Holonomic least angle regression

Marc Härkönen

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.
Atlanta, GA, USA 18.09.2017

Thesis supervisor:

Prof. Camilla Hollanti

Thesis advisors:

Prof. Kaie Kubjas

Prof. Tomonari Sei

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Marc Härkönen

Title: Holonomic least angle regression

Date: 18.09.2017 Language: English Number of pages: 6+69

Department of Mathematics and Systems Analysis

Professorship: Mathematics

Supervisor: Prof. Camilla Hollanti

Advisors: Prof. Kaie Kubjas, Prof. Tomonari Sei

One of the main problems studied in statistic is the fitting of models. If we assume that the "true" distribution belongs to some model, we can use data and observation to find the distribution that fits the best. Ideally, we would like to explain a large dataset with as few parameters as possible. This is because models with few parameters are simpler, so computation becomes faster and interpreting the model also becomes easier. The downside is that simpler models tend to have larger errors than less simple ones.

In generalized linear models each parameter corresponds to one covariate. One might then ask which of these actually useful, or the most "impactful" in our fitting procedure, and how do we actually decide which covariates to include in the model. There have been numerous attempts at automatizing this process. Most notably, the Least Angle Regression algorithm, or LARS, by Efron et al. [2004] is a computationally efficient algorithm that ranks the covariates of a linear model. The LARS algorithm was extended by Hirose and Komaki [2010] for a class of distributions in the generalized linear model by using properties of the manifold of exponential families as dually flat manifolds. However this extension assumes that the normalizing constant of the joint distribution of observations is "easy" to compute. This is often not the case, for example the normalizing constant may contain complicated integral.

We circumvent this issue if normalizing constant satisfies a holonomic system. In this thesis we present a modification of the holonomic gradient method [Nakayama et al., 2011] and add it to the extended LARS algorithm. We call this the holonomic extended least angle regression algorithm, or HELARS. The algorithm was implemented using the statistical software `R`, and was tested with real and simulated datasets.

Keywords: statistics, information geometry, algebraic geometry, holonomic, LARS

AALTO-YLIOPISTO
PERUSTIETEIDEN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Marc Härkönen

Työn nimi: Holonomic Least Angle Regression

Päivämäärä: 18.09.2017 Kieli: Englanti Sivumäärä: 6+69

Matematiikan ja systeemianalyysin laitos

Professuuri: Matematiikka

Työn valvoja: Prof. Camilla Hollanti

Työn ohjaajat: Prof. Kaie Kubjas, Prof. Tomonari Sei

Yksi tilastotieteen tärkeimmistä ongelmista on mallien sovittaminen. Jos oletetaan "oikean" jakauman kuuluvan tiettyyn malliin, voidaan käyttää dataa ja havaintoja apuna parhaiten sopivan mallin valitsemisessa. Parhaassa tapauksessa suuri määrä dataa voidaan selittää pienellä määrällä parametreja. Tilastolliset mallit, joilla on vähäinen määrä parametreja ovat luonnostaan yksinkertaisempia, mikä ansiosta mallista tulee laskennallisesti tehokkaampi, ja mallin tulkitseminen helpottuu. Toisaalta parametrin vähentäminen yleensä lisää mallin virhettä.

Yleistetyssä lineaarisessa mallissa (*generalized linear models*) jokaista kovariaattia vastaa yksi parametri. Herää siis kysymys, mistä kovariaateista on eniten hyötyä analyysissä, ja miten voidaan päättää mitkä kovariaatit kannattaa sisällyttää malliin. Tämän prosessin automatisoimiseen löytyy lukuisia algoritmeja. Esimerkiksi Least Angle Regression algoritmi, eli LARS [Efron et al., 2004] on laskennallisesti tehokas algoritmi lineaarisen mallin kovariaattien järjestämiseen. Hirose ja Komaki [2010] laajensivat LARS algoritmin yleistetyn lineaarisen mallin alaluokkaan kuuluville malleille. Tämä laajennus kuitenkin olettaa, että jakauman normitusvakio voidaan laskea tehokkaasti. Usein asia ei ole näin, vaan normitusvakio voi sisältää esimerkiksi monimutkaisia integraaleja.

Tämä ongelma voidaan kuitenkin kiertää, jos normitusvakio on holonomisen systeemin ratkaisu. Tämän diplomityön päätuloksena esitetään holonomisen gradienttimenetelmän [Nakayama et al., 2011] modifikaatio, and ja tämä implementoidaan laajennettuun LARS algoritmiin. Lopputuloksena saadaan holonominen LARS algoritmi, eli HELARS. Algoritmi implementoitiin `R` ohjelmointikielellä, ja testattiin oikealla sekä simuloidulla datalla.

Avainsanat: tilastotiede, informaatiogeometria, algebrallinen geometria, holonominen, LARS

Preface

I want to thank to my BSc. and MSc. supervisor Professor Camilla Hollanti for her patience and support during all these years, and for giving me many priceless opportunities at the Dept. of Mathematics at Aalto.

I'm also very thankful to Prof. Kaie Kubjas for having introduced algebraic geometry, and in particular the holonomic gradient method, which led me to spend an unforgettable 10 months in Tokyo. Prof. Tomonari Sei and everyone in the 4^{th} laboratory of the Department of Mathematical Informatics also deserve my full gratitude for hosting me at the University of Tokyo and introducing me to the ups and downs of life as a graduate student in Japan. I also want to give special thanks to Prof. Bernd Sturmfels who helped me regain my focus and motivation at a time when I was overwhelmed with PhD applications.

Finally, I wish to give a very special thank you to Misaki for her constant support and giving me the opportunity to think about other things than mathematics. I wouldn't be where I am without you.

Atlanta, GA, 18.09.2017

Marc Härkönen

Contents

1 Introduction

One might argue that the most important application of statistics is fitting models to data. To make this more precise, consider a statistical model $\mathscr{P} = \{p_\xi | \xi \in \Xi\}$, and let the random variables $Y_1, \ldots, Y_n$ denote n independent observations of some probability distribution. We now wish to find the best fitting distribution in $\mathscr{P}$, which is equivalent to finding the parameter $\xi \in \Xi$ that leads to the "best fitting" distribution. One of the simplest and most powerful ways of estimating parameters ξ of a model, i.e. fitting the model parameters to observed data, is by using maximum likelihood estimation (MLE). Assume we observe the values $y_1, y_2, \ldots, y_n$. Since we assume that all of our observations are independent, the joint probability distribution function is of the form

$$f(y_1, \ldots, y_n \mid \xi) = \prod_{i=1}^{n} p_\xi(y_i).$$

The value of f at some point $\boldsymbol{y} = (y_1, \ldots, y_n)$ can be thought of as the likelihood of observing the values $\boldsymbol{y}$ given a probability distribution p_ξ. A natural question we might ask is which value of ξ maximizes $f(y_1, \ldots, y_n \mid \xi)$, or in other words which parameters make the observation the most likely. This parameter is precisely the maximum likelihood estimator

$$\hat{\xi} = \arg\max_{\xi \in \Xi} f(y_1, \ldots, y_n \mid \xi).$$

One of the main advantages of the maximum likelihood estimator is that it is consistent, i.e. it converges to the "true" parameter value as $n \to \infty$. In addition, it is quite general in that it can accommodate for probability distributions of any form.

Another question that arises commonly especially in linear or generalized linear regression regards the selection of variables to include in our model. For example in a linear model, we assume that the response Y is a random variable that depends linearly on some fixed *covariates* $x_1, x_2, \ldots, x_d$, and contains a normally distributed error term ϵ

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d + \epsilon,$$

where $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_d) \in \mathbb{R}^n$. As a concrete example we could consider the response Y as a person's salary, and the covariates as some data that may influence the salary, such as age, number of years in the workforce, education level etc. Given a dataset containing all the data and salary level for different people, we can use maximum likelihood estimation to compute the best estimate $\hat{\boldsymbol{\theta}}$. Finally, this $\hat{\boldsymbol{\theta}}$ can then be used for statistical inference, such as to predict the average salary given a person's age, work experience, etc.

Unfortunately not all covariates are created equal: some may be more "impactful", and some might not tell anything about the response. In the salary example, it is easy to imagine that the number of years in the workforce is much better at predicting salary than the color of one's car. Removing car color from the list of covariates leads to a simpler model with roughly the same predictive power. Some covariates might

also be heavily correlated, and removing one of the correlated covariates will again lead to a simpler model, but hardly affect the results.

Deciding which covariates are more useful than others is a very delicate problem, which usually involves quite a bit of intuition from the statistician doing the analysis. Ideally we want to find the best balance between model parsimony and accuracy. There are also many efforts to automatize the process of covariate selection. Given a model with k covariates, there are 2^k possible subsets of covariates. The naive method is to try regression on each of these subsets, and then choose the most suitable. The downside is that as k increases, the number of regressions becomes quickly far too large for any practical applications. Backwards elimination starts from the full model, and at each step removes the least impactful variable. Forward stepwise selection starts with an empty model (i.e. the model with no covariates), and sequentially adds the best covariate. This process is repeated until all covariates are present in the model. Both the Forward Stagewise and the recent Lasso [Tibshirani, 1996] start from the empty model, and move towards the final model in thousands of small steps. Compared to the Forward stepwise method, these are usually more careful and less greedy, but the payoff is that computation takes more time. For more details and a more complete list, the reader is referred to such books as [Wakefield, 2013] and [Hastie et al., 2009].

Closely related to the Lasso and the Forward stagewise methods is the Least Angle Regression (LARS) algorithm by Efron et al. [2004]. The algorithm creates a piecewise linear path from the empty model towards the maximum likelihood estimate of the full model. Each line segment is a diagonal of an k-dimensional cube, which is where the LARS gets its name. Because of this, the LARS is also called the bisector regression algorithm. The surprising aspect of LARS is that it is relatively efficient computationally: only k steps are needed to process all k covariates. In addition, one can think of LARS as a generalization of the Lasso and Forward stagewise methods: modifications to the LARS can implement both methods. Using the LARS implementation allows considerably faster computation of both the Lasso and the Forward stagewise method.

Unfortunately, the LARS algorithm is only defined for linear models. Hirose and Komaki [2010] presented an extension to the LARS algorithm, which implements a modification of LARS to a class of distributions of a generalized linear model. However, the assumption is that the underlying distribution in the generalized linear model is an exponential family with an "easily" computable normalizing constant. For example, the unnormalized normal distribution

$$p(y \mid \mu, \sigma^2) = e^{-\frac{(y-\mu)^2}{2\sigma^2}}, \ y \in \mathbb{R}$$

has a well known, and easily computable normalizing constant

$$\int_{\mathbb{R}} p(y \mid \mu, \sigma^2) = e^{-\frac{(y-\mu)^2}{2\sigma^2}} \, \mathrm{d}y = \frac{1}{\sqrt{2\pi\sigma^2}}.$$

On the other hand, things get more tricky with an example as simple as the truncated normal distribution, which has an unnormalized probability density function of the

form

$$p(y \mid \mu, \sigma^2) = e^{-\frac{(y-\mu)^2}{2\sigma^2}}, \ y \in [0, \infty).$$

Its normalizing constant

$$\int_0^\infty e^{-\frac{(y-\mu)^2}{2\sigma^2}} \, \mathrm{d}y$$

does not have a closed form solution, which means that we have to use numerical methods to compute it. The extended LARS algorithm needs the value of the normalizing constant in almost every step, so we would ideally want to use methods faster than numerical integration.

This motivates the holonomic bisector regression algorithm: if the normalizing constant satisfies a holonomic system, we can use a modification of the holonomic gradient method to keep track of its value at each step of the algorithm and update it when needed in a computationally efficient way.

The main result of the thesis is an implementation in `R` of the holonomic bisector regression algorithm. We choose the truncated normal distribution as the underlying distribution, as it is simple enough to handle due to its similarities with the well-known normal distribution. Despite the truncated normal having no closed from normalizing constant, our implementation of the algorithm does not use numerical integration. In Section 2 we review basic definitions and results in statistics. Section 3 concerns information geometry, and goes through definitions and results needed for the bisector regression algorithm. In particular we concentrate on studying the manifold of probability distributions in an exponential family. Section 4 presents the necessary algebra and algebraic geometry used in the algorithm. In Section 5 we present the holonomic gradient method, a computationally efficient method for computing normalizing constants introduced by Nakayama et al. [2011]. Section 6 discusses the Extended Bisector Regression algorithm by Hirose and Komaki [2010], and we look at what necessary changes and additions are needed needed for the Holonomic Extended Bisector Regression algorithm. In Section 7 we use the truncated normal distribution as the underlying distribution, and implement the Holonomic Extended Bisector Regression algorithm. We validate the algorithm using both real and simulated datasets. Finally, we end with a discussion of the results in Section 8.

2 Statistics

In this section we will review some foundations in statistics, notably the linear and generalized linear models. We will follow the book by Agresti [2015].

2.1 Linear models

Consider n independent observations $\boldsymbol{y} = [y_1, \ldots, y_n]^\intercal$ with means $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_n]^\intercal$ and a common variance σ^2. In the linear model, we assume each observation to follow a normal distribution, that is $\boldsymbol{y} \sim N(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$, where $\boldsymbol{I}$ is the $(n \times n)$ identity matrix. The *covariance* matrix $\sigma^2 \boldsymbol{I}$ is diagonal because we assume that the observations are independent. Because of this, the distribution of $\boldsymbol{y}$ (i.e. the joint distribution of $y_1, y_2, \ldots, y_n$) is

$$p(\boldsymbol{y} \mid \boldsymbol{\mu}, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \mu_i)^2}{2\sigma^2}}. \tag{2.1}$$

Additionally, the linear model assumes that the mean μ_i of observation i is a linear combination of d *covariates* $x_{i1}, \ldots, x_{id}$ so that there exists a set of d parameters $\beta_1, \ldots, \beta_d$ such that

$$\mu_i = \sum_{j=1}^{d} x_{ij} \beta_j,$$

or equivalently in matrix form

$$\boldsymbol{\mu} = \boldsymbol{X} \boldsymbol{\beta},$$

where the $(n \times d)$ *design matrix* $\boldsymbol{X}$ is defined as $\boldsymbol{X} = (x_{ij})$. Naturally we can also assume $d \leq n$, since a model with more parameters than observations can always model the data perfectly without any error. We also usually have d considerably smaller than n, since a model with too many parameters loses much of its predictive power.

Given a design matrix $\boldsymbol{X}$ and a vector of observations $\boldsymbol{y}$, we want to find the parameters $\hat{\boldsymbol{\beta}}$ that best fit the given data. Then, under the linear model and the design matrix $\boldsymbol{X}$, we can predict the values of $\boldsymbol{y}$ as

$$\hat{\boldsymbol{\mu}} = \boldsymbol{X} \hat{\boldsymbol{\beta}}.$$

The *error* of our prediction is simply $\boldsymbol{y} - \hat{\boldsymbol{\mu}}$.

