pyAffy: An efficient Python/Cython implementation of the RMA method for processing raw data from Affymetrix expression microarrays

Florian Wagner[1,2,*]

[1]**Graduate Program in Computational Biology and Bioinformatics, Duke University, Durham, NC, USA**
[2]**Center for Genomic and Computational Biology, Duke University, Durham, NC, USA**
[*]**Email: florian.wagner@duke.edu**

ABSTRACT

Robust multi-array average (RMA) is a highly successful method for processing raw data from Affymetrix expression microarrays. However, most of the work on microarray data processing predates the widespread use of Python in scientific computing. Here, I describe `pyAffy`, an efficient implementation of the RMA method in Python/Cython. Using data from the MAQC project, I show that this implementation produces virtually identical results compared to the RMA reference implementation in the `affy` R package, while running more than five times faster and consuming significantly less memory. I also show how individual steps of the RMA method affect the final expression estimates. The source code for `pyAffy` is available from PyPI and GitHub (`https://github.com/flo-compbio/pyaffy`) under an OSI-approved license. I intend to periodically revise this manuscript to ensure that it accurately reflects the functionalities available in the `pyAffy` Python package.

Keywords: microarray, expression, transcriptomics, Affymetrix, RMA, normalization, Python, Cython

INTRODUCTION

The development of high-density DNA microarrays in the late 1990s represented a breakthrough in terms of the accuracy and the price at which expression levels could be measured on a genome-wide scale (Brown and Botstein 1999). In subsequent years, much work was carried out to develop, validate, and benchmark computational methods for deriving gene expression levels from microarray data, e.g. as part of the MAQC project (Shippy et al. 2006; Tong et al. 2006). However, since the emergence of next-generation sequencing and its adoption for expression profiling (Mortazavi et al. 2008), most computational method development has (sensibly) focused on accurately and efficiently estimating expression levels from RNA-Seq data (e.g., Pertea et al. 2015).

Despite the dominance of RNA-Seq as the standard platform for expression profiling today, and the exciting prospects of even more powerful technologies currently under development, there is no reason to ignore the large amount of microarray expression data that has already been accumulated in public databases. The NCBI GEO database (Barrett et al. 2013) and the EBI ArrayExpress database (Kolesnikov et al. 2015) contain more than 40,000 and 7,000 microarray-based expression datasets, respectively (see Methods for how these numbers were ascertained). Some of the most commonly used microarray designs were manufactured by Affymetrix: In the GEO database, about 11,800 datasets (approx. 25%) relied on one of five Affymetrix array designs (see Table 1).

While processed versions of these datasets (i.e., expression matrices) are often stored in the database alongside the raw data, there are many reasons for why one would want to download and process the raw data. The first and most obvious one is consistency: Different researchers have used different methods or different parameter settings to process their data, resulting in inconsistencies and systematic biases when different datasets are combined. Second, some researchers may also have chosen suboptimal methods

Microarray Design	# Datasets
Affymetrix Human Genome U133 Plus 2.0	4,232
Affymetrix Mouse Genome 430 2.0	3,585
Affymetrix Mouse Gene 1.0 ST	1,651
Affymetrix Human Gene 1.0 ST	1,286
Affymetrix Human Genome U133A	1,043

Table 1. Number of datasets in the NCBI GEO database for various Affymetrix microarray designs (as of 2/24/2016).

or otherwise inadvertantly introduced artifacts during processing of the raw data. A third concern is reproducibility: If the original software used to generate the expression values is no longer available, or if the version and parameters used are not fully documented in the database, it may be difficult or even impossible to reproduce the expression values from the raw data. Any conclusions based on such data would therefore be difficult to validate externally. Lastly, many processing methods involve transformations such as quantile normalization (B. M. Bolstad et al. 2003), which depend on the joint distribution of expression values in all samples. However, frequently only a subset of the samples in a study are of interest. In these cases, one would want to exclude the other samples before normalization, again making it necessary to process the raw data.

Expression microarray technologies employed by Affymetrix and its competitors have been extensively reviewed in the literature (see e.g., Lipshutz et al. 1999; Bumgarner 2013). Briefly, Affymetrix expression microarrays are glass slides that typically contain more than one million *probes*. A probe is a cluster of identical, in-situ synthesized 25-mer single-stranded DNA oligonucleotides designed to hybridize to biotinylated cRNA (anti-sense RNA generated from cDNA by in-vitro transcription) with a complementary base sequence. The probe sequences are designed with the aim of matching transcript sequences of a single gene in the genome, and there are 11-20 probes for each gene present on the array, forming a so-called *probe set*. Once cRNA has been hybridized to the array, it can then be *stained* using a streptavidin-conjugated fluorescent dye, and a specialized scanner is used to measure the intensity of the fluorescent light emitted at the position of each feature.[1] These intensity values contain information about the expression level of each gene in the original RNA extraction, and constitute the most important type of raw data produced by Affymetrix microarrays.

