A New Approach to Sample Deconvolution

by

Gregory Hunt

A dissertation proposal
submitted as part of the requirements
for the oral preliminary examination
for the department of statistics

November 23, 2016

Committee:
Assistant Professor Johann Gagnon-Bartsch, Chair
Professor Kerby Shedden
Professor Naisyin Wang

TABLE OF CONTENTS

ABSTRACT

A New Approach to Sample Deconvolution

by

Greg Hunt

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

CHAPTER 1

Introduction

High-throughput gene expression profiling technologies such as DNA microarrays and RNA sequencing have become important microbiological tools. Their power and popularity come from their ability to make a large number of simultaneous measurements. Such technologies allow the measurement of tens of thousands of gene expressions. This gives biologists a high-level view of genome activity as well as allowing investigation of individual genes. While these profiling techniques have proven to be invaluable tools their analyses are not without caveats. One issue which confounds gene expression profiling is sample heterogeneity.

Profiling techniques measure gene expression of a sample. A sample is simply a collection of cells. These cells typically are cultivated from animal tissue, for example, a rat tumor biopsy or human blood titer. The gene expressions measured reflect the average gene expression across cells in the sample. Thus, for a sample consisting of precisely one cell type the measurements typify a gene expression profile for that type. However, for a heterogeneous sample of several cell types the measured expressions are not as interpretable. In this case the measurements will be a convolution of the expression profiles from the constituent types.

The convoluted nature of heterogeneous data makes comparing expression measurements across samples difficult. It becomes necessary to determine if changes in measurements come from genuine expression changes in cells or from compositional changes affecting the samples' mixing proportions. More to the point, it is often of interest to analyze expression changes in individual cell types and not a mixture thereof. Unfortunately heterogeneous samples are the rule not the exception and comparative analyses, like differential expression analysis, are a primary application of interest.

Accordingly, researchers have two goals when dealing with heterogeneous data. First, they would like to remove any confounding effects of sample heterogeneity. Secondly, they want to be able to get cell subtype specific gene expression profiles. Methods to these ends are dubbed *cell type deconvolution*. Solutions have been pioneered at both the physical cell sample level and data analytic level. Physical techniques like flow cytometry attempt to solve the heterogeneity problems by sorting cells into their respective types before analysis is performed. These techniques are employed to purify heterogeneous samples into homogeneous ones allowing researchers to be sure that the final samples analyzed contain only one cell type. However these techniques are expensive, time intensive and have been noted to change gene expression. Thus, in lieu of physical methods, there is ample room for data analytical techniques to deconvolve expression signal from heterogeneous samples.

CHAPTER 2

Scientific Background

All gene profiling techniques involve the so-called *central dogma of genetics*. That is, DNA is transcribed into messenger RNA (mRNA) and mRNA is translated into proteins. Thus, to a first approximation, when looking to determine which genes are being expressed by a cell one needs only look at what mRNA is being created by a cell. This follows because every time a gene is expressed an mRNA molecule specific to that gene is produced.

This one-to-one correspondence between gene expression and mRNA production is exploited by gene profiling techniques. A profiling experiment starts with a sample of cells. From these cells the mRNA is extracted and quantified. The scientist then fragments some fixed amount of this mRNA for use in analysis. Fragmentation is the process of breaking up long mRNA molecules into smaller chunks. This aids in the main profiling pursuit, namely, identification of the mRNA. The goal is to determine which species of mRNA is present. Determining what types of mRNA is present will tell scientists which genes are being expressed. Each of the many profiling techniques have a different way of determining what types of mRNA is present. For example, RNA-Seq determines the base-pair ordering on the mRNA fragments. These base-pair sequences can then be mapped to mRNA molecules and subsequently genes. On the other hand, DNA microarrays determine mRNA quantities by fluorescent tagging. Our discussion in this paper will focus mainly on DNA microarray technology.

A DNA microarray is a small chip (about one inch on a side) resembling a microscope slide. The surface of the chip is divided up into an array of squares we will call "probe squares." Each of these probe squares will probe for mRNA fragments containing a specific base sequence. That is, each of the squares will be used to measure the amount of a specific type of mRNA fragment. Hence we can map each of these squares from the fragment type they measure to a specific mRNA molecule and thus a specific gene. In this way we can determine the expression of a gene in our sample. We do this by looking at all the probes sensitive to mRNA fragments produced by the gene. The probe squares will tell us about gene expression as follows. To each of the probe squares on the microarray there are affixed thousands of nucleic acid fragments called probes. Each of the probes on a probe square have base-pair sequences complementary to one specific mRNA fragment type. Thus the probes on any particular probe square will hybridize to one specific species of mRNA fragment. The scientist then fluorescently tags the mRNA fragments extracted from the cell sample. These fluorescently tagged mRNA fragments are then put on the DNA microarray. The tagged fragments will hybridize to their complementary probes which are located on a specific probe square. All non-hybridized fragments are washed from the microarray. The array is then hit with a laser to excite the hybridized and fluorescently tagged sample fragments. If a gene is being expressed highly in the cell sample then there will be a lot of its mRNA molecules present and thus many fluorescently tagged mRNA fragments corresponding to that gene. Once hybridized to the array and hit with a laser the probe squares sensitive to mRNA fragments from that gene will fluoresce brightly. They fluoresce brightly because there are many fluorescently tagged mRNA fragments hybridized to their probes. Thus we see that the gene is highly expressed because its probe squares are fluorescing a lot. On the other hand if a gene is not being expressed much by cells in our sample there will be relatively few of its mRNA molecules and subsequently few of its fluorescently tagged fragments. Thus the probe squares corresponding to this gene will fluoresce only a little as there are not many tagged fragments available to hybridize to the probes. In this way we can look at the microarray and determine which genes are relatively highly expressed by looking at which probe squares are lighting up brightly. To enable quantitative analysis of the microarray a picture is taken of the fluorescing array. For each of the probe squares in the picture a fluorescence intensity is recorded. Thus for each probe square on the microarray the scientists get an intensity measurement that is positively related to the expression of a particular gene in our sample. We call these measurements the probe-level intensity measurements. In many cases these probe-level measurements are summarized to a gene-level by aggregating all the probe intensities related to a particular gene.

In any case, whether we are working with DNA microarrays or RNA-seq or some other profiling technology the data analyst gets a list of genes and corresponding expression signals. Sometimes (as when working with probe-level data) there are repeated measurements such that each gene has several measurements somehow capturing its expression. Furthermore there are often pre-processing and post-processing adjustments and transformations applied to gene expressions. Regardless, the expression signals a scientist gets through one of these methods are positively related to the average expression of a gene by the cells in the sample.

CHAPTER 3

Literature Review

The first computational method for cell type deconvolution is generally attributed to Venet et al. (2001). Since then there has been much activity in this area with dozens of methods published as recently as this past year (Newman et al. 2015). Notably, the model used for cell type deconvolution is largely the same across methods. Authors use some version of a linear model. There are, however, important differences in data assumptions and fitting methods to be considered. Our attempt here is to lay out a comprehensive, yet concise, survey of existing cell type deconvolution methodology.

3.1 Model Basics

Let's assume that for any cell sample we make N expression measurements. We are purposely ambiguous here about exactly what we mean by " expression measurements." For a DNA microarray these N measurements may be probe-level intensity measurements. They may also be gene-level expressions summarizing the probe intensities by an algorithm like MAS5 (Hubbell et al. 2002) or RMA (Irizarry et al. 2003). Furthermore, while we focus on DNA microarrays, the methods presented are likely easily applicable to some transformed mRNA counts from RNA-seq. In any case each of the N measurements captures the quantity of some nucleic acid in our sample. For ease of terminology we will henceforth refer to the nucleic acid fragments measured by each of the N expressions as "oligonucleotides" or, concisely, "oligos." Depending on context the measurement of a "oligo" may refer to the expression measurement of some mRNA fragment itself or to the summary measurement of all the mRNA transcripts from a gene. Without loss of generality we can label these oligos from 1 to N. We will need to assume that some of these oligo measurements are "characteristic" of a particular cell type. We call these "characteristic oligos." An oligo is characteristic of a particular cell type if it is abundant in cells of that type and not abundant in cells of any other type. The hope is that these characteristic oligos will allow us to distinguish among cell types.

Usually experiments are done in batches. Let's assume that we have S cell samples and conduct the same assay on each of the samples. Let X_{sn} be the expression measurement of the n^{th} oligo in the s^{th} sample. Then the matrix $X \in \mathbb{R}^{S \times N}$ has rows X_s each containing the expression measurement of all N oligos in sample s. The columns of X, denoted X^n, are the expression measurements of the n^{th} oligo across all samples. Furthermore posit that each of the s samples is a heterogeneous mixture of K cell types. Define the mixing matrix $M \in \mathbb{R}^{S \times K}$ to be so that M_{sk} is the percent of sample s comprised of cells of type k. The rows of M are the mixing proportions of the K cell types in the s^{th} sample meaning that M is row-wise a probability matrix. Finally define the matrix $U \in \mathbb{R}^{K \times N}$ to be the matrix of characteristic oligo measurements. Specifically U_{kn} is the typical oligo expression of oligo n in cell type k. Each row of the matrix, $U_k \in \mathbb{R}^N$, is what we call a oligo expression profile for type k. It is a vector that contains the "characteristic" or "typical" expression measurements of cells of type k for each of the N oligos.

By and far the most common manner to model cell-type convolution is as a linear model such that

$$X = MU + E \tag{3.1}$$

where E is a random matrix of errors. Almost universally cell-type convolution is modeled as such a matrix product of M and U however most of the deconvolution literature does not explicitly state such a statistical model. Mainly authors do not discuss an error term E much less its distribution. Thus we are correspondingly vague about E.

This model is attractive because it is simple. The model posits that the convoluted expressions from heterogeneous mixtures of K cell types, as captured in X, are linear combinations of some "characteristic" expressions of each of the K types, captured by U, with weights that are simply the mixing proportions of the types in each of the samples, the matrix M. Put another way, cell type convolution is a linear mixing process. Some version of this linear mixing model is used in all the methods we surveyed. While the true relation between the characteristic expression profiles U and the convoluted data X is known to be non-linear this model has proven both approximately true and empirically quite useful (Shen-Orr et al. 2010).

Beyond this basic model there are several facets which set apart existing deconvolution techniques. Firstly there are the assumptions about which data is known. As X is the data we want to deconvolve, it is always assumed that X is known. However we can posit either knowing M to predict U or knowing U to predict M or knowing neither M nor U and jointly estimating them. Hand in hand with these data assumptions are the preprocessing normalizations and transformations applied to the data by the authors. The second major facet to a deconvolution algorithm is the marker oligos. Marker oligos are expression measurements that are particularly indicative of one cell type over the others. It has been noted almost universally that restricting analysis to these marker oligos, in one way or another, can improve model fit. Thus the manner in which the markers are chosen and applied set apart the deconvolution techniques. Finally, the approach to fitting the linear model distinguishes the methods. Part of the choice when fitting the model is what constraints to enforce. For example, we would like the mixing proportions to be positive and sum to one.

For this paper we are mostly interested in methods that can predict the mixing proportions M. Thus we begin by focusing on methods where U is assumed to be known and M is inferred. We will then talk about methods of estimating U and M jointly without prior knowledge or either. We will not discuss methods that assume the mixing proportions M are known and attempt to predict U.

3.2 Known Characteristic Profiles

Let's look at methods of cell type deconvolution where the expression profile matrix U is known and one is interested in estimating the mixing matrix M. The simplest such algorithms comes from Abbas et al. (2009). Here the authors work with linearly summarized gene level microarray data normalized by the MAS5 algorithm. The cell type expression profiles are gathered either from external databases or created from expression profiles of known pure mixtures. The convolution problem is modeled as in equation 3.1. The mixing matrix M is thus estimated by regressing the convoluted profiles X on the known expression profiles U using by a linear least squares optimization. The fit is done using only a subset of marker oligos (genes in this case) chosen by a combination of biological knowledge and differential expression of genes between the cell types as measured by absolute differences and t-test p-values of the gene expressions. The precise marker genes are then chosen by minimizing the condition number of the resulting sub-matrix of U. To enforce the sum-to-one (STO) and non-negativity (NN) constraints on the rows of the resulting estimate of M a heuristic iterative algorithm is used which involves removing the smallest negative estimated regression coefficient and refitting the model. The rows of the estimated M matrix are then re-scaled such that they sum to one. An implementation of their algorithm in `R` (The R Core Team 2016) is available as part of the `CellMix` package (Gaujoux et al. 2013).

A similar approach to Abbas et al. is taken by Gong et al. (2011) following the model in equation 3.1. However, in this paper Gong et al. work with RMA summarized log-level gene expression data from microarrays as opposed to linearly summarized data. Marker genes are chosen in a similar fashion to Abbas et al. but the model is fit using quadratic programming instead of regression. The quadratic programming framework allows explicit encoding of the NN and STO constraints. An implementation of this algorithm is also available as part of the `CellMix` package.

In a similar fashion the linear model in equation 3.1 is used in Lu et al. (2003) and Wang et al. (2006) to estimate the mixing proportions M using simulated annealing. Lu et al. deconvolve log-level standardized microarray data using their algorithm `DECONVOLUTE` (no implementation currently available). Notably they do not use marker genes although they do mention that such genes may be useful for fitting. Meanwhile Wang et al. (2006) implement a similar algorithm at the linear gene level (normalized by MAS5) and choose to fit only using marker genes. Marker genes are found by differential expression analysis between pure cell type expression profiles using direct comparison, t-test p-values and step-wise discriminant analysis. No implementation of their algorithm is currently available.

Other ways to fit the linear model can be found in Qiao et al. (2012) who implement an algorithm using non-negative least squares. Qiao et al. (2012) work with log-level gene expression data normalized by RMA. The use of non-negative least squares allows them to deal with the NN constraint. The STO constraint is handled by post hoc re-normalizing the rows of the estimated M matrix such that they sum to one. Marker genes are chosen by differential expression comparisons between pure expression profiles of the types. An implementation of their algorithm in `Octave` can be found in the supplemental material of their paper (Qiao et al. 2012).