In linear modeling, the value of $\hat{\boldsymbol{\beta}}$ is usually found by minimizing the *sum of square errors*

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \mu_i)^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2,$$

where $\|\cdot\|$ is the Euclidean norm. We see that this corresponds to the maximum likelihood estimate. The log-likelihood of the joint distribution of the observations (eq. (2.1)) is

$$\log(p(\boldsymbol{y} \mid \boldsymbol{\mu}, \sigma^2)) = n \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^{n} \frac{(y_i - \mu_i)^2}{2\sigma^2},$$

and given a fixed σ^2, maximizing the log-likelihood is equivalent to minimizing $\sum_{i=1}^{n}(y_i - \mu_i)^2$.

The sum of square $L(\beta)$ is minimized when its derivatives in terms $\beta_1, \ldots, \beta_d$ are all simultaneously zero. Using properties of the Euclidean norm, we can write

$$\begin{aligned} L(\beta) &= \|\boldsymbol{y} - \boldsymbol{X}\beta\|^2 \\ &= (\boldsymbol{y} - \boldsymbol{X}\beta)^T(\boldsymbol{y} - \boldsymbol{X}\beta) \\ &= \|\boldsymbol{y}\|^2 - 2\boldsymbol{y}^T\boldsymbol{X}\beta + \beta^T\boldsymbol{X}^T\boldsymbol{X}\beta. \end{aligned}$$

We set the derivative with regards to β to zero to get

$$\frac{\partial L}{\partial \beta} = 2\boldsymbol{X}^T\boldsymbol{X}\beta - 2\boldsymbol{X}^T\boldsymbol{y} = 0,$$

which gives us the estimate

$$\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

If $\boldsymbol{X}$ is of full rank, i.e. rank d, the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is invertible and we have a unique $\hat{\beta}$.

The linear model is a simple, yet powerful model. Finding the maximum likelihood estimate requires only simple matrix operations, which is computationally efficient. However, the assumption that the response is a linear combination of the covariates if often too restrictive. Also, the assumption that the variance is the same for all observations is not always correct either. In Section 2.3 we will study the generalized linear model, which includes the linear model as a special case. The notion of an exponential family is essential in the study of generalized linear models, and we will be using exponential families and their properties later in Section 3 when talking about information geometry.

2.2 Exponential family

Consider a set of independent observations $\boldsymbol{y} = (y_1, \ldots, y_n)$, where each y_i has a probability density of the form

$$p(y_i \mid \theta_i, \phi) = e^{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)}. \tag{2.2}$$

This is called the *exponential dispersion family*, where θ_i is called the *natural parameter*, and ϕ is called the *dispersion parameter*. The *natural exponential family*

arises when $a(\phi) = 1$ and $c(y_i, \phi) = c(y_i)$, in this case we denote the probability density function as

$$p(y_i \mid \theta_i) = \frac{a(y_i)e^{y_i\theta_i}}{Z(\theta)}, \tag{2.3}$$

where $Z(\theta)$ is the *normalizing constant*

$$Z(\theta_i) = \int a(y_i)e^{y_i\theta_i}\,\mathrm{d}y_i,$$

and the integral is taken over the sample space. This assures that the integral of $f(y_i;\theta_i)$ over the sample space is equal to 1, so that f is a probability density function.

Theorem 2.1. *The mean and variance of a single observation from the exponential dispersion family are*

$$\mu_i = \mathrm{E}[y_i] = b'(\theta_i)$$
$$\mathrm{var}(y_i) = b''(\theta_i)a(\phi)$$

Proof. Since $p(y_i \mid \theta_i, \phi)$ is a probability distribution, we have $\int p(y_i \mid \theta_i, \phi)\,\mathrm{d}y_i = 1$. Differentiating both sides, we get

$$\begin{aligned}
0 &= \frac{\mathrm{d}}{\mathrm{d}\theta_i}\int p(y_i \mid \theta_i, \phi)\,\mathrm{d}y_i \\
&= \int \frac{\mathrm{d}}{\mathrm{d}\theta_i} e^{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)}\,\mathrm{d}y_i \\
&= \int \frac{y_i - b'(\theta_i)}{a(\phi)} e^{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)}\,\mathrm{d}y_i \\
&= \frac{1}{a(\phi)}\int y_i e^{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)}\,\mathrm{d}y_i - \frac{b'(\theta_i)}{a(\phi)}\int e^{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)}\,\mathrm{d}y_i \\
&= \frac{\mathrm{E}[y_i]}{a(\phi)} - \frac{b'(\theta_i)}{a(\phi)},
\end{aligned}$$

hence $\mathrm{E}[y_i] = b'(\theta_i)$.

For the variance, note that

$$\begin{aligned}
\frac{\mathrm{d}^2 p}{\mathrm{d}\theta_i^2} &= \frac{\mathrm{d}}{\mathrm{d}\theta_i}\left(\frac{y_i - b'(\theta_i)}{a(\phi)}p\right) \\
&= -\frac{b''(\theta_i)}{a(\phi)}p + \left(\frac{y_i - \mu_i}{a(\phi)}\right)^2 p.
\end{aligned}$$

Now similarly

$$\begin{aligned}
0 &= \frac{\mathrm{d}^2}{\mathrm{d}\theta_i^2} \int p(y_i \mid \theta_i, \phi)\ \mathrm{d}y_i \\
&= \int \frac{\mathrm{d}^2}{\mathrm{d}\theta_i^2} p(y_i \mid \theta_i, \phi)\ \mathrm{d}y_i \\
&= -\frac{b''(\theta_i)}{a(\phi)} \int p(y_i \mid \theta_i, \phi)\ \mathrm{d}y_i + \frac{1}{a(\phi)^2} \int (y_i - \mu_i)^2 p(y_i \mid \theta_i, \phi)\ \mathrm{d}y_i \\
&= -\frac{b''(\theta_i)}{a(\phi)} + \frac{1}{a(\phi)^2} \operatorname{var}(y_i),
\end{aligned}$$

and hence $\operatorname{var}(y_i) = b''(\theta_i)a(\phi)$. □

Example 2.2. The *Poisson distribution* has the probability density function

$$p(k \mid \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}.$$

The Poisson distribution is a member of the exponential family. We can rewrite the pdf as

$$\begin{aligned}
p(k \mid \lambda) &= e^{k \log \lambda - \lambda - \log k!} \\
&= e^{k\theta - e^\theta - \log k!},
\end{aligned}$$

where the natural parameter $\theta = \log \lambda$. Using the notation in Equation 2.2, we have

$$a(\phi) = 1, \quad b(\theta) = e^\theta, \quad c(k, \phi) = -\log k!,$$

and we can check that the mean and variances are

$$\begin{aligned}
\mu &= b'(\theta) = e^\theta = \lambda, \\
\sigma^2 &= a(\phi) b''(\theta) = 1 \cdot e^\theta = \lambda,
\end{aligned}$$

as expected.

Example 2.3. The *normal distribution* is also a member of the exponential family. It has the probability density function

$$\begin{aligned}
p(y \mid \mu, \sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2} \right] \\
&= \exp\left[-\frac{y^2 - 2y\mu + \mu^2}{2\sigma^2} - \log \sqrt{2\pi\sigma^2} \right] \\
&= \exp\left[\frac{y\mu - \frac{1}{2}\mu^2}{\sigma^2} - \log \sqrt{2\pi\sigma^2} - \frac{y^2}{2\sigma^2} \right].
\end{aligned}$$

Note that here the natural parameter is $\theta = \mu$ and the dispersion parameter is $\phi = \sigma^2$. Using the notation in Equation 2.2, we have

$$a(\phi) = \phi, \quad b(\theta) = \frac{1}{2}\theta^2, \quad c(y, \phi) = -\log\sqrt{2\pi\sigma^2} - \frac{y^2}{2\sigma^2}.$$

The expectation and variance are

$$\mathrm{E}[y] = b'(\theta) = \theta = \mu,$$

$$\mathrm{var}(y) = b''(\theta)a(\phi) = 1 \cdot \phi = \sigma^2.$$

Definition 2.4. The *Fisher information matrix* of a model $p(y \mid \xi)$ at a point ξ is an $(n+r) \times (n+r)$ matrix $G(\xi) = (g_{i,j})$ defined by

$$g_{i,j} = E\left[\frac{\partial \log p(y \mid \xi)}{\partial \xi^i} \frac{\partial \log p(y \mid \xi)}{\partial \xi^j} \,\middle|\, \xi\right]$$

Corollary 2.5. *We may write the elements of the Fisher information matrix as*

$$g_{i,j} = -E\left[\frac{\partial^2}{\partial \xi^i \partial \xi^j} \log p(y \mid \xi) \,\middle|\, \xi\right]$$

Proof. Note that $E\left[\frac{\partial \log p(y|\xi)}{\partial \xi^i} \,\middle|\, \xi\right] = 0$, since

$$\begin{aligned} E\left[\frac{\partial \log p(y \mid \xi)}{\partial \xi^i} \,\middle|\, \xi\right] &= \int \frac{\partial \log p(y \mid \xi)}{\partial \xi^i} p(y \mid \xi)\,\mathrm{d}y \\ &= \int \frac{\partial}{\partial \xi^i} p(y \mid \xi)\,\mathrm{d}y \\ &= \frac{\partial}{\partial \xi^i} \int p(y \mid \xi)\,\mathrm{d}y \\ &= \frac{\partial}{\partial \xi^i} 1 = 0. \end{aligned}$$

Now apply $\frac{\partial}{\partial \xi^j}$ to both sides of $E\left[\frac{\partial \log p(y|\xi)}{\partial \xi^i} \,\middle|\, \xi\right] = 0$ to get

$$\begin{aligned} &\frac{\partial}{\partial \xi^j} E\left[\frac{\partial \log p(y \mid \xi)}{\partial \xi^i} \,\middle|\, \xi\right] = 0 \\ \iff &\frac{\partial}{\partial \xi^j} \int \frac{\partial \log p(y \mid \xi)}{\partial \xi^i} p(y \mid \xi)\,\mathrm{d}y = 0 \\ \iff &\int \left[\frac{\partial^2 \log p(y \mid \xi)}{\partial \xi^i \partial \xi^j} p(y \mid \xi) + \frac{\partial \log p(y \mid \xi)}{\partial \xi^i} \frac{\partial p(y \mid \xi)}{\partial \xi^j}\right] \mathrm{d}y = 0 \\ \iff &\int \left[\frac{\partial^2 \log p(y \mid \xi)}{\partial \xi^i \partial \xi^j} p(y \mid \xi) + \frac{\partial \log p(y \mid \xi)}{\partial \xi^i} \frac{\partial \log p(y \mid \xi)}{\partial \xi^j} p(y \mid \xi)\right] \mathrm{d}y = 0 \\ \iff &E\left[\frac{\partial^2 \log p(y \mid \xi)}{\partial \xi^i \partial \xi^j} \,\middle|\, \xi\right] E\left[\frac{\partial \log p(y \mid \xi)}{\partial \xi^i} \frac{\partial \log p(y \mid \xi)}{\partial \xi^j} \,\middle|\, \xi\right] = 0 \end{aligned}$$

□

In Section 3 we will use the following slightly more general definition for a natural exponential family. Let $\boldsymbol{y} \in \mathbb{R}$, $\boldsymbol{\xi} \in \Xi \subset \mathbb{R}^d$, $\boldsymbol{F}\colon \mathbb{R}^n \to \mathbb{R}^d$ and $C\colon \mathbb{R}^n \to [0,\infty)$. Then

$$p(\boldsymbol{y} \mid \boldsymbol{\xi}) = e^{C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}) - \psi(\boldsymbol{\xi})}, \tag{2.4}$$

where $\psi(\boldsymbol{\xi})$ is the logarithm of the normalizing constant. This definition allows the use of more than one parameter at a time.

Example 2.6. Consider a random variable $Y \sim \Gamma(\alpha, \beta)$ distributed according to the Gamma distribution with shape parameter α and rate β, where $\alpha, \beta > 0$. The probability density function is

$$p(y \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} e^{-\beta y},$$

where $\Gamma(\alpha) := \int_0^\infty y^{\alpha-1} e^{-y} \, \mathrm{d}y$. The gamma distribution belongs to the exponential family, since

$$\begin{aligned} p(y \mid \alpha, \beta) &= \exp\left(-\beta y + (\alpha - 1)\log y + \alpha \log \beta - \log \Gamma(\alpha)\right) \\ &= \exp\left(\xi_1 y + \xi_2 \log y - (\log \Gamma(\xi_2 + 1) - (\xi_2 + 1)\log -\theta_2)\right) \\ &= \exp\left(\boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}) - \psi(\boldsymbol{\xi})\right), \end{aligned}$$

with

$$\begin{aligned} \xi_1 &= -\beta & \xi_2 &= \alpha - 1 \\ \boldsymbol{\xi} &= (\xi_1, \xi_2)^T & \boldsymbol{F}(y) &= (y, \log y)^T \\ \psi(\boldsymbol{\xi}) &= (\log \Gamma(\xi_2 - 1) - (\xi_2 - 1)\log -\theta_2) \end{aligned}$$

2.3 Generalized linear models

For n observations, let y_i denote the ith response value, and $\boldsymbol{x^i} = (x_1^i, \ldots, x_d^i)$ denote the covariate vector corresponding to response i. Additionally, for each $i = 1, \ldots, n$, we assume that y_i is independent of all other y_j, and is y_i distributed according to an exponential dispersion family

$$p(y_i \mid \xi_i, \phi) = e^{\frac{y_i \xi_i - b(\xi_i)}{a(\phi)} + c(y_i, \phi)}$$

As we saw in Theorem 2.1, the expectation of each observation is $\mu_i := E[y_i] = b'(\xi_i)$.

Next, we assume that the covariate vector $\boldsymbol{x_i}$ influences the distribution of y_i only via the *linear predictor* η_i, defined as

$$\eta_i := \sum_{j=1}^{d} \theta_j x_j^i,$$

or using matrices

$$\boldsymbol{\eta} := \boldsymbol{X}\boldsymbol{\theta}$$

We can also add an intercept term by defining a new design matrix $\tilde{\boldsymbol{X}}$

$$\boldsymbol{\eta} := \begin{bmatrix} \mathbf{1}_{\boldsymbol{n}\times\mathbf{1}} & X \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} = \tilde{\boldsymbol{X}}\boldsymbol{\theta},$$

for some vector $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_d)^T$. In the following, we will always use an intercept term.