Generally speaking, methods that attempt to derive accurate gene expression levels from raw microarray data must address several issues: First, some microarray designs were completed at a time when databases of gene transcript sequences contained inaccurate and/or redundant information. In those cases, the sequences of a significant fraction of probes may not match the genes they were supposed to target, or they may not be specific to that gene, resulting in undesired cross-hybridization effects. Such problems can be effectively addressed by re-mapping and re-annotating probe sequences, which has been done systematically for dozens of different array designs (Dai et al. 2005). These probe annotations are stored in *custom CDF files*, which can be downloaded from the Brainarray website[2]. A second issue is the fact that when scanned, expression microarrays exhibit certain amounts of background fluorescence. In other words, even for genes that are not expressed in a sample, the scanner is not expected to measure an intensity of zero. This requires each intensity value to be somehow corrected for the background (noise) level. Furthermore, the absolute signal intensities may vary systematically between individual arrays, making it necessary to adopt procedures that put all arrays in one study "on the same scale". Lastly, in order to obtain a single expression value per gene, the intensities of all probes in a probe set must be summarized in some way.

One of the most successful methods for processing raw data from Affymetrix expression microarrays is the RMA method (Irizarry et al. 2003) which continues to be very actively cited (776 citations in 2015 alone, according to Google Scholar[3]). Similarly, the `affy` R package (Gautier et al. 2004), which

[1] For an overview of Affymetrix microarray design and protocols, see also: http://www.ohsu.edu/xd/research/research-cores/gene-profiling-shared-resource/project-design/array-technology/affymetrix-genechip-arrays.cfm

[2] http://brainarray.mbni.med.umich.edu/Brainarray/Database/CustomCDF/genomic_curated_CDF.asp

[3] Source: https://scholar.google.com/scholar?hl=en&as_sdt=5%2C34&sciodt=0%2C34&cites=

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

implements the RMA method and also supports the use of custom CDF files (see above), still enjoys great popularity, as evidenced by Bioconductor download statistics (a monthly average of about 4,500 distinct IP downloads from Bioconductor in 2015[4]). It should be noted that the `affy` package provides many additional functionalities besides performing RMA, including alternative processing and normalization methods and tools for performing quality control. However, perhaps due to the aforementioned shift in attention to RNA-Seq data, software packages for efficiently processing microarray expression data do not appear to be readily available in other programming environments. In particular, I was unable to find a good replacement for the `affy` package in Python. I therefore set out to create an efficient implementation of the RMA algorithm in Python/Cython, resulting in the `pyAffy` package.

RESULTS

`pyAffy` expression values are nearly identical to those produced by `affy`

To assess the fidelity of the `pyAffy` implementation of RMA, I downloaded the raw data for all Affymetrix U133 Plus 2.0 microarray samples from the MAQC study. This dataset comprises four different types of samples (A-D), assayed at six different sites (1-6). Each of the resulting 24 datasets (1A-D, 2A-D, etc.) is available in five replicates, for a total of 120 samples. To test whether expression values produced by `pyAffy` are identical to those produced by `affy`, I processed each of the 24 datasets independently with both packages. A comparison of the expression values generated by each package based on the first replicate from each dataset showed that both implementations produce nearly identical expression values (see Figure 1).

`pyAffy` outperforms `affy` in terms of speed and memory usage

In order to compare the speed and memory usage of the RMA implementations in both `affy` and `pyAffy`, I used both packages to process one small (n = 5) and one large (n = 120) dataset. The small dataset consisted of the five MAQC replicates from site 2, sample type "A" (`MAQC-2A`). The large dataset consisted of all MAQC datasets for sample types A-D (`MAQC-all`). The resulting run times and peek memory usage values are shown in Table 2. On both datasets, `pyAffy` processed the samples about five times faster than `affy`, while requiring significantly less memory, especially for the larger dataset (1 GB vs 5.4 GB).

Dataset	Number of samples	`affy` run time (s)	`pyAffy` run time (s)	`affy` peak memory usage (MB)	`pyAffy` peak memory usage (MB)
`MAQC-2A`	5	108	**23**	415	**140**
`MAQC-all`	120	619	**113**	5,400	**1,000**

Table 2. Benchmark results pm am for processing MAQC data with `affy` and `pyAffy`. The benchmark results for the `affy` package do not include the time required for creating a custom CDF R package using `makecdfenv`.

The effects of individual steps in the RMA algorithm

The RMA algorithm can be understood as a series of individual decisions and procedures:

1. Before any processing is done, all data from mismatch (MM) probes is discarded. In contrast to the primary, so-called perfect match (PM) probes, these probes harbor a single-nucleotide mismatch when compared to their intended gene target. They were originally intended to control for cross-hybridization artifacts, but Irizarry et al. (2003) demonstrated that ignoring MM intensities resulted in more accurate expression measures than previously adopted expression measures that attempted to take the intensities of MM probes into account.
2. The RMA model for correcting intensities for background noise (Irizarry et al. 2003; Bolstad 2004) is applied to each sample.