Newman et al. (2015) and Altboum et al. (2014) fit the linear model using penalized fitting methods.

Altboum et al. use an elastic-net penalization to deconvolve log-transformed RNA-seq data. They compare their method across several ways of choosing marker genes. Their algorithm, named `DCQ` for "digital cell quantification", is available as part of the `ComICS` package in `R` (Gat-viks and Steuerman 2016). On the other hand, Newman et al. solve the model using support vector regression, specifically ν-SVR with a linear kernel. They look at linear normalized gene-level data and, similar to others, choose marker genes looking at differential expressions across groups by p-values and the 2-norm condition number of the resulting cell type profile sub-matrices. The NN constraint is dealt with by zeroing out negatively estimated mixing proportions and the STO constraint is then satisfied by re-scaling the mixing proportions so they sum to one. Their algorithm, dubbed `CIBERSORT`, is available in `R` from their proprietary website (Newman et al. 2015).

Finally there has been some work on Bayesian modeling of deconvolution. While these models are Bayesian they still respect the linear model we posit in equation 3.1. However, being Bayesian, they put priors on quantities we consider fixed, for example, the mixing proportions. The first work in this area was pioneered by Quon and Morris (2009) with their algorithm `ISOLATE`. The model is basically the same as that of Latent Dirichlet Allocation (Blei et al. 2003) but applied to cell type deconvolution. The model is fit by maximum posterior likelihood estimates obtained by the expectation maximization algorithm. The algorithm has since been refined and expanded but the application of an LDA-like Bayesian algorithm still is at the heart of the work (Qiao et al. 2012, Quon et al. 2013). The latest version of the algorithm, `ISOpure` available in a `Matlab` in a supplement to their paper (Quon et al. 2013), deconvolves RMA-normalized log-level gene expression data from microarrays. A unique feature of these Bayesian algorithms is that they work without constraining the analysis to marker genes. Furthermore the NN and STO constraints are taken care implicitly by the estimates coming from the maximum likelihood Bayesian formulation.

There are a couple of points to highlight in closing. Firstly we can see that, in one way or another, all the proposed methods of estimating M from U use the model formulation in equation 3.1. Furthermore in basically all methods, excepting the Bayesian formulation, marker genes are used explicitly in the analysis. However these marker genes are often chosen in an ad hoc manner for each data set considered. The various ways of fitting the linear model are as follows: (1) least squares (2) quadratic programming (3) simulated annealing (4) non-negative least squares (5) regression with elastic-net penalization (6) support-vector regression, or (7) maximum a posteriori estimation. Some of these fitting methods deal with the non-negativity and sum-to-one constraints elegantly and other not so elegantly. Furthermore there is little agreement on how to properly normalize the data. Some methods use well-known algorithms like RMA or MAS5. Others use simpler procedures like mean-centering and standardization. There is generally no agreement on what transformations to apply to the data, for example, whether to work with data on the log level or the linear level.

Finally, as a practical point, at the time of the writing of this paper there are limited implementations currently available for these methods. Gaujoux et al. have implemented the methods of Abbas et al. and Gong et al. in their `R` package CellMix, Altboum et al. have an implementation of their elastic-net based algorithm as part of the `R` package ComICS, Qiao et al. have stand-alone implementations of their Bayesian methods in `Octave` (PERT) and an `R` package (ISOpure) and Newman et al. has `R` code available for CIBERSORT.

3.3 Unknown Profiles and Proportions

The next set of deconvolution algorithms we'd like to talk about are methods of "full deconvolution" (Gaujoux et al. 2013). That is, methods which attempt to predict, simultaneously, both the mixing proportions constituting the matrix M and the characteristic expression profiles defining the matrix U. These are distinct from what Gaujoux et al. call "partial deconvolution" methods which use either knowledge of the matrix U, as in Section 3.2, to predict M or knowledge of M to predict U. While full deconvolution problem uses less information than the partial methods the differences are not as straightforward as it may appear.

One can imagine all deconvolution algorithms on a sliding scale of supervision. On one end of the scale there are highly supervised deconvolution algorithms like that in Section 3.2. There we give the deconvolution methods prototypical examples of the different cells types as encoded in the expression profile matrix U. This matrix U is in some sense the training data for the algorithm. We may also give the algorithms supervisory information in the form of marker genes for each of the cell types. The "full deconvolution" methods we will discuss in this section are on the other end of this supervision spectrum. For these algorithms we do not give detailed supervision in the form of the prototypical expression profiles from the matrix U. Indeed there are such deconvolution algorithms which require neither cell type expression profiles nor marker genes for the cell types. These types of algorithms are truly unsupervised. Unfortunately they are also very under-constrained. Hence, such blind deconvolution is not as useful as one might hope. Indeed even with these algorithms practitioners find the need to *post hoc* look at prototypical expressions for the sake of interpretability thus undercutting the attempt to avoid explicit use of known expression profiles for the cell types. Likewise, authors for most full deconvolution methods find that some level of supervision is needed for the deconvolution algorithms. Thus while the methods mentioned in this section do not make explicit use of characteristic expression profiles they do, mostly, use marker genes. However, this brings up somewhat of a contradiction as such marker gene lists do not appear out of thin air. Indeed most of the cell type marker lists available in the biological literature are created from such pure cell type expression profiles that the full deconvolution algorithms eschew. This is all to say that one should be aware that the claim that these methods operate without knowledge of U is not quite as straight-forward as it may seem. These algorithms use marker genes which are, arguably, some processed form of U. To be fair, however, this level of supervision is less than that of the methods requiring entire cell type expression profiles as discussed in Section 3.2. Furthermore such full deconvolution methods have proven to be very accurate and are some of the most competitive methods for accurate estimation of the mixing matrix M. This is all the more impressive as they use less information (i.e. supervision) than those partial deconvolution methods mentioned heretofore. Furthermore, estimating the profiles, i.e. the matrix U, is a fundamentally important task because it allows practitioners to use such deconvolution techniques to conduct differential expression analysis of expression profiles estimated from separate samples while accounting for the confounding effects of the mixing proportions.

The earliest full deconvolution algorithm, and indeed the earliest computational deconvolution algorithm, is that of Venet et al. (2001). Venet et al., like all others, use the basic linear model described in equation 3.1. The authors work with gene-level summarized microarray data which has been normalized in an ad hoc manner. Notably their algorithm is completely unsupervised using neither gene expression profiles nor marker genes. Instead they posit the non-trivial hypothesis that the number of cell types, K, is known. From here the the matrices U and M are fit by matrix factorization of X. The authors consider two methods of estimation. The first is an sequential two-step algorithm which attempts to minimize the objective $||X - UM||_2^2$. After starting with initial random estimates of U and M it bounces back and forth estimating U and M sequentially using a non-negative least squares optimization algorithm to estimate

U given the current estimate of M and then M given the current estimate of U. The non-negative least squares algorithm allows the algorithm to easily satisfy the NN constraint however at each step they must re-normalize the rows of M such that they sum to one. The second method of matrix factorization that is considered is using principal components analysis or some form of factor analysis to factor X as, approximately, MU. Unfortunately such methods do not respect the STO or NN constraints and furthermore impose orthogonality constraints which are biologically not plausible. Finally, as mentioned previously, since this method is completely unsupervised cell types are not implicitly associated with the estimated expression profiles of U. Much like principal components analysis there is a problem of interpretability of the columns of U. The authors suggest that one looks at the estimated profiles, in U, and associate them with cell types by comparing the estimated profiles to known expression profiles of known cell types. A mapping between estimated profiles and cell types can then be made by determining approximate matches. However such comparisons are not very practical as having expression profiles of known cell types rather undercuts the idea of doing a completely unsupervised cell type deconvolution. No software implementation of this algorithm could be found.

Following the spirit of Venet et al. others have attempted to solve the full deconvolution problem with non-negative matrix factorization. Notably Repsilber et al. (2010) take a very similar deconvolution approach. They too implement a completely unsupervised algorithm using iterated sequential non-negative least squares fitting. They enforce the STO constraint similarly to Venet et al.. Their method is largely an updated version of that of Venet et al.. Repsilber et al. work on gene-level microarray data that has been normalized by limma (Ritchie et al. 2015) and RMA but is considered at the linear level rather than the log level. Their algorithm is implemented in the `R` package `deconf` (Repsilber et al. 2010) or as part of the `CellMix` package (Gaujoux et al. 2013). However, being a completely unsupervised algorithm `deconf` suffers from the same cell-type identification problems as discussed previously. Repsilber et al. suggest a similar approach where one *a posteriori* identifies the estimated expression profiles with cell types by comparing the estimated profiles with the known profiles of known cell types. Not only is this undesirable but, as pointed out by Gaujoux and Seoighe (2012), inclusion of marker gene information actually improves the estimates of U and M.

A comprehensive comparison of non-negative matrix factorization full deconvolution methods is done in Gaujoux and Seoighe (2012). The authors with with linearly summarized (MAS5) gene expression data from microarrays. However these authors make use of marker genes to enforce constraints on the U matrix during fitting so that that marker genes are expressed in only one cell type. The matrix factorization is done with three types of non-negative matrix factorization. The first is due to Lee and Seung (2000) which minimizes the Euclidean distance using gradient descent, the second is from Brunet et al. (2004) which minimizes the Kullback-Leibler divergence and the final from Pascual-Montano et al. (2006) uses a constant smoothing matrix to obtain sparse results. The NN constraint is taken care of by the non-negative matrix factorization algorithms and the STO constraint is achieved by re-normalizing the rows of M post hoc. The method for choosing the marker genes follows Abbas et al. (2009). An implementation of their algorithm is available as part of the `CellMix` package as Gaujoux is the creator and maintainer of that package.

A different full deconvolution approach using marker genes is explored in Zhong et al. (2013). Here the authors follow a two-step algorithm to estimate M and U from X. First they estimate cell type frequencies defining the matrix M. This is done by restricting the model to only include the marker genes instead of all genes. From here constraints on the rows of M to sum to one is exploited to form a system of equations involving the unknown average marker gene expression for each cell type. This system is solved to generate the average marker gene expressions which is then used to estimate the mixing proportions for each cell type, i.e. the matrix M. From this matrix M we can estimate the complete gene expression for each cell type by non-linear least squares solving $\arg\min_U ||X - MU||_2^2$ through quadratic programming. The algorithm is used on gene-level microarray data summarized by RMA. However the authors suggest that one works at the linear scale as opposed to the log-scale. The gene used as markers are chosen from biological databases specific to the cell types in question or through differential expression analysis of known pure samples of the types. The algorithm, dubbed `DSA` for "digital sorting algorithm", is available in the `DSA` package in `R` (Zhong et al. 2013).

The final method which we will cite is that from Liebner et al. (2014). The algorithm put forth by Liebner et al. works with log-level gene-summarized microarray data pre-processed by RMA. If biological knowledge or pure sample expression profiles are available then their method will use this information to curate a set of markers for the various cell types. However the algorithm can work completely unsupervised if desired. In this case the genes with the top 1% of variability in the sample data X are assumed most likely to be differentially expressed across groups and are assigned putatively to a group by k-means clustering. This hypothesis is not necessarily straightforwardly true. Nonetheless, once marker genes are determined the model is fit on the subset of marker genes by a least squares minimization. The method is called "microarray microdissection with analysis of differences" or MMAD and is available as MATLAB code (Liebner et al. 2014).

The concluding remarks for the full deconvolution methods mirror those of the partial methods mentioned in section 3.2 and the comments at the beginning of the section. All of the methods surveyed here use some version of a linear model for solving the full deconvolution problem. While some methods attempt to be completely unsupervised non truly achieve this end. Marker genes obtained either through biological databases or through some pure expression profiles are needed. The marker genes are either used post hoc to map estimated profiles and proportions to cell types or the markers are used to aid in the fitting process of the algorithms. Selection of marker genes seems no more sophisticated than those methods previously presented. Furthermore all fitting methods seem to be some version of non-negative matrix factorization either standard or ad hoc. There is little agreement on the correct transformations or normalizations to use as preprocessing steps. However all methods surveyed here use gene-level microarray data. Helpfully, these full deconvolution algorithms have much more supplementary code and information available. Indeed there are implementations of the algorithms of Gaujoux and Seoighe, and Repsilber et al. in CellMix, an R implementation of DSA by Zhong et al. and a Matlab implementation of MMAD by Liebner et al..

CHAPTER 4

Our Method

Here we wish to put forth some new statistical methodology towards solving the problem of cell-type deconvolution. The primary focus of the development will involve probe-level microarray data. However the methods presented here are very applicable to gene-level microarray data and potentially to other technologies such as RNA-seq.

4.1 A Basic Example

First we will start with a basic deconvolution example to give the reader an understanding of our methodology in a simple setting. Assume that we have three cell samples A, B and C. Furthermore assume that each these samples is a mixture of two cell types. Let A be a sample consisting only of cell of type one, B a sample of only cell type two and C be a mixture of the two cell types. We would like to deconvolve sample C and determine how the cell types are mixed.

Generally, the way a microarray experiment is conducted is that the scientist extracts oligos from the cell samples and dilutes them into an aqueous solution called a "hybridization solution." Simply, this hybridiziation solution is some water-like mixture with the oligos from the sample floating around. Given a hybridization solution, assume that we have microarray technology to measure the expression of N oligos in the solution. Let η_{An} be the concentration of oligo n in the hybridization solution made from sample A. Similarly define η_{Bn} and η_{Cn}. Now since sample C is a mixture of cell types A and B then then its hybridization solution is going to effectively be a mixture of the hybridization solutions of cell types A and B. Thus the concentration of oligo n in the hybridization solution created from sample C is going to be a convex combination of the concentration of oligo n in samples A and B. That is,

$$\eta_{Cn} = p_A \eta_{An} + p_B \eta_{Bn}$$

where p_A and p_B are the proportions the sample A and B hybridization solutions are mixed together to make the sample C hybridization solution. So $p_A, p_B \geq 0$ and $p_A + p_B = 1$.

