Frozen Robust Multi-Array Analysis and the Gene Expression Barcode

Matthew N. McCall

May 3, 2016

Contents

The frma implements methods for the preprocessing (frma) and analysis (barcode) of single microarrays. The fRMA algorithm allows the user to preprocess arrays individually while retaining the advantages of multi-array preprocessing methods such as RMA. The Gene Expression Barcode provides estimates of absolute expression rather than relative expression.

1 Frozen Robust Multiarray Analysis (fRMA)

Frozen RMA (fRMA) is a microarray preprocessing algorithm that allows one to analyze microarrays individually or in small batches and then combine the data for analysis. This is accomplished by utilizing information from the large publicly available microarray databases. Specifically, estimates of probe-specific effects and variances are precomputed and frozen. Then, with new data sets, these frozen parameters are used in concert with information from the new array(s) to preprocess the data.

The fRMA is particularly useful when it is not feasible to preprocess all of the data simultaneously. Such situations often arise when:

- large meta-analyses require one to preprocess more data than can be stored in memory,
- datasets grow incrementally and it would be laborious to preprocess all of the data each time a new array is added,

- microarrays are used to aid in clinical diagnosis and treatment, one needs to obtain information based on a single sample hybridized to a single microarray.

Additionally, fRMA down-weights probes that appear to be *batchy*, those that are most susceptible to batch-effects.

1.1 From CEL files to expression estimates

For Affymetrix microarray data, it is customary to view CEL files as the starting point for preprocessing and analysis. One or more CEL files can be read into **R** using the `ReadAffy` function from the affy package to produce an `AffyBatch` object or using the `read.celfiles` function from the oligo to produce a `FeatureSet`.

The goal of fRMA is to obtain reliable gene-level intensities from the raw microarray data. This amounts to converting raw probe-level intensities into background-corrected, normalized, and summarized gene-level intensities. The `frma` function takes an `AffyBatch` or `FeatureSet` as input and produces an object containing gene-level expression values. This object can take one of two forms, an `ExpressionSet` or a `frmaExpressionSet`, depending upon the additional information that is requested. The gene-level intensities stored in both these objects can be accessed using the `exprs` method.

In addition to the raw data, the fRMA algorithm requires a number of *frozen* parameter vectors. Among these are the reference distribution to which the data are normalized and the probe-effect estimates. We have computed these frozen parameters for many popular Affymetrix platforms. The data for each of these platforms is stored in an **R** package of the form <platform>frmavecs. By default, the `frma` function attempts to load the appropriate data package for the input data object.

The remainder of this section describes typical use of the `frma`. For details of the statistical methodology implemented in the frma function, please read the following papers:

McCall MN, Bolstad BM, and Irizarry RA (2010). Frozen Robust Multi-Array Analysis (fRMA), Biostatistics, 11(2):242-253.

McCall MN, Jaffee HA, Irizarry RA (2012). fRMA ST: Frozen robust multiarray analysis for Affymetrix Exon and Gene ST arrays, Bioinformatics, 28(23):3153-3154.

The following is a simple description of how to preprocess a single CEL file using the default version of frma:

1. Download and install frma and the appropriate frozen parameter package (i.e. hgu133afrmavecs, hgu133plus2frmavecs, etc.).

2. Load the frma package.

```
> library(frma)
```

3. Read in the data as either an `AffyBatch` or `FeatureSet` object. For the early Affymetrix platforms (e.g. HGU133plus2), you should use the `ReadAffy` function from the affy package to read in the CEL files. For the later Affymetrix platforms (e.g. Gene ST or Exon ST), you should use the `read.celfiles` function from the oligo package.

4. In this vignette we will load an example `AffyBatch` object:

```
> library(frmaExampleData)
> data(AffyBatchExample)
```

5. Preprocess the raw data object using the default version of frma.

```
> object <- frma(AffyBatchExample)
```

The final `object` will be an `ExpressionSet` or `frmaExpressionSet`. The latter is an extension of the ExpressionSet class to hold additional information related to the preprocessing procedure such as weights and residuals. To obtain a matrix of gene-level expression values, enter the following command:

```
> e <- exprs(object)
```

To assess the quality of the expression estimates, one can use the `GNUSE` function:

```
> GNUSE(object,type="stats")
```

```
       GSM391693.CEL.gz GSM391694.CEL.gz GSM391695.CEL.gz
median        1.0827271        1.1134342        1.1636686
IQR           0.2921626        0.2989743        0.3332328
95%           1.5780362        1.5958426        1.7124421
99%           1.9868534        2.0011816        2.0978847
```

The GNUSE is a modification of the NUSE quality metric that allows one to assess the quality of individual samples relative to the large training data set used to compute the frozen parameter vectors. The GNUSE is described in more detail in the following paper:

McCall MN, Murakami PN, Lukk M, Huber W, Irizarry RA (2011). Assessing Affymetrix GeneChip Microarray Quality, BMC Bioinformatics, 12:137.

Depending on the number of CEL files and on the memory available to your system, you might experience errors like 'Cannot allocate vector ...'. An obvious option is to increase the memory available to your R process (by adding memory and/or closing external applications). You might also consider analyzing the data in smaller batches. Recall that fRMA allows you to preprocess data separately and then combine the preprocessed data for further analysis.

1.2 Advanced Options

The default arguments to the `frma` function will be sufficient for most users; however, additional options have been implemented to allow the user to control each stage of the preprocessing, as well as, the information returned by the `frma` function. This flexibility is instrumental in allowing users to easily explore alternative methods of preprocessing.