9621984605976884670&scipsc=&as_ylo=2015&as_yhi=2015

[4]Source: http://bioconductor.org/packages/stats/bioc/affy.html

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

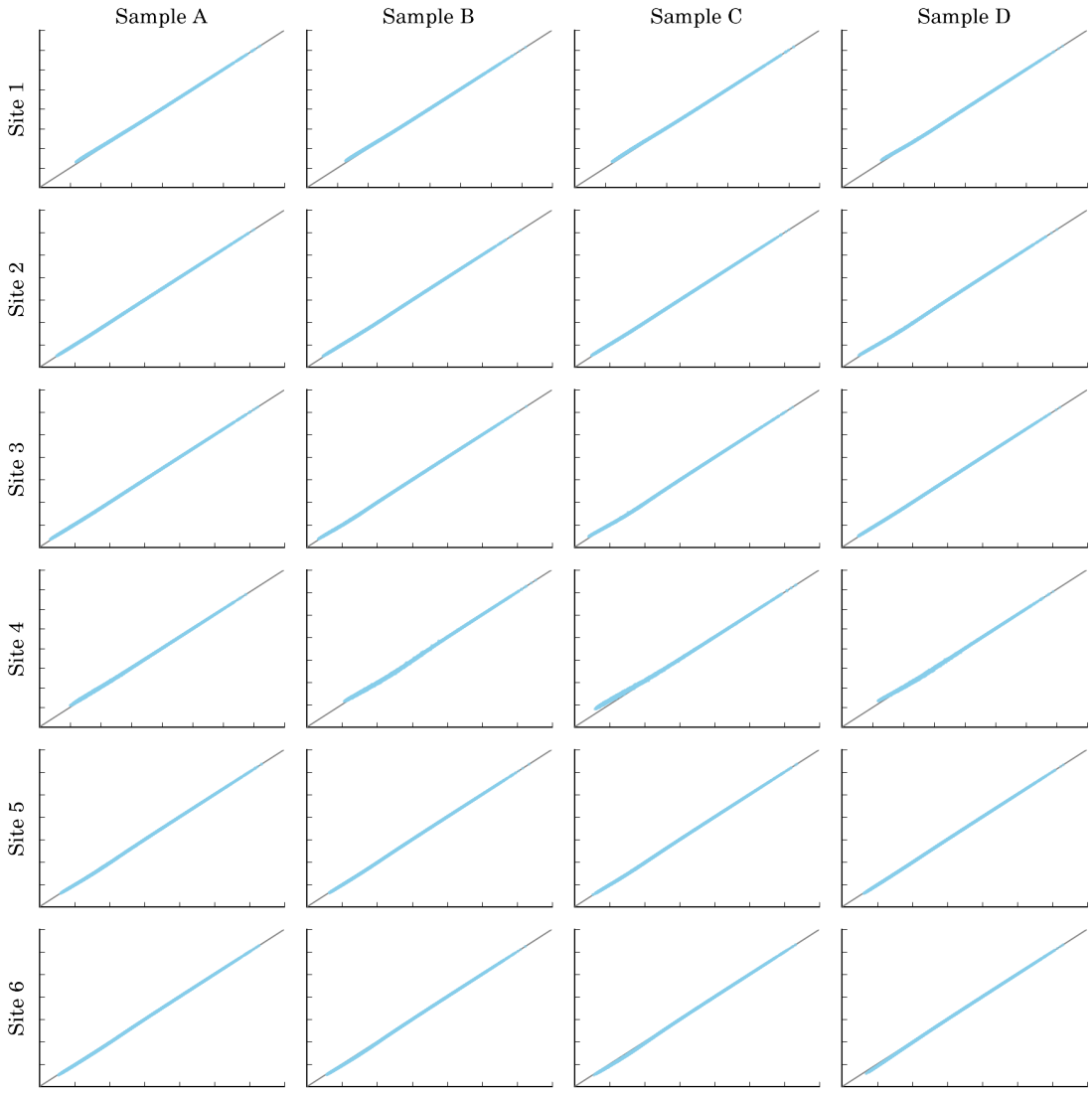

Figure 1. Comparison of RMA expression values produced by `affy` and `pyAffy`. Each plot shows the result of an independent analysis, in which raw data from the same set of five replicates was processed with both packages. Shown are scatter plots of the log-scale expression values (blue) of all genes in the first replicate of each analysis. The horizontal (x) axis in each plot represents the `affy` values, and the vertical (y) axis the `pyAffy` values. The gray line indicates the identity function. All plots show the entire range of expression values. Axis scales are adjusted separately for each plot.

3. The entire probe-by-sample intensity matrix is subjected to quantile normalization (B. M. Bolstad et al. 2003).
4. All intensity values are log2-transformed.
5. Probes are grouped according to the gene (probe set) they belong to. For each gene, a sub-matrix containing only the intensities of probes of that gene is generated. Then, median polish (Tukey 1977) is applied to this sub-matrix, resulting in estimates for row effects, column effects, and an overall effect. The expression values of the gene then correspond to the column effects plus the overall effect. Here, the median polish procedure serves to summarize the data from different probes for each gene.

