```
$org.Hs.eg.db_version
[1] NA

$posCtrlData_summaryFunction
[1] "sum"
```

```
> head(pData(adj_example_rccSet)$PosFactor)
```

```
[1] 1.1336986 0.8377521 0.7364953 0.8895509 0.8347573 1.1523211
```

4.2 Background correction

As mentioned above, NanoString® includes several probes in each codeset for which no target is expected to be present. These negative controls can be used to produce global estimates for each lane. Importantly, such global estimates will not capture probe-specific differences in background — which we have seen to often be quite large. The *NanoStringQCPro* package offers three ways to estimate the non-specific noise in a NanoString® gene expression experiment, two of which are probe specific:

- **Lane-sepecific background**, based on signals obtained for negative controls in each lane.
- **Probe-specific background**, based on signals obtained for each probe in blank measurements without RNA present.
- **Combined probe- and lane-specific background**.

A few examples are shown below using the `getBackground()` function. The algorithm that mimics the calculation used in NanoString® nSolver™ Analysis Software takes effect when the option to do combined probe- and lane-specific background is selected, and a full description of the algorithm and its parameters can be found in the man page for the `nSolverBackground()` function in this package.

Note: for datasets that do not contain blanks, use `bgReference="negatives"` or `preproRccSet()` will throw an error.

```
> # Get background based on median signal for each probe in blank measurements:
> bg1 <- getBackground(adj_example_rccSet, bgReference="blanks", summaryFunction="median")
> # ...based on mean of negative controls:
> bg2 <- getBackground(adj_example_rccSet, bgReference="negatives", summaryFunction="mean")
> # ...using an implementation mimicking that of the nSolver software:
> bg3 <- getBackground(adj_example_rccSet, bgReference="both", stringency=1)
```

Background signal can then be subtracted from the data by calling `subtractBackground()` on the background estimates matrix as obtained above.

```
> bgcorr_example_rccSet <- subtractBackground(adj_example_rccSet, bgEstimates=bg1)
> ls(assayData(bgcorr_example_rccSet))
```

```
[1] "bgCorrData"  "bgEstimates" "exprs"       "posCtrlData"
```

```
> preproc(bgcorr_example_rccSet)
```

```
$org.Hs.eg.db_version
[1] NA

$posCtrlData_summaryFunction
[1] "sum"

$bgEstimates_bgReference
[1] NA
```

```
$bgEstimates_summaryFunction
[1] NA

$bgEstimates_stringency
[1] NA

$bgEstimates_nSolverBackground.w1
[1] NA

$bgEstimates_nSolverBackground.shrink
[1] NA

$bgEstimates_inputMatrix
[1] NA
```

4.3 RNA content normalization

NanoStringQCPro offers two ways to normalize the NanoString® count data to account for differences in input RNA content:

- **Median normalization.** Median expression values across all probes are equated (typically after positive control normalization and background correction have already been applied).
- **Housekeeping normalization.** Expression values for each probe are scaled relative the average signal from a small number of pre-specified genes "housekeeping" genes.

Median normalization is only recommended if a large number of genes are assayed in the experiment (NanoString® recommends >300 probes) and most of them are not expected to change dramatically across conditions. Note that in this setting, other non-linear normalization techniques traditionally applied to microarray or RNA-seq data — e.g., variance stabilizing transforms or quantile normalization — may also be appropriate. (These are not currently implemented in *NanoStringQCPro*, but relevant code in other Bioconductor packages can be applied directly to the matrices in `assayData`.)

Housekeeping normalization is recommended if a reasonable number of low-variance features can be identified for the experiment. As in PCR experiments, the housekeeping genes should be carefully evaluated as part of the codeset design process. NanoString® recommends using at least 3 housekeeping features, and ideally even more. If housekeeping normalization is selected, the normalization function will by default use the panel-specified housekeeping features (i.e., those whose `CodeClass` is recorded as "`Housekeeping`" in the `.RCC` and `.RLF` files), but these can be overridden by passing in a different list of feature identifiers or gene symbols. The *RccSet* output by the normalization function will have a `featureData` column named '`Housekeeping`' that indicates which features were used for the normalization.