What we want is a model relating the oligo expressions measured by the microarray to the amount of oligos in the experiment. We assume that the expressions of oligos measured on the microarray are a function of the concentration, or amount, of oligos in solution. To this end we assume a linear model (on a log versus log level) for the relationship between the concentration of oligos in the hybridization solution and the expression value measured by the microarray. Thus, for example, if I_{An} is the probe intensity measurement of the n^{th} oligo given by the microarray experiment on sample A then we assume that

$$Y_{An} \overset{def}{=} \log_2 \left(I_{An} \right) = \theta_n + \gamma \log_2 \left(\eta_{An} \right) + \epsilon_{An}.$$

We assume the same model for the microarray measurements from samples B and C such that

$$Y_{Bn} \overset{def}{=} \log_2 \left(I_{Bn} \right) = \theta_n + \gamma \log_2 \left(\eta_{Bn} \right) + \epsilon_{Bn}$$

and

$$Y_{Cn} \overset{def}{=} \log_2 \left(I_{Cn} \right) = \theta_n + \gamma \log_2 \left(\eta_{Cn} \right) + \epsilon_{Cn}$$

where the quantities are defined similarly. There are couple notes to this model. Firstly the intercept term θ_n is unique to each of the n oligos but is shared among experiments. Secondly the slope of our linear model, γ, is universal and depends neither on the sample A, B or C nor the oligo n. Finally there are error terms $\epsilon_{An}, \epsilon_{Bn}$ and ϵ_{Cn} which we presume are random with mean zero and some finite shared variance σ^2. Furthermore we assume these epsilon terms are mutually independent not only across samples A, B and C but also across the oligos $n = 1, \ldots, N$. Note that we use log base two as is tradition in DNA microarray analysis (c.f. Irizarry et al. 2003). Now that we have rigorously defined the quantities of interest and posited a model between the expressions and oligos we are ready to put forth a deconvolution estimating procedure.

In order to estimate p_A and p_B we need one last piece of information. We need to assume that there is a "marker oligo" for each cell type. For a given cell type we define a marker oligo as an oligo that is expressed in that cell type and not in any other. Thus these marker oligos are indicative of one particular cell type and not any other. In particular we assume there is an oligo n_1 that is expressed by cells of type one only and not of cells of type two. Similarly we assume there is some marker oligo n_2 that is expressed by cells of type two and not type one. In terms of notation this means that we are assuming some marker oligos n_1 and n_2 such that $\eta_{An_2} = 0$ and $\eta_{Bn_1} = 0$. This assumption will be crucial to our deconvolution pursuit because it means that

$$\eta_{Cn_1} = p_A\eta_{An_1} + p_B\eta_{Bn_1} = p_A\eta_{An_1}$$

and similarly

$$\eta_{Cn_2} = p_A\eta_{An_2} + p_B\eta_{Bn_2} = p_B\eta_{Bn_2}.$$

This fact will ultimately allow us to get estimates of p_A and p_B from the microarray expressions by exploiting our linear model. For the type one marker oligo n_1 we have an microarray expression measurement Y_{An_1} from sample A and measurement Y_{Cn_1} from sample C. We modeled these as

$$Y_{An_1} = \theta_{n_1} + \gamma \log_2 (\eta_{An_1}) + \epsilon_{An_1}.$$

and

$$Y_{Cn_1} = \theta_{n_1} + \gamma \log_2 (\eta_{Cn_1}) + \epsilon_{Cn_1}.$$

However we know that $\eta_{Cn_1} = p_A\eta_{An_1}$ hence

$$\begin{aligned} Y_{Cn_1} &= \theta_{n_1} + \gamma \log_2 (p_A\eta_{An_1}) + \epsilon_{Cn_1} \\ &= \theta_{n_1} + \gamma \log_2 (p_A) + \gamma \log_2 (\eta_{An_1}) + \epsilon_{Cn_1} \end{aligned}$$

This means that

$$Y_{Cn_1} - Y_{An_1} = \gamma \log_2 (p_A) + \epsilon_{Cn_1} - \epsilon_{An_1}$$

hence

$$\exp_2\left(\frac{Y_{Cn_1} - Y_{An_1}}{\gamma}\right) = \lambda_{n_1} p_A \text{ where } \lambda_{n_1} \overset{def}{=} \exp_2\left(\frac{\epsilon_{Cn_1} - \epsilon_{An_1}}{\gamma}\right).$$

The term λ_{n_1} is a term that depends on the random errors which ends up as a multiplicative error term once we exponentiate. Thus if we can find some estiamtor $\widehat{\gamma}$ of γ we can form the estimator

$$\widehat{q_A} = \exp_2\left(\frac{Y_{Cn_1} - Y_{An_1}}{\widehat{\gamma}}\right)$$

with the hopes that if $\widehat{\gamma} \to \gamma$ then $\widehat{q_A} \to \lambda_{n_1} p_A$. In a similar fashion we can define $\widehat{q_B} = \exp_2\left(\frac{Y_{Cn_2} - Y_{Bn_2}}{\widehat{\gamma}}\right)$. Importantly, however, we cannot guarantee that $\widehat{q_A}, \widehat{q_B} \leq 1$ nor that $\widehat{q_A} + \widehat{q_B} = 1$. Firstly, even if our data exactly follows the linear model we assumed we can't guarantee that the lambda terms λ_{n_1} and λ_{n_2} don't perturb the estimates to be bigger than one. Furthermore, the two lambda terms are assumed independent so they can move p_A and p_B independently in ways such that they do not sum to one. Secondly, as a matter of practicality, our data will not exactly follow our linear model and so these estimates do not need to be as well behaved in practice as in theory. At most we can guarantee that $\widehat{q_A}$ and $\widehat{q_B}$ will be positive since they are both exponential terms. Ensuring that our estimators of p_A and p_B satisfy the sum-to-one and non-negativity constraints of a proportion estimator is important so we re-normalize $\widehat{q_A}$ and $\widehat{q_B}$ estimators and define

$$\widehat{p_A} = \frac{\widehat{q_A}}{\widehat{q_A} + \widehat{q_B}}$$

and similarly

$$\widehat{p_B} = \frac{\widehat{q_B}}{\widehat{q_A} + \widehat{q_B}}$$

as our final estimators of p_A and p_B respectively. Since $\widehat{q_A}$ and $\widehat{q_B}$ must be positive then $\widehat{p_A}$ and $\widehat{p_B}$ will be positive and sum to one.

In this section we have discussed a simplified version of the cell type deconvolution problem. Next we will discuss in more detail the linear modeling of the oligo concentrations and expressions that is at the heart of our method. We will then develop a more comprehensive deconvolution method that can incorporate multiple marker oligos and multiple cell types. In closing we will highlight how this model differs from those existing in the literature.

4.2 Concentrations and Expressions

Let's look concretely at the relationship between the concentration of oligos in our cell samples and the expression measurements the DNA microarray produces. To do this we will look at the Affymetrix Latin Square data set (Affymetrix). This data set consists of 14 microarray experiments (with three technical replicates each for a total of 42 experiments) where known concentrations of 42 different transcripts were spiked into a hybridization solution containing a complex human RNA background mix. These transcripts were spiked in at known concentrations ranging from 0.125pM to 512pM. This data set is very useful because for a handful of oligos we know the concentration and can look correspondingly at their measured expressions.

Let's work with un-summarized probe level data at the log level. That is, in this case the oligos $n = 1, \ldots, N$ are the individual probes on the microarrray. The microarray experiments reported in this data set were done using a U133A Affymetrix GeneChip and thus measure $N = 506,944$ probes per experiment. For each of these probes the microarray data consists of an intensity measurement I_n. We are going to work with $\log_2$ level data doing analysis with the $\log_2$ probe measurements $Y_n \overset{def}{=} \log_2(I_n)$. In Figure 4.1 we have plotted the log intensity measurement, Y_n, against the corresponding log concentration for oligo $n = 479376$. There is nothing special about this oligo it simply exemplifies the relationship between expression and concentration.

We can see from looking at this figure that there is a positive relationship (on the log-log level) between the concentration and expression level of the probe. Notably, however, the relationship is not linear. Indeed the curve tends to flatten out for very high and very low concentrations. One might imagine that the true relationship on this scale is something of a sigmoid curve. We fit such a curve to our data in Figure 4.2.

This simple sigmoid curve fits the expression-concentration data very well on the log-log level. The important features of this graph are the flattened tail below about zero, the flattened tail above about eight and, since the sigmoid curve is smooth, the approximately linear relationship between the two tails. In retrospect, this sort of curve isn't too surprising. Indeed some sort of sigmoid curve seems likely given the constraints of the fluorescently activated probes. The probes have a finite maximum amount they can fluorescently shine and hence a saturation point beyond which increasing concentration does not increase expression measurement. This is the right-hand plateau after the $\log_2$ concentration rises above about 8.

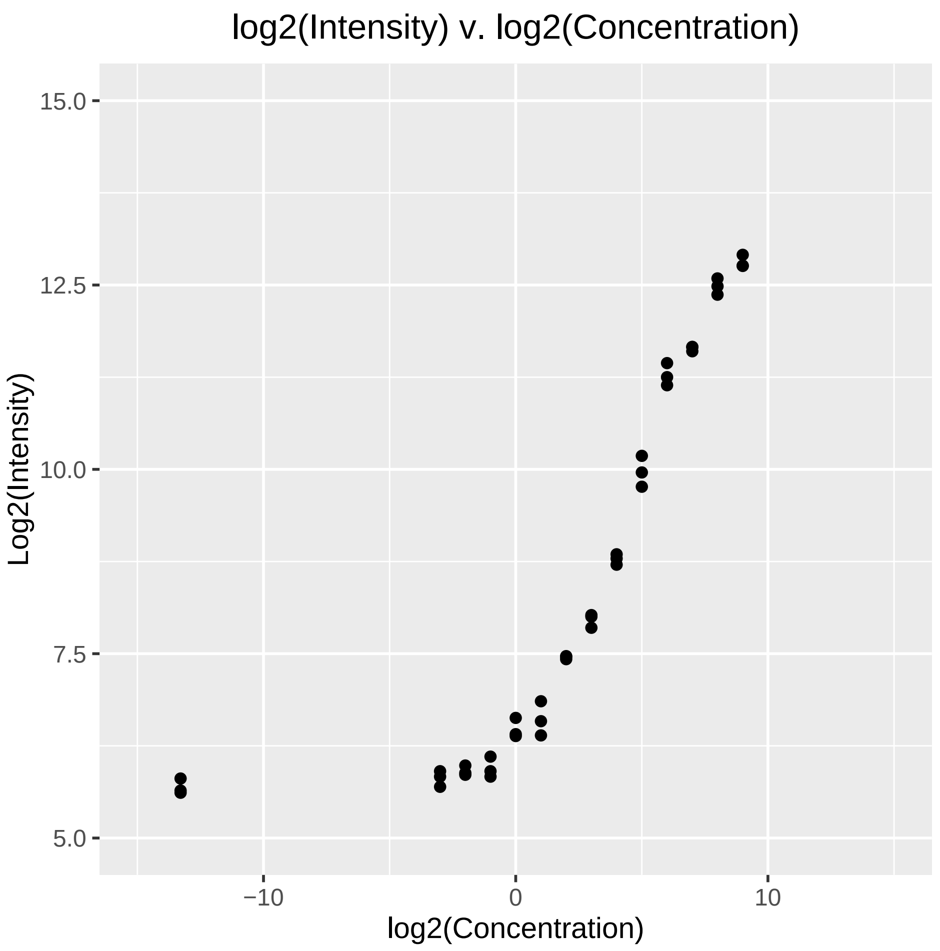


Figure 4.1: Expression of probe 479376 against concentration on a log − log level.

On the lower end there is a flat tail on the left-hand side. This is likely because there is some background fluorescence off the microarray probes even when none of the corresponding oligos are present. This might be caused by phenomena like competitive cross hybridization. Thus even as the concentration of the oligo approaches zero the log intensity doesn't go to negative infinity because there is background fluorescence. Thus we see, instead, the the expression measurement plateau's for low concentrations as well as for high. Both of these phenomena mean that the expression measurement is bounded. Thus the most simple and smooth manner for which the concentration to move from the lower bound to the upper bound as we increase concentration is going to look something like a sigmoid curve. This likely explains why we see such a good fit. Obviously there is no guarantee that there be such a nice relationship between expression and concentration however given the simple continuous fluorescence process it's not too surprising, either.