I decided to again use the five replicates from the `MAQC-2A` dataset (see above) to better understand how some of the key steps of the RMA algorithm affect the final expression values. I first compared the RMA-based expression levels produced by `pyAffy` to expression levels obtained directly from the raw

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

data with minimal processing. In this analysis, the intensities of MM probes are *not* discarded (Step 1), the background correction model is *not* applied (Step 2), quantile normalization is *not* performed (Step 3), and median polish is *not* used to summarize probe-level data (Step 5). Instead of median polish, the median intensity value across probes is used. As can be seen from Figure 2 (top row), omitting all these steps of the RMA algorithm results in a very different global distribution of expression values, as well as a significant amount of variation at the gene level ($r \approx 0.93$).

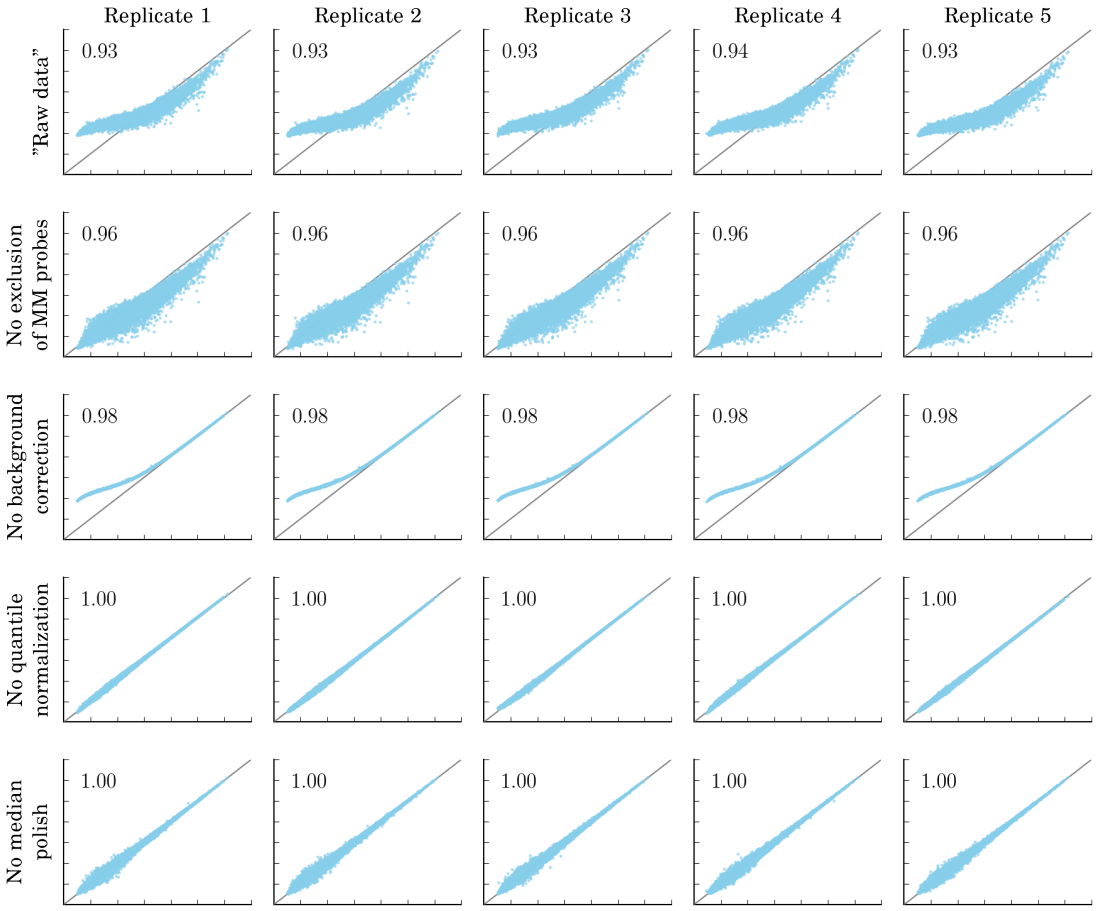

Figure 2. Effects of individual steps of the RMA algorithm (as implemented in `pyAffy`) on final gene expression levels. Each row shows the expression levels obtained with a modified version of the RMA algorithm compared to the standard version, for all five replicates of the `MAQC-2A` dataset. Gene expression levels and the identity function are shown as in Figure 1. All plots are shown on the same scale, and numbers indicate Pearson correlation coefficients.