The final piece of a generalized linear model, the *link function* $g(\mu)$, determines in which way the linear predictor influences the distribution by setting

$$\eta_i = g(\mu_i).$$

The *canonical link* is the link function g for which $\xi_i = \eta_i$ for all $i = 1, 2, \ldots, n$. This happens when $g = b'^{-1}$, since

$$\eta_i = g(\mu_i) = b'^{-1}(b'(\xi_i)) = \xi_i.$$

The combination of an exponential family, design matrix and link function define a *generalized linear model* (GLM). The model has $d+1$ parameters $\theta_0, \theta_1, \ldots, \theta_d$, that we want to estimate given the response $\boldsymbol{y}$ and design matrix $\boldsymbol{X}$. Fitting a GLM is usually more delicate than fitting a linear model. Again, the goal is to find the parameters $\theta_1, \ldots, \theta_d$ which maximize the (log-)likelihood. The log-likelihood of the joint distribution of n observations is

$$L = \sum_{i=1}^{n} L_i,$$

where $L_i = \log p(y_i \mid \xi_i, \phi) = \frac{y_i \xi_i - b(\xi_i)}{a(\phi)} + c(y_i, \phi)$. This is maximized when its derivative is 0, so

$$\begin{aligned} \frac{\partial L}{\partial \theta_k} &= \sum_{i=1}^{n} \frac{\partial L_i}{\partial \xi_i} \frac{\partial \xi_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial \eta_i} \frac{\partial \eta_i}{\partial \theta_k} \\ &= \sum_{i=1}^{n} \frac{y_i - b'(\xi_i)}{a(\phi)} \frac{a(\phi)}{\operatorname{var}(y_i)} \frac{\partial \mu_i}{\partial \eta_i} x_k^i \\ &= \sum_{i=1}^{n} \frac{(y_i - \mu_i) x_k^i}{\operatorname{var}(y_i)} \frac{\partial \mu_i}{\partial \eta_i} = 0. \end{aligned}$$

Since $\eta_i = \sum_{k=1}^{d} \theta_k x_k^i = g(\mu_i)$, whether or not the equation above is linear will depend on the link function $g(\mu)$. Since in general it is not linear, we have to resort to numerical methods to compute the MLE.

A common iterative method to find the estimate $\hat{\boldsymbol{\theta}}$ is the Newton-Raphson method. Note that we will use this method extensively in our implementation (see Section 7) for both maximum likelihood estimation and other optimization tasks. We start with an initial guess $\boldsymbol{\theta}^{(0)}$. For each $k \geq 0$, we approximate the function at the point $\boldsymbol{\theta}^{(k)}$ with a polynomial of degree 2. Finding the extremum of the approximation is easy, and we set the point reaching the extremum as the next estimate $\boldsymbol{\theta}^{(k+1)}$.

More precisely, let $\boldsymbol{\theta}^{(k)} \in \mathbb{R}^d$ be the current estimate. The Taylor expansion up to the second order term at this point is

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}^{(k)}) + \boldsymbol{u}^{(k)} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}^{(k)}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(k)})^T \boldsymbol{H}^{(k)} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(k)}),$$

where $\boldsymbol{u}^{(k)}$ and $\boldsymbol{H}^{(k)}$ are respectively the gradient and Hessian evaluated at $\boldsymbol{\theta}^{(k)}$:

$$u_i^{(k)} = \left.\frac{\partial L}{\partial \theta_i}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}} \qquad (\boldsymbol{H}^{(k)})_{ij} = \left.\frac{\partial^2 L}{\partial \theta_i \partial \theta_j}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}.$$

Setting the derivative of $\tilde{L}$ to zero yields the value of the next estimate

$$\begin{aligned} &\boldsymbol{u}^{(k)} + \boldsymbol{H}^{(k)}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(k)}) = 0 \\ \Longrightarrow\ &\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \left(\boldsymbol{H}^{(k)}\right)^{-1} \boldsymbol{u}^{(k)}. \end{aligned}$$

Given a good initial guess, the method will converge to the maximum likelihood estimate as $k \to \infty$.

3 Information geometry

The extended least angle regression algorithm by Hirose and Komaki [2010] uses the fact that we can view distributions as points on a manifold. In this section we will glance over some necessary definitions and important results relating to manifolds and exponential families as dually flat manifolds. For a more detailed discussion, see the excellent book by Amari and Nagaoka [2000].

3.1 Preliminaries

A differentiable manifold S in n dimensions can be understood as a set with one or several coordinate systems. The coordinate system $\phi\colon S \to \mathbb{R}^n$ is a one-to-one mapping that maps a point $s \in S$ to its coordinates, a vector of n real numbers. Thus we can identify every point on the manifold by a real vector of length n. Such a coordinate system is called a *global* coordinate system. We could also consider a set of *local* coordinate systems, each of which maps a subset of the manifold to an open subset of $\mathbb{R}^n$, and where each point on the manifold is in the domain of at least one of the coordinate systems. In information geometry however it is often enough to consider only global coordinate systems. Hence we have the following definition

Definition 3.1. Let S be a set. We call S a *manifold* if there exists a set of coordinate systems $\mathscr{A}$ such that

1. Each element $\phi \in \mathscr{A}$ is a one-to-one mapping from S to some open subset of $\mathbb{R}^n$

2. For all $\phi \in \mathscr{A}$, given any one-to-one mapping $\psi\colon S \to \mathbb{R}^n$, the following holds

   $$\psi \in \mathscr{A} \iff \psi \circ \phi^{-1} \text{ and } \phi \circ \psi^{-1} \text{ are } C^\infty \text{ (infinitely many times differentiable)}$$

For a point $p \in S$, we will write its coordinates as $\phi(p) = (\xi^1, \xi^2, \ldots, \xi^n)$. Note that for all $i = 1, \ldots, n$, the coordinates $\xi^i = \xi^i(p)$ are functions $S \to \mathbb{R}$, called the *coordinate functions*. Thus the $i$the coordinate function $\xi^i(p)$ is simply the ith element of the coordinate system $\phi(p)$.

Example 3.2. From this definition, we can see that probability models can be viewed as manifolds. Let S be a set of probability distributions on Ω. S is called a parametric model if

$$S = \{p_\xi \mid \xi \in \Xi\}.$$

If $\Xi \subset \mathbb{R}^n$ is an open subset, then we can characterize each distribution in S by its parameters $(\xi_1, \xi_2, \ldots, \xi_n)$. Thus the mapping $p_\xi \mapsto \xi$ is a coordinate system for the manifold.

Consider a curve $\gamma\colon I \to S$ on the manifold, where $I \subset \mathbb{R}$ is an interval, and a C^∞ function $f : S \to \mathbb{R}$.[1] The directional derivative of f along the curve γ on a point $\gamma(t)$ is

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\gamma(t)) = \sum_{i=1}^{n} \frac{\mathrm{d}\gamma^i(t)}{\mathrm{d}t} \left(\frac{\partial f}{\partial \xi^i}\right)_{\gamma(t)}.$$

In the above we shall abuse notation by writing γ^i for the ξ^i coordinate of $\gamma(t)$, i.e. $\gamma^i(t) = \xi^i(\gamma(t))$. Hence we can view the derivative of γ at point $\gamma(t)$, i.e. the *tangent vector*, as a differential operator

$$\left(\frac{\mathrm{d}\gamma}{\mathrm{d}t}\right)_{\gamma(t)} := \sum_{i=1}^{n} \frac{\mathrm{d}\gamma^i(t)}{\mathrm{d}t} \left(\frac{\partial}{\partial \xi^i}\right)_{\gamma(t)},$$

where $\left(\frac{\partial}{\partial \xi^i}\right)_{\gamma(t)}$ is the operator $f \mapsto \left(\frac{\partial f}{\partial \xi^i}\right)_{\gamma(t)}$.

The *tangent space* of point $p \in S$, denoted $T_p(S)$, is the span of tangent vectors of all curves going through p. Given a coordinate system $\boldsymbol{\xi} = (\xi^1, \ldots, \xi^n)$, we have

$$T_p(S) = \left\{ \sum_{i=1}^{n} c_i \left(\frac{\partial}{\partial \xi^i}\right)_p : \boldsymbol{c} \in \mathbb{R}^n \right\}.$$

The manifold S may also have another coordinate system $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)$. Because of property 2 in the definition of a manifold (definition 3.1), the tangent space $T_p(S)$ is also spanned by $\left(\frac{\partial}{\partial \mu_i}\right)_p$, $1 \le i \le n$. In other words, the tangent space is invariant under coordinate transformations. The change of coordinates is given by an expression which resembles closely the chain rule of derivatives [Amari and Nagaoka, 2000]

$$\frac{\partial}{\partial \mu_i} = \sum_{j=1}^{n} \frac{\partial \xi^j}{\partial \mu_i} \frac{\partial}{\partial \xi^j}. \tag{3.1}$$

Definition 3.3. Let S be a manifold and $p \in S$ a point. An *inner product* is a mapping $\langle\ ,\ \rangle_p : T_p(S)^2 \mapsto \mathbb{R}$ satisfying the following for all $D, D', D'' \in T_p(S)$

1. $\langle aD + bD', D'' \rangle_p = a\langle D, D'' \rangle_p + b\langle D', D'' \rangle_p \quad (\forall a, b \in \mathbb{R})$
2. $\langle D, D' \rangle_p = \langle D', D \rangle_p$
3. If $D \neq 0$, then $\langle D, D \rangle_p > 0$

Definition 3.4. A *Riemannian metric* is a mapping $g : p \mapsto \langle\ ,\ \rangle_p$ from a point $p \in S$ to an inner product. A pair (S, g) is called a *Riemannian manifold*.

[1] Here by a C^∞ function we mean a continuous function that is differentiable arbitrary many times.

Since the tangent space $T_p(S)$ has a linearly independent basis $\left(\frac{\partial}{\partial \xi^i}\right)_p$, we can write any element $D, D' \in T_p(S)$ as

$$D = \sum_{i=1}^{n} D^i \left(\frac{\partial}{\partial \xi^i}\right)_p$$
$$D' = \sum_{j=1}^{n} D'^j \left(\frac{\partial}{\partial \xi^j}\right)_p,$$

and by multilinearity of the inner product, we have

$$\langle D, D' \rangle_p = \sum_{i=1}^{n} \sum_{j=1}^{n} D^i D'^j \left\langle \left(\frac{\partial}{\partial \xi^i}\right)_p, \left(\frac{\partial}{\partial \xi^j}\right)_p \right\rangle_p .$$

We can see that for each $p \in S$, a Riemannian metric g corresponds to a positive definite $(n \times n)$ matrix $G(p)$ with entries

$$g_{ij}(p) = \left\langle \left(\frac{\partial}{\partial \xi^i}\right)_p, \left(\frac{\partial}{\partial \xi^j}\right)_p \right\rangle_p .$$

Conversely, if for all $p \in P$ the $(n \times n)$ matrix $G(p)$ is positive definite, this fully determines a Riemannian metric with respect to any coordinate system $\phi = [\xi^1, \ldots, \xi^n]$ [Amari and Nagaoka, 2000].

3.2 Exponential family as a dually flat manifold

The general exponential family is of the form

$$p(\boldsymbol{y} \mid \boldsymbol{\xi}) = \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}) - \psi(\boldsymbol{\xi})). \tag{3.2}$$

Using the argument in Example 3.2, the exponential family is a manifold, and its coordinate system is the natural parameter $\boldsymbol{\xi}$.

Corollary 3.5. *For a distribution in the general exponential family in eq.* (3.2)*, the Fisher information matrix is equal to the Hessian of the potential function* $\psi(\boldsymbol{\xi})$*.*

Proof. By Corollary 2.5.

$$g_{ij}(\boldsymbol{\xi}) = -E\left[\frac{\partial^2}{\partial \xi^i \partial \xi^j}(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}) - \psi(\boldsymbol{\xi})) \,\middle|\, \xi\right] = -E\left[-\frac{\partial^2}{\partial \xi^i \partial \xi^j}\psi(\boldsymbol{\xi})) \,\middle|\, \xi\right] = \frac{\partial^2 \psi(\boldsymbol{\xi})}{\partial \xi^i \partial \xi^j}$$

□

At each point $p \in S$, with coordinates $\phi(p) = \boldsymbol{\xi}$, the matrix $G = (g_{ij})$ is a symmetric, positive definite matrix, and hence it defines a Riemannian metric, with $\left\langle \left(\frac{\partial}{\partial \xi^i}\right)_p, \left(\frac{\partial}{\partial \xi^j}\right)_p \right\rangle_p := g_{ij}(\boldsymbol{\xi})$. This metric is called the *Fisher metric*, or the *information metric*.

Throughout this subsection, we will assume that $S = \{p(\boldsymbol{y} \mid \boldsymbol{\xi}) = \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}) - \psi(\boldsymbol{\xi})) \mid \boldsymbol{\xi} \in \mathbb{R}^n\}$ is the manifold of exponential families with the canonical parameter $\boldsymbol{\xi}$, and g is the Fisher metric. We define the vector $\boldsymbol{\mu}$ as the expectation parameter

$$\boldsymbol{\mu} = \boldsymbol{\mu}(\boldsymbol{\xi}) := \mathrm{E}[\boldsymbol{F} \mid \boldsymbol{\xi}].$$

It turns out that for exponential families, there is a one-to-one map between $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ [Amari and Nagaoka, 2000], meaning that $\boldsymbol{\mu}$ is another coordinate system for the manifold.

Since $\psi(\boldsymbol{\xi})$ is the logarithm of the normalizing constant, we have the relation

$$\begin{aligned}
\frac{\partial \psi}{\partial \xi^i} &= \frac{\partial}{\partial \xi^i}\left(\log \int \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{y}\right) \\
&= \frac{\frac{\partial}{\partial \xi^i}\int \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{y}}{\int \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{y}} \\
&= \frac{\int F_i(\boldsymbol{y}) \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{y}}{\int \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{y}} \\
&= E[F_i(\boldsymbol{y}) \mid \boldsymbol{\xi}] \\
&= \mu_i
\end{aligned}$$

Note that with Corollary 3.5 this implies that

$$\frac{\partial \mu_i}{\partial \xi^j} = \frac{\partial^2 \psi}{\partial \xi^i \partial \xi^j} = g_{ij}.$$

We call $\psi(\boldsymbol{\xi})$ the *potential function* of the coordinates $\boldsymbol{\xi}$. The inverse transformation is given by

$$\frac{\partial \phi(\boldsymbol{\mu})}{\partial \mu_i} = \xi^i,$$

where $\phi(\boldsymbol{\mu}) := \sum_{i=1}^n \xi^i \mu_j - \psi(\boldsymbol{\xi}) = \boldsymbol{\xi} \cdot \boldsymbol{\mu} - \psi(\boldsymbol{\xi})$.[2] Hence $\phi(\boldsymbol{\mu})$ is the potential function of the coordinates $\boldsymbol{\mu}$. Note the definition of $\phi(\boldsymbol{\mu})$ above is equivalent to $\phi(\boldsymbol{\mu})$ being equal to the negative entropy

$$\phi(\boldsymbol{\mu}) = \mathrm{E}\left[\log(p(y \mid \boldsymbol{\mu}))\right],$$

and both $\psi(\boldsymbol{\xi})$ and $\phi(\boldsymbol{\mu})$ are convex [Amari, 2016]. Again we abuse notation by denoting $p(y \mid \boldsymbol{\mu}) := p(y \mid \boldsymbol{\xi}(\boldsymbol{\mu}))$.

Definition 3.6. Let (S, g) be a Riemannian manifold and let $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ be two coordinates systems. Denote the basis elements of the tangent spaces $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ as $\partial_i := \frac{\partial}{\partial \xi^i}$ and $\partial^j := \frac{\partial}{\partial \mu_j}$ respectively. The coordinate systems $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ are said to be *mutually dual* if

$$\langle \partial_i, \partial^j \rangle = \delta_i^j$$

for all $1 \le i, j \le n$.