While this sigmoid curve fits out data quite well we do not use a sigmoidal model in our deconvolution

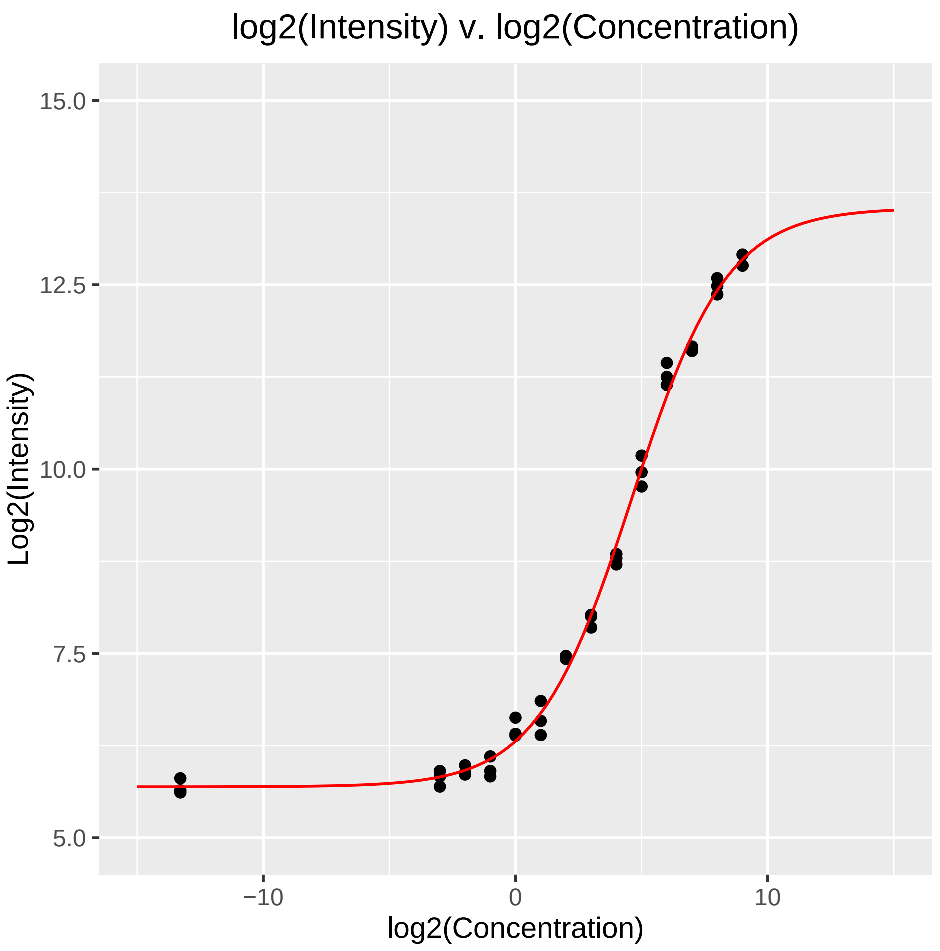


Figure 4.2: Expression of probe 479376 against concentration on a $\log - \log$ level. Sigmoid curve fit to points.

method. Remember that we modeled the $\log_2$ expression, Y_n, as linear in the $\log_2$ concentration such that

$$Y_n = \log_2(I_n) = \theta_n + \gamma \log_2(\eta_n) + \epsilon_n$$

where η_n is the concentration of oligo n in the hybridization solution, θ_n is the oligo-specific intercept, γ is the slope for all oligos and ϵ_n is the oligo-specific error term. We plot such a linear curve for our data in Figure 4.3.

The linear fit in Figure 4.3 captures the relationship between the expression and concentration quite well so long as the concentration is not too high nor too low. Obviously the linear fit is not quite as good as the sigmoidal fit however there are some advantages. In the figures plotted in this section we assume we know both concentration and expression. However in a real deconvolution setting we know only

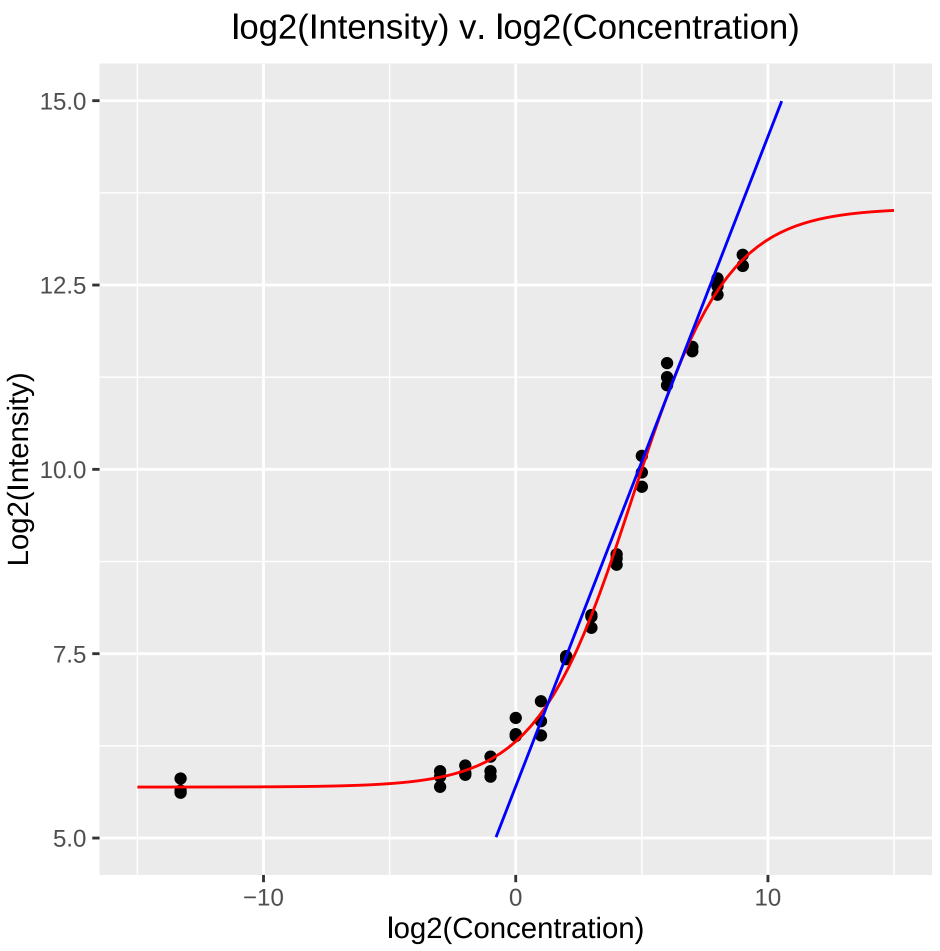


Figure 4.3: Expression of probe 479376 against concentration on a log − log level. Sigmoid fit in red, linear fit in blue.

the expression and wish to determine the concentration. Indeed our deconvolution method involves using the posited relationship to go from expressions to concentrations. Unfortunately the sigmoidal curve is not a very well behaved curve in this direction. As we increase our expression towards the maximum of our sigmoidal curve the concentration as predicted by the sigmoidal curve will increase without bound. At such high levels of expression small changes in expression will increase the predicted concentration drastically. If instead of working with the red sigmoidal curve we look at the blue linear fit then we see that the predicted concentration does not increase very quickly without bound at such high levels. Indeed it increases linearly with a slope of γ at all points. One might view the blue linear fit as some sort of regularized version of the sigmoidal curve which keeps the model from behaving poorly at very high expressions. In a similar sense the linear fit is regularized for low expressions. An expression of, say, five would be predicted as a $\log_2$ concentration near negative infinity from the sigmoidal fit whereas the linear fit predicts a $\log_2$ concentration around zero. Thus our linear fit seems to track the true sigmoidal relationship when the expressions are neither very high nor very small. However for edge cases the linear fit regularizes our estimator keeping the a very highly expressed oligo from dominating the procedure by estimating a very high concentration. At the extreme we see that the red sigmoidal curve has a finite maximum and minimum. Thus if we encounter an expression above or below these extrema our sigmoidal model doesn't give us a straight-forward way of recovering concentration estimates since the curve is not invertible for all real numbers. Our linear model doesn't encounter such problems. For these reasons we choose to model the relationship between concentration and expression in a linear fashion.

4.3 A General Model

Now that we have discussed the manner in which we model the relationship between concentration and expressions let's develop a general model for deconvolution of several cell types by multiple marker oligos.

Let's assume that we have K cell types under consideration and microarray technology which can measure N oligos. Define η_{kn} to be the concentration of oligo n in hybridization solution produced by cells of type k where $k \in [\![1, K]\!]$ and $n \in [\![1, N]\!]$. Assume that for each cell type $k = 1, \ldots, K$ we have ν_k pure samples of cells of type k only. For each of these samples we generate hybridization solution and run a microarray experiment. From this we get microarray data

$$Z_{kr} \in \mathbb{R}^{1 \times N}$$

for $k = 1, \ldots, K$ and $r = 1, \ldots, \nu_k$. Let's assume that these Z_{kr} are $\log_2$ transformed as previously discussed. In this case then our linear model relating the concentrations η_{kn} and the expression measurements Z_{krn} tells us that

$$Z_{krn} = \theta_n + \gamma \log_2 (\eta_{kn}) + \epsilon_{krn}$$

where Z_{krn} is the measurement of the n^{th} oligo in the r^{th} pure sample of type k and the ϵ_{krn} are mutually indpendent across k, r and n.

Now assume we also have some heterogeneous sample that is a mixture of the K cell types. This is the sample we would like to deconvolve. Let the hybridization solution of our heterogeneous sample be a mixture of pure sample hybridization solutions with mixing proportions $p_1, \dots, p_K$. Then if c_n is the concentration of the n^{th} oligo in our heterogeneous sample we have that

$$c_n = \sum_{k=1}^{K} p_k \eta_{kn}.$$

We run DNA microarray analysis on the heterogeneous sample's hybridization solution and get $Y \in \mathbb{R}^{1 \times N}$, the $\log_2$ level expressions of oligos $1, \dots, N$ in the heterogeneous sample. Then our linear model tells us that Y_n, the n^{th} oligo expression measurement from this heterogeneous sample, is related to c_n as

$$Y_n = \theta_n + \gamma \log_2 (c_n) + \epsilon_n.$$

We assume that the ϵ_n and the ϵ_{krn} from the pure samples are mutually independent with mean zero and some finite fixed variance σ^2.

In order to deconvolve the heterogeneous sample assume there are some number of marker oligos for each cell type $k \in [\![1, K]\!]$. Let $G_k \subset [\![1, N]\!]$ be the set of marker oligos for cell type k such that if $n \in G_k$ then $\eta_{tn} = 0$ for all $t \neq k$. That is, if n is a marker oligo of cell type k then it is not expressed in any other cell type than k. Assume that the marker oligos are mutually exclusive across cell types such that $G_k \cap G_t = \emptyset$ for all $t \neq k$.

Now consider some oligo n that is a marker of cell type k such that $n \in G_k$. Then we have that

$$c_n = \sum_{t=1}^{K} \eta_{tn} p_t = \eta_{kn} p_k$$

as $\eta_{tn} = 0$ for all $t \neq k$. That is, if n is a marker oligo of cell type k then the concentration of n in the heterogeneous sample is simply the concentration of n in any of the pure type-k samples, η_{kn}, scaled down by the mixing proportion p_k.

Thus if $n \in G_k$ we have that

$$\begin{aligned} Y_n &= \theta_n + \gamma \log_2 (c_n) + \epsilon_n \\ &= \theta_n + \gamma \log_2 (\eta_{nk} p_k) + \epsilon_n \\ &= \theta_n + \gamma \log_2 (\eta_{nk}) + \gamma \log_2 (p_k) + \epsilon_n \end{aligned}$$

and

$$\overline{Z_{kn}} \overset{def}{=} \frac{1}{\nu_k}\sum_{r=1}^{\nu_k} Z_{krn} = \frac{1}{\nu_k}\sum_{r=1}^{\nu_k}\left(\theta_n + \gamma\log_2\left(\eta_{kn}\right) + \epsilon_{krn}\right)$$

$$= \theta_n + \gamma\log_2\left(\eta_{kn}\right) + \overline{\epsilon_{kn}}$$

where $\overline{\epsilon_{kn}} \overset{def}{=} \frac{1}{\nu_k}\sum_{r=1}^{\nu_k}\epsilon_{krn}$. Thus

$$Y_n - \overline{Z_{kn}} = \gamma\log_2\left(p_k\right) + \epsilon_n - \overline{\epsilon_{kn}}$$

and so if we average such terms over all marker oligos $n \in G_k$ we get

$$\begin{aligned} E_k &\overset{def}{=} \frac{1}{\Gamma_k}\sum_{n\in G_k}\left(Y_n - \overline{Z_{kn}}\right) \\ &= \frac{1}{\Gamma_k}\sum_{n\in G_k}\left(\gamma\log_2\left(p_k\right) + \epsilon_n - \overline{\epsilon_{kn}}\right) \\ &= \gamma\log_2\left(p_k\right) + \frac{1}{\Gamma_k}\sum_{n\in G_k}\left(\epsilon_n - \overline{\epsilon_{kn}}\right) \end{aligned}$$

where $\Gamma_k \overset{def}{=} |G_k|$. If we knew γ then we could solve for p_k, approximately, as

$$\exp_2\left(\frac{E_k}{\gamma}\right) = \lambda_k p_k$$

where $\lambda_k = \exp_2\left(\frac{1}{\Gamma_k}\sum_{n\in G_k}\left(\epsilon_n - \overline{\epsilon_{kn}}\right)\right)$ is some multiplicative error term. Since we don't know γ let's suppose we can estimate γ by some estimator $\widehat{\gamma}$ and then plug into the previous equation to get an estimator

$$\widehat{q_k} = \exp_2\left(\frac{E_k}{\widehat{\gamma}}\right)$$

with the hopes that if $\widehat{\gamma} \to \gamma$ then $\widehat{q_k} \to \lambda_k p_k$.

Ideally these $\widehat{q_k}$ would be our final estimators of the p_k. However in reality there is no guarantee that the collection of these $\widehat{q_k}$ are bounded above by one and sum to one. So we opt instead to re-normalize the $\widehat{q_k}$ and define the estimators

$$\widehat{p_k} = \frac{\widehat{q_k}}{\sum_{t=1}^{K}\widehat{q_t}}$$

as our final estimators of the p_k. These $\widehat{p_k}$ are all non-negative and sum to one as desired.

In the next section we will contrast our estimators with those discussed in the literature. Then we will dig into the details of how to estimate γ and how to choose the marker oligos.