Note: Final normalized data are on the log2 scale, unlike all other matrices in `assayData`.

```
> gmnorm_example_rccSet <- contentNorm(rccSet=bgcorr_example_rccSet, method="median")
> hknorm_example_rccSet <- contentNorm(rccSet=bgcorr_example_rccSet, method="housekeeping")
>
> # Warning message:
> # In contentNorm(srccSet, method = "housekeeping", hk = hk) :
> #   Less than three housekeeping features are defined
```

5 QC report and quality control of NanoString® expression data

For quality control of NanoString® expression data, *NanoStringQCPro* provides a single function, `makeQCReport()`, that generates a QC report in HTML format:

```
> qc_example_rccSet <- makeQCReport(norm_example_rccSet, "example_QC_report")
>
> # Generating QC report...
> # Report file has been generated: /gne/home/richarbo/tmp/example_QC_report.html
```

The report includes detailed descriptions of the different quality control steps it performs and metrics it uses. Please see the example report generated above for full detail (a copy of the report can be found in `extdata/example_output`). In addition to generating the report, `makeQCReport()` also generates additional QC files:

- `SampleFlags.txt`: a tab-delimited file listing all samples along with their sample identifiers and sample types. Three `TRUE/FALSE` columns specify QC flags for each sample: one flag based on technical assay run information (`TechnicalFlags`), one based on QC steps using signals obtained for positive and negative controls (`ControlFlags`), and one based on QC steps using signals obtained for endogenous genes (`CountFlags`).
- `HousekeepingGeneStats.txt`: a tab-delimited file providing some metrics on the housekeeping genes. For each gene, values are provided for the variance, inter-quartile range (IQR), median expression, and correlation (cor) with all other housekeeping genes across all samples. (Note that this file will only be generated if housekeeping genes are indicated in the input via `CodeClass` "`Housekeeping`" or if housekeeping normalization has been evinced by the `featureData` `Housekeeping` column.)

6 Example files

The package comes with the following example files corresponding to the example discussed throughout the vignette:

- `extdata/RCC` contains `.RCC` files as they would be produced by the nCounter® instrument.
- `extdata/nSolver/RCC_collector_tool_export.csv` contains a `.CSV` file as would be generated by the RCC Collector Tool Format Export feature of the NanoString® nSolver™ Analysis Software. (This file contains the same data as the `.RCC` files.)
- `extdata/RLF` has the `.RLF` annotation corresponding to the codeset used in the example.
- `extdata/CDR` has the CDR "Design Data" annotation corresponding to the codeset used in the example.
- `extdata/extraPdata` has additional sample annotation passed to the `extraPdata` argument of `newRccSet()`.
- `examples` contains `.R` scripts illustrating the overall workflow.

7 Session info

```
> sessionInfo()

R version 3.4.0 (2017-04-21)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: OS X El Capitan 10.11.6
```

```
Matrix products: default
BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] NanoStringQCPro_1.8.0 bigmemory_4.5.19      bigmemory.sri_0.1.3
[4] Biobase_2.36.0        BiocGenerics_0.22.0

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.10         highr_0.6            plyr_1.8.4
 [4] compiler_3.4.0       RColorBrewer_1.1-2   iterators_1.0.8
 [7] tools_3.4.0          rngtools_1.2.4       digest_0.6.12
[10] tibble_1.3.0         RSQLite_1.1-2        evaluate_0.10
[13] memoise_1.1.0        gtable_0.2.0         gridBase_0.4-7
[16] NMF_0.20.6           png_0.1-7            foreach_1.4.3
[19] DBI_0.6-1            registry_0.3         yaml_2.1.14
[22] stringr_1.2.0        pkgmaker_0.22        knitr_1.15.1
[25] cluster_2.0.6        S4Vectors_0.14.0     IRanges_2.10.0
[28] stats4_3.4.0         rprojroot_1.2        grid_3.4.0
[31] AnnotationDbi_1.38.0 rmarkdown_1.4        reshape2_1.4.2
[34] ggplot2_2.2.1        org.Hs.eg.db_3.4.1   magrittr_1.5
[37] backports_1.0.5      scales_0.4.1         codetools_0.2-15
[40] htmltools_0.3.5      BiocStyle_2.4.0      xtable_1.8-2
[43] colorspace_1.3-2     stringi_1.1.5        lazyeval_0.2.0
[46] munsell_0.4.3        doParallel_1.0.10    markdown_0.8
```