To study the effect of individual steps in the RMA algorithm in isolation, I repeated the previously described anlaysis while only omitting a single step at a time. Not excluding MM probes had the most dramatic effect (see Figure 2, second row), affecting both the global distribution and introducing gene-level variation. In contrast, omitting the background normalization step only changed the distribution, but did not introduce gene-level variation (third row). This was expected based on the details of how the RMA background correction model is designed (see Methods). Somewhat surprisingly, the omission of quantile normalization did not affect expression levels much, introducing only minimal amounts of gene-level variation (fourth row). One reason for this might have been that all samples in each analysis were technical replicates, and were therefore expected to exhibit the same distribution of expression values. If the datasets analyzed had been more heterogeneous, the application of quantile normalization could have had a much more pronounced effect. Finally, the omission of median polish introduced small but noticeable amounts of gene-level variation, especially for lowly expressed genes (bottom row).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

DISCUSSION

Current limitations of `pyAffy`

Currently, `pyAffy` has a number of limitations which I intend to address in future work:

- `pyAffy`'s CEL file parser cannot handle CEL files in "Command Console" format (all MAQC CEL files were in "Version 3" or "Version 4" formats. (This is an important limitation that I will address first.)
- `pyAffy` is ignorant of missing data: Individual probes on microarrays can be masked as "outliers", either automatically by the Affymetrix image processing software, or manually by the researcher. Currently, `pyAffy` treats all data as "present", ignoring all information about outliers etc.
- Working with compressed (`gzip`'ed) CEL files is only supported on Unix and Unix-like platforms (i.e., Linux and Mac OS).
- `pyAffy` has not been tested on microarray designs other than the Human Genome U133 Plus 2.0.

Given these and other limitations, it is clear that `pyAffy` does not yet provide the same range of features and functionalities as the `affy` R package. The goal of the work described here was to provide an efficient implementation of the RMA method that produces highly similar values compared to the implementation found in `affy`. `pyAffy` therefore represents an alternative to researchers who either prefer to use a Python package or simply want to benefit from the shorter processing time and smaller memory footprint for this particular task. Studying the literature in combination with analyzing the `affy` and `preprocessCore` source code also proved to be an illuminating experience that facilitated a more thorough understanding of the RMA method, as well as the relationship between the processed and the raw data.

Practical considerations in comparing expression microarray to RNA-Seq data

For large datasets, the RMA implementation presented here processes microarray data in under one second per sample (see Table 2). In my experience, this is roughly 50 times faster than a RNA-Seq processing pipeline that uses efficient tools for read mapping and expression quantification (Kim, Langmead, and Salzberg 2015; Pertea et al. 2015). In combination with the significantly smaller file sizes, this can make microarray data much easier to work with than RNA-Seq data. For example, when the same experiment has been carried out using both platforms, and both datasets are stored in a database like NCBI GEO, a researcher might choose to work with the microarray data initially, simply because it takes much less time to download and process the raw data. Indeed, it might be useful to incorporate a feature in the `pyAffy` package that would allow researchers to automatically download and process arbitrary datasets based on GEO series accession number.

METHODS

NCBI GEO and EBI ArrayExpress statistics

All statistics were collected on 2/24/2016. The current number of microarray expression studies in the NCBI GEO database was obtained from the summary page (`http://www.ncbi.nlm.nih.gov/geo/summary/`), under "Expression profiling by array". The current number of microarray expression studies in the EBI ArrayExpress database was obtained from the ArrayExpress browser (`https://www.ebi.ac.uk/arrayexpress/browse.html`), by using the "experiment type" filters "RNA assay" and "array assay" in combination, and selecting "ArrayExpress data only".

Efficient parsing of Affymetrix CEL and CDF files

The first challenge in processing Affymetrix expression data is to implement an efficient parser for the CEL files containing the intensity values. CEL files come in three different formats (one plain-text and two binary), which are well-documented by Affymetrix[5]. My initial implementation of a CEL file parser in Python was decidedly too slow, taking approximately 50 times longer than the current Cython implementation. Cython as a programming language is a superset of Python, and the Cython compiler generates C code that makes calls to the Python C-API in order to work with Python objects Behnel et al. 2011. Importantly, Cython also provides an interface for the efficient manipulation of NumPy arrays. All

[5] See: `media.affymetrix.com/support/developer/powertools/changelog/gcos-agcc/cel.html`

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

portions of the Cython code that do not require Python C-API calls run extremely fast. This makes the Python/Cython a very attractive choice for implementing bioinformatics tools, which can often benefit enormously from the combination of high-level programming with flexible data structures in Python on the one hand, and the runtime efficiency obtained with Cython on the other. The Cython CEL file parser relies on file reading and string manipulation routines from the C standard library, and also makes use of named pipes[6] to efficiently unzip any `gzip`-compressed CEL files in the background.

Another parsing task is to read the custom CDF files from the Brainarray website. These are plain-text files in "Windows INI"-style format (complete with Windows-style `\r\n` newlines). Again, the CDF file format is well-documented by Affymetrix[7]. I initially used a Python implementation based on the `configparser` package[8] for parsing, but this was again much too slow compared to the Cython-based implementation in use now. The current implementation supports reading only perfect match (PM) probes (the default), only mismatch (MM) probes, or all probes. The parser uses the following criterion to determine whether a probe (corresponding to a row in the CDF file) represents a PM or an MM probe: If the `CBASE` attribute (eigth column) matches the `TBASE` attribute (ninth column), the probe is considered an MM probe. Otherwise, it is considered a PM probe. Based on the validation results (see Figure 1), I believe that this is the correct assignment, even though I did not find this information in the documentation.