4.4 Comparison

Let's compare our method to the most similar methods existing in the deconvolution literature. Our method estimated the mixing proportions p_k through relationships between the p_k, Y, and the Z_{kr} where Y and Z_{kr} are log-level expression measurements. Some of the methods in the literature worked at the linear level working instead with X and W_{kr} such that $Y_n = \log_2(X_n)$ and $Z_{krn} = \log_2(W_{krn})$ so that X and the W_{krn} are the linear-level expression measurements. These methods would use the microarray expressions from pure samples, the W_{kr}, to choose marker genes and generate pure cell type expression profiles as encoded in the profile matrix $U \in \mathbb{R}^{K\times N}$. One obvious way to create U is to let the k^{th} row of U be the average of W_{kr} from $r = 1, \ldots, \nu_k$. That is,

$$U_k = \frac{1}{\nu_k}\sum_{r=1}^{\nu_k} W_{kr}$$

so that U is the matrix of characteristic expression profiles of the K types in so much as row k is the typical expressions of the N oligos in a pure sample of cell type k. From here the model assumes that $M = (p_1, \ldots, p_K) \in \mathbb{R}^{1\times K}$ and posits the relationship

$$X = MU + E$$

where $E \in \mathbb{R}^{1\times N}$ is some random matrix of errors. Thus,

$$X_n = \sum_{k=1}^{K} p_k U_{kn} + e_n$$

so that the model posits that for each oligo n the expression measurement in the heterogeneous sample, X_n, is a convex combination of the expression of this oligo in each of the pure samples (i.e. the characteristic profiles) U_{kn} with weights as the mixing proportions p_k and some error term e_n. If we log-transform both sides then we get that the model posits

$$Y_n = \log_2 (X_n) = \log_2 \left(\sum_{k=1}^{K} p_k U_{kn} + e_n \right). \tag{4.1}$$

Compare this to the linear model our method posits. We model the relationships as

$$c_n = \sum_{k=1}^{K} p_k \eta_{kn}$$

and

$$Y_n = \theta_n + \gamma \log_2 (c_n) + \epsilon_n$$

so that

$$Y_n = \theta_n + \gamma \log_2 \left(\sum_{k=1}^{K} p_k \eta_{kn} \right) + \epsilon_n. \tag{4.2}$$

There are striking simliarities between a standard model posited in the literature in Equation 4.1 and our model in Equation 4.2. Both equations model the Y_n as the log of a convex combination with weights that are the mixing proportions $p_1, \ldots, p_K$. However there are several important differences between the methods. First is the placement of the error terms e_n and ϵ_n. Secondly, there are differences in the terms in the convex combination. Thirdly, our model includes slope and intercept terms γ and θ_n. Finally, although not aparent from the modeling equations alone, there are differences in fitting methods among the models.

The differenes in the placements of the error terms is not very important. This essentially arises because one model is fit at the linear level and the other is fit on the log level. More importantly, most of the methods in literature do ont explicitly include a random error term. We add an error term to their models assuming that since they estimate the p_k by by minimizing terms like $||X - MU||$ over M they are implicitly assuming some sort of linear error term at the linear level. The authors put very little emphasis on such terms and hence focusing on something the authors do not explicitly state does not seem important.

The second difference between our model and a typical model in the literature is that while our model posits Y_n is dependent on the log of the convex combination $\sum_{k=1}^{K} p_k \eta_{kn}$ those models in the literature instead assume that Y_n is dependent on the log of the convex combination $\sum_{k=1}^{K} p_k U_{kn}$. The difference here being that the models in the literature assume some sort of direct relationship of Y_n on the characteristic *expressions* U_{kn} while our model posits a relationship of Y_n on the *concentrations* of such oligos in the pure sampes, η_{kn}. Thus our model uses concentrations while theirs use expressions. Now obviously the characteristic expression profiles U_{kn} are typical generated from such pure sample expressions W_{krn} and thus themselves depend on the concentration of the oligos η_{kn}. However the oligo expressions W_{krn} and the oligo concentrations η_{kn} are not interchagable. Most methods in the literature seem to assume that they are. That is, most methods to date assume that the expression measurements from the microarrays are indistinguishable from the concentration of those oligos in the samples. The implicit assumption is that the expressions directly give us the concentration of the oligos. Obviously there is a relationship between the expressions and the concentrations. However in our method we separate the two quantites modeling the expressions as a linear function of the concentrations. We believe that this distinction is crucial to accurate modeling of the convolution phenomena. Effectively most methods in the literature assume that $W_{krn} = \eta_{krn}$ such that $Z_{krn} = \log_2(\eta_{kn})$ while in our method we model the relationship as $Z_{krn} = \theta_n + \gamma \log_2(\eta_{kn}) + \epsilon_{krn}$ including an intercept θ_n and a slope γ to explain the relationship. This difference in modeling can be seen again in the modeling equation of Y_n. In Equation 4.2 we include the slope and intercept when modeling the dependence of Y_n on $\log_2\left(\sum_{k=1}^{K} p_k \eta_{kn}\right)$ while the prototypical method in the literature typified by Equation 4.1 directly set Y_n equal to $\log_2\left(\sum_{k=1}^{K} p_k U_{kn}\right)$. Thus our model is more complex than those in the literature because it introduces intercept parameters θ_n and a slope parameter γ. Notice that if we set each of these θ_n to zero and γ to be one then we get something very similar to Equation 4.1.

While there are some similarities in the models used by methods in the existing literature and those used in our method there are big differences in how the two models are fit. Most of the methods in the literature view the p_k as parameters in a linear model and estimate the p_k according to some classic method like least squares or convex optimization. For our technique we do not view the mixing proportions as parameters in a linear model. Instead we estimate our slope parameter γ from an independent data set and then solve for the mixing proportions directly in the model using information from the pure samples to account for the θ_n and the η_{kn}. The fitting method is much different than any of the classic methods used in the literature. Furthermore our method distinguishes itself from other algorithms by the data on which it works. Most other methods work on data that has been highly pre-processed by algorithms like RMA or MAS5. Those algorithms summarize the low-level probe data into gene-level data. While looking at gene-level data in this way can be helpful our method does not require any such preprocessing. Instead we work with very low level raw intensity measurements from the microarrays at the level of probes on the array rather than summarizations of probes into genes.

Finally it should be noted that the conversation in this section was based upon algorithms that modeled the expressions as linear at the linear (non-log) level. The models used such linear-level algorithms are most comparable to our own. Thus our conversation has focused upon them. However fully half of the algorithms we reviewed in the literature were not such models. Indeed many methods modeled expressions as linear relationships at the log level. These types of algorithms are even more different to our own. Briefly, log-level models typical model the expression as something along the lines of

$$Y_n = \sum_{k=1}^{K} p_k \log_2 (U_{kn}) + e_n$$

which is a completely different model than ours. Adding an intercept θ_n and slope γ will not reconcile the differences between these log-based models and ours since the summation and parameters p_k have been brought outside the log.

In conclusion then there are some methods which model expressions as linear models at the linear level. These are similar to our algorithm however they use a much simpler model than our own. There are also methods which model expressions as linear at the log level. These models are fundamentally different than the model we propose. In any case, algorithms at both the linear and log levels fit their parameters as classic linear models which is much different to our estimation procedure.

4.5 Estimating γ and Choosing Markers.

The estimators $\widehat{p_k}$ depend on being able to estimate γ and being able to choose marker oligos for each of the types. Here we will discuss the details of how those tasks are accomplished.

4.5.1 Estimating γ

Now the parameter γ is a global parameter which does not depend on which oligo measurement we are considering. Indeed if the precise pre-processing of our data Y is fixed then this parameter can be estimated once for all data sets. The ability to do such an estimation depends on the existence of a data set linking the concentration (or amounts) of oligos in a data set with the expression measurements by the profiling technology.

For technology like DNA microarrays there are many such data sets from which we can estimate γ. In particular we use the Latin Square data set (Affymetrix). In this experiment a DNA microarray was performed on several samples in which were spiked in a common background human RNA cocktail as well as particular oligonucleotides at known concentrations. A simple linear regression model was then fit to trimmed data regressing expression levels on concentration for those measurements with known concentration. The estimated slope from this model was then used as $\widehat{\gamma}$. This method is very simple and can definitely be improved. It also requires that there be data sets with known concentrations and expressions. Nicely, however, γ needs only be estimated once for each technology and normalization regime.

4.5.2 Choosing Markers

The last step in describing our complete deconvolution algorithm is to describe the manner in which marker oligos are chosen. As discussed in Chapter 3 there is quite a bit of variety in how marker genes are chosen. Here we decide to leverage the DNA microarray experiments on pure samples, those Z_{kr}, to determine which oligos are indicative of one, and only one, type. To do this let

$$Z_n = ((Z_{11})_n, \ldots, (Z_{1R_1})_n, (Z_{21})_n, \ldots, (Z_{2R_2})_n, \ldots, (Z_{K1})_n, \ldots (Z_{KR_K})_n)$$

be the measurements of the n^{th} oligo across all the Z_{kr}. Similarly let X_k be the indicator for Z such that $(X_k)_i = 1$ if and only if $(Z_n)_i$ is a measurement coming from a pure cell sample of type k. Precisely, we say that if $R_{k^-} = \sum_{i=1}^{k-1} R_i$ then $(X_K)_i = 1$ if $R_{k^-} < i \leq R_{(k+1)^-}$ and zero elsewhere. For each n and k we fit the simple linear regression of Z_n on X_k and estimate a slope $\widehat{\beta_{kn}}$. Using these slopes we assign a top type to each oligo n defining sets τ_k of the top oligos for each cell type k so that

$$\tau_k = \left\{ n \,\middle|\, k = \arg \max_{t=1,\ldots,K} \widehat{\beta_{tn}} \right\}$$

so that $\tau_k \subset [\![1, n]\!]$ is the set of oligos for which cell type k has the largest slope among the regressions. Given some parameter M, the number of marker oligos we wish to use, we then define the marker oligos for type k as

$$G_k = \left\{ n \in \tau_k \middle| \widehat{\beta_{kn}} \geq \widehat{\beta_{k(M)}} \right\}$$

where $\widehat{\beta_{k(M)}}$ is the M^{th} order statistic of the set $\left\{ \widehat{\beta_{kn}} \,|n \in \tau_k \right\}$. This is to say that G_k, the set of marker oligos for cell type k, is comprised of the set of oligos n for which the slope $\widehat{\beta_{kn}}$ is one of the top M slopes among all oligos where cell type k has the largest slope across all cell types.

The slopes from our regression of Z_n on the indicator X_k tell us about how much the oligo n differentiates cell type k from all other oligos. Thus we pick those oligos to be markers if they are the best at discriminating between cell type k and the other types. Implicit in having a large regression slope is that the oligo n must be much more expressed in cell type k than other types. Thus satisfying the definition of a marker oligo. There are definitely simpler ways of choosing marker oligos which give good results. However our hope is to extend and robustify this marker oligo selection using methods like RUV (Gagnon-Bartsch and Speed 2012). In this light, then, the regression framework is useful. Choosing the marker oligos is a very important step in any deconvolution algorithm thus much of our future work will focus on exploring precisely how to do this well. This includes determining how many marker oligos, M, to pick for each cell type.

CHAPTER 5

Analysis on Real Data

5.1 Methods and Implementations

our method – say it's available on github – basically just choose 1000 marker

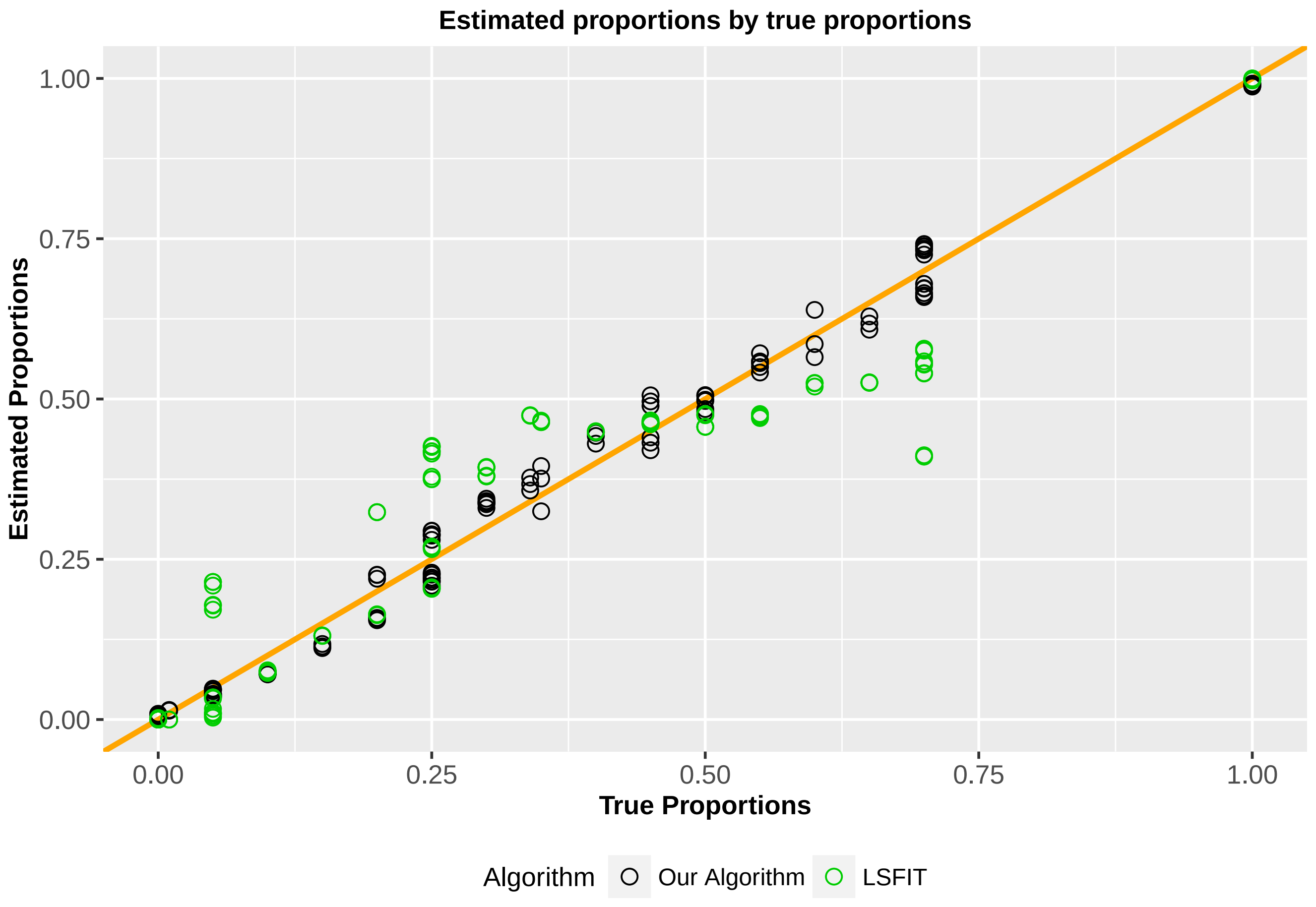
their methods – mention methods and citations – say cell mix – say how genes are chosen – say linear level – appendix for code