[2] Note that here the vectors $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ are the two coordinates of the same point $p \in S$.

Theorem 3.7. *The canonical parameter ξ and the expectation parameter μ are mutually dual.*

Proof. Since $(\partial^j)_p \in T_p(S)$, we can express it as a linear combination of the basis vectors ∂_i. We use Equation (3.1) to get the coordinate conversion

$$\partial^j = \sum_{k=1}^{n} \frac{\partial \xi^k}{\partial \mu^j} \frac{\partial}{\partial \xi^k}.$$

Hence

$$\langle \partial_i, \partial^j \rangle = \sum_{k=1}^{n} \frac{\partial \xi^k}{\partial \mu^j} \langle \partial_i, \partial_k \rangle = \sum_{k=1}^{n} \frac{\partial \xi^k}{\partial \mu^j} g_{ik} = \sum_{k=1}^{n} \frac{\partial \xi^k}{\partial \mu^j} \frac{\partial \mu_j}{\partial \xi^k} = \delta_i^j.$$

□

As a summary, we now have a pair of dual coordinate systems ξ and μ, with a pair of potential functions $\psi(\xi)$ and $\phi(\mu)$. The coordinates are related through the derivative of the potential functions, that is $\frac{\partial \psi(\xi)}{\partial \xi} = \mu$ and $\frac{\partial \phi(\mu)}{\partial \mu} = \xi$. These transformations are called *Legendre transformations*.

From Amari [2001], we have the following definition

Definition 3.8. A manifold S is said to be *exponential flat*, or *e-flat*, if there is a coordinate system ξ such that for all $1 \le i, j, k \le n$

$$E\left[\frac{\partial^2}{\partial \xi^i \partial \xi^j} \log(p(y \mid \xi)) \frac{\partial}{\partial \xi^k} \log(p(y \mid \xi)) \,\middle|\, \xi \right] = 0$$

We will show that the manifold defined by the general exponential family (eq. (3.2)) is e-flat with regards to ξ. Indeed, since

$$\begin{aligned} \frac{\partial^2}{\partial \xi^i \partial \xi^j} \log(p(y \mid \xi)) &= -\frac{\partial^2 \psi(\xi)}{\partial \xi^i \partial \xi^j} = -g_{ij} \\ \frac{\partial}{\partial \xi^k} \log(p(y \mid \xi)) &= F_k(y) - \frac{\partial \psi(\xi)}{\partial \xi^k} = F_k(y) - \mu_k, \end{aligned}$$

we get

$$E\left[\frac{\partial^2}{\partial \xi^i \partial \xi^j} \log(p(y \mid \xi)) \frac{\partial}{\partial \xi^k} \log(p(y \mid \xi)) \,\middle|\, \xi \right] = E\left[-g_{ij}(F_k(y) - \mu_k) \,\middle|\, \xi \right] = -g_{ij}(\mu_k - \mu_k) = 0.$$

We say that ξ is the *e-affine* coordinate.

Definition 3.9. A manifold S is said to be *mixture-flat*, or *m-flat*, if there is a coordinate system μ such that for all $1 \le i, j, k \le n$

$$E\left[\frac{1}{p(y \mid \mu)} \frac{\partial^2}{\partial \mu^i \partial \mu^j} p(y \mid \mu) \frac{\partial}{\partial \mu^k} \log(p(y \mid \mu)) \,\middle|\, \mu \right] = 0.$$

In an exponential family, the expectation parameter μ satisfies the equation above [Amari and Nagaoka, 2000], and so $\boldsymbol{\mu}$ is called the *m-affine* coordinate.

Definition 3.10. The *Kullbach-Liebler divergence* between two probability distributions with densities $p(\boldsymbol{y})$ and $q(\boldsymbol{y})$ is defined as

$$D_{\mathrm{KL}}(p,q) := \int p(\boldsymbol{y}) \log \frac{p(\boldsymbol{y})}{q(\boldsymbol{y})} \, \mathrm{d}y.$$

In the manifold S consisting of distribution from the general exponential family (3.2), define the divergence between points $p = \boldsymbol{\xi}_p \in S$ and $q = \boldsymbol{\xi}_q \in S$ as the Kullbach-Liebler divergence

$$D(p,q) := D_{\mathrm{KL}}(p(\boldsymbol{y} \mid \boldsymbol{\xi}_p), p(\boldsymbol{y} \mid \boldsymbol{\xi}_q)) = \int p(\boldsymbol{y} \mid \boldsymbol{\xi}_p) \log \frac{p(\boldsymbol{y} \mid \boldsymbol{\xi}_p)}{p(\boldsymbol{y} \mid \boldsymbol{\xi}_q)} \, \mathrm{d}y$$

As an immediate consequence of this definition, we have a simpler expression for the divergence.

Proposition 3.11. *Let S be the manifold consisting of distributions in the general exponential family (3.2). Let the point $p, q \in S$ have the e-affine coordinates $\boldsymbol{\xi}_p$ and $\boldsymbol{\xi}_q$, and the m-affine coordinates $\boldsymbol{\mu}_p$ and $\boldsymbol{\mu}_q$ respectively. Then*

$$D(p,q) = \phi(\boldsymbol{\mu}_p) + \psi(\boldsymbol{\xi}_q) - \boldsymbol{\mu}_p \cdot \boldsymbol{\xi}_q.$$

Proof. By direct calculation

$$\begin{aligned}
D(p,q) &= \int p(\boldsymbol{y} \mid \boldsymbol{\xi}_p) \log \frac{p(\boldsymbol{y} \mid \boldsymbol{\xi}_p)}{p(\boldsymbol{y} \mid \boldsymbol{\xi}_q)} \, \mathrm{d}y \\
&= \int p(\boldsymbol{y} \mid \boldsymbol{\xi}_p) \Big(\boldsymbol{F}(\boldsymbol{y}) \cdot \boldsymbol{\xi}_p - \psi(\boldsymbol{\xi}_p) - \boldsymbol{F}(\boldsymbol{y}) \cdot \boldsymbol{\xi}_q + \psi(\boldsymbol{\xi}_q) \Big) \, \mathrm{d}y \\
&= \boldsymbol{\xi}_p \cdot \mathrm{E}[\boldsymbol{F}(\boldsymbol{y}) \mid \boldsymbol{\xi}_p] - \psi(\boldsymbol{\xi}_p) - \boldsymbol{\xi}_q \cdot \mathrm{E}[\boldsymbol{F}(\boldsymbol{y}) \mid \boldsymbol{\xi}_p] + \psi(\boldsymbol{\xi}_q) \\
&= \boldsymbol{\xi}_p \cdot \boldsymbol{\mu}_p - \psi(\boldsymbol{\xi}_p) - \boldsymbol{\xi}_q \cdot \boldsymbol{\mu}_p + \psi(\boldsymbol{\xi}_q) \\
&= \phi(\boldsymbol{\mu}_p) + \psi(\boldsymbol{\xi}_q) - \boldsymbol{\mu}_p \cdot \boldsymbol{\xi}_q,
\end{aligned}$$

where the last equality comes from the fact that $\phi(\boldsymbol{\mu}) = \boldsymbol{\xi} \cdot \boldsymbol{\mu} - \psi(\boldsymbol{\xi})$. □

Definition 3.12. Let S be a manifold with e-affine coordinates $\boldsymbol{\xi}$ and m-affine coordinates $\boldsymbol{\mu}$ and let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$. An *e-geodesic* is a curve lying on S of the form

$$\boldsymbol{\xi}(t) = t\boldsymbol{a} + \boldsymbol{b}, \; t \in \mathbb{R}.$$

Similarly, and *m-geodesic* is a curve lying on S of the form

$$\boldsymbol{\mu}(t) = t\boldsymbol{a} + \boldsymbol{b}, \; t \in \mathbb{R}.$$

Essentially, geodesics are images of straight lines in $\mathbb{R}^n$ under the appropriate coordinate system. Let $p, q \in S$ be two points on the manifold S, and let $\boldsymbol{\xi}(p)$ and $\boldsymbol{\xi}(q)$ be their e-affine coordinates. Then, the e-geodesic connecting p and q is

$$\boldsymbol{\xi}(t) = t\boldsymbol{\xi}(p) + (1-t)\boldsymbol{\xi}(q),$$

and similarly for the m-geodesic.

Let l_e be an e-geodesic and let l_m be an m-geodesic. If the geodesics intersect at some point $p \in S$, and the two tangent vectors at this point are orthogonal, we say that the geodesics are orthogonal. The following is a direct consequence from Theorem 3.7

Corollary 3.13. *For $i \neq j$, the ith e-coordinate curve and jth m-coordinate curve (the curves $\boldsymbol{\xi}(t) = t\xi_i$ and $\boldsymbol{\mu}(t) = t\mu_j$ respectively) are orthogonal.*

We also get a simple characterization of orthogonal curves.

Corollary 3.14. *Let l_e be an e-geodesic going through points $A, B \in S$ and let l_m be an m-geodesic going through points $B, C \in S$. Then l_e and l_m are orthogonal if*

$$\sum_{i=1}^{n} (\xi_i(A) - \xi_i(B))(\mu_i(C) - \mu_i(B)) = 0$$

Proof. The geodesics can be parametrized as

$$\begin{aligned} l_e(t) &= t\boldsymbol{\xi}(A) + (1-t)\boldsymbol{\xi}(B) \\ l_m(t) &= t\boldsymbol{\mu}(C) + (1-t)\boldsymbol{\mu}(B), \end{aligned}$$

and their tangent vectors at point B are

$$\begin{aligned} \frac{\mathrm{d}l_e}{\mathrm{d}t} &= \sum_{i=1}^{n} (\xi^i(A) - \xi^i(B))\partial_i \\ \frac{\mathrm{d}l_m}{\mathrm{d}t} &= \sum_{j=1}^{n} (\mu_j(C) - \mu_j(B))\partial^j, \end{aligned}$$

respectively. From the definition, the curves are orthogonal if

$$\begin{aligned} 0 &= \left\langle \frac{\mathrm{d}l_e}{\mathrm{d}t}, \frac{\mathrm{d}l_m}{\mathrm{d}t} \right\rangle \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} (\xi^i(A) - \xi^i(B))(\mu_j(C) - \mu_j(B)) \underbrace{\langle \partial_i, \partial^j \rangle}_{\delta_i^j} \\ &= \sum_{i=1}^{n} (\xi^i(A) - \xi^i(B))(\mu_i(C) - \mu_i(B)) \end{aligned}$$

□

Let T be a submanifold of S such that for any e-geodesic connecting two points $p, q \in T$ lies in T. Then we call T an e-flat submanifold. We naturally have an equivalent definition for m-flat submanifold. We will not state the definition of a submanifold here, but the reader is referred to Appendix A or Amari and Nagaoka [2000] for a precise definition of a submanifold and a proof of the two propositions below. For this thesis, it is enough to know the following

Proposition 3.15. *Let $I \subset \{1,2,\dots,n\}$ be a nonempty set, and let $\{c_i\}_{i\in I}$ be a set of size $|I|$ of constants indexed by elements of I. The subspace fixing coordinates $i \in I$, that is*

$$M = \{\boldsymbol{\xi} \mid \xi^i = c_i \ \forall i \in I\},$$

is an $n - |I|$ dimensional submanifold of S with a coordinate system $(\xi^j)_{j\notin I}$. Furthermore, it is a dually flat manifold.

Proposition 3.16. *Let $\boldsymbol{X}$ be an $(n \times d)$ matrix of rank d, where $d < n$. The subspace*

$$M = \{\boldsymbol{\xi} \mid \boldsymbol{\xi} = \boldsymbol{X}\boldsymbol{\theta},\ \boldsymbol{\theta} \in \mathbb{R}^d\},$$

is a d dimensional submanifold on S with coordinate system $\boldsymbol{\theta}$. Furthermore, it is a dually flat manifold.

Let M be an e-flat submanifold of S and let $p \in S$. The m-projection $\bar{p} \in M$ is the projection of p on M along the m-geodesic orthogonal to M. Using Corollary 3.14, we see that the m-projection $\bar{p}$ satisfies

$$\sum_{i=1}^{n} (\mu_i(p) - \mu_i(\bar{p}))(\xi^i - \xi^i(\bar{p})) = 0.$$

for all $\boldsymbol{\xi} = (\xi^1,\dots,\xi^n) \in M$. In the bisector regression algorithm, we will be doing m-projections onto the submanifolds that fix some of the components (as in Proposition 3.15). Using the same notation as Proposition 3.15, let $I \subset \{1,2,\dots,n\}$, $\{c_i\}_{i\in I}$ and

$$M = \{\boldsymbol{\xi} \mid \xi^i = c_i \ \forall i \in I\}.$$

Now choose any $p \in P$. For all $\boldsymbol{\xi} \in M$ the m-projection satisfies

$$\begin{aligned}
0 &= \sum_{i=1}^{n} (\mu_i(p) - \mu_i(\bar{p}))(\xi^i - \xi^i(\bar{p})) \\
&= \sum_{i\in I} (\mu_i(p) - \mu_i(\bar{p}))\underbrace{(\xi^i - \xi^i(\bar{p}))}_{=c_i - c_i} + \sum_{i\notin I} (\mu_i(p) - \mu_i(\bar{p}))(\xi^i - \xi^i(\bar{p})) \\
&= \sum_{i\notin I} (\mu_i(p) - \mu_i(\bar{p}))(\xi^i - \xi^i(\bar{p}))
\end{aligned}$$

Since ξ^i is arbitrary when $i \notin I$, we conclude that $\mu_i(\bar{p}) = \mu_i(p)$. Hence the point $\bar{p}$ determined uniquely by a mix of e-affine and m-affine coordinates. It is the unique point that has c_i as its $\boldsymbol{\xi}$ coordinates in positions $i \in I$, and has $\mu_j(p)$ as its $\boldsymbol{\mu}$ coordinates in positions $j \notin I$. For example when $n = 3$ and $I = \{1,2\}$, the m-projection of point p to the submanifold $M = \{\boldsymbol{\xi} \mid \xi^1 = 0, \xi^2 = 2\}$ is determined by the *mixed coordinates*

$$p = (\xi^1 = 0, \xi^2 = 2, \mu_3 = \mu_3(p)).$$

4 Algebra

In this section we will review the basic definitions and results in algebraic geometry. We will concentrate mostly on topics required for the holonomic gradient method in Section 5.

4.1 Polynomial rings

Throughout this thesis we will be working over polynomial rings over a field K. In practice, K will either be the set of rational numbers $\mathbb{Q}$, the set of real numbers $\mathbb{R}$ or the set of complex numbers $\mathbb{C}$.

Definition 4.1. A *monomial* in the variables $x_1,\dots,x_n$ is a product of the form

$$x_1^{\alpha_1}\cdot x_2^{\alpha_2}\cdots x_n^{\alpha_n},$$

where each α_i is a non-negative integer. The *degree* of this monomial is $\sum_{i=1}^n \alpha_i$. A *term* is a monomial together with a non-zero *coefficient*.

