THE UNIVERSITY OF CHICAGO

EMPIRICAL BAYES MATRIX FACTORIZATION: METHODS AND APPLICATIONS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
PH.D.

DEPARTMENT OF STATISTICS

BY
JASON WILLWERSCHEID

CHICAGO, ILLINOIS
SEPTEMBER 2021

For Ursula Méabh

Let us toss our umbrella, said Mercier. It will fall in a certain way, according to laws of which we know nothing. Then all we have to do is press forward in the designated direction.

— Sam Beckett, *Mercier and Camier* (1946)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I owe an incalculable debt of gratitude to my thesis advisor, Matthew Stephens. Nearly every idea in the pages that follow has its originary spark in the dynamism of his thinking, at once boundlessly creative and startlingly clear. Many thanks also to committee members Dan Nicolae and John Novembre. I am in awe of John, a model for ethical and rigorous scholarship in a field as potentially fraught as human genetics. And Dan, the heart and soul of UChicago statistics: I am immensely grateful for your leadership and example. I would also like to single out Svetlana Jitomirskaya, who got me into this mess a decade ago by rekindling a profound admiration for the precision of mathematical thought. Спасибо за помощь.

Thanks also to Mihai Anitescu, Chao Gao, and Peter McCullagh for their mentorship, exactitude, and encouragement; Yibi Huang and Mei Wang for their kind generosity with teaching materials and consulting advice; and Keisha Prowoznik, Laura Rigazzi, and Kirsten Wellman for their patience in helping me to navigate administrative waters. Finally, my work has been deeply enriched by numerous conversations with current and former members of the UChicago Statistics Department and the Stephens Lab: in particular, I'd like to thank Matt Bonakdarpour, Ahmed Bou-Rabee, Peter Carbonetto, You-Lin Chen, Kushal Dey, Andrew Goldstein, Youngseok Kim, Joe Marcus, Chris McKennan, Fabio Morgante, Jean Morrison, Sen Na, Vivak Patel, Abhishek Sarkar, Lei Sun, Micol Tresoldi, Gao Wang, Zihao Wang, Rob Webber, Dongyue Xie, and Yuxin Zou.

But above all: thanks to my wife Maria for her pragmatism and strength, and for the time on Lake Como when we crashed our tandem and lost our shanks; and to my daughter for joy inutterable.

ABSTRACT

Matrix factorization methods are commonly used to explore structure in multivariate data. When the structure can be expected to have a sparse representation, then a sparsity-inducing method will often be preferred. Empirical Bayes matrix factorization (EBMF), a recent approach that uses the observed data to estimate priors, can adaptively model sparsity and thus yield representations with interpretable components while also performing well on inferential tasks. Further, since fitting the EBMF model can be reduced to solving a series of empirical Bayes normal means (EBNM) subproblems, which can be relatively easily solved for a wide variety of prior families — sparse and otherwise —, the approach is very general.

The dissertation extends the reach of EBMF in several ways. The first chapter describes the R package **ebnm**, which I developed in order to provide a unified interface for efficiently solving the EBNM problem using a range of prior families. Existing packages are harnessed when practical; in other cases, solutions are implemented from scratch. The second chapter details my implementation of the EBMF algorithm, **flashier**, which was designed to handle much larger datasets and offer more flexibility than the original implementation of EBMF. In particular, I show that EBMF can yield insight into single-cell RNA sequencing data, outperforming more commonly used methods on tasks such as rare cell type detection in spite of the fact that the EBMF model is misspecified for count data. The final chapter considers data with an underlying tree-like structure, with particular attention to population genetics data. In addition to providing theory that helps to elucidate the kind of factorization that one should be looking for, I propose a tailored EBMF method that can successfully identify tree-like structure in both simulated and real datasets.

INTRODUCTION

I started working on empirical Bayes matrix factorization (EBMF) in early 2018, shortly after Wei Wang and Matthew Stephens uploaded the first version of their EBMF paper to arXiv (Wang and Stephens [2021]). They had named their software implementation **flashr**, an acronym for "factors and loadings by adaptive shrinkage in R." In effect, the EBMF model is:

$$\mathbf{Y} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{1}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}_\ell^{(k)} \tag{2}$$

$$f_{jk} \sim g_f^{(k)} \in \mathcal{G}_f^{(k)}, \tag{3}$$

where $\mathbf{L}$ is a $n \times K$ matrix of "loadings," $\mathbf{F}$ is a $p \times K$ matrix of "factors," and $\mathbf{E}$ is a $n \times p$ matrix of normally distributed errors. The column-wise priors $g_\ell^{(k)}$ and $g_f^{(k)}$ are estimated via empirical Bayes from among prior families $\mathcal{G}_\ell^{(k)}$ and $\mathcal{G}_f^{(k)}$, and Wang and Stephens were particularly interested in the families of "adaptive shrinkage" or "ash" priors introduced by Stephens in a previous paper (Stephens [2017]). In brief, ash priors are families of scale mixture distributions (for example, scale mixtures of normals) with mixture proportions to be estimated. By using ash priors, EBMF can learn about the scales of the loadings and factors from the data, and can "adaptively" shrink an estimate of ℓ_{ik} or f_{jk} depending on which component it most likely "belongs" to. In particular, EBMF can adaptively model sparsity for columns of $\mathbf{L}$ and $\mathbf{F}$, which can help to produce highly interpretable results.

As I later discovered, the software also contains a somewhat concealed reference to photography. The original implementation included a function `flash_fill` that fills in missing data in $\mathbf{Y}$ with posterior means estimated using EBMF. I wanted to remove the function from the interface, since the same can be accomplished via a single line of R code:

```
is.na(Y) <- f$ldf[is.na(Y)],
```

where `f` is the fitted `flash` object. Matthew Stephens resisted, however, on the grounds that he was fond of the "photography joke." Fill flash is a technique whereby a photographer uses artificial light to reduce unwanted shadows (missing data, in a sense). It can be useful when, for example, a background is brightly illuminated by sunlight but the subject is shaded. The photographic metaphor is also apt for EBMF more generally. Just as a flash bulb illuminates otherwise obscure aspects of things as they appear, **flashr** casts light on the structure of matrix data by revealing interpretable factors. The "flash" of `flash` yields an image that, like a good photograph, makes sense of a messy reality.

When I first started working with **flashr**, however, it felt — with apologies to Wei and Matthew — more like a disposable camera than a professional-grade apparatus. Some aspects of EBMF seemed less than perfectly understood; parts of **flashr** were clunky; and, in general, it lacked the flexibility and range of settings that I required for my applications. In short, **flashr** wasn't flashy enough. A portion of my project thus consisted in software engineering: building a bigger and better camera with more polished and various lenses. The other portion was to show what kinds of pictures could be taken: my implementation, **flashier**, opens EBMF up to not only new subject matter (much larger datasets) but also new techniques, such as semi-nonnegative matrix factorization and co-occurrence matrix factorization. The thesis that follows is thus part technical manual and part photo album.

In Chapter 1, I revisit the empirical Bayes normal means (EBNM) problem, which is central to EBMF in the same way that shutter speed or aperture is central to photographic technique. The EBNM problem can be written:

$$x_i \sim \mathcal{N}\left(\theta_i, s_i^2\right) \tag{4}$$

$$\theta_i \sim g \in \mathcal{G}, \tag{5}$$

where $\mathbf{x}$ is a vector of observations with known standard errors $\mathbf{s}$, $\boldsymbol{\theta}$ is a vector of unknown means, and g is to be estimated via empirical Bayes from among some family of priors $\mathcal{G}$.

As Wang and Stephens noticed, the EBMF model can be fit by solving a series of EBNM problems. For example, in the rank-one case, where

$$\mathbf{Y} = \boldsymbol{\ell}\mathbf{f}^T + \mathbf{E} \tag{6}$$

$$\ell_i \sim g_\ell \in \mathcal{G}_\ell \tag{7}$$

$$f_j \sim g_f \in \mathcal{G}_f \tag{8}$$

$$e_{ij} \sim \mathcal{N}\left(0, \sigma_{ij}^2\right), \tag{9}$$

fixing $\mathbf{f}$ yields

$$\frac{y_{ij}}{f_j} \sim \mathcal{N}\left(\ell_i, \frac{\sigma_{ij}^2}{f_j^2}\right) \tag{10}$$

$$\ell_i \sim g_\ell \in \mathcal{G}_\ell, \tag{11}$$

which is an EBNM problem but with p observations for each "mean" ℓ_i. In essence, the rank-one `flash` algorithm proceeds by fixing $\mathbf{f}$ and solving an EBNM problem to estimate $\boldsymbol{\ell}$, then fixing $\boldsymbol{\ell}$ and solving an EBNM problem to estimate $\mathbf{f}$, and so on until convergence.

The choice of prior families $\mathcal{G}_\ell$ and $\mathcal{G}_f$ can have dramatic effects. Prior families can encode assumptions about, for example, sparsity, tail weight, and nonnegativity, and can range in flexibility from the one-parameter family of zero-mean normal distributions to the nonparametric family of all distributions. Like a fast shutter speed, the family of all distributions can capture very fine-grained information about a prior distribution, while the family of zero-mean normals blurs details. And just as the appropriate choice of shutter speed depends on the application (fast for sporting events; slow for night photography), so will the choice of prior family. To take full advantage of the range of possibilities offered by different prior families, I developed the R package **ebnm**, which provides a unified interface for previously available packages that solve the EBNM problem as well as providing fast

and stable implementations for useful prior families that (to my knowledge) were not readily available.

With these fundamentals in place, Chapter 2 discusses my EBMF implementation, **flashier**. The application that I had most particularly in mind when I developed the package was single-cell RNA sequencing (scRNA-seq) data, a task to which **flashr** proved unequal: due to inefficiencies in the implementation, larger scRNA-seq datasets easily exhausted 64 GB of memory, and smaller datasets took an unnecessarily long time to fit. Fitting these datasets required core changes to the implementation, including the modularization of matrix operations and the use of acceleration techniques to reduce computational time. In addition to describing these changes, I introduce semi-nonnegative EBMF, which uses a family of priors with nonnegative support for the loadings $\mathbf{L}$ and a family of priors with support on the reals for the factors $\mathbf{F}$ (or vice versa). I conclude the chapter with a pair of photographs: I fit two scRNA-seq datasets using both semi-nonnegative and vanilla EBMF in order to illustrate some of the advantages of EBMF over competing methods like topic modeling and GLM-PCA, as well as showing how semi-nonnegative EBMF can enhance interpretability over vanilla EBMF.

Chapter 3 takes fullest advantage of the newer, flashier apparatus. The primary object of the chapter is to understand data for observations coming from a set of populations whose relations can be described by a rooted, binary tree. Since not all matrix factorizations are equally informative about population structure, and since different factorizations are more or less easy to "find" via optimization, some theory is required in order to define a factorization that both reveals population structure and has a decent chance of being discovered via a sparse matrix factorization method such as EBMF. I call my factorization of choice the "divergence factorization," and I propose a tailored method for using EBMF to find it. Additionally, I develop theory for factorizing the co-occurrence matrix $\mathbf{Y}\mathbf{Y}^T$ rather than the full data matrix $\mathbf{Y}$: although the model is less well specified, the co-occurrence

matrix is often much smaller. Further, there are technical reasons that make it possible for a factorization of the co-occurrence matrix to succeed where the full factorization fails. Both of these methods ("tree-EBMF" and "tree-EBcovMF") would be difficult to execute without the newer implementation, as they require a considered choice of prior family, flexible initialization, the ability to fix and unfix specific loadings, and extreme flexibility in the order of operations. I conclude by using the methods to illuminate a pair of population genetics datasets: a set of North American wolf genomes, as well as data from the 1000 Genomes Project, which includes genome data for individuals from 26 human populations.

CHAPTER 1

THE EBNM PROBLEM

1.1 Introduction

Given n observations x_i with known standard errors s_i, the normal means model assumes that

$$x_i \overset{\text{ind.}}{\sim} \mathcal{N}(\theta_i, s_i^2) \tag{1.1}$$

with "true means" θ_i to be estimated. The maximum likelihood estimate of θ_i is simply x_i. A key Bayesian insight, however, is that the observations supply information not only about the individual means, but also about how the means, taken together, are distributed. Specifically, the empirical Bayes (EB) approach makes the additional modeling assumption

$$\theta_i \overset{\text{ind.}}{\sim} g \in \mathcal{G}, \tag{1.2}$$

where the prior g is to be estimated from among some family of distributions $\mathcal{G}$ that is specified in advance.

I refer to the problem of estimating $g \in \mathcal{G}$ in order to perform inference on the means θ_i as the "empirical Bayes normal means" (EBNM) problem. In Section 1.2, I give a brief history of the EBNM problem, describe several applications in which it crucially arises, and cite a number of software packages that can be used to solve it. These packages typically fix the choice of $\mathcal{G}$ or restrict it to suit a particular application: as a result, some simple but useful choices of $\mathcal{G}$ lack an implementation. To bridge these gaps, I developed the R package **ebnm**, which provides a unified interface for efficiently solving the EBNM problem under a wide variety of distributional assumptions. I leverage existing packages where possible; in other cases, solutions are implemented from scratch. Section 1.3 provides implementation details and develops theory that can help guide the user in choosing

settings for previously implemented prior families. In Section 1.4, I use simulated data to compare different prior families $\mathcal{G}$. Often, the choice of prior family must weigh trade-offs between speed and flexibility, and a "correct" choice rarely exists for a given application. Finally, Section 1.5 illustrates the use of **ebnm** on two real-data examples, Rubin's eight schools data (Rubin [1981]) and a dataset of eQTLs derived from GTEx project data (Lonsdale et al. [2013]). Package **ebnm** is available at `https://github.com/stephenslab/ebnm`. Figures and results from this chapter can be reproduced by following the instructions at `https://github.com/willwerscheid/ebnm-paper`, with some variation to be expected in the timing results.

1.2 Background and Applications

Stein [1956] famously discovered that under quadratic loss, the maximum likelihood estimate $\hat{\theta}_i = x_i$ is an inadmissible solution to the normal means problem (Equation 1.1) when $n \geq 3$. James and Stein [1961] subsequently gave an explicit formula for a shrinkage estimator that dominates the MLE. As Efron and Morris [1973] showed, a lightly modified version of the James-Stein estimator (the so-called "positive-part James-Stein estimator") can be derived via an empirical Bayes approach that takes the prior family to be the conjugate family of zero-mean normal distributions

$$\mathcal{G}_{\text{norm}} := \left\{ g : g \sim \mathcal{N}(0, \sigma^2) \text{ for some } \sigma^2 \geq 0 \right\}. \tag{1.3}$$

Estimating $\hat{g} \in \mathcal{G}_{\text{norm}}$ via maximum likelihood is straightforward: indeed, when all standard errors s_i are identical, the problem has a closed-form solution.

In applications, however, one is often interested in scenarios where the means vector $\boldsymbol{\theta}$ is known to be sparse. In such cases, prior families that are able to model sparsity directly are preferable. In wavelet denoising, for example, one assumes that a signal has a sparse

representation within a suitable wavelet basis. This assumption can be modeled by putting a spike-and-slab prior on the wavelet coefficients $\boldsymbol{\theta}$ — that is, a mixture distribution consisting of two components, one a point mass at zero (the "spike") and the other belonging to some family of continuous distributions $\mathcal{F}$ that are symmetric and unimodal at zero (the "slab"). Within a fully Bayesian framework, an early popular choice was the point-normal family

$$\mathcal{G}_{\text{pn}} := \left\{ g : g \sim \pi_0 \delta_0 + (1 - \pi_0) \mathcal{N}(0, \sigma^2) \text{ for some } 0 \leq \pi_0 \leq 1, \sigma^2 \geq 0 \right\}. \tag{1.4}$$

More recently, Johnstone and Silverman [2005] showed that empirical Bayes methods enjoy better theoretical guarantees when $\mathcal{F}$ is taken to be a family of distributions whose tails are exponential or heavier. Their companion R package **EbayesThresh** implements both the point-Laplace family

$$\mathcal{G}_{\text{pl}} := \{ g : g \sim \pi_0 \delta_0 + (1 - \pi_0) \text{Laplace}(a) \text{ for some } 0 \leq \pi_0 \leq 1, a \geq 0 \} \tag{1.5}$$

and a family of priors in which the slab component has Cauchy-like tails.

A second application in which assumptions about sparsity have proven useful is false discovery rate (FDR) control. In genetics, for example, one often measures effect sizes for thousands of genes (say, differences in gene expression between cancerous and non-cancerous cells) and would like to disentangle null and non-null effects. A natural approach is to put a spike-and-slab prior on the effects and then, via a latent variable representation, estimate for each effect the posterior probability that it is generated from the non-null (slab) component. Specifically, if $\mathcal{G}$ is a spike-and-slab prior family, then it is equivalent to write the prior

$\theta_i \sim g \in \mathcal{G}$ as:

$$y_i \sim \text{Bernoulli}\,(1 - \pi_0) \tag{1.6}$$

$$\theta_i \;\big|\; y_i = 0 \sim \delta_0 \tag{1.7}$$

$$\theta_i \;\big|\; y_i = 1 \sim f \in \mathcal{F}. \tag{1.8}$$

In this representation, the "local false discovery rate" (Efron [2008]) is the posterior probability that effect i is null:

$$\text{fdr}(i) := \mathbb{P}\left(y_i = 0 \;\big|\; x_i, s_i, \hat{\pi}_0, \hat{f}\right). \tag{1.9}$$

In genetics, practitioners have generally been reluctant to make parametric assumptions about $\mathcal{F}$. Efron has developed various empirical Bayes methods for estimating false discovery rates without making any assumptions about $\mathcal{G}$ and without modeling g directly (see, in particular, Efron [2010]). More recently, Stephens [2017] has argued that simply requiring $\mathcal{G}$ to be a family of priors that is unimodal at zero can improve FDR control while retaining the flexibility of nonparametric approaches. Two families that work well in practice are the family of scale mixtures of normals

$$\mathcal{G}_{\text{smn}} := \left\{g : g \sim \int_0^\infty \mathcal{N}\left(0, \sigma^2\right)\; dh\left(\sigma^2\right) \text{ for some distribution } h\right\} \tag{1.10}$$

and the family of all symmetric distributions g that are unimodal at zero. For analogy with $\mathcal{G}_{\text{smn}}$, the latter can be represented as a family of scale mixtures of uniforms:

$$\mathcal{G}_{\text{symm}} := \left\{g : g \sim \int_0^\infty \text{Unif}\,[-a, a]\; dh\,(a) \text{ for some distribution } h\right\}. \tag{1.11}$$

These families and others are implemented in Stephens's R package **ashr**. To solve the

EBNM problem for nonparametric prior families, **ashr** borrows ideas from Koenker and Gu's R package **REBayes** (Koenker and Gu [2017]), which is primarily concerned with the problem of finding the Kiefer-Wolfowitz nonparametric maximum likelihood estimate — that is, the EBNM problem with $\mathcal{G}$ the family of all distributions (Kiefer and Wolfowitz [1956]).

All of the prior families defined so far are families of shrinkage priors, which guarantee that posterior means will never be larger than raw observations:

$$|\mathbb{E}\left(\theta_i \mid x_i, s_i, \hat{g}\right)| \leq |x_i|. \tag{1.12}$$

To motivate prior families that have a very different flavor, I consider semi-nonnegative matrix factorization (Ding et al. [2010]) as a third application. Wang and Stephens [2021] showed that an EB approach to matrix factorization can be reduced to a series of EBNM problems. As a schematic illustration, consider the rank-one model:

$$\mathbf{X} = \boldsymbol{\ell}\mathbf{f}' + \mathbf{E} \tag{1.13}$$

$$e_{ij} \sim \mathcal{N}\left(0, \sigma^2\right) \tag{1.14}$$

$$\ell_i \sim g_\ell \in \mathcal{G}_\ell \tag{1.15}$$

$$f_j \sim g_f \in \mathcal{G}_f. \tag{1.16}$$

Here, $\mathbf{X}$ is an $n \times p$ matrix of observations, $\boldsymbol{\ell}$ is an n-vector of loadings, $\mathbf{f}$ is a p-vector of factor scores, and $\mathbf{E}$ is an $n \times p$ matrix of residual noise. Now, if $\boldsymbol{\ell}$ were fixed, then one could write

$$\frac{x_{ij}}{\ell_i} \overset{\text{ind}}{\sim} \mathcal{N}\left(f_j, \frac{\sigma^2}{\ell_i^2}\right) \tag{1.17}$$

$$f \sim g_f \in \mathcal{G}_f, \tag{1.18}$$

which is an EBNM problem but with n observations for each "true mean" f_j. This suggests an algorithm for empirical Bayes matrix factorization (EBMF) in which one fixes $\boldsymbol{\ell}$ and solves an EBNM problem to estimate $\mathbf{f}$, then fixes $\mathbf{f}$ to estimate $\boldsymbol{\ell}$, and iterates until convergence. For a more rigorous treatment, see Wang and Stephens (the details are somewhat different from what has been described here).

A particular appeal of the EBMF approach is that different prior families can be specified for loadings and factor scores, as well as for each of the factors in a rank-K factorization. Semi-nonnegative matrix factorizations can be obtained by allowing factor scores to take arbitrary values while constraining loadings to be nonnegative by way of a prior family with nonnegative support. For example, one can set $\mathcal{G}_f = \mathcal{G}_{\text{norm}}$ while taking $\mathcal{G}_\ell$ to be the family of nonnegative distributions that are unimodal at zero, which, similarly to $\mathcal{G}_{\text{symm}}$, can be represented as a family of mixtures of uniforms:

$$\mathcal{G}_{\text{nn}} := \left\{ g : g \sim \int_0^\infty \text{Unif}\left[0, a\right] \; dh\left(a\right) \text{ for some distribution } h \right\}. \tag{1.19}$$

As I argue in Chapter 2, a semi-nonnegative approach to matrix factorization can offer substantial gains in interpretability without the computational difficulties that a fully nonnegative approach entails (see, in particular, Section 2.3.3). In Section 1.5, I will tease this larger argument by obtaining a semi-nonnegative matrix factorization of the GTEx dataset discussed by Wang and Stephens.

1.3 Software: Methods and Theory

Several software packages have been developed to solve the EBNM problem for specific choices of prior family. In the previous section, I cited **EbayesThresh** (point-Laplace and point-Cauchy prior families), **ashr** (nonparametric unimodal prior families), and **REBayes** (the family of all distributions). Other R packages include **horseshoe**, which estimates a

Function	Form of Prior	Package	Sign	Symm.
`normal`	$\mathcal{N}\left(\mu, \sigma^2\right)$	**ebnm**		✓
`point_normal`	$\pi_0 \delta_\mu + (1 - \pi_0)\, \mathcal{N}\left(\mu, \sigma^2\right)$	**ebnm**		✓
`point_laplace`	$\pi_0 \delta_\mu + (1 - \pi_0)\, \text{Laplace}\left(\mu, a\right)$	**ebnm**		✓
`point_exponential`	$\pi_0 \delta_0 + (1 - \pi_0)\, \text{Exp}\left(a\right)$	**ebnm**	+	
`horseshoe`	$\text{Horseshoe}\left(\tau\right)$	**horseshoe**		✓
`normal_scale_mixture`	$\int_0^\infty \mathcal{N}\left(0, \sigma^2\right)\ dh\left(\sigma^2\right)$	**ebnm**		✓
`unimodal_symmetric`	$\int_0^\infty \text{Unif}[-a, a]\ dh(a)$	**ashr**		✓
`unimodal`	$\int_{-\infty}^\infty \text{Unif}[0, a]\ dh(a)$	**ashr**		
`unimodal_nonnegative`	$\int_0^\infty \text{Unif}[0, a]\ dh(a)$	**ashr**	+	
`unimodal_nonpositive`	$\int_0^\infty \text{Unif}[-a, 0]\ dh(a)$	**ashr**	-	
`npmle`	$\int_{-\infty}^\infty \delta_x\ dh(x)$	**ebnm**		
`deconvolver`	Narasimhan and Efron [2020]	**deconvolveR**		

Table 1.1: Prior families implemented in **ebnm**. "Sign" indicates the family's support (nonnegative, nonpositive, or real-valued). "Symm." indicates whether the family is symmetric about its mode.

horseshoe prior (Carvalho et al. [2010]), and **deconvolveR**, which models a smooth prior using a natural spline basis (Narasimhan and Efron [2020]).

I designed package **ebnm** with two intentions. First, I wanted to provide an interface with common inputs and outputs for previously implemented prior families. Second, implementations for some simple but useful prior families (in particular, zero-mean normal and point-normal families) are, to my knowledge, not available via a commonly used repository such as CRAN or Bioconductor. I implemented such families from scratch. In other cases, I re-implemented existing families when efficiency could be improved or when desired outputs weren't available.

Table 1.1 lists all prior families implemented in **ebnm**. I've divided the table into three main groups. The first group, which is discussed in Section 1.3.1, consists of families defined by a small number of parameters. Some are spike-and-slab families (point-normal, point-Laplace, point-exponential); others are families of single-component distributions (normal,

horseshoe). Section 1.3.2 details the approach used by **ashr** (Stephens [2017]) to estimate g for the second group, which consists of nonparametric families that are unimodal at zero. The approach is similar to the one used by **REBayes** (Koenker and Gu [2017]) to find the NPMLE: both discretize the problem by choosing a "grid" of scale or location parameters and then estimate mixture proportions for each component via maximum likelihood. In Section 1.3.3, I provide new theoretical results that can more rigorously guide the selection of the grid. Finally, Section 1.3.4 describes the third group, which consists of the NPMLE and the **deconvolveR** prior, which are not in general unimodal. Results for NPMLE grid selection are provided as well.

1.3.1 Parametric Families

While, in general, any method that "peeks" at the data in order to estimate g can be qualified as empirical Bayes, I use maximum likelihood to estimate $g \in \mathcal{G}$ throughout. Thus inference for the EBNM problem is a two-step procedure in which g is first estimated:

$$\hat{g} = \underset{g \in \mathcal{G}}{\operatorname{argmax}} \prod_i \int p\left(x_i \mid \theta_i, s_i\right) \, dg\left(\theta_i\right), \tag{1.20}$$

and then quantities of interest are calculated using posterior distributions $p\left(\theta_i \mid x_i, s_i, \hat{g}\right)$.

When all standard errors s_i are identical (say, $s_i = s$ for all i), then the family of zero-mean normal distributions $\mathcal{G}_{\text{norm}}$ offers a simple closed-form solution to the EBNM problem:

$$\hat{g} \sim \mathcal{N}\left(0, \max\left\{0, \frac{1}{n}\sum x_i^2 - s^2\right\}\right) \tag{1.21}$$

I note in passing that this solution is different from the one implied by the positive-part James-Stein estimator, which divides $\sum x_i^2$ by $n-2$ rather than by n (Efron and Morris [1973]).

For heteroskedastic observations and for all other parametric families (point-normal,

point-Laplace, point-exponential, and horseshoe), a closed-form solution is not available. Since at most three parameters need to be estimated (the mode, the scale of the slab component, and the spike/slab mixture proportions), off-the-shelf optimizers are sufficient. By default, **ebnm** uses `nlm`, a Newton-type method included in package **stats**. Other options include the L-BFGS-B algorithm as implemented by **stats** function `optim` and the trust-region method implemented in package **trust**. In each of my experiments, `nlm` was either the fastest method or differed from the fastest by less than a factor of two. For details, see the benchmarking results in Section 1.7.

I implemented likelihood, gradient, and Hessian calculations for all parametric families except the horseshoe. Function `ebnm_horseshoe` calls into package **horseshoe** (van der Pas et al. [2019]).

1.3.2 Nonparametric Unimodal Families

To estimate $g \in \mathcal{G}$ when the prior family $\mathcal{G}$ is nonparametric, **ebnm** uses the strategy outlined in Stephens [2017] and implemented in package **ashr** (which in turn borrows ideas from Koenker and Gu [2017]). Namely, with $\mathcal{G}$ represented as a family of infinite mixtures of parametric distributions, **ashr** first chooses a suitably dense subfamily of finite mixtures $\tilde{\mathcal{G}} \subseteq \mathcal{G}$. A careful selection of this (parametric) subfamily $\tilde{\mathcal{G}}$ ensures that the solution $\tilde{g} \in \tilde{\mathcal{G}}$ will be a good approximation to the exact MLE $\hat{g} \in \mathcal{G}$.

For example, the family of scale mixtures of normals is defined as the family of infinite mixtures

$$\mathcal{G}_{\text{smn}} := \left\{ g : g \sim \int_0^\infty \mathcal{N}\left(0, \sigma^2\right) \, dh\left(\sigma^2\right) \text{ for some distribution } h \right\}. \tag{1.22}$$

By choosing a sufficiently dense grid of scale parameters $\{\sigma_1^2, \ldots, \sigma_K^2\}$, it's possible to obtain

a solution to the EBNM problem for the subfamily

$$\tilde{\mathcal{G}} = \left\{ g : g \sim \pi_1 \mathcal{N}\left(0, \sigma_1^2\right) + \ldots + \pi_K \mathcal{N}\left(0, \sigma_K^2\right) \;\middle|\; \pi_1, \ldots, \pi_K \geq 0,\ \sum_{k=1}^{K} \pi_k = 1 \right\} \quad (1.23)$$

that is arbitrarily close to the exact MLE $\hat{g} \in \mathcal{G}_{\text{smn}}$.

Similarly, one can write the family of symmetric distributions that are unimodal at zero as

$$\mathcal{G}_{\text{symm}} := \left\{ g : g \sim \int_0^\infty \text{Unif}\left[-a, a\right]\ dh(a) \text{ for some distribution } h \right\}. \quad (1.24)$$

By choosing a dense grid $\{a_1, \ldots, a_K\}$, one can obtain an arbitrarily good approximation to $\hat{g} \in \mathcal{G}_{\text{symm}}$ by optimizing over

$$\tilde{\mathcal{G}} = \left\{ g : g \sim \pi_1 \text{Unif}\left[-a_1, a_1\right] + \ldots + \pi_K \text{Unif}\left[-a_K, a_K\right] \;\middle|\; \pi_1, \ldots, \pi_K \geq 0,\ \sum_{k=1}^{K} \pi_k = 1 \right\}. \quad (1.25)$$

For most nonparametric prior families, **ebnm** calls into **ashr**, which in turn uses **mixsqp** (Kim et al. [2020]) as the default method for estimating the mixture proportions $\pi_1, \ldots, \pi_K$. An algorithm based on sequential quadratic programming that was specifically developed with **ashr**-like problems in mind, **mixsqp** tends to be faster than interior-point solvers such as **REBayes** when the number of mixture components is not too large. However, the R implementation of **mixsqp**, which is slower than the Julia implementation, can be outpaced by **REBayes** when there are more than 100 or so mixture components. See the benchmarking results in Section 1.7.

Because **ashr** is intended to be quite general — for example, it can solve empirical Bayes means problems for likelihoods other than normal — there are some inefficiencies in the manner in which it solves the EBNM problem. In particular, I was able to streamline the algorithm for scale mixtures of normals such that **ebnm** is roughly twice as fast as **ashr** when $\mathcal{G} = \mathcal{G}_{\text{smn}}$ (again, see Section 1.7).

1.3.3 Quality of Grid-Based Approximations to Nonparametric Families

Care must be taken in choosing the grid of scale parameters. It should be dense enough to give a good approximation to the exact solution $\hat{g} \in \mathcal{G}$ but not so dense that computation crawls to a halt. To provide guidelines for setting the grid, it is useful to bound the distance from the parametric subfamily $\tilde{\mathcal{G}}$ to the MLE $\hat{g} \in \mathcal{G}$.

For the sake of simplicity, consider the case where all standard errors s_i are identical to s. The quality of the grid-based approximation can be measured as the minimum Kullback-Leibler divergence, over all $\tilde{g} \in \tilde{\mathcal{G}}$, from the convolution of $\tilde{g}$ with the normal error distribution $h \sim \mathcal{N}\left(0, s^2\right)$ to the convolution of the exact solution $\hat{g}$ with the error distribution h:

$$\mathrm{KL}\left(\hat{g} * h \mid \tilde{\mathcal{G}} * h\right) := \min_{\tilde{g} \in \tilde{\mathcal{G}}} \int \log \frac{(\hat{g} * h)(x)}{(\tilde{g} * h)(x)} \, d(\hat{g} * h)(x), \tag{1.26}$$

where

$$(\hat{g} * h)(x) := \int \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{y^2}{2s^2}\right) \hat{g}(x - y) \, dy \tag{1.27}$$

and where $(\tilde{g} * h)(x)$ is similarly defined.

This divergence has a very natural interpretation. Recall that the objective function that is being maximized in order to estimate $g \in \mathcal{G}$ is the marginal log likelihood of the data, $\sum_i \log p\left(x_i \mid s_i, g\right)$. The KL divergence defined by Equation 1.26 gives the expected per-observation reduction in log likelihood that one incurs when one moves from the nonparametric family $\mathcal{G}$ to the parametric subfamily $\tilde{\mathcal{G}}$, assuming that the data is truly generated from the exact solution $\hat{g} \in \mathcal{G}$.

For the prior family $\mathcal{G}_{\mathrm{smn}}$, the following result allows the grid to be described by a single parameter, the "grid multiplier" m, which maps to a tight bound on the KL divergence from $\tilde{\mathcal{G}} * h$ to $\hat{g} * h$:

Theorem 1.1 (Scale mixtures of normals)**.** *Let $\mathcal{G} = \mathcal{G}_{smn}$ and let all standard errors s_i be identical to s. Choose $m > 1$ and set the grid of scale parameters to be $\{0, (m-1)s^2, (m^2 -$*

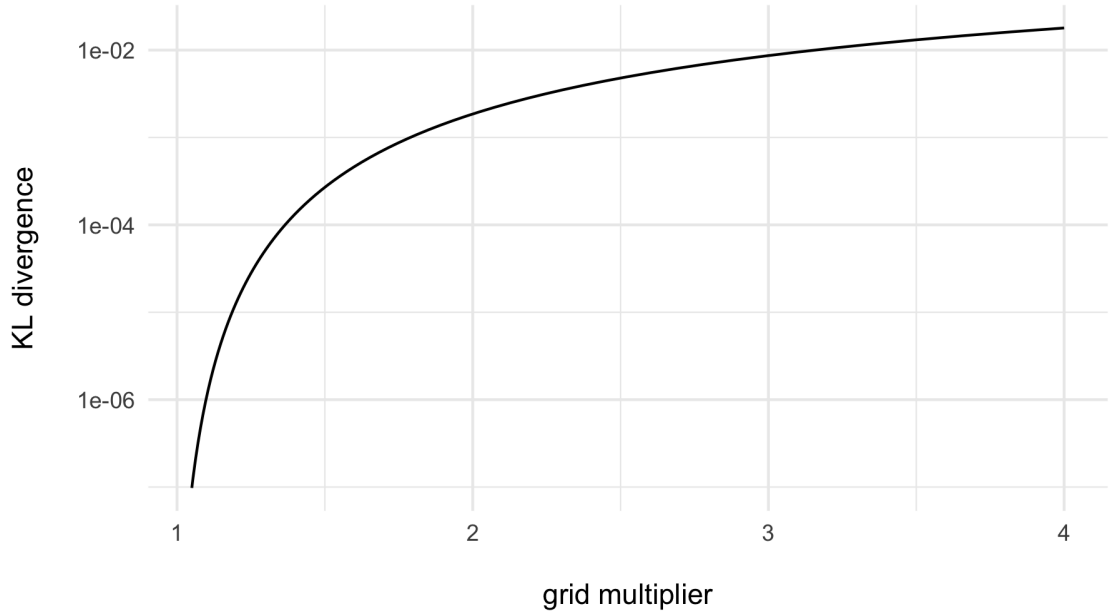


Figure 1.1: Quality of grid-based approximations for scale mixtures of normals as a function of the grid multiplier m. Plotted is an upper bound on the KL-divergence from the approximate solution convolved with the error distribution to the exact solution convolved with the error distribution.

$1)s^2, \ldots, (m^{K-1}-1)s^2\}$, *with* K *taken to be sufficiently large so that* $K - 1 \geq \log_m \frac{max_i x_i^2}{s^2}$. *Then*

$$KL\left(\hat{g} * h \mid \tilde{\mathcal{G}} * h\right) \leq \max_{1 \leq \sigma^2 \leq m} \min_{0 \leq \omega \leq 1} KL\left(\mathcal{N}\left(0, \sigma^2\right) \;\middle|\; \omega\mathcal{N}\left(0,1\right) + \left(1-\omega\right)\mathcal{N}\left(0,m\right)\right). \tag{1.28}$$

The quantity on the right-hand side can be evaluated using Monte Carlo sampling and is plotted over a range of m in Figure 1.1. The proof of the theorem is deferred until the end of the section.

For example, the default **ashr** grid is $\{0, c, mc, m^2c, \ldots, m^{K-1}c\}$, with $c = \frac{s^2}{100}$ and $m = 2$. Since $c \leq (m-1)s^2$ and $m \leq \frac{m^{k+1}-1}{m^k-1}$ for all k, this grid is finer than the grid given by the theorem, with the grids equally dense as $k \to \infty$. Theorem 1.1 and Figure 1.1 thus give an upper bound on KL $\left(\hat{g} * h \mid \tilde{\mathcal{G}} * h\right)$ of about 0.002, which implies that the objective attained by the approximate solution is guaranteed to be within one log likelihood unit of

the objective attained by the exact solution for up to 500 observations or so (again assuming that the exact solution matches the true data distribution).

In general, with n observations, I find that choosing m such that

$$\mathrm{KL}\left(\hat{g} * h \mid \tilde{\mathcal{G}} * h\right) \leq 1/n \tag{1.29}$$

is a good rule of thumb, since $1/n$ is the approximate order of magnitude of the KL-divergence from $\hat{g} * h$ to the *true* prior g convolved with the error distribution h when $g \in \mathcal{G}_{\text{smn}}$. (Indeed, it would be of doubtful utility to require an approximation $\tilde{g} \approx \hat{g}$ that is orders of magnitude more accurate than the estimate $\hat{g} \approx g$.) While calculating $\mathbb{E}\left(\mathrm{KL}(g * h \mid \hat{g} * h)\right)$ is not feasible for $\mathcal{G}_{\text{smn}}$, the following result holds for the family of zero-mean normal distributions $\mathcal{G}_{\text{norm}}$ (a proof is given at the end of the section):

Theorem 1.2. *Let $\boldsymbol{x}$ be a vector of homoskedastic observations generated from some true prior belonging to the family of zero-mean normal distributions $\mathcal{G}_{norm}$. Estimate $\hat{g}$ as*

$$\hat{g} = \underset{g \in \mathcal{G}_{norm}}{\operatorname{argmax}} \prod_i \int p\left(x_i \mid \theta_i, s_i\right) \, dg\left(\theta_i\right). \tag{1.30}$$

Then

$$\mathbb{E}\left(KL\left(g * h \mid \hat{g} * h\right)\right) = O\left(\frac{1}{n}\right). \tag{1.31}$$

Since $\mathbb{E}\left(\mathrm{KL}(g * h \mid \hat{g} * h)\right)$ ought to be at least as small for larger prior families such as $\mathcal{G}_{\text{smn}}$, Theorem 1.2 justifies the rule of thumb given by Equation 1.29.

To test the theory, I simulate, for each of $n \in \{100, 1000, 10000\}$, 20 datasets with, in each case, a different scale mixture of normals as the true data-generating distribution. I use function `ebnm_normal_scale_mixture` to solve the EBNM problem, first using the default **ebnm** grid (which uses Theorem 1.1 with the target KL-divergence $\mathbb{E}\left(\mathrm{KL}(g * h \mid \hat{g} * h)\right)$ equal to $1/n$) and then using several finer grids in order to estimate the approximate optimal

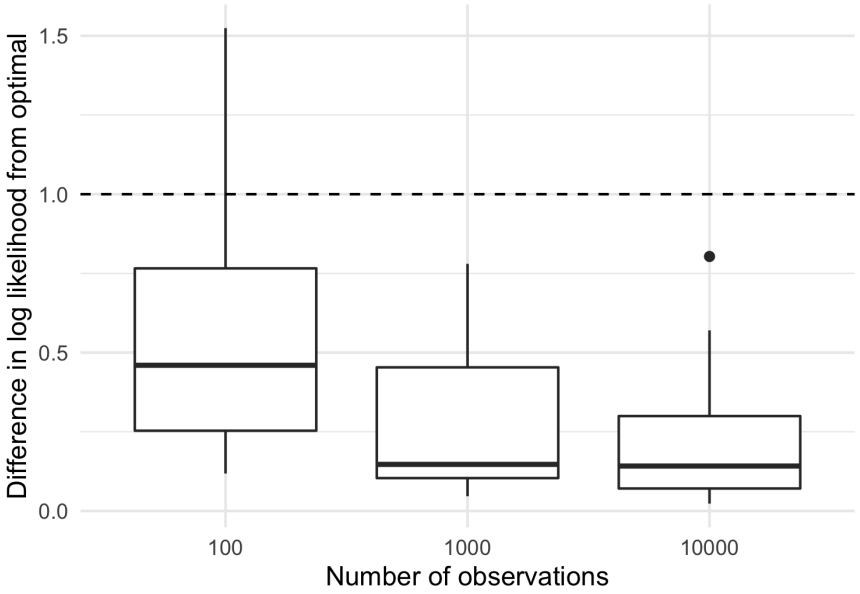


Figure 1.2: Difference in log likelihood between the `ebnm_normal_scale_mixture` solution obtained using the default grid and using an estimated optimal grid. In each case, the data-generating distribution consists of one to five mixture components, with the scales of the components sampled from an exponential distribution with rate 1/5. $\mathcal{N}(0, 1)$ noise is added to the observations.

log likelihood. The theory asserts that, on average, the log likelihood obtained using the default grid should be within one log likelihood unit of optimal. Figure 1.2 suggests that this is indeed the case.

Proofs

Proof of Theorem 1.1. Denote the exact MLE $\hat{g} \in \mathcal{G}_{smn}$ as

$$\hat{g} \sim \pi_1 \mathcal{N}\left(0, \tau_1^2\right) + \ldots + \pi_L \mathcal{N}\left(0, \tau_L^2\right), \tag{1.32}$$

so that

$$\hat{g} * h \sim \pi_1 \mathcal{N}\left(0, \tau_1^2 + s^2\right) + \ldots + \pi_L \mathcal{N}\left(0, \tau_L^2 + s^2\right). \tag{1.33}$$

Assign the τ_ℓ^2s to their respective grid intervals $\sigma_{a(\ell)}^2 \leq \tau_\ell^2 < \sigma_{b(\ell)}^2$ (that is, set $\sigma_{a(\ell)}^2 = \max\left\{\sigma_k^2 : \sigma_k^2 \leq \tau_\ell^2\right\}$ and $\sigma_{b(\ell)}^2 = \min\left\{\sigma_k^2 : \sigma_k^2 > \tau_\ell^2\right\}$; note, in particular, that $b(\ell) = a(\ell)+1$). Choose $0 \leq \omega \leq 1$ and consider

$$\tilde{g}_\omega \sim \pi_1 \left(\omega \mathcal{N}\left(0, \sigma_{a(1)}^2\right) + (1-\omega)\mathcal{N}\left(0, \sigma_{b(1)}^2\right)\right) + \ldots + \pi_L \left(\omega \mathcal{N}\left(0, \sigma_{a(L)}^2\right) + (1-\omega)\mathcal{N}\left(0, \sigma_{b(L)}^2\right)\right). \quad (1.34)$$

Now, by the chain rule for relative entropy (see, for example, Hershey and Olsen [2007]):

$$\begin{aligned}
&\mathrm{KL}\left(\hat{g} * h \;\middle|\; \tilde{g}_\omega * h\right) && (1.35)\\
&\quad \leq \sum_\ell \pi_\ell \mathrm{KL}\left(\mathcal{N}\left(0, \tau_\ell^2 + s^2\right) \;\middle|\; \omega \mathcal{N}\left(0, \sigma_{a(\ell)}^2 + s^2\right) + (1-\omega)\mathcal{N}\left(0, \sigma_{b(\ell)}^2 + s^2\right)\right) && (1.36)\\
&\quad = \sum_\ell \pi_\ell \mathrm{KL}\left(\mathcal{N}\left(0, \frac{\tau_\ell^2 + s^2}{\sigma_{a(\ell)}^2 + s^2}\right) \;\middle|\; \omega \mathcal{N}(0,1) + (1-\omega)\mathcal{N}(0,m)\right) && (1.37)\\
&\quad \leq \max_{1 \leq a \leq m} \mathrm{KL}\left(\mathcal{N}(0,a) \;\middle|\; \omega \mathcal{N}(0,1) + (1-\omega)\mathcal{N}(0,m)\right). && (1.38)
\end{aligned}$$

Thus, since $\tilde{g}_\omega \in \tilde{\mathcal{G}}$ and ω was chosen arbitrarily,

$$\begin{aligned}
\mathrm{KL}\left(\hat{g} * h \;\middle|\; \tilde{\mathcal{G}} * h\right) &\leq \min_{0 \leq \omega \leq 1} \mathrm{KL}\left(\hat{g} * h \;\middle|\; \tilde{g}_\omega * h\right) && (1.39)\\
&\leq \max_{1 \leq a \leq m} \min_{0 \leq \omega \leq 1} \mathrm{KL}\left(\mathcal{N}(0,a) \;\middle|\; \omega \mathcal{N}(0,1) + (1-\omega)\mathcal{N}(0,m)\right). && (1.40)
\end{aligned}$$

□

Proof of Theorem 1.2. Let $g * h \sim \mathcal{N}(0, \sigma^2)$. Then

$$\hat{g} * h \sim \mathcal{N}(0, \tau^2), \quad (1.41)$$

where

$$\tau^2 = \frac{1}{n}\sum_i x_i^2 \sim \frac{\sigma^2}{n}\chi_n^2. \tag{1.42}$$

(I assume that σ^2 is large enough so that $\mathbb{P}(\tau^2 > 1) \approx 1$). Also,

$$\mathrm{KL}\left(g * h \mid \hat{g} * h\right) = \mathrm{KL}\left(\mathcal{N}\left(0, \sigma^2\right) \mid \mathcal{N}\left(0, \tau^2\right)\right) \tag{1.43}$$

$$= \frac{1}{2}\left(\frac{\sigma^2}{\tau^2} - 1 + \log\left(\frac{\tau^2}{\sigma^2}\right)\right). \tag{1.44}$$

Now,

$$\frac{\sigma^2}{\tau^2} \sim n\Gamma^{-1}\left(\frac{n}{2}, \frac{1}{2}\right), \tag{1.45}$$

so

$$\mathbb{E}\frac{\sigma^2}{\tau^2} = \frac{n}{n-2}, \tag{1.46}$$

and

$$\log\left(\frac{\tau^2}{\sigma^2}\right) \sim -\log n + \log\Gamma\left(\frac{n}{2}, 2\right), \tag{1.47}$$

so

$$\mathbb{E}\log\left(\frac{\tau^2}{\sigma^2}\right) = -\log n + \log 2 + \psi\left(\frac{n}{2}\right), \tag{1.48}$$

where ψ is the digamma function. Thus

$$\mathbb{E}\left(\mathrm{KL}(g * h \mid \hat{g} * h)\right) = \frac{2}{n-2} - \log\left(\frac{n}{2}\right) + \psi\left(\frac{n}{2}\right) \tag{1.49}$$

$$= \frac{2}{n-2} - \frac{1}{n} + O(n^{-2}) \tag{1.50}$$

$$= \frac{1}{n-2} + O(n^{-2}) \tag{1.51}$$

□

1.3.4 Nonparametric Non-Unimodal Families

In the literature, the term "nonparametric maximum likelihood estimator," or NPMLE, denotes the prior $\hat{g}$ that is estimated from among the family of all distributions $\mathcal{G}_{\text{np}}$. Although it has found somewhat limited use in applications, it has received considerable theoretical attention: see, for example, Dicker and Zhao [2016] and Koenker and Gu [2017].