Implementation of the RMA algorithm in Python/Cython

To create a faithful implementation of the RMA algorithm, I primarily relied on three sources: First, the well-known RMA paper by Irizarry et al. (2003), which describes the general methodology. Second, the PhD dissertation thesis by Bolstad (2004), which contains a derivation of the formula that is used to calculate the background-corrected signal, and goes into more detail regarding the way parameters are estimated. Finally, I analyzed relevant parts of the source code of the `affy` (version 1.48.0), `preprocessCore` (version 1.32.0), and `stats` (version 3.1.1) R packages. The `affy` package provides the `ReadAffy` and `expresso` functions that serve as front-ends for reading the data and applying the RMA method, respectively. The background correction and quantile normalization steps are implemented in C as part of the `preprocessCore` package, and the median polish algorithm is implemented in the `stats` package. While a detailed line-by-line analysis of the relevant portions of code in those packages is beyond the scope of this manuscript, I will summarize some similarities and differences between the implementations in `affy` and `pyAffy` in the following paragraphs.

The RMA background correction model (Irizarry et al. 2003) assumes that each sample (microarray) exhibits a certain average background level β_i (here, i is the sample index). Irizarry et al. 2003 suggested that a "naive" approach would be to simply substract β_i from all intensity values on array i, but that this would fail since some intensity values would become negative. However, rather than simply setting such values to zero (or perhaps 1, to obtain zero values after log-transformation), they proposed to treat the observed probe intensities $PM_i j$ (j is the probe index) as realizations of a random variable that represents the sum of a background intensity variable b_i with a truncated normal distribution, and an exponentially distributed signal variable s_i, with b_i and s_i independent. Why this particular model with these particular distributions was chosen is not completely clear; according to Bolstad (2004, p. 21) (who is not an author of Irizarry et al. (2003)), it is "motivated by looking at the distribution of probe intensities". This is curious insofar as the probability density function of an exponential distribution is strictly decreasing, while the probability of an intensity representing true signal should generally increase for higher intensities. As further described in Bolstad (2004, p. 21), the RMA method relies on an "ad-hoc" approach for estimating the parameters of the normal and exponential distributions. The mode of the kernel density estimate of the probe intensities is used as the normal (noise) mean μ, and the "lower tail" of the density is used to estimate the normal (noise) standard deviation σ. Finally, "an exponential is fitted to the right tail to estimate α", the parameter of the exponential (signal) distribution. The adjusted signal intensity is then given by the conditional expectation $E(X \mid S = s)$[9]. I visualized the net effect of the RMA background correction model on MAQC data, and found that it retains a certain fraction of the signal (i.e., the intensity value) that generally increases with probe intensity, except for very low intensities, where the model

[6] See: https://en.wikipedia.org/wiki/Named_pipe

[7] See: http://media.affymetrix.com/support/developer/powertools/changelog/gcos-agcc/cdf.html

[8] https://docs.python.org/3/library/configparser.html

[9] see (Bolstad 2004, p. 20f) for the formula.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

retains a majority of the signal (see Figure 3). However, this inconsistency does not appear to significantly affect the data in practice (see Figure 2, third row).

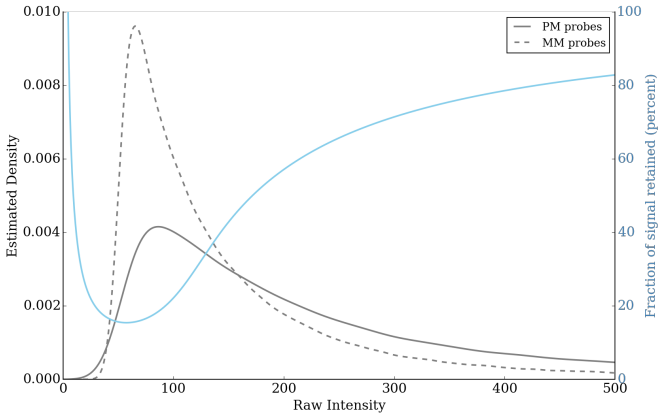

Figure 3. The effect of the RMA background correction model. For one sample from the MAQC study, kernel density estimtates of the distributions of the PM and MM probe intensities are shown in gray (the MM probes are not used for parameter estimation and are only shown for refernce). For each intensity level, the fraction of signal retained after applying RMA background correction is shown in blue (c.f. Figure 2, third row). The unexpected rise of the fraction of signal retained for very low intensities can be explained by the choice of an exponential distribution to model the true signal, which has its maximum density at $x = 0$. However, this effect does not appear to have a noticeable impact on the data in practice.