5.2 The Rat Data Set

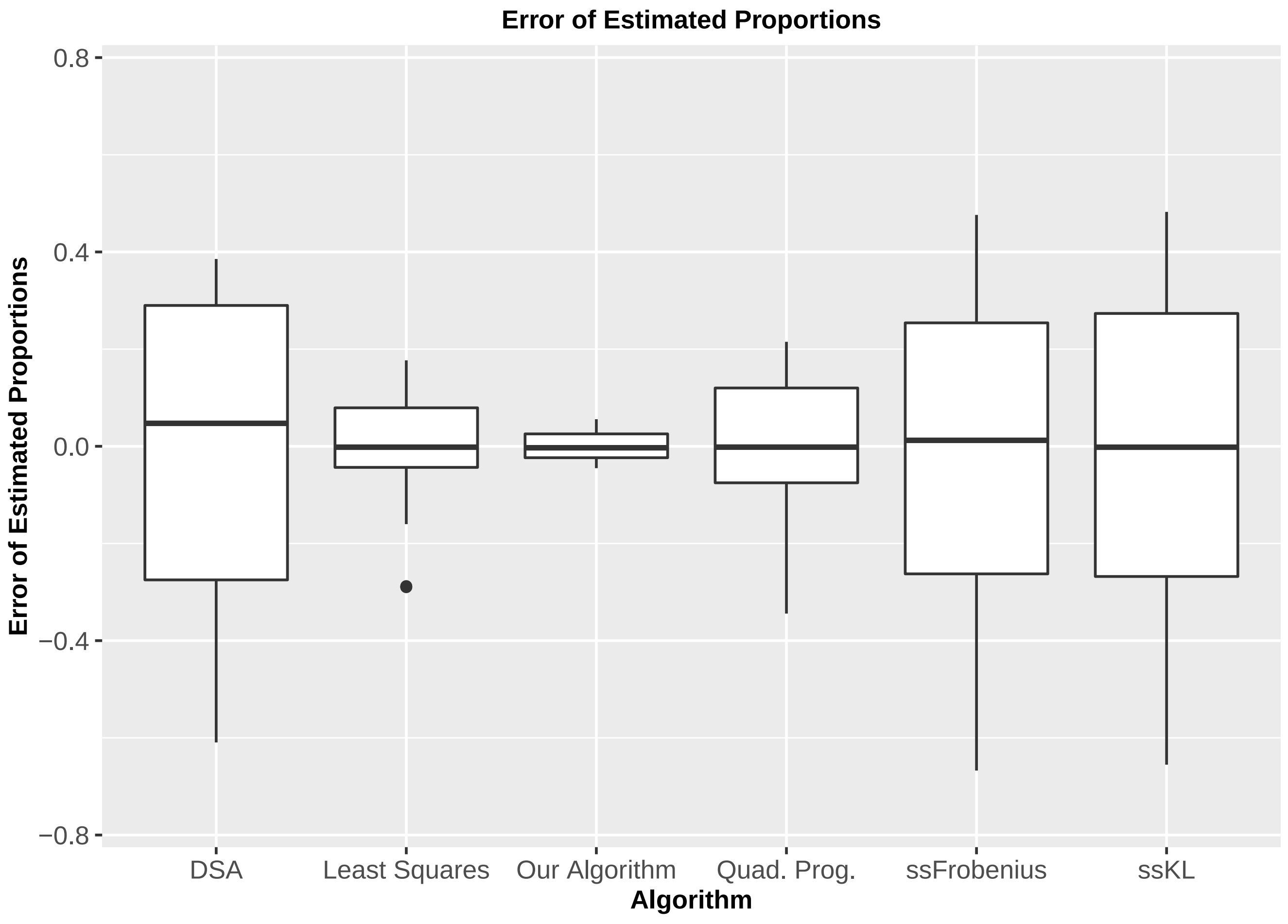
Percentile	Our Algorithm	DSA	Least Squares	Quad. Prog.	ssKL	ssFrobenius
0%	0	0.003	0	0	0.002	0.004
25%	0.009	0.102	0.017	0.022	0.11	0.109
50%	0.025	0.28	0.046	0.1	0.269	0.255
75%	0.036	0.358	0.124	0.162	0.346	0.341
100%	0.056	0.76	0.29	0.344	0.655	0.667

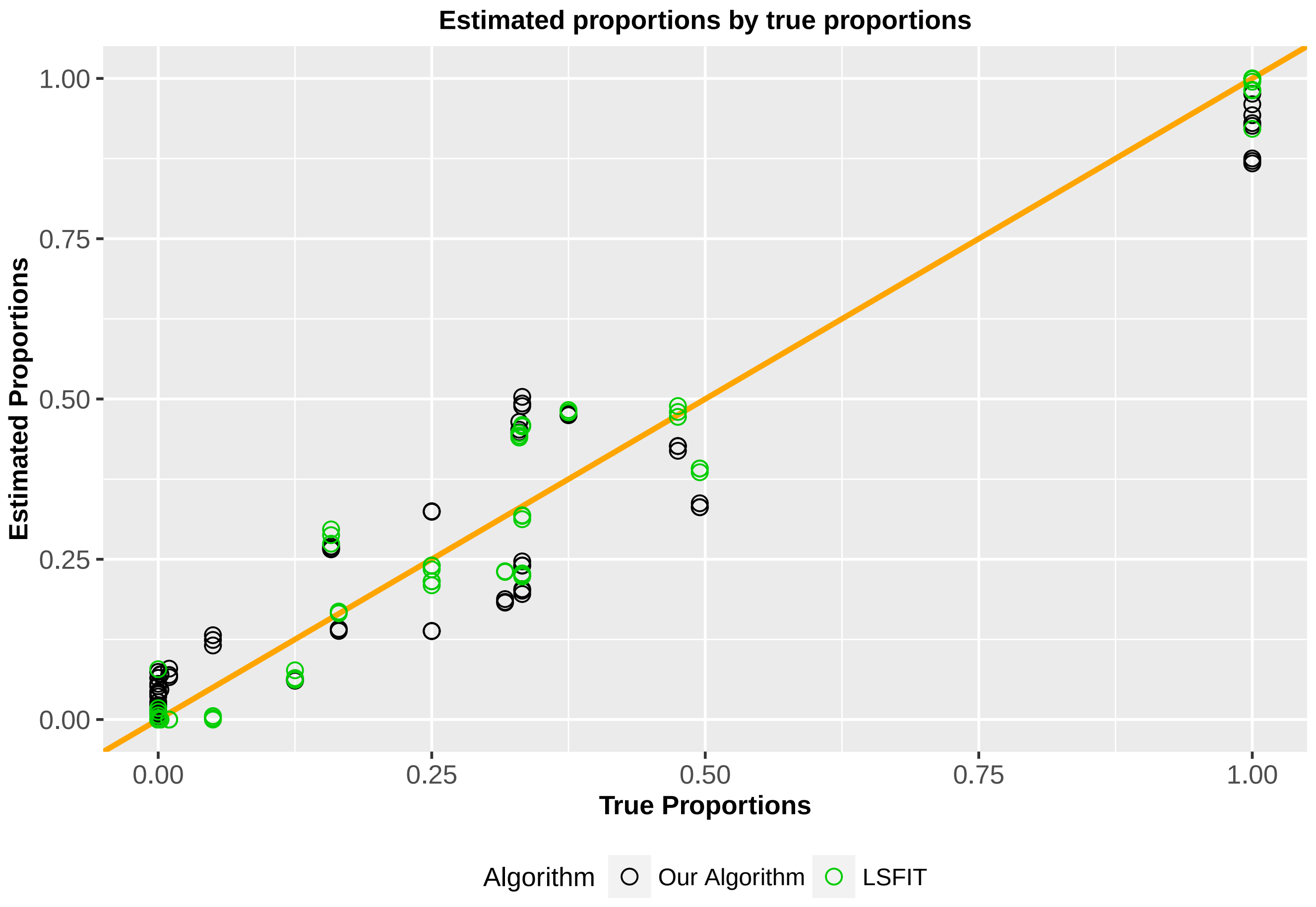
5.3 The Blood Data set

Percentile	Our Algorithm	DSA	Least Squares	Quad. Prog.	ssKL	ssFrobenius
0%	0.006	0.001	0	0	0.02	0
25%	0.034	0.046	0.001	0.001	0.049	0.052
50%	0.068	0.077	0.013	0.015	0.076	0.076
75%	0.112	0.123	0.086	0.109	0.137	0.187
100%	0.171	0.451	0.139	0.191	0.435	0.789

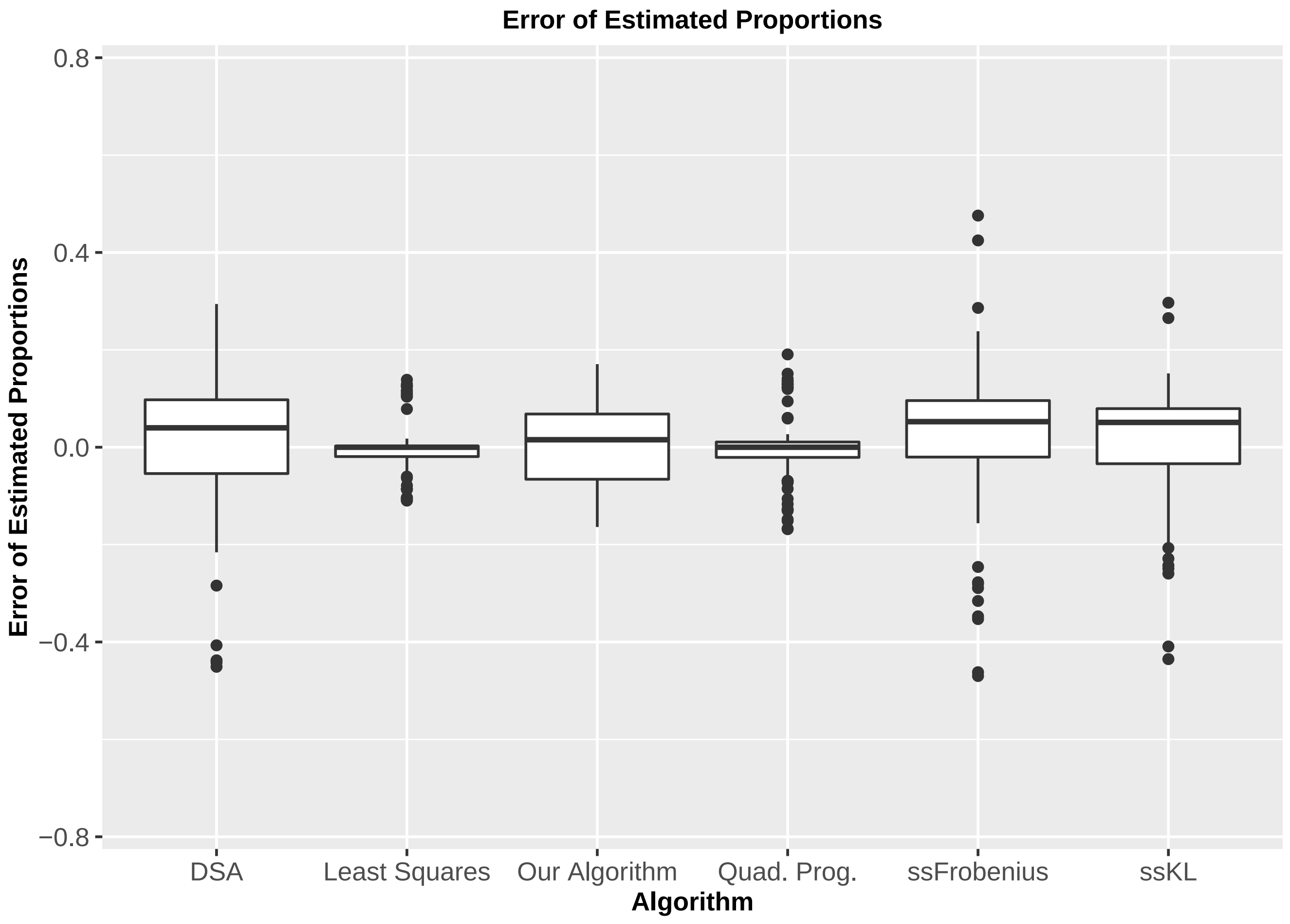


Estimated proportions by true proportions
Estimated Proportions
True Proportions
0.00
0.25
0.50
0.75
1.00
Algorithm
Our Algorithm
LSFIT


Error of Estimated Proportions
0.8
0.4
0.0
−0.4
−0.8
Error of Estimated Proportions
DSA
Least Squares
Our Algorithm
Quad. Prog.
ssFrobenius
ssKL
Algorithm



Estimated proportions by true proportions
Estimated Proportions
1.00
0.75
0.50
0.25
0.00
True Proportions
0.00
0.25
0.50
0.75
1.00
Algorithm
○ Our Algorithm
○ LSFIT


Error of Estimated Proportions
0.8
0.4
0.0
−0.4
−0.8
Error of Estimated Proportions
DSA
Least Squares
Our Algorithm
Quad. Prog.
ssFrobenius
ssKL
Algorithm

CHAPTER 6

Conclusion

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

6.1 Future Work

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

BIBLIOGRAPHY

A. R. Abbas, K. Wolslegel, D. Seshasayee, Z. Modrusan, and H. F. Clark. Deconvolution of blood microarray data identifies cellular activation patterns in systemic lupus erythematosus. *PLoS ONE*, 4(7), 2009. ISSN 19326203. doi: 10.1371/journal.pone.0006098.

Affymetrix. Latin square data for expression algorithm assessment.