To approximate the NPMLE (which is known to be a finite mixture of point masses), one can choose grids of location parameters $\{\mu_1, \ldots, \mu_K\}$ and scale parameters $\{\sigma_1^2, \ldots, \sigma_K^2\}$ and then optimize over the family of Gaussian mixtures

$$\tilde{\mathcal{G}} = \left\{ g : g \sim \pi_1 \mathcal{N}\left(\mu_1, \sigma_1^2\right) + \ldots + \pi_K \mathcal{N}\left(\mu_K, \sigma_K^2\right) \;\middle|\; \pi_1, \ldots, \pi_K \geq 0,\ \sum_{k=1}^{K} \pi_k = 1 \right\}. \tag{1.52}$$

As above, an analysis of the KL divergence from the grid-based approximation to the exact MLE can guide grid selection.

Theorem 1.3 (NPMLE, Gaussian mixture approximation)**.** *Let $\mathcal{G} = \mathcal{G}_{np}$ and let all standard errors s_i be identical to s. Set the grid of location parameters to be equally spaced over the range of the observations x_i: $\{m, m + d, \ldots, M = m + (K-1)d\}$, where $m = \min_i x_i$, $M = \max_i x_i$, and $d = \frac{M-m}{K-1}$. Fix the scale parameters σ_k^2 at $d^2/4$ for all k. Then*

$$KL(\hat{g} * h \;\big|\; \tilde{\mathcal{G}} * h) \leq \frac{1}{2} \log\left(1 + \frac{d^2}{4s^2}\right) \tag{1.53}$$

More simply, taking $\sigma_k \to 0$ for all k, one can optimize over mixtures of point masses

$$\tilde{\mathcal{G}} = \left\{ g : g \sim \pi_1 \delta_{\mu_1} + \ldots + \pi_K \delta_{\mu_K} \;\middle|\; \pi_1, \ldots, \pi_K \geq 0,\ \sum_{k=1}^{K} \pi_k = 1 \right\} \tag{1.54}$$

Theorem 1.4 (NPMLE, Point mass mixture approximation)**.** *Again set $\mathcal{G} = \mathcal{G}_{np}$, let all standard errors s_i be identical to s, and set the grid of location parameters to be equally*

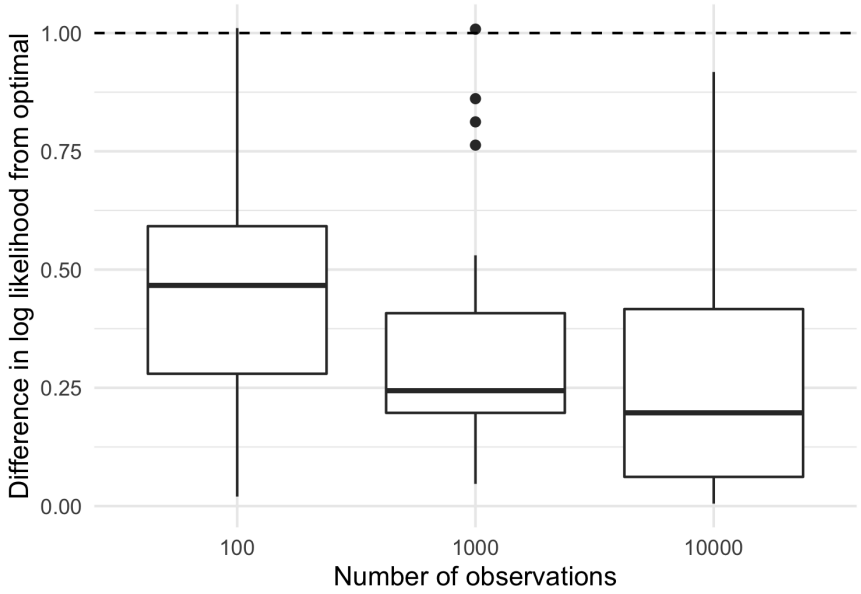


Figure 1.3: Difference in log likelihood between the `ebnm_npmle` solution obtained using the default grid (which uses Theorem 1.4 with the target KL-divergence equal to $1/n$) and using an estimated optimal grid. Each data-generating distribution consists of one to five point masses sampled from an exponential distribution with rate 1/5.

spaced over the range of the observations. Using a family of mixtures of point masses to approximate $\mathcal{G}_{np}$*,*

$$KL(\hat{g} * h \mid \tilde{\mathcal{G}} * h) \leq \frac{d^4}{64s^4} \tag{1.55}$$

Proofs are included at the end of the section. When d is small relative to s (roughly, when $d < 2.3s$), then the latter bound is better. For most problems, package **mixsqp** can comfortably handle hundreds of mixture components, so a mixture of point masses should be preferred unless the range of the observations is unusually large relative to s.

As in Section 1.3.3, I confirm the theory via simulation. Results are shown in Figure 1.3.

In addition to `ebnm_npmle`, I've included a function that interfaces with Narasimhan and Efron's R package **deconvolveR** (Narasimhan and Efron [2020]), which estimates g using a natural spline basis. Like the NPMLE, the **deconvolveR** prior is nonparametric and not necessarily unimodal. In contrast to the "spikiness" of the NPMLE, however, **deconvolveR** yields a smooth $\hat{g}$. Depending on the application, one might be preferred to the other.

Indeed, Koenker [2017] has shown that **deconvolveR** can outperform **REBayes** in terms of the Wasserstein distance between the true and estimated priors when the true prior is in fact smooth.

Proofs

Proof of Theorem 1.3. Note that the exact NPMLE is always a finite mixture of point masses (see, for example, Dicker and Zhao [2016]). Using the chain rule for relative entropy as in the proof of Theorem 1.1 gives

$$\begin{aligned}&\mathrm{KL}(\hat{g} * h \mid \tilde{\mathcal{G}} * h)\\ &\quad\leq \max_{0\leq\mu\leq d}\min_{0\leq\omega\leq 1}\mathrm{KL}\left(\mathcal{N}(\mu,s^2)\mid \omega\mathcal{N}\left(0,\frac{d^2}{4}+s^2\right)+(1-\omega)\mathcal{N}\left(d,\frac{d^2}{4}+s^2\right)\right). \quad (1.56)\end{aligned}$$

Choosing $\omega = 0$ whenever $0 \leq \mu \leq d/2$ and $\omega = 1$ whenever $d/2 < \mu \leq d$ yields

$$\begin{aligned}\mathrm{KL}(\hat{g} * h \mid \tilde{\mathcal{G}} * h) &\leq \max_{0\leq\mu\leq d/2}\mathrm{KL}\left(\mathcal{N}(\mu,s^2)\mid \mathcal{N}\left(0,\frac{d^2}{4}+s^2\right)\right) &(1.57)\\ &= \max_{0\leq\mu\leq d/2}\frac{1}{2}\left(\frac{\mu^2+s^2}{\frac{d^2}{4}+s^2}-1+\log\left(\frac{\frac{d^2}{4}+s^2}{s^2}\right)\right) &(1.58)\\ &= \frac{1}{2}\log\left(1+\frac{d^2}{4s^2}\right). &(1.59)\end{aligned}$$

□

Proof of Theorem 1.4. Again use the chain rule for relative entropy:

$$\mathrm{KL}(\hat{g} * h \mid \tilde{\mathcal{G}} * h) \tag{1.60}$$

$$\leq \max_{-d/2 \leq \mu \leq d/2} \min_{0 \leq \omega \leq 1} \mathrm{KL}\left(\mathcal{N}(\mu, s^2) \mid \omega \mathcal{N}\left(-\frac{d}{2}, s^2\right) + (1-\omega)\mathcal{N}\left(\frac{d}{2}, s^2\right)\right) \tag{1.61}$$

$$= \max_{-d/2 \leq \mu \leq d/2} \min_{0 \leq \omega \leq 1} \mathbb{E}_{z \sim \mathcal{N}(\mu, s^2)} \log \left(\frac{\exp(-\frac{(z-\mu)^2}{2s^2})}{\omega \exp(-\frac{(z+d/2)^2}{2s^2}) + (1-\omega)\exp(-\frac{(z-d/2)^2}{2s^2})} \right) \tag{1.62}$$

The RHS is maximized by setting $\mu = 0$ and then minimized by setting $\omega = 1/2$. Simplifying and then using a Taylor expansion gives:

$$\mathrm{KL}(\hat{g} * h \mid \tilde{\mathcal{G}} * h) \leq \frac{d^2}{8s^2} - \mathbb{E}_{z \sim \mathcal{N}(0, s^2)} \log\left(\cosh\left(\frac{dz}{2s^2}\right)\right) \tag{1.63}$$

$$\leq \frac{d^2}{8s^2} - \frac{d^2}{8s^4}\mathbb{E}z^2 + \frac{d^4}{192s^8}\mathbb{E}z^4 \tag{1.64}$$

$$= \frac{d^4}{64s^4} \tag{1.65}$$

□

1.4 Prior Families Comparisons

It's clear that certain prior families offer more flexibility than others. A direct comparison is possible in the case where families are nested. For example,

$$\mathcal{G}_{\text{norm}} \subseteq \mathcal{G}_{\text{pn}} \subseteq \mathcal{G}_{\text{smn}} \subseteq \mathcal{G}_{\text{symm}} \subseteq \mathcal{G}_{\text{np}}, \tag{1.66}$$

where the prior families $\mathcal{G}$ are, respectively, the family of zero-mean normals (Equation 1.3), point-normal priors (Equation 2.5), scale mixtures of normals (Equation 1.10), symmetric unimodal priors (Equation 1.11), and the nonparametric family of all distributions. At

one extreme, $\mathcal{G}_{\text{norm}}$ has good shrinkage properties and allows the EBNM problem to be easily solved by optimizing over a single variable. However, the inability to model exact sparsity or heavy tails proves too inflexible for many applications. At the other extreme, the nonparametric family of all distributions is, by definition, more flexible than any other family, but its implementation requires more ingenuity. Further, estimates $\hat{g} \in \mathcal{G}_{\text{np}}$ tend to be "spiky" distributions that often perform well in terms of, say, mean squared error, but sometimes feel unnatural from the point of view of the application.

As I will show in Section 1.4.1, flexibility typically comes at the cost of runtime. When the EBNM problem only needs to be solved once, runtime is not usually a concern, but in applications such as EBMF that must iteratively solve a large number of EBNM problems, speed can be critical. On the other hand, one of the advantages of flexibility is that, as I illustrate in Section 1.4.2, it can improve predictive performance. There are limits, however: in particular, since $\mathcal{G}_{\text{np}}$ does not encode any assumptions about sparsity, the NPMLE does worse than unimodal prior families when the data-generating distribution is sparse.

Throughout, I use three data-generating distributions:

- **Point-normal**. A point-normal prior with 90% sparsity:

$$0.9\delta_0 + 0.1\mathcal{N}\left(0, 2^2\right). \tag{1.67}$$

- **Point-t**. A 80-20 mixture of a point mass at zero and a scaled t_5 distribution:

$$0.8\delta_0 + 0.2 * 1.5t_5. \tag{1.68}$$

- **Asymmetric tophat distribution**. A 50-50 mixture of a point mass at zero and a uniform distribution on $[-5, 10]$

$$0.5\delta_0 + 0.5\text{Unif}[-5, 10]. \tag{1.69}$$

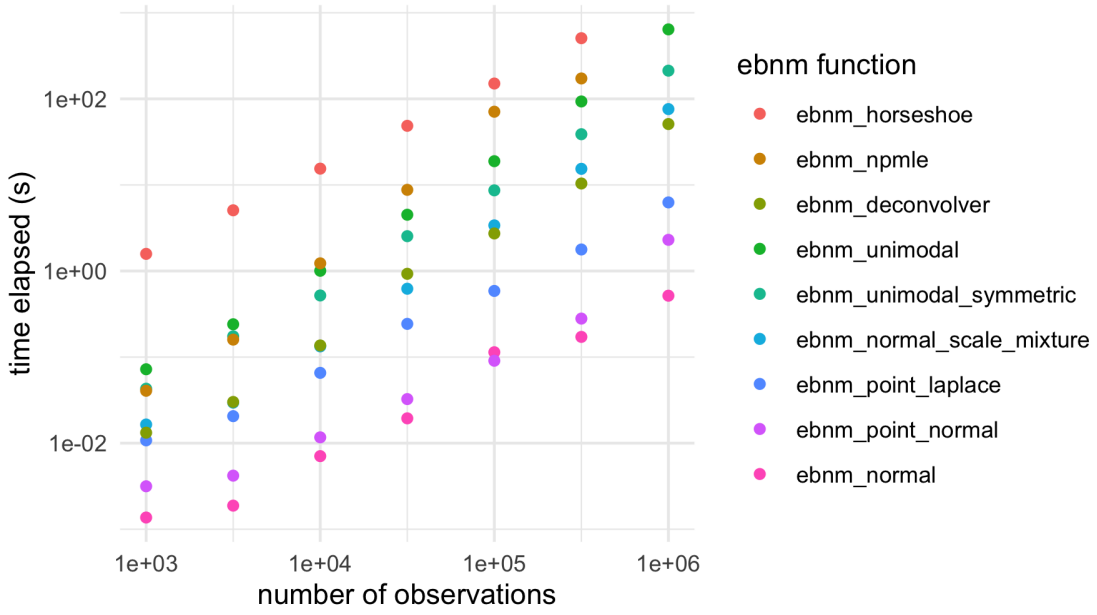


Figure 1.4: Runtime for variously sized datasets and for different choices of prior family.

1.4.1 Timing Comparisons

To provide an overview of how the choice of prior family affects runtime, I fit **ebnm** to datasets simulated from a point-t prior (Equation 1.68) with the number of observations ranging from 10^3 to 10^6 (I experimented with different data-generating distributions, but results did not substantially change). All trials were performed on a 2018 MacBook Pro with a 2.6 GHz Intel Core i7 processor and 16 GB of DDR4-2400 RAM.

Results are displayed in Figure 1.4. In general, there is a clear correlation between runtime and the size of the prior family. Note that this is not an obvious result, since the EBNM problem is convex for the (discretized) nonparametric prior families discussed in Sections 1.3.2 and 1.3.4 but not in general convex for the parametric point-normal and point-Laplace families (see Section 1.3.1). The family of horseshoe distributions is an exception: although it is parametrized by a single parameter, it is slower than nearly every other family by an order of magnitude or more.

Function	Log Likelihood	RMSE	CI Coverage
ebnm_normal	−28.2	0.536	0.942
ebnm_point_normal	−3.4	0.443	0.948
ebnm_point_laplace	−3.9	0.443	0.954
ebnm_normal_scale_mixture	−3.3	0.442	0.951
ebnm_unimodal_symmetric	−2.8	0.443	0.943
ebnm_unimodal	−1.1	0.445	0.942
ebnm_npmle	0.0	0.450	0.893
ebnm_horseshoe	−14.1	0.446	0.946
ebnm_deconvolver	−8.0	0.476	0.951

Figure 1.5: Simulation results for the point-normal data-generating prior. See text for details.

1.4.2 Predictive Performance Comparisons

For each of the data-generating distributions listed above, I run ten simulations in which I simulate $n = 1000$ "true means" $\boldsymbol{\theta}$, add $\mathcal{N}(0, 1)$ noise, and then estimate $\boldsymbol{\theta}$ using various prior families. In Figures 1.5-1.7, I show the mean log likelihood attained by each prior family relative to the `ebnm_npmle` log likelihood; the root mean-squared error

$$\sqrt{\sum_{i=1}^{n} \left(\hat{\theta}_i - \theta_i\right)^2}, \tag{1.70}$$

where $\hat{\theta}_i$ is the posterior mean $\mathbb{E}\left(\theta_i \mid x_i, s_i, \hat{g}\right)$; and the proportion of true means that are covered by 90% credible intervals (which were obtained via the posterior samplers returned by **ebnm**).

In every case, `ebnm_npmle` attains the maximum log likelihood, which is expected since $\mathcal{G}_{\text{np}}$ includes every other prior family as a subfamily. However, it does slightly worse in terms of RMSE than prior families that encode assumptions about sparsity, provided that these families are flexible enough to model the data-generating distribution (for example, only the family of unimodal priors is flexible enough to model the asymmetric tophat).

Function	Log Likelihood	RMSE	CI Coverage
ebnm_normal	−62.5	0.663	0.927
ebnm_point_normal	−7.1	0.531	0.896
ebnm_point_laplace	−5.5	0.527	0.920
ebnm_normal_scale_mixture	−4.5	0.527	0.922
ebnm_unimodal_symmetric	−4.2	0.528	0.903
ebnm_unimodal	−1.8	0.529	0.904
ebnm_npmle	0.0	0.534	0.790
ebnm_horseshoe	−17.1	0.532	0.917
ebnm_deconvolver	−18.5	0.582	0.928

Figure 1.6: Simulation results for the point-t data-generating prior.

Function	Log Likelihood	RMSE	CI Coverage
ebnm_normal	−277.1	0.971	0.898
ebnm_point_normal	−122.5	0.841	0.914
ebnm_point_laplace	−146.4	0.849	0.922
ebnm_normal_scale_mixture	−125.9	0.841	0.913
ebnm_unimodal_symmetric	−103.0	0.833	0.907
ebnm_unimodal	−5.4	0.820	0.910
ebnm_npmle	0.0	0.830	0.801
ebnm_horseshoe	−269.3	0.864	0.912
ebnm_deconvolver	−36.9	0.881	0.899

Figure 1.7: Simulation results for the asymmetric tophat data-generating prior.

Further, the NPMLE does very poorly in terms of credible interval coverage, which suggests that it's much more useful for point estimates than for uncertainty estimates. Results for `ebnm_deconvolver` are also poor, but recall that **deconvolveR** estimates smooth but non-sparse priors, so it's hardly surprising that it's bested by prior families that are able to model sparsity directly.

1.5 Real-Data Examples

1.5.1 Eight Schools

As a first example of how **ebnm** can yield insights into real data — and how, vice versa, real data can yield insights into the EBNM problem — I consider the well-known "eight schools" dataset originally published by Rubin [1981] and subsequently discussed by Gelman et al. [2014]. In each of eight US high schools, randomized experiments were conducted in order to estimate the effect of coaching programs on SAT scores. The data includes estimates of treatment effects and standard errors.

I used a point-normal prior family to fit the data with, first, the mode fixed at zero (Figure 1.8) and, second, the mode to be estimated (Figure 1.9). Since the first prior family models sparsity, it's more useful for testing whether treatment effects are non-null, while the second prior family is potentially more useful for modelling variation across schools.

In both cases, the prior g is estimated to be a point mass (either δ_0 or δ_μ with $\mu \approx 7.7$), which is consistent with a lack of variation across schools. (The difference in log likelihoods is about 1.8, so that the prior δ_μ is $\exp(1.8) \approx 6$ times more likely than the prior δ_0.) A possibly unintended consequence is that credible intervals have width zero. Indeed, a known limitation of empirical Bayes methods is that they don't account for uncertainty in the estimate of g. In the eight schools example, the effect is especially exaggerated, since EB estimates yield credible intervals that are useless for all practical purposes. A fully Bayesian

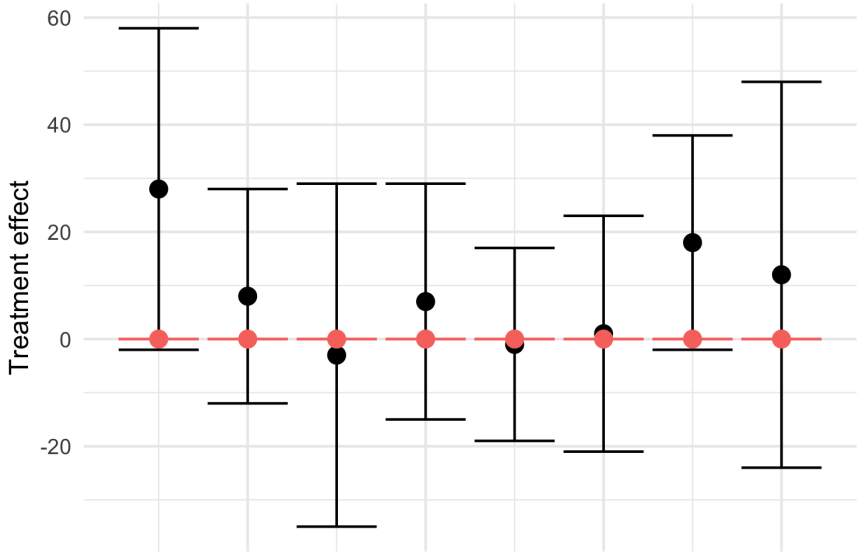


Figure 1.8: Results for the eight schools example (Rubin [1981]) with `mode = 0`. The data is in black, where the points are the treatment effect estimates and the error bars are ± 2 standard errors; the **ebnm** estimates of the "true" means and 95% credible intervals are in red. Since `ebnm_point_normal` estimates the prior g to be essentially a point mass δ_0, the credible intervals are very narrow.

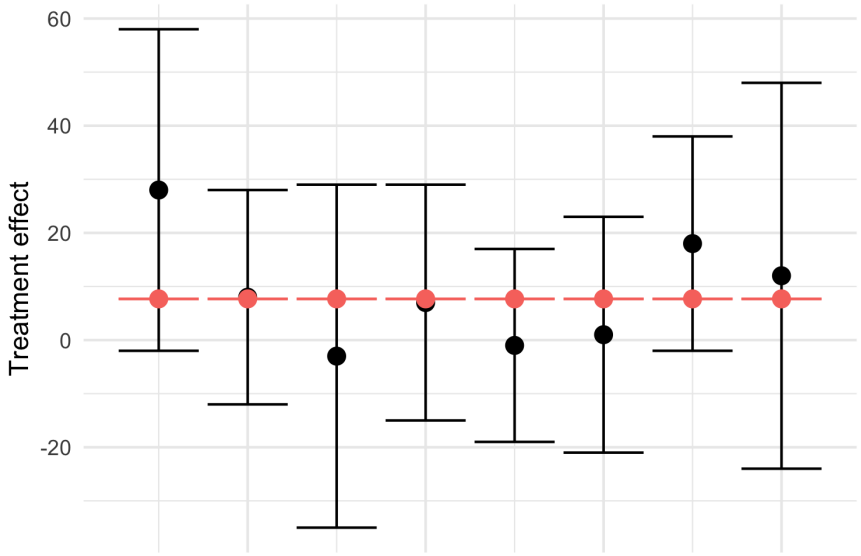


Figure 1.9: Results for the eight schools example with `mode = "estimate"`. The prior is estimated to be a point mass at δ_μ, where $\mu \approx 7.7$.

method would be more appropriate for interval estimation.

1.5.2 GTEx

Next I consider the GTEx dataset used in Wang and Stephens [2021] (see the introduction to this chapter for a brief overview of EBMF). The dataset is derived from data made available by the Genotype Tissue Expression (GTEx) project (Lonsdale et al. [2013]), which provides z-scores for the effects of SNPs on gene expression across 44 human tissues. To reduce the data to a more manageable size, Urbut et al. [2019] choose the "top" eQTL for each gene — that is, the SNP associated with the largest (absolute) z-score over all 44 tissues. This selection process yields a $16,069 \times 44$ matrix of z-scores, with rows corresponding to SNP-gene pairs and columns corresponding to tissues.

First, I ran EBMF using point-normal priors (while Wang and Stephens use scale mixtures of normals, I find that results are very similar, at least to the eye, for the point-normal and scale-mixture-of-normal prior families). Next, I obtained a semi-nonnegative matrix factorization (SNMF) by using a nonnegative prior family (Equation 1.19) for loadings (tissues) and point-normal priors for factors (eQTLs).

Results are shown in Figures 1.10-1.12. I sorted point-normal results by descending proportion of variance explained and then "matched" point-normal factors with semi-nonnegative factors for ease of comparison. While results are similar, many of the semi-nonnegative factors are noticeably cleaner (2, 4, 6, 7, 10, 12, 15, and especially 8), while only one factor (16) gets somewhat busier. (I note, however, that it's possible that this "busyness" is biologically relevant.) The greatest advantage to SNMF, I would argue, is the lack of anti-correlations, which are somewhat difficult to motivate biologically in this particular setting. Consider Factor 10, for example: the semi-nonnegative factorization suggests only that there is correlation of eQTL effect sizes in cerebellar hemisphere and cerebellum tissues, while the point-normal factorization suggests that eQTLs that increase gene activity in cerebellar tissues also *de-*

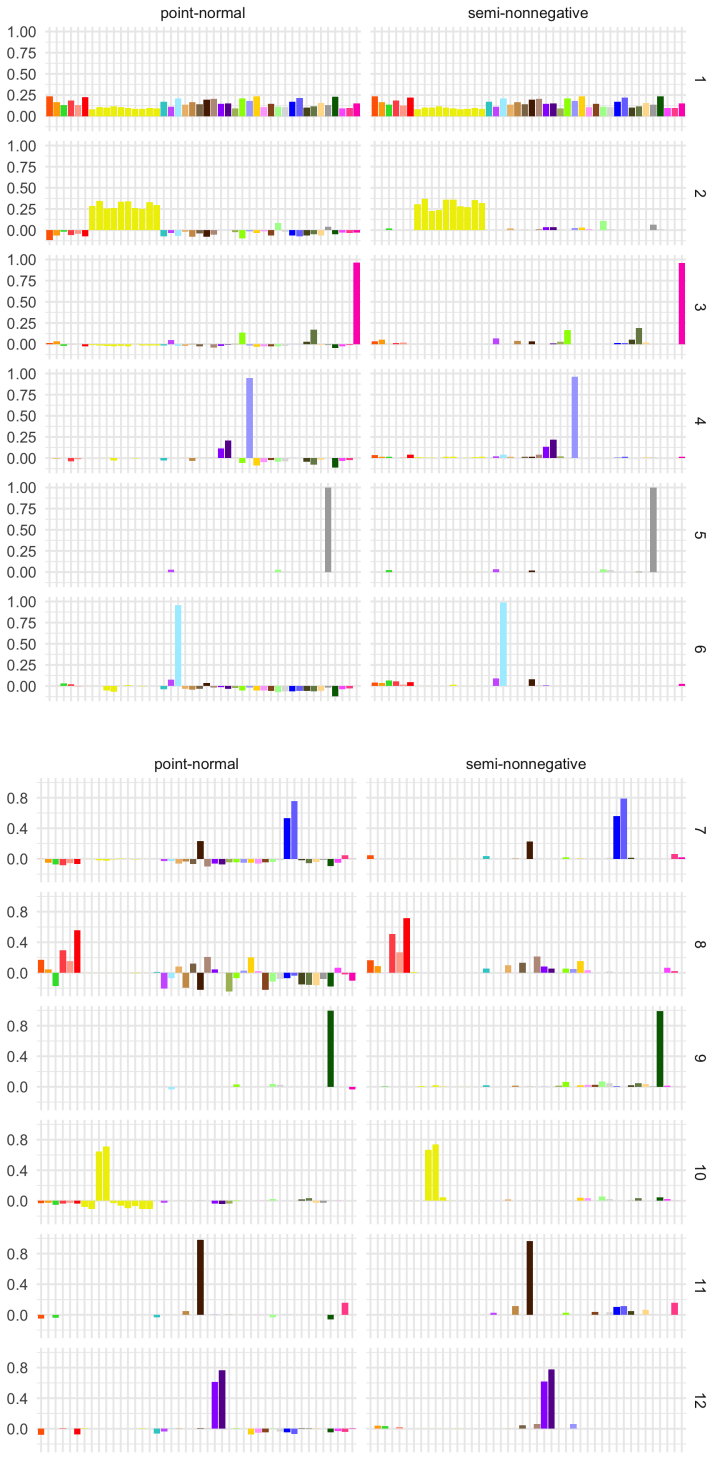


Figure 1.10: GTEx data (Factors 1-12). See Figure 1.12 for a legend.

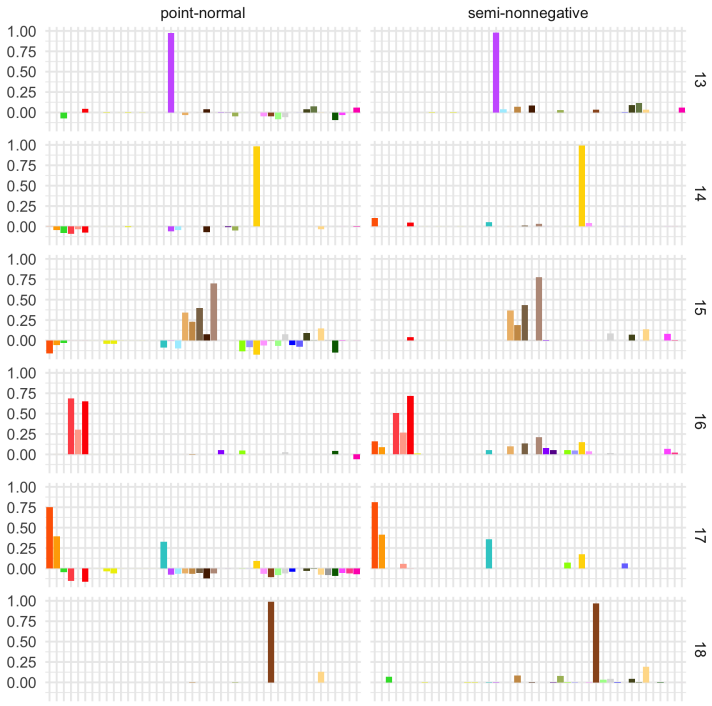


Figure 1.11: GTEx data (Factors 13-18). See Figure 1.12 for a legend.

Figure 1.12: GTEx data legend.

crease it in other brain tissues, which seems to me to be somewhat less plausible from a biological standpoint.

For a different semi-nonnegative matrix factorization method which also uses GTEx data as an application (but a much larger dataset), see He et al. [2020].

1.6 Discussion

The main objective of this chapter has been to introduce R package **ebnm**, which provides common inputs and outputs for solving the EBNM problem with different choices of prior family $\mathcal{G}$. Since there is a trade-off between speed and flexibility, and since different prior families $\mathcal{G}$ encode different assumptions about the prior g, $\mathcal{G}$ should be chosen with the application in mind. When the number of observations is large or when many EBNM problems must be solved in succession, smaller prior families such as $\mathcal{G}_{\text{pn}}$ or $\mathcal{G}_{\text{pl}}$ (Equations 2.5 and 1.5) are generally good choices. When speed is less critical and little is known about the distribution of means, then a more flexible family such as $\mathcal{G}_{\text{smn}}$ or $\mathcal{G}_{\text{symm}}$ might be preferred.

A secondary aim has been to provide guidelines for nonparametric prior families so that a "good enough" solution is obtained. In Section 1.3.3, I argued that an approximate MLE is good enough when it differs from the exact MLE by the same order of magnitude as the exact MLE differs from the "true" prior, and I gave grid selection details for scale mixtures of distribution. For unimodal and symmetric unimodal priors, which are represented as scale mixtures of uniforms, the math is not as straightforward. In package **ebnm**, I use the same grids for scale mixtures of normals and uniforms, but it's far from obvious that this is the correct thing to do. Indeed, Figure 1.13 suggests that while in most cases the default grid will give a solution that is reasonably close to optimal (within one or two log likelihood units), it can also give solutions that are quite poor. I reserve a rigorous examination of unimodal grid selection for future work.

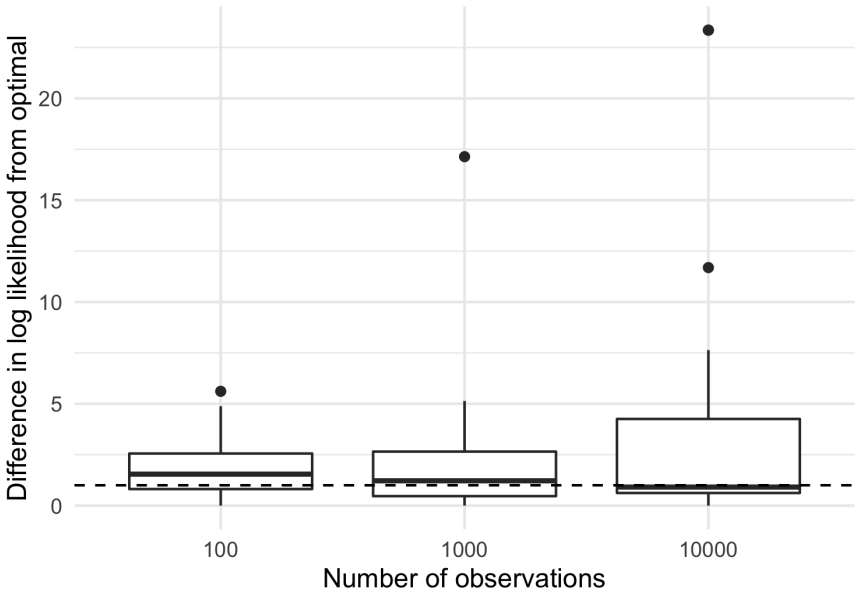


Figure 1.13: Difference in log likelihood between the `ebnm_symmetric_unimodal` solution obtained using the default grid and using an estimated optimal grid. Each data-generating distribution is a mixture of uniforms consisting of one to five components Unif$[-a_k, a_k]$, with $a_k \sim \text{Exp}(1/5)$.

1.7 Supplementary Benchmarking Results

1.7.1 Optimization Methods for Parametric Families

I first compare the performance of six optimization methods, all of which are implemented in **ebnm** via parameter `optmethod`. Three call into function `nlm`, a Newton-type algorithm included in the base **stats** package. Gradient and Hessian functions can be passed into `nlm`; if not, they are estimated numerically. Option `optmethod = "nlm"` provides both the gradient and Hessian functions; `optmethod = "nohess_nlm"` provides the gradient but not the Hessian; `optmethod = "nograd_nlm"` provides neither. Options `optmethod = "lbfgsb"` and `optmethod = "nograd_lbfgsb"` call into function `optim`, also in the **stats** package, with argument `method = "L-BFGS-B"`. The former provides the gradient function; the second does not. By definition, L-BFGS-B does not accept a Hessian. Finally, `optmethod = "trust"` calls into function `trust`, a trust-region algorithm implemented in package **trust**.

Since `trust` requires both a gradient and Hessian function, there is only one corresponding `optmethod`. In sum, then, there are two methods that use both gradients and Hessians (`nlm` and `trust`); two that use only gradients (`nohess_nlm` and `lbfgsb`); and two that estimate all derivatives numerically (`nograd_nlm` and `nograd_lbfgsb`).

For both `ebnm_point_normal` and `ebnm_point_laplace`, I ran tests for $2 * 3 * 2 * 3 = 36$ scenarios:

- The mode is either fixed at zero via argument `mode = 0` or estimated (via `mode = "estimate"`).
- The data-generating prior distribution g is: a true member of the prior family $\pi_0\delta_\mu + (1-\pi_0)N(\mu, a^2)$ or $\pi_0\delta_\mu + (1-\pi_0)\text{Laplace}(\mu, a)$, with $\pi_0 \sim \text{Beta}(10, 2)$, $a \sim \text{Gamma}(4, 1)$, and either $\mu = 0$ or $\mu \sim \text{Unif}[-10, 10]$; the null distribution δ_0; or a distribution not in the prior family, so that $\mathcal{G}$ is misspecified. When `mode = 0`, the misspecified prior is a point-normal prior as above but with $\mu \sim \text{Unif}[-10, 10]$; when `mode = "estimate"`, the misspecified prior is the point-t_5 distribution $\pi_0\delta_0 + (1 - \pi_0)at_5$, with π_0 and a distributed as above.
- The $N(0, s_i^2)$ noise added to the true means $\theta_i \sim g$ is either homoskedastic, with $s_i = 1$ for all i, or heteroskedastic, with $s_i^2 \sim \text{Exp}(1)$.
- The number of observations n is 1000, 10000, or 100000.

For each scenario, I ran $10^6/n$ simulations (so, depending on n, 1000, 100, or 10 simulations) and compared runtimes using package **microbenchmark**. All experiments were performed on a 2018 MacBook Pro with a 2.6 GHz Intel Core i7 processor and 16 GB of DDR4-2400 RAM. Results are displayed in Figures 1.14 and 1.15.

In general, the methods that supplied the gradient function outperformed the methods that required derivatives to be estimated numerically. Of the four methods that do supply

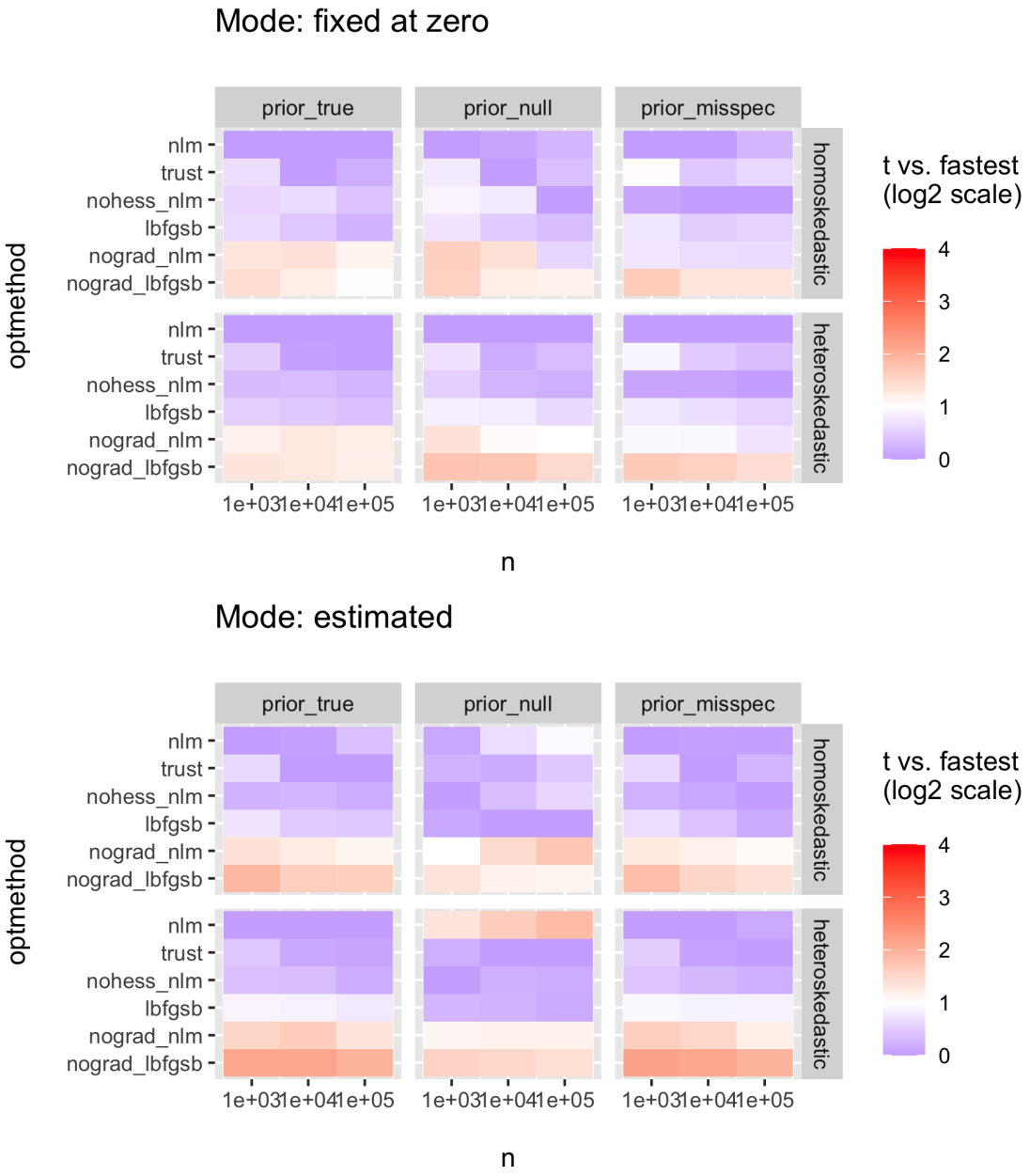


Figure 1.14: Timing comparisons for `ebnm_point_normal`. Tests in the top figure fix the mode at zero (by setting parameter `mode = 0`) while those in the bottom estimate the mode (by setting `mode = "estimate"`). Different columns correspond to different data-generating priors: a true member of the point-normal prior family; the null distribution δ_0; or a distribution from outside the prior family. Different rows correspond to different noise models, with the noise added to the "true" observations either homoskedastic (with $s_i = 1$ for all i) or heteroskedastic (with $s_i^2 \sim \text{Exp}(1)$). Plotted is the time as a multiple of the fastest time for a given combination of mode, prior, noise, and value of n. Shades of blue indicate optimization methods that are within a factor of 2 of the fastest method for that mode, prior, noise, and value of n, while shades of red indicate slower methods.

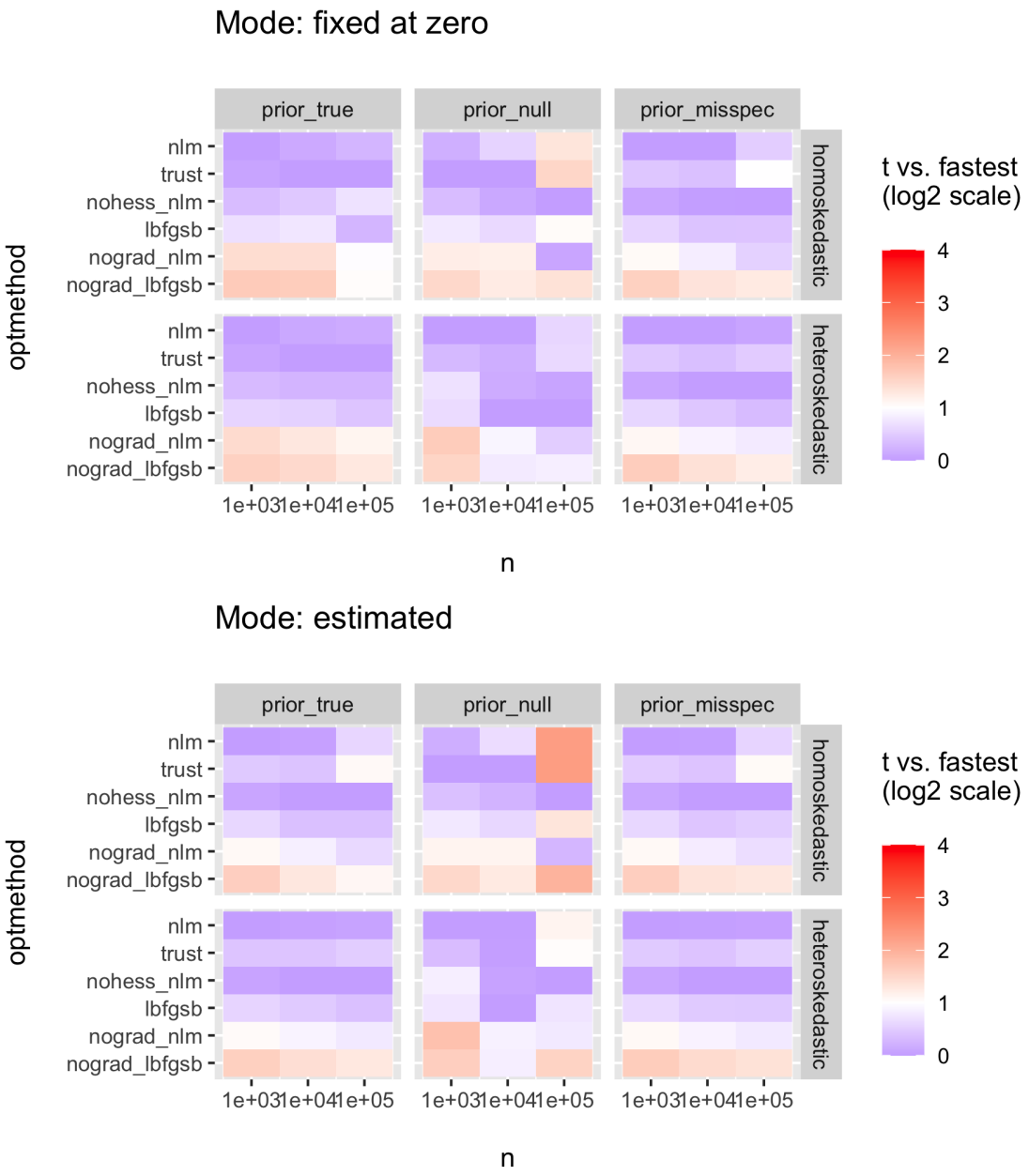


Figure 1.15: Timing comparisons for `ebnm_point_laplace`. See Figure 1.14 caption and text for details.

the gradient, exactly one, `nohess_nlm`, was "blue" across the board — that is, it was within a factor of 2 of the fastest method for every scenario. The methods that supply Hessians (`nlm` and `trust`) can struggle when the data-generating prior is the null distribution δ_0. Finally, method `lbfgsb` did nearly as well as `nohess_nlm` in terms of runtime but it errored out in several of the `prior_null` simulations (in some scenarios, up to 10% of simulations resulted in an error). I thus feel confident in recommending the default setting `optmethod = "nohess_nlm"`.

1.7.2 Comparisons with Existing Packages

Next I compare the performance of **ebnm** against three packages with directly comparable functions: `ebnm_point_laplace` is closely related to function `ebayesthresh` in package **EbayesThresh**; `ebnm_normal_scale_mixture` is modelled on function `ash` in package **ashr** (with option `mixcompdist = "normal"`) but, as mentioned above, is implemented in a much simpler manner; and `ebnm_npmle` is very similar to function `GLmix` in package **REBayes**.

I ran tests for the same scenarios as the previous subsection, with the difference that the mode is always fixed at zero (mode estimation is not possible with **EbayesThresh**). Further, since it's not possible to "misspecify" the prior for the family of all distributions $\mathcal{G}_{\text{np}}$, I only considered a single data-generating distribution (the point-Laplace), but I varied the number of grid points (mixture components) from 10 to 300. The number of simulations, `microbenchmark` settings, and hardware were as described in the previous section. I set parameters to make outputs as similar as possible. For **EbayesThresh**, I set `threshrule = "mean"` and `universalthresh = FALSE`; for `ash`, I set `prior = "uniform"`. Results are given in Figures 1.16-1.18.