The `affy` package relies on C code in the `preprocessCore` package to implement this procedure. Specifically, the parameter values are estimated by the function `rma_bg_parameters` (line 216 in `rma_background4.c`[10]), which itself calls the functions `get_sd` to estimate σ and `get_alpha` to estimate α. Three implementation details appear noteworthy: First, in the course of estimating the three parameters (μ, σ, and α), three different kernel density estimations are performed (lines 227, 237, and 196), each one using an Epanechnikov kernel with a bandwidth estimated according to Silverman's rule of thumb[11]. Second, the estimate of σ appears to get arbitrarily scaled by a factor of $\sqrt{2}$ (line 216). Third, the estimate of α is set to the mode of a kernel density estimated based on the probes with original intensity values larger than μ. However, this estimate is obtained *after substracting μ from the intensity values of these probes* (line 193). It is unclear how this procedure is useful for fitting "an exponential [...] to the right tail [of the distribution of probe intensities] to estimate α". Due to the L-shaped distribution of probe intensity values to the right side of μ, this procedure must result in values close to zero — since after substracting μ, most intensity values will be close to zero, and so the mode of the density estimate will be close to zero as well. After all parameters are estimated, the probe intensity values are adjusted by the function `rma_bg_adjust` (line 300).

The RMA implementation in `pyAffy` optimizes the procedure implemented in the `preprocessCore` package for speed, while attempting to obtain highly similar parameter estimates. Concretely, for estimating μ, I decided to replace two kernel density estimation steps with a simple histogram calculation (with a fixed bin width of 4.0), using the bin with the largest number of probes as a substitute for the mode of the density estimate[12]. I left the estimation of σ (given μ) unchanged and retained the scaling factor of $\sqrt{2}$. Finally, based on the observations described above, I decided to simply set α to a fixed value value close to zero (0.03). I determined this value empirically, by testing which value would result in experssion values that appeared most similar to those produced by `affy`. Even though this procedure of estimating the three parameter appeared radically simplified compared to the one performed in `preprocessCore`, it resulted in highly similar expression values in practice (see Figure 1) while

[10] See https://github.com/Bioconductor-mirror/preprocessCore/blob/release-3.2/src/rma_background4.c.

[11] See https://github.com/Bioconductor-mirror/preprocessCore/blob/release-3.2/src/weightedkerneldensity.c#L799 and https://stat.ethz.ch/R-manual/R-devel/library/stats/html/bandwidth.html.

[12] More precisely, μ is set to mean of the two edge values of that bin.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

avoiding all three time-consuming kernel density estimation steps.

Both quantile normalization and median polish are relatively simple algorithms. For quantile normalization, `pyAffy` relies on the implementation of quantile normalization from the `genometools` Python package that is also authored by me[13]. I also implemented median polish using code from the R `stats` package as a template.

Benchmarking of the `affy` R package

In order to process the MAQC microarrray data with using accurate probe annotations, I downloaded the corresponding custom CDF file from the Brainarray website[14] and used the `makecdfenv` R package to create an annotation package (termed `hgu133plus2hsentrezgcdf`) for use with `affy`. I then wrote two R scripts to benchmark the `affy` package: One for the the `MAQC-2A` dataset, and one for the `MAQC-all` dataset. Both scripts use the `ReadAffy` function to read compressed (`gzip`'ed) CEL files from the MAQC study (GEO accession: GSE5350), and the `expresso` function to process the data. Specifically, the commands used were:

```
dataset <- ReadAffy(
    widget=FALSE,
    filenames=cel_files,
    sampleNames=samples,
    celfile.path=cel_dir,
    compress=TRUE,
    cdfname='hgu133plus2hsentrezgcdf'
)

eset <- expresso(
    dataset,
    bgcorrect.method='rma',
    normalize.method='quantiles',
    pmcorrect.method='pmonly',
    summary.method='medianpolish'
)

write.exprs(eset, file = output_file)
```

The benchmarking was performed for `affy` version 1.44.0, using R 3.1.1. Execution time was measured by calling the `Sys.time()` function at the beginning and end of the script and calculating the time difference. Memory consumption was measured by monitoring the output of the `top` Linux program (column "`RES`").

Benchmarking of the `pyAffy` Python package

`pyAffy` does not require preprocessing of the custom CDF annotations, and was directly provided with the custom CDF file contained in the Brainarray zip file (see above). The benchmark was performed from within a Jupyter notebook, with exact same CEL files as input as for the `affy` benchmark. The commands used were:

```
from pyaffy import rma
genes, samples, X = rma(cdf_file, sample_cel_files)
E = ExpMatrix(genes, samples, X)
E.write_tsv(output_file)
```

The benchmarking was performed for `pyAffy` version 0.2.0 and `genometools` version 1.2.2, using Python 2.7.9. Execution time was measured by calling the `time.time()` function at the beginning and end of the script and calculating the time difference. Memory consumption was measured as in the `affy` benchmark.