1.2.1 Summarization Methods

Summarization refers to the method used to combine probe-level expression values to obtain gene-level expression estimates. There are several summarization methods that one can choose from when running frma. A brief description of each of the methods follow:

- **average:** subtract the probe-effect and then compute the mean of the probes in each probeset.
- **median:** subtract the probe-effect and compute the median of the probes in each probeset.
- **median_polish:** this is the same as the median summarization because the probe-effects have already been removed.
- **weighted average:** compute a weighted average of the probes in each probeset with weights equal to the inverse of the sum of the precomputed within and between batch variance estimates.
- **robust_weighted_ average:** compute a weighted average of the probes in each probeset with weights equal to the weights returned by an M-estimation procedure divided by the sum of the precomputed within and between batch variance estimates.
- **random_ effect:** the robust weighted average method adapted for a batch of new arrays (see the fRMA paper for details).

1.2.2 Input Vectors

While the vast majority of users will use the precomputed vectors provided in the frmavecs packages, the `frma` function will accept user-supplied frozen parameter vectors. The frmaTools package contains functions to create your own frozen parameter vectors. There are several situations in which creating your own frozen parameter vectors may be beneficial. These are described in detail in:

McCall MN and Irizarry RA (2011). Thawing Frozen Robust Multi-array Analysis (fRMA), BMC Bioinformatics, 12:369.

If you create your own frozen parameter vectors using functions in the frmaTools package, the vectors will already be in the correct format: a list with elements normVec, probeVec, probeVarBetween, probeVarWithin, probesetSD, and medianSE. A description of each of these elements follows:

- **normVec:** a vector containing values of the reference distribution to which samples will be quantile normalized
- **probeVec:** a vector of probe-effect estimates
- **probeVarBetween:** a vector of the between batch variance for each probe
- **probeVarWithin:** a vector of the within batch variance for each probe
- **probesetSD:** a vector of average within probeset standard deviations
- **medianSE:** a vector of median standard errors of the expression estimates (used by the GNUSE function)

1.2.3 Output Parameters

While the default is to only return the gene-level expression estimates and if applicable their standard errors, the frma function can also return additional information about the estimates depending on the summarization method chosen. A description of the arguments that can be included in the output.param argument follows:

- **weights:** the weights from the M-estimation procedure
- **residuals:** the residuals from fitting the probe-level model
- **randomeffects:** estimated random effects from fitting the probe-level model with random effect summarization

Note that not all of these outputs are available for all of the summarization methods.

2 Creation of Gene Expression Barcodes

The barcode algorithm is designed to estimate which genes are expressed and which are unexpressed in a given microarray hybridization. This is accomplished by: (1) using the distribution of observed log_2 intensities across a wide variety of tissues to estimate an expressed and an unexpressed distribution for each gene, and (2) using these estimated distributions to determine which genes are expressed / unexpressed in a given sample. The first step is accomplished by fitting a hierarchical mixture model to the plethora of publicly available data. The second step is accomplished by determining where the observed intensities from the new array fall in the estimated distributions. The default output of the `barcode` function is a vector of ones and zeros denoting which genes are estimated to be expressed (ones) and unexpressed (zeros). We call this a *gene expression barcode*.

For more details about the Gene Expression Barcode, please read the following papers:

McCall MN, Uppal K, Jaffee HA, Zilliox MJ, and Irizarry RA (2011). The Gene Expression Barcode: leveraging public data repositories to begin cataloging the human and murine transcriptomes, Nucleic Acids Research, 39:D1011-D1015.

McCall MN, Jaffee HA, Zelisko SJ, Sinha N, Hooiveld G, Irizarry RA, Zilliox MJ (2014). The Gene Expression Barcode 3.0: improved data processing and mining tools, Nucleic Acids Research, 42(D1):D938-D943.

2.1 Getting Started

To create a gene expression barcode, one needs estimates of the gene expression distributions – specifically the mean and variance of the unexpressed distribution for each gene. We have computed these for several popular Affymetrix platforms. To use one of these, simply preprocess your data using the default options of `frma` and then run the `barcode` function on the resulting object.

Similar to the `frma` function, the `barcode` function requires platform specific precomputed parameters. These parameters are stored in the same **R** packages as the frozen parameter vectors used by `frma`. In the default implementation, the `barcode` function attempts to load the appropriate set of parameters for the given `ExpressionSet` or `frma-ExpressionSet` object. It is also possible for the user to supply the necessary parameters via optional arguments.

2.1.1 Example

1. Download and install the frma package and the appropriate data package(s) (i.e. hgu133afrmavecs).

2. Load the frma package.

```
> library(frma)
```

3. Read in the data and preprocess using the default options.

```
> library(frmaExampleData)
> data(AffyBatchExample)
> object <- frma(AffyBatchExample)
```

4. Convert the expression values to a gene expression barcode.

```
> bc <- barcode(object)
```

2.2 Output Options

The default output of the `barcode` function is to return a vector of ones (expressed) and zeros (unexpressed); however, there are alternative output options. A brief description of each of these follows:

- **weight:** a vector of weights which roughly correspond to the probability of expression for each gene.
- **z-score:** a vector of z-scores under the unexpressed normal distribution for each gene.
- **p-value:** a vector of p-values under the unexpressed normal distributionfor each gene.
- **lod:** a vector of LOD scores under the unexpressed normal distributionfor each gene.