In almost every scenario, package **ebnm** outperformed both **EbayesThresh** and **ashr**. `ash` was consistently slower than `ebnm_normal_scale_mixture` by a factor of 2 to 4 except for large n with heteroskedastic errors. In some scenarios, **EbayesThresh** was nearly as

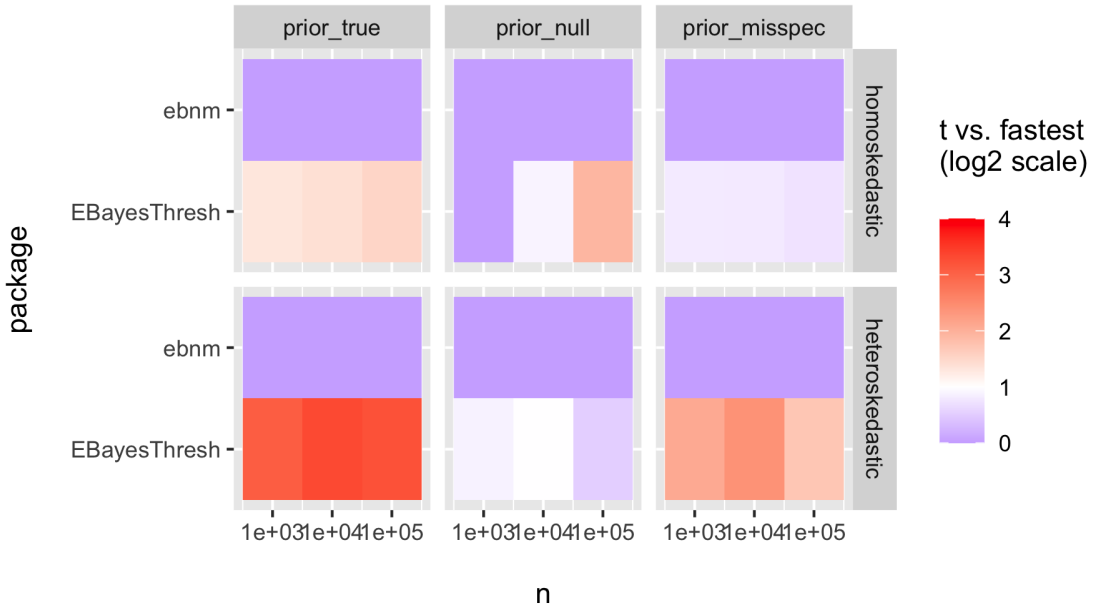


Figure 1.16: Timing comparisons: `ebnm_point_laplace` vs. `ebayesthresh`.

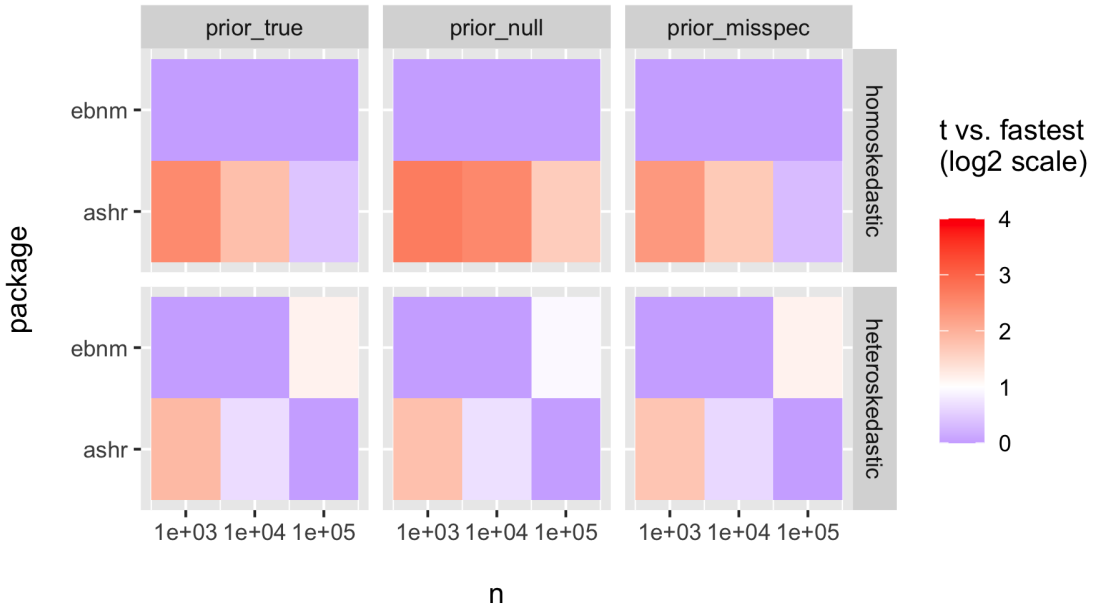


Figure 1.17: Timing comparisons: `ebnm_normal_scale_mixture` vs. `ash`.

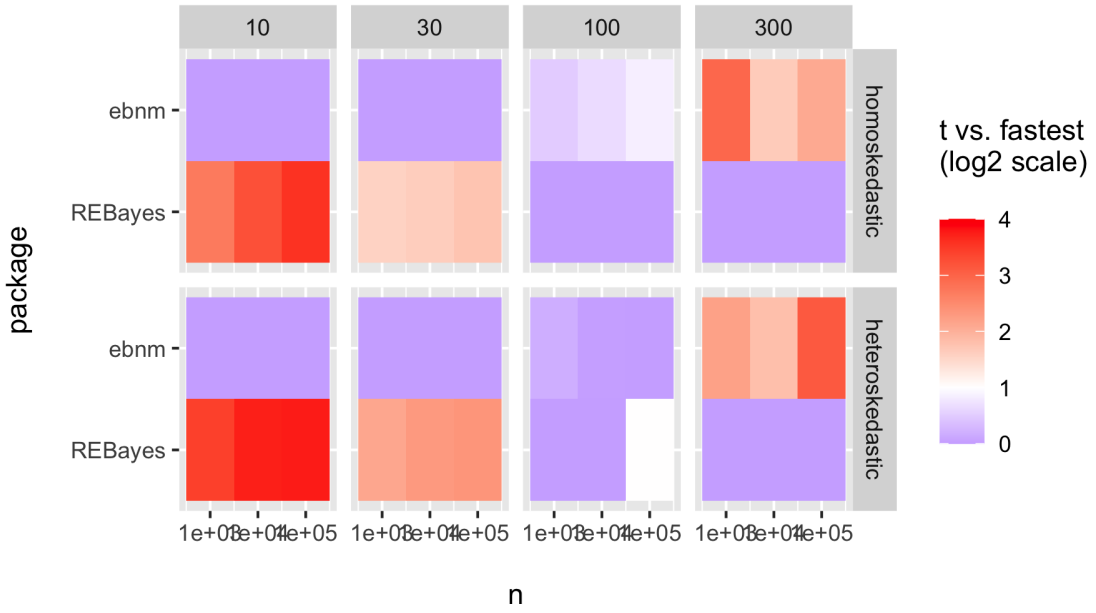


Figure 1.18: Timing comparisons: `ebnm_npmle` vs. **REBayes**.

fast as `ebnm_point_laplace`, but in others it was outperformed by a full order of magnitude. Further, **ebnm** regularly found significantly better solutions than **EbayesThresh** (in terms of the final objective attained) except when the data-generating prior was the null distribution, in which cases the packages found solutions of similar quality.

Results in the comparison between `ebnm_npmle` and **REBayes** were mixed. **ebnm** was regularly faster when the number of mixture components was small (fewer than 100), while **REBayes** was consistently faster when a dense grid was used (more than 100 components). According to the theory developed in Section 1.3.4, 100 components should be "good enough" for homoskedastic observations when

$$n^{1/4}\left(\frac{\text{range}(x)}{s}\right) \leq 400. \tag{1.71}$$

For example, if the number of observations n is equal to 10^6, then 100 components should suffice as long as

$$\frac{\max(x) - \min(x)}{s} \leq \frac{40}{\sqrt{10}}. \tag{1.72}$$

1.8 Supplementary Example with Code: MLB Data

In the empirical Bayes literature, it has become almost customary to include an analysis of batting averages using Major League Baseball (MLB) data: see, for example, Brown [2008]; Jiang and Zhang [2010]; and Gu and Koenker [2017]. Historically, batting averages have been the most important measurement of a hitter's performance, with the prestigious "batting title" going to the hitter with the highest average. However, with the rise of baseball analytics, metrics that better correlate to teams' overall run production have become increasingly widely used. As a supplementary example, I consider wOBA (weighted on-base average), which is both an excellent measure of a hitter's overall offensive production and, unlike competing metrics such as Statcast's xwOBA or Baseball Prospectus's DRC+, can be calculated using data and methods that are publicly available.

The idea, which was initially proposed by Tango et al. [2007], is that different values ("weights") can be assigned to different hitting outcomes according to how much the outcome contributes to run production. For example, while batting average treats singles identically to home runs, wOBA gives a hitter more than twice as much credit for a home run. This makes sense because, all else being equal, a home run is guaranteed to score more runs than a single (the 2020 wOBA weights for singles and home runs were, respectively, 0.883 and 1.979).

Formally, wOBA models innings as Markov chains over "game states" $\mathcal{Y}$ (runs scored, outs, and baserunners):

$$\{Y_0 = (\text{no runs, no outs, no runners}), Y_1, \ldots, Y_T = (\text{R runs, three outs})\}. \tag{1.73}$$

Transition probabilities vary according to the event (e.g., single, home run, strikeout) that occurs at time t:

$$P_{i,j}^{x} = \mathbb{P}\left(Y_t = j \;\middle|\; Y_{t-1} = i, X_t = x\right). \tag{1.74}$$

Assuming independence of game states and events, the average number of runs created by event x can be calculated as

$$w(x) := \int_{i \in \mathcal{Y}} \left[\mathbb{E}\left(R \mid Y = j\right) - \mathbb{E}\left(R \mid Y = i\right) \right] \mu(i) P^x_{i,j}. \tag{1.75}$$

(In practice, expected runs, event probabilities $\mu(i)$, and transition probabilities $P^x_{i,j}$ are all estimated using empirical proportions from a season's worth of data.) Finally, the "weights" $w(x)$ are shifted and scaled so that $w(\text{out}) = 0$ and the league-average wOBA is equal to the league-average on-base percentage.

Given a vector of wOBA weights $\mathbf{w}$, hitter j's wOBA is simply

$$\frac{1}{n_j} \mathbf{w}^T \mathbf{z}^{(j)}, \tag{1.76}$$

where $\mathbf{z}^{(j)}$ tallies outcomes over the hitter's n_j plate appearances. This can be regarded as a point estimate $\hat{x}_j$ for the hitter's "true" wOBA skill

$$x_j := \mathbf{w}^T \boldsymbol{\pi}^{(j)}, \tag{1.77}$$

where $\boldsymbol{\pi}^{(j)}$ is the vector of "true" outcome probabilities for hitter j.

To obtain shrunken estimates of the x_js using **ebnm**, standard errors are required as well as point estimates. I derive these standard errors by assuming that

$$\mathbf{z}^{(j)} \overset{\text{ind}}{\sim} \text{Multinomial}(n_j, \boldsymbol{\pi}^{(j)}) \tag{1.78}$$

and plugging in

$$\hat{\boldsymbol{\pi}}^{(j)} = \frac{1}{n_j} \mathbf{z}^{(j)}. \tag{1.79}$$

To handle small sample sizes, I conservatively lower bound each standard error by the stan-

dard error that would be obtained by plugging in league-average event probabilities

$$\hat{\boldsymbol{\pi}}_{\text{lg}} = \frac{1}{\sum_j n_j} \sum_j \mathbf{z}^{(j)}. \tag{1.80}$$

In the following example, I estimate hitters' wOBA skill using only data from 2020, then using only data from 2019, then using data from 2018-2020 using a 3/4/5 weighting scheme that is a commonly used rule of thumb in sabermetrics. I believe that the comparison between the 2019 season, in which a full complement of 162 games was played, and the 2020 season, which was shortened to 60 games as a result of the COVID-19 pandemic, provides a convenient illustration of how much can be concluded from a single season's worth of hitting statistics. The answer turns out to be surprisingly little, especially in 2020, so I include the three-year calculations as better estimates of hitters' abilities at the time of writing (namely, the 2020-2021 offseason).

I use (asymmetric) unimodal priors because there is no reason to expect players' wOBA to be symmetrically distributed around the league average (and in fact, I'd expect the distribution to be right-skewed because poor hitters will be demoted to the minor leagues). I estimate the mode and set `output = output_all()` to obtain (among other things) a function that will sample from the posterior:

```
R> ebnm2020 <- ebnm_unimodal(wOBA2020$wOBA, wOBA2020$wOBA_sd,
R>                           mode = "estimate",
R>                           output = output_all())
R> wOBA2020 <- wOBA2020 %>%
R>   add_column(year = "2020", postmean = ebnm2020$posterior$mean)
R>
R> ebnm2019 <- ebnm_unimodal(wOBA2019$wOBA, wOBA2019$wOBA_sd,
R>                           mode = "estimate",
R>                           output = output_all())
R> wOBA2019 <- wOBA2019 %>%
R>   add_column(year = "2019", postmean = ebnm2019$posterior$mean)
```

I plot results as follows:

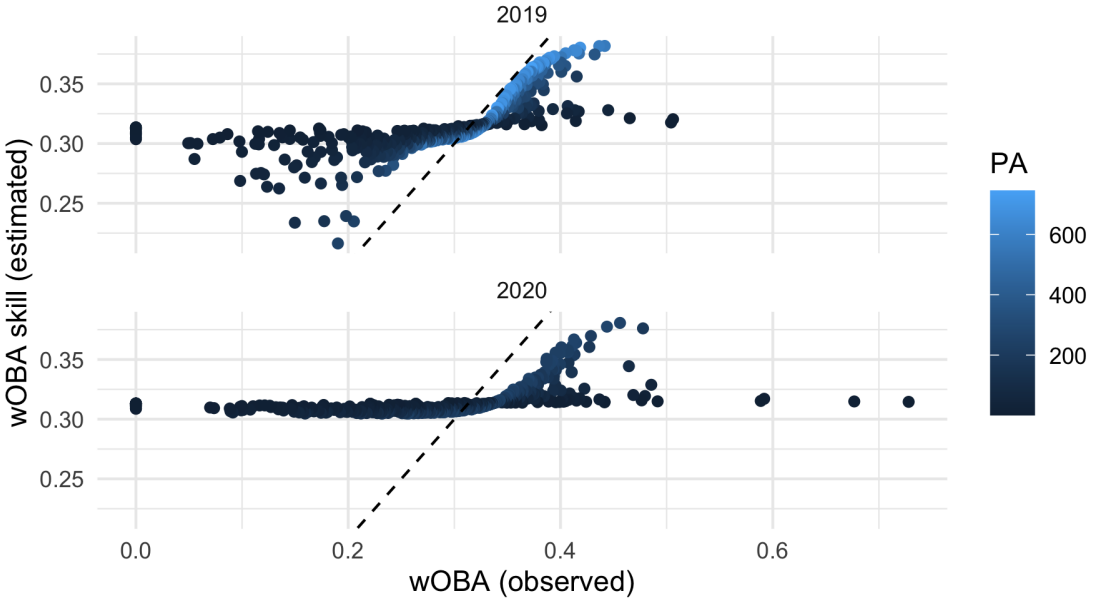


Figure 1.19: Estimates of true wOBA skill obtained by separately running `ebnm_unimodal` on MLB data from the 2019 and 2020 seasons. "PA" indicates the number of plate appearances. The dashed line is the line $y = x$: estimates to the right of the line are less than the corresponding observations and estimates to the left are greater.

```
R> ggplot(wOBA2020 %>% bind_rows(wOBA2019),
R>        aes(x = wOBA, y = postmean, col = n)) +
R>   geom_point() +
R>   geom_abline(slope = 1, linetype = "dashed") +
R>   labs(x = "wOBA (observed)", y = "wOBA skill (estimated)", col = "PA") +
R>   facet_wrap(~year, nrow = 2) +
R>   theme_minimal()
```

The output is displayed in Figure 1.19. There is very strong shrinkage towards the mode for the 2020 season. In particular, none of the posterior means are far below the league mode. For both seasons, there is also strong shrinkage near the top of the range of observed values, which is especially characteristic of the family of unimodal priors.

To interpret these results, it's useful to compare against FanGraphs' "wOBA rules of thumb," according to which .400 can be considered as "excellent," .370 as "great," and .340 as "above average." I sample from the posterior (using the sampler obtained by setting `output = output_all()`) to estimate the probabilities that, based solely on the results of a

single season, a hitter can be described as excellent, great, or above average.

```
R> samp2020 <- ebnm2020$posterior_sampler(5000)
R> samp2019 <- ebnm2019$posterior_sampler(5000)
R>
R> hitter_quality <- function(samp) {
R>   c("excellent" = mean(samp > .4),
R>     "great" = mean(samp > .37),
R>     "above_avg" = mean(samp > .34))
R> }
R>
R> wOBA2020 <- wOBA2020 %>%
R>   bind_cols(as_tibble(t(apply(samp2020, 2, hitter_quality))))
R> wOBA2019 <- wOBA2019 %>%
R>   bind_cols(as_tibble(t(apply(samp2019, 2, hitter_quality))))
```

For example, the number of hitters per season that can with at least 80% probability be described as excellent, great, or above average (again, based solely on the single season results) is:

```
R> wOBA2020 %>%
R>   bind_rows(wOBA2019) %>%
R>   group_by(year) %>%
R>   summarize_at(vars(excellent:above_avg), ~sum(. > .8)) %>%
R>   as.data.frame()
  year excellent great above_avg
1 2019         0     3        47
2 2020         0     1         6
```

I surmise that the average baseball fan would be quite surprised to learn that only six hitters performed well enough in 2020 to be confidently described as "above average"! In addition to the National League MVP Freddie Freeman (the only hitter to qualify as "great" with greater than 80% probability), Marcell Ozuna, Juan Soto, DJ LeMahieu, Trea Turner, and Jose Ramirez made the cut.

To get better estimates of hitters' current ability (as of the date of writing), I combine data from 2018 to 2020 using a 3/4/5 weighting scheme that is common in sabermetrics. To be precise, denoting θ as a hitter's true wOBA skill, I model observed wOBAs from 2018 to

2020 as:

$$\text{wOBA}_{2020} \sim \mathcal{N}\left(\theta, s_{2020}^2\right), \tag{1.81}$$

$$\text{wOBA}_{2019} \sim \mathcal{N}\left(\theta, \left(\frac{4s_{2019}}{3}\right)^2\right), \tag{1.82}$$

$$\text{wOBA}_{2018} \sim \mathcal{N}\left(\theta, \left(\frac{5s_{2018}}{3}\right)^2\right), \tag{1.83}$$

where s_{2018}, s_{2019}, and s_{2020} denote the standard errors in the wOBA estimates, as described at the beginning of this section. I combine observations in the usual manner (with weights proportional to precision) and then fit **ebnm** as above. This yields 31 hitters who can be described as "above average" with at least 80% probability. For a complete list, along with observed wOBA, estimated wOBA skill, and posterior probabilities of being "excellent," "great," or "above average," see Table 1.2.

Name	wOBA	wOBA_sd	postmean	p_excellent	p_great	p_above_avg
Mike Trout	0.431	0.021	0.396	0.339	0.855	1.000
Freddie Freeman	0.402	0.018	0.376	0.046	0.528	0.996
Juan Soto	0.411	0.020	0.379	0.084	0.586	0.995
Anthony Rendon	0.400	0.019	0.374	0.037	0.467	0.993
Christian Yelich	0.403	0.020	0.374	0.041	0.474	0.992
Mookie Betts	0.398	0.019	0.372	0.022	0.428	0.989
Alex Bregman	0.395	0.019	0.371	0.019	0.396	0.989
Nelson Cruz	0.399	0.022	0.369	0.022	0.334	0.972
Xander Bogaerts	0.380	0.019	0.362	0.003	0.185	0.948
Ronald Acuna Jr.	0.383	0.020	0.363	0.006	0.195	0.943
Bryce Harper	0.377	0.018	0.361	0.002	0.154	0.936
Trevor Story	0.378	0.019	0.361	0.002	0.161	0.930
DJ LeMahieu	0.374	0.018	0.359	0.001	0.123	0.925
Justin Turner	0.377	0.020	0.359	0.002	0.143	0.906
Cody Bellinger	0.372	0.019	0.358	0.001	0.116	0.905
George Springer	0.375	0.020	0.358	0.002	0.123	0.903
Fernando Tatis Jr.	0.395	0.026	0.360	0.010	0.209	0.898
Jose Ramirez	0.372	0.019	0.357	0.002	0.116	0.890
Paul Goldschmidt	0.368	0.018	0.356	0.000	0.085	0.890
Jeff McNeil	0.374	0.021	0.357	0.001	0.119	0.875
Aaron Judge	0.384	0.024	0.358	0.005	0.158	0.871
Charlie Blackmon	0.367	0.018	0.356	0.001	0.085	0.870
Michael Conforto	0.366	0.018	0.355	0.000	0.081	0.860
Nolan Arenado	0.369	0.019	0.355	0.001	0.084	0.856
Luke Voit	0.376	0.023	0.355	0.002	0.114	0.836
Anthony Rizzo	0.363	0.018	0.353	0.000	0.064	0.834
J.D. Martinez	0.365	0.019	0.353	0.000	0.073	0.830
Michael Brantley	0.364	0.018	0.353	0.000	0.069	0.826
Brandon Nimmo	0.373	0.022	0.354	0.002	0.103	0.817
Trea Turner	0.363	0.018	0.353	0.001	0.056	0.816
Eugenio Suarez	0.366	0.020	0.353	0.001	0.079	0.814

Table 1.2: Hitters that have a 80% or greater probability of being "above average" (wOBA skill $\geq$ .340) based on their performance during the 2018-2020 seasons, using a 3/4/5 weighting scheme. According to FanGraphs, an "excellent" wOBA is over .400, while a "great" wOBA is over .370.

CHAPTER 2

FLASHIER: EBMF FOR SCRNA-SEQ DATA

2.1 Introduction

In this chapter, I argue for the usefulness of empirical Bayes matrix factorization (EBMF) as a tool for exploring and analyzing single-cell RNA sequencing (scRNA-seq) data. EBMF is as fast as or faster than most other commonly used model-based matrix factorization methods, and often outperforms them on important tasks — as an example, I will show that EBMF detects rare cell types with relative ease. Since the EBMF model assumes Gaussian errors, the application to count data requires some care, but as I hope to show, the method performs well enough for EBMF to merit inclusion in any practitioner's toolkit.

The use of dimensionality reduction techniques has long been standard in scRNA-seq data analysis. Most often, one uses a method such as t-SNE (Van der Maaten and Hinton [2008]) or UMAP (McInnes et al. [2018]) to find a two-dimensional embedding of the data in order to identify clusters of cells. While essential for exploratory analysis, such methods can be blunt instruments. More recently, a number of practitioners (see, for example, Bielecki et al. [2021]) have deployed matrix factorization methods such as topic modeling, which can paint a more nuanced picture of cell population structure while also yielding results that can be used in downstream analysis. For example, factors can show gradual transitions among and within cell types, and factor loadings can link cell types to expression profiles.

EBMF (Wang and Stephens [2021]) fits the following model:

$$\mathbf{Y} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{2.1}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}_\ell^{(k)} \tag{2.2}$$

$$f_{jk} \sim g_f^{(k)} \in \mathcal{G}_f^{(k)} \tag{2.3}$$

$$e_{ij} \sim \mathcal{N}\left(0, 1/\tau_{ij}\right), \tag{2.4}$$

where $\mathbf{Y} \in \mathbb{R}^{n \times p}$ is the data matrix, $\mathbf{L} \in \mathbb{R}^{n \times K}$ is a matrix of loadings, $\mathbf{F} \in \mathbb{R}^{p \times K}$ is a matrix of factors, $\mathbf{E} \in \mathbb{R}^{n \times p}$ is a matrix of mutually independent Gaussian errors, and $g_\ell^{(k)}$ and $g_f^{(k)}$ are column-wise priors to be estimated via empirical Bayes from among some families of priors $\mathcal{G}_\ell^{(k)}$ and $\mathcal{G}_f^{(k)}$. (I will also refer to columns $\boldsymbol{\ell}_k$ and $\mathbf{f}_k$ considered jointly as a "factor": in this sense, an EBMF fit has K factors. In each instance the meaning of "factor" should be clear from the context.)

One of the key advantages of EBMF is that it can directly model sparsity by way of the choice of prior families $\mathcal{G}_\ell^{(k)}$ and $\mathcal{G}_f^{(k)}$. For example, the family of point-normal priors

$$\mathcal{G}_{\text{pn}} := \left\{ g : g \sim \pi_0 \delta_0 + (1 - \pi_0) \mathcal{N}\left(0, \sigma^2\right),\ 0 \leq \pi_0 \leq 1, \sigma^2 > 0 \right\} \tag{2.5}$$

models exact sparsity via the point mass component δ_0 and also shrinks posterior means towards zero (since the priors are unimodal and symmetric about zero). In general, sparsity enhances interpretability: applied to scRNA-seq data, sparsity-inducing priors can yield factors in which a subset of genes (with, perhaps, a relatively small number of biological functions) is activated or de-activated in a subset of cells (for example, a single cell type).

While the philosophy behind EBMF is appealing, the original R implementation — which Wang and Stephens called **flashr** — was not designed to handle large, sparse datasets. Modern scRNA-seq datasets such as the larger "pulse-seq" dataset in Montoro et al. [2018] are very slow to fit and can exhaust 64 GB of memory. As a result, the application of EBMF to scRNA-seq data required a re-design of the software from the ground up. My new implementation, **flashier**, can handle much larger datasets and, in terms of run time, is very competitive with other model-based matrix factorization methods such as topic modeling (Pritchard et al. [2000]; Blei et al. [2003]; Carbonetto et al. [2021]) and GLM-PCA (Townes et al. [2019]).

In the "Software" section of this chapter, I detail the most important changes that were necessary to fit EBMF models to scRNA-seq data. Some changes targeted the efficiency of

the algorithm; others made the algorithm more flexible; still others extended the principle of modularity that is at the heart of the `flash` algorithm. Effectively, `flash` reduces the problem of fitting an EBMF model to a sequence of empirical Bayes normal means (EBNM) problems:

$$x_i \sim \mathcal{N}\left(0, s_i^2\right) \tag{2.6}$$

$$x_i \sim g \in \mathcal{G}. \tag{2.7}$$

In essence, one repeatedly alternates between, on the one hand, fixing $\mathbf{F}$ and solving an EBNM problem to update $\mathbf{L}$, and on the other, fixing $\mathbf{L}$ and solving an EBNM problem to update $\mathbf{F}$. Thus the `flash` algorithm encourages a modular approach: in principle, *any* function that solves the EBNM problem can simply be plugged into `flash` to fit the EBMF model. (A brief note on terminology is in order: I use "EBMF" to refer to the model and `flash` to refer to the fitting algorithm as described by Wang and Stephens. **flashr** and **flashier** are, respectively, older and newer implementations of `flash`.)

The next section, "Model and Methods," addresses some issues that arise from the fact that — unlike, say, topic modeling and GLM-PCA — EBMF is clearly misspecified for count data. I follow common practice in using the data transformation

$$x_{ij} = \log\left(c_i y_{ij} + \lambda\right), \tag{2.8}$$

where c_i is a cell-specific "size factor" and λ is a "pseudo-count" that's added to avoid taking logarithms of zero. I give some thought to the problems of choosing the size factors and pseudo-count but since these are in general quite difficult I focus on issues that are more particular to EBMF. Additionally, I introduce a novel use of EBMF, semi-nonnegative matrix factorization, which I argue can further improve interpretability of fits.

Finally, the "Results" section fits EBMF models, topic models, and GLM-PCA to two

modern scRNA-seq datasets, a set of peripheral blood mononuclear cells (PBMCs) published by Zheng et al. [2017] and a set of mouse epithelial cells published by Montoro et al. [2018]. The datasets include counts for, respectively, 3,205 and 7,913 cells, so both are challenging but small enough to make it practical to experiment with multiple fits. For both, there exist published cell types, which I use to visualize fits (inference for the PBMCs dataset was performed by Freytag et al. [2018]). I demonstrate in particular that EBMF excels at picking out rare cell types: CD34+ and dendritic cells in the PBMCs dataset; and ionocytes in the epithelial dataset.

Package **flashier** is available at `https://github.com/willwerscheid/flashier`. Figures and results from this chapter can be reproduced by following the instructions at `https://github.com/willwerscheid/flashier-chapter`.

2.2 Software

First, I detail changes that were incorporated into **flashier** to improve run time and memory usage. In Section 2.2.1, I discuss the importance of avoiding the creation of objects that are the same size as the original data matrix whenever possible. Section 2.2.2 describes a simple acceleration technique inspired by Ang and Gillis [2019] that can greatly improve the performance of the `flash` backfitting algorithm, and Section 2.2.3 describes a factor initialization routine that improves the performance of the greedy algorithm when the data is sparse. (A brief description of the greedy and backfitting algorithms is provided at the beginning of Section 2.2.2.) Next, Section 2.2.4 discusses the modularization of matrix operations, which allows **flashier** to take advantage of sparsity by working directly with sparse data structures but also extends the `flash` algorithm to tensor objects and low-rank matrix representations. In Section 2.2.5, I detail variance structures that were added to **flashier** (that is, possible assumptions about structure in the matrix of error variances $\mathbf{\Sigma}$), and Section 2.2.6 provides a brief discussion of various changes that I consider to be

important improvements over **flashr** but are not necessarily instrumental in the application of EBMF to scRNA-seq data. Finally, Section 2.2.7 benchmarks the **flashr** and **flashier** implementations using the GTEx dataset that was discussed in the original EBMF paper (Wang and Stephens [2021]) as well as the two scRNA-seq datasets discussed in the "Results" section of this chapter.

2.2.1 Elimination of Expensive Matrix Operations

The original **flashr** implementation stores several auxiliary matrices that are the same size as the full-data matrix $\mathbf{Y}$: a matrix of precision parameters $\boldsymbol{\tau}$ as well as both expected residuals and expected squared residuals. While this poses no major problems for the smaller data matrices in which the original paper is primarily interested, it creates an undue strain on memory resources for larger scRNA-seq datasets. Indeed, these auxiliary matrices are typically low-rank: the matrices of expected residuals and squared residuals are both rank-K (where K is the number of factors in the fitted `flash` object), while the matrix of precision parameters is usually rank-one (if, however, the variance structure includes a "noisy" full-rank matrix of known standard errors, then $\boldsymbol{\tau}$ will also be full-rank: see Section 2.2.5). Since K is typically much smaller than either the number of cells n or the number of genes p, it is in most cases more efficient to perform matrix multiplications using full-rank ($n \times K$ or $p \times K$) matrices than it is to form larger rank-deficient matrices.

An example will prove useful. As mentioned in the introduction to this chapter, the guts of the `flash` algorithm consist in iteratively solving multiple EBNM problems. For simplicity, assume a "constant" variance structure, so that all precision parameters τ_{ij} are identical to a single parameter τ, and assume that there are no missing entries in $\mathbf{Y}$. Then

an update to the kth loadings column $\boldsymbol{\ell}_k$ is performed by solving an EBNM problem with

$$s^2 = \frac{1}{\tau \mathbf{1}_p^T \mathbb{E}\mathbf{f}_k^2} \tag{2.9}$$

$$\mathbf{x} = \tau s^2 \mathbf{R}^{(k)} \mathbb{E}\mathbf{f}_k. \tag{2.10}$$

Here, $\mathbf{x} \in \mathbb{R}^n$ and $s > 0$ are inputs to the EBNM problem, $\mathbf{1}_p$ is an all-ones vector of length p, expectations are with respect to the variational approximations of the posteriors on $\mathbf{L}$ and $\mathbf{F}$, exponentiation ($\mathbb{E}\mathbf{f}_k^2$) is element-wise, and $\mathbf{R}^{(k)}$ is the $n \times p$ matrix of expected residuals excluding the kth factor:

$$\mathbf{R}^{(k)} = \mathbf{Y} - \mathbb{E}\mathbf{L}_{-k} \mathbb{E}\mathbf{F}_{-k}^T, \tag{2.11}$$

where $\mathbf{L}_{-k}$ is the $n \times (k-1)$ matrix of loadings that excludes the kth column $\boldsymbol{\ell}_k$ and $\mathbf{F}_{-k}$ is similar. (Equations 2.9-2.10 can be compared to Equations A.39-A.40 inWang and Stephens [2021].)

The original **flashr** implementation solves Equation 2.10 directly by forming the $n \times p$ matrix $\mathbf{R}^{(k)}$ and then multiplying it against the p-vector $\mathbb{E}\mathbf{f}_k$. However, when n and p are both large, then it is much more efficient to write (plugging in Equation 2.11):

$$\mathbf{x} = \tau s^2 \left(\mathbf{Y}\mathbb{E}\mathbf{f}_k - \mathbb{E}\mathbf{L}_{-k} \left(\mathbb{E}\mathbf{F}_{-k}^T \mathbb{E}\mathbf{f}_k \right) \right) \tag{2.12}$$

Notice that all matrix multiplications in Equation 2.12 yield vectors. In particular, no new $n \times p$ matrices are even temporarily created: the only object of size $n \times p$ is the original data matrix $\mathbf{Y}$.

As it turns out, no new matrices of size $n \times p$ *ever* need to be formed provided that the choice of variance structure is simple ("constant," "row-wise," or "column-wise" — for definitions, see Section 2.2.5). In such cases, the memory requirements for `flash` should not, at least in theory, be much greater than the memory required to store the original data

matrix. In practice, the savings in both time and memory are considerable when the data matrix is very large (on the order of 1 GB or larger when loaded into R). For examples, see the benchmarking results in Section 2.2.7.

2.2.2 Acceleration of Backfits via Extrapolation

Wang and Stephens [2021] distinguish between two `flash` algorithms. The "greedy" algorithm adds factors one at a time, adding the kth factor by, in essence, fixing the previous $k-1$ factors and fitting a rank-one EBMF model to the matrix of expected residuals. Once a factor has been added, it remains fixed while subsequent factors are added. In contrast, the "backfitting" algorithm takes a K-factor initialization and iteratively loops over all K factors, performing one update at a time until all K factors have converged. The approaches can be combined by using a greedy EBMF fit as an initialization to the backfitting algorithm.

While a greedy fit can be obtained very quickly, the backfitting algorithm can offer a substantial improvement both with respect to fit (the EBMF objective attained) and with respect to the sparsity and interpretability of the resulting factors. However, the objective surface for the rank-K EBMF model is much more challenging than for the rank-one model, and increasingly so as K grows (likely due to the complicated interdependencies among factor estimates). As a result, the backfitting algorithm is almost always far slower than the greedy algorithm.

To speed up backfits, **flashier** uses an acceleration scheme loosely based on the "extrapolation" technique outlined in Ang and Gillis [2019]. The idea is simple. Denote the set of optimization parameters as θ and let $\theta^{(m)}$ denote the parameter values after m iterations. (Here, an iteration is defined as a single update to each of the K factors in the fitted `flash` object, as well as an update to the shared precision parameter $\boldsymbol{\tau}$.) Let g be the usual `flash`

update function (without acceleration), so that

$$\theta^{(m+1)} = g(\theta^{(m)}). \tag{2.13}$$

The updates will become smaller in magnitude as the algorithm nears convergence, but it's reasonable to expect that they will be in roughly the same direction from one iteration to the next. If so, then it should be possible to accelerate an update by first "nudging" the parameters in the direction of the previous update by setting

$$\tilde{\theta}^{(m)} = \theta^{(m)} + \beta_m(\theta^{(m)} - \theta^{(m-1)}),$$

where β_m is a "momentum" parameter, and then performing the usual update on $\tilde{\theta}^{(m)}$:

$$\theta^{(m+1)} = g(\tilde{\theta}^{(m)}). \tag{2.14}$$

In this scheme, β_m can be adjusted depending on the success of the update. If the objective increases, then I attempt to hasten convergence by setting $\beta_{m+1} = \alpha\beta_m$, where $\alpha > 1$ is a fixed "acceleration" parameter. If it decreases, then I discard $\tilde{\theta}^{(m)}$ and instead set

$$\theta^{(m+1)} = g(\theta^{(m)}). \tag{2.15}$$

Further, I set $\beta_{m+1} = \delta\beta_m$ so that the next attempt at extrapolation is more cautious ($\delta < 1$ is a fixed "deceleration" parameter). Since the usual `flash` update is guaranteed to increase the objective function at each step, falling back to a standard update (Equation 2.15) ensures that the objective function increases monotonically even with acceleration.

Despite the simplicity of the scheme, it usually leads to marked improvements in run time as well as in overall fit. Indeed, the scheme often helps the backfitting algorithm to accelerate through very flat portions of the objective surface, where without acceleration it

can slow to the point of having seemingly converged. See Section 2.2.7 for examples.

2.2.3 Homespun Initialization Routine

By default, the original **flashr** implementation uses package **softImpute** (Mazumder et al. [2010]) to find a rank-one fit to the matrix of expected residuals. For complete data, there are faster alternatives, including `svd` and, for large, sparse matrices, approximate methods such as the one implemented by R package **irlba** (Baglama et al. [2019]). In general, however, these methods cannot handle missing data.

Still, the problem of finding a good factor initialization does not require the full apparatus offered by a package like **softImpute**. Indeed, such packages are typically designed to find a rank-K approximation to possibly incomplete data for arbitrary K. The factor initialization problem, however, is much simpler, as it only requires a rank-one approximation.

The task of finding the first singular vectors $\mathbf{u}$ and $\mathbf{v}$ can be formulated as an optimization problem:

$$\arg\min_{\mathbf{u},d,\mathbf{v}} \|\mathbf{Y} - \mathbf{u}d\mathbf{v}^T\|_2^2 \text{ s.t. } \|u\|_2 = 1,\ \|v\|_2 = 1, \tag{2.16}$$

or, absorbing the scalar d into $\mathbf{u}$ and $\mathbf{v}$ and removing the constraints (at the expense of making the solution non-unique):

$$\arg\min_{\mathbf{u},\mathbf{v}} \sum_{i,j} \left(y_{ij} - u_i v_j\right)^2. \tag{2.17}$$

A solution can be obtained using the well-known "power method," which performs alternating updates of $\mathbf{u}$ and $\mathbf{v}$ until convergence (Golub and Van Loan [1996]): see Table 2.1 for details.

The power method can easily be adapted to find a rank-one approximation for a matrix with missing data. I update the notation to match the EBMF notation I use elsewhere, so that $\mathbf{R}$ is a matrix of expected residuals and $\boldsymbol{\ell}$ and $\mathbf{f}$ provide the initialization for a new

Algorithm 1: Power method for finding rank-one approximation (complete data)

Input: A complete data matrix $\mathbf{Y} \in \mathbb{R}^{n \times p}$; a tolerance parameter $\epsilon > 0$
initialize $\mathbf{u} = \mathbf{0}_n$, $\mathbf{v} \sim \mathcal{N}(0, \mathbf{I}_p)$, $\delta = \mathbf{Inf}$;
while $\delta > \epsilon$ **do**
  set $\mathbf{u}_{\text{old}} = \mathbf{u}$, $\mathbf{v}_{\text{old}} = \mathbf{v}$;
  set $\mathbf{u} = \frac{\mathbf{Y}\mathbf{v}}{\mathbf{v}^T\mathbf{v}}$;
  set $\mathbf{v} = \frac{\mathbf{Y}^T\mathbf{u}}{\mathbf{u}^T\mathbf{u}}$;
  set $\delta = \max\left\{|\frac{\mathbf{u}^T\mathbf{u}_{\text{old}}}{\mathbf{u}^T\mathbf{u}} - 1|, |\frac{\mathbf{v}^T\mathbf{v}_{\text{old}}}{\mathbf{v}^T\mathbf{v}} - 1|\right\}$;
return $\mathbf{u}$, $\mathbf{v}$

Table 2.1: Algorithm for finding a rank-one approximation to a complete data matrix. In the updates to $\mathbf{u}$ and $\mathbf{v}$, division is element-wise.

factor. For incomplete data, then, an analogous optimization problem is:

$$\arg\min_{\boldsymbol{\ell},\mathbf{f}} \sum_{(i,j)\notin\mathcal{M}} \left(r_{ij} - \ell_i f_j\right)^2, \tag{2.18}$$

where $\mathcal{M}$ is the set of pairs (i, j) such that y_{ij} (and thus r_{ij}) is missing. Let $\tilde{\mathbf{R}}$ be the matrix that fills in missing data with zeros:

$$\tilde{r}_{ij} = \begin{cases} r_{ij}, & (i,j) \notin \mathcal{M} \\ 0, & (i,j) \in \mathcal{M} \end{cases} \tag{2.19}$$

and let $\mathbf{Z}$ be a binary matrix that indicates whether an entry is non-missing:

$$z_{ij} = \begin{cases} 1, & (i,j) \notin \mathcal{M} \\ 0, & (i,j) \in \mathcal{M} \end{cases} \tag{2.20}$$

Algorithm 2: Power method for finding rank-one approximation (incomplete data)

Input: A (possibly incomplete) data matrix $\mathbf{R} \in \mathbb{R}^{n \times p}$; a tolerance parameter $\epsilon > 0$
initialize $\tilde{\mathbf{R}} = \mathbf{R}$, $\mathbf{Z} = \mathbf{1}_{n \times p}$;
for *i = 1, . . . , n, j = 1, . . . , p* **do**
  if r_{ij} *is missing* **then**
    set $\tilde{r}_{ij} = 0$;
    set $z_{ij} = 0$;
initialize $\boldsymbol{\ell} = \mathbf{0}_n$, $\mathbf{f} \sim \mathcal{N}(0, \mathbf{I}_p)$, $\delta = \mathbf{Inf}$;
while $\delta > \epsilon$ **do**
  set $\boldsymbol{\ell}_{\text{old}} = \boldsymbol{\ell}$, $\mathbf{f}_{\text{old}} = \mathbf{f}$*;*
  set $\boldsymbol{\ell} = \frac{\tilde{\mathbf{R}}\mathbf{f}}{\mathbf{Z}\mathbf{f}^2}$*;*
  set $\mathbf{f} = \frac{\tilde{\mathbf{R}}^T\boldsymbol{\ell}}{\mathbf{Z}\boldsymbol{\ell}^2}$*;*
  set $\delta = \max\left\{|\frac{\boldsymbol{\ell}^T\boldsymbol{\ell}_{\text{old}}}{\boldsymbol{\ell}^T\boldsymbol{\ell}} - 1|, |\frac{\mathbf{f}^T\mathbf{f}_{\text{old}}}{\mathbf{f}^T\mathbf{f}} - 1|\right\}$*;*
return $\boldsymbol{\ell}$, $\mathbf{f}$

Table 2.2: Algorithm for finding a rank-one approximation to an incomplete data matrix. In the updates to $\boldsymbol{\ell}$ and $\mathbf{f}$, division and exponentiation are element-wise.

It is easy to verify that, given $\boldsymbol{\ell}$, the solution is

$$\mathbf{f} = \frac{\tilde{\mathbf{R}}^T\boldsymbol{\ell}}{\mathbf{Z}^T\boldsymbol{\ell}^2}, \tag{2.21}$$

where division and exponentiation are both element-wise. Similarly, given $\mathbf{f}$, the solution is

$$\boldsymbol{\ell} = \frac{\tilde{\mathbf{R}}\mathbf{f}}{\mathbf{Z}\mathbf{f}^2}. \tag{2.22}$$

These observations lead to the power method algorithm detailed in Table 2.2.

Note that $\mathbf{R}$ does not need to be explicitly formed. Indeed, **flashier** stores $\tilde{\mathbf{Y}}$ and $\mathbf{Z}$

(defined similarly to Equations 2.19-2.20) rather than $\mathbf{Y}$ and calculates, for example:

$$\tilde{\mathbf{R}}\mathbf{f} = \left(\tilde{\mathbf{Y}} - \mathbf{Z} \odot \mathbb{E}\mathbf{L}_{\text{old}}\mathbb{E}\mathbf{F}_{\text{old}}^T\right)\mathbf{f} \tag{2.23}$$

$$= \tilde{\mathbf{Y}}\mathbf{f} - \left(\mathbf{Z}\left(\mathbb{E}\mathbf{F}_{\text{old}} \odot_{\text{b}} \mathbf{f}\right) \odot \mathbb{E}\mathbf{L}_{\text{old}}\right)\mathbf{1}_K, \tag{2.24}$$

where $\mathbf{L}_{\text{old}} \in \mathbb{R}^{n\times K}$ and $\mathbf{F}_{\text{old}} \in \mathbb{R}^{p\times K}$ are previously added factors and loadings, $\odot$ denotes element-wise multiplication, and $\odot_{\text{b}}$ denotes element-wise multiplication with broadcasting:

$$(\mathbf{A} \odot_{\text{b}} \mathbf{b})_{ij} = a_{ij}b_j. \tag{2.25}$$

As it turns out, Algorithm 2 is faster than **softImpute** (the default **flashr** method) for complete data but somewhat slower for incomplete data. On the other hand, it is probably slightly slower than **irlba** for complete data. (See Section 2.2.7 for benchmarking results.) For sparse datasets, however, the algorithm can be much faster than both **softImpute** and **irlba**, since it only requires the sparse matrix $\mathbf{Y}$ (or the equally sparse matrices $\tilde{\mathbf{Y}}$ and $\mathbf{Z}$) and the dense but relatively small matrices $\mathbb{E}\mathbf{L}$ and $\mathbb{E}\mathbf{F}$, whereas sparsity is destroyed when the matrix of expected residuals $\mathbf{R}$ is formed as input to an external package.

In sum, the true benefit of implementing Algorithm 2 is only seen when it is possible to take advantage of sparsity via sparse data structures such as those implemented by R package **Matrix** (Bates and Maechler [2021]). The ability to handle such structures required some very fundamental changes to **flashr**, which are detailed in the following section.

2.2.4 Modularization of Matrix Operations

As discussed in the introduction to this chapter, modularity is an important principle in the original **flashr** design. Any function that solves the EBNM problem can simply be "plugged in" to **flashr** — or, more precisely, *any* function can be plugged in as long as inputs and outputs are correctly formatted. In effect, this design decision was the impetus

for the R package **ebnm**, which collects different functions for solving the EBNM problem (some implemented by external packages, and some original to **ebnm**) and provides a unified interface for use by **flashr** (or by any other package for which the EBNM problem appears as a subroutine). See Chapter 1 for a fuller discussion.

The **flashier** implementation goes a few steps further by modularizing matrix operations. For example, all matrix-vector products are calculated via a call to the internal function `nmode.prod.vec`. The terminology is borrowed from tensor algebra, in which the "n-mode product" is a generalization of matrix multiplication. If A is a tensor with N dimensions or "modes", then the n-mode product of A with a vector v of length m is defined such that

$$[Av]_{i_1,\cdots,i_{n-1},i_{n+1},\cdots,i_N} = \sum_{j=1}^{m} A_{i_1,\cdots,i_{n-1},j,i_{n+1},\cdots,i_N} v_j. \tag{2.26}$$

Function `nmode.prod.vec` takes arguments `X` (the matrix or tensor), `v` (the vector), and `n` (the mode), and returns an object (vector, matrix, or tensor) of dimension $N - 1$. To date, it has been implemented for matrices, sparse matrices as implemented by R package **Matrix** (Bates and Maechler [2021]), three-dimensional tensors, and low-rank matrices (lists with components `u`, `d`, and `v`, like the objects returned by function `svd`).

The most important consequence for scRNA-seq data is that **flashier** can take advantage of sparsity. Since many scRNA-seq datasets are on the order of 90% sparse (or more), supplying a sparse `Matrix` object rather than a dense `matrix` can lead to a huge reduction in run time and, usually more importantly, memory usage. See Section 2.2.7 for examples.

In addition to this more immediate boon, the modularization of matrix operations opens up promising avenues of research in methods (for example, the use of low-rank approximations rather than full-data matrices) and applications (tensor data). Further, since there are only a handful of matrix operations that need to be implemented for any given data structure, **flashier** could fairly easily be extended to sparse matrices as implemented by packages other than **Matrix**, sparse tensors, or higher-dimensional tensors.