Remark. We may also write the exponents of a monomial as a *multidegree*. Let $\alpha=(\alpha_1,\dots,\alpha_n)\in\mathbb{N}^n$ be an n-tuple. Then we denote

$$x^\alpha = x_1^{\alpha_1}\cdot x_2^{\alpha_2}\cdots x_n^{\alpha_n}.$$

The degree of this monomial is denoted as $|\alpha|=\sum_{i=1}^n \alpha_i$.

Definition 4.2. A *polynomial* f in the variables $x_1,\dots,x_n$ and coefficients in k is a finite linear combination

$$f=\sum_{\alpha\in\mathscr{I}} a_\alpha x^\alpha,$$

where $\mathscr{I}$ is a finite set of multidegrees and $a_\alpha\in K$ for all $\alpha\in\mathscr{I}$.

The degree of f, denoted $\deg(f)$, is equal to the degree of the largest term

$$\deg(f)=\max_{\alpha\in\mathscr{I}}|\alpha|$$

Example 4.3. The coefficient of the term $\frac{3}{2}x_1x_3^2$ is $\frac{3}{2}$ and its degree is $1+2=3$. The degree of the polynomial $2+3x_1^4x_2-6x_1x_2^3x_3^2$ is 6.

We denote the set of all polynomials in $x_1,\dots,x_n$ with coefficients in the field K by $K[x_1,\dots,x_n]$. This set is in fact a ring: it is an abelian group with relation to addition, a monoid with relation to multiplication, and addition and multiplication are distributive. The additive and multiplicative identities of K serve as the the additive and multiplicative identities of $K[x_1,\dots,x_n]$. The ring $K[x_1,\dots,x_n]$ is called the *polynomial ring*.

In ring theory, one of the most important structures is the *ideal*.

Definition 4.4. Let R be a ring. A subset $I\subseteq R$ is an *ideal* of R if

1. $(I,+)$ is a subgroup of $(R,+)$, and
2. for all $r \in R$, $f \in I$ the product $rf \in I$.

Definition 4.5. Let R be a ring. An ideal $I \in R$ is said to be *generated* by the set $\{f_\lambda : \lambda \in \Lambda\}$ when

$$I = \left\{ \sum_{\lambda \in \Lambda} g_\lambda f_\lambda \;\middle|\; \text{a finite number of } g_\lambda \text{ are nonzero} \right\}$$

We denote such an ideal $I = \langle \{f_\lambda : \lambda \in \Lambda\} \rangle$.

In the polynomial ring, we have a rather natural way to define *divisibility* between two polynomials: $f \in K[x_1,\ldots,x_n]$ is divisible by $g \in K[x_1,\ldots,x_n]$ if there is some $q \in K[x_1,\ldots,x_n]$ such that $f = qg$. We write $g \mid f$ when g divides f. In the case of monomials $u = x^\alpha$, $v = x^\beta$, where α and β are n-tuples, we see that $u \mid v$ if and only if $\alpha_i \leq \beta_i$ for all $i = 1,\ldots,n$. Let M be a (possibly infinite) set of monomials. We will show that M has a finite set of *minimal* elements. A monomial $m \in M$ is said to be minimal the following condition is satisfied: when $u \in M$ is a monomial such that $u \mid m$, then $u = m$.

Theorem 4.6 (Dickson's lemma)**.** *Let M be a set of monomials in $K[x_1,\ldots,x_n]$. The set of minimal elements is at most finite.*

Proof. We will use induction on the number of variables n. Let $\mathscr{M}_n$ be the set of all monomials in n variables.

$n = 1$ M contains elements of the form x_1^k where $k \in \mathbb{N}$. Clearly the monomial with the smallest degree divides every other monomial, and it is the unique monomial with this property, hence it is the unique minimal element.

$n = 2$ Denote the minimal elements by $u_1 = x_1^{a_1} x_2^{b_1}, u_2 = x_1^{a_2} x_2^{b_2}, \ldots$, with $a_1 < a_2 < \cdots$. Note that we can only have strict inequalities here, because if we had $a_i = a_{i+1}$ for some i, then we would have either $u_i \mid u_{i+1}$ or $u_{i+1} \mid u_i$. Due to minimality, we must also have $b_1 > b_2 > \cdots$. Since b_1 is a non-negative integer, we must have a finite number of minimal elements.

Induction step Assume the lemma holds for $n-1$ variables. Denote $y = x_n$. We may write any element $m \in \mathscr{M}_n$ as uy^a, where $u \in \mathscr{M}_{n-1}$. Let N be the set of monomials

$$N = \left\{ u \;\middle|\; uy^b \in M \text{ for some } b \geq 0 \right\}.$$

By the induction hypothesis, the set N has a finite number of minimal elements $u_1,\ldots,u_s$, and for each u_i there is a b_i such that $u_i y^{b_i} \in M$. Let b be the largest of the b_i. Now for all $0 \leq c < b$, let N_c be the set of monomials

$$N_c = \left\{ u \;\middle|\; uy^c \in M \right\}.$$

Again by the induction hypothesis, each N_c has a finite set of generators $u_1^{(c)},\dots,u_{s_c}^{(c)}$.

Choose any monomial $w = uy^e \in M$. If $e \geq b$, then u is divisible by u_i for some $i = 1,\dots,s$ and hence $u_i y^{b_i}$ divides w. On the other hand, if $e < b$, then u is divisible by some $u_i^{(e)}$ for some $i = 1,\dots,s_e$, and hence $u_i y^e$ divides w. In other words, any monomial $w \in M$ is divided by one of the following

$$\begin{gathered} u_1 y^{b_1},\dots,u_s y^{b_s} \\ u_1^{(0)},\dots,u_{s_0}^{(0)} \\ u_1^{(1)} y,\dots,u_{s_1}^{(1)} y \\ \dots \\ u_1^{(b-1)} y^{b-1},\dots,u_{s_{b-1}}^{(b-1)} y^{b-1}. \end{gathered}$$

Any minimal monomial of M has to appear in the list above, so the set of minimal monomials has to be finite. □

Let I be an ideal generated by monomials $\{u_\lambda : \lambda \in \Lambda\}$. Such an ideal is called a *monomial ideal*. Any element in $w \in I$ can be expressed as

$$w = \sum_{\lambda\in\Lambda} g_\lambda u_\lambda,$$

where at most a finite number of g_λ are non-zero.

The next result will be needed when dealing with monomial ideals and Gröbner bases in Section 4.2.

Corollary 4.7. *Every nonempty monomial ideal $I \subset K[x_1,\dots,x_n]$ is finitely generated, and has a unique minimal set of monomial generators.*

Proof. Let u_λ, where $\lambda \in \Lambda$, be the set of generators of I. By Dickson's lemma, the set $\{u_\lambda : \lambda \in \Lambda\}$ has a finite set of minimal elements $u_1,\dots,u_s$. We claim that these elements generate I. Clearly $\langle u_1,\dots,u_s\rangle \subset I$. Let $w \in I$. Now for each $\lambda \in \Lambda$, the monomial u_λ is divisible by one of the minimal elements, say u_{i_λ}, where $1 \leq i_\lambda \leq s$. Hence we can express w as

$$w = \sum_{\lambda\in\Lambda} g_\lambda u_\lambda = \sum_{\lambda\in\Lambda} g_\lambda \frac{u_\lambda}{u_{i_\lambda}} u_{i_\lambda},$$

which implies that $w \in \langle u_1,\dots,u_s\rangle$. Hence $I = \langle u_1,\dots,u_s\rangle$.

For uniqueness, assume that $u_1,\dots,u_s$ and $v_1,\dots,v_r$ are both minimal sets of generators of I. Fix any $i = 1,\dots,s$. By minimality, there is some v_j that divides u_i. Furthermore, there is some u_k that divides v_j. Since we have $v_j \mid u_i$ and $u_k \mid v_j$ then we must have $u_k \mid u_j$. By minimality $u_k = u_i = v_j$, and hence $\{u_1,\dots,u_s\} \subset \{v_1,\dots,v_r\}$. By a similar argument we also have $\{u_1,\dots,u_s\} \subset \{v_1,\dots,v_r\}$, hence the equality. □

In the polynomial ring with one variable $K[x]$, ordering each term of a polynomial is easy, since we can just look at the degree of each term. This makes long division of

polynomials easy, since we start with the highest order term. Long division can also be done with polynomials of more than one variable. With two or more variables however it is not as clear cut which of the terms are of "highest degree". We introduce the concept of monomial ordering to answer this question.

Definition 4.8. A *partial order* on a set Σ is a relation $\leq$ on Σ such that for all $x, y, z \in \Sigma$ we have

1. $x \leq x$ (reflexivity)
2. $x \leq y$ and $y \leq x$ if and only if $x = y$ (antisymmetry)
3. $x \leq y$ and $y \leq z$ implies $x \leq z$ (transitivity)

A *total order* on a set Σ is a partial order $\leq$ with for every pair $x, y \in \Sigma$ either $x \leq y$ or $y \leq x$.

Definition 4.9. A *monomial ordering* on $K[x_1, \ldots, x_n]$ is a relation $<$ on the set of monomials $\mathcal{M}_n$ satisfying the following

1. $<$ is a total order
2. if $u, v \in \mathcal{M}_n$ and $u < v$, then $uw < vw$ for all $w \in \mathcal{M}_n$
3. $1 < u$ for all $1 \neq u \in \mathcal{M}_n$

Remark. Note that condition 3 in Definition 4.9 is equivalent to $<$ being a *well ordering*, that is every decreasing sequence in $\mathcal{M}_n$

$$\alpha(1) > \alpha(2) > \alpha(3) > \cdots$$

eventually terminates.

Example 4.10. The *graded lexicographic order* $<_{\text{grlex}}$ orders monomials based on the degree first, and then in lexicographical order. Let $u = x^\alpha$ and $v = x^\beta$ be monomials in $\mathcal{M}_n$, with $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$. We set $u <_{\text{grlex}} v$ if (a) $|\alpha| < |\beta|$, or (b) $|\alpha| = |\beta|$ and the leftmost non-zero element of the $(\alpha_1 - \beta_1, \ldots, \alpha_n - \beta_n)$ is negative. Under lexicographical ordering we have for example $x_1^2 x_3^4 <_{\text{grlex}} x_2^7$ and $x_1^2 x_2^2 x_3^3 <_{\text{grlex}} x_1^2 x_2^4 x_3$. The lexicographic order is indeed a monomial order [Cox et al., 2007, p. 57].

Next we will define some common terminology, much of which will be used in Subsection 4.2 about Gröbner bases.

Definition 4.11. Let $f = \sum_\alpha a_\alpha x^\alpha \in K[x_1, \ldots, x_n]$ be a non-zero polynomial, and let $<$ be a monomial order.

(a) The *multidegree* of f, denoted multideg(f) is the degree of the maximal monomial appearing in f

$$\text{multideg}(f) = \max(\alpha \in \mathbb{Z}^n_{\geq 0} : a_\alpha \neq 0),$$

where the maximum is taken with regard to $<$.

(b) The *leading coefficient* is

$$\mathrm{LC}(f) = a_{\mathrm{multideg}(f)} \in K$$

(c) The *leading monomial* is

$$\mathrm{LM}(f) = x^{\mathrm{multideg}(f)} \in \mathscr{M}_n$$

(d) The *leading term* is

$$\mathrm{LT}(f) = a_{\mathrm{multideg}(f)} x^{\mathrm{multideg}(f)}$$

4.2 Gröbner bases

We will first consider the polynomial ring in one variable $K[x]$.

Theorem 4.12 (Division algorithm)**.** *Let $p, q \in K[x]$. There are unique polynomials u (the* quotient*) and r (the* remainder*) such that*

$$p = uq + r,$$

where $\deg(r) < \deg(q)$.

Let I be an ideal generated by the set of polynomials $\{f_\lambda : \lambda \in \Lambda\}$. It is well known that the polynomial ring is a *principal ideal domain*, meaning that every ideal is generated by exactly one polynomial. This polynomial is the greatest common divisor of the generators. Using the division algorithm, we can deduce whether a polynomial belongs to the ideal or not: we divide our polynomial by the generator of the ideal, and deduce that the polynomial belongs to the ideal if and only if the remainder is 0.

In more than one variable, thing are more complicated. First of all, the polynomial ring in more than one variable is not a principal ideal ring. As an example, let $I = \langle x, y \rangle$ in $K[x, y]$. If there was a single polynomial f generating I, then we could write $x = gf$ for some polynomial g. Then f cannot contain any terms in y. On the other hand, we can write $y = g'f$ for some polynomial g', but then f cannot contain any terms in x. Hence f must be 1, but we see that $\langle 1 \rangle \neq I$.

The division algorithm works also in the multivariate case

Theorem 4.13 (Division algorithm)**.** *Let $f \in K[x_1, \ldots, x_n]$ and $F = (f_1, \ldots, f_k)$, where each $f_i \in K[x_1, \ldots, x_n]$ and let $<$ be a monomial order. There are polynomials $u_1, \ldots, u_k$ and r such that*

$$f = u_1 f_1 + \cdots + u_k f_k + r,$$

where $r = 0$ or r is a linear combination in K of monomials, none of which are divisible by $\mathrm{LM}(f_1), \ldots, \mathrm{LM}(f_k)$.

In the univariate case, we can determine if a polynomial belongs to the ideal by dividing the polynomial by the generator of the ideal and checking the remainder. In the multivariate case, things are a bit more complicated. For example, let $K[x,y]$ be our polynomial ring, $<$ be the graded lex order, and let $\langle xy, x+y\rangle$ be an ideal. Then, applying the division algorithm on the polynomial $f = y^2$ yields

$$f = 0 \cdot xy + 0 \cdot (x+y) + y^2,$$

but we know y^2 belongs to the ideal, since $y^2 = y(x+y) - xy$. Another downside of division in $K[x_1,\ldots,x_n]$ is that the result depends on the order in which we divide, i.e. dividing by f_1 first and the f_2 will in general result in a different looking expression than first dividing by f_2 and then f_1.

Definition 4.14. Let $K[x_1,\ldots,x_n]$ be a polynomial ring, $<$ a monomial ordering and $I \subset K[x_1,\ldots,X_n]$ a nonzero ideal. The *Gröbner basis* of I is a finite set $g_1, g_2, \ldots, g_s$ such that $\mathrm{LT}(g_1), \mathrm{LT}(g_2), \ldots, \mathrm{LT}(g_s)$ generate the monomial ideal

$$\mathrm{LT}(I) := \langle \{\mathrm{LT}(f) \colon 0 \neq f \in I\}\rangle.$$

Dickson's lemma (Theorem 4.6) guarantees that a Gröbner basis exists. It turns out that the Gröbner basis also generates the ideal, that is

$$\mathrm{LT}(I) = \langle \mathrm{LT}(g_1), \mathrm{LT}(g_2), \ldots, \mathrm{LT}(g_s)\rangle \implies I = \langle g_1, g_2, \ldots, g_s\rangle.$$

This result implies Hilbert's Basis Theorem

Theorem 4.15 (Hilbert's Basis Theorem)**.** *Every ideal of the polynomial ring is finitely generated.*

An other property of Gröbner bases is that dividing a polynomial by a Gröbner basis results in a unique remainder. Furthermore, a polynomial belongs to an ideal if and only if the residual when dividing the polynomial by the Gröbner basis is 0. *Reduced* Gröbner bases are unique for each ideal, so they can be used to check if two ideals are equal. For more details and algorithms regarding Gröbner bases, see for example Cox et al. [2007].