Z. Altboum, Y. Steuerman, E. David, Z. Barnett-Itzhaki, L. Valadarsky, H. Keren-Shaul, T. Meningher, E. Mendelson, M. Mandelboim, I. Gat-Viks, and I. Amit. Digital cell quantification identifies global immune cell dynamics during influenza infection. *Molecular Systems Biology*, 10(2):1–14, 2014. ISSN 17444292. doi: 10.1002/msb.134947.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. 3:993–1022, 2003.

J.-p. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. 2004.

J. A. Gagnon-Bartsch and T. P. Speed. Using control genes to correct for unwanted variation in microarray data. *Biostatistics*, 13(3):539–552, 2012. ISSN 14654644. doi: 10.1093/biostatistics/kxr034.

Y. Gat-viks and I. Steuerman. ComICS: Computational Methods for Immune Cell-Type Subsets. (1):1–7, 2016.

R. Gaujoux and C. Seoighe. Semi-supervised Nonnegative Matrix Factorization for gene expression deconvolution: A case study. *Infection, Genetics and Evolution*, 12(5):913–921, 2012. ISSN 15671348. doi: 10.1016/j.meegid.2011.08.014. URL http://dx.doi.org/10.1016/j.meegid.2011.08.014.

R. Gaujoux, C. Seoighe, and G. E. Deconvolution. BIOINFORMATICS Gene expression CellMix : a comprehensive toolbox for gene expression deconvolution. pages 1–2, 2013. doi: 10.1093/bioinformatics/btt351.

T. Gong, N. Hartmann, I. S. Kohane, V. Brinkmann, F. Staedtler, M. Letzkus, S. Bongiovanni, and J. D. Szustakowski. Optimal deconvolution of transcriptional profiling data using quadratic programming with application to complex clinical blood samples. *PLoS ONE*, 6(11), 2011. ISSN 19326203. doi: 10.1371/journal.pone.0027156.

E. Hubbell, W.-m. Liu, and R. Mei. Robust estimators for expression analysis. 18(12):1585–1592, 2002.

R. A. Irizarry, B. Hobbs, Y. D. Beazer-barclay, K. J. Antonellis, U. W. E. Scherf, and T. P. Speed. Exploration , normalization , and summaries of high density oligonucleotide array probe level data. pages 249–264, 2003.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. 401(October 1999):788–791, 2000.

D. A. Liebner, K. Huang, and J. D. Parvin. MMAD: Microarray microdissection with analysis of differences is a computational tool for deconvoluting cell type-specific contributions from tissue samples. *Bioinformatics*, 30(5):682–689, 2014. ISSN 13674803. doi: 10.1093/bioinformatics/btt566.

P. Lu, A. Nakorchevskiy, and E. M. Marcotte. Expression deconvolution: a reinterpretation of DNA microarray data reveals dynamic changes in cell populations. *Proceedings of the National Academy of Sciences of the United States of America*, 100(18):10370–5, 2003. ISSN 0027-8424. doi: 10.1073/pnas.1832361100. URL http://www.pnas.org/content/100/18/10370{%}5C.

A. M. Newman, Chih Long Liu, Michael R. Green, W. F. Andrew J. Gentles, C. D. H. Yue Xu, M. Diehn, and A. A. Alizadeh. Robust enumeration of cell subsets from tissue expression profiles. *Nat Methods.*, 12(5):193–201, 2015. ISSN 1527-5418. doi: 10.1016/j.molmed.2014.11.008.Mitochondria.

A. Pascual-Montano, J. M. Carazo, S. Member, K. Kochi, D. Lehmann, and R. D. Pascual-marqui. Nonsmooth Nonnegative Matrix Factorization (ns NMF). 28(3):403–415, 2006.

W. Qiao, G. Quon, E. Csaszar, M. Yu, Q. Morris, and P. W. Zandstra. PERT: A Method for Expression Deconvolution of Human Blood Samples from Varied Microenvironmental and Developmental Conditions. *PLoS Computational Biology*, 8(12), 2012. ISSN 1553734X. doi: 10.1371/journal.pcbi.1002838.

G. Quon and Q. Morris. ISOLATE: A computational strategy for identifying the primary origin of cancers using high-throughput sequencing. *Bioinformatics*, 25(21):2882–2889, 2009. ISSN 13674803. doi: 10.1093/bioinformatics/btp378.

G. Quon, S. Haider, A. G. Deshwar, A. Cui, P. C. Boutros, and Q. Morris. Computational purification of individual tumor gene expression profiles leads to significant improvements in prognostic prediction. *Genome medicine*, 5(3):29, 2013. ISSN 1756-994X. doi: 10.1186/gm433.

D. Repsilber, S. Kern, A. Telaar, G. Walzl, G. F. Black, J. Selbig, S. K. Parida, S. H. E. Kaufmann, and M. Jacobsen. Biomarker discovery in heterogeneous tissue samples -taking the in-silico deconfounding approach. *BMC bioinformatics*, 11:27, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-27.

M. E. Ritchie, B. Phipson, D. Wu, Y. Hu, C. W. Law, W. Shi, and G. K. Smyth. limma powers differential expression analyses for RNA-sequencing and microarray studies. 43(7), 2015. doi: 10.1093/nar/gkv007.

S. S. Shen-Orr, R. Tibshirani, P. Khatri, D. L. Bodian, F. Staedtler, N. M. Perry, T. Hastie, M. M. Sarwal, M. M. Davis, and A. J. Butte. Cell type-specific gene expression differences in complex tissues. *Nat Methods*, 7(4):287–289, 2010. ISSN 1548-7105. doi: 10.1038/nmeth.1439.

The R Core Team. R : A Language and Environment for Statistical Computing. 2016. URL https://www.r-project.org/.

D. Venet, F. Pecasse, C. Maenhaut, and H. Bersini. Separation of samples into their constituents using gene expression data. *Bioinformatics*, 17(Suppl 1):S279–S287, 2001. ISSN 1367-4803.

M. Wang, S. R. Master, and L. a. Chodosh. Computational expression deconvolution in a complex mammalian organ. *BMC bioinformatics*, 7:328, 2006. ISSN 1471-2105. doi: 10.1186/1471-2105-7-328. URL http://www.ncbi.nlm.nih.gov/pubmed/16817968{%}5C.

Y. Zhong, Y.-W. Wan, K. Pang, L. M. L. Chow, and Z. Liu. Digital sorting of complex tissues for cell type-specific gene expression profiles. *BMC bioinformatics*, 14:89, 2013. ISSN 1471-2105. doi: 10.1186/1471-2105-14-89.

APPENDICES

APPENDIX A

Code Appendix

Listing A.1: Chapter 4 Knitr Code (Method.R)

```
## ----chap4-fig1,eval=TRUE,cache=TRUE,echo=FALSE,warning=FALSE,fig.height=5,fig.width
    =5----
source('./latin_R/chap4-fig1.R')

## ----chap4-fig2,eval=TRUE,cache=TRUE,echo=FALSE,warning=FALSE,fig.height=5,fig.width
    =5----
source('./latin_R/chap4-fig2.R')

## ----chap4-fig3,eval=TRUE,cache=TRUE,echo=FALSE,warning=FALSE,fig.height=5,fig.width
    =5----
source('./latin_R/chap4-fig3.R')
```

Listing A.2: Chapter 4 Figure 1 (Method/latin_R/chap4-fig1.R)

```
source('./latin_R/latin_data_read.R',chdir=TRUE,local=TRUE)
source('./latin_R/latin_data_plot.R',chdir=TRUE,local=TRUE)
library('ggplot2')

group = 1
gene = 1
probe = 11
```

```

fit = fitd(group,gene,probe,pips)
data = data.frame(fit$dat)

p = ggplot(data,aes(x=vlog2t,y=vlog2d))+geom_point()
p = p + labs(title=expression("log2(Intensity)␣v.␣log2(Concentration)"))
p = p + labs(x="log2(Concentration)",y="Log2(Intensity)")
p = p + xlim(-15,15) + ylim(5,15)
plot(p)
```

Listing A.3: Chapter 4 Figure 2 (Method/latin_R/chap4-fig2.R)

```
source('./latin_R/latin_data_read.R',chdir=TRUE,local=TRUE)
source('./latin_R/latin_data_plot.R',chdir=TRUE,local=TRUE)
library('ggplot2')

group = 1
gene = 1
probe = 11

fit = fitd(group,gene,probe,pips)
data = data.frame(fit$dat)

p = ggplot(data,aes(x=vlog2t,y=vlog2d))+geom_point()
p = p + labs(title=expression("log2(Intensity)␣v.␣log2(Concentration)"))
p = p + labs(x="log2(Concentration)",y="Log2(Intensity)")
p = p + xlim(-15,15) + ylim(5,15)
sig=function(x){sapply(x,function(b){sigmoid(b,fit$theta)})}
p = p + stat_function(fun=sig,color='red')
plot(p)
```

Listing A.4: Chapter 4 Figure 3 (Method/latin_R/chap4-fig3.R)

```
source('./latin_R/latin_data_read.R',chdir=TRUE,local=TRUE)
source('./latin_R/latin_data_plot.R',chdir=TRUE,local=TRUE)
library('ggplot2')
```

```

group = 1
gene = 1
probe = 11

fit = fitd(group,gene,probe,pips)
data = data.frame(fit$dat)

p = ggplot(data,aes(x=vlog2t,y=vlog2d))+geom_point()
p = p + labs(title=expression("log2(Intensity)␣v.␣log2(Concentration)"))
p = p + labs(x="log2(Concentration)",y="Log2(Intensity)")
p = p + xlim(-15,15) + ylim(5,15)
sig=function(x){sapply(x,function(b){sigmoid(b,fit$theta)})}
p + stat_function(fun=sig,color='red')
min = quantile(data$vlog2d,.05)
max = quantile(data$vlog2d,.95)
slope=diff(quantile(data$vlog2d,c(.05,.95)))/8

pw = function(x){
    return(min+slope*x)
}

vpw = function(x){sapply(x,pw)}
s = seq(-15,10,.01)
s = s + runif(length(s))

df = data.frame(as.matrix(t(rbind(s,vpw(s)))))

p = ggplot(data,aes(x=vlog2t,y=vlog2d))+geom_point()
p = p + labs(title=expression("log2(Intensity)␣v.␣log2(Concentration)"))
p = p + labs(x="log2(Concentration)",y="Log2(Intensity)")
p = p + xlim(-15,15) + ylim(5,15)
p = p + stat_function(fun=sig,color='red')
plot(p+geom_line(data=df,aes(x=s,y=V2),color='blue'))
```

Listing A.5: Chapter 4 Latin Square Preprocessing (Method/latin_R/latin_data_read.R)

```
library('affy')

data_top_dir = './latin_data/'
affy_batch = ReadAffy(celfile.path=data_top_dir)

## probe-level data
pips = indexProbes(affy_batch,"pm")
mips = indexProbes(affy_batch,"mm")
d = t(intensity(affy_batch))

# order according to experiment number
ord = order(sapply(rownames(d),function(x){as.integer(substr(strsplit(x,"_")[[1]][8],5,10)
    )}))
d = d[ord,]

# determine gene spike-in titrations
anno = read.csv('anno.csv',stringsAsFactors=FALSE)
genes = sapply(anno[1,2:15],function(x){strsplit(x,"\n",fixed=TRUE)})
titration = as.matrix(anno[2:15,2:15])
titration = apply(titration,c(1,2),as.numeric)

# rows are spiked-in genes, columns are experiment
all_titration = array(0,c(dim(titration)[1]*3,dim(titration)[2]))
for(i in 1:dim(all_titration)[1]){
    all_titration[i,] = titration[ceiling(i/3),]
}
all_titration_order = apply(all_titration,2,order)

# gene-level data
dg = data.frame(rma(affy_batch,verbose=FALSE,
                    normalize=TRUE,background=FALSE))
dg = dg[ord,]
```

Listing A.6: Chapter 4 Latin Square Fitting Routine (Method/latin_R/latin_data_plot.R)

```
sigmoid = function(x,t){return(t[1]+t[2]/(1+exp(-(t[3]+t[4]*x))))}
sigmoid_inv = function(x,t){return((-1/t[4])*(log((t[2]/(x-t[1]))-1)+t[3]))}

F = function(x,y){
    return(x-y)
}

fitd=function(group,gene,probe,ips){

    probe_groups = ips[unlist(lapply(genes[group],'[',gene))]
    probes = unlist(lapply(probe_groups,'[',probe))
    od = d[,probes]

    vlog2d = as.vector(t(log2(od)))
    if(!is.null(dim(od))){
        vlog2t = rep(log2(all_titration[,group]+.0001),dim(od)[2])
    } else {
        vlog2t = log2(all_titration[,group]+.0001)
    }
    vlog2t = vlog2t + runif(length(vlog2t),0,1/1000)

    fail = TRUE
    i=0
    max_try = 10
    t = c(6.3657341,  6.6679978, -3.0497695,  0.6106055)

    while(fail){
        i = i + 1
        if(i > max_try){
            fail=2
            break;
        }

        fail = tryCatch({
            if(i!=1){
                t = t + rnorm(length(t),0,1)
            }
            init = list(t1=t[1],t2=t[2],t3=t[3],t4=t[4])
            mod = nls(vlog2d~sigmoid(vlog2t,c(t1,t2,t3,t4)),
                      start=init,
                      control=list('warnOnly'=FALSE,'maxiter'=1000,'minFactor'=1E-12))
            fail = FALSE
        }, error = function(e){
            return(TRUE)
        })
    }

    if(fail<2){
        t1=coef(mod)[1];t2=coef(mod)[2];t3=coef(mod)[3];t4=coef(mod)[4]
        sigmoid = function(x){return(t1+t2/(1+exp(-(t3+t4*x))))}
        return(list(dat=cbind(vlog2t,vlog2d),fail=FALSE,theta=c(t1,t2,t3,t4)))
    }

    return(list(dat=cbind(vlog2t,vlog2d),fail=TRUE,theta=rep(0,4)))
}
```