2.2.5 Generalized Variance Structures

Recall that the EBMF model is

$$\mathbf{Y} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{2.27}$$

$$e_{ij} \sim \mathcal{N}\left(0, 1/\tau_{ij}\right), \tag{2.28}$$

where the "errors" e_{ij} are mutually independent (I omit the priors on $\mathbf{L}$ and $\mathbf{F}$). The precision parameters τ_{ij}^2 can either be fixed by the user or estimated via maximum likelihood. When estimated, it's necessary to assume some structure in the matrix of variance parameters $\boldsymbol{\tau}$ in order to avoid over-parametrization. In total, four variance structures are available in **flashr**:

- Constant: A single precision parameter τ is to be estimated. It is shared among all entries (that is, $\tau_{ij} = \tau$).
- Row-wise (`"by_row"`): There are n precision parameters τ_i to be estimated, each of which is shared across a given row ($\tau_{ij} = \tau_i$).
- Column-wise (`"by_column"`): The column-wise analog of the row-wise variance structure ($\tau_{ij} = \tau_j$).
- Fixed (`"zero"`): The precision parameters τ_{ij} are known and supplied by the user.

The **flashr** terminology proceeds from the idea that one can write

$$\mathbf{Y} = \mathbf{L}\mathbf{F}^T + \mathbf{E}^{(1)} + \mathbf{E}^{(2)} \tag{2.29}$$

$$e_{ij}^{(1)} \sim \mathcal{N}\left(0, s_{ij}^2\right) \tag{2.30}$$

$$e_{ij}^{(2)} \sim \mathcal{N}\left(0, 1/\tau_{ij}\right), \tag{2.31}$$

where the s_{ij}^2s are all supplied by the user and the τ_{ij}s are all estimated. The terms `"constant"`, `"by_row"`, `"by_column"`, and `"zero"` all describe the structure of $\boldsymbol{\tau}$ (only in the last case, however, is $\mathbf{S}$ allowed to be nonzero).

A number of other choices are provided by **flashier**:

- "Kronecker": $\boldsymbol{\tau}$ is an arbitrary rank-one matrix to be estimated. Any such matrix can be written as the outer (Kronecker) product of an n-vector $\boldsymbol{\tau}_r$ and a p-vector $\boldsymbol{\tau}_c$, so a total of $n + p - 1$ parameters must be estimated. (There is redundancy in the $n + p$ parametrization because one can scale $\boldsymbol{\tau}_r$ by any constant α while also scaling $\boldsymbol{\tau}_c$ by $1/\alpha$.) To form estimates, **flashier** uses an alternating scheme. With $\boldsymbol{\tau}_r$ fixed, there is a simple closed-form expression for the maximum likelihood estimate of $\boldsymbol{\tau}_c$:

$$\frac{\mathbf{1}_p}{\hat{\boldsymbol{\tau}}_c} = \frac{1}{n}\boldsymbol{\tau}_r^T \mathbb{E}\left(\mathbf{R}^2\right), \tag{2.32}$$

  where $\mathbb{E}\left(\mathbf{R}^2\right)$ is the matrix of expected squared residuals and division is element-wise. A similar expression exists for $\hat{\boldsymbol{\tau}}_r$, so that one can estimate $\boldsymbol{\tau}$ by alternately updating $\hat{\boldsymbol{\tau}}_r$ and $\hat{\boldsymbol{\tau}}_c$ until convergence.

- "Noisy": A non-zero $\mathbf{S}$ is supplied by the user, but additional variance in $\boldsymbol{\tau}$ is to be estimated as well. Any of the four assumptions about $\boldsymbol{\tau}$ listed above are possible, so that the "noisy" case is in fact a class that includes four possible variance structures: "noisy/constant", "noisy/row-wise", "noisy/column-wise", and "noisy/Kronecker."

This flexibility can offer both more sensible models and better fits to the data. For example, when fitting scRNA-seq datasets using a gene-wise variance structure, I've observed that some of the estimates τ_j can become very large, which can cause numerical difficulties for **flashier**. Since the corresponding columns $\mathbf{y}_j$ tend to be very sparse, I suspect that `flash` is overfitting to a few non-zero entries. This problem can usually be circumvented by

using a noisy/column-wise variance structure with, say,

$$s_{ij}^2 = \frac{1}{n} \tag{2.33}$$

for all i, j. Effectively, this puts the total residual variance for entry y_{ij} at

$$s_{ij}^2 + 1/\tau_{ij} = \max\left\{\frac{1}{n}, \frac{1}{\hat{\tau}_j}\right\}, \tag{2.34}$$

where $\hat{\tau}_j$ is the precision estimate that would be obtained using a simple column-wise variance structure (that is, with $s_{ij}^2 = 0$). A noisy variance structure can thus provide a reasonable floor for residual variance estimates. (The precise value of s_{ij}^2 in Equation 2.33 is not critical, so long as it's smaller — but not much smaller — than one ought to expect the residual variance for any entry to be.)

2.2.6 Miscellaneous New Features

I include in this section brief descriptions of additional **flashier** features that I consider to be major improvements over **flashr** but that aren't immediately relevant to the application of EBMF to scRNA-seq data. Some are primarily enhancements to user experience, while others add important functionality that's most useful in other applications:

- Fixed factors. Package **flashier** allows the user to fix any subset of entries of either $\mathbf{L}$ or $\mathbf{F}$ (or, more precisely, $\mathbb{E}\mathbf{L}$ and $\mathbb{E}\left(\mathbf{L}^2\right)$ or $\mathbb{E}\mathbf{F}$ and $\mathbb{E}\left(\mathbf{F}^2\right)$). Suppose, for example, that one knows which features contribute to a given factor but that the specific factor loadings are unknown. (For a concrete example, each factor might correspond to a set of genes that participate in a particular biological pathway.) Then one can fix the sparsity pattern in $\mathbf{F}$ and estimate the non-zero factor loadings. For a detailed example of an application that uses fixed factors, see Chapter 3, and especially Section 3.3.4. (This functionality was eventually added to **flashr** as well.)

- Posterior samplers. Since only the posterior first and second moments $\mathbb{E}\mathbf{L}$, $\mathbb{E}\mathbf{F}$, $\mathbb{E}\left(\mathbf{L}^2\right)$, and $\mathbb{E}\left(\mathbf{F}^2\right)$ are needed to perform `flash` updates, the original **flashr** implementation only returned those moments. Depending on the function used to solve the EBNM problem, however, much more information about the (variational) posteriors can be made available. In particular, **flashier** returns a function that samples from the posterior on $\mathbf{L}$ and $\mathbf{F}$, provided that the EBNM functions used also return posterior samplers. Note that all of the functions included in package **ebnm** can return samplers: for an example of how they might be used, see Section 1.8. (This functionality was also added to **flashr**, but **flashr** inefficiently returns samples from the posterior on $\mathbf{L}\mathbf{F}^T$ rather than separately sampling $\mathbf{L}$ and $\mathbf{F}$.)

- Fully customizable output. Function `flash.set.verbose` allows the user to print the output of an arbitrary function after each iteration of a greedy fit or backfit. Thus, for example, the value of a particular loading can be monitored for exploratory or debugging purposes. Further, whereas the output of **flashr** only includes the value of the objective function and is printed in a format that's difficult to parse, **flashier** includes the option to print output in a tab-delimited format. It's thus a simple matter to produce visualizations such as the plots of the change in objective over time shown in Figure 2.1 (the time of each backfitting iteration's completion was recorded via a call to `Sys.time`).

- Pipeable interface. The pipeability of **flashier** allows the user to build complex yet highly readable `flash` fits from discrete operations such as `flash.add.greedy`, `flash.backfit`, `flash.fix.factors`, and `flash.remove.factors`. For example, the call to the main `flash` function

  ```
  fl <- flash(dat, greedy.Kmax = 10L, backfit = TRUE)
  ```

  is equivalent to

```
fl <- flash.init(dat) %>%
  flash.add.greedy(Kmax = 10L) %>%
  flash.backfit() %>%
  flash.nullcheck(remove = TRUE).
```

While the main `flash` function can suffice for basic EBMF fits, this expanded interface offers a much wider range of customizability (whether to perform extrapolation, how to initialize new factors, whether to "warmstart" backfits, which stopping criterion to use, etc.) as well as making clear the order of operations.

2.2.7 Benchmarks

I use three datasets:

- **"Urbut-GTEx":** The GTEx dataset used in the original EBMF paper (Wang and Stephens [2021]). The dataset is derived from data made available by the Genotype Tissue Expression (GTEx) project (Lonsdale et al. [2013]), which provides z-scores for the effects of SNPs on gene expression across 44 human tissues. To reduce the data to a more manageable size, Urbut et al. [2019] choose the "top" eQTL for each gene — that is, the SNP associated with the largest z-score over all 44 tissues. This selection process yields a $16,069 \times 44$ matrix of z-scores, with rows corresponding to SNP-gene pairs and columns corresponding to tissues.

- **"PBMC-3k":** One of the smaller scRNA-seq datasets published by Zheng et al. [2017]. It consists of $3,205$ peripheral blood mononuclear cells (PBMCs) collected from a single donor, which were profiled using Unique Molecular Identifiers (UMIs) to yield reads for $24,565$ genes. Cell types were subsequently inferred by Freytag et al. [2018]. The dataset takes up 602.7 MB of memory when loaded into R as a dense matrix, but only 4.9% of all counts are nonzero. When loaded as a `Matrix` object, the dataset takes

up 46.3 MB. I applied a `log1p` transform to the data, but no other pre-processing was done for benchmarking purposes.

- **"Montoro-droplet":** The smaller of the two scRNA-seq datasets published by Montoro et al. [2018]. It includes UMI counts for $7,193$ mouse epithelial cells across $18,388$ genes. Cell types were inferred by the authors. The dataset is less sparse than the PBMC-3k dataset, with 9.3% of entries greater than zero. When loaded into R as a dense matrix, the dataset takes up 838.4 MB of memory; as a sparse matrix, it takes up 142.4 MB. Again, the only pre-processing step was to apply a `log1p` transform before fitting.

For each dataset, I fit `flash` using both greedy and backfitting algorithms. The greedy fits add 10 factors without backfitting. The backfits are initialized using a rank-5 approximate SVD obtained using R package **irlba** (Baglama et al. [2019]) and are capped at 100 iterations. I perform each fit using four methods, each of which builds upon the last by incorporating one of the improvements discussed in the previous sections:

- **"Flashr"**: Uses the original **flashr** implementation. All other methods use **flashier**.
- **"Efficient ops"**: Takes advantage of the more efficient matrix operations that are at the core of the **flashier** implementation (see Section 2.2.1). For stricter comparability with **flashr**, it uses **softImpute** to initialize new factors and does not use extrapolation to accelerate backfits.
- **"Fast init"** (greedy only): Uses the homespun initialization function described in Section 2.2.3 rather than **softImpute**.
- **"Extrapolate"** (backfit only): Accelerates backfits using the simple extrapolation technique described in Section 2.2.2.

- **"Sparse"** (sparse datasets only): converts the dataset to a sparse `Matrix` object before running **flashier** (with the improved initialization scheme and with extrapolation). See Section 2.2.4.

All tests are performed on a 2018 MacBook Pro with a 2.6 GHz Intel Core i7 processor and 16 GB of DDR4-2400 RAM. Timing is accomplished using profiling function `Rprof` for greedy fits and outputting the current time via a custom column for backfits (see Section 2.2.6; since **flashr** does not allow for custom output, I simply assign an equal proportion of the total fitting time to each **flashr** iteration).

Results are shown in Figure 2.1. For the scRNA-seq datasets (PBMC-3k and Montoro-droplet), the more efficient matrix operations speed up backfits by about an order of magnitude, while extrapolation and sparse data structures speed them up by another, so that **flashier** backfits are over 100 times faster than **flashr**. Gains for the greedy algorithm are similar, with most of the improvement resulting from the use of sparse `Matrix` objects.

Next, I benchmark initialization functions. I consider the default **flashier** initialization function described in Section 2.2.3, function `softImpute` from package **softImpute**, and function `irlba` from package **irlba**. As discussed in Section 2.2.3, the latter two functions cannot be used to full advantage in all settings: neither can exploit sparsity in the data matrix $\mathbf{Y}$, and `irlba` is unable to handle missing data. I summarize these properties in Table 2.3.

I use **flashier** to fit ten-factor `flash` objects to each of the two scRNA-seq datasets described above (PBMC-3k and Montoro-droplet) using the greedy algorithm. I first fit to the complete data matrix using all three initialization functions as well as converting the data to a sparse `Matrix` and using the default **flashier** function (labeled "sparse" in the figure). I then delete 20% of entries at random and re-fit using the "default," "softImpute," and "sparse" methods (since `irlba` cannot handle missing data). For each factor added, I use profiling function `Rprof` to parse the time required to initialize and then refine the factor.

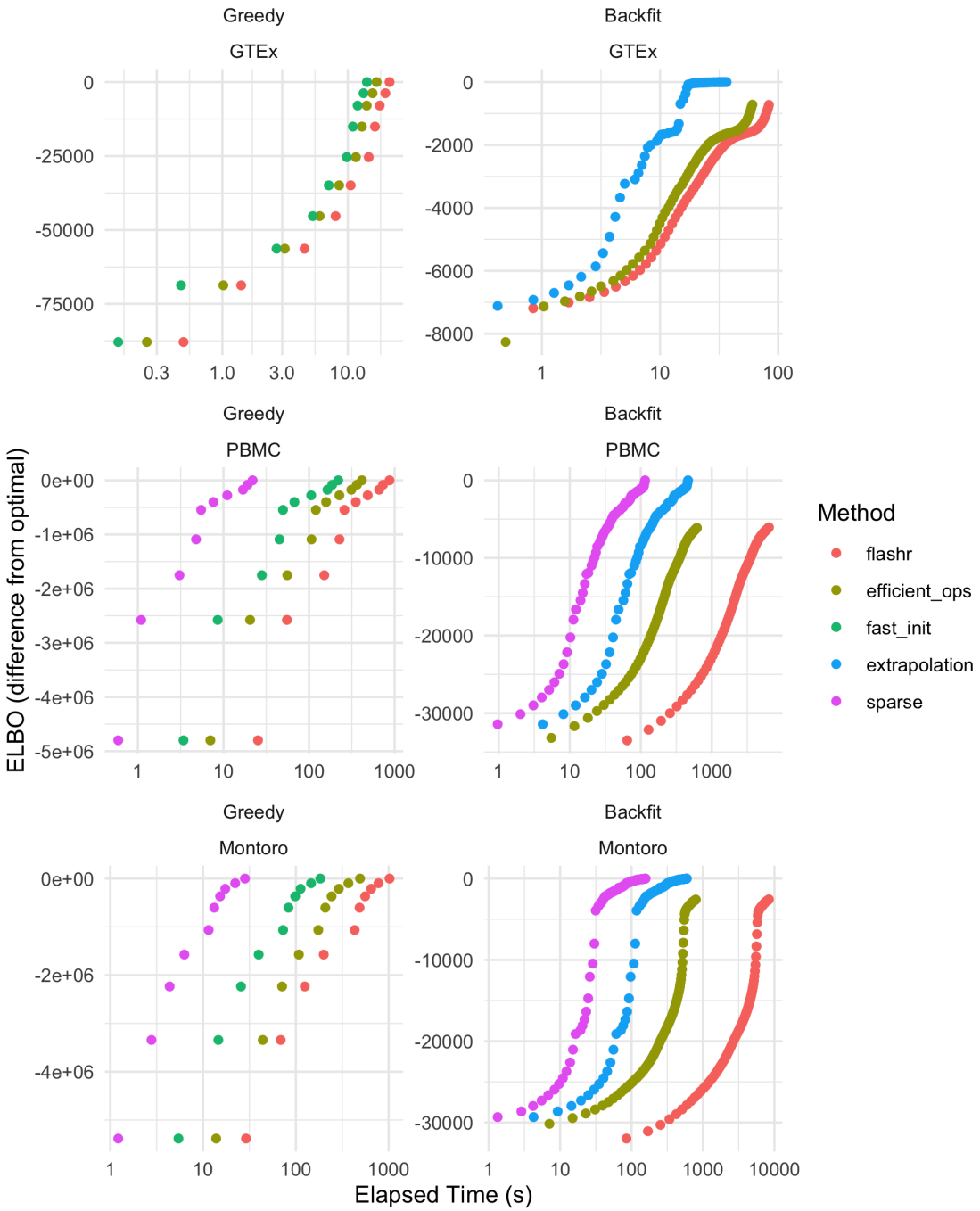


Figure 2.1: Plots of the EBMF objective over time for three datasets using greedy and backfitting algorithms and five different methods (see text for a description of the datasets and methods). Each greedy fit adds ten factors (each point corresponds to a factor). The backfits are five-factor `flash` objects initialized using **irlba**: each point corresponds to a backfitting iteration (an update to each of the five factors and one or more updates to the precision parameter).

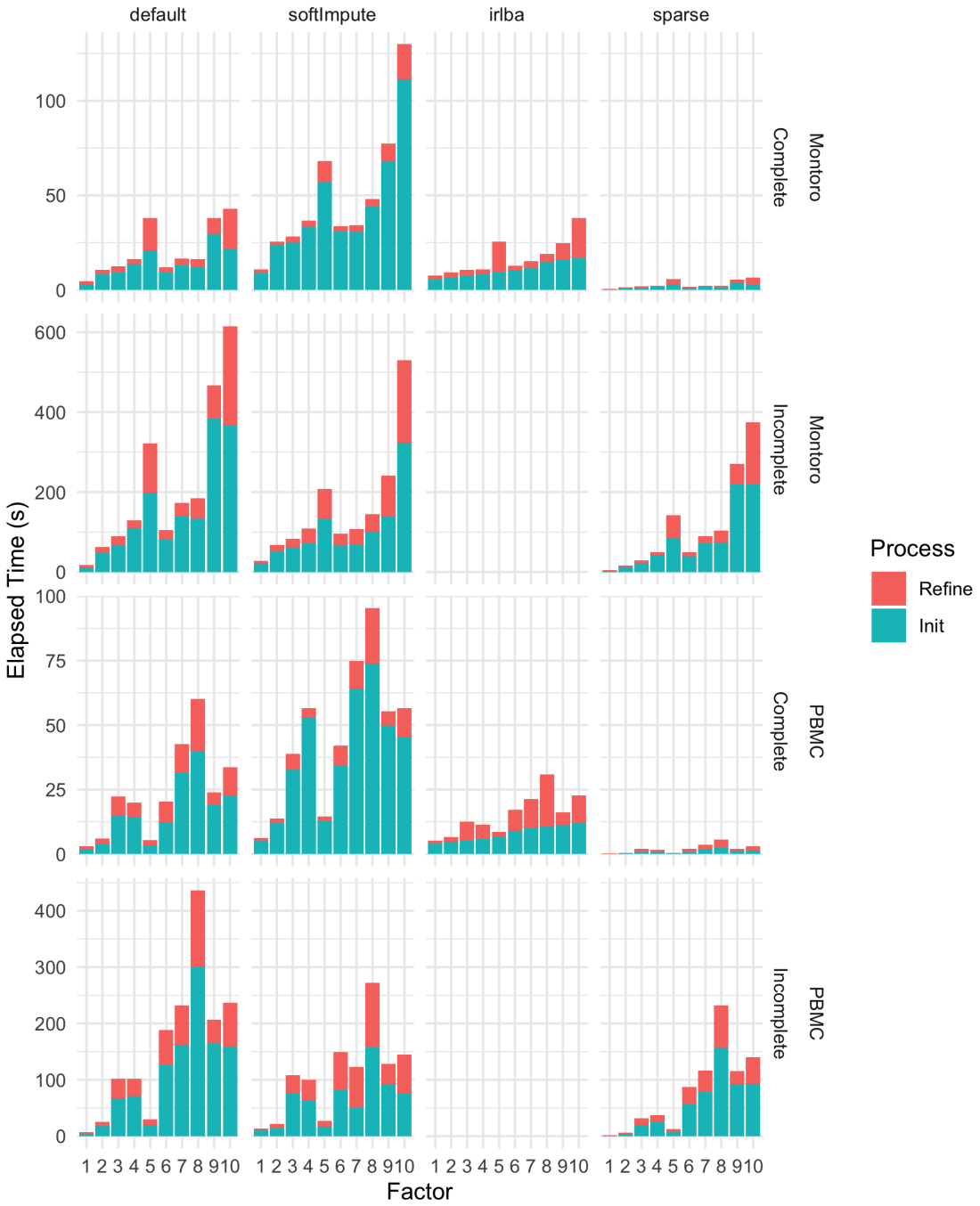


Figure 2.2: Time taken to fit ten greedy factors to the PBMC-3k and Montoro-droplet datasets using different initialization functions (see text for details). The "incomplete" datasets were generated by deleting 20% of entries at random. Each bar corresponds to a single factor, with the elapsed time partitioned between the time required to initialize the factor and the time required to refine the factor using alternating `flash` updates.

	Dense `matrix`		Sparse `Matrix`	
Function	Complete	Incomplete	Complete	Incomplete
flashier default	✓	✓	✓	✓
`softImpute` (**flashr** default)	✓	✓	∗	∗
`irlba`	✓		∗	

Table 2.3: Properties of initialization functions used for benchmarking. The functions are implemented as `init.fn.default`, `init.fn.softImpute`, and `init.fn.irlba` in **flashier**. An asterisk (∗) indicates that the function can accept the corresponding type of sparse object, but does not take advantage of sparsity since the matrix of expected residuals used to initialize factors will in general be dense for all but the first factor. See Section 2.2.3.

Results are shown in Figure 2.2. While the default **flashier** initialization function is probably a bit slower than `irlba` for data that is both dense and complete, and somewhat slower than `softImpute` for data that is dense and incomplete, it's clear that there are huge gains when the data is both sparse and complete (as is the case for the majority of scRNA-seq datasets). Further, it's at least competitive with (and probably a bit faster than) `softImpute` when the data is sparse and incomplete.

2.3 Model and Methods

The following sections delve into issues that arise from the fact that the EBMF model is misspecified for count data. Section 2.3.1 describes a handful of models that, in contrast to EBMF, directly model matrix factorizations for count data. I focus on topic models and GLM-PCA since they're fairly widely used to analyze scRNA-seq data. Next, Section 2.3.2 discusses the family of log transformations that is most commonly used when Gaussian methods are applied to count data, a family of transformations that I also adopt. The full EBMF model for count data is given at the end of the section (Equations 2.57-2.62). Section 2.3.3 takes a brief detour to discuss semi-nonnegative matrix factorization, which I argue can improve the interpretability of EBMF results but also showcases the flexibility of the EBMF approach in general. The last two sections return to the question of model misspecification

in order to detail some consequences for EBMF: Section 2.3.4 demonstrates that EBMF can no longer reliably select the correct number of factors, while 2.3.5 stresses the difficulty of using objective metrics to compare data transformations. In particular, I demonstrate that the more obvious (and at least one less obvious) means for choosing a "pseudo-count" fail.

2.3.1 Matrix Factorization Models for Count Data

For count data such as scRNA-seq datasets, the EBMF model is clearly misspecified, since it models continuous rather than discrete data. This shortcoming appears avoidable, since there exist matrix factorization methods that directly model count data. Poisson NMF, for example, models a matrix of counts as a low-rank matrix of "true" means with Poisson noise added to the observations:

$$y_{ij} \sim \text{Poisson}(\lambda_{ij}) \tag{2.35}$$

$$\lambda_{ij} = \sum_{k=1}^{K} \ell_{ik} f_{jk} \tag{2.36}$$

$$\mathbf{L} \geq 0, \ \mathbf{F} \geq 0, \tag{2.37}$$

where the inequality constraints are element-wise. As Carbonetto et al. [2021] show, an equivalent model (in the sense that log likelihoods are identical up to a constant) is the topic model

$$y_{i1}, \ldots, y_{im} \ \Big| \ \sum_j y_{ij} = n_i \sim \text{Multinomial}\left(n_i; \pi_{i1}, \ldots, \pi_{im}\right) \tag{2.38}$$

$$\mathbf{\Pi} = \tilde{\mathbf{L}}\tilde{\mathbf{F}}^T, \tag{2.39}$$

where there exists a simple one-to-one mapping between, on the one hand, $\mathbf{L}$ and $\mathbf{F}$, and on the other, $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{F}}$: see Carbonetto et al.'s Definition 1 for details. (The obvious constraints

on the matrix of probabilities $\mathbf{\Pi}$ apply: that is, $\mathbf{\Pi} \geq 0$ and $\sum_j \pi_{ij} = 1$ for all i.) Townes et al. [2019] propose a similar model, but assume low-rank structure in the log-transformed means rather than in the means themselves (or, equivalently, the multinomial proportions):

$$y_{ij} \sim \text{Poisson}\left(c_i \exp\left(\eta_{ij}\right)\right) \tag{2.40}$$

$$\eta_{ij} = \sum_{k=1}^{K} \ell_{ik} f_{jk}. \tag{2.41}$$

Note that the "size factors" c_i can be absorbed into the low-rank structure, so that the model can also be written

$$y_{ij} \sim \text{Poisson}\left(\exp\left(\lambda_{ij}\right)\right) \tag{2.42}$$

$$\mathbf{\Lambda} = \mathbf{L}\mathbf{F}^T, \tag{2.43}$$

with $\mathbf{L} \in \mathbb{R}^{n \times (K+1)}$, $\mathbf{F} \in \mathbb{R}^{p \times (K+1)}$, $\boldsymbol{\ell}_1 = (c_1, \ldots, c_n)$, and $\mathbf{f}_1 = \mathbf{1}_p$.

While the topic model assumes that effects (factors) are additive, the Townes et al. model — which the authors call GLM-PCA — assumes multiplicative effects. Both models are plausible: whether one assumption is better than the other is an open question. Other authors have modeled noise as "zero-inflated":

$$y_{ij} \sim \pi_0 \delta_0 + (1 - \pi_0)\,\text{Poisson}\left(\lambda_{ij}\right), \tag{2.44}$$

where δ_0 is a point mass at zero and $0 \leq \pi_0 \leq 1$. However, Sarkar and Stephens [2021] have convincingly argued from both biological principle and empirical examples that Poisson noise (Equation 2.35) is in fact the correct assumption for scRNA-seq data — at least, for data that has been generated using unique molecular identifiers (UMIs), as is the case for the large majority of recent high-quality datasets. (Of course, there is typically structure among cells, so that the distribution of counts for a given gene is marginally a mixture of Poissons

rather than Poisson.) Since I agree that zero inflation is an unnecessary assumption, I omit discussion of methods that incorporate it (but for one of the better-known examples, see Risso et al. [2018]'s ZINB-WaVE).

Since scRNA-seq counts are affected by a large number of complex variables, including cell type, cell cycle phase at the moment of sequencing, and technical noise, whether one ought to model the "true" means (transformed or not) as low-rank is questionable. For example, one might prefer to add a noise term to Equation 2.43:

$$\mathbf{\Lambda} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{2.45}$$

$$e_{ij} \sim \mathcal{N}\left(0, \sigma_{ij}^2\right), \tag{2.46}$$

so that the observations are marginally Poisson-lognormal. GLM-PCA allows for a negative binomial link function, so that:

$$y_{ij} \sim \mathcal{NB}\left(\exp\left(\lambda_{ij}\right), \phi_j\right) \tag{2.47}$$

$$\mathbf{\Lambda} = \mathbf{L}\mathbf{F}^T. \tag{2.48}$$

I find the Poisson-lognormal model somewhat more appealing, since it adds noise to the "true" means rather than inflating the sampling variance, but there is an additional subtlety. In the negative binomial model (Equation 2.47), it's clear that $\mathbb{E}\left(y_{ij}\right) = \exp\left(\lambda_{ij}\right)$, so that the log-transformed matrix of true means $\log\left(\mathbb{E}\mathbf{Y}\right)$ is assumed to be low-rank. In the Poisson-lognormal model (Equations 2.45-2.46),

$$\mathbb{E}\left(y_{ij}\right) = \mathbb{E}\left(\exp\left(\lambda_{ij}\right)\right) \tag{2.49}$$

$$= \exp\left(\lambda_{ij} + \frac{\sigma_{ij}^2}{2}\right), \tag{2.50}$$

so that the low-rank assumption applies to the matrix

$$\log\left(\mathbb{E}\mathbf{Y}\right) - \frac{1}{2}\mathbf{\Sigma}. \tag{2.51}$$

Thus there is embedded within the Poisson-lognormal model an assumption about low-rank stucture in $\mathbf{\Sigma}$.

2.3.2 EBMF Model for Count Data

Despite the obvious model misspecification, there are distinct advantages to using Gaussian methods such as EBMF: they're mathematically convenient and as a result, they can often be fit relatively quickly. While the GLM-PCA model with negative binomial link is clearly a more plausible model for count data, it can be almost prohibitively slow to fit to very large datasets. Further, GLM-PCA does not model sparsity in loadings, so that the trade-off between EBMF and GLM-PCA is not merely between accuracy and performance, but also between accuracy (correctly modeling noise) and interpretability (flexibly modeling sparsity). For illustrations, see the Results in Section 2.4: as it turns out, Gaussian methods often work surprisingly well in spite of the model misspecification.

The application of Gaussian methods is hardly straightforward, however: in particular, they typically do very poorly on the untransformed data matrix $\mathbf{Y}$. In scRNA-seq datasets, for example, sampling errors will typically differ by several orders of magnitude across entries (assuming Poisson variance). There have been many proposals for transforming $\mathbf{Y}$, including using variance-stabilizing transformations and computing Pearson residuals: for a brief review, see Ahlmann-Eltze and Huber [2021]. By far the most common practice is to scale each cell's counts by a "size factor" c_i and then transform the data to a logarithmic scale,

adding a "pseudo-count" α to avoid taking logarithms of zero:

$$x_{ij} = \log\left(\frac{y_{ij}}{c_i} + \alpha\right). \tag{2.52}$$

Note that the family of transformations

$$x_{ij} = \log\left(\frac{y_{ij}}{\alpha c_i} + 1\right) \tag{2.53}$$

is equivalent up to a constant offset $\log \alpha$ but preserves sparsity. Typically, some variation of "library-size normalization" is used, so that, for example,

$$c_i = \frac{\sum_j y_{ij}}{\frac{1}{n}\sum_{i,j} y_{ij}}. \tag{2.54}$$

In this case, α in the sparsity-preserving family of transformations (Equation 2.53) is perhaps more naturally thought of as a scaling factor than a pseudo-count.

I follow common practice in using a log transformation, since it's approximately variance-stabilizing while also remaining interpretable: for sufficiently large counts,

$$\log\left(\frac{y_{ij}}{c_i} + \alpha\right) \approx \log\left(\frac{y_{ij}}{c_i}\right), \tag{2.55}$$

so that factors in a decomposition $\mathbf{X} \approx \mathbf{L}\mathbf{F}^T$ can be interpreted as (approximately) multiplicative effects on gene expression. Further, I use the library-size normalization given by Equation 2.54 rather than a more sophisticated (but also more time-consuming) method such as the approach detailed by Lun et al. [2016] and implemented in R package **scran**. In spite of its simplicity, there are good theoretical arguments in its favor: in particular, the Poisson model can best be understood as an approximation to a more biologically plausible multinomial model, for which library-size normalization is the correct choice of size factor (Townes et al. [2019]). Finally, I heed the advice of Lun [2018] in giving some thought to the

choice of scaling factor α. If the aim were simply the approximation given by Equation 2.55, then it would make sense to choose a very small pseudo-count. However, as Lun shows, a too small choice of α produces artificial population structure. In particular, cells with similar size factors c_i will tend to cluster nearer one another than cells with widely differing size factors. But on the other hand, as $\alpha \to \infty$,

$$\log\left(\frac{y_{ij}}{\alpha c_i} + 1\right) \approx \frac{y_{ij}}{\alpha c_i}, \tag{2.56}$$

so that a factorization of $\mathbf{X}$ essentially amounts to factorizing the untransformed matrix $\mathbf{Y}$.

In sum, then, I'm interested in the family of models

$$x_{ij} = \log\left(\frac{y_{ij}}{\alpha c_i} + 1\right) \tag{2.57}$$

$$c_i = \frac{\sum_j y_{ij}}{\frac{1}{n}\sum_{i,j} y_{ij}} \tag{2.58}$$

$$\mathbf{X} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{2.59}$$

$$e_{ij} \sim \mathcal{N}\left(0, \sigma_{ij}^2\right) \tag{2.60}$$

$$\ell_{ik} \sim g_k^{(\ell)} \in \mathcal{G}_\ell \tag{2.61}$$

$$f_{jk} \sim g_k^{(f)} \in \mathcal{G}_f, \tag{2.62}$$

where $\alpha > 0$. In Section 2.3.5, I consider methods for choosing the scaling factor α. All of my more objective attempts fail, so I end up following Lun [2018] in setting

$$\alpha = \max\left\{1,\ \frac{1}{\min_i c_i} - \frac{1}{\max_i c_i}\right\}, \tag{2.63}$$

where the c_is are as in Equation 2.58.

2.3.3 Semi-Nonnegative Matrix Factorization

One methodological innovation I'd like to highlight in this chapter is the use of semi-nonnegative matrix factorization (SNMF) to understand scRNA-seq data. As the name implies, SNMF is situated somewhere in between non-negative matrix factorization (NMF) and unconstrained matrix factorization methods such as PCA or factor analysis.

Non-nonnegative matrix factorization (NMF) was popularized by Lee and Seung [1999] as a method for understanding non-negative matrix data by decomposing it into a sum of component parts:

$$\mathbf{Y} \approx \mathbf{L}\mathbf{F}^T,\ \mathbf{L} \geq 0,\ \mathbf{F} \geq 0, \tag{2.64}$$

where the inequality constraints are element-wise. Often, the "components" $\boldsymbol{\ell}_k$ and $\mathbf{f}_k$ turn out to be quite interpretable. In Lee and Seung's original analysis of images of human faces, they show that while PCA produces "eigenfaces," only some of which "resemble distorted versions of whole faces," NMF decomposes images of faces into "localized features that correspond better with intuitive notions of the parts of faces." In a companion paper, Lee and Seung [2000] provide algorithms that perform NMF by minimizing the loss function

$$\|\mathbf{Y} - \mathbf{L}\mathbf{F}^T\|_2^2 \tag{2.65}$$

or, alternatively, the "KL-divergence" loss function

$$D\left(\mathbf{Y} \| \mathbf{L}\mathbf{F}^T\right) = \sum_{i,j}\left(y_{ij}\log\left(\frac{y_{ij}}{\sum_{k=1}^K \ell_{ik}f_{jk}}\right) - y_{ij} + \sum_{k=1}^K \ell_{ik}f_{jk}\right), \tag{2.66}$$

where both loss functions are subject to non-negative constraints on $\mathbf{L}$ and $\mathbf{F}$. (In both cases, K must be fixed in advance, and typically $K \ll \min(n, p)$.) Note that NMF with KL-divergence loss is equivalent to Poisson NMF (see Section 2.3.1), since the minimizer of the KL-divergence loss function is also the maximizer of the log likelihood of the Poisson

model given by Equations 2.35-2.37.

Another way to perform model-based NMF is to fit EBMF with prior families $\mathcal{G}$ consisting of distributions with non-negative support, such as the "non-negative" family described in Chapter 1:

$$\mathcal{G}_{\text{nn}} := \left\{ g : g \overset{\mathcal{D}}{=} \int_0^\infty \text{Unif}[0, a] \; dh(a) \text{ for some distribution } h \right\} \tag{2.67}$$

It's not yet clear how well non-negative EBMF works in practice. Since each new factor can only add a positive quantity to previous factors, the greedy algorithm doesn't work well, and in my experience backfitting from a NMF initialization usually doesn't usually give noticeably different results from NMF itself (since NMF already yields sparse results).

While NMF has been an extremely productive area of research, SNMF has received little attention. The idea is similar, but only $\mathbf{L}$ or $\mathbf{F}$ is constrained rather than both. Thus there are two possibilities: either

$$\mathbf{Y} \approx \mathbf{L}\mathbf{F}^T, \; \mathbf{L} \geq 0 \tag{2.68}$$

or

$$\mathbf{Y} \approx \mathbf{L}\mathbf{F}^T, \; \mathbf{F} \geq 0. \tag{2.69}$$

Like NMF, SNMF can enhance interpretability with respect to unconstrained matrix factorizations. As I will show, however, an SNMF interpretation is usually quite different from NMF. As a result, SNMF can be more or less appropriate depending on the application.

In the setting in which I'm most particularly interested, there are hundreds if not thousands of gene sets (corresponding to, say, biological pathways) that are differentially expressed across cells and are not in general strictly coordinated, so an NMF-style interpretation in which factors are "component parts" is daunting and perhaps not entirely useful. SNMF offers a simpler interpretation according to which factors are shared departures from some "average" pattern of gene expression. This average pattern is given by the first fac-

tor: because $\mathbf{Y}$ is non-negative, it's almost always the case that both $\boldsymbol{\ell}_1 \succ 0$ and $\mathbf{f}_1 \succ 0$, regardless of any constraints on $\mathbf{L}$ or $\mathbf{F}$. Typically, then, $\mathbf{f}_1$ is strongly correlated with mean expression (but not identical to it), while $\boldsymbol{\ell}_1$ correlates with library size.

Now, if $\mathbf{L} \geq 0$, then $\boldsymbol{\ell}_k$ (with $k > 1$) picks out a (typically sparse) set of cells whose patterns of expression all deviate from average gene expression in a similar way. Some genes will be expressed to a greater degree (these will have positive loadings f_{jk}); others will be repressed ($f_{jk} < 0$); and others will be scarcely affected. It is thus possible to interpret $\mathbf{f}_k$ as a coordinated biological program in which a number of pathways are activated while others are de-activated. Thus $\mathbf{f}_k$ can yield useful biological information about cell types, especially when $\boldsymbol{\ell}_k$ is very sparse. If $\mathbf{L}$ were unconstrained, then interpretation would become somewhat more strained, as it is difficult to imagine that there are two biological programs, one of which is the mirror image of the other (in this interpretation, cells for which $\ell_{ik} > 0$ would express the biological program $\mathbf{f}_k$, while cells for which $\ell_{ik} < 0$ would express the biological program $-\mathbf{f}_k$). It would of course be possible to "absorb" some of $\mathbf{f}_k$ into $\mathbf{f}_1$ so that an unconstrained $\boldsymbol{\ell}_k$ would represent the relative level at which $\mathbf{f}_k$ is present — either above or below some "mean" level — but then sparsity in $\boldsymbol{\ell}_k$ becomes somewhat less meaningful.

If, on the other hand, $\mathbf{F} \geq 0$, then $\mathbf{f}_k$ picks out a set of genes (again, typically sparse) that all act in the same direction — that is, they are all either activated or de-activated together. This constraint can vastly simplify the task of gene set enrichment analysis: when $\mathbf{F}$ is unconstrained, then what one often finds is that the positive loadings in a factor $\mathbf{f}_k$ correspond to one or more pathways or functions, while the negative loadings correspond to very different pathways that are not obviously related to the first. In such cases one can be forced to offer different interpretations for positive and negative loadings. SNMF removes this difficulty.

Note also that it's usually far easier to perform SNMF than NMF, since any unconstrained matrix factorization $\mathbf{Y} \approx \mathbf{LF}$ can be written as a matrix factorization with a nonnegative

constraint on, say, $\mathbf{F}$ by writing

$$\mathbf{Y} \approx \tilde{\mathbf{L}}\tilde{\mathbf{F}}^T \tag{2.70}$$

$$\tilde{\mathbf{L}} = \begin{bmatrix} \mathbf{L} & -\mathbf{L} \end{bmatrix} \tag{2.71}$$

$$\tilde{\mathbf{F}} = \begin{bmatrix} \max\left\{\mathbf{F}, \mathbf{0}_{p\times k}\right\} & \max\left\{-\mathbf{F}, \mathbf{0}_{p\times k}\right\} \end{bmatrix}, \tag{2.72}$$

where the maxima are element-wise. Thus any unconstrained matrix factorization method can potentially provide a good initialization for SNMF.

2.3.4 Incorrect Selection of Number of Factors

One of the advantages of EBMF over other matrix factorization methods is that it can select the number of factors in an "automatic" fashion. Since the EBMF objective is comprised of two terms — on the one hand, the expected data log likelihood, and on the other, the sum of KL-divergences from the priors to the variational posteriors, which effectively act as "penalization terms" for factors and loadings — a new factor will only be added when the penalty incurred is compensated for by a sufficient reduction in the expected squared residuals.

Together with the data transformation, however, the misspecification of the noise model causes `flash` to no longer be able to reliably select the correct number of factors. For this reason, one must either determine K using a method such as parallel analysis or "biwhitening" (Dobriban and Owen [2019]; Landa et al. [2021]) — or, because the rank of scRNA-seq data is usually estimated to be quite large, one can simply fix K so that a manageable number of factors is obtained.

To demonstrate that K can no longer be automatically selected, I simulate data under different noise models and fit `flash` objects using the greedy algorithm. For each trial, I use the same $n \times n$ matrix of true means $\mathbf{L}\mathbf{F}^T$, where $\mathbf{L}, \mathbf{F} \in \mathbb{R}^{n\times K}$. I fix $K = 10$ for all

trials but I vary n from 100 to 10000. I simulate entries of $\mathbf{L}$ and $\mathbf{F}$ using point-normal distributions

$$\ell_{ik} \sim \pi_\ell^{(k)} \delta_0 + \left(1 - \pi_\ell^{(k)}\right) \mathcal{N}\left(0, \sigma_\ell^{2(k)}\right) \tag{2.73}$$

$$f_{jk} \sim \pi_f^{(k)} \delta_0 + \left(1 - \pi_f^{(k)}\right) \mathcal{N}\left(0, \sigma_f^{2(k)}\right), \tag{2.74}$$

with

$$\pi_\ell^{(k)},\ \pi_f^{(k)} \sim \text{Beta}\left(0.5, 0.5\right) \tag{2.75}$$

$$\sigma_\ell^{2(k)},\ \sigma_f^{2(k)} \sim \text{Gamma}\left(1, 2\right). \tag{2.76}$$

Given a matrix of true means $\mathbf{LF}^T$, I simulate a data matrix $\mathbf{Y}$ using four different noise models:

- **Gaussian**. This is simply the EBMF model:

$$\mathbf{Y} = \mathbf{LF}^T + \mathbf{E} \tag{2.77}$$

$$e_{ij} \sim \mathcal{N}\left(0, 0.5^2\right). \tag{2.78}$$

  Given enough data, `flash` should be able to recover the correct number of factors.

- **Heavy-tailed**. Noise is again additive, but I use a heavier-tailed t_5 distribution:

$$\mathbf{Y} = \mathbf{LF}^T + \mathbf{E} \tag{2.79}$$

$$e_{ij} \sim 0.5 t_5. \tag{2.80}$$

- **Poisson**. The GLM-PCA model with Poisson link:

$$y_{ij} \sim \text{Poisson}\left(\exp\left(\lambda_{ij}\right)\right) \tag{2.81}$$

$$\mathbf{\Lambda} = \mathbf{L}\mathbf{F}^T. \tag{2.82}$$

- **Poisson-lognormal**. Similar to the GLM-PCA model, but with Gaussian noise added to the Poisson means:

$$y_{ij} \sim \text{Poisson}\left(\exp\left(\lambda_{ij}\right)\right) \tag{2.83}$$

$$\mathbf{\Lambda} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{2.84}$$

$$e_{ij} \sim \mathcal{N}\left(0, 0.5^2\right). \tag{2.85}$$

For the Poisson and Poisson-lognormal noise models, I fit `flash` to the `log1p`-transformed matrix

$$\mathbf{X} = \log\left(\mathbf{Y} + 1\right). \tag{2.86}$$

Results are shown in Figure 2.3. The upper-left panel confirms Wang and Stephens [2021]'s claim that, given enough data and given Gaussian noise, `flash` should be able to select the correct number of factors. The upper-right panel suggests that `flash` is fairly robust to misspecification of the noise model when the errors are simply leptokurtic (yet still symmetric and heteroskedastic). In the Poisson and Poisson-lognormal cases, however, the number of factors selected by `flash` is completely unreliable.

2.3.5 Pseudo-counts and the Implied Discrete Distribution

While comparing `flash` objectives is a natural way to compare different EBMF fits to the same data, there's no simple way to compare different data transformations. To compare pseudo-counts, for example, one can't delete matrix entries and calculate Poisson likelihoods

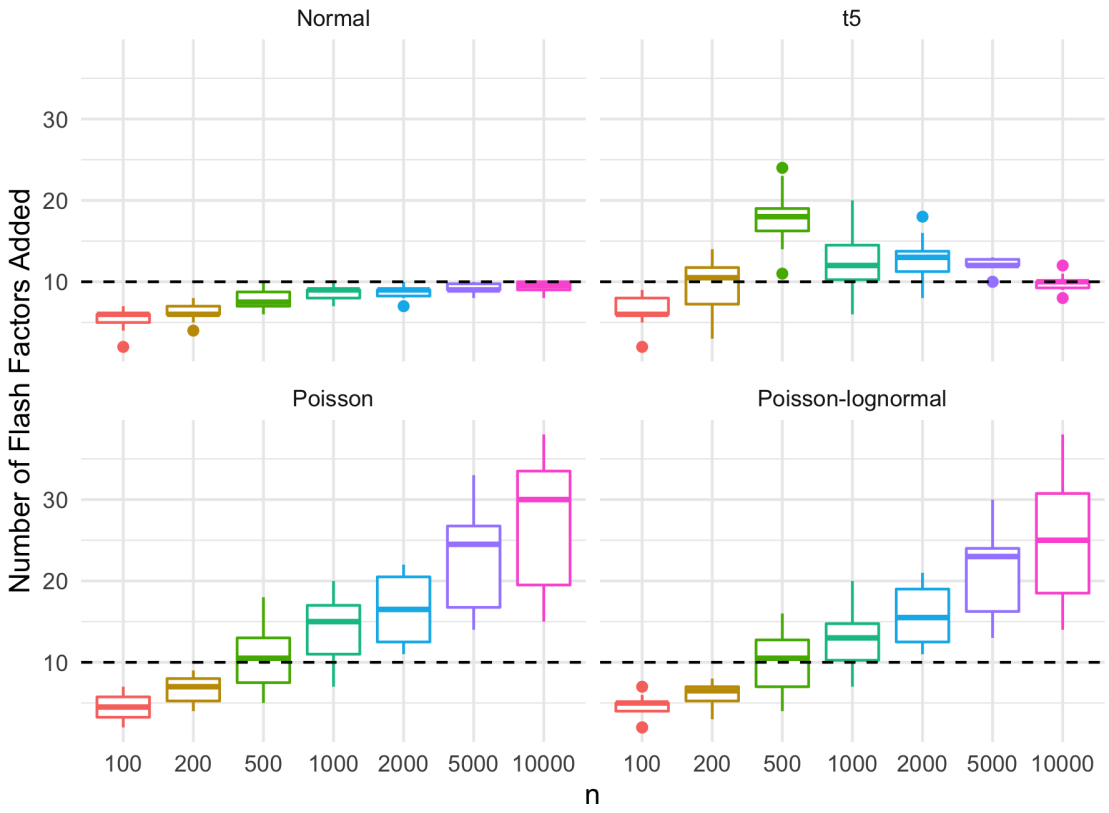


Figure 2.3: Effects of different noise models on the number of factors added by `flash`. For each trial, a $n \times n$ matrix $\mathbf{LF}^T$ with ten true factors is simulated and then data matrices are generated under four different noise models: additive Gaussian and t_5 errors; Poisson noise with means $\exp\left(\mathbf{LF}^T\right)$; and Poisson noise with means $\exp\left(\mathbf{LF}^T + \mathbf{E}\right)$, where entries e_{ij} are i.i.d. Gaussian. In the latter two cases, `flash` is fit to the transformed data matrix $\log\left(\mathbf{Y} + 1\right)$.

for imputed means, since the inverse transformation

$$\lambda_{ij} = \alpha c_i \left(e^{\mu_{ij}} - 1\right) \tag{2.87}$$

does not guarantee that the imputed Poisson means λ_{ij} will be nonnegative.