4.3 Differential operator rings

In this section we will study *differential operator rings*, which can be thought of as polynomial rings with $2n$ "variables": $x_1,\ldots,x_n$ denote the regular variables of a polynomial ring, and $\partial_1,\ldots,\partial_n$ denote the partial differential operator of each variable x_i.

Let $\mathbb{C}(x_1,\ldots,x_n)$ be the ring of rational functions

$$\mathbb{C}(x_1,\ldots,x_n) = \left\{ \frac{p}{q} \;\middle|\; p,q \in \mathbb{C}[x_1,\ldots,x_n], q \neq 0 \right\}.$$

In this section we will study two rings R_n and D_n of differential operators.

Definition 4.16. The ring of differential operators with rational function coefficients, denoted R_n, is

$$R_n = C(x_1, \ldots, x_n)\langle \partial_1, \ldots, \partial_n \rangle,$$

where the operator ∂_i corresponds to differentiation with relation to x_i, i.e.

$$\partial_i = \frac{\partial}{\partial x_i},$$

and the operators x_i just multiply by x_i.

Note that the "multiplication" operation inside the ring is actually a composition of operators. Since every element in R_n is an operator, there is a natural action on the set of C^∞ functions, denoted by $\bullet$. If $f(x_1, \ldots, x_n) \in C^\infty$, then

$$\partial_i \bullet f(x_1, \ldots, x_n) = \frac{\partial f}{\partial x_i}$$
$$x_i \bullet f(x_1, \ldots, x_n) = x_i f(x_1, \ldots, x_n).$$

R_n is not a commutative, since

$$\partial_i x_i = x_i \partial_i + 1,$$

which can be viewed as the chain rule of derivatives, since

$$\partial_i x_i \bullet f = \partial_i \bullet (x_i f) = \frac{\partial (x_i f)}{\partial x_i} = x_i \frac{\partial f}{\partial x_i} + f = (x_i \partial_i + 1) \bullet f.$$

When $i \neq j$, everything commutes:

$$\partial_i \partial_j = \partial_j \partial_i$$
$$x_i x_j = x_j x_i$$
$$x_i \partial_j = \partial_j x_i.$$

Because of these commuting rules, any element in R_n can be written as a sum of terms with the δ_i on the right of each term:

$$\sum_\alpha c(x_1, \ldots, x_n) \partial^\alpha,$$

where α is a multi-index, $\partial^\alpha = \partial_1^{\alpha_1} \partial_2^{\alpha_2} \cdots \partial_n^{\alpha_n}$ and $c_\alpha \in \mathbb{C}(x_1, \ldots, x_n)$ with only finitely many nonzero c_α.

Example 4.17. Let $R_2 = \mathbb{C}(x, y)\langle \partial_x, \partial_y \rangle$. If $p = x \partial_x^2 y \partial_y x$, we can rewrite it as

$$\begin{aligned} p &= x \partial_x^2 y \partial_y x \\ &= xy \partial_y \partial_x (x \partial_x + 1) \\ &= xy \partial_y \partial_x x \partial_x + xy \partial_y \partial_x \\ &= xy \partial_y (x \partial_x + 1) \partial_x + xy \partial_x \partial_y \\ &= x^2 y \partial_x^2 \partial_y + 2xy \partial_x \partial_y. \end{aligned}$$

The equivalent of a monomial order in R_n is called a term order. In fact, any monomial order of $K[x_1,\dots,x_n]$ can be viewed as an ordering of $\mathbb{Z}^n_{\geq 0}$, and it induces a term order on R_n defined by

$$a(x)\partial^\alpha < b(x)\partial^\beta \iff \alpha < \beta.$$

It turns out that the arguments used to build the theory of Gröbner bases in $K[x_1,\dots,x_n]$ also work in R_n with minor modifications. In particular, all properties concerning Gröbner bases mentioned in Section 4.2 also hold in R_n. An interested reader is referred to Hibi [2013] for details.

The next theorem, which is also true in $K[x_1,\dots,x_n]$, will reveal some information about the structure of the quotient R_n/I.

Theorem 4.18 (Macaulay's theorem)**.** *Let $I \subset R_n$ be an ideal. The set of* standard monomials

$$\{w \in R_n \mid w \text{ is a monomial}, \ w \notin \mathrm{LT}(I)\}$$

is a basis of R_n/I as a vector space over R_n.

We say I is 0-dimensional, if there are finitely many standard monomials. A necessary and sufficient condition for I to be 0-dimensional is that for all i the following holds

$$I \cap \mathbb{C}(x_1,\dots,x_n)\langle\partial_i\rangle \neq \{0\}.$$

Example 4.19. In this example, let $R_2 = C(x,y)\langle\partial_x,\partial_y\rangle$. The ideal $I = \langle\partial_x^2 - 1\rangle$ is clearly not 0-dimensional, since ∂_y^k is a standard monomial for all $k \in \mathbb{N}$, or alternatively $I \cap \mathbb{C}(x,y)\langle\partial_y\rangle = \{0\}$, because every element in I contains a term in ∂_x. On the other hand $J = \langle\partial_x^2 - 1, \partial_y^3\rangle$ is a 0-dimensional ideal: the standard monomials are precisely 1, ∂_x, ∂_y, $\partial_x\partial_y$, ∂_y^2, and $\partial_x\partial_y^2$.

Next we will present another ring of differential operators

Definition 4.20. The ring of differential operators with polynomial coefficients D_n, also known as the *Weyl algebra*, is defined as

$$D_n = \mathbb{C}\langle x_1,\dots,x_n,\partial_1,\dots,\partial_n\rangle.$$

Note that we have the same commutation relations as in R_n, so any element in $p \in D_n$ can be written as

$$p = \sum_{\alpha,\beta} c_{\alpha\beta} x^\alpha \partial^\beta,$$

where at most finitely many $c_{\alpha,\beta}$ are nonzero. The ring D_n is in general more useful in algorithms, but it does not behave as consistently as R_n. For example, in D_n we

will often use a term order that depends on a weight vector $(\boldsymbol{u}, \boldsymbol{v})$ and a monomial order $<$:

$$x^{\alpha}\partial^{\beta} <_{(\boldsymbol{u},\boldsymbol{v})} x^{\alpha'}\partial^{\beta'} \iff \alpha\cdot\boldsymbol{u}+\beta\cdot\boldsymbol{v} < \alpha'\cdot\boldsymbol{u}+\beta'\cdot\boldsymbol{v} \text{ or}$$
$$\alpha\cdot\boldsymbol{u}+\beta\cdot\boldsymbol{v} = \alpha'\cdot\boldsymbol{u}+\beta'\cdot\boldsymbol{v} \text{ and } x^{\alpha}\partial^{\beta} < x^{\alpha'}\partial^{\beta'}.$$

Such an order may fail to be a well-order: for example in D_1, if $(u,v) = (-1,1)$, we have an infinite descending chain

$$x >_{(u,v)} x^2 >_{(u,v)} x^3 >_{(u,v)} \dots$$

Details on Gröbner bases in D_n can be found in Saito et al. [2000]. In D_n we also have the notion of a *holonomic ideal*, which corresponds to the notion of a 0-dimensional ideal in R_n. The exact definition is omitted here; we refer the reader to Hibi [2013] for a more thorough discussion. Instead, we state two theorems relating holonomic ideals and 0-dimensional ideals.

Theorem 4.21. *Let I be a holonomic ideal in D_n. Then the ideal R_nI is a 0-dimensional ideal in R_n.*

Theorem 4.22. *Let J be a 0-dimensional ideal in R_n. Then the ideal $J \cap D_n$ is a holonomic ideal in D_n.*

5 Holonomic gradient method

In this section we will describe the holonomic gradient method (HGM), first proposed by Nakayama et al. [2011]. Consider first the "classical" gradient descent algorithm, which is used to find a local minimum of a function $F\colon \mathbb{R}^n \to \mathbb{R}$. Given a starting point (or initial guess) $\boldsymbol{x^{(0)}}$, we know that the value of the function decreases the fastest in the direction opposite to the gradient. In other words, we should choose the next point

$$\boldsymbol{x^{(1)}} = \boldsymbol{x^{(0)}} - \gamma_1 \nabla F(\boldsymbol{x^{(0)}}), \tag{5.1}$$

for some step $\gamma_1 > 0$. Now, given a suitably chosen stepsize γ_1, we have $F(\boldsymbol{x^{(1)}}) < F(\boldsymbol{x^{(0)}})$. We then iterate

$$\boldsymbol{x^{(k+1)}} = \boldsymbol{x^{(k)}} - \gamma_{k+1} \nabla F(\boldsymbol{x^{(k)}}), \tag{5.2}$$

while choosing a suitable stepsize $\gamma_k > 0$ at each iteration $k \in \mathbb{N}$. We can terminate the algorithm when the gradient is small enough (i.e. when we are close to a local minimum), or when a certain number of iterations have elapsed. Issues concerning the choice of step size and efficiency of this method will not be discussed here; details can be found in any standard numerical optimization textbook, see for example the book [Ascher and Greif, 2011].

The issue with this method is that it requires computing the gradient $\nabla F(\boldsymbol{x^{(k)}})$ at each step. In many statistical applications, the function we want to optimize will be a likelihood function, which will in some cases contain an integral that does not have a closed form solution, and has to be computed using numerical methods. One such example is the 1-dimensional truncated normal distribution, which has the probability density function

$$p(x) = \frac{e^{-(x^2-\mu)^2/(2\sigma^2)}}{\int_0^\infty e^{-(x^2-\mu)^2/(2\sigma^2)}\,\mathrm{d}x}.$$

Note that while the normalizing constant of the normal distribution is easy to compute ($\int_{-\infty}^{\infty} e^{(x-\mu)^2/(2\sigma^2)}\,\mathrm{d}x = \sqrt{2\pi\sigma^2}$), the normalizing constant of the *truncated* normal distribution $\int_0^\infty e^{-(x^2-\mu)^2/(2\sigma^2)}\,\mathrm{d}x$ does not have a closed form solution.

The holonomic gradient method takes a different approach to function minimization. The main idea is still the same: we use the same iterative step as in the classical gradient descent

$$\boldsymbol{x^{(k+1)}} = \boldsymbol{x^{(k)}} - \gamma_{k+1} \nabla F(\boldsymbol{x^{(k)}}). \tag{5.3}$$

The difference is how we compute the gradient. We will construct a vector $\boldsymbol{Q}$ and compute matrices $\boldsymbol{P_i}$ to form a *Pfaffian system*

$$\frac{\partial \boldsymbol{Q}}{\partial x_i} = \boldsymbol{P_i Q}. \tag{5.4}$$

The vector $\boldsymbol{Q}$ will be chosen so that the gradient $\nabla F(\boldsymbol{x^{(k)}})$ is easily recoverable, typically $\nabla F(\boldsymbol{x^{(k)}}) = A(\boldsymbol{x^{(k)}})\boldsymbol{Q}(\boldsymbol{x^{(k)}})$ for some matrix A with entries in $\mathbb{C}(x_1,\ldots,x_n)$. With the gradient, we can determine the next point $\boldsymbol{x^{(k+1)}}$ using (5.3). Given $\boldsymbol{Q}(\boldsymbol{x^{(k)}})$, the value of $\boldsymbol{Q}$ in the previous step, we can compute its value in the next step $\boldsymbol{Q}(\boldsymbol{x^{(k+1)}})$ by solving the Pfaffian system (5.4) using standard numerical ODE solvers.

Observe that we can also implement the Newton-Raphson method using the holonomic gradient framework. The update step will be

$$\boldsymbol{x^{(k+1)}} = \boldsymbol{x^{(k)}} - \mathbf{Hess}(F(\boldsymbol{x^{(k)}}))^{-1}\boldsymbol{\nabla} F(\boldsymbol{x^{(k)}}), \tag{5.5}$$

and we can also recover the Hessian easily from the Pfaffian system, since there is a matrix $\boldsymbol{B}(\boldsymbol{x^{(k)}})$ with elements in $\mathbb{C}(x_1,\ldots,x_n)$ such that $\mathbf{Hess}(F) = \boldsymbol{BQ}$

5.1 Pfaffian systems

Let $f(x_1,\ldots,x_n) \in C^\infty$ be a function, and $\ell \in R_n$. When $\ell \bullet f = 0$, we say that f is *annihilated* by ℓ. We say that f is annihilated by an ideal $I \subset R_n$ if f is annihilated by all $\ell \in I$. Observe that if $I = \langle \ell_1,\ldots,\ell_s\rangle$, f is annihilated by I if and only if f is annihilated by all ℓ_i. Assume that I is 0-dimensional, and let $s_1 = 1, s_2, \ldots, s_r$ be the standard monomials, which are the generators of R_n/I. For all $1 \le i \le n$ and $1 \le j \le r$, we can look at the image of the operator $\partial_i s_j$ in the quotient R_n/I (under the canonical map $p \mapsto p + I$), and write it as a $\mathbb{C}(x_1,\ldots,x_n)$ linear combination of the basis elements

$$\partial_i s_j = \sum_{k=1}^{r} p^i_{jk} s_k,$$

where $p^i_{jk} \in \mathbb{C}(x_1,\ldots,x_n)$ for all $1 \le i \le n$ and $1 \le j,k \le r$. Thus, if we define the vector $\boldsymbol{S} = (s_1, s_2, \ldots, s_r)^T$, then for each i, there is a matrix $\boldsymbol{P_i}$ such that

$$\partial_i \boldsymbol{S} = \boldsymbol{P_i S}, \tag{5.6}$$

with $(P_i)_{jk} = p^i_{jk}$.

Define the vector

$$\boldsymbol{Q} = \begin{bmatrix} s_1 \bullet f \\ s_2 \bullet f \\ \vdots \\ s_r \bullet f \end{bmatrix}.$$

Since f is annihilated by all elements in I, Equation (5.6) is true when we replace $\boldsymbol{S}$ by $\boldsymbol{Q}$. We get the following system of differential equations

$$\frac{\partial \boldsymbol{Q}}{\partial x_i} = \boldsymbol{P_i Q},$$

the Pfaffian system. Because we chose $s_1 = 1$, the gradient of f can be recovered from the first elements of each Pfaffian system

$$\nabla f = \begin{bmatrix} (\boldsymbol{P}_1 \boldsymbol{Q})_1 \\ (\boldsymbol{P}_2 \boldsymbol{Q})_2 \\ \vdots \\ (\boldsymbol{P}_n \boldsymbol{Q})_n \end{bmatrix}$$

By following the procedure above, one can construct a Pfaffian system given a 0-dimensional ideal annihilating our function f. This is indeed desirable, since finding a 0-dimensional annihilating ideal is often easier than to find a Pfaffian system from scratch. One noteworthy fact is that if f has a holonomic annihilating ideal, then its integral over one variable $\int f \, \mathrm{d}x_i$ also has a holonomic annihilating ideal. This is extremely useful when using the holonomic gradient method in maximum likelihood estimation, since the normalizing constant will usually contain an integral. Oaku [1997] describes an algorithm for computing the (holonomic) annihilating ideal of an integral.