[13] See https://pypi.python.org/pypi/genometools.

[14] Source: http://mbni.org/customcdf/20.0.0/entrezg.download/HGU133Plus2_Hs_ENTREZG_20.0.0.zip.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

Applying RMA with individual processing steps omitted.

The `rma` function in the `pyAffy` takes optional parameters (with default value `True`) that allow individual steps of the RMA algorithm to be skipped (if `False` is specified). To obtain the "raw data" expression levels shown in the top row of Figure 2, the following command was used:

```
rma(
    cdf_file, sample_cel_files,
    pm_probes_only = False, # no exclusion of MM probes
    bg_correct = False, # no background correction
    quantile_normalize = False, # no quantile normalization
    medianpolish = False # no median polish
)
```

ACKNOWLEDGMENTS

I would like to thank Dr. Sandeep Dave for his support.

REFERENCES

Barrett, Tanya et al. (2013). "NCBI GEO: archive for functional genomics data sets–update". In: *Nucleic Acids Research* 41 (Database issue), pp. D991–995. DOI: 10.1093/nar/gks1193.

Behnel, Stefan et al. (2011). "Cython: The Best of Both Worlds". In: *Computing in Science and Engg.* 13.2, pp. 31–39. DOI: 10.1109/MCSE.2010.118. URL: http://dx.doi.org/10.1109/MCSE.2010.118 (visited on 02/10/2015).

Bolstad (2004). "Low-level analysis of high-density oligonucleotide array data: background, normalization and summarization". PhD thesis. University of California, Berkeley. URL: http://bmbolstad.com/Dissertation/Bolstad_2004_Dissertation.pdf (visited on 02/08/2016).

Bolstad, B. M. et al. (2003). "A comparison of normalization methods for high density oligonucleotide array data based on variance and bias". In: *Bioinformatics (Oxford, England)* 19.2, pp. 185–193.

Brown, P. O. and D. Botstein (1999). "Exploring the new world of the genome with DNA microarrays". In: *Nature Genetics* 21.1, pp. 33–37. DOI: 10.1038/4462.

Bumgarner, Roger (2013). "Overview of DNA microarrays: types, applications, and their future". In: *Current Protocols in Molecular Biology / Edited by Frederick M. Ausubel ... [et Al.]* Chapter 22, Unit 22.1. DOI: 10.1002/0471142727.mb2201s101.

Dai, Manhong et al. (2005). "Evolving gene/transcript definitions significantly alter the interpretation of GeneChip data". In: *Nucleic Acids Research* 33.20, e175. DOI: 10.1093/nar/gni179.

Gautier, Laurent et al. (2004). "affy–analysis of Affymetrix GeneChip data at the probe level". In: *Bioinformatics (Oxford, England)* 20.3, pp. 307–315. DOI: 10.1093/bioinformatics/btg405.

Irizarry, Rafael A. et al. (2003). "Exploration, normalization, and summaries of high density oligonucleotide array probe level data". In: *Biostatistics (Oxford, England)* 4.2, pp. 249–264. DOI: 10.1093/biostatistics/4.2.249.

Kim, Daehwan, Ben Langmead, and Steven L. Salzberg (2015). "HISAT: a fast spliced aligner with low memory requirements". In: *Nature Methods* 12.4, pp. 357–360. DOI: 10.1038/nmeth.3317.

Kolesnikov, Nikolay et al. (2015). "ArrayExpress update–simplifying data submissions". In: *Nucleic Acids Research* 43 (Database issue), pp. D1113–1116. DOI: 10.1093/nar/gku1057.

Lipshutz, R. J. et al. (1999). "High density synthetic oligonucleotide arrays". In: *Nature Genetics* 21.1, pp. 20–24. DOI: 10.1038/4447.

Mortazavi, Ali et al. (2008). "Mapping and quantifying mammalian transcriptomes by RNA-Seq". In: *Nature Methods* 5.7, pp. 621–628. DOI: 10.1038/nmeth.1226. URL: http://www.ncbi.nlm.nih.gov/pubmed/18516045 (visited on 01/11/2012).

Pertea, Mihaela et al. (2015). "StringTie enables improved reconstruction of a transcriptome from RNA-seq reads". In: *Nature Biotechnology*. DOI: 10.1038/nbt.3122.

Shippy, Richard et al. (2006). "Using RNA sample titrations to assess microarray platform performance and normalization techniques". In: *Nature Biotechnology* 24.9, pp. 1123–1131. DOI: 10.1038/nbt1241.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016

Tong, Weida et al. (2006). "Evaluation of external RNA controls for the assessment of microarray performance". In: *Nature Biotechnology* 24.9, pp. 1132–1139. DOI: 10.1038/nbt1237.

Tukey, John Wilder (1977). *Exploratory Data Analysis*. Addison-Wesley Publishing Company. 714 pp.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1790v1 | CC-BY 4.0 Open Access | rec: 27 Feb 2016, publ: 27 Feb 2016