Nor is it sufficient to do a change-of-variables adjustment to the `flash` objective: as it turns out, the adjusted ELBO is monotonically decreasing as a function of the pseudo-count α. An illustration of this phenomenon is given in Figure 2.4. To gain some intuition, imagine fitting a simple rank-one model with a constant variance structure. The data log likelihood is

$$-\frac{np}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i,j}\mathbb{E}(x_{ij} - \ell_i f_j)^2. \tag{2.88}$$

Recall that σ^2 is estimated (via maximum likelihood) as the mean expected squared residual, so that the data log likelihood can be written

$$-\frac{np}{2}\log(2\pi\hat{\sigma}^2) - \frac{np}{2} = -np\log\hat{\sigma} + C. \tag{2.89}$$

Meanwhile, the change-of-variables adjustment to the ELBO is

$$np\log(1/\alpha) - \sum_{i,j} x_{ij}. \tag{2.90}$$

Now take $\alpha \to 0$. Zero entries in $\mathbf{Y}$ are zero in the transformed matrix $\mathbf{X}$, and nonzero entries are $\log\left(\frac{y_{ij}}{c_i}\right) - \log\alpha \to -\log\alpha$, so

$$\sum_{i,j} x_{ij} \to snp\log(1/\alpha), \tag{2.91}$$

where s is the sparsity of $\mathbf{Y}$ (that is, the proportion of entries that are nonzero). Next, since the rank-one fit will yield a $\hat{\sigma}^2$ that is approximately equal to $\mathrm{Var}(\mathbf{X})$, which (in the limit)

is $(\log(1/\alpha))^2 s(1-s)$, the data log likelihood (see Equation 2.89) is approximately

$$-np\log(\sqrt{s(1-s)}\log(1/\alpha)) + C = -np\log\log(1/\alpha) + C. \tag{2.92}$$

Thus, for α near zero, the adjusted log likelihood (plugging Equation 2.91 into Equation 2.90) is approximately

$$(1-s)np\log(1/\alpha) - np\log\log(1/\alpha) + C, \tag{2.93}$$

which blows up as $\alpha \to 0$.

To take a different tack, I've attempted to replace the `flash` data log likelihood with the likelihood of the "implied discrete distribution." The EBMF model given by Equations 2.57-2.62 yields marginal distributions

$$y_{ij} \;\Big|\; \mathbf{L}, \mathbf{F}, \sigma_{ij}^2 \sim \text{Lognormal}\left(\left[\mathbf{L}\mathbf{F}^T\right]_{ij} + \log\left(\alpha c_i\right), \sigma_{ij}^2\right) - \alpha c_i, \tag{2.94}$$

which has cumulative distribution function (CDF)

$$\xi(z) = \Phi\left(\frac{\log\left(\frac{z}{\alpha c_i} + 1\right) - \left[\mathbf{L}\mathbf{F}^T\right]_{ij}}{\sigma_{ij}^2}\right) \tag{2.95}$$

where Φ denotes the CDF of the standard normal distribution. I convert the shifted lognormal distribution to a discrete distribution on the nonnegative integers by binning values:

$$\mathbb{P}\left(y_{ij} = z\right) = \begin{cases} \xi(0.5), & z = 0 \\ \xi(z+0.5) - \xi(z-0.5), & z \in \mathbb{Z}^+, \\ 0, & z \notin \mathbb{Z}^{0+} \end{cases} \cdot \tag{2.96}$$

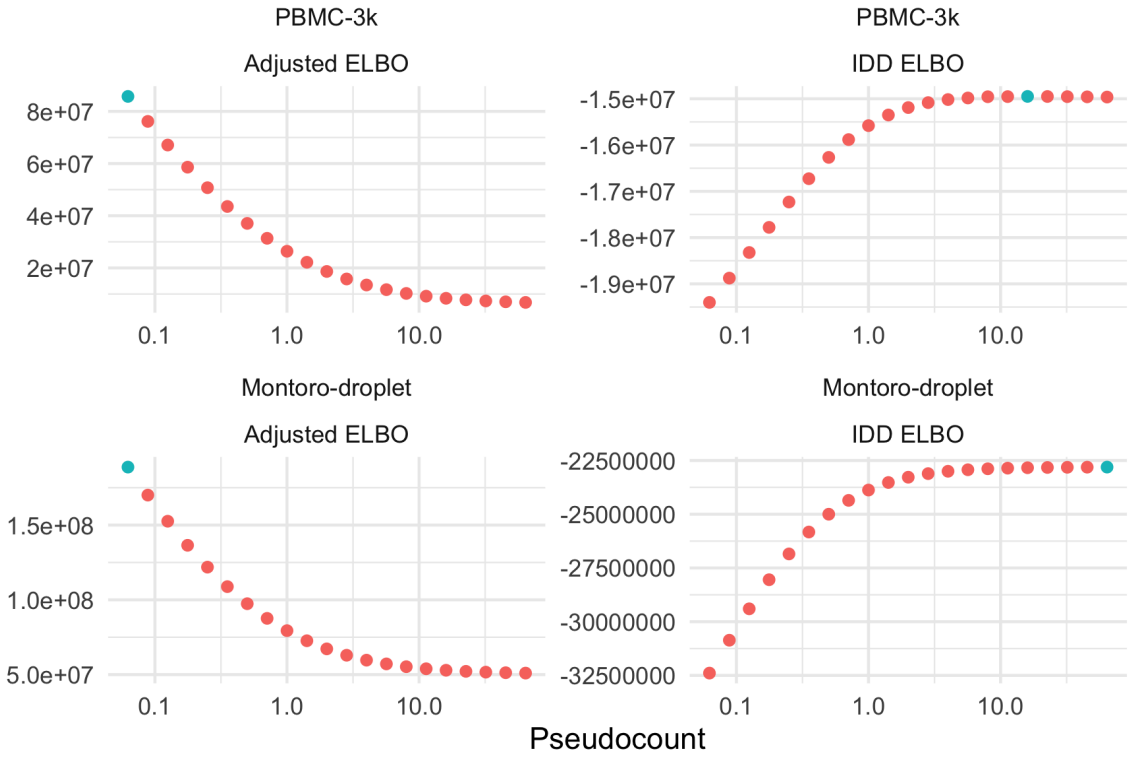


Figure 2.4: Comparison of different pseudo-counts when fitting the PBMC-3k and Montoro-droplet datasets. Each fit adds 20 factors using the greedy algorithm without backfitting. The blue point represents the maximum value obtained in each plot. The "adjusted ELBO" uses a change-of-variables adjustment to attempt to account for the data transformation. The IDD (implied discrete distribution) ELBO bins the shifted lognormal distributions of the EBMF model and calculates the data log likelihood using the resulting discrete distribution. See text for details.

Since these probabilities are with respect to the observed (untransformed) data $\mathbf{Y}$, they should be comparable across different values of α. (In practice, I fix $\mathbf{L}$ and $\mathbf{F}$ at their expected values since their posterior variances are relatively unimportant in comparison to the residual variance σ^2.)

In Figure 2.4, I plot the implied discrete distribution (IDD) ELBO (which replaces the EBMF data log likelihood with the log likelihood of the data under the implied discrete distribution) for different pseudo-counts using the PBMC-3k and Montoro-droplet datasets (see Section 2.2.7 for descriptions). For the latter, the IDD ELBO increases over the range

of pseudo-counts used. For the PMBC-3k dataset, it peaks at $\alpha = 16$, which is still quite a bit larger than most authors recommend, since large pseudo-counts wash out cell-to-cell differences for very sparse genes.

In sum, it seems very difficult to choose a pseudo-count using any objective measure. One is all but compelled to resort to heuristic arguments in order to choose a pseudo-count that, Goldilocks-like, is not too small but not too large: not so small that population artefacts become a serious problem, but not so large so as to dilute the effects of genes that are very sparse but important for differentiating among cell types. For this reason, I defer to Lun and use the value of α given above (Equation 2.63).

2.4 Results

To illustrate the potential of EBMF to yield insight into scRNA-seq data, I use **flashier** to fit EBMF models to two UMI-based datasets, both of which were used for benchmarking purposes in Section 2.2.7: the "PBMC-3k" dataset, which consists of $n = 3{,}205$ peripheral blood mononuclear cells from a single donor (Zheng et al. [2017]), and the "Montoro-droplet" dataset, which consists of $n = 7{,}193$ mouse epithelial cells (Montoro et al. [2018]).

I performed minimal pre-processing before fitting. First, I removed cells with unusually large library sizes relative to the rest of the dataset. Three outlying cells were removed from the PBMC-3k dataset; two were removed from the Montoro-droplet dataset. Next, I removed features with nonzero counts in fewer than ten cells (no other feature selection was performed). 14,462 features remained in the PBMC-3k dataset, and 14,481 features in the Montoro-droplet dataset.

I obtained matrix factorizations for each dataset using five different methods, fixing the number of factors at $K = 12$ for the PBMC-3k dataset and $K = 20$ for the Montoro-droplet dataset:

- **"flashier-pn"**: Uses **flashier** to fit an EBMF model using point-normal prior families

(Equation 2.5). I normalized counts using library-size normalization (Equations 2.57-2.58) with scaling factor $\alpha = 2.94$ for the PBMC-3k dataset and $\alpha = 3.49$ for the Montoro-droplet dataset. (These values of α were obtained using Lun's formula, given by Equation 2.63.) I added factors using the greedy algorithm and then backfitted until convergence. To avoid convergence issues, I set the floor for residual variance estimates at $1/n$ (see Section 2.2.5).

- **"flashier-snn"**: Semi-nonnegative EBMF with nonnegative priors (Equation 2.67) on the loadings (cells) and scale mixture of normal priors (see Section 1.3.2) on the factors (genes). Normalization, scaling factors, and residual variance lower bounds were identical to the flashier-pn fit. I used the flashier-pn fit to initialize the loadings and factors (see Equations 2.70-2.72). Since this initialization yields $2K$ factors, I pruned the `flash` object after running a small number of backfitting iterations, retaining only the K factors with the largest proportion of variance explained (as calculated by **flashier**). After pruning, I backfitted until convergence.

- **"topic-model"**: Uses the R package **fastTopics** (Carbonetto et al. [2021]) to fit a topic model to the data (see Section 2.3.1). No normalization or data transformation is required for topic models. All default settings were used.

- **"glmpca-pois"**: Fits GLM-PCA (see Section 2.3.1) via the R package **glmpca** (Townes et al. [2019]) using a Poisson link function (`fam = "poi"`). While GLM-PCA accepts raw counts, the model does require size factors, which were again calculated using library-size normalization (Equation 2.58). Default settings were used.

- **"glmpca-nb"**: Similar to glmpca-pois, but uses a negative binomial link function with feature-specific overdispersion (`fam = "nb2"`).

The time required to fit each dataset is given in Table 2.4.

	flashier.pn	flashier.snn	topic.model	glmpca.pois	glmpca.nb
PBMC-3k	0.17	0.64	0.11	0.38	1.84
Montoro-droplet	1.42	4.69	0.46	1.65	NA

Table 2.4: Hours required to factorize the PBMC-3k and Montoro-droplet datasets using each of the methods described in the text. The "glmpca-nb" method was given 24 hours to fit the Montoro-droplet dataset, but didn't finish.

2.4.1 PBMCs and Dendritic Cells

The PBMC-3k dataset includes 11 cell types, which were inferred by Freytag et al. [2018] via comparison with sorted cells. The two rarest cell types are CD34+ (17 cells) and dendritic (19 cells).

Results are shown in Figure 2.5, with the top genes for each factor in the flashier-snn fit listed in Figure 2.6. Clearly, the EBMF and topic model fits yield factors that are much more cleanly correlated with cell type than the GLM-PCA fit. Further, only EBMF yields a clean "dendritic factor" (factor 8 in the flashier-pn fit; factor 9 in the flashier-snn fit). Thus the EBMF fit appears to be most successful at picking out rare cell types (assuming, of course, that cell types have been correctly inferred).

The differences between the flashier-pn and flashier-snn fits are intriguing. For example, whereas factors 2, 3, and 6 seem difficult to interpret in the flashier-pn fits, factor 6 in the flashier-snn fit more clearly picks out CD56+ NK cells. In other words, the flashier-snn factor loadings $\mathbf{f}_6$ can be interpreted as a "gene programs" for CD56+ NK cells, whereas interpretation for the corresponding factor in the flashier-pn fit is less clear (see Section 2.3.3 for a discussion).

Since the EBMF fits are best poised among all methods to yield biological insights into dendritic cells, I include a volcano plot for the flashier-snn dendritic factor in Figure 2.7. CLEC4C, for example, is a marker gene for plasmacytoid dendritic cells (pDCs), and IRF8 has been identified as playing an essential role in the development of pDCs (Harman et al. [2013]; Sichien et al. [2016]).

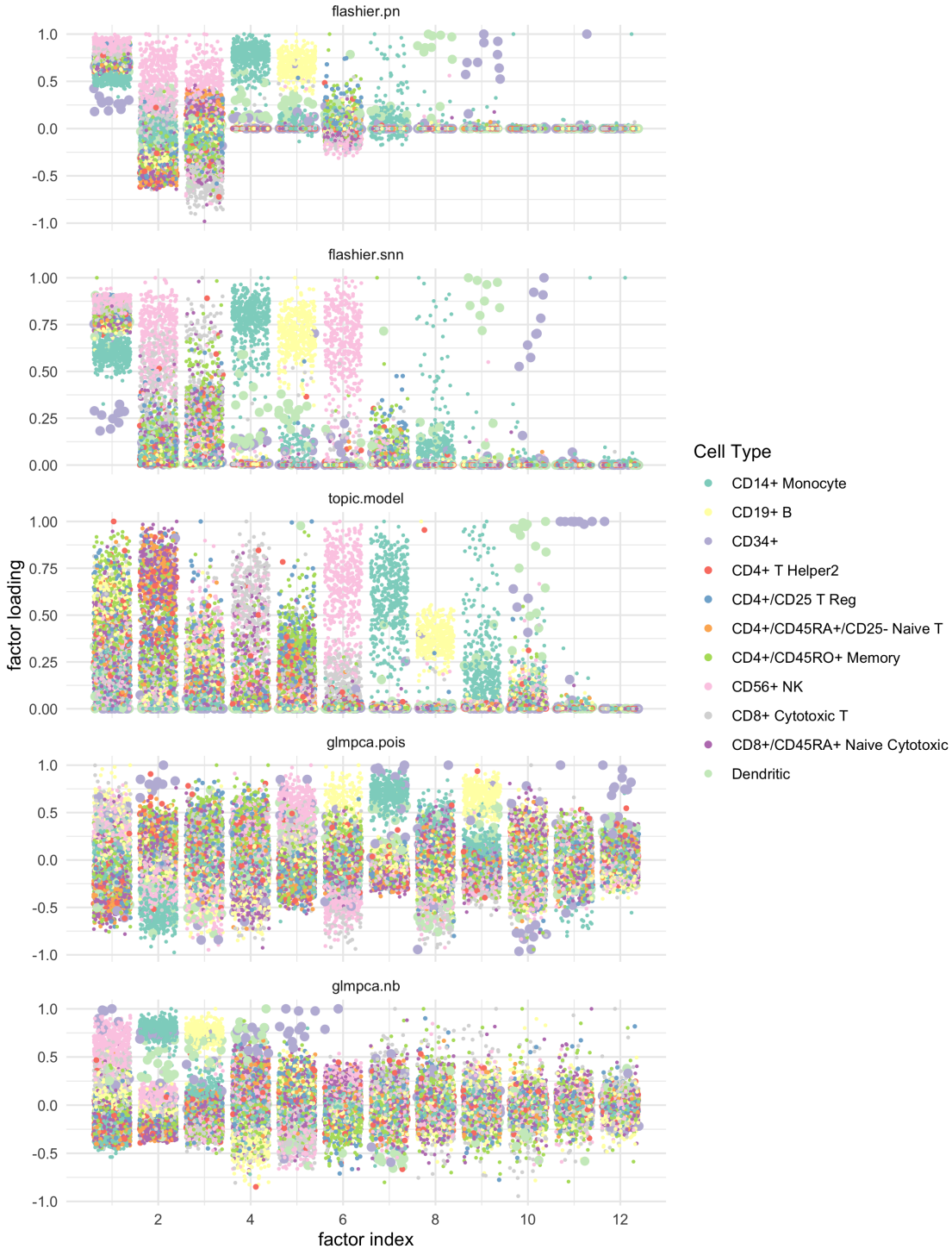


Figure 2.5: Factor plots for the PBMC-3k dataset obtained using five different methods (see text for details). The factors are normalized so that $\|\mathbf{f}_k\|_\infty = 1$ and then sorted by ℓ_1-norm so that "sparser" factors appear to the right. Points are colored according to cell type, with cell size proportional to the rarity of the cell type. Horizontal jitter is added randomly.

Factor	Top Genes (by |z|-score)
1	RPS27, RPL41, RPL21, RPL13, EEF1A1, RPL34, RPL10, RPL13A, MALAT1, RPL32
2	NKG7, RPL13, RPL34, CCL5, RPL32, RPL21, EEF1A1, RPS27, RPL39, RPS15A
3	KLRB1, GZMK, S100A4, ZFP36L2, LYAR, TMSB10, IL7R, SLC4A10, IL32, CCR7
4	LYZ, S100A9, S100A8, VCAN, FTL, TYROBP, FOS, CST3, CTSS
5	CD74, HLA-DRA, HLA-DRB1, CD79A, MS4A1, LINC00926, HLA-DPB1, HLA-DQB1, HLA-DPA1
6	KLRF1, CCL5, FCER1G, KLRB1, SPON2, IL32, CLIC3, GZMH, TYROBP
7	NKG7, RPL21, RPS27, CCL5, MALAT1, RPL3, RPS14, RPL34, RPS27A, RPL13
8	LYPD2, CDKN1C, MS4A7, LST1, FCGR3A, AIF1, VMO1, SMIM25, CKB
9	CLEC4C, SERPINF1, SCT, LILRA4, LRRC26, TPM2, PPP1R14B, PLD4, SMPD3, DNASE1L3
10	CLEC1B, ACRBP, CMTM5, PTCRA, LY6G6F, TMEM40, C2orf88, PF4, CAVIN2, ARHGAP6
11	CLC, ZNF81, MBOAT2, SEMA3C, HCG27, FGD6, FZD1, INPP5A, DENND2C
12	DPRXP4, CXCL1, CMTM2, FCGR3B, FFAR2, MPV17L, MARVELD1, AMPD3, PDE7B, PCYOX1L

Figure 2.6: Top genes for the flashier-snn fit to the PBMC-3k dataset. Genes are ordered by magnitude of z-score.

2.4.2 Mouse Epithelial Cells and Ionocytes

The Montoro-droplet dataset is the smaller of the two datasets published by Montoro et al. [2018]. Cell types were inferred by the authors. The rarest cell type is ionocytes (26 cells), but goblet (63), neuroendocrine (96), and tuft cells (158) are also relatively scarce.

Loadings for the flashier-pn, flashier-snn, topic-model, and glmpca-pois fits are shown in Figure 2.8 (the glmpca-nb fit didn't finish within 24 hours). The top genes for the flashier-snn fit are listed in Figure 2.9. Again EBMF most clearly yields factors that correlate with rare cell types: tuft factors (factors 12 and 15 in the semi-nonnegative fit), a neuroendocrine factor (16), a goblet factor (18), and an ionocyte factor (20) are all present, while in the topic model tuft, neuroendocrine, and ionocyte cells are all entangled (factor 17).

The ionocyte factor is a particularly strong argument in favor of EBMF, as Montoro et al. consider their main contribution to be the discovery of Cftr-expressing ionocyte cells. And indeed, Cftr is one of the genes that jumps out as significant in the volcano plot in Figure

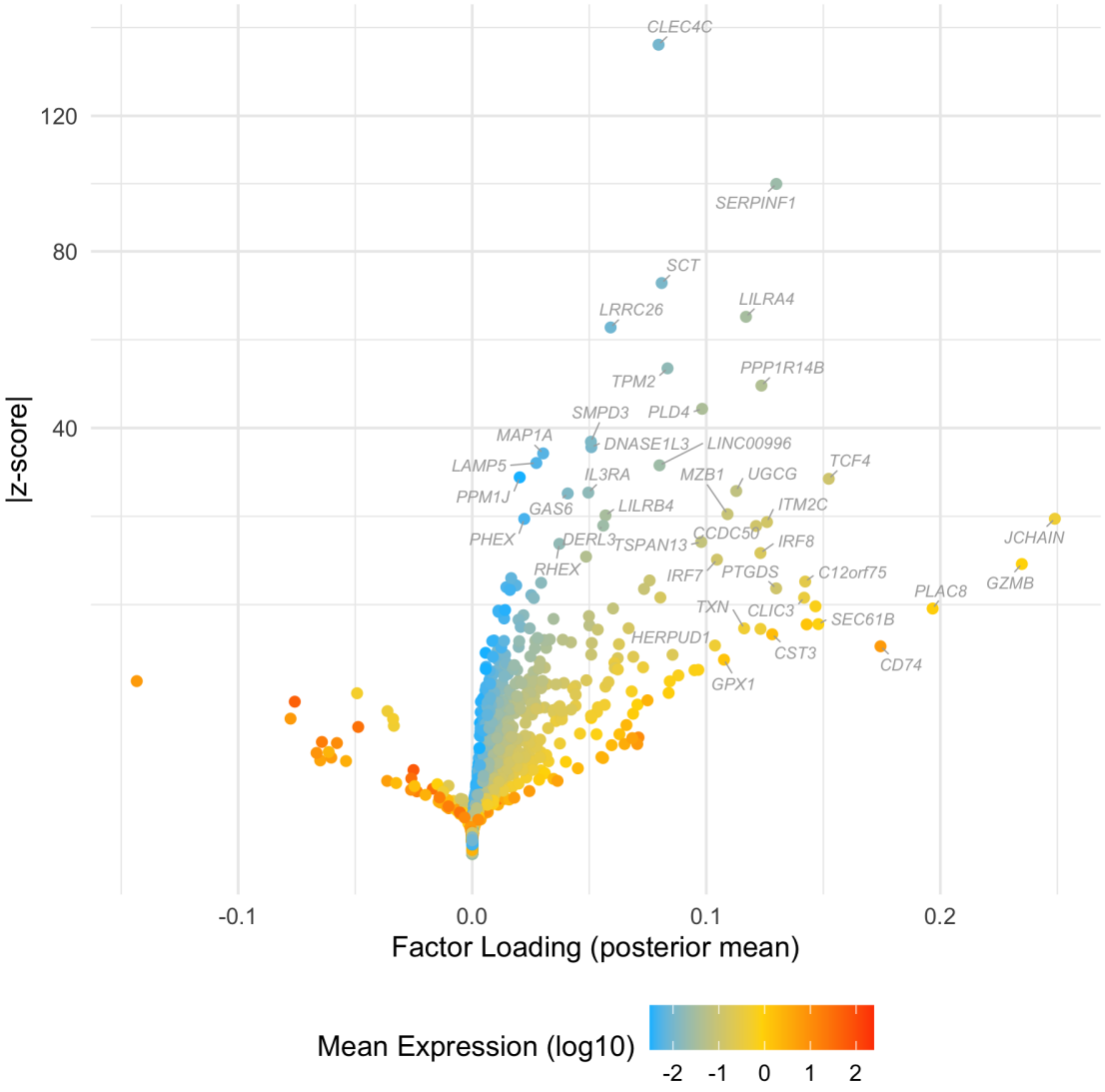


Figure 2.7: Volcano plot for the flashier-snn "dendritic factor" (factor 9 in Figure 2.5).

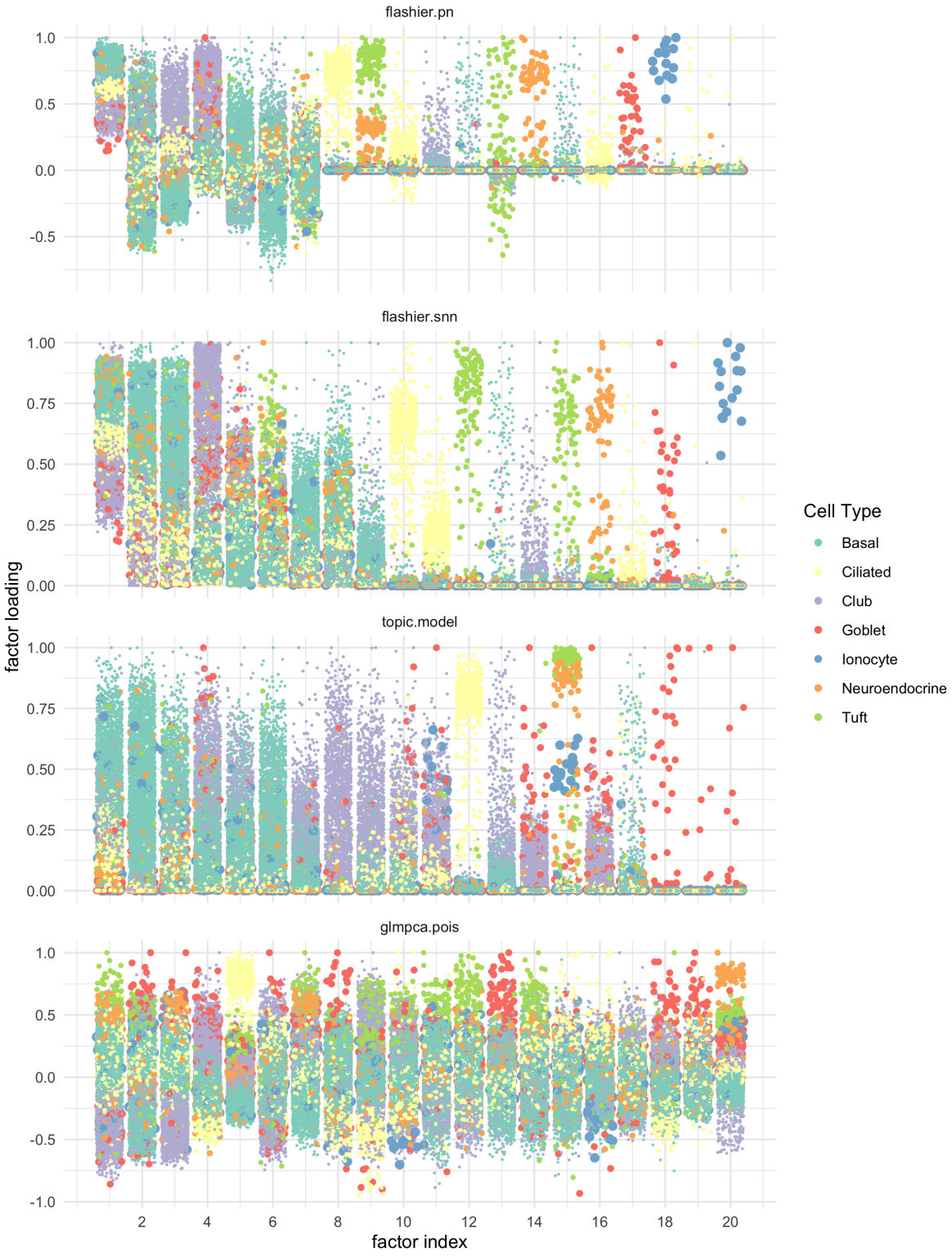


Figure 2.8: Factor plots for the epithelial dataset. See Figure 2.5 and text for details.

Factor	Top Genes (by |z|-score)
1	Cyp2f2, S100a6, Wfdc2, Lgals3, Selenbp1, Gsto1, Fth1, S100a11, Anxa1, Krt19
2	Rplp0, Rps18, Rps14, Rps19, Rpl14, Rpl13a, Rps5, Rpl13, Rps9, Ltf
3	Junb, Atf3, Ubc, Sfn, Dnajb1, Ier2, H3f3b, Jun, Rplp0, Hspb1
4	Scgb1a1, Wfdc2, Scgb3a1, Cyp2f2, Gsto1, Bpifa1, Scgb3a2, Reg3g, Fth1, Bpifb1
5	S100a6, Reg3g, Msln, Scgb1a1, Bpifa1, Lgals3, Rpl32, Junb, Hspa8, Rps14
6	Rpl32, Rpl13a, Rps14, Rplp1, Rpl26, Rps19, Rps18, Rps24, Rps5, Lgals3
7	Apoe, Lgals3, Igfbp7, Bpifa1, Krt18, Dlk2, Tgm2, Anxa1, Bcam, Tagln2
8	Cyp2f2, Tst, Bpifa1, Lypd2, Tppp3, Cav1, Phlda1, Gsto1, Gstm1, Gstm2
9	Krt13, Krt4, Lgals7, Krt6a, Krt5, Upk3bl, Lgals3, Krt14, Aqp3, Tpm2
10	Ccdc153, Dynlrb2, 1110017D15Rik, Tmem212, Fam183b, Rsph1, 1700016K19Rik, 1700007K13Rik, Cfap126, Sntn
11	Cdhr4, Adam8, Cdh26, Mapk15, Dnah6, Dnah9, Cdhr3, Wnt11, Hydin, Dnah10
12	Lrmp, Ltc4s, Alox5ap, Trpm5, Ly6g6f, Dclk1, Gng13, Sh2d6, Rgs13, Selm
13	Birc5, Ube2c, Cdc20, Cdca3, Ccnb1, Cdca8, Cenpa, Ccnb2, Cdk1, Ccna2
14	Ly6g6c, Serpinb2, Nccrp1, Psca, Gm9573, Plac8, Ifit1bl1, Csta1, Krt13, 2300002M23Rik
15	Gnb3, Ovol3, Gng13, Alox5ap, Fxyd6, Dclk1, Gm4952, Ltc4s, Sucnr1, Lrmp
16	Chga, Scg5, Pkib, Calca, Cxcl13, Nov, Ngf, Scg2, Ly6h, Uchl1
17	Shisa8, Foxn4, Ccdc67, Lrrc23, Ccna1, Tmem89, Mcidas, Crocc2, Meig1, Pbx4
18	Lman1l, Cgref1, Gp2, Tff2, Tff1, Nkx3-1, Fkbp11, Fgl2, BC048546, Muc5b
19	C1qc, C1qb, C1qa, Pf4, Ms4a6b, C5ar1, Fcgr3, H2-Eb1, Dab2, Cd48
20	Gm933, Foxi1, Ascl3, Stap1, Moxd1, Syn2, Atp6v0d2, Cftr, Coch, Asgr1

Figure 2.9: Top genes for the flashier-snn fit to the epithelial dataset. Genes are ordered by magnitude of z-score.

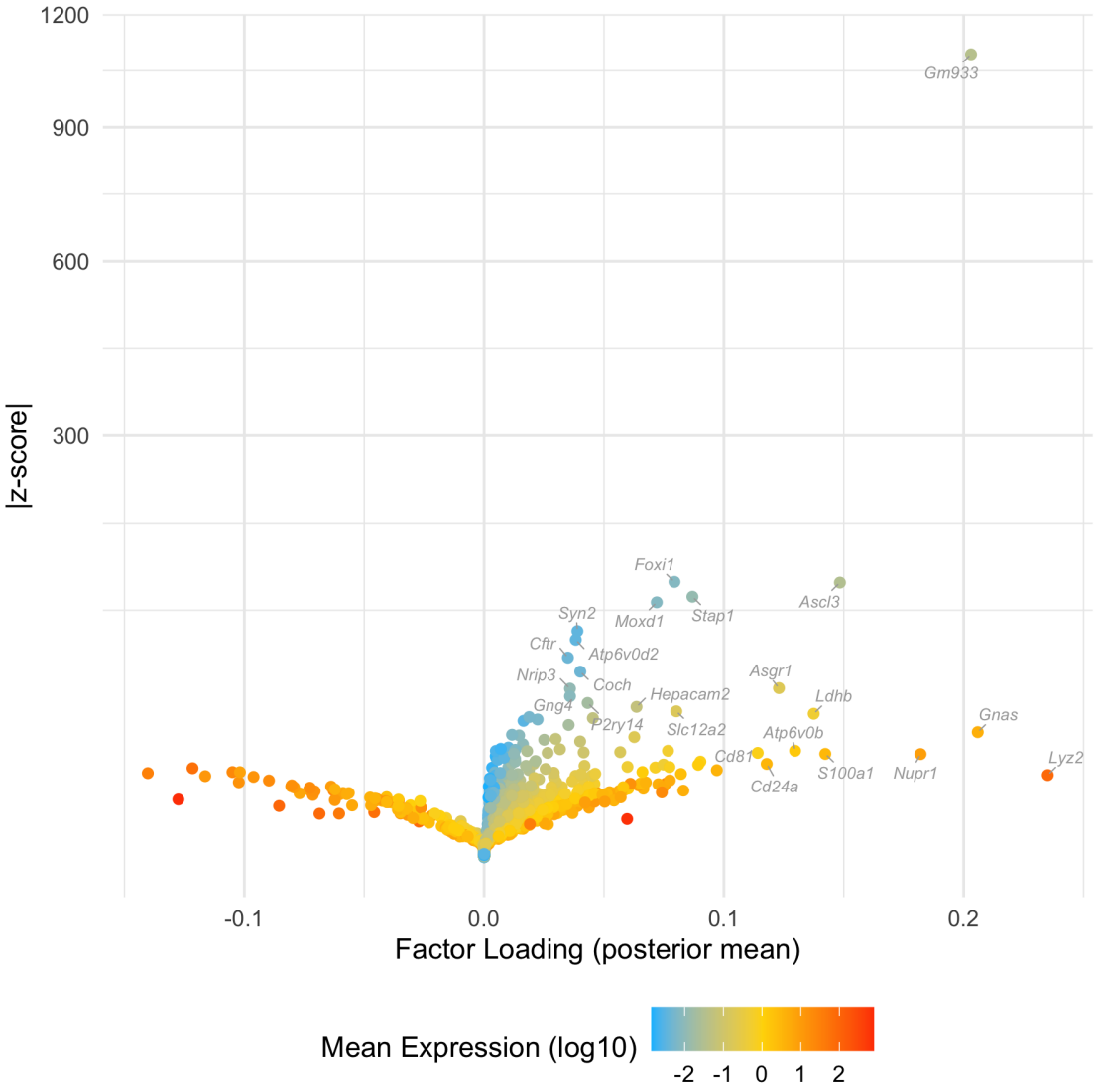


Figure 2.10: Volcano plot for the flashier-pn "ionocyte factor" (the second-to-last factor in Figure 2.8).

2.10, as do the majority of the other genes discussed by the authors, including Foxi1, Ascl3, Atp6v0d2, Gm933 (now known as Smbd1), Stap1, Moxd1, Asgr1, and P2ry14.

2.5 Discussion

While I believe that the results presented in Section 2.4 make a convincing case for the usefulness of EBMF in the analysis of scRNA-seq data, more quantitative measures of the method's performance would be helpful. One thorny issue, however, is that high-quality datasets with a known "ground truth" are not easy to come by, and realistic simulations are not straightforward (see, however, Zappia et al. [2017]; and Gerard [2020]). It is thus not a particularly simple matter to obtain a realistic test dataset by which to measure, for example, the success of various methods in recovering rare cell types.

In Section 2.3, I discussed another context in which a more quantitative measure was lacking: comparing different data transformations. While there has been significant recent interest in the problem of transforming count data for use by Gaussian methods (among others, see Lun [2018]; Townes et al. [2019]; and Ahlmann-Eltze and Huber [2021]), many of the arguments are theoretical without extensive application to real data (but see Sun et al. [2019]). In particular, the family of log transformations given by Equations 2.57-2.58 seems to work much better than the theory suggests that it should. A rigorous comparison of transformations using quantitative measures on real (or realistically simulated) data would help to either shore up the theory or bolster current practice (which overwhelmingly relies on log transformations).

Finally, while Gaussian methods like EBMF are appealing because they're convenient, relatively fast, and seem to work well in practice, it would also be useful to have a method that, like GLM-PCA, were able to model Poisson or negative binomial noise directly, but unlike GLM-PCA, could also model sparsity in factors and loadings. Wang [2017] sketches an idea for Poisson EBMF, but the approach has not yet been shown to work well in practice.

In particular, it involves additional approximations on top of the usual EBMF variational approximation, and it's not clear whether those approximations give better results than the approximations required to apply Gaussian methods to transformed count data. Indeed, a practical method for fitting generalized EBMF models appears to remain somewhat far off. In the meantime, I hope that this chapter will encourage practitioners to adopt **flashier** more widely to gain insights into scRNA-seq and other datasets.

CHAPTER 3

EBMF FOR TREE-STRUCTURED DATA

3.1 Introduction

This chapter considers the potential of matrix factorization methods to identify tree-like structure in matrix data. The particular setting I have in mind is population genetics, where one is often interested to characterize population structure given, say, a matrix of genotype data. Of course, trees are usually quite inexact representations of population structure, so that practitioners are better served by methods that are flexible enough to account for phenomena such as population admixture as well as the strict pattern of drift and divergence that is implied by a tree-like representation. Matrix factorization methods can offer such flexibility (since matrix elements typically have continuous support) while the usual binary methods for identifying trees — for example, hierarchical clustering — often do not.

A number of matrix factorization methods are already commonly used to explore population structure. Principal components analysis (PCA) is the most widely used of the methods, since it can be quickly obtained, even for very large datasets, and is often interpretable (with some effort — see, for example, Novembre et al. [2008]). Nearly as widely used are methods that fit the Pritchard, Donnelly, and Stephens (PSD) or "admixture" model (Pritchard et al. [2000]), which is also known in the machine learning community as the latent Dirichlet allocation (LDA) or "topics" model (Blei et al. [2003]). However, researchers have become increasingly familiar with the limitations of these methods. In many scenarios, PCA can yield mathematical artefacts that are easily misinterpretable as real biological processes (Novembre and Stephens [2008]; McVean [2009]). Admixture models tend to produce clustered results, so that it can prove difficult to infer shared evolutionary histories among populations (Lawson et al. [2018]).

More recently, extensions of PCA and admixture models have been proposed in an effort

to address some of these limitations. Several methods have been developed which help to eliminate artefacts by accounting for spatial or temporal confounding (Bradburd et al. [2018]; Caye et al. [2018]; François et al. [2019]; Frichot et al. [2012]; Joseph and Pe'er [2019]). Other approaches have emphasized sparse matrix factorization methods, since sparsity should in principle enhance interpretability (Engelhardt and Stephens [2010]; Frichot et al. [2014]). This chapter extends the latter line of research by making more precise the *kind* of sparsity that should be expected in order to detect tree-like structure, and by elaborating methods that are better suited to finding appropriate factorizations.

The chapter also draws on work that proposes methods for recovering an underlying tree from a covariance matrix (Bravo et al. [2009]; McCullagh [2006]), singular value decomposition (Eriksson [2005]), or distance matrix (Zhang et al. [2011]). Pickrell and Pritchard [2012] and Yan et al. [2019] have extended these ideas to more complex topologies by developing methods for finding admixture graphs. However, this work has prioritized trees or graphs as output and has not, as far as I know, been used to motivate or understand matrix factorization methods.

The chapter is divided into three main parts. The "Theory" section discusses various sparse factorizations of tree-structured data and culminates with the "divergence factorization," which, I argue, both contains meaningful information about population structure and can be obtained without too much difficulty. The "Methods" section investigates existing sparse matrix factorization methods as well as proposing novel modifications to my preferred method, empirical Bayes matrix factorization (Wang and Stephens [2021]). I refer to the modified method as "tree-EBMF", and I propose an additional modification that makes the method suitable for factorizing co-occurrence matrices ("tree-EBcovMF"). Finally, the "Examples" section tests out tree-EBMF and tree-EBcovMF on both simulated and real-data examples, with results that I find to be encouraging. All figures in this chapter can be reproduced by following the instructions at `https://github.com/willwerscheid/trees-paper`.

3.2 Theory

In Section 3.2.1, I define what I mean by "tree-structured data" and I introduce terminology and notation that will be used throughout the chapter. I then describe three interpretable low-rank representations of tree-structured data

$$\mathbf{X} \approx \mathbf{L}\mathbf{F}^T. \tag{3.1}$$

The "population factorization" yields a factor for each leaf or population (Section 3.2.2); the "drift factorization," a factor for each edge or "drift event" (Section 3.2.3); and the "divergence factorization," a factor for each non-leaf vertex or "divergence event" (Section 3.2.4). Given the structure of the tree, the first two factorizations are immediate. However, the tree is not recoverable from the loadings matrix $\mathbf{L}$ in the population factorization; and while there is a simple mapping between trees and drift factorization loadings matrices, the latter are not in general full-rank and are exceedingly difficult to find via optimization. The divergence factorization, in contrast, is both full-rank and allows the underlying tree to be reconstructed with only slightly more effort than the drift factorization. However, the exact values of the divergence factorization loadings are not immediately available; further, it's not obvious that such a factorization always exists. In Section 3.2.5, I prove that it does exist, and in doing so, I highlight some useful features. Finally, Section 3.2.6 makes some connections to factor analysis by suggesting that it should often be sufficient to work with the co-occurrence matrix $\frac{\mathbf{X}\mathbf{X}^T}{p}$ rather than the full data matrix $\mathbf{X}$, which can be more convenient when there are many more features than observations.

3.2.1 Tree-Structured Data

I use the term "tree-structured" to encompass three assumptions about a dataset:

- Each observation is sampled from one of K populations. In particular, observations

from the same population share a common mean.

- The relationship among populations can be described by a rooted, binary tree with K leaves (one for each population). The edges of the tree correspond to mutually independent "drift events," during which population means experience random change as a result of genetic drift or some other stochastic process. Vertices correspond to "divergence events," where ancestral populations split in two.

- All drift events have mean zero, with the possible exception of the "first" drift event that proceeds from the root. (For most results, this assumption is not strictly necessary, but it will prove useful in dealing with co-occurrence matrices: see Section 3.2.6.)

A detailed example is in order. In population genetics, one would often like to identify structure in a matrix $\mathbf{X}$ of genotype data. I will assume that the observed individuals are diploid, so that $\mathbf{X} \in \{0, 1, 2\}^{n \times p}$, where n is the number of individuals and p is the number of SNPs. In the example depicted in Figure 3.1, each individual belongs to one of four populations A, B, C, or D. There is a single ancestral population at v_{root}, which experiences drift event $\boldsymbol{\alpha}$ before splitting into two populations. The first experiences drift event $\boldsymbol{\beta}_{AB}$ and then splits into populations A and B (each of which experiences further drift); the latter is subject to drift event $\boldsymbol{\beta}_{CD}$ before splitting into populations C and D.

I assume that ancestral alleles have been coded as zeroes and derived alleles as ones across all genes, which implies that the allele frequencies for the ancestral population at v_{root} are zero. The population means are then:

$$\boldsymbol{\mu}_A = \boldsymbol{\alpha} + \boldsymbol{\beta}_{AB} + \boldsymbol{\gamma}_A \tag{3.2}$$

$$\boldsymbol{\mu}_B = \boldsymbol{\alpha} + \boldsymbol{\beta}_{AB} + \boldsymbol{\gamma}_B \tag{3.3}$$

$$\boldsymbol{\mu}_C = \boldsymbol{\alpha} + \boldsymbol{\beta}_{CD} + \boldsymbol{\gamma}_C \tag{3.4}$$

$$\boldsymbol{\mu}_D = \boldsymbol{\alpha} + \boldsymbol{\beta}_{CD} + \boldsymbol{\gamma}_D, \tag{3.5}$$

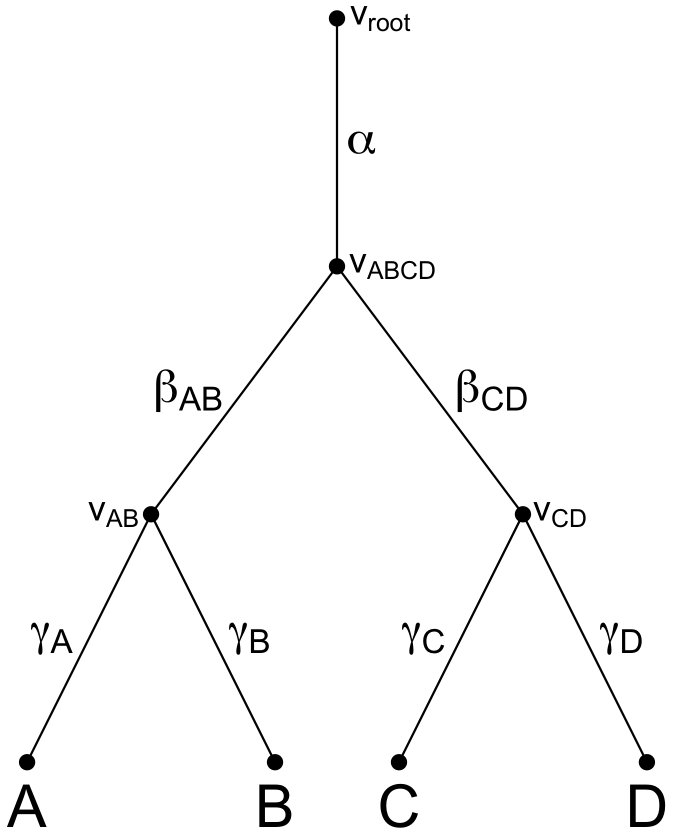


Figure 3.1: A four-population tree. Greek letters denote branches or "drift events"; vertices correspond to "divergence events"; and capitalized Roman letters denote populations.

where $\boldsymbol{\mu}_A$, $\boldsymbol{\mu}_B$, $\boldsymbol{\mu}_C$, and $\boldsymbol{\mu}_D$ are all vectors with values on $[0,2]^p$. Conditional on population means and population memberships, the data is distributed

$$x_{ij} \mid \text{Pop}(i) = k \sim \text{Binomial}\left(2,\ \frac{\mu_{kj}}{2}\right), \tag{3.6}$$

where $\text{Pop}(i) = k$ is shorthand notation expressing that individual i belongs to population k.

Since drift events are assumed to be mutually independent, $\mathbb{E}\left(\mathbf{x}_i^T \mathbf{x}_{i'}\right)$ is a function of $\text{Pop}(i)$ and $\text{Pop}(i')$. For concreteness, let $\boldsymbol{\alpha} \sim \mathcal{N}\left(\nu_\alpha \mathbf{1}_p, \sigma_\alpha^2 \mathbf{I}_p\right)$, $\boldsymbol{\beta}_{AB} \sim \mathcal{N}\left(\mathbf{0}, \sigma_{\beta_{AB}}^2 \mathbf{I}_p\right)$, $\boldsymbol{\gamma}_A \sim \mathcal{N}\left(\mathbf{0}, \sigma_{\gamma_A}^2 \mathbf{I}_p\right)$, and so on. (In my example, population means are constrained to lie between 0 and 2, but if the variances of the drift events are small, then normal distributions might provide reasonable approximations to the drift process. In any case, normality is not a crucial assumption in most of what follows.) Then, if $\text{Pop}(i) = A$:

$$\frac{1}{p}\mathbb{E}\left(\mathbf{x}_i^T \mathbf{x}_{i'}\right) = \begin{cases} \nu_\alpha^2 + \sigma_\alpha^2 + \sigma_{\beta_{AB}}^2 + \sigma_{\gamma_A}^2, & \text{Pop}(i') = A \text{ and } i \neq i' \\ \nu_\alpha^2 + \sigma_\alpha^2 + \sigma_{\beta_{AB}}^2, & \text{Pop}(i') = B \\ \nu_\alpha^2 + \sigma_\alpha^2, & \text{Pop}(i') \in \{C, D\} \end{cases} \tag{3.7}$$

In general, the covariance between two individuals is the sum of the variances of the drift events that were jointly experienced by their respective populations. More recent divergence leads to higher covariance (and thus correlation), as one would expect.