Listing A.7: Chapter 5 Knitr Code (Analysis.R)

```
## ----plot1,cache=FALSE,warning=FALSE,echo=FALSE,error=FALSE,message=FALSE,fig.height=5,
    fig.width=7----
source('./R/Analysis_script.R',chdir=TRUE)
sel = sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
aplot(19830,'diag',sel)

## ----plot2,cache=FALSE,warning=FALSE,echo=FALSE,error=FALSE,message=FALSE,fig.height=5,
    fig.width=7----
source('./R/Analysis_script.R',chdir=TRUE)
sel = sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
aplot(19830,'err_boxplot',sel)
```

```

## ----table1,cache=FALSE,warning=FALSE,echo=FALSE,error=FALSE,message=FALSE,results='asis'----
library('xtable')
source('./R/Analysis_script.R',chdir=TRUE)
sel = sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
t = aplot(19830,'summ',sel)
print(xtable(t),include.rownames=FALSE)

## ----plot3,cache=FALSE,warning=FALSE,echo=FALSE,error=FALSE,message=FALSE,fig.height=5,fig.width=7----
source('./R/Analysis_script.R',chdir=TRUE)
sel = sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
aplot(11058,'diag',sel)

## ----plot4,cache=FALSE,warning=FALSE,echo=FALSE,error=FALSE,message=FALSE,fig.height=5,fig.width=7----
source('./R/Analysis_script.R',chdir=TRUE)
sel = sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
aplot(11058,'err_boxplot',sel)

## ----table2,cache=FALSE,warning=FALSE,echo=FALSE,error=FALSE,message=FALSE,results='asis'----
library('xtable')
source('./R/Analysis_script.R',chdir=TRUE)
sel = sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
t = aplot(11058,'summ',sel)
print(xtable(t),include.rownames=FALSE)
```

Listing A.8: Chapter 5 Analysis script (Analysis/R/Analysis_script.R)

```
source('update.R')
source('read_data.R')
library('deconv')
library('reshape2')
library('ggplot2')
library('xtable')

this_dir = getwd()

aplot = function(num,plot_type,sel=rep(TRUE,5)){
    mgs = function(affy_batch,marker_probes){
        ips = indexProbes(affy_batch,which='both')

        compars = array(0,c(length(ips),length(marker_probes)))
        rownames(compars) = names(ips)

        melted_ips = melt(ips)
        colnames(melted_ips)=c("probe","gene")
        melted_ips$probe = sapply(melted_ips$probe,toString)

        marker_gtables = lapply(marker_probes,function(x){table(melted_ips$gene[which(
           melted_ips$probe %in%  x)])})
        for(i in 1:length(marker_probes)){
            compars[names(marker_gtables[[i]]),i] = marker_gtables[[i]]
        }
        maxes = apply(compars[rowMeans(compars)>0,],1,which.max)
        marker_genes = lapply(1:length(marker_probes),function(i){names(maxes)[maxes==i]})
        names(marker_genes) = names(marker_probes)
        return(marker_genes)
    }

    n_choose = 1000

    saved_dir = paste(this_dir,"/data/",sep="")
    saved_dname = paste(num,"_",n_choose,sep="")
    saved_f = paste(saved_dir,saved_dname,sep="")

    if(!file.exists(saved_f)){

        updt("Generating␣data␣set.",init=TRUE)
        d = get_dset(num)
        tmp = lapply(d$pure_samples,function(x){sample(x,length(x)/2)})
        names(tmp)=names(d$pure_samples)
        d$pure_samples=tmp
        updt()

        updt("Generating␣marker␣probes.\n")
        mps = deconv(d$affy_batch,d$pure_samples,marker_info_only=TRUE)$marker_info
        marker_probes = lapply(mps,function(x){rownames(x[1:n_choose,])})
        marker_genes = mgs(d$affy_batch,marker_probes)
        updt()

        updt("Running␣my␣linear␣deconvolution␣algorithm.",init=TRUE)
        my_ests_lin= deconv(d$affy_batch,d$pure_samples,marker_probes=marker_probes,model=
            'linear')$estimates
        updt()

        logv = FALSE

        updt("Running␣DSA.")
        res_dsa=ged(object=d$cellmix,x=MarkerList(marker_genes),method='dsa',log=logv)
        dsa_ests = t(coef(res_dsa))
        updt()

        updt("Running␣ssKL.")
        res_sskl=ged(object=d$cellmix,x=MarkerList(marker_genes),method='sskl',log=logv)
        sskl_ests = t(coef(res_sskl))
        updt()

        updt("Running␣ssF.")
        res_ssf=ged(object=d$cellmix,x=MarkerList(marker_genes),method='ssfrobenius',log=
            logv)
        ssf_ests = t(coef(res_ssf))
        updt()

        updt("Running␣lsfit.")
        res_lsfit = ged(object=d$cellmix,basis(d$cellmix)[unlist(marker_genes),],method='
            lsfit')
        lsfit_ests = t(coef(res_lsfit))
        updt()

        updt("Running␣qprog.\n")
        res_qprog = ged(object=d$cellmix,x=basis(d$cellmix)[unlist(marker_genes),],method=
            'qprog')
        qprog_ests = t(coef(res_qprog))
        updt()

        res = list("my_ests_lin"=my_ests_lin,
                   "dsa"=dsa_ests,
                   "lsfit"=lsfit_ests,
                   "qprog"=qprog_ests,
                   "sskl"=sskl_ests,
                   "ssf"=ssf_ests,
                   "true"=d$conv,
                   "pure_samples"=d$pure_samples)

        saveRDS(res,saved_f)
    } else {
        res = readRDS(saved_f)
    }

    #sel = c(input$ours_linear,input$dsa,input$lsfit,input$qprog,input$sskl,input$ssf)
    sel=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)

    df = data.frame(type=melt(res$true)[,2],
                    true=melt(res$true)$value,
                    my_ests_lin=melt(res$my_ests_lin)$value,
                    dsa=melt(res$dsa)$value,
                    lsfit=melt(res$lsfit)$value,
                    qprog=melt(res$qprog)$value,
                    sskl=melt(res$sskl)$value,
                    ssf=melt(res$ssf)$value
                    )

    data_sel = 1:dim(df)[1]

    K = length(res$pure_samples)
    ps = unlist(res$pure_samples)
    ps_map = melt(data.frame(t(sapply(1:dim(res$true)[1],function(x){rep(x,K)}))))[,2]
    ps_rows = which(ps_map%in%ps)
    data_sel = data_sel[-ps_rows]

    df = df[data_sel,]

    results=list()
    results$df = df
    results$sel = sel

    ## diag plot
    if(plot_type=='diag'){
        algo_types = names(df)[-(1:2)]
        algo_names = c("Our␣Algorithm","DSA","LSFIT","QPROG","ssKL","ssFrobenius")
        algo_colors = 1:length(algo_names)

        algo_types = algo_types[sel]
        algo_names = algo_names[sel]
        algo_colors = algo_colors[sel]

        rep_true = rep(df$true,length(algo_types))

        plot_df=melt(df[,algo_types])
        plot_df$true = rep_true
        if(length(algo_types)==1){
            plot_df$variable=rep(algo_types[1],dim(plot_df)[1])
        }

        ggplot = qplot(data=df,x=true,y=my_ests_lin,col=I('black'),size=I(2.5),shape=I(1),
           alpha=I(0)) +
            ggtitle("Estimated␣proportions␣by␣true␣proportions") +
            labs(x="True␣Proportions",y="Estimated␣Proportions")
        ggplot = ggplot + geom_abline(slope=1,intercept=0,color='orange',size=I(1))
        ggplot = ggplot + theme(axis.text=element_text(size=10),
                                axis.title=element_text(size=10,face="bold"),
                                plot.title=element_text(size=10,face="bold"),
                                legend.title=element_text(size=10),
                                legend.position="bottom")
        ggplot = ggplot +
            geom_point(aes(x=true,y=value,col=variable),data=plot_df,size=I(2.5),shape=I
               (1))
        ggplot = ggplot + labs(color="Algorithm")+
            scale_color_manual(labels = algo_names,values=algo_colors)
        ggplot = ggplot + xlim(0,1)
        ggplot = ggplot + ylim(0,1)
        return(ggplot)
    }

    # error plots
    if(plot_type=='err_boxplot'){
        abst = FALSE
        transform = function(x){
            if(abst){
                return(abs(x))
            } else {
                return(x)
            }
        }

        yl=ylim(-.75,.75)
        if(abst) yl=ylim(0,.75)

        df = results$df

        errdf = data.frame("Our␣Algorithm" = transform(df$my_ests_lin- df$true),check.
            names=FALSE)
        errdf["DSA"] = transform(df$dsa - df$true)
        errdf["Least␣Squares"] = transform(df$lsfit - df$true)
        errdf["Quad.␣Prog."] = transform(df$qprog - df$true)
        errdf["ssKL"] = transform(df$sskl - df$true)
        errdf["ssFrobenius"] = transform(df$ssf - df$true)

        p = ggplot(stack(errdf), aes(x = ind, y = values)) +
            geom_boxplot() + yl +
            ggtitle("Error␣of␣Estimated␣Proportions") +
            labs(x="Algorithm",y="Error␣of␣Estimated␣Proportions") +
            theme(axis.text=element_text(size=10),
                  axis.title=element_text(size=10,face="bold"),
                  plot.title=element_text(size=10,face="bold"),
                  legend.title=element_text(size=10),
                  legend.position="bottom")

        return(p)
    }

    # summary
    if(plot_type=='summ'){

        abst = TRUE
        transform = function(x){
            if(abst){
                return(abs(x))
            } else {
                return(x)
            }
        }


        df = results$df

        errdf = data.frame("Our␣Algorithm" = transform(df$my_ests_lin- df$true),check.
            names=FALSE)
        errdf["DSA"] = transform(df$dsa - df$true)
        errdf["Least␣Squares"] = transform(df$lsfit - df$true)
        errdf["Quad.␣Prog."] = transform(df$qprog - df$true)
        errdf["ssKL"] = transform(df$sskl - df$true)
        errdf["ssFrobenius"] = transform(df$ssf - df$true)

        qs = c(0,.25,.5,.75,1)
        tbl = round(apply(errdf,2,function(x){quantile(x,)}),3)
        colnames(tbl)[1] = "Our␣Algorithm"
        tbl = cbind(paste(round(qs*100),"%",sep=""),tbl)
        colnames(tbl)[1] = "Percentile"
        rownames(tbl) = NULL

        return(tbl)
    }
}
```

Listing A.9: Chapter 5 Data Reading Script (Analysis/R/read_data.R)

```
suppressPackageStartupMessages({
    require('affy')
    require('CellMix')
})
```

```

geo_dl = list('GSE5350'='ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE5nnn/GSE5350/suppl/
    GSE5350_MAQC_AFX_123456_120CELs.zip',
              'GSE19830'='ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE19nnn/GSE19830/suppl/
                  GSE19830_RAW.tar',
              'GSE33076'='ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE33nnn/GSE33076/suppl/
                  GSE33076_RAW.tar',
              'GSE29832'='ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE29nnn/GSE29832/suppl/
                  GSE29832_RAW.tar',
              'GSE11058'='ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE11nnn/GSE11058/suppl/
                  GSE11058_RAW.tar'
              )

unpack = function(str){
    s = strsplit(str,"\\.")
    end = tolower(rev(s[[1]])[1])
    if(end=='tar'){
        return('tar␣xvf')
    } else if(end=='gz'){
        return('tar␣xvzf')
    } else if(end=='zip'){
        return('unzip')
    }
}

this_dir = getwd()

get_dset = function(geo_acc_num){
    geo_acc = paste("GSE",geo_acc_num,sep="")

    curr_dir = getwd()
    geo_data_dir = '/home/greg/Dropbox/work/research/2016/geo_data'
    this_data_dir = paste(geo_data_dir,"/",geo_acc,sep="")

    if(!dir.exists(this_data_dir)){
        dir.create(this_data_dir)
        setwd(this_data_dir)
        dl = geo_dl[[geo_acc]]
        system(paste('wget', dl))
        system(paste(unpack(dl)," * "))
        setwd(curr_dir)
    }

    affy_batch = ReadAffy(celfile.path=this_data_dir)
    cmData = ExpressionMix(geo_acc)

    conv = t(coef(cmData))
    colnames(conv) = tolower(colnames(conv))

    get_pure = function(cm_coef){
        ones = apply(cm_coef,1,function(x){which(x==1)[1]})
        pure = ones[!is.na(ones)]
        return(lapply(1:length(unique(pure)),function(x){which(pure==x)}))
    }

    pure_samples = get_pure(conv)
    names(pure_samples) = colnames(conv)

    return(list("affy_batch"=affy_batch,"pure_samples"=pure_samples,"conv"=conv,"cellmix"=
        cmData))
}
```

Listing A.10: Chapter 5 Helper Script (Analysis/R/update.R)

```
library(crayon)

sze = function(unit='Mb',envir=globalenv()){
    os = function(x){object.size(get(x))}
    os_pretty = function(x){format(object.size(get(x)),units=unit)}
```

```r
    ord = order(sapply(ls(envir),os),decreasing=TRUE)
    print(sapply(ls(envir),os_pretty)[ord])
    cat("Total:␣",sum(sapply(ls(envir),os)) * 1E-9,"Gb␣\n")
}


VERBOSE = TRUE
TIME = Sys.time()
updt = function(msg=NULL,init=FALSE,done=FALSE){
    if(is.null(msg)) done=TRUE
    if(VERBOSE){
        if(!done){
            last_msg <<- msg
        } else {
            msg <- paste(last_msg,green("[DONE]"))
        }
        t2 = Sys.time()
        if(init){
            TIME<<-t2
            FIRST<<-t2
        }
        MSG = paste(msg,"␣(",format(t2-FIRST,digits=1),"␣ellapsed)",sep="")
        TIME <<- t2
        cat(paste(blue(paste("[",t2,"]␣",sep="")),MSG,sep=""),"\n")
        return(MSG)
    }
    return("")
}
```