6 Bisector regression

In this section, we will describe the extended least angle regression algorithm [Hirose and Komaki, 2010]. We will also compute explicit forms for many of the expressions found in Section 3, such as the coordinate conversion functions, Fisher information matrix, and divergence for the manifold used in Hirose and Komaki [2010].

Consider a set of observed data $\{y_a, \boldsymbol{x}^a = (x_1^a, \ldots, x_d^a) \mid a = 1, 2, \ldots, n\}$, where $\boldsymbol{y} = (y_1, \ldots, y_n)^T$ is the response vector, and $\boldsymbol{X} = (x_j^a)$ is the design matrix, which has dimensions $(n \times d)$. We will also add an intercept term to the model, so the design matrix becomes $\tilde{\boldsymbol{X}} = \begin{bmatrix} \mathbf{1}_{\boldsymbol{n}\times\mathbf{1}} & \boldsymbol{X} \end{bmatrix}$, where $\mathbf{1}_{\boldsymbol{n}\times\mathbf{1}}$ is a column vector of n 1's. We will consider exponential families of the form

$$p(\boldsymbol{y} \mid \boldsymbol{\xi}) = \exp\left(\sum_{a=1}^{n} y_a \xi^a + \sum_{b=1}^{r} u_b(\boldsymbol{y}) \xi^{b+n} - \psi^*(\boldsymbol{\xi}) \right). \tag{6.1}$$

We will define some notation. Let $\boldsymbol{\xi}$ be the natural parameter, a $n+r$ sized vector containing elements ξ_i. We can split $\boldsymbol{\xi}$ into two subvectors: we call $\boldsymbol{\xi}'$ the subvector containing the first n elements, i.e. $\boldsymbol{\xi}' = (\xi^a)_{a=1}^n$, and we call $\boldsymbol{\xi}''$ the subvector containing the last r elements, i.e. $\boldsymbol{\xi}'' = (\xi^b)_{b=n+1}^{n+r}$. Hence $\boldsymbol{\xi} = (\boldsymbol{\xi}', \boldsymbol{\xi}'')^T$. The function $\psi^*(\boldsymbol{\xi})$ is the potential function of $\boldsymbol{\xi}$, and it is equal to the logarithm of the normalizing constant of the distribution

$$\psi^*(\boldsymbol{\xi}) = \log \int \exp\left(\sum_{a=1}^{n} y_a \xi^a + \sum_{b=1}^{r} u_b(\boldsymbol{y}) \xi^{b+n} \right) \mathrm{d}\boldsymbol{y}$$

In a generalized linear model with canonical link function, the natural parameter is related to linear predictor by

$$\boldsymbol{\xi}' = \tilde{\boldsymbol{X}} \boldsymbol{\theta}',$$

where $\boldsymbol{\theta}' = (\theta_0, \theta_1, \ldots, \theta_d)^T$ is a parameter vector. In addition, as in Hirose and Komaki [2010], we require r additional parameters that are equal to $\boldsymbol{\xi}''$. Hence we can write $\boldsymbol{\theta}'' = \boldsymbol{\xi}''$, and define $\boldsymbol{\theta} = (\boldsymbol{\theta}', \boldsymbol{\theta}'')^T$. Equation (6.1) thus becomes

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \exp\left(\boldsymbol{y}^T \tilde{\boldsymbol{X}} \boldsymbol{\theta}' + \sum_{b=1}^{r} u_b(\boldsymbol{y}) \theta^{b+d} - \psi(\boldsymbol{\theta}) \right),$$

where the potential function of $\boldsymbol{\theta}$ is $\psi(\boldsymbol{\theta}) = \psi^*(\tilde{\boldsymbol{X}} \boldsymbol{\theta}', \boldsymbol{\theta}'')$. Alternatively, define

$$\boldsymbol{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ u_1(\boldsymbol{y}) \\ \vdots \\ u_r(\boldsymbol{y}), \end{bmatrix}$$

and an $(n+r)\times(d+r+1)$ block-diagonal matrix

$$\boldsymbol{X}_B = \begin{bmatrix} \tilde{\boldsymbol{X}} & 0_{n\times r} \\ 0_{r\times(d+1)} & \boldsymbol{I}_{r\times r}, \end{bmatrix}$$

where $0_{n\times m}$ and $\boldsymbol{I}_{n\times n}$ are respectively the $(n\times m)$ zero matrix and the $(n\times n)$ identity matrix. Then we have the identities

$$\begin{aligned} \boldsymbol{\xi} &= \boldsymbol{X}_B\boldsymbol{\theta} \\ \psi(\boldsymbol{\theta}) &= \psi^*(\boldsymbol{X}_B\boldsymbol{\theta}) \end{aligned} \tag{6.2}$$

and the probability density function becomes

$$p(\boldsymbol{y}\mid\boldsymbol{\theta}) = \exp(\boldsymbol{Y}^T\boldsymbol{X}_B\boldsymbol{\theta} - \psi(\boldsymbol{\theta})).$$

The $\boldsymbol{\xi}$ coordinate, being the natural parameter of an exponential family, is the e-affine coordinate of the model manifold. The corresponding m-affine coordinate $\boldsymbol{\mu}$ is the expectation parameter $\boldsymbol{\mu} = \mathrm{E}[\boldsymbol{Y}]$ and its potential function is defined as $\phi^*(\boldsymbol{\mu}) = \boldsymbol{\xi}\cdot\boldsymbol{\mu} - \psi^*(\boldsymbol{\xi})$. By Proposition 3.16, the model manifold defined by the coordinates $\boldsymbol{\theta}$ is a submanifold of the model defined by the $\boldsymbol{\xi}$ coordinates. The e-affine coordinate of this submanifold is $\boldsymbol{\theta}$, and there is a dual m-affine coordinate $\boldsymbol{\eta}$ which is related to $\boldsymbol{\mu}$ by

$$\boldsymbol{\eta} = \mathrm{E}[(\boldsymbol{Y}^T\boldsymbol{X}_B)^T] = \boldsymbol{X}_B^T\,\mathrm{E}[\boldsymbol{Y}] = \boldsymbol{X}_B^T\boldsymbol{\mu}. \tag{6.3}$$

By Corollary 3.5, we note that the Fisher information matrix of the model in eq. (6.1) is equal to the Hessian of the potential function $\psi^*(\boldsymbol{\xi})$, denoted $G^* = (g^*_{i,j})$. Similarly, denote the Hessian of the potential function of the m-affine coordinates $\phi^*(\mu)$ as the matrix $G_* = (g_*^{i,j})$. We see that

$$\begin{aligned} (G^*G_*)_{i,j} &= \sum_{k=1}^{n} g^*_{i,k} g_*^{i,k} \\ &= \sum_{k=1}^{n} \frac{\partial\mu_i}{\partial\xi^k}\frac{\partial\xi_k}{\partial\mu^j} \\ &= \frac{\partial\mu_i}{\partial\mu_j} = \delta_i^j, \end{aligned}$$

where

$$\delta_i^j = \begin{cases} 1, & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

is the Kroenecker delta. Hence G^* and G_* are inverse matrices of each other.

The Hessian $G(\boldsymbol{\theta}) = (g_{i,j})$ of the potential function $\psi(\boldsymbol{\theta})(= \psi^*(\tilde{\boldsymbol{X}}\boldsymbol{\theta}', \boldsymbol{\theta}''))$ can be recovered using the chain rule:

$$g_{i,j} = \frac{\partial^2 \psi(\boldsymbol{\theta})}{\partial \theta^i \partial \theta^j} \tag{6.4}$$

$$= \frac{\partial \eta_i}{\partial \theta_j} \tag{6.5}$$

$$= \sum_{a=0}^{n+d} \sum_{b=0}^{n+d} \frac{\partial \eta_i}{\partial \mu_a} \frac{\partial \mu_a}{\partial \xi_b} \frac{\partial \xi_b}{\partial \theta_j}. \tag{6.6}$$

Using the identities in eqs. (6.2) and (6.3), we see that

$$\frac{\partial \eta_i}{\partial \mu_a} = \frac{\partial (\boldsymbol{X}_B^T \boldsymbol{\mu})_i}{\partial \mu_a} = (\boldsymbol{X}_B^T)_{i,a}$$
$$\frac{\partial \xi_b}{\partial \theta_j} = \frac{\partial (\boldsymbol{X}_B \boldsymbol{\theta})_b}{\partial \theta_j} = (\boldsymbol{X}_B)_{b,j}.$$

Thus eq. (6.6) becomes

$$g_{i,j} = \sum_{a=1}^{n} \sum_{b=1}^{n} (\boldsymbol{X}_B^T)_{i,a} g^*_{a,b} (\boldsymbol{X}_B)_{b,j},$$

which implies that the $\boldsymbol{G} = \boldsymbol{X}_B^T \boldsymbol{G}^* \boldsymbol{X}_B$. We denote elements of its inverse with superscripts: $\boldsymbol{G}^{-1} = (g^{i,j})$. Similar to the previous case, we have

$$\begin{aligned} (\boldsymbol{G})_{i,j} &= \frac{\partial \eta_i}{\partial \theta^j} = g_{i,j} \\ (\boldsymbol{G}^{-1})_{i,j} &= \frac{\partial \theta^i}{\partial \eta_j} = g^{i,j} \end{aligned} \tag{6.7}$$

Remark. In subsequent sections we will make extensive use of matrix and vector differentiation. The convention in Minka [1997] will be used: the shape of $\frac{\partial f}{\partial x}$ depends either on the shape of $\boldsymbol{f}$ or the shape of $\boldsymbol{x}^T$. For example, differentiating a scalar by a length n column vector yields a length n row vector

$$\frac{\partial f}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n}. \end{bmatrix}$$

Differentiating a length m column vector by a scalar yields a length m column vector

$$\frac{\partial \boldsymbol{f}}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \\ \vdots \\ \frac{\partial f_m}{\partial x}. \end{bmatrix}$$

Finally, differentiating a length m column vector by a length n column vector yields an $(m \times n)$ matrix, the Jacobian.

$$\mathbf{Jac}(\boldsymbol{f}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

This notation allows the natural use of the chain rule for derivatives

$$\frac{\partial \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}))}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}))}{\partial \boldsymbol{g}(\boldsymbol{x})} \frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

with the usual matrix multiplication between the two terms on the right hand side. Furthermore, we can express the Hessian of a scalar valued function $f(\boldsymbol{x})$ as

$$\mathbf{Hess}(f) = \frac{\partial^2 f}{\partial \boldsymbol{x} \partial \boldsymbol{x}^T}$$

6.1 Mixed coordinate conversion

Next, consider a point P on the dually flat manifold

$$S = \{p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \exp(\boldsymbol{Y}^T \boldsymbol{X}_B \boldsymbol{\theta} - \psi(\boldsymbol{\theta})) \mid \boldsymbol{\theta} \in \mathbb{R}^{d+r+1}\}.$$

It is characterized by the e- and m-affine coordinates $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_{d+r})$ and $\boldsymbol{\eta} = (\eta_0, \eta_1, \ldots, \eta_{d+r})$. Alternatively, we may use *mixed coordinates*, i.e. for some $J \subset \{0, 1, \ldots, d+r\}$ we represent P as $(\boldsymbol{\eta}_J, \boldsymbol{\theta}^{\bar{J}})$, where $\boldsymbol{\eta}_J = \{\eta_j \mid j \in J\}$ is the subvector of $\boldsymbol{\eta}$ containing only elements which have indices in J, and $\boldsymbol{\theta}^{\bar{J}} = \{\theta^j \mid j \notin J\}$ is the subvector of $\boldsymbol{\theta}$ containing elements with indices not in J.

Let $J \subseteq \{0, 1, 2, \ldots, d+r\}$, $\overline{J} = \{0, 1, 2, \ldots, d+r\} \setminus J$ and let $P = (\boldsymbol{\eta}_J, \boldsymbol{\theta}^{\bar{J}})$ denote a mixed coordinate. Essentially, we want to recover $\boldsymbol{\eta}_{\bar{J}}$ and $\boldsymbol{\theta}^J$ given $\boldsymbol{\eta}_J$ and $\boldsymbol{\theta}^{\bar{J}}$. Let $\tilde{\boldsymbol{\theta}}(\boldsymbol{\theta}^J) = (\boldsymbol{\theta}^J, \boldsymbol{\theta}^{\bar{J}})$, a function $\mathbb{R}^{|J|} \to \mathbb{R}^{d+r+1}$ obtained by mixing the fixed $\boldsymbol{\theta}^{\bar{J}}$ and the unknown $\boldsymbol{\theta}^J$ coordinates in the positions defined by J. Thus, the function $\tilde{\boldsymbol{\theta}}$ outputs the full $\boldsymbol{\theta}$ coordinates, where $\boldsymbol{\theta}^{\bar{J}}$ are always constant, and $\boldsymbol{\theta}^J$ are allowed to vary. Similarly, let $\tilde{\boldsymbol{\eta}}(\boldsymbol{\eta}_{\bar{J}}) = (\boldsymbol{\eta}_J, \boldsymbol{\eta}_{\bar{J}})$ be the same function for the $\boldsymbol{\eta}$ coordinates.

We will use Newton's method to find the root of the function

$$F(\boldsymbol{\theta}^J, \boldsymbol{\eta}_{\bar{J}}) = \tilde{\boldsymbol{\eta}}(\boldsymbol{\eta}_{\bar{J}}) - \boldsymbol{\eta}(\tilde{\boldsymbol{\theta}}(\boldsymbol{\theta}^J)). \tag{6.8}$$

Proposition 6.1. *The Jacobian* $\mathbf{Jac}(F)$ *of* F *in eq. (6.8), has columns*

$$(\mathbf{Jac})_i = \begin{cases} -\left(G(\tilde{\theta})\right)_i & \text{if } i \in J \\ \boldsymbol{e}_i & \text{if } i \in \overline{J} \end{cases},$$

where $G(\tilde{\theta})_i$ *is the* i*th column of* $G(\tilde{\theta}) = \left.\frac{\partial^2 \psi(\boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T}\right|_{\boldsymbol{\theta}=\tilde{\boldsymbol{\theta}}}$ *(see eq. (6.7)), and the vector* $\boldsymbol{e}_i$ *is the* i*th standard basis of* $\mathbb{R}^{n+d+1}$.

Proof. Let $i \in \overline{J}$. Then

$$\frac{\partial F}{\partial \eta_i} = \frac{\partial}{\partial \eta_i} \tilde{\eta}(\eta_{\bar{J}}) - 0.$$

Since the ith element of $\tilde{\eta}(\eta_{\bar{J}})$ is simply η_i and none of the other elements depend on η_i, we have $\frac{\partial F}{\partial \eta_i} = e_i$.