3.2.2 Population Factorization

Tree-structured data can be written as the sum of a low-rank matrix and a matrix of "error" terms:

$$\mathbf{X} = \mathbf{L}\mathbf{F}^T + \mathbf{E}. \tag{3.8}$$

Here, the matrix $\mathbf{LF}^T$ is the matrix of population means, which is rank-K under the extremely mild assumption that population means are linearly independent, and the error distributions are assumed to have mean zero. In the above example,

$$x_{ij} \sim \text{Binomial}\left(2,\ \frac{1}{2}\sum_{k=1}^{K}\ell_{ik}f_{jk}\right). \tag{3.9}$$

Of course, the factorization $\mathbf{LF}^T$ is not unique. Perhaps the simplest factorization is what I will refer to as the "population factorization," in which $\mathbf{L}$ captures population membership and $\mathbf{F}$ contains population allele frequencies. Specifically, each column $\boldsymbol{\ell}_k$ is a binary vector with $\ell_{ik} = 1$ if individual i belongs to population k and $\ell_{ik} = 0$ otherwise; allele frequencies for population k are given by column $\mathbf{f}_k$. For notational convenience, arrange the individuals by population, so that $\text{Pop}(i) = A$ for $i \in \{1, \ldots, n_A\}$, $\text{Pop}(i) = B$ for $i \in \{n_A+1, \ldots, n_A+n_B\}$, and so on. Then the four-population tree depicted in Figure 3.1 can be factorized:

$$\begin{bmatrix} \mathbf{1}_{n_A} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{n_B} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_C} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_D} \end{bmatrix} \begin{bmatrix} | & | & | & | \\ \boldsymbol{\mu}_A & \boldsymbol{\mu}_B & \boldsymbol{\mu}_C & \boldsymbol{\mu}_D \\ | & | & | & | \end{bmatrix}^T, \tag{3.10}$$

where $\boldsymbol{\mu}_A, \ldots, \boldsymbol{\mu}_D$ are given in Equations 3.2-3.5 and the bold-faced ones and zeroes in the loadings matrix are all-ones or all-zeroes vectors of appropriate size (I omit subscripts from the zero vectors to reduce clutter).

The population factorization loadings matrix $\mathbf{L}$ reveals population membership, but conveys no information about relationships among populations. Given the same population memberships, any tree with four leaves will yield an identical loadings matrix. For example, the tree depicted in Figure 3.1 cannot be distinguished from the four-population "star" shown

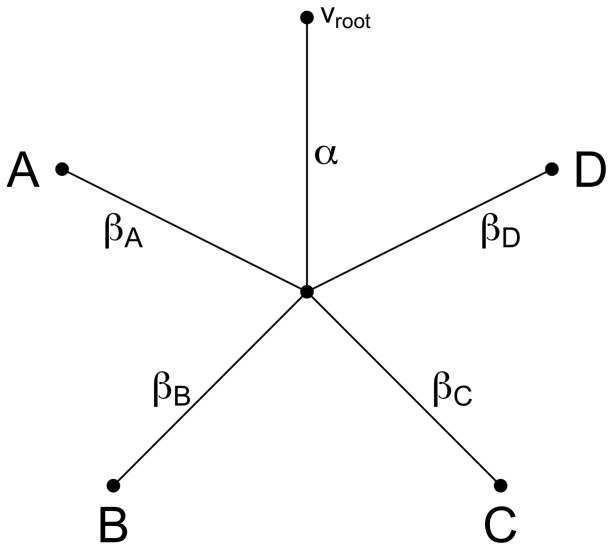


Figure 3.2: A four-population star.

in Figure 3.2 (which I will use as a "null" case in the simulation studies below: see Section 3.4.1). Since I'm interested precisely in factorizations that can capture tree-like structure, the population factorization will not be further considered.

3.2.3 Drift Factorization

A factorization that assigns a factor to drift events rather than populations provides much more information about population structure. In this factorization, which I call a "drift factorization," the matrix of loadings is binary, with $\ell_{i\kappa} = 1$ if individual i experienced drift event κ and $\ell_{i\kappa} = 0$ otherwise. For example, the drift factorization for the four-population

tree depicted in Figure 3.1 is:

$$\begin{bmatrix} \mathbf{1}_{n_A} & \mathbf{1}_{n_A} & \mathbf{0} & \mathbf{1}_{n_A} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}_{n_B} & \mathbf{1}_{n_B} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_B} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}_{n_C} & \mathbf{0} & \mathbf{1}_{n_C} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_C} & \mathbf{0} \\ \mathbf{1}_{n_D} & \mathbf{0} & \mathbf{1}_{n_D} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_D} \end{bmatrix} \begin{bmatrix} | & | & | & | & | & | & | \\ \boldsymbol{\alpha} & \boldsymbol{\beta}_{AB} & \boldsymbol{\beta}_{CD} & \boldsymbol{\gamma}_A & \boldsymbol{\gamma}_B & \boldsymbol{\gamma}_C & \boldsymbol{\gamma}_D \\ | & | & | & | & | & | & | \end{bmatrix}^T . \tag{3.11}$$

Given a drift factorization, it would be straightforward to reconstruct the population tree. I omit details, but the idea is to partially order columns $\boldsymbol{\ell}_\kappa$ using the usual partial order $\preccurlyeq$ on $\mathbb{R}^n$ (in Equation 3.11, for example, $\boldsymbol{\ell}_4 \preccurlyeq \boldsymbol{\ell}_2 \preccurlyeq \boldsymbol{\ell}_1$): any largest ordered set corresponds to a path from a leaf (or equivalently, a population) to the root.

I want to emphasize, however, that my primary goal in using matrix factorization methods is not to find trees. On the one hand, there are simple, widely used methods for constructing trees (for example, hierarchical clustering) and it's not easy to see how they would be improved upon by matrix factorization methods. But on the other, a tree-like structure is an idealized representation that fails to adequately explain much real-world data.

In particular, I want to be able to find admixtures, both at the population level (results of "admixture events") and at the individual level ("admixed individuals"). I include Figure 3.3 as an example: here, the dashed arrows indicate an event in which individuals from populations B and C combine in proportions π and $1 - \pi$ to produce a new, admixed population BC with population mean

$$\boldsymbol{\mu}_{BC} = \pi \left(\boldsymbol{\alpha} + \boldsymbol{\beta}_{AB} + \boldsymbol{\gamma}_B \right) + (1 - \pi) \left(\boldsymbol{\alpha} + \boldsymbol{\beta}_{CD} + \boldsymbol{\gamma}_C \right). \tag{3.12}$$

(The idea is the same at the individual level: in the example under consideration, let BC be a population of one and set $\pi = 0.5$.)

The loadings matrix $\mathbf{L}$ in the drift factorization for the admixture scenario, again assum-

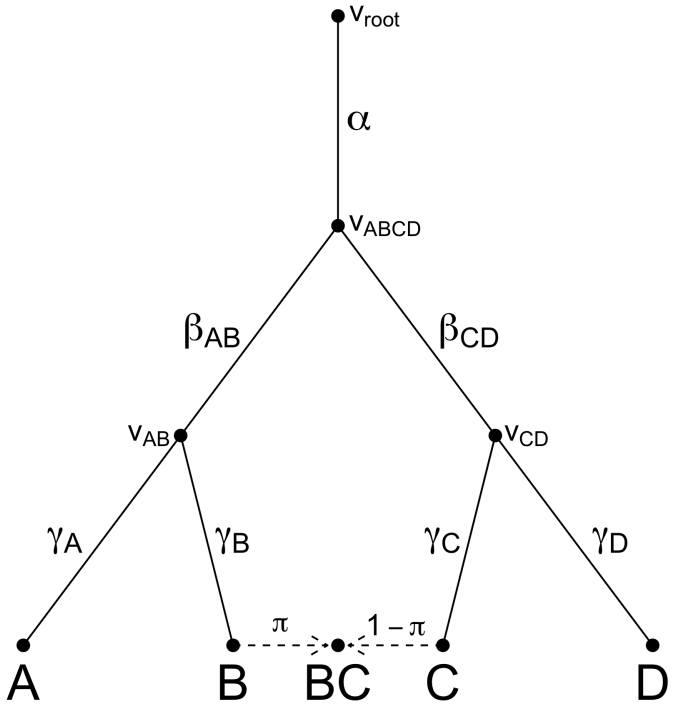


Figure 3.3: A four-population tree with admixture. Population BC results from the admixture of populations B and C, with admixture proportions π and $1 - \pi$.

ing that the individuals have been arranged by population, is

$$\mathbf{L} = \begin{bmatrix} \mathbf{1}_{n_A} & \mathbf{1}_{n_A} & \mathbf{0} & \mathbf{1}_{n_A} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}_{n_B} & \mathbf{1}_{n_B} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_B} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}_{n_{BC}} & \pi\mathbf{1}_{n_{BC}} & (1-\pi)\mathbf{1}_{n_{BC}} & \mathbf{0} & \pi\mathbf{1}_{n_{BC}} & (1-\pi)\mathbf{1}_{n_{BC}} & \mathbf{0} \\ \mathbf{1}_{n_C} & \mathbf{0} & \mathbf{1}_{n_C} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_C} & \mathbf{0} \\ \mathbf{1}_{n_D} & \mathbf{0} & \mathbf{1}_{n_D} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}_{n_D} \end{bmatrix}, \tag{3.13}$$

with the factor matrix $\mathbf{F}$ as in Equation 3.11. Even though the topology has been made slightly more complicated, the loadings remain immediately interpretable: for example, column $\boldsymbol{\ell}_2$ implies that a proportion π of the ancestry of population BC experienced a drift event in common with populations A and B; column $\boldsymbol{\ell}_3$ implies that a proportion $1 - \pi$ experienced a drift event in common with C and D; and so on.

3.2.4 Divergence Factorization

The drift factorization is immediately interpretable without recourse to a reconstructed topology (tree-like or otherwise), and unlike the population factorization, it conveys information about both population membership and population structure. However, even when the data is strictly tree-structured, it can be very difficult to find without prior knowledge about the population structure. It's notoriously difficult to perform matrix factorization with integer constraints, and moreover, requiring that $\mathbf{L} \in \{0, 1\}^{n \times p}$ is not sufficient to yield a drift factorization. (In particular, it must be true that for every pair of columns $\boldsymbol{\ell}_\kappa$ and $\boldsymbol{\ell}_{\kappa'}$, at least one of the following holds: $\boldsymbol{\ell}_\kappa^T \boldsymbol{\ell}_{\kappa'} = 0$; $\boldsymbol{\ell}_\kappa \preccurlyeq \boldsymbol{\ell}_{\kappa'}$; or $\boldsymbol{\ell}_{\kappa'} \preccurlyeq \boldsymbol{\ell}_\kappa$.) Further, $\mathbf{L}$ is not full-rank: while there are $2K$ - 1 factors (one for each edge in the population tree), $\mathbf{L}$ is rank-K. Thus, even if the usual tricks are employed to find an approximate drift factorization — for example, relaxing the integer constraints to $\mathbf{L} \in [0, 1]^{n \times p}$ or $\mathbf{L} \in \mathbb{R}_+^{n \times p}$ — the difficulty of

searching for a very specific overcomplete basis remains.

To simplify the optimization problem while still yielding interpretable information about population structure, I propose a "divergence factorization," which takes divergence events rather than drift events as its organizing principle. The interpretation is slightly more complicated, but the problem turns out to be much more tractable.

Each of the K factors in a divergence factorization corresponds to a vertex other than a leaf — either the root or a divergence event. Given any one of these vertices, one can form a sub-tree by taking the vertex along with its descendants. A divergence event can be interpreted as splitting this sub-tree in two by taking the two immediate descendants of the vertex along with their respective descendants. I refer to the latter sub-trees as the "left" and "right" descendant trees (the distinction between the two is, of course, arbitrary). In the exceptional case of the root, there is only one descendant tree, which I designate a "left" sub-tree.

In a divergence factorization, the loadings ℓ_{ik} can take three values depending on the population to which individual i belongs: a positive constant if the leaf corresponding to $\mathrm{Pop}(i)$ is included in the left descendant tree; a negative constant if it is included in the right; and zero if it is included in neither. For the example depicted in Figure 3.1, the matrix of loadings $\mathbf{L}$ is of the form:

$$\mathbf{L} = \begin{bmatrix} \lambda_1 \mathbf{1}_{n_A} & \lambda_2 \mathbf{1}_{n_A} & \lambda_3 \mathbf{1}_{n_A} & \mathbf{0} \\ \lambda_1 \mathbf{1}_{n_B} & \lambda_2 \mathbf{1}_{n_B} & -\nu_3 \mathbf{1}_{n_B} & \mathbf{0} \\ \lambda_1 \mathbf{1}_{n_C} & -\nu_2 \mathbf{1}_{n_C} & \mathbf{0} & \lambda_4 \mathbf{1}_{n_C} \\ \lambda_1 \mathbf{1}_{n_D} & -\nu_2 \mathbf{1}_{n_D} & \mathbf{0} & -\nu_4 \mathbf{1}_{n_D} \end{bmatrix}, \tag{3.14}$$

where $\lambda_1, \ldots, \lambda_4$ and $\nu_2, \ldots, \nu_4$ are all positive scalars. For reasons that will become apparent in Section 3.2.6, I also require the columns of $\mathbf{F}$ to be mutually independent.

While the constraints appear to be more complicated than for the drift factorization, the relaxation of the problem is potentially more convenient since one can simply search over $\mathbf{L} \in \mathbb{R}^{n \times K}$ with, in particular, no nonnegative constraints on the values ℓ_{ik}. More importantly, though, the matrices $\mathbf{L}$ and $\mathbf{F}$ are full-rank, so that the problem no longer requires searching for a specific overcomplete basis.

The divergence factorization remains immediately interpretable for non-admixed populations. Individuals that have a nonzero loading for some column $\boldsymbol{\ell}_k$ can be interpreted as having jointly experienced some divergence event (and thus, the drift event leading up to the divergence event), with positively loaded individuals branching off in one direction and negatively loaded individuals in another.

Admixtures can also be readily identified; however, their interpretation becomes more difficult. Consider the matrix of loadings $\mathbf{L}$ for the admixture example depicted in Figure 3.3:

$$\mathbf{L} = \begin{bmatrix} \lambda_1 \mathbf{1}_{n_A} & \lambda_2 \mathbf{1}_{n_A} & \lambda_3 \mathbf{1}_{n_A} & \mathbf{0} \\ \lambda_1 \mathbf{1}_{n_B} & \lambda_2 \mathbf{1}_{n_B} & -\nu_3 \mathbf{1}_{n_B} & \mathbf{0} \\ \lambda_1 \mathbf{1}_{n_{BC}} & (\pi\lambda_2 - (1-\pi)\nu_2)\mathbf{1}_{n_{BC}} & -\pi\nu_3 \mathbf{1}_{n_{BC}} & (1-\pi)\lambda_4 \mathbf{1}_{n_{BC}} \\ \lambda_1 \mathbf{1}_{n_C} & -\nu_2 \mathbf{1}_{n_C} & \mathbf{0} & \lambda_4 \mathbf{1}_{n_C} \\ \lambda_1 \mathbf{1}_{n_D} & -\nu_2 \mathbf{1}_{n_D} & \mathbf{0} & -\nu_4 \mathbf{1}_{n_D} \end{bmatrix}. \tag{3.15}$$

Now, when

$$\pi = \frac{\nu_2}{\lambda_2 + \nu_2}, \tag{3.16}$$

then

$$\boldsymbol{\ell}_2 = \begin{bmatrix} \lambda_2 \mathbf{1}_{n_A} \\ \lambda_2 \mathbf{1}_{n_B} \\ \mathbf{0}_{n_{BC}} \\ -\nu_2 \mathbf{1}_{n_C} \\ -\nu_2 \mathbf{1}_{n_D} \end{bmatrix}, \tag{3.17}$$

so that $\boldsymbol{\ell}_2$ alone cannot distinguish between the case where population BC did not experience the corresponding divergence event and the case where it did in fact experience it, but a proportion of the ancestry (namely, the proportion given by Equation 3.16) went in one direction and the rest in the other.

In effect, this difficulty of interpretation persists for all loadings ℓ_{ik} such that $-\nu_k < \ell_{ik} < \lambda_k$. If π_{div} is the proportion of ancestry of individual i that experienced divergence event k and π_{left} is the proportion of that proportion that descended along the left branch, then

$$\ell_{ik} = \pi_{\text{div}} \left(\pi_{\text{left}} \lambda_k - (1 - \pi_{\text{left}}) \nu_k \right). \tag{3.18}$$

From this loading alone, all that can be concluded is that

$$1 \geq \pi_{\text{div}} \geq \begin{cases} \frac{\ell_{ik}}{\lambda_k}, & \ell_{ik} \geq 0 \\ \frac{|\ell_{ik}|}{\nu_k}, & \ell_{ik} < 0 \end{cases} \tag{3.19}$$

and

$$1 \geq \pi_{\text{left}} \geq \frac{\ell_{ik} + \nu_k}{\lambda_k + \nu_k}. \tag{3.20}$$

In particular, π_{div} and π_{left} remain unidentifiable. For this reason, it might be useful to use a divergence factorization (which is typically easier to obtain) as an initialization to a method that yields a drift factorization (which is less ambiguous). Due to the difficulty of obtaining

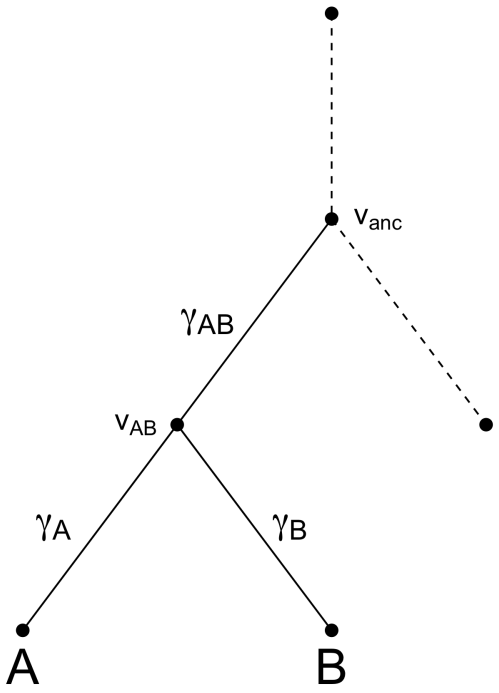
drift factorizations, however, I will reserve a comprehensive examination of this strategy for future work.

3.2.5 Existence of the Divergence Factorization

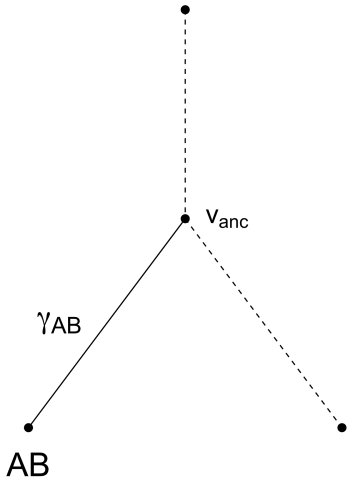
It's clear that the column space of the drift factorization is identical to that of the divergence factorization (indeed, both are identical to the column space of the population factorization). However, since I also require the factors of the divergence factorization (the columns of $\mathbf{F}$) to be mutually independent, it remains to be shown that a divergence factorization exists. The following theorem asserts that it does.

Theorem 3.1 (Existence of divergence factorization)**.** *For any tree-structured dataset* $\mathbf{X} \in \mathbb{R}^{n\times p}$ *in which* K *populations are represented, a divergence factorization exists. That is, letting* $\mathbf{Y}$ *be the rank-*K *matrix of population means, one can find* $\mathbf{L} \in \mathbb{R}^{n\times K}$ *and* $\mathbf{F} \in \mathbb{R}^{p\times K}$ *such that* $\mathbf{Y} = \mathbf{L}\mathbf{F}^T$*, the columns of* $\mathbf{F}$ *are mutually uncorrelated linear combinations of drift events, and each column* $\boldsymbol{\ell}_k$ *of* $\mathbf{L}$ *corresponds to a non-leaf vertex of the tree in that it takes a maximum of three values depending on the location of the leaf corresponding to* $Pop(i)$*:* $\ell_{ik} = \lambda_k \geq 0$ *if it is in the left descendent tree,* $\ell_{ik} = -\nu_k \leq 0$ *if it is in the right descendent tree, and* $\ell_{ik} = 0$ *if it is in neither.*

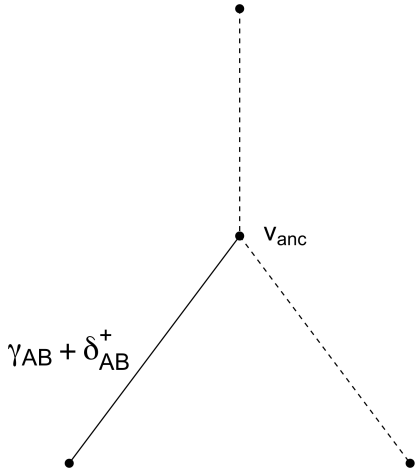
Proof. The proof proceeds by induction. For $K = 1$, the drift factorization is also a divergence factorization. So assume that the theorem holds for $k < K$. Let the population means be structured according to the tree $\mathcal{T}_K$. There must be at least one pair of leaves A and B that share a neighboring vertex (i.e., an immediate ancestor) v_{AB}: let $\boldsymbol{\gamma}_A$ be the drift event from v_{AB} to A and let $\boldsymbol{\gamma}_B$ be the drift event from v_{AB} to B. Removing edges $\boldsymbol{\gamma}_A$ and $\boldsymbol{\gamma}_B$ and re-labeling v_{AB} as the proto-population AB yields a population tree $\mathcal{T}_{K-1}$ with $K-1$ populations. Now let v_{anc} be the neighboring vertex (the immediate ancestor) of v_{AB} in $\mathcal{T}_{K-1}$ and let $\boldsymbol{\gamma}_{AB}$ be the drift event from v_{anc} to v_{AB}. (For an illustration of the proof setup, see Figure 3.4.)



(a) Tree $\mathcal{T}_K$.



(b) Tree $\mathcal{T}_{K-1}$.



(c) Tree $\tilde{\mathcal{T}}_{K-1}$.

Figure 3.4: Setup for the proof of the existence of the divergence factorization. See text for details.

If necessary, arrange the individuals so that $\text{Pop}(i) = A$ for $i \in \{1, \ldots, n_A\}$ and $\text{Pop}(i) = B$ for $i \in \{n_A + 1, \ldots, n_A + n_B\}$. Write:

$$\mathbf{Y}_K = \mathbf{Y}_{K-1} + \begin{bmatrix} \mathbf{1}_{n_A} & \mathbf{0}_{n_A} \\ \mathbf{0}_{n_B} & \mathbf{1}_{n_B} \\ \mathbf{0}_{n-n_A-n_B} & \mathbf{0}_{n-n_A-n_B} \end{bmatrix} \begin{bmatrix} | & | \\ \boldsymbol{\gamma}_A & \boldsymbol{\gamma}_B \\ | & | \end{bmatrix}^T \tag{3.21}$$

$$= \mathbf{Y}_{K-1} + \begin{bmatrix} \mathbf{1}_{n_A} & \frac{\mathbb{E}(\boldsymbol{\gamma}_A^T \boldsymbol{\gamma}_A)}{p} \mathbf{1}_{n_A} \\ \mathbf{1}_{n_B} & -\frac{\mathbb{E}(\boldsymbol{\gamma}_B^T \boldsymbol{\gamma}_B)}{p} \mathbf{1}_{n_B} \\ \mathbf{0}_{n-n_A-n_B} & \mathbf{0}_{n-n_A-n_B} \end{bmatrix} \begin{bmatrix} | & | \\ \boldsymbol{\delta}_{AB}^+ & \boldsymbol{\delta}_{AB}^- \\ | & | \end{bmatrix}^T, \tag{3.22}$$

where

$$\boldsymbol{\delta}_{AB}^+ = \frac{\mathbb{E}(\boldsymbol{\gamma}_B^T \boldsymbol{\gamma}_B)}{\mathbb{E}(\boldsymbol{\gamma}_A^T \boldsymbol{\gamma}_A) + \mathbb{E}(\boldsymbol{\gamma}_B^T \boldsymbol{\gamma}_B)} \boldsymbol{\gamma}_A + \frac{\mathbb{E}(\boldsymbol{\gamma}_A^T \boldsymbol{\gamma}_A)}{\mathbb{E}(\boldsymbol{\gamma}_A^T \boldsymbol{\gamma}_A) + \mathbb{E}(\boldsymbol{\gamma}_B^T \boldsymbol{\gamma}_B)} \boldsymbol{\gamma}_B, \tag{3.23}$$

$$\boldsymbol{\delta}_{AB}^- = \frac{p}{\mathbb{E}(\boldsymbol{\gamma}_A^T \boldsymbol{\gamma}_A) + \mathbb{E}(\boldsymbol{\gamma}_B^T \boldsymbol{\gamma}_B)} (\boldsymbol{\gamma}_A - \boldsymbol{\gamma}_B). \tag{3.24}$$

It is straightforward to confirm that $\boldsymbol{\delta}_{AB}^+$ and $\boldsymbol{\delta}_{AB}^-$ are uncorrelated.

Next, notice that

$$\tilde{\mathbf{Y}}_{K-1} := \mathbf{Y}_{K-1} + \begin{bmatrix} \mathbf{1}_{n_A} \\ \mathbf{1}_{n_B} \\ \mathbf{0}_{n-n_A-n_B} \end{bmatrix} \begin{bmatrix} | \\ \boldsymbol{\delta}_{AB}^+ \\ | \end{bmatrix}^T \tag{3.25}$$

is the matrix of population means for the tree $\tilde{\mathcal{T}}_{K-1}$ that is formed by replacing the drift event $\boldsymbol{\gamma}_{AB}$ from v_{anc} to v_{AB} with the drift event $\boldsymbol{\gamma}_{AB} + \boldsymbol{\delta}_{AB}^+$. (Again see Figure 3.4.) By

the inductive hypothesis, $\tilde{\mathbf{Y}}_{K-1}$ has a divergence factorization, and since

$$\mathbf{Y}_K = \tilde{\mathbf{Y}}_{K-1} + \begin{bmatrix} \frac{\mathbb{E}(\boldsymbol{\gamma}_A^T \boldsymbol{\gamma}_A)}{p} \mathbf{1}_{n_A} \\ -\frac{\mathbb{E}(\boldsymbol{\gamma}_B^T \boldsymbol{\gamma}_B)}{p} \mathbf{1}_{n_B} \\ \mathbf{0}_{n-n_A-n_B} \end{bmatrix} \begin{bmatrix} | \\ \boldsymbol{\delta}_{AB}^- \\ | \end{bmatrix}^T, \tag{3.26}$$

the proof is complete.

□

3.2.6 Factorization of the Co-occurrence Matrix

In addition to their interpretability, a considerable advantage of the drift and divergence factorizations is that if the loadings matrix $\mathbf{L}$ is of primary interest (as is typically the case), then it can suffice to work with the $n \times n$ co-occurrence matrix $\frac{\mathbf{X}\mathbf{X}^T}{p}$. In the population genetics settings in which I'm especially interested, it's often the case that $n \ll p$ so that finding a factorization of $\frac{\mathbf{X}\mathbf{X}^T}{p}$ can be orders of magnitude faster than factorizing the full data matrix $\mathbf{X}$.

Indeed, since for both factorizations the columns of $\mathbf{F}$ are mutually uncorrelated with mean zero (with the possible exception of the first factor $\mathbf{f}_1$),

$$\mathbb{E}\left(\frac{\mathbf{X}\mathbf{X}^T}{p}\right) = \mathbf{L}\mathbf{D}\mathbf{L}^T + \tilde{\mathbf{D}}, \tag{3.27}$$

where $\mathbf{D}$ is the diagonal matrix with entries $d_{kk} = \mathbb{E}\left(\frac{\mathbf{f}_k^T \mathbf{f}_k}{p}\right)$ and $\tilde{\mathbf{D}}$ is the diagonal matrix with entries $\tilde{d}_{kk} = \mathbb{E}\left(\frac{\mathbf{e}_k^T \mathbf{e}_k}{p}\right)$. Thus any method that uses constraints or penalty terms to factorize $\mathbf{X} \approx \mathbf{L}\mathbf{F}^T$ can potentially be used in a similar fashion (that is, with similar constraints or penalty terms applied to $\mathbf{L}$) to obtain a factorization $\frac{\mathbf{X}\mathbf{X}^T}{p} \approx \mathbf{L}\mathbf{D}\mathbf{L}^T$ or $\frac{\mathbf{X}\mathbf{X}^T}{p} \approx \mathbf{L}\mathbf{D}\mathbf{L}^T + \tilde{\mathbf{D}}$.

This setup is a familiar one in the literature, where it usually goes by the name of factor analysis. Below, I propose various methods for finding a divergence factorization, and in Section 3.3.5, I consider how those methods can be adapted to factorize a co-occurrence matrix rather than a full data matrix. An accompanying example in Section 3.3.6 will yield useful insight into the differences in the approaches.

3.3 Methods

The following sections explore methods for finding divergence factorizations. In Section 3.3.1, I show that in favorable (but by no means overly idealized) cases, the SVD yields a factorization that is quite close to a divergence factorization. This suggests that, since scores of fast implementations exist for finding the SVD, a divergence factorization might most easily be found by "sparsifying" the SVD or "rotating" the loadings to obtain a sparser solution. Section 3.3.2 details such an approach by outlining several methods that are commonly used to find sparse matrix factorizations, as well as my preferred method, empirical Bayes matrix factorization (EBMF). In Section 3.3.3, I use the toy example of a "balanced tree" to compare these methods. As it turns out, many of the commonly used methods struggle to find the desired solution even in this very simple scenario. Sections 3.3.4 and 3.3.5 propose two approaches to EBMF that are more particularly adapted to tree-structured data: one approach for full data matrices and the other for co-occurrence matrices. Finally, Sections 3.3.6 and 3.3.7 test these approaches on slightly more challenging toy examples: respectively, "unbalanced trees" and trees with admixture. The latter require another modification in order to not force admixed populations to conform to a strict tree-like structure. This last modification yields the method that I prefer to use to fit real data, both because it does reasonably well at finding an accurate representation of the true underlying tree and because it's flexible enough to handle admixtures. I summarize this method, which I refer to as "tree-EBMF," in Section 3.3.8.

3.3.1 SVD of Tree-Structured Data

In many typical scenarios, the divergence factorization of a tree-structured dataset is similar to the singular value decomposition (SVD). I use the compact representation

$$\mathbf{Y} = \mathbf{U}\mathbf{D}\mathbf{V}^T, \tag{3.28}$$

where $\mathbf{Y} \in \mathbb{R}^{n \times p}$ is rank-K, the columns of both $\mathbf{U} \in \mathbb{R}^{n \times K}$ and $\mathbf{V} \in \mathbb{R}^{p \times K}$ are mutually orthogonal with unit norm, and $\mathbf{D} \in \mathbb{R}^{K \times K}$ is a diagonal matrix whose diagonal entries $d_1 \geq d_2 \geq \ldots \geq d_K > 0$ are the non-zero singular values of $\mathbf{Y}$. In the case where $\mathbf{Y}$ is symmetric (for example, when $\mathbf{Y}$ is a covariance matrix), then $\mathbf{U} = \mathbf{V}$. Also note that if there are any repeated singular values — that is, if $d_k = d_{k'}$ for some $k \neq k'$ — then the SVD is not unique, since $\mathbf{u}_k$ and $\mathbf{u}_{k'}$ can be replaced by any two orthogonal unit vectors spanning the same column space (with, of course, the inverse rotation applied to $\mathbf{v}_k$ and $\mathbf{v}_{k'}$).

In very special cases, the divergence factorization is exactly the SVD of the matrix of population means $\mathbf{Y} = \mathbf{L}\mathbf{F}^T$ (after scaling and, if necessary, permuting the columns of the divergence factorization). To explore this possibility, I introduce the notion of a "balanced tree," where all leaves are at the same depth, all drift events at a given depth have equal variance, and each population is of identical size. In Figure 3.1, for example, a balanced tree is obtained by setting $\sigma^2_{\beta_{AB}} = \sigma^2_{\beta_{CD}}$, $\sigma^2_{\gamma_A} = \sigma^2_{\gamma_B} = \sigma^2_{\gamma_C} = \sigma^2_{\gamma_D}$, and $n_A = n_B = n_C = n_D$. I omit details, but it can be inferred from the proof of the existence of the divergence factorization (Section 3.2.5) that a balanced tree will yield a divergence factorization that is also an SVD of $\mathbf{Y}$ in the limit $p \to \infty$ (so that the values of $\frac{1}{p}\left(\mathbf{f}_k^T \mathbf{f}_{k'}\right)$ converge to their expectations).

Now, it is also the case that for balanced trees, all divergence events at the same depth will yield divergence factors of equal (expected) magnitude. Thus, if a balanced tree has

at least four populations, then there will be at least one pair of singular vectors that share a singular value, so the SVD will not be unique. It's more difficult to come by examples where the divergence factorization is (again after scaling and permuting columns) the unique SVD of $\mathbf{Y}$, since parameters must align in an unnatural way. For example, if in Figure 3.1 $n_A = n_B = n_C = n_D$, and if additionally $\sigma^2_{\gamma_A} = \sigma^2_{\gamma_B} < \sigma^2_{\gamma_C} = \sigma^2_{\gamma_D}$, then the divergence factorization is the (unique) SVD when

$$\sigma^2_{\beta_{AB}} - \sigma^2_{\beta_{CD}} = \frac{1}{2}\left(\sigma^2_{\gamma_C} - \sigma^2_{\gamma_A}\right). \tag{3.29}$$

Even in these unlikely scenarios, the additive errors in $\mathbf{E}$ and the finite-sample randomness in $\mathbf{F}$ will make vanishingly rare the case where the SVD of $\mathbf{X} = \mathbf{L}\mathbf{F}^T + \mathbf{E}$ yields a divergence factorization with only minimal post-processing. I find in general, however, that if a tree is not too unbalanced, then the SVD will be reasonably close to the divergence factorization, up to the non-identifiability caused by divergence events of similar magnitude: see Figure 3.5 for examples. This similarity suggests that the SVD should generally provide a good initialization for methods that are more specifically tailored to find a divergence factorization.

3.3.2 Sparse Matrix Factorization Methods

The discussion in Section 3.3.1 suggests that in more favorable cases where the tree is not too unbalanced, it might be possible to obtain a good approximation to the divergence factorization by, first, properly rotating the SVD loadings corresponding to singular values of similar magnitude and, second, "nudging" the loadings so that the resulting columns are as close to ternary as possible. In both cases, it is (in part) a question of "sparsifying" $\mathbf{U}$, either via a rotation to sparsity or via some prior or penalty term on the loadings u_{ik}.

I claim, however, that the most commonly used methods for finding sparse matrix factorizations often do not produce the desired results. I consider the following methods:

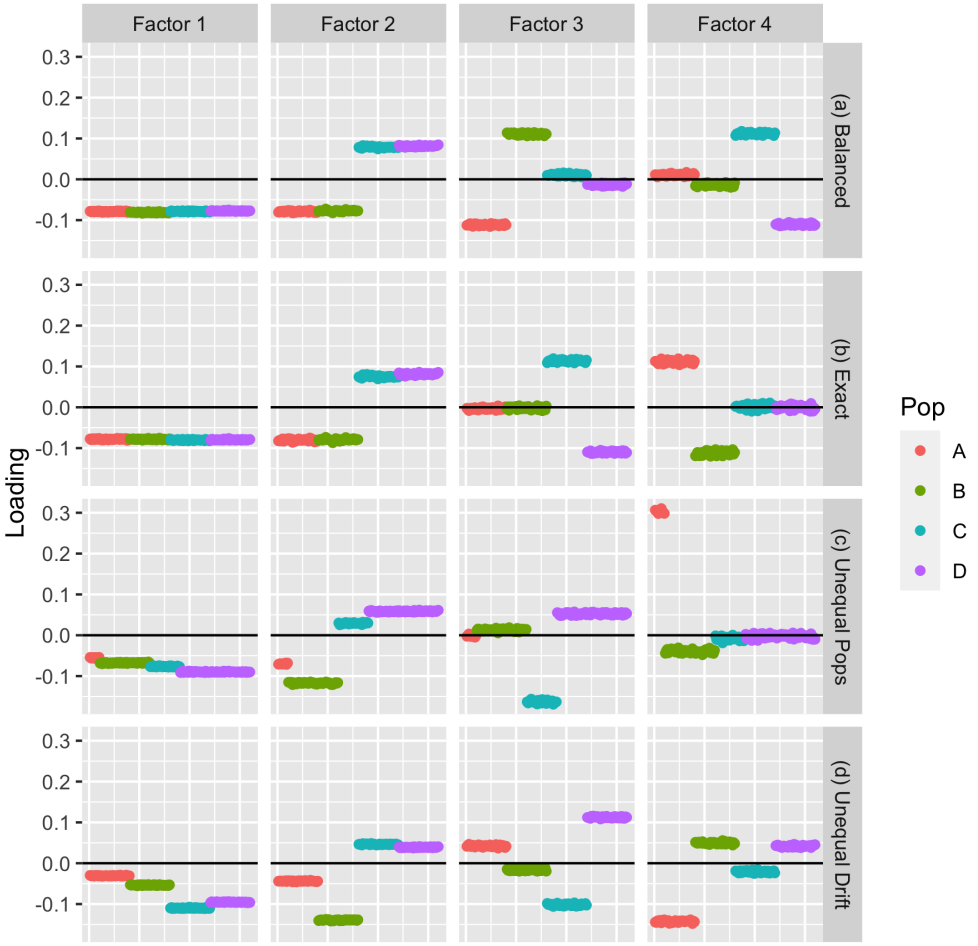


Figure 3.5: SVDs of tree-structured data. Each dataset has the structure depicted in Figure 3.1, with $\sigma^2_\epsilon = 1$ and $p = 1000$. Shown are loadings $\mathbf{u}_1, \ldots, \mathbf{u}_4$ obtained by applying SVD to four simulated datasets: (a) a "balanced" tree; (b) a tree for which the divergence factorization of $\mathbf{LF}^T$ is, in the limit $p \to \infty$, exactly the unique SVD; (c) a tree with unequal population sizes; (d) a tree where the variance differs among drift events. For tree (c), $n_A = 10$, $n_B = 50$, $n_C = 30$, and $n_D = 70$; for all others, $n_A = n_B = n_C = n_D = 40$. The variance of all drift events in trees (a) and (c) is $\sigma^2 = 4$; tree (b) puts $\sigma^2_\alpha = 4$, $\sigma^2_{\beta_{AB}} = 2.5$, $\sigma^2_{\beta_{CD}} = 1$, $\sigma^2_{\gamma_A} = \sigma^2_{\gamma_B} = 1$, and $\sigma^2_{\gamma_C} = \sigma^2_{\gamma_D} = 4$; and tree (d) puts $\sigma^2_\alpha = 4$, $\sigma^2_{\beta_{AB}} = 4$, $\sigma^2_{\beta_{CD}} = 16$, $\sigma^2_{\gamma_A} = 1$, $\sigma^2_{\gamma_B} = 16$, $\sigma^2_{\gamma_C} = 9$, and $\sigma^2_{\gamma_D} = 4$. Both drift events and errors are normally distributed.

- Varimax rotation (R **stats** function `varimax`). Given $\mathbf{U} \in \mathbb{R}^{n \times K}$, the varimax method attempts to find an orthonormal rotation $\mathbf{Q} \in \mathcal{O}(K)$ such that $\mathbf{UQ}$ is sparse. To do so, it maximizes the sum of the column-wise variances of the squared loadings. Formally, it sets

$$\mathbf{Q} = \text{argmax}_{\mathbf{R} \in \mathcal{O}(K)} \sum_{k=1}^{K} \frac{1}{n} \sum_{i=1}^{n} \left([\mathbf{UR}]_{ik}^4 - \frac{1}{n} \sum_{i=1}^{n} [\mathbf{UR}]_{ik}^2 \right). \tag{3.30}$$

  For a contemporary treatment of varimax rotations, see Rohe and Zeng [2020]. An important limitation of the varimax method is that it depends on how the columns of $\mathbf{U}$ are scaled. While the drift factorization yields a binary $\mathbf{L}$, so that the scale of the columns of $\mathbf{F}$ can be interpreted in terms of the variance of the corresponding drift events, it's not immediately obvious how one should scale the loadings of the divergence factorization. Given an SVD $\mathbf{X} = \mathbf{UDV}^T$, I apply the varimax method to the matrix $\mathbf{UD}^{1/2}$ on the grounds that the co-occurrence matrix has an SVD of form $\mathbf{UDU}^T$ — that is, "factors" and "loadings" are not distinguishable, so it makes sense to scale them identically.

- Sparse SVD (function `ssvd.iter.thresh` in R package **ssvd**). This method iteratively thresholds and orthogonalizes the columns of $\mathbf{U}$ and $\mathbf{V}$: for details, see Yang et al. [2014]. I initialize using the SVD rather than using the fast initialization method implemented by function `ssvd`, and I skip the thresholding of $\mathbf{V}$ (which is not expected to be sparse) by setting `gamma.v = 0`. Like the varimax method, the SSVD yields a matrix $\mathbf{U}$ with mutually orthogonal columns, so I don't expect it to do well on very asymmetric trees.

- Penalized Matrix Decomposition (function `PMD` in R package **PMA**). This method, proposed by Witten et al. [2009], finds an approximate SVD while enforcing sparsity via a ℓ_1 penalty on the entries of $\mathbf{U}$ and $\mathbf{V}$. The objective function for the rank-1 PMD

is

$$\|\mathbf{X} - \mathbf{u}d\mathbf{v}^T\|_2^2 + \nu_u\|\mathbf{u}\|_1 + \nu_v\|\mathbf{v}\|_1, \tag{3.31}$$

where ν_u and ν_v are sparsity parameters that are typically chosen using cross-validation (**PMA** provides function `PMD.cv` for this purpose). Again, since $\mathbf{V}$ is not expected to be sparse, I fix $\nu_v = 0$. A rank-K PMD must be constructed iteratively, with $\mathbf{u}_k$, d_k, and $\mathbf{v}_k$ obtained by performing a rank-1 PMD on the matrix of residuals

$$\mathbf{X} - \mathbf{U}_{k-1}\mathbf{D}_{k-1}\mathbf{V}_{k-1}^T, \tag{3.32}$$

where the columns of $\mathbf{U}_{k-1}$ are $\mathbf{u}_1, \ldots, \mathbf{u}_{k-1}$, and where $\mathbf{D}_{k-1}$ and $\mathbf{V}_{k-1}$ are similarly defined. Because cross-validation is required for PMD to be effective, it is considerably slower than the other methods.

- Empirical Bayes matrix factorization (function `flash` in `R` package **flashier**). The EBMF model is:

$$\mathbf{X} = \mathbf{L}\mathbf{F}^T + \mathbf{E} \tag{3.33}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}_\ell^{(k)},\ k = 1, \ldots, K \tag{3.34}$$

$$f_{jk} \sim g_f^{(k)} \in \mathcal{G}_f^{(k)},\ k = 1, \ldots, K \tag{3.35}$$

$$e_{ij} \sim \mathcal{N}\left(0, \sigma_{ij}^2\right), \tag{3.36}$$

where $\mathcal{G}_\ell^{(k)}$ and $\mathcal{G}_f^{(k)}$ are "prior families" that are chosen in advance. (As the notation suggests, a different prior family can be chosen for each column of $\mathbf{L}$ and for each column of $\mathbf{F}$.) The priors $g_\ell^{(1)}, \ldots, g_\ell^{(K)},\ g_f^{(1)}, \ldots, g_f^{(K)}$ are estimated by maximizing a variational approximation to the log likelihood. While several methods can be used to estimate the variance parameters σ_{ij}^2, the default is to assume that all entries y_{ij} share a common variance parameter σ^2 and to estimate σ^2 via maximum likelihood.

Throughout, I fix each $\mathcal{G}_f^{(k)}$ as the family of zero-mean normal distributions $\{g : g \sim \mathcal{N}(0, \sigma^2),\ \sigma^2 \geq 0\}$, but I will consider various choices of prior family for $\mathcal{G}_\ell^{(k)}$.

3.3.3 Example: Balanced Tree

I first consider an "easy" example where the SVD is already very nearly a divergence factorization. Specifically, I simulate data from a balanced four-population tree (see Figure 3.1) with all population sizes $n_A, \ldots, n_D = 40$, $p = 1000$, all drift event variances $\sigma_\alpha^2, \ldots, \sigma_{\gamma_D}^2 = 2^2$, and the error variance $\sigma_\epsilon^2 = 1$. (These parameter settings are identical to the ones used in example (a) in Figure 3.5.) Both drift events and errors are normally distributed (of course, this is in not the case with genotype data, where "errors" are binomially distributed). I fit EBMF using two spike-and-slab prior families: the family of point-normal distributions

$$\left\{g : g \sim \pi\delta_0 + (1-\pi)\mathcal{N}(0, \sigma^2),\ 0 \leq \pi \leq 1,\ \sigma^2 \geq 0\right\} \tag{3.37}$$

and the family of point-Laplace distributions

$$\left\{g : g \sim \pi\delta_0 + (1-\pi)\text{Laplace}(\lambda),\ 0 \leq \pi \leq 1,\ \lambda \geq 0\right\}. \tag{3.38}$$

Figure 3.6 shows results for a single simulation. To evaluate factorizations, I calculate the "crossproduct similarity"

$$\min_{\Pi \in \mathrm{S}_K} \frac{1}{K} \sum_{k=1}^{K} \frac{|\boldsymbol{\ell}_{\Pi(k)}^T \tilde{\boldsymbol{\ell}}_k|}{\|\boldsymbol{\ell}_{\Pi(k)}\|_2 \|\tilde{\boldsymbol{\ell}}_k\|_2}, \tag{3.39}$$

where S_K is the symmetric group of permutations on $\{1, \ldots, K\}$, $\tilde{\boldsymbol{\ell}}_k$ is the kth loadings column of the desired factorization, and $\boldsymbol{\ell}_k$ is the kth loadings column of the obtained factorization. Note that this value does not depend on how the columns $\boldsymbol{\ell}_k$ are ordered or scaled. Also note that in the special case of balanced trees, $\tilde{\mathbf{L}}$ is readily available (up to

right multiplication by a diagonal matrix), as it follows from the proof of the existence of the divergence factorization (Section 3.2.5) that for balanced trees, $\lambda_k = \nu_k$ for all k. The crossproduct similarity is given in the right margins of Figure 3.6 and is calculated over multiple balanced tree simulations in Figure 3.7.

In this idealized scenario, only EBMF with point-Laplace priors regularly recovers the divergence factorization. Sparse SVD occasionally recovers it as well, but it often doesn't return results that are substantially different from the SVD. Indeed, if the SVD is not sufficiently sparse to begin with, then `ssvd` won't perform any thresholding and the iterates won't budge from the initial value. Cross-validated PMD suffers from the opposite problem, in that it over-sparsifies the second factor and thus yields results that are inferior to the SVD.

The other three methods are clearly unsatisfactory:

- Without cross-validation (and using default parameter settings), PMD does exceptionally poorly.

- The varimax rotation tends to find a factorization approaching the population factorization. In effect, this is the sparsest possible factorization, so that the method is doing exactly what it is intended to do. Some information about the tree structure is retained — for example, it's clear from the first factor that Population B is more closely related to Population A than Populations C or D — but the tree is much more difficult to reconstruct from the varimax results than from, for example, the SVD. Recall also from Section 3.3.2 that varimax results will differ depending on how one decides to scale loadings, which further detracts from the appeal of the method.

- With point-normal priors on loadings, EBMF generally returns loadings that are identical to the SVD. This is a consequence of the coordinate-wise descent algorithm used by **flashier**, where the columns $\boldsymbol{\ell}_k$ are $\mathbf{f}_k$ are sequentially updated by solving an empirical Bayes normal means (EBNM) problem. If the point-normal prior is estimated as

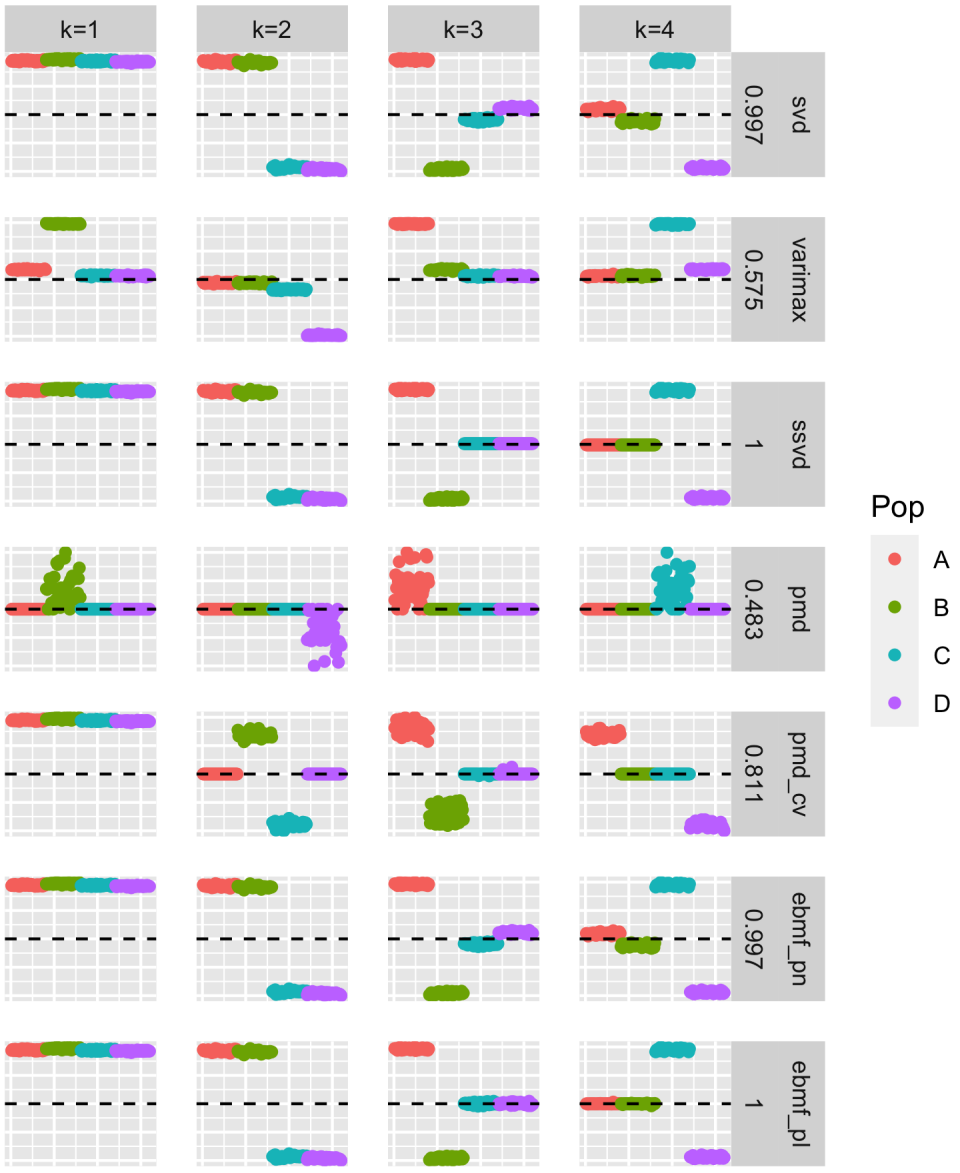


Figure 3.6: Comparison of sparse factorization methods for a balanced tree (example (a) in Figure 3.5). The methods considered are varimax rotation, sparse SVD, PMD without and with CV, and EBMF with point-normal and point-Laplace priors. The "crossproduct similarity" from each factorization to the truth (see text for a definition) is given in the right margins beside the method names.

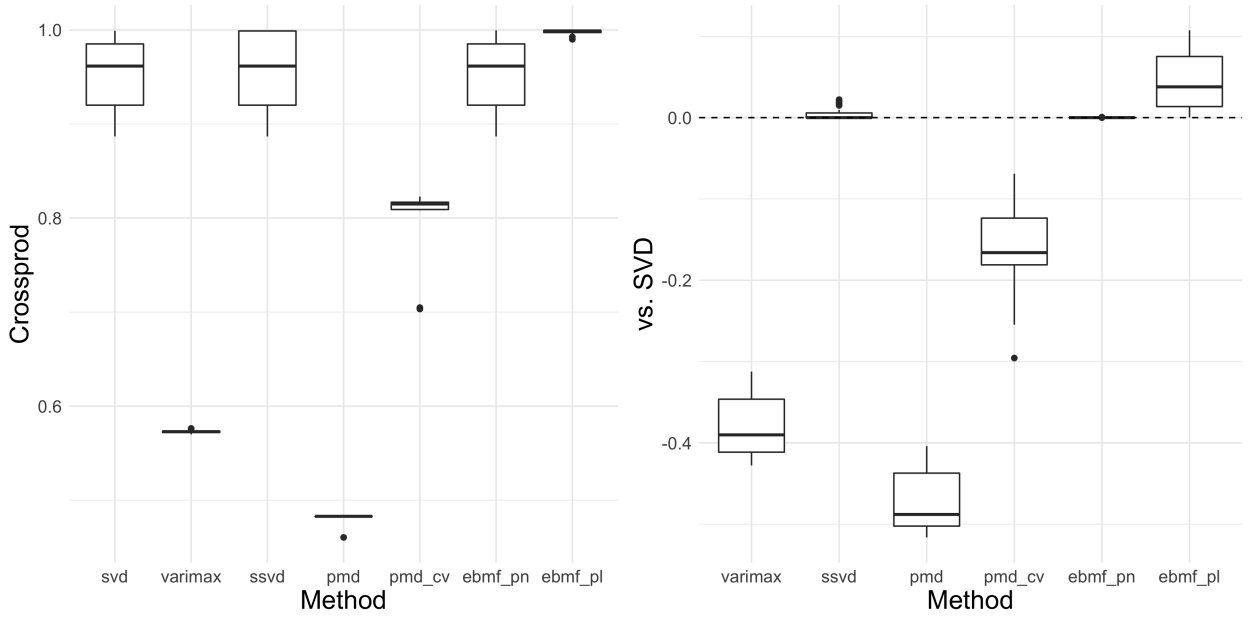


Figure 3.7: Crossproduct similarity over multiple balanced tree simulations. The plot on the left gives the absolute crossproduct similarity obtained by each method, while the plot on the right gives the distances relative to SVD. See Figure 3.6 and text for details.

normal (that is, if the weight π on the point mass δ_0 is estimated to be zero), then the posterior means will be exactly proportional to the observations, so that the updates modify the scale of the loadings but not their respective (expected) positions. Thus, if all SVD loadings are sufficiently far from zero, point-normal priors become normal priors, which fail to induce sparsity in an empirical Bayes matrix factorization. For a more precise statement, see Nakajima and Sugiyama [2011]. Due to this technical difficulty, I will only consider priors with heavier tails — here, the point-Laplace prior family — and, in following sections, priors that are better suited to the application.

3.3.4 EBMF for Tree-Structured Data

The flexibility of the EBMF framework (as implemented by R package **flashier**) allows for an approach that is more specifically tailored to tree-structured data. Some of the techniques outlined in this section are, to my knowledge, novel; others merely offer insight into practices

that are already common.

Greedy EBMF

I suggested in Section 3.3.1 that since the SVD of tree-structured data is, in favorable cases, similar to a divergence factorization, methods that "sparsify" the former might suffice to recover the latter. In the terminology of Wang and Stephens [2021], this amounts to "backfitting" an SVD with priors $g_\ell^{(k)}$ that promote sparsity. However, the other algorithm outlined by Wang and Stephens — the "greedy" `flash` algorithm — also has intuitive appeal.

In brief, while the backfitting algorithm begins with K factors and performs updates by looping over all K factors until convergence, the greedy algorithm adds one factor at a time, fixing the first $k-1$ factors and finding an optimal fit for the kth factor ($\boldsymbol{\ell}_k$ and $\mathbf{f}_k$) before adding factor $k+1$. In other words, the backfitting algorithm begins with a complete factorization and "rotates" it so that the loadings better fit their priors. In contrast, the greedy algorithm builds a factorization from scratch by successively finding best rank-one fits to matrices of residuals. Intuitively, it's possible to understand the latter as looking for the best "split" in the data at each juncture. For tree-structured data, one can imagine the greedy algorithm beginning at the root and moving leafward, creating a new branch with each factor.

Empirically, I find that the backfitting algorithm is prone to either getting stuck in local optima or stalling out on flat portions of the optimization surface (it is of course difficult to distinguish between the two problems). Since the SVD is already a good fit to the data, the backfitting algorithm often does not rotate away from the initial solution unless strong incentives are provided by the priors. For this reason, it can be preferable to use the greedy fit rather than the SVD to initialize the backfitting routine.

Tree Constraints

One can double down on the above interpretation of the greedy `flash` algorithm by enforcing a tree-like structure as factors are successively added. Ignore for a moment the first factor, which corresponds to the root. The second factor corresponds to the "earliest" divergence event and partitions the set of individuals $\{1, \ldots, n\}$ into two subsets according to the signs of the loadings ℓ_{i2}:

$$\mathcal{S}_+ := \{i : \ell_{i2} \geq 0\}, \tag{3.40}$$

$$\mathcal{S}_- := \{i : \ell_{i2} < 0\}. \tag{3.41}$$

In the tree interpretation, $\mathcal{S}_+$ is the set of individuals who diverge along the left branch and $\mathcal{S}_-$ diverges along the right branch.

The third factor then corresponds to a divergence event that is experienced by either the set $\mathcal{S}_+$ or the set $\mathcal{S}_-$, but not by both. Let it be the former, so that the third factor defines the partition:

$$\mathcal{S}_{++} := \{i : i \in \mathcal{S}_+,\ \ell_{i3} \geq 0\}, \tag{3.42}$$

$$\mathcal{S}_{+-} := \{i : i \in \mathcal{S}_+,\ \ell_{i3} < 0\}. \tag{3.43}$$

In this interpretation, it should be the case that $\ell_{i3} = 0$ for all $i \in \mathcal{S}_-$. Indeed, to ensure that such an interpretation is possible, one can fix these loadings at zero before fitting the third factor. One can then fix ℓ_{i4} at zero for all $i \in \mathcal{S}_+$ to get a fourth factor that partitions $\mathcal{S}_-$, and so on.

In detail, one is guaranteed to obtain a fit that can be easily interpreted as a tree by proceeding as follows:

1. Fix all loadings $\ell_{i1} = 1$ and fit $\mathbf{f}_1$. This factor corresponds to the drift event that

precedes the first divergence event, which is equally experienced by all individuals. As long as the prior family $\mathcal{G}_f^{(1)}$ is closed under scaling, the magnitude of the loadings $\boldsymbol{\ell}_1$ can be chosen arbitrarily.

2. Fit $\boldsymbol{\ell}_2$ and $\mathbf{f}_2$. Partition the set of individuals as in Equations 3.40-3.41.

3. Now iterate. One begins with a partition of the set of individuals into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_K$. Add K new factors $K+1, \ldots, 2K$ with $\ell_{i,K+k}$ fixed at 0 for $i \notin \mathcal{S}_k$ and with $\ell_{i,K+k}$ to be estimated for $i \in \mathcal{S}_k$. Define $\mathcal{S}_{k+}$ and $\mathcal{S}_{k-}$ similarly to Equations 3.42-3.43. Note, however, that `flash` will not retain factors when they no longer increase the EBMF objective: in this case, do not partition $\mathcal{S}_k$. This step yields a partition with between K and $2K$ blocks. Terminate when no new factors are added.

Strong Priors

While I used point-Laplace priors in the balanced tree example from Section 3.3.3, it's possible to use prior families that much more strongly favor the ternary structure of a divergence factorization. The most aggressive choice is what I will refer to as the family of "divergence priors," which are mixtures of three point masses, one at zero, one nonnegative, and one nonpositive:

$$\{g : g \sim \pi_\nu \delta(-\nu) + \pi_0 \delta_0 + \pi_\lambda \delta(\lambda),\ \nu \geq 0,\ \lambda \geq 0\}. \tag{3.44}$$

(For concision, I omit the obvious constraints $\pi_\nu, \pi_0, \pi_\lambda \geq 0$ and $\pi_\nu + \pi_0 + \pi_\lambda = 1$.)

I recommend caution, however, because the use of divergence priors makes it very difficult for `flash` to escape local optima. A choice that allows for more flexibility is what I call the family of "admixture priors," which adds a uniform component on $[-\nu, \lambda]$:

$$\{g : g \sim \pi_\nu \delta(-\nu) + \pi_0 \delta_0 + \pi_\lambda \delta(\lambda) + \pi_{\text{mix}} \text{Unif}[-\nu, \lambda],\ \nu \geq 0,\ \lambda \geq 0\}. \tag{3.45}$$

Note, however, that unlike unimodal families such as the family of point-Laplace priors, neither the family of divergence priors nor the family of admixture priors is a family of shrinkage priors. In other words, if one is given observations x_i with normally distributed standard errors s_i, so that $x_i \sim \mathcal{N}\left(\theta_i, s_i^2\right)$, and if one makes the additional empirical Bayes assumption that $x_i \sim g$, with g to be estimated from among the family of either divergence or admixture priors, then it is not guaranteed that $|\mathbb{E}\left(\theta_i \mid x_i, s_i, g\right)| \leq |x_i|$. Thus it is likely that these "strong" priors are somewhat less effective at inducing sparsity than a symmetric unimodal family with heavy tails.

3.3.5 Co-occurrence Matrix Factorization

If the prior families $\mathcal{G}_f^{(k)}$ are chosen to be the family of zero-centered normal distributions

$$\left\{g : g \sim \mathcal{N}\left(0, s^2\right),\ s^2 > 0\right\} \tag{3.46}$$

then, provided that the prior families $\mathcal{G}_\ell^{(k)}$ are closed under scaling, one can fix the priors $g_f^{(k)}$ to be standard normal distributions $\mathcal{N}(0,1)$. If, further, all standard errors σ_{ij} are assumed to be identical to σ, then one can integrate $\mathbf{F}$ out of the likelihood for the EBMF model:

$$\int_{\mathbf{L}} \int_{\mathbf{F}} \prod_{i,j} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(x_{ij} - \sum_k \ell_{ik} f_{jk}\right)^2\right) \prod_{j,k} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} f_{jk}^2\right) df_{jk} dg_{\mathbf{L}}(\mathbf{L}) \tag{3.47}$$

$$\propto \int_{\mathbf{L}} \left(\sigma^2\right)^{-\frac{np}{2}} \exp\left(-\frac{1}{2}\sum_j \left(\frac{1}{\sigma^2}\left(\mathbf{x}_j - \mathbf{L}\mathbf{f}_j\right)^T\left(\mathbf{x}_j - \mathbf{L}\mathbf{f}_j\right) + \mathbf{f}_j^T \mathbf{f}_j\right)\right) d\mathbf{f}_j dg_{\mathbf{L}}(\mathbf{L}) \tag{3.48}$$

$$\propto \int_{\mathbf{L}} \left(\sigma^2\right)^{-\frac{np}{2}} \det\left(\mathbf{L}\mathbf{L}^T + \sigma^2 \mathbf{I}_n\right)^{-\frac{p}{2}} \exp\left(-\frac{1}{2}\sum_j \mathbf{x}_j^T \left(\mathbf{L}\mathbf{L}^T + \sigma^2 \mathbf{I}_n\right)^{-1} \mathbf{x}_j\right) dg_{\mathbf{L}}(\mathbf{L}). \tag{3.49}$$

Thus if, as is often the case, estimates of $\mathbf{F}$ are not needed, then one can obtain estimates of $\mathbf{L}$ by fitting the model

$$\mathbf{x}_j \sim \mathcal{N}\left(0, \mathbf{L}\mathbf{L}^T + \sigma^2 \mathbf{I}_n\right),\ j = 1, \ldots, p \tag{3.50}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}_\ell^{(k)},\ k = 1, \ldots, K, \tag{3.51}$$

or equivalently,

$$\mathbf{X}\mathbf{X}^T \sim \mathcal{W}_n\left(\mathbf{L}\mathbf{L}^T + \sigma^2 \mathbf{I}_n,\ p\right) \tag{3.52}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}_\ell^{(k)},\ k = 1, \ldots, K, \tag{3.53}$$

where $\mathcal{W}_n\left(\Sigma,\ p\right)$ denotes an n-dimensional Wishart distribution with scale matrix Σ and p degrees of freedom.

Instead of fitting this model directly, I suggest using EBMF to fit the co-occurrence matrix $\frac{\mathbf{X}\mathbf{X}^T}{p}$. It's useful to compare models to see when and why such an approximation is reasonable. The model I have in mind is:

$$\frac{1}{p}\mathbf{X}\mathbf{X}^T = \mathbf{L}\mathbf{L}^T + \sigma^2 \mathbf{I}_n + \mathbf{E} \tag{3.54}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}_\ell^{(k)},\ k = 1, \ldots, K \tag{3.55}$$

$$e_{ij} \sim \mathcal{N}\left(0, \tau^2\right). \tag{3.56}$$

While the means $\mathbb{E}\left(\mathbf{X}\mathbf{X}^T\right)$ are the same in both models, the error distributions differ. The noise terms e_{ij} in the latter model are independent and identically distributed (with τ^2 to be estimated), which is of course not the case for the Wishart distribution. Marginally, however, the approximation is not unreasonable: the distribution of entry $\left[\mathbf{X}\mathbf{X}^T\right]_{ij}$ in Equa-

tion 3.52 is approximately normal as $p \to \infty$ with variance

$$p\left(\left(\left[\mathbf{L}\mathbf{L}^T\right]_{ij} + \sigma^2 \cdot 1_{i=j}\right)^2 + \left(\left[\mathbf{L}\mathbf{L}^T\right]_{ii} + \sigma^2\right)\left(\left[\mathbf{L}\mathbf{L}^T\right]_{jj} + \sigma^2\right)\right). \tag{3.57}$$

(Indeed, when the data $\mathbf{X}$ is a matrix of counts, then modeling errors as normally distributed is arguably *more* reasonable for the co-occurrence matrix than for the full-data matrix.) The terms $\left[\mathbf{L}\mathbf{L}^T\right]_{ij}$ have simple interpretations: they are the sum of the variances of the drift events jointly experienced by populations Pop(i) and Pop(j) (or, if Pop(i) = Pop(j), the sum of the variances of the drift events experienced by population Pop(i)). If all populations experience roughly the same total amount of drift, then the variances given by Equation 3.57 will differ by at most a factor of two or so, and if, as is the case in many scenarios, the total drift is dominated by the common drift event that proceeds from the root to the first divergence event (drift event $\boldsymbol{\alpha}$ in Figure 3.1), all variances will be very similar. Thus, if one can safely ignore the correlations among entries $\left[\mathbf{X}\mathbf{X}^T\right]_{ij}$, then fitting the model given by Equations 3.54-3.56 should give a good approximation to the "true" Wishart model.

Now, note that Equations 3.54-3.56 differ from the standard EBMF model (Equations 3.33-3.36) in two small but important ways. First, there is an additional parameter σ^2 in Equation 3.54 that must be estimated. I find that it works well to run `flash` for a fixed number of iterations, then alternate between estimating σ^2 using maximum likelihood by setting

$$\hat{\sigma}^2 = \max\left\{0, \frac{1}{n}\sum_{i=1}^{n}\left(\left[\frac{1}{p}\mathbf{X}\mathbf{X}^T\right]_{ii} - \mathbb{E}\left[\mathbf{L}\mathbf{L}^T\right]_{ii}\right)\right\} \tag{3.58}$$

and running `flash` on the matrix $\mathbf{X}\mathbf{X}^T - \hat{\sigma}^2\mathbf{I}_n$ with a fixed maximum number of iterations. This algorithm can be considered to have converged when both the estimate $\hat{\sigma}^2$ has converged and `flash` has converged before reaching the maximum number of iterations.

The second difference from the standard EBMF model is that there are two "copies" of the loadings matrix $\mathbf{L}$ rather than distinct matrices $\mathbf{L}$ and $\mathbf{F}$. However, it's not clear how

one might enforce the constraint $\mathbf{L} = \mathbf{F}$ within the `flash` framework. Further, I find that simply fitting the model

$$\frac{1}{p}\mathbf{X}\mathbf{X}^T = \mathbf{L}\mathbf{F}^T + \sigma^2\mathbf{I}_n + \mathbf{E} \tag{3.59}$$

$$\ell_{ik} \sim g_\ell^{(k)} \in \mathcal{G}^{(k)},\ k = 1, \ldots, K \tag{3.60}$$

$$f_{jk} \sim g_f^{(k)} \in \mathcal{G}^{(k)},\ k = 1, \ldots, K \tag{3.61}$$

$$e_{ij} \sim \mathcal{N}\left(0, \tau^2\right) \tag{3.62}$$

yields estimates such that $\mathbf{L} \approx \mathbf{F}$ (note that I require the prior family used for $\boldsymbol{\ell}_k$ to be the same as the one used for $\mathbf{f}_k$). I will refer to the model given by Equations 3.59-3.62 as "empirical Bayes covariance matrix factorization" (EBcovMF). The terminology is somewhat abusive, since I'm more interested here in co-occurrence matrices than in covariance matrices. I want to emphasize, however, that EBcovMF is in fact a factor analytic method (as is apparent from the model described by Equations 3.54-3.56), so I've opted for the term that practitioners of factor analysis are more accustomed to.

3.3.6 Example: Unbalanced Tree

To illustrate the ideas outlined in the previous section, I simulate a four-population tree in which both population sizes and drift event variances differ. Recall that my primary criterion is that a method's results should be interpretable in terms of a divergence factorization. However — using the notation from Section 3.2.4 — it's not clear what the values of the λ_ks and ν_ks should be, except in the special case of balanced trees. In particular, the "crossproduct similarity" that I used to compare fits in Section 3.3.3 is not generally applicable.

I propose the following general measure to determine whether a given variational method successfully recovers the desired divergence factorization. Let $q_\ell^{(k)}$ be the variational posterior

for the kth loadings column of a proposed factorization and let $\tilde{\boldsymbol{\ell}}_k$ be the kth column of the target divergence factorization. I only require knowledge of whether $\tilde{\ell}_{ik} < 0$, $\tilde{\ell}_{ik} = 0$, or $\tilde{\ell}_{ik} > 0$: in particular, λ_k and ν_k need not be known. Define the "accuracy" of the proposed factorization as:

$$\max_{\Pi \in \mathrm{S}_k} \frac{1}{nK} \sum_{k=1}^{K} \max_{c_k \in \{-1,+1\}} \left(\mathrm{acc}_0 \left(q_\ell^{(\Pi(k))}, \tilde{\boldsymbol{\ell}}_k \right) + \mathrm{acc}_\lambda \left(c_k q_\ell^{(\Pi(k))}, \tilde{\boldsymbol{\ell}}_k \right) + \mathrm{acc}_\nu \left(c_k q_\ell^{(\Pi(k))}, \tilde{\boldsymbol{\ell}}_k \right) \right), \tag{3.63}$$

where

$$\mathrm{acc}_0 \left(q_\ell^{(k)}, \tilde{\boldsymbol{\ell}}_k \right) = \sum_{\{i : \tilde{\ell}_{ik} = 0\}} \mathrm{lfsr} \left(q_{\ell_i}^{(k)} \right) \tag{3.64}$$

$$\mathrm{acc}_\lambda \left(q_\ell^{(k)}, \tilde{\boldsymbol{\ell}}_k \right) = \sum_{\{i : \tilde{\ell}_{ik} = \lambda_k\}} \left(1 - \mathrm{lfsr} \left(q_{\ell_i}^{(k)} \right) \right) \cdot 1_{\mathbb{E}_q(\ell_{ik}) > 0} \tag{3.65}$$

$$\mathrm{acc}_\nu \left(q_\ell^{(k)}, \tilde{\boldsymbol{\ell}}_k \right) = \sum_{\{i : \tilde{\ell}_{ik} = -\nu_k\}} \left(1 - \mathrm{lfsr} \left(q_{\ell_i}^{(k)} \right) \right) \cdot 1_{\mathbb{E}_q(\ell_{ik}) < 0}. \tag{3.66}$$

The components acc_0, acc_λ, and acc_ν measure the number of individuals that are "correctly" identified as having a zero, positive, or negative loading, where the confidence of the assignment is a function of the local false sign rate $\mathrm{lfsr} \left(q_{\ell_i}^{(k)} \right)$. As Stephens [2017] defines it, the local false sign rate is "the probability that we would make an error in the sign of [an] effect if we were forced to declare it either positive or negative":

$$\mathrm{lfsr} \left(q_{\ell_i}^{(k)} \right) = \min \left(\mathbb{P} \left(\ell_{ik} \geq 0 \mid \mathbf{X}, g \right), \mathbb{P} \left(\ell_{ik} \leq 0 \mid \mathbf{X}, g \right) \right), \tag{3.67}$$

where g is shorthand for the collection of priors $g_\ell^{(1)}, \ldots, g_\ell^{(K)}, g_f^{(1)}, \ldots, g_f^{(K)}$. The constants $c_k \in \{-1, +1\}$ in Equation 3.63 are introduced so that the accuracy does not change if the signs of the loadings $\boldsymbol{\ell}_k$ are flipped. Finally, the maximum is taken over the symmetric group of permutations S_k so that the accuracy does not depend upon the ordering of the columns

$\boldsymbol{\ell}_k$.

Figure 3.8 shows details for a single simulation of an unbalanced tree, while Figure 3.9 plots the accuracy of methods over multiple simulations. All simulated trees have the four-population tree structure illustrated in Figure 3.1, with the population sizes randomly set to between 20 and 80, the standard deviation of drift event $\boldsymbol{\alpha}$ fixed at 10, and the standard deviations of the other drift events randomly sampled from a uniform distribution on $[1, 6]$. All standard errors are fixed at 1, and the number of genes p is set to 10000.

Interestingly, results obtained using the greedy algorithm (`ebmf_greedy` in the figures) typically don't differ appreciably from results obtained by backfitting from the SVD, even though the greedy algorithm is, in most scenarios, a much quicker and dirtier way to fit an EBMF model. Using admixture priors (`ebmf_admix`) rather than point-Laplace yields a very small improvement, but it appears that these priors are simply not "strong" enough to produce a noticeable difference in fits. The divergence priors (`ebmf_div`), on the other hand, are probably too strong. Although they are much more effective at producing sparse factors, they also incorrectly set many loadings to zero (see, for example, Population D in factor $k = 2$).

EBcovMF improves upon EBMF in terms of accuracy for reasons that are illuminated by the example shown in Figure 3.8. Even though results seem similar at first glance, EBcovMF finds a sparse solution for $k = 3$, while EBMF loadings for Populations C and D are small but not exactly zero. Indeed, a known issue with variational methods is that estimates of posterior variances are too small: see, for example, Blei et al. [2017]. To be sure, both EBMF and EBcovMF are variational. However, the priors $g_\ell^{(k)}$ have more relative importance in EBcovMF when p is large, so that they have a better chance at acting as an effective "penalty term" and shrinking loadings to zero. To see this, note that the EBMF objective can be written as the sum of the expected data log likelihood and the KL-divergence from the priors

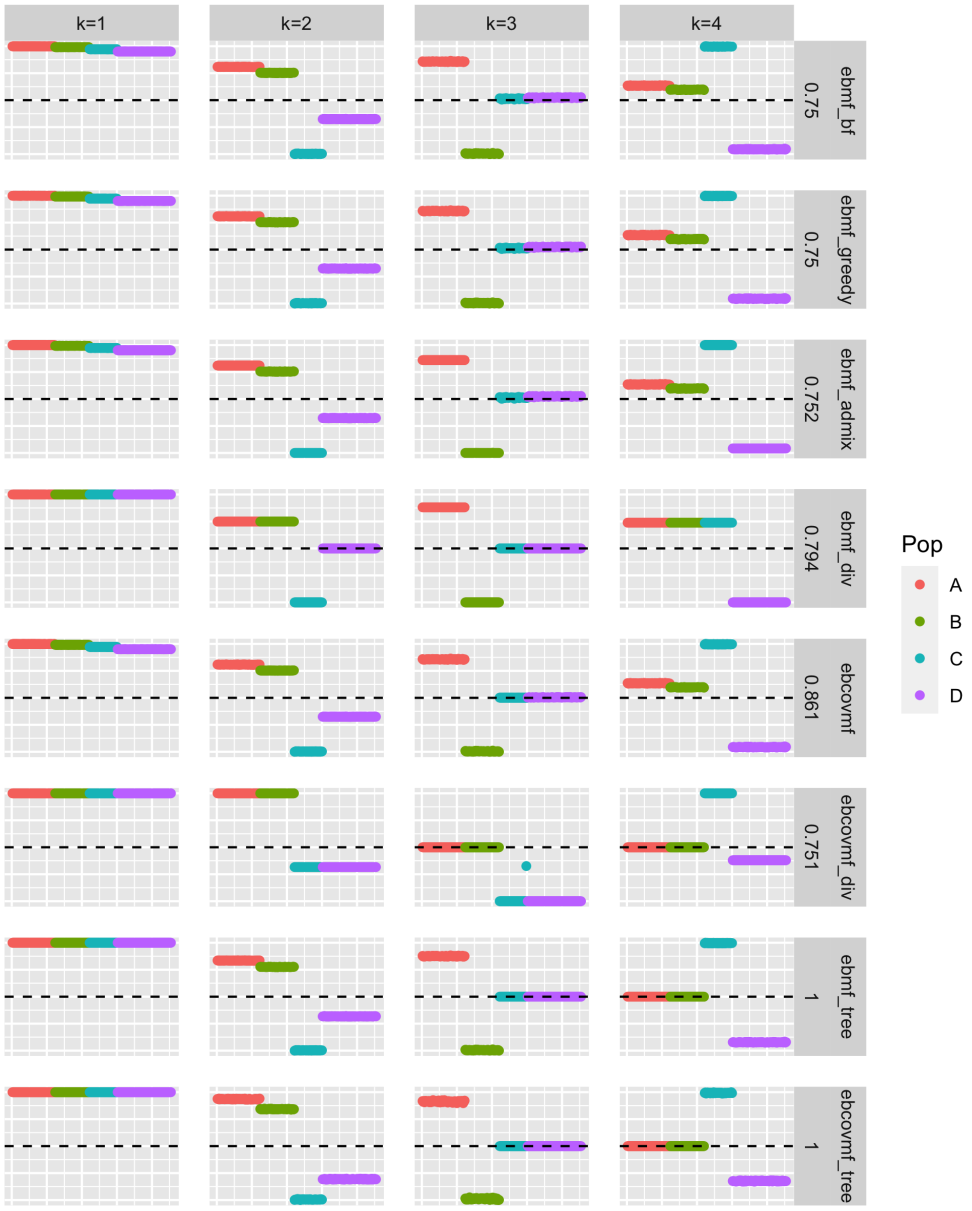


Figure 3.8: Comparison of EBMF methods for an unbalanced tree. The methods considered are the standard "backfitting" and "greedy" algorithms with point-Laplace priors; EBcovMF with point-Laplace priors; greedy EBMF with "admixture" and "divergence" priors; EBcovMF with divergence priors; and EBMF and EBcovMF with tree constraints (and point-Laplace priors). The "accuracy" of each factorization (see text for a definition) is given in the right margins.

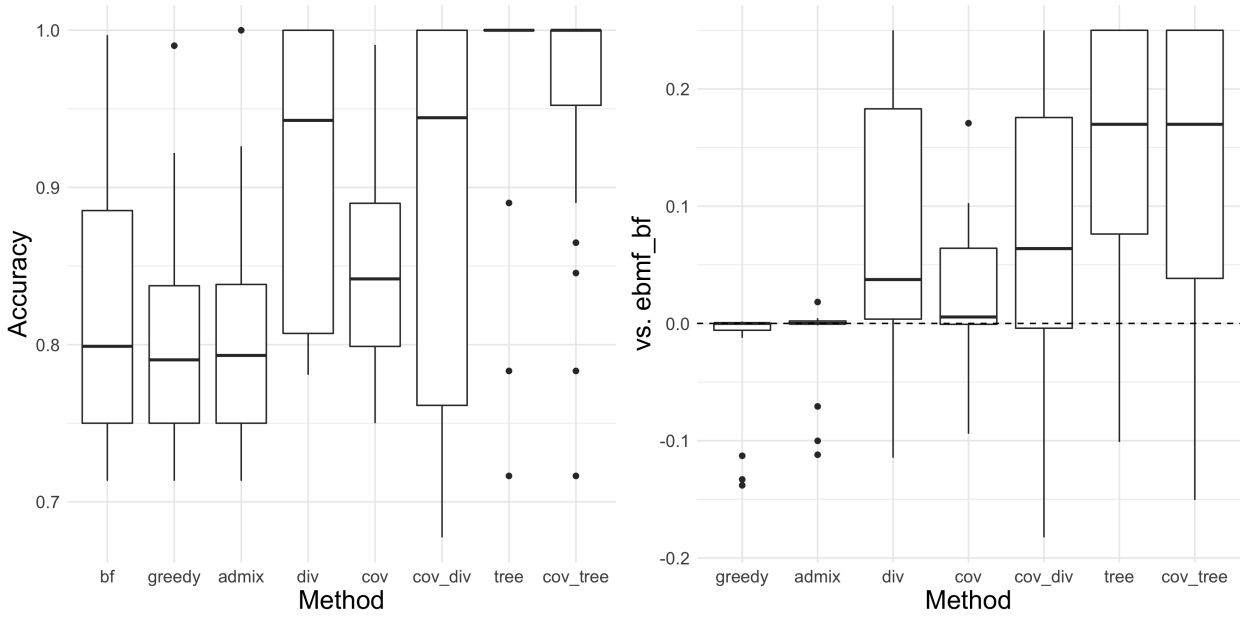


Figure 3.9: Accuracy of tree reconstruction over multiple unbalanced tree simulations. The plot on the left gives the absolute accuracy obtained by each method, while the plot on the right gives the accuracy relative to EBcovMF. See Figure 3.8 and text for details.

on the loadings to the posteriors (Equation (A.4) in Wang and Stephens [2021]):

$$\mathbb{E}_{q_{\mathbf{L}},q_{\mathbf{F}}} \log p\left(\mathbf{X} \mid \mathbf{L}, \mathbf{F}, \sigma^2\right) + \sum_{k=1}^{K} \mathbb{E}_{q_\ell^{(k)}} \log \frac{g_\ell^{(k)}}{q_\ell^{(k)}} + \sum_{k=1}^{K} \mathbb{E}_{q_f^{(k)}} \log \frac{g_f^{(k)}}{q_f^{(k)}}, \tag{3.68}$$

where $q_\ell^{(k)}$ and $q_f^{(k)}$ respectively denote the posterior distribution of columns $\boldsymbol{\ell}_k$ and $\mathbf{f}_k$. Since $\mathbf{L} \approx \mathbf{F}$ in EBcovMF, the EBcovMF objective is approximately

$$\mathbb{E}_{q_{\mathbf{L}}} \log p\left(\frac{\mathbf{X}\mathbf{X}^T}{p} \mid \mathbf{L}, \tau^2\right) + 2\sum_{k=1}^{K} \mathbb{E}_{q_\ell^{(k)}} \log \frac{g_\ell^{(k)}}{q_\ell^{(k)}} \tag{3.69}$$

If n is fixed as $p \to \infty$, then the expected data log likelihood scales as p in EBMF but only as $\log p$ in EBcovMF (p affects the residual variance term τ^2 but not the number of entries in $\mathbf{X}\mathbf{X}^T$). The KL-divergence from $g_{\mathbf{L}}$ to $q_{\mathbf{L}}$ does not, of course, scale with p, so that the expected data log likelihood dominates the EBMF objective much more quickly

than it does the EBcovMF objective. I speculate that this is why the problem of small but non-zero loadings appears in Figure 3.8, where I set $p = 10000$, but not Figure 3.6, where I set $p = 1000$.

Finally, the methods that use tree constraints (`ebmf_tree` and `ebcovmf_tree`) do very well in terms of the "accuracy" metric outlined above. This result is hardly surprising, since the tree structure is forced upon the dataset so that EBMF and EBcovMF only need to find the correct splits at each juncture.

3.3.7 Example: Admixture

While methods that enforce tree constraints yield results that allow the correct tree structure to be easily identified (at least for the simple four-population trees considered above), they give misleading information about admixed populations. In effect, such methods uniquely assign every individual to a single leaf, so that it's difficult to distinguish between admixed populations and the "pure" populations that they most closely resemble.

To be able to detect admixtures, I suggest "unfixing" the loadings that are constrained to be zero and backfitting. The resulting "relaxed" fit is simply an EBMF (or EBcovMF) fit with, for example, point-Laplace priors, but the tree-constrained fit usually provides a better initialization than the SVD, allowing it to find a local optimum that is much more easily interpretable in terms of a divergence factorization.

To illustrate, I simulate the four-population tree with admixture illustrated above in Figure 3.3. I simulate 20 admixed inviduals (Population BC) and 40 individuals from each "pure" population. The admixture proportion is randomly sampled from a uniform distribution on $[0.5, 0.9]$. The standard deviation of drift event $\boldsymbol{\alpha}$ is fixed at 10, the standard deviations of other drift events are randomly sampled from Unif$[3, 6]$, and standard errors are fixed at 1. The number of genes p is fixed at 10000.

Results are shown in Figures 3.10 and 3.11. I give two measures of accuracy. Both are

as in the previous section (Equation 3.63), but the overall accuracy is over the set of all individuals while the admixture accuracy only considers admixed individuals. The loadings of admixed individuals for the first divergence event ($\pi\lambda_2 - (1-\pi)\nu_2$ in Equation 3.15) are disregarded, since the sign cannot be determined when λ_2 and ν_2 are unknown.

The relaxed EBMF method yields an ideal fit for the simulation shown in Figure 3.10, but as Figure 3.11 shows, the more usual outcome is that the fit to the tree becomes slightly worse while the fit to the admixed population improves substantially. For reasons that I suspect are technical but that I've been unable to elucidate, the relaxed EBcovMF fit tends to be a bit worse than the relaxed EBMF fit (note, in particular, that the first factor seems to absorb some noise in `ebmf_relax`, while the fourth factor appears to play the same role for `ebcovmf_relax`). Nonetheless, I will typically prefer to use EBcovMF in the examples that follow due to the computational convenience of working with the co-occurrence matrix rather than with the full data matrix.

3.3.8 Summary: tree-EBMF

The method that I prefer to use for tree-structured data with the possible presence of admixtures is greedy EBMF (or EBcovMF) with point-Laplace priors, tree constraints, and relaxation. For brevity, I refer to this method as tree-EBMF (or tree-EBcovMF). To summarize, the method is as follows:

- Use the greedy `flash` algorithm to sequentially add factors with point-Laplace priors on the loadings. For each new factor, fix a subset of the loadings so that the pattern of zeroes conforms to a strict tree-like structure, as described in detail in Section 3.3.4. One can either allow `flash` to add as many factors as possible (since the algorithm terminates when new factors no longer improve the optimization objective) or set a maximum number of factors or maximum tree depth.

- Unfix all loadings and backfit until convergence.

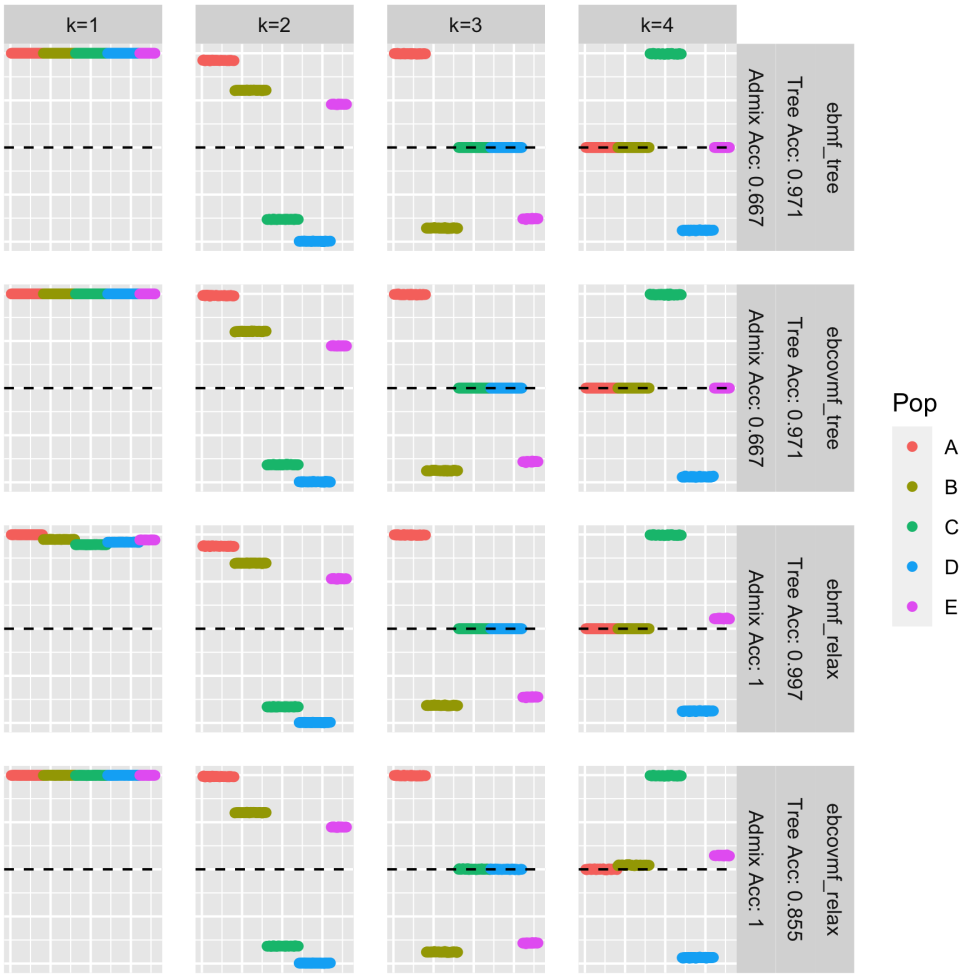


Figure 3.10: Comparison of EBMF methods for an unbalanced tree with admixture (Population E). In addition to EBMF and EBcovMF with tree constraints (which were also used to fit unbalanced trees), I consider a "relaxation" technique whereby I unfix loadings and backfit. The right margins give two "accuracy" measures: while the "underlying tree" accuracy is as above (Figure 3.7), the "admixed population" accuracy restricts the calculation to the subset of admixed individuals.

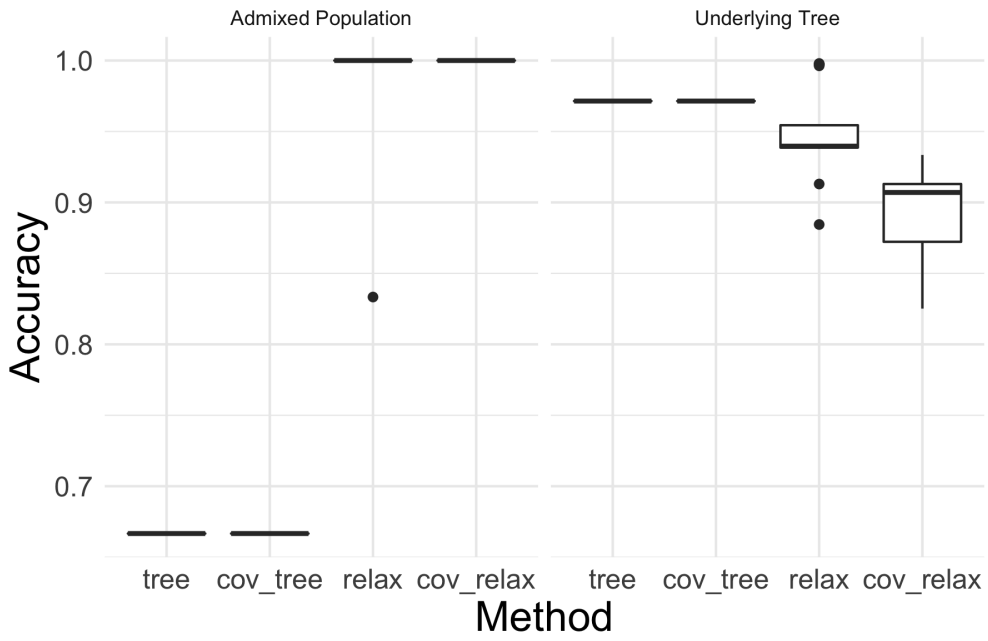


Figure 3.11: Accuracy of tree reconstruction over multiple simulations of unbalanced trees with admixture. See Figure 3.10 and text for details.

- For EBcovMF only: estimate the error variance $\hat{\sigma}^2$ and re-run the backfitting algorithm using $\mathbf{X} - \hat{\sigma}^2\mathbf{I}_n$ as the data matrix. Repeat, updating $\hat{\sigma}^2$ and backfitting until the change in $\hat{\sigma}^2$ is small.

For reference, I include a code snippet that implements tree-EBMF in Section 3.6.

3.4 Examples

The following sections apply tree-EBMF and tree-EBcovMF to simulated and real-data examples. Section 3.4.1 includes two simulations that are a bit more challenging than previous ones: a six-population tree with an additional admixed population; and a four-population "star" in which there is no tree-like structure and which can therefore serve as a "null" test case. Section 3.4.2 uses tree-EBcovMF to explore population structure in two population genetics datasets: the 1000 Genomes Project Phase 3 dataset, which collected genotypes for 2,504 individuals from 26 populations worldwide (1000 Genomes Project Consortium et al.

[2015]); and a dataset published by Schweizer et al. [2016] that includes genotypes for 111 North American gray wolves.

3.4.1 Simulations

Six-Population Tree with Admixture

I first consider a larger, six-population tree with an additional admixed population (see Figure 3.12). I choose more "realistic" parameter settings in that the number of genes is larger ($p = 50000$) and the overall drift is dominated by the shared drift event $\boldsymbol{\alpha}$, with drift variances decreasing leafward (see the figure for the exact settings). In Figure 3.13, I show the SVD as well as results obtained using both tree-EBMF and tree-EBcovMF (as summarized in Section 3.3.8).

In this unbalanced setting, the underlying tree structure is already apparent from the SVD, and all that is needed is to "sparsify" the loadings. tree-EBMF and tree-EBcovMF produce visually identical results. In particular, both drop the factor corresponding to the divergence event in which populations E and F split ($k = 6$ in Figure 3.13). In effect, the drift events $\boldsymbol{\delta}_E$ and $\boldsymbol{\delta}_F$ have relatively small variance, with both $\sigma^2_{\delta_E}$ and $\sigma^2_{\delta_F}$ equal to the error variance σ^2_ϵ, so the absence of this divergence event is probably not overly concerning. In any case, both tree-EBMF and tree-EBcovMF produce extremely clean and interpretable factors, with very sensible loadings for the admixed population CD.

Four-Population Star

Next I simulate the four-population "star" structure illustrated in Figure 3.2. To make the scenario more challenging, I make all drift event variances equal to one another (I omit the shared drift event $\boldsymbol{\alpha}$). Thus each population mean is independent and identically distributed with distribution $\mathcal{N}(0, \sigma^2 I_p)$, and there is no underlying tree structure that relates the pop-

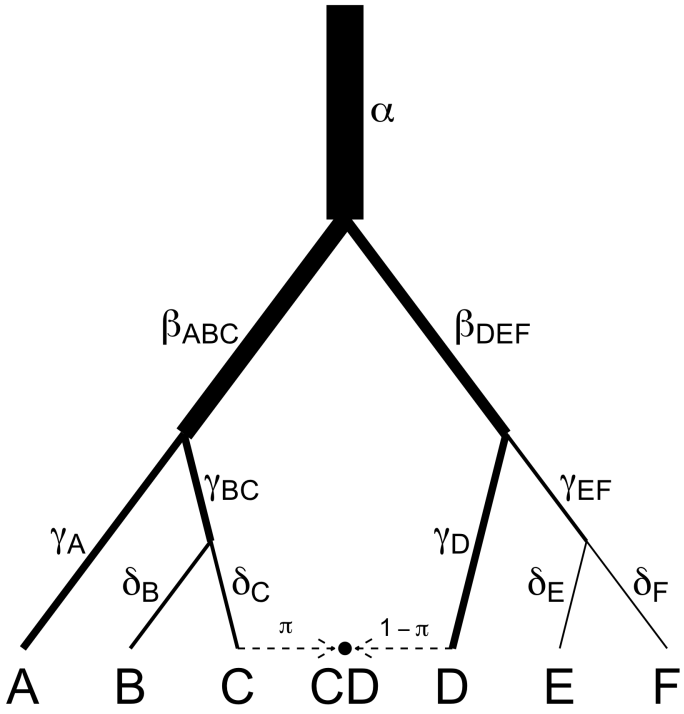


Figure 3.12: A six-population tree with admixture. In the simulated example, all drift events are normally distributed with mean zero and variances $\sigma^2_\alpha = 20^2$, $\sigma^2_{\beta_{ABC}} = 10^2$, $\sigma^2_{\beta_{DEF}} = 6^2$, $\sigma^2_{\gamma_A} = \sigma^2_{\gamma_{BC}} = \sigma^2_{\gamma_D} = 4^2$, $\sigma^2_{\gamma_{EF}} = \sigma^2_{\delta_B} = \sigma^2_{\delta_C} = 2^2$, and $\sigma^2_{\delta_E} = \sigma^2_{\delta_F} = \sigma^2_\epsilon = 1$. (In the figure, the width of each edge is proportional to the standard deviation of the corresponding drift event.) The admixture proportion π is set at 0.5.

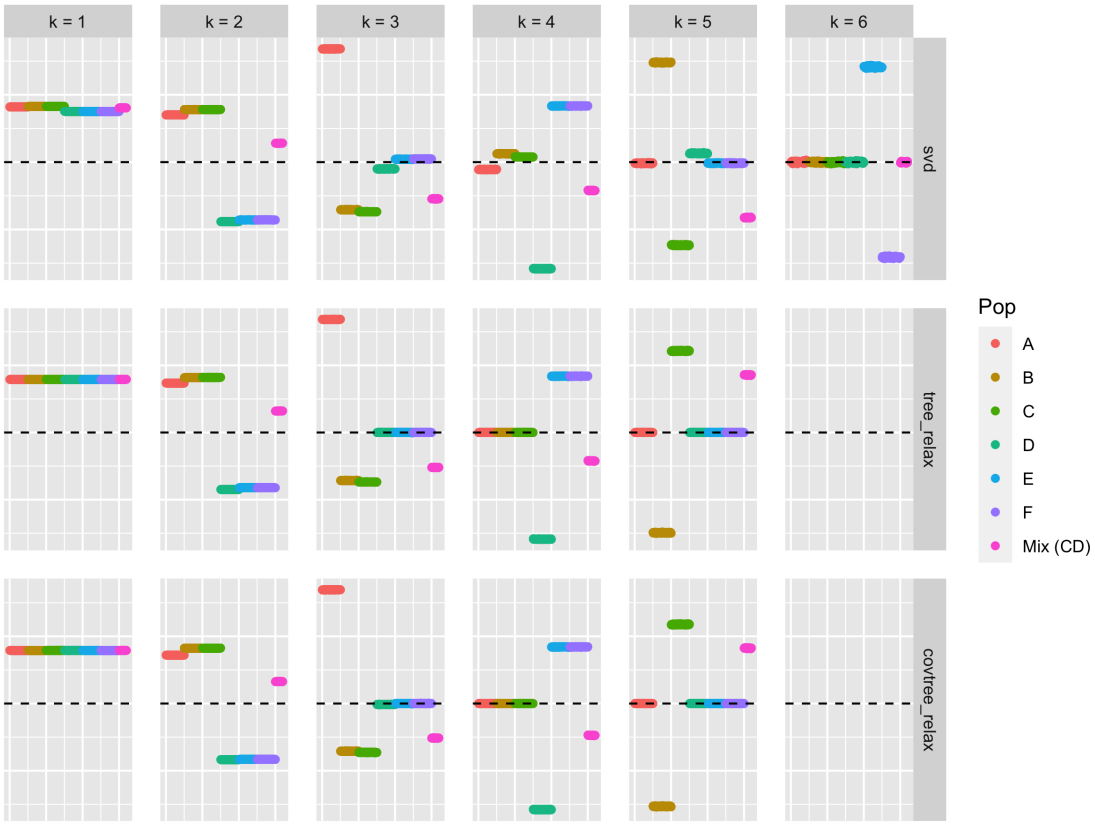


Figure 3.13: Simulation of a six-population tree with admixture (see Figure 3.12). In addition to the SVD, I include results for tree-EBMF and tree-EBcovMF (as summarized in Section 3.3.8). I set $\sigma^2_\alpha = 20^2$, $\sigma^2_{\beta_{ABC}} = 10^2$, $\sigma^2_{\beta_{DEF}} = 6^2$, $\sigma^2_{\gamma_A} = \sigma^2_{\gamma_{BC}} = \sigma^2_{\gamma_D} = 4^2$, $\sigma^2_{\gamma_{EF}} = \sigma^2_{\delta_B} = \sigma^2_{\delta_C} = 2^2$, and $\sigma^2_{\delta_E} = \sigma^2_{\delta_F} = \sigma^2_\epsilon = 1$, with the admixed population a 50-50 admixture of populations C and D. Each "pure" population has 100 individuals, while the admixed population has 40, and p is set at 50000.

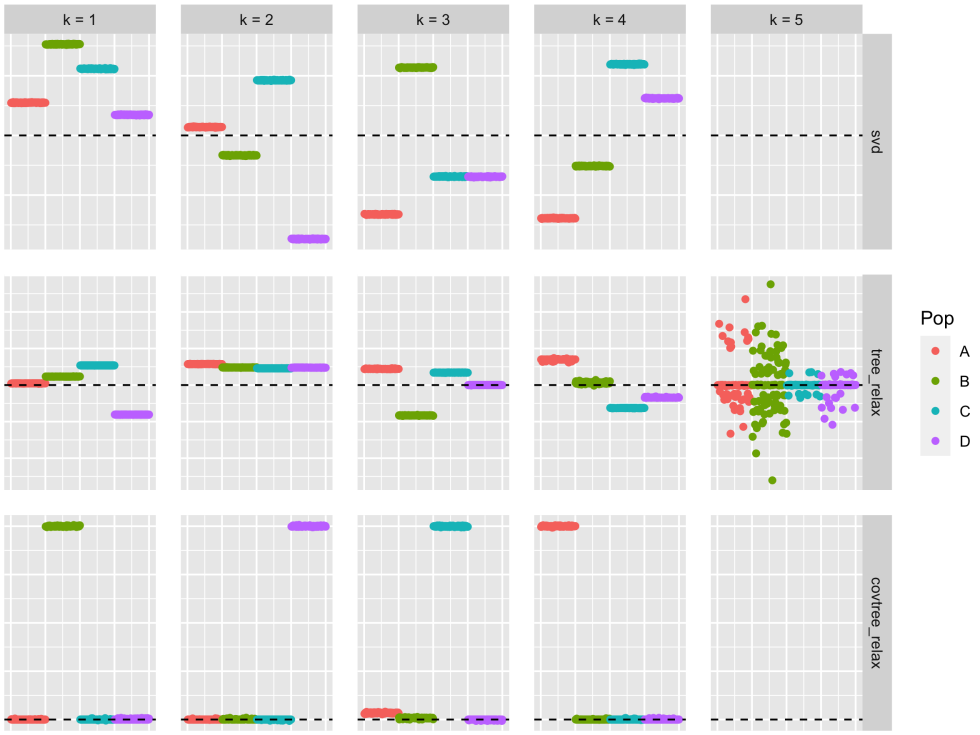


Figure 3.14: Simulation of a four-population "star" (see Figure 3.2). As in the previous figure, I include SVD, tree-EBMF, and tree-EBcovMF results. I omit the shared drift event $\boldsymbol{\alpha}$, and I set $\sigma^2_{\beta_A} = \sigma^2_{\beta_B} = \sigma^2_{\beta_C} = \sigma^2_{\beta_D} = 4$ and $\sigma^2_\epsilon = 1$. Each population has 100 individuals, with $p = 50000$.

ulations to one another. I set $p = 50000$, and I use the same methods as in the previous simulation (SVD, tree-EBMF, and tree-EBcovMF). Results are shown in Figure 3.14.

As discussed in Section 3.3.1, it follows from the balanced nature of this example that the SVD is not unique in the limit $p \to \infty$. Indeed, all four top eigenvalues, which correspond to the four drift events $\boldsymbol{\beta}_A, \ldots, \boldsymbol{\beta}_D$, are identical in the limit (and very similar in the finite regime). Thus the SVD is not expected to be even approximately sparse and a tree-based method must be able to "rotate" the loadings in addition to sparsifying them.

Unlike in the previous example, the tree-EBcovMF results are much different from the tree-EBMF results: indeed, only tree-EBcovMF easily finds the desired factorization (which is just the population factorization described in Section 3.2.2). Now, the results shown for tree-EBMF use the default convergence tolerance, and by decreasing the tolerance enough

one can eventually obtain the population factorization. However, convergence is very slow, and not only because it uses the full-data matrix rather than the co-occurrence matrix — so that each iteration takes more time —, but also because the "rotation" is much more gradual, so that more iterations are required to find the desired solution. I suspect that tree-EBcovMF is more successful here because, as suggested in section 3.2.6, the point-Laplace prior acts as a more effective "penalty term" in EBcovMF than in EBMF.

3.4.2 Real Data Examples

1000 Genomes Project

My next example considers the Phase 3 data from the 1000 Genomes Project, which includes genotypes for 2,504 individuals from 26 populations across the globe. I use the same dataset used in the flagship paper (1000 Genomes Project Consortium et al. [2015]), which filtered out rare variants and thinned the genome to reduce the effect of linkage disequilibrium. I then oriented the data so that the counts are of derived alleles rather than ancestral ones. As discussed in Section 3.2.1, this orientation allows the drift event $\boldsymbol{\alpha}$ to be interpreted as the drift from an ancestral population to the first divergence event. Because the dataset is quite large ($p = 184{,}224$), I elected to discard the full-data matrix and to only run tree-EBcovMF on the co-occurrence matrix $\frac{\mathbf{X}\mathbf{X}^T}{p}$. I added factors using tree constraints up to a depth of 5 for a maximum of $2^5 = 32$ factors (see Section 3.3.4) and then relaxed the constraints and backfit (as in Section 3.3.7). I removed any factors that were zeroed out as a result of the backfit, ordered factors by decreasing proportion of variance explained (as calculated by `flash`), and normalized loadings so that $\|\boldsymbol{\ell}_k\|_\infty = 1$ for all k.

Results are shown in Figures 3.15 and 3.16. The plot labels are my own interpretations of the factors. Apart from the first and last factors, each is easily interpretable as either a drift event that is largely specific to a single super-population or a divergence event that differentiates either among or within super-populations. In order, I interpret the factors as:

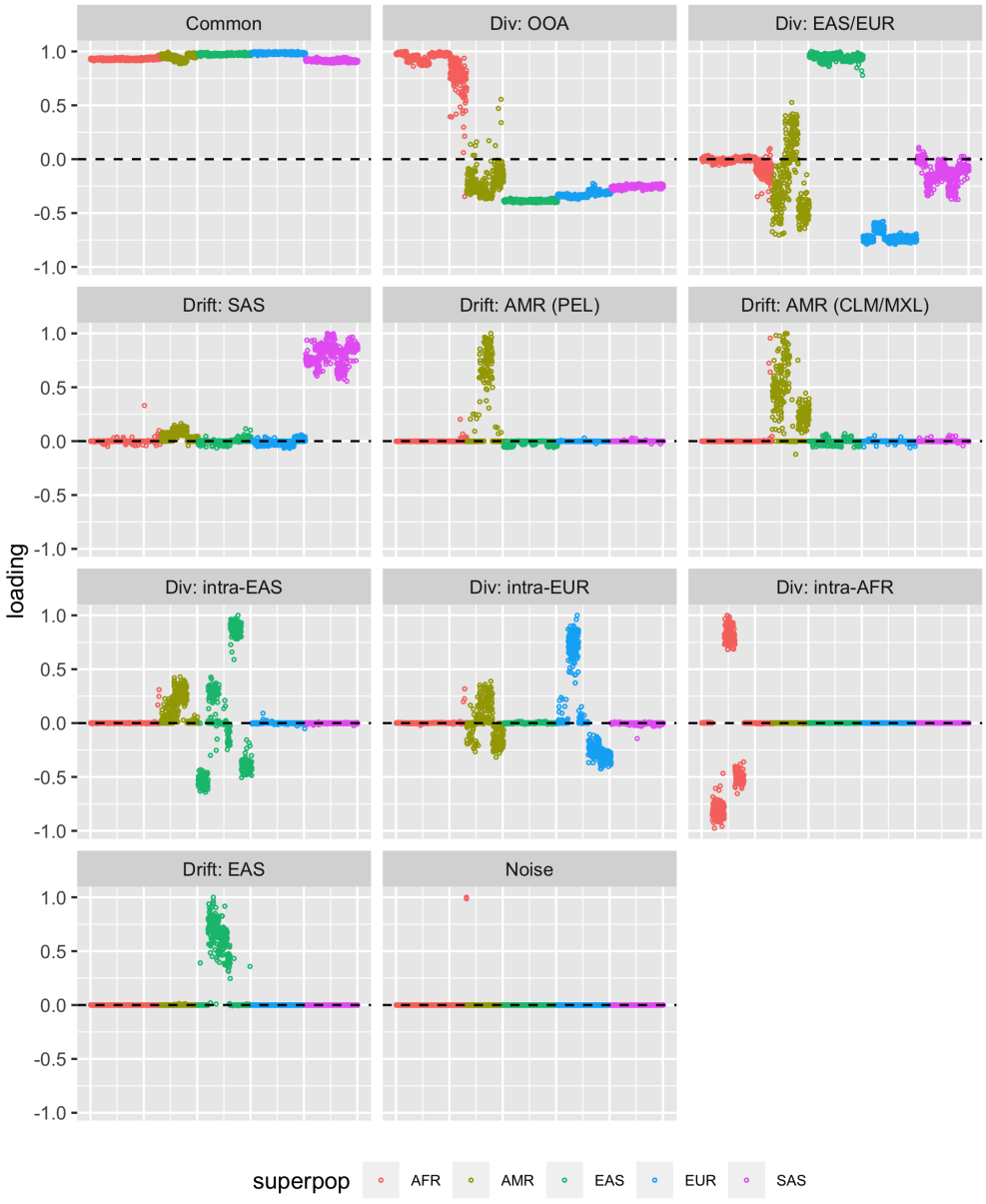


Figure 3.15: tree-EBcovMF results for the 1000 Genome Project Phase 3 dataset. The five "super-populations" are Africa (AFR), the Americas (AMR), East Asia (EAS), Europe (EUR), and South Asia (SAS). The factor titles are my own interpretations of the loadings. OOA is an abbreviation for "out of Africa." PEL, CLM, and MXL are populations from, respectively, Peru, Columbia, and Mexico. See also 3.16, which "zooms in" on each super-population.

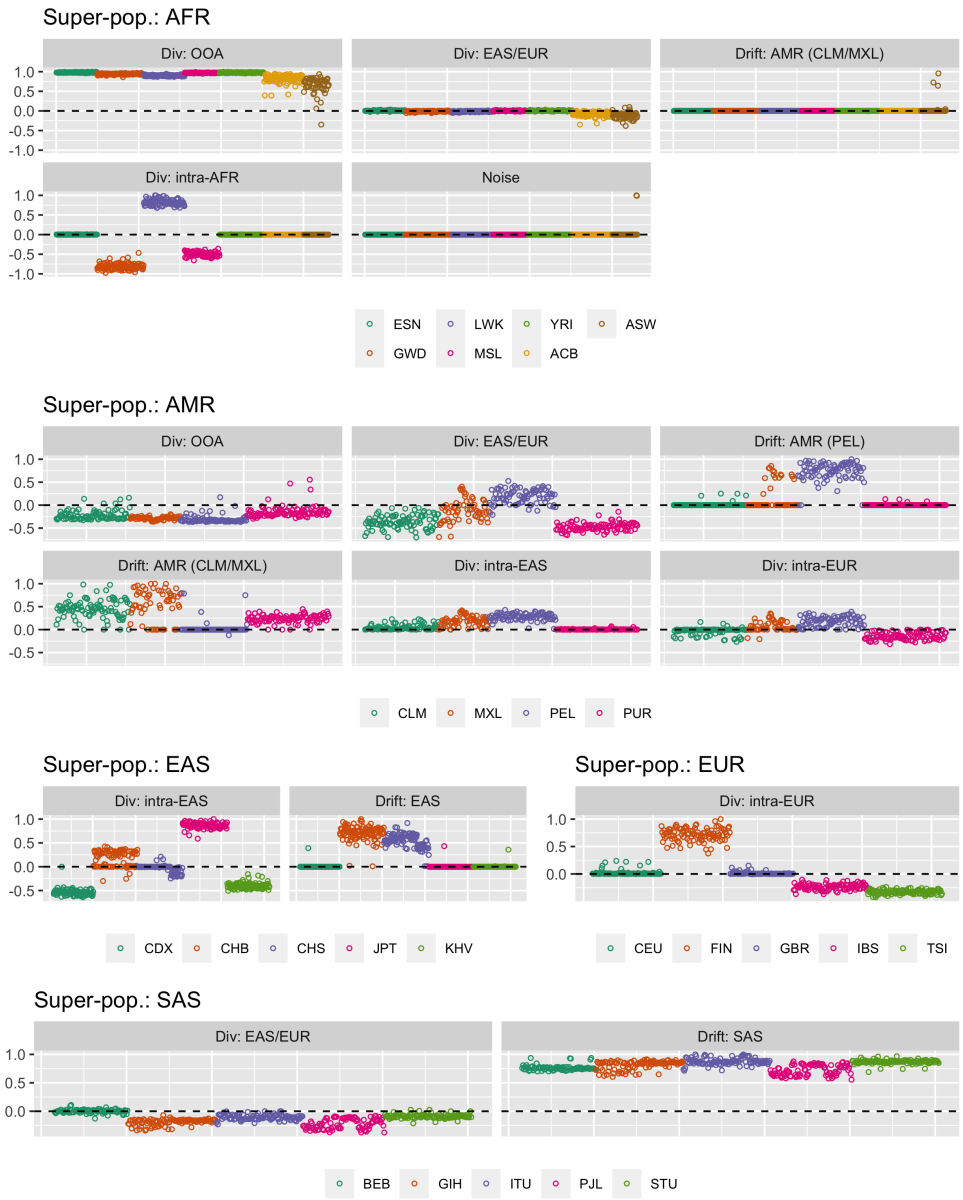


Figure 3.16: tree-EBcovMF results for the 1000 Genome Project Phase 3 dataset. Loadings are the same as in Figure 3.15, but here I zoom in on each super-population. ESN: Esan in Nigeria; GWD: Gambians in the Western Division; LWK: Luhya in Webuye, Kenya; MSL: Mende in Sierra Leone; YRI: Yoruba in Ibadan, Nigeria; ACB: African Caribbeans in Barbados; ASW: African Americans in the Southwest US. CLM: Columbians in Medellín; MXL: Mexicans in Los Angeles; PEL: Peruvians in Lima; PUR: Puerto Ricans. CDX: Chinese Dai in Xishuangbanna; CHB: Han Chinese in Beijing; CHS: Han Chinese in South China; JPT: Japanese in Tokyo; KHV: Kinh in Ho Chi Minh City. CEU: Utah residents with Northern and Western European ancestry; FIN: Finns; GBR: British; IBS: Iberians in Spain; TSI: Toscani. BEB: Bengali in Bangladesh; GIH: Gujarati Indians in Houston, Texas; ITU: Indian Telegu in the UK; PJL: Punjabi in Lahore, Pakistan; STU: Sri Lankan Tamil in the UK.

1. Shared drift from some ancestral population to the first divergence event.

2. An out-of-Africa "divergence event." Admixed populations ASW (African Americans in the Southwest United States) and ACB (African Caribbeans in Barbados) are appropriately represented, and populations from the Americas appear as admixtures as well.

3. Divergence between East Asian and European populations, with South Asian populations GIH (Gujarati Indians in Houston, Texas) and PJL (Punjabi in Lahore, Pakistan) skewing more towards East Asia. American populations are variously loaded, with PEL (Peruvians in Lima) showing the most European admixture and CLM and PUR (Columbians in Medellín and Puerto Ricans) showing the most East Asian (or, here, indigenous American) admixture.

4. Drift that is largely exclusive to (but does not really differentiate among) South Asian populations.

5. Drift that is primarily experienced by PEL and, to a lesser extent, MXL (Mexicans in Los Angeles, California).

6. Drift experienced by CLM, PUR, and some but not all of MXL. This factor and the previous one probably correspond to indigenous American ancestry.

7. Divergence among East Asian populations, approximately oriented along a north-south axis. At one end is JPT (Japanese in Tokyo); at the other are Southeast Asian populations CDX (Chinese Dai in Xishuangbanna) and KHV (Kinh in Vietnam). The two Han Chinese populations CHB and CHS (from, respectively, Beijing and South China) are intermediary.

8. Divergence among European populations, again oriented along a north-south axis. FIN (Finns) are at one end, while southern European populations IBS and TBI (Iberians in

Spain and Toscani in Italy) are at the other. The European populations not represented here are British (GBR) and Utahns with Northern and Western European ancestry (CEU).

9. Divergence among African populations, oriented along an east-west axis with LWK (Luhya in Webuye, Kenya) at one end and GWD and MSL (Gambians in the Western Division and Mende in Sierra Leone) at the other. The African populations not represented are the Nigerian Esan (ESN) and Yoruba (YRI) populations.

10. Drift experienced by Han Chinese populations CHB and CHS.

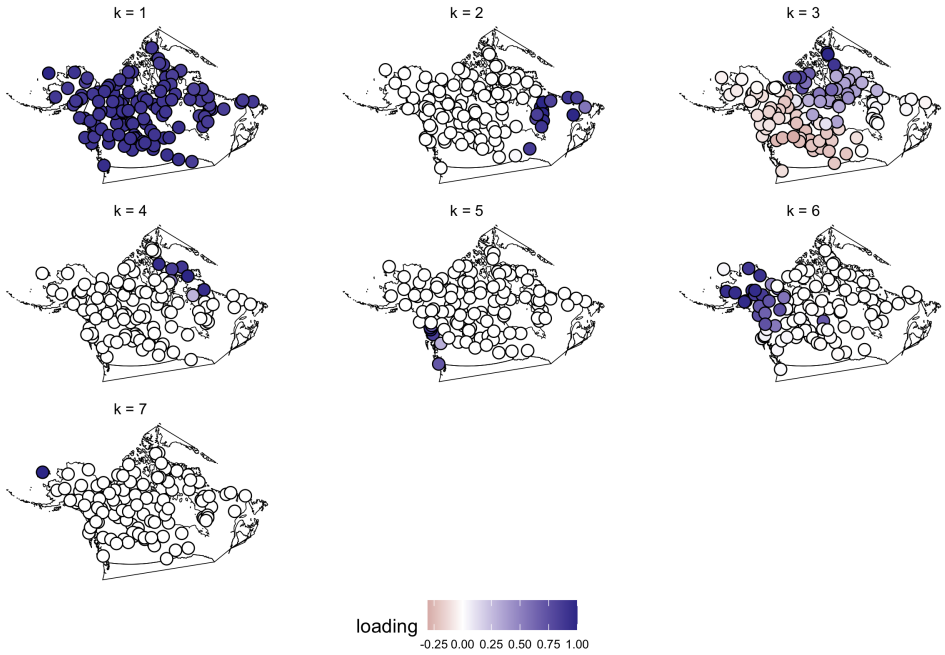
11. Noise. Only one African American individual is loaded on this factor.

North American Wolves

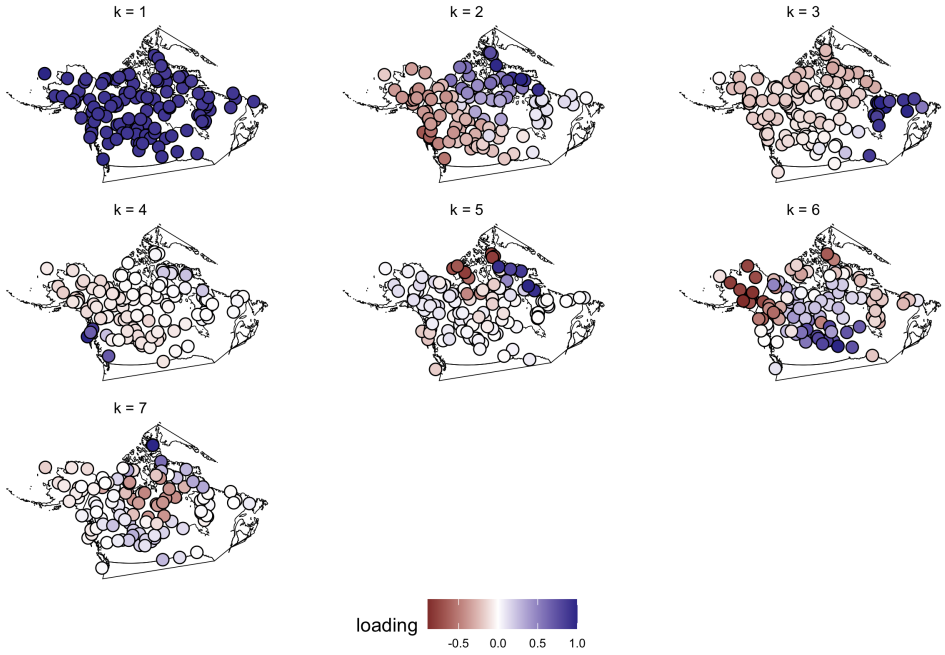
My final example considers a dataset published by Schweizer et al. [2016] consisting of 111 gray wolves from North America which were genotyped at 17,729 SNPs. For the authors, the dataset illustrates a phenomenon of isolation-by-distance, where the genetic distance between samples increases as their geographical distance increases.

As in the previous example, I choose to work with the much smaller $n \times n$ co-occurrence matrix rather than the full $n \times p$ data matrix. I fit tree-EBcovMF with a maximum tree depth of 4 for a maximum of $2^4 = 16$ factors. I again removed factors that were numerically zero, ordered factors by the proportion of variance explained, and normalized loadings so that $\|\boldsymbol{\ell}_k\|_\infty = 1$ for all k. Conveniently, the dataset includes the sample location (longitude and latitude) for each wolf, which I use to plot results in Figure 3.17. (Note that I don't use the location data in the fitting process.) In addition to loadings for the 7 factors obtained using tree-EBcovMF, I plot the top 7 eigenvectors yielded by `svd`.

Observe, first, that the tree-EBcovMF results imply that there is not strong evidence that any tree-like structure underlies the data. Apart from the first factor, which is shared



(a) EBcovMF loadings.



(b) SVD loadings.

Figure 3.17: Results from the North American gray wolves dataset from Schweizer et al. [2016].

among all wolves, and the third factor, which suggests a pattern of gradual genetic variation oriented along a northeast-southwest axis, all factors appear to correspond to isolated drift events that support the isolation-by-distance hypothesis. The second factor is loaded on a group of wolves east of the Hudson Bay; the fourth, a group on Baffin Island; the fifth, a group residing along the Pacific coast (and hemmed in by the Coast Mountains); the sixth, a population west of the Rockies in Alaska and the Yukon; and the seventh, a lone wolf residing on St. Lawrence Island.

The contrast with SVD is clear. Not only are the tree-EBcovMF loadings much cleaner than the SVD loadings, but they are free of the kinds of artefacts that are known to appear when PCA is applied to smoothly varying spatial data: see, for example, Novembre and Stephens [2008]). Notice, for example, the "saddle" pattern in the seventh SVD factor, which is not likely to correspond to any real pattern of migration.

Along with the "four-population star" simulation, these results demonstrate that tree-EBcovMF can yield highly interpretable results even in the absence of any underlying tree. Indeed, I would argue that an important criterion for any method that is tailored to tree-structured data is to not find trees where they do not in fact exist. And, as this example also works to show, while the tree framework is convenient for formalizing notions of "drift" and "divergence," these concepts remain much more widely applicable in interpreting EBMF and EBcovMF results.

3.5 Discussion

The problem of using matrix factorization methods to reveal population structure is a difficult one. To define the kind of factorization that's most useful, it's necessary to find a balance between interpretability — fairly complex information about population structure should be inferable from the factorization with minimal effort — and feasibility — the factorization should be obtainable without knowing anything about the population structure in advance.

I've proposed a "divergence factorization" as the best compromise between these two goals, but the optimization problem remains tricky. I've explored a variety of methods in order to identify the ones that I think have the best chance of giving results that are tree-like but not *too* tree-like. That is, if there is an underlying tree that strongly structures the data, I'd like it to be apparent, but I don't want to force all observations to conform to a tree-like structure when the topology is in fact more complex. To these ends, I've elaborated a pair of methods — one for full-data matrices (tree-EBMF) and one for co-occurrence matrices (tree-EBcovMF) — and I've demonstrated that they do well on both simulated and real-data examples.

I want to emphasize, however, that room for improvement remains. In particular, I've often alluded to the importance of initialization (for example, using a greedy fit with tree constraints as an initialization for an unconstrained backfit). Indeed, a good initialization is crucial because it improves the chances of finding a good local optimum. On the other hand, I've largely ignored the question of when and whether my preferred local optimum — the divergence factorization — is in fact a global optimum, or at least more optimal than less desirable solutions such as the population factorization. If the divergence factorization were globally optimal, then any shortcomings in the method could be attributed to the optimization framework; if not, then the model itself becomes problematic. I've anecdotally verified that the divergence factorization usually gives a more optimal solution than the population factorization. However, I'm not confident that better solutions don't generally exist, and if they do, then it would be useful to know what they look like. Thus an important direction for future work will be to more systematically survey the objective surface over a range of scenarios, and if necessary to refine the model to encourage solutions that have more potential to reveal population structure.

3.6 Supplementary Code Example

Using R package **flashier** (version 0.2.9), tree-EBMF and tree-EBcovMF can be implemented as follows:

```
# Initialize an all-ones factor.
ones <- matrix(1, nrow = nrow(dat), ncol = 1)
ls.soln <- t(solve(crossprod(ones), crossprod(ones, dat)))

# Add the all-ones factor and a second factor.
fl <- flash.init(dat) %>%
  flash.init.factors(list(ones, ls.soln)) %>%
  flash.fix.loadings(kset = 1, mode = 1) %>%
  flash.backfit() %>%
  flash.add.greedy(
    Kmax = 1, prior.family = c(prior.point.laplace(), prior.normal())
  )

# Start with the second factor.
current_k <- 2
K <- fl$n.factors

while(current_k <= K && K < Kmax) {
  # Partition individuals according to the sign of the current factor's
  #   loadings.
  splus <- matrix(1L * (fl$loadings.pm[[1]][, current_k] > 0), ncol = 1)
  sminus <- matrix(1L * (fl$loadings.pm[[1]][, current_k] < 0), ncol = 1)

  # Add two new factors, one for the positive loadings and one for the
  #   negative. Fix everything else at zero.
  if (sum(splus) > 0 && sum(sminus) > 0) {
    ls.soln.plus <- t(solve(crossprod(splus),
                            crossprod(splus, dat - fitted(fl))))
    ls.soln.minus <- t(solve(crossprod(sminus),
                             crossprod(sminus, dat - fitted(fl))))
    EF <- list(cbind(splus, sminus),
               cbind(ls.soln.plus, ls.soln.minus))

    fl <- fl %>%
      flash.init.factors(EF) %>%
      flash.fix.loadings(
        kset = K + 1:2, mode = 1L, is.fixed = (EF[[1]] == 0)
      ) %>%
```

```
      flash.backfit(kset = K + 1:2) %>%
      flash.nullcheck(remove = TRUE)
  }

  # Move on to the next factor.
  current_k <- current_k + 1
  K <- fl$n.factors
}

# Unfix the loadings and backfit.
fl <- fl %>%
  flash.fix.loadings(kset = 1:K, mode = 1L, is.fixed = FALSE) %>%
  flash.backfit()

# EBcovMF:
if (is_ebcovmf) {
  s2 <- max(0, mean(diag(dat) - diag(fitted(fl))))
  s2_diff <- Inf

  # Alternate between estimating s2 and backfitting until convergence.
  while(s2 > 0 && abs(s2_diff - 1) > 1e-4) {
    dat_minuss2 <- dat - diag(rep(s2, ncol(dat)))
    fl <- flash.init(dat_minuss2) %>%
      flash.init.factors(
        EF = fl$flash.fit$EF, EF2 = fl$flash.fit$EF2
      ) %>%
      flash.backfit()

    old_s2 <- s2
    s2 <- max(0, mean(diag(dat) - diag(fitted(fl))))
    s2_diff <- s2 / old_s2
  }
}
```

BIBLIOGRAPHY

1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

C. Ahlmann-Eltze and W. Huber. Transformation and preprocessing of single-cell RNA-seq data. *bioRxiv*, 2021.

A. M. S. Ang and N. Gillis. Accelerating nonnegative matrix factorization algorithms using extrapolation. *Neural Computation*, 31(2):417–439, 2019.

J. Baglama, L. Reichel, and B. W. Lewis. *irlba: Fast Truncated Singular Value Decomposition and Principal Components Analysis for Large Dense and Sparse Matrices*, 2019. R package version 2.3.3.

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2021. R package version 1.3-4.

P. Bielecki, S. J. Riesenfeld, J.-C. Hütter, E. T. Triglia, M. S. Kowalczyk, R. R. Ricardo-Gonzalez, M. Lian, M. C. A. Vesely, L. Kroehling, H. Xu, et al. Skin-resident innate lymphoid cells converge on a pathogenic effector state. *Nature*, 592(7852):128–132, 2021.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

G. S. Bradburd, G. M. Coop, and P. L. Ralph. Inferring continuous and discrete population genetic structure across space. *Genetics*, 210(1):33–52, 2018.

H. C. Bravo, S. Wright, K. Eng, S. Keles, and G. Wahba. Estimating tree-structured covariance matrices via mixed-integer programming. In *Artificial Intelligence and Statistics*, pages 41–48. PMLR, 2009.

L. D. Brown. In-season prediction of batting averages: A field test of empirical Bayes and Bayes methodologies. *Annals of Applied Statistics*, 2(1):113–152, 2008.

P. Carbonetto, A. Sarkar, Z. Wang, and M. Stephens. Non-negative matrix factorization algorithms greatly improve topic model fits. *arXiv*, 2021.

C. M. Carvalho, N. G. Polson, and J. G. Scott. The horseshoe estimator for sparse signals. *Biometrika*, 97(2):465–480, 2010.

K. Caye, F. Jay, O. Michel, and O. François. Fast inference of individual admixture coefficients using geographic data. *Annals of Applied Statistics*, 12(1):586–608, 2018.

L. H. Dicker and S. D. Zhao. High-dimensional classification via nonparametric empirical Bayes and maximum likelihood inference. *Biometrika*, 103(1):21–34, 2016.

C. H. Ding, T. Li, and M. I. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):45–55, 2010.

E. Dobriban and A. B. Owen. Deterministic parallel analysis: an improved method for selecting factors and principal components. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 81(1):163–183, 2019.

B. Efron. Microarrays, empirical Bayes and the two-groups model. *Statistical Science. A Review Journal of the Institute of Mathematical Statistics*, 23(1):1–22, 2008.

B. Efron. *Large-scale inference*, volume 1 of *Institute of Mathematical Statistics (IMS) Monographs*. Cambridge University Press, Cambridge, 2010.

B. Efron and C. Morris. Stein's estimation rule and its competitors—an empirical Bayes approach. *Journal of the American Statistical Association*, 68:117–130, 1973.

B. E. Engelhardt and M. Stephens. Analysis of population structure: A unifying framework and novel methods based on sparse factor analysis. *PLOS Genetics*, 6(9):1–12, 2010.

N. Eriksson. Tree construction using singular value decomposition. In L. Pachter and B. Sturmfels, editors, *Algebraic Statistics for Computational Biology*, pages 347–358. Cambridge University Press, Cambridge, 2005.

O. François, S. Liégeois, B. Demaille, and F. Jay. Inference of population genetic structure from temporal samples of DNA. *bioRxiv*, 2019.

S. Freytag, L. Tian, I. Lönnstedt, M. Ng, and M. Bahlo. Comparison of clustering tools in R for medium-sized 10x genomics single-cell RNA-sequencing data [version 2; peer review: 3 approved]. *F1000Research*, 7(1297), 2018.

E. Frichot, S. D. Schoville, G. Bouchard, and O. François. Correcting principal component maps for effects of spatial autocorrelation in population genetic data. *Frontiers in Genetics*, 3(254):1–9, 2012.

E. Frichot, F. Mathieu, T. Trouillon, G. Bouchard, and O. François. Fast and efficient estimation of individual ancestry coefficients. *Genetics*, 196(4):973–983, 2014.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Texts in Statistical Science Series. CRC Press, Boca Raton, FL, third edition, 2014.

D. Gerard. Data-based RNA-seq simulations by binomial thinning. *BMC Bioinformatics*, 21:1–14, 2020.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

J. Gu and R. Koenker. Empirical Bayesball remixed: empirical Bayes methods for longitudinal data. *Journal of Applied Econometrics*, 32(3):575–599, 2017.

A. N. Harman, C. R. Bye, N. Nasr, K. J. Sandgren, M. Kim, S. K. Mercier, R. A. Botting, S. R. Lewin, A. L. Cunningham, and P. U. Cameron. Identification of lineage relationships and novel markers of blood and skin human dendritic cells. *Journal of Immunology*, 190 (1):66–79, 2013.

Y. He, S. B. Chhetri, M. Arvanitis, K. Srinivasan, F. Aguet, K. G. Ardlie, A. N. Barbeira, R. Bonazzola, H. K. Im, C. D. Brown, A. Battle, and GTEx Consortium. sn-spMF: matrix factorization informs tissue-specific genetic regulation of gene expression. *Genome Biology*, 21(235):1–25, 2020.

J. R. Hershey and P. A. Olsen. Approximating the Kullback Leibler divergence between Gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages 317–320, 2007.

W. James and C. Stein. Estimation with quadratic loss. In *Berkeley Symposium on Mathematical Statistics and Probability, 1961*, pages 361–379, Berkeley, 1961. University of California Press.

W. Jiang and C.-H. Zhang. Empirical Bayes in-season prediction of baseball batting averages. In *Borrowing strength: theory powering applications—a Festschrift for Lawrence D. Brown*, volume 6 of *Institute of Mathematical Statistics Collections*, pages 263–273. Institute of Mathematical Statistics, Beachwood, OH, 2010.

I. M. Johnstone and B. W. Silverman. Needles and straw in haystacks: empirical Bayes estimates of possibly sparse sequences. *Annals of Statistics*, 32(4):1594–1649, 2004.

I. M. Johnstone and B. W. Silverman. Empirical Bayes selection of wavelet thresholds. *Annals of Statistics*, 33(4):1700–1752, 2005.

T. A. Joseph and I. Pe'er. Inference of population structure from time-series genotype data. *The American Journal of Human Genetics*, 105(2):317–333, 2019.

J. Kiefer and J. Wolfowitz. Consistency of the maximum likelihood estimator in the presence of infinitely many incidental parameters. *Annals of Mathematical Statistics*, 27:887–906, 1956.

Y. Kim, P. Carbonetto, M. Stephens, and M. Anitescu. A fast algorithm for maximum likelihood estimation of mixture proportions using sequential quadratic programming. *Journal of Computational and Graphical Statistics*, 29(2):261–273, 2020.

R. Koenker. Bayesian deconvolution: an R vinaigrette. Technical report, cemmap working paper, 2017.

R. Koenker and J. Gu. REBayes: An R package for empirical Bayes mixture methods. *Journal of Statistical Software*, 82(8):1–26, 2017.

B. Landa, T. T. Zhang, and Y. Kluger. Biwhitening reveals the rank of a count matrix. *arXiv*, 2021.

D. J. Lawson, L. Van Dorp, and D. Falush. A tutorial on how not to over-interpret structure and admixture bar plots. *Nature Communications*, 9(1):1–11, 2018.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *NIPS*, pages 556–562. MIT Press, 2000.

J. Lonsdale, J. Thomas, M. Salvatore, et al. The genotype-tissue expression (GTEx) project. *Nature Genetics*, 45(6):580–585, 2013.

A. Lun. Overcoming systematic errors caused by log-transformation of normalized single-cell RNA sequencing data. *bioRxiv*, 2018.

A. T. Lun, K. Bach, and J. C. Marioni. Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biology*, 17(1):1–14, 2016.

R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of Machine Learning Research*, 11:2287–2322, 2010.

P. McCullagh. Structured covariance matrices in multivariate regression models. Technical report, Department of Statistics, University of Chicago, 2006.

L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv*, 2018.

G. McVean. A genealogical interpretation of principal components analysis. *PLOS Genetics*, 5(10):e1000686, 2009.

D. T. Montoro, A. L. Haber, M. Biton, V. Vinarsky, B. Lin, S. E. Birket, F. Yuan, S. Chen, H. M. Leung, J. Villoria, et al. A revised airway epithelial hierarchy includes CFTR-expressing ionocytes. *Nature*, 560(7718):319–324, 2018.

S. Nakajima and M. Sugiyama. Theoretical analysis of Bayesian matrix factorization. *Journal of Machine Learning Research*, 12:2583–2648, 2011.

B. Narasimhan and B. Efron. deconvolveR: A g-modeling program for deconvolution and empirical Bayes estimation. *Journal of Statistical Software*, 94(11):1–20, 2020.

J. Novembre and M. Stephens. Interpreting principal component analyses of spatial population genetic variation. *Nature Genetics*, 40(5):646–649, 2008.

J. Novembre, T. Johnson, K. Bryc, Z. Kutalik, A. R. Boyko, A. Auton, A. Indap, K. S. King, S. Bergmann, M. R. Nelson, et al. Genes mirror geography within Europe. *Nature*, 456(7218):98–101, 2008.

J. K. Pickrell and J. K. Pritchard. Inference of population splits and mixtures from genome-wide allele frequency data. *PLOS Genetics*, 8(11):1–17, 2012.

J. K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000.

D. Risso, F. Perraudeau, S. Gribkova, S. Dudoit, and J.-P. Vert. A general and flexible method for signal extraction from single-cell RNA-seq data. *Nature Communications*, 9 (1):284, 2018.

K. Rohe and M. Zeng. Vintage factor analysis with varimax performs statistical inference. *arXiv*, 2020.

D. B. Rubin. Estimation in parallel randomized experiments. *Journal of Educational Statistics*, 6(4):377–401, 1981.

A. Sarkar and M. Stephens. Separating measurement and expression models clarifies confusion in single-cell RNA sequencing analysis. *Nature Genetics*, 53(6):770–777, 2021.

R. M. Schweizer, B. M. Vonholdt, R. Harrigan, J. C. Knowles, M. Musiani, D. Coltman, J. Novembre, and R. K. Wayne. Genetic subdivision and candidate genes under selection in North American grey wolves. *Molecular Ecology*, 25(1):380–402, 2016.

D. Sichien, C. L. Scott, L. Martens, M. Vanderkerken, S. Van Gassen, M. Plantinga, T. Joeris, S. De Prijck, L. Vanhoutte, M. Vanheerswynghels, et al. IRF8 transcription factor controls survival and function of terminally differentiated conventional and plasmacytoid dendritic cells, respectively. *Immunity*, 45(3):626–640, 2016.

C. Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, 1954–1955, vol. I*, pages 197–206. University of California Press, Berkeley and Los Angeles, 1956.

M. Stephens. False discovery rates: a new deal. *Biostatistics*, 18(2):275–294, 2017.

S. Sun, J. Zhu, Y. Ma, and X. Zhou. Accuracy, robustness and scalability of dimensionality reduction methods for single-cell RNA-seq analysis. *Genome Biology*, 20(1):1–21, 2019.

T. Tango, M. Lichtman, and A. Dolphin. *The Book: Playing the Percentages in Baseball*. Potomac Books, 2007.

F. W. Townes, S. C. Hicks, M. J. Aryee, and R. A. Irizarry. Feature selection and dimension reduction for single-cell RNA-seq based on a multinomial model. *Genome Biology*, 20(1): 1–16, 2019.

S. M. Urbut, G. Wang, P. Carbonetto, and M. Stephens. Flexible statistical methods for estimating and testing effects in genomic studies with multiple conditions. *Nature Genetics*, 51(1):187–195, 2019.

L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

S. van der Pas, J. Scott, A. Chakraborty, and A. Bhattacharya. *horseshoe: Implementation of the Horseshoe Prior*, 2019. R package version 0.2.0.

M. Wang, J. Fischer, and Y. S. Song. Three-way clustering of multi-tissue multi-individual gene expression data using semi-nonnegative tensor decomposition. *Annals of Applied Statistics*, 13(2):1103–1127, 2019.

W. Wang. *Applications of Adaptive Shrinkage in Multiple Statistical Problems*. PhD thesis, University of Chicago, Chicago, 2017.

W. Wang and M. Stephens. Empirical Bayes matrix factorization. *Journal of Machine Learning Research*, 22(120):1–40, 2021.

D. M. Witten, R. Tibshirani, and T. Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, 10(3):515–534, 2009.

J. Yan, N. Patterson, and V. Narasimhan. miqoGraph: Fitting admixture graphs using mixed-integer quadratic optimization. *bioRxiv*, page 801548, 2019.

D. Yang, Z. Ma, and A. Buja. A sparse singular value decomposition method for high-dimensional data. *Journal of Computational and Graphical Statistics*, 23(4):923–942, 2014.

L. Zappia, B. Phipson, and A. Oshlack. Splatter: simulation of single-cell RNA sequencing data. *Genome Biology*, 18(1):1–15, 2017.

S.-B. Zhang, S.-Y. Zhou, J.-G. He, and J.-H. Lai. Phylogeny inference based on spectral graph clustering. *Journal of Computational Biology*, 18(4):627–637, 2011.

G. X. Zheng, J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson, S. B. Ziraldo, T. D. Wheeler, G. P. McDermott, J. Zhu, et al. Massively parallel digital transcriptional profiling of single cells. *Nature Communications*, 8(1):1–12, 2017.