Next let $i \in J$. Using the chain rule we get

$$\begin{aligned}
\frac{\partial F}{\partial \theta^i} &= 0 - \frac{\partial}{\partial \theta^i}\left(\eta(\tilde{\theta}(\theta^J))\right) \\
&= -\frac{\partial \eta(\tilde{\theta})}{\partial \tilde{\theta}} \cdot \frac{\partial \tilde{\theta}(\theta^J)}{\theta^i} \\
&= -(G(\tilde{\theta})) \cdot e_i \\
&= -(G(\tilde{\theta}))_i
\end{aligned}$$

□

Newton's method will iteratively output a vector $(\theta^J, \eta_{\bar{J}})$ with the following update step

$$(\theta^J, \eta_{\bar{J}})^{(k+1)} = (\theta^J, \eta_{\bar{J}})^{(k)} - \mathbf{Jac}(F)^{-1} F,$$

where the Jacobian and F are evaluated at $(\theta^J, \eta_{\bar{J}})^{(k)}$. Given a suitable initial guess (see section 7.4 for implementations details), the method converges very quickly.

6.2 Extended bisector regression algorithm

We describe shortly the algorithm by Hirose and Komaki [2010]. Let I be the set containing the indices of covariates present in the model. We first start with the model containing all covariates, that is $I = \{1, 2, \ldots, d\}$, and compute the maximum likelihood estimate $\hat{\theta}_{\text{MLE}}$. In addition, we also compute the maximum likelihood estimate $\hat{\theta^{\emptyset}}$ of the empty model, i.e. the model where $\theta_1, \ldots, \theta_d = 0$. We will work in the d dimensional submanifold of S

$$M = \{\eta \mid \eta_0 = \hat{\eta}_0^{\emptyset}, \eta_{d+1} = \hat{\eta}_{d+1}^{\emptyset}, \ldots, \eta_{d+r} = \hat{\eta}_{d+r}^{\emptyset}\}, \tag{6.9}$$

and set $\hat{\theta}_{(0)} := \hat{\theta}_{\text{MLE}}$ and $k = 1$.

For each $i \in I$, let $\bar{\theta}_i$ be the m-projection of the current point $\hat{\theta}_{(k)}$ to the e-flat submanifold corresponding to $\theta^i = 0$. Let i^* be the coordinate which has smallest divergence between the point $\hat{\theta}_{(k)}$ and its m-projection $\bar{\theta}_i$, and let this divergence be t^*. Now for each $i \in I$, look at the m-geodesic connecting $\hat{\theta}_{(k)}$ and $\bar{\theta}_i$, and find the point θ_i^* along that geodesic that has divergence t^* from $\hat{\theta}_{(k)}$. The estimate for the next step $\hat{\theta}_{(k+1)}$ is constructed as follows: for all $i \in I$, set the ith coordinate of $\hat{\theta}_{(k+1)}$ to the ith coordinate of θ_i^*, and for all $j \notin I$, set the jth coordinate of $\hat{\theta}_{(k+1)}$ to 0. Notice that the i^*th coordinate will also be 0. We now remove i^* from the list of

"active" covariates I, and restart at the beginning of the paragraph, this time in the submanifold

$$M_I = \{\eta \mid \eta_0 = \hat{\eta}_0^{\emptyset}, \eta_{d+1} = \hat{\eta}_{d+1}^{\emptyset}, \ldots, \eta_{d+r} = \hat{\eta}_{d+r}^{\emptyset}, \theta^j = 0 \ \forall j \notin I\}.$$

We quit the algorithm after d steps, when no covariates are left. We denote the divergence in M_I by $D^{[I]}$. It results naturally from the divergence D in M. Denote the coordinates in M_I as $\boldsymbol{\theta}_I = (\theta^i)_{i \in I}$ and $\boldsymbol{\eta}_I = (\eta_i)_{i \in I}$. The potential functions become $\psi_I(\boldsymbol{\theta}_I) = \psi(\boldsymbol{\theta}_I, 0_{\bar{I}})$ and $\phi_I(\boldsymbol{\eta}_I) = \boldsymbol{\eta}_I \cdot \boldsymbol{\theta}_I - \psi_I(\boldsymbol{\theta}_I)$. The divergence is then

$$D^{[I]}(p,q) = \phi_I(\boldsymbol{\eta}_I(p)) + \psi_I(\boldsymbol{\theta}_I(q)) - \boldsymbol{\eta}_I(p) \cdot \boldsymbol{\theta}_I(q).$$

The algorithm starts from the full model and proceeds step by step towards the empty model, which is the opposite direction compared to the LARS. Other than that, the geometric idea of the algorithm is the same as in LARS: at each step k, we move the current estimate $\hat{\boldsymbol{\theta}}^{(k)}$ towards the origin, in a direction that bisects the m-geodesics corresponding to each m-projection. We hit the next estimate $\hat{\boldsymbol{\theta}}^{(k+1)}$ exactly when the first of the coordinates $i \in I$ of the vector $\hat{\boldsymbol{\theta}}^{(k)}$ hits 0.

The following pseudocode describes the algorithm more precisely. An implementation in R can be found in Appendix B. The algorithm is described in the submanifold M of eq. (6.9), so we do not write down coordinates $0, d+1, \ldots, d+r$ explicitly. We input the data (observations and design matrix) and an underlying distribution (essentially the functions $u_1(\boldsymbol{y}), \ldots, u_r(\boldsymbol{y})$ in eq. (6.1)), and we get as an output a sequence of estimators $(\hat{\boldsymbol{\theta}}_{(0)}, \ldots, \hat{\boldsymbol{\theta}}_{(d)})$, where the estimator obtained in the kth step corresponds to a model with k covariates removed.

1. Let $I = \{1, 2, \ldots, d\}$, $\hat{\boldsymbol{\theta}}_{(0)} := \hat{\boldsymbol{\theta}}_{\mathrm{MLE}}$, and $k = 0$.

2. For all $i \in I$, let $M(i, 0, I) = \{\boldsymbol{\theta} \mid \theta^i = 0, \theta^j = 0 \ (j \notin I)\} = M(I \setminus \{i\})$ and calculate the m-projection $\overline{\boldsymbol{\theta}}(i, I)$ of $\hat{\boldsymbol{\theta}}_{(k)}$ on $M(i, 0, I)$.

3. Let $t^* = \min_{i \in I} D^{[I]}(\hat{\boldsymbol{\theta}}_{(k)}, \overline{\boldsymbol{\theta}}(i, I))$ and $i^* = \arg\min_{i \in I} D^{[I]}(\hat{\boldsymbol{\theta}}_{(k)}, \overline{\boldsymbol{\theta}}(i, I))$.

4. For every $\alpha^i \in \mathbb{R}, i \in I$, let $M(i, \alpha^i, I) = \{\theta \mid \theta^i = \alpha^i, \theta^j = 0 \ (j \notin I)\}$. For every $i \in I$, compute α^i such that the m-projection $\overline{\boldsymbol{\theta}}'(i, \alpha^i, I)$ of $\hat{\boldsymbol{\theta}}_{(k)}$ on $M(i, \alpha^i, I)$ satisfies $t^* = D^{[I]}(\hat{\boldsymbol{\theta}}_{(k)}, \overline{\boldsymbol{\theta}}'(i, \alpha^i, I))$.

5. Let $\hat{\theta}^i_{(k+1)} = \alpha^i \ (i \in I)$ and $\hat{\theta}^j_{(k+1)} = 0 \ (j \notin I)$.

6. If $k = d - 1$, then go to step 7. If $k < d - 1$, then go to step 2 with $k := k + 1$, $I := I \setminus \{i^*\}$.

7. Let $\hat{\boldsymbol{\theta}}_{(d)} = 0$. Output $\hat{\boldsymbol{\theta}}_{(0)}, \ldots, \hat{\boldsymbol{\theta}}_{(d)}$ and quit the algorithm.

We can now rank the covariates in order of importance by looking at the output of the algorithm. The zeroth estimator $\hat{\boldsymbol{\theta}}_{(0)}$ was defined as the maximum likelihood estimate of the full model containing every covariate, and at each subsequent estimator, one of the components will vanish, i.e. the kth estimator $\hat{\boldsymbol{\theta}}_{(k)}$ will have exactly k

of its elements equal to zero. The element that vanishes corresponds to the covariate that is deemed the least impactful at that particular step. Thus by looking at the order in which the covariates vanish in the sequence $\hat{\boldsymbol{\theta}}_{(0)}, \ldots, \hat{\boldsymbol{\theta}}_{(d)}$, we can order the covariates from least to most important.

6.3 Adding holonomicity

We will focus our attention to the potential function $\psi(\boldsymbol{\xi})$ in eq. (6.1), which can be written as

$$\psi^*(\boldsymbol{\xi}) = \log \int \exp\left(\sum_{a=1}^{n} y_a \xi^a + \sum_{b=1}^{r} u_b(\boldsymbol{y}) \xi^{b+n} \right) \mathrm{d}\boldsymbol{y}.$$

Whether or not $\psi^*(\boldsymbol{\xi})$ has a closed form representation depends on the underlying distribution. We could use numerical integration if no closed form expression for $\psi^*(\boldsymbol{\xi})$ exists, but such an approach is computationally inefficient.

Instead, we assume that the potential function can be written as

$$\psi^*(\boldsymbol{\xi}) = G_\psi(\boldsymbol{\xi}, \boldsymbol{L}(\boldsymbol{\xi})), \tag{6.10}$$

where G_ψ is an elementary function with easily computable derivatives, and $\boldsymbol{L}(\boldsymbol{\xi})$ is a scalar or vector valued function with a Pfaffian system

$$\frac{\partial L_i}{\partial \xi_j} = H(\boldsymbol{\xi}, \boldsymbol{L}(\boldsymbol{\xi}))_{i,j}. \tag{6.11}$$

Now we obtain the gradient of $\psi^*(\boldsymbol{\xi})$ as a function of $\boldsymbol{\xi}$ and $\boldsymbol{L}(\boldsymbol{\xi})$

$$\frac{\partial \psi^*}{\partial \boldsymbol{\xi}} = \frac{\partial G_\psi}{\partial \boldsymbol{\xi}} + \frac{\partial G_\psi}{\partial \boldsymbol{L}} \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\xi}}. \tag{6.12}$$

Here $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\xi}}$ is equal to the matrix $\boldsymbol{H}$ in eq. (6.11), and the derivatives $\frac{\partial G_\psi}{\partial \boldsymbol{\xi}}$ and $\frac{\partial G_\psi}{\partial \boldsymbol{L}}$ are easily computed by the assumption that G_ψ was elementary. We can also write Fisher information matrix, which is equal to the Hessian of $\psi^*(\boldsymbol{\xi})$ by Corollary 3.5, by differentiating eq. (6.12). Note also that since $\boldsymbol{\xi}$ is linear in $\boldsymbol{\theta}$, i.e. $\boldsymbol{\xi} = \boldsymbol{X}_B \boldsymbol{\theta}$, eq. (6.11) also gives a Pfaffian system in terms of $\boldsymbol{\theta}$ using the chain rule[3]

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\theta}} = \boldsymbol{H} \boldsymbol{X}_B, \tag{6.13}$$

Example 6.2 (Truncated normal)**.** When each observation is distributed according to the truncated normal distribution, we get a potential function of the form

$$\psi^*(\boldsymbol{\xi}) = \sum_{a=1}^{n} \log \int_0^\infty \exp\left(y \xi^a + y^2 \xi^{n+1} \right) \mathrm{d}y$$

[3]The vector $\boldsymbol{L}(\boldsymbol{\xi})$ is technically only defined for $\boldsymbol{\xi}$ coordinates. However, we will abuse notation and write $\boldsymbol{L}(\boldsymbol{\theta})$ when the meaning is clear from context. When writing $\boldsymbol{L}(\boldsymbol{\theta})$, we will do an implicit coordinate conversion, i.e. we actually mean $\boldsymbol{L}(\boldsymbol{X}_B \boldsymbol{\theta})$.

If we define

$$\boldsymbol{L}(\boldsymbol{\xi}) = \begin{bmatrix} \log \int_0^\infty \exp(y\xi^1 + y^2\xi^{n+1})\,\mathrm{d}y \\ \log \int_0^\infty \exp(y\xi^2 + y^2\xi^{n+1})\,\mathrm{d}y \\ \vdots \\ \log \int_0^\infty \exp(y\xi^n + y^2\xi^{n+1})\,\mathrm{d}y \end{bmatrix},$$

then $G_\psi(\boldsymbol{\xi}, \boldsymbol{L}(\boldsymbol{\xi})) = \sum_{a=1}^n \boldsymbol{L}(\boldsymbol{\xi})_a = \psi^*(\boldsymbol{\xi})$. By eq. (7.6) $\boldsymbol{L}$ has a Pfaffian system. See Section 7 for complete details.

6.3.1 Holonomic update of the vector L

Nearly every step of the algorithm requires the knowledge of the vector L at some point P with coordinates $\boldsymbol{\xi}$. For example in the case of the truncated normal distribution in Example 6.2, computing the vector $\boldsymbol{L}$ requires n separate numerical integrations. Using numerical methods to compute $\boldsymbol{L}(\boldsymbol{\xi})$ at every step is computationally costly. Fortunately we have a Pfaffian system for every element $L_a(\boldsymbol{\xi})$ in eq. (7.6). Given another point $\boldsymbol{\xi}_{\text{old}}$ and $\boldsymbol{L}(\boldsymbol{\xi}_{\text{old}})$, we can use standard ODE solvers such as Runge-Kutta to find the value of $\boldsymbol{L}(\boldsymbol{\xi})$ at some other point $\boldsymbol{\xi}$. In the implementation we use the `R` package `hgm` by Takayama et al. [2017], which uses the RK4(5)7 method from Dormand and Prince [1980].

We can also find the value of $\boldsymbol{L}$ after a change in $\boldsymbol{\theta}$ coordinates. Since $\boldsymbol{\xi} = \boldsymbol{X}_B\boldsymbol{\theta}$ we have

$$\frac{\partial \boldsymbol{L}(\boldsymbol{\xi})}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{L}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{X}_B\boldsymbol{\theta}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{L}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \cdot \boldsymbol{X}_B$$

Again, if both $\boldsymbol{\theta}_{\text{old}}$ and $\boldsymbol{L}(\boldsymbol{\theta}_{\text{old}})$ are known, then we can use numerical ODE solvers to obtain $\boldsymbol{L}(\boldsymbol{\theta})$.

There are also cases where we need to conduct the holonomic update step in terms of mixed coordinates. Let $J \subset \{0, 1, \ldots, d+r\}$ and assume $P_{\text{old}} = (\boldsymbol{\theta}^{\bar{J}}_{\text{old}}, \boldsymbol{\eta}_{J\,\text{old}})$ and $\boldsymbol{L}(P_{\text{old}})$ are known. In order to obtain $\boldsymbol{L}(P)$ for some other $P = (\boldsymbol{\theta}^{\bar{J}}, \boldsymbol{\eta}_J)$, we will need to find a Pfaffian system for $\boldsymbol{L}$ in terms of the mixed coordinates

Theorem 6.3. *Let $\emptyset \neq J \subsetneq \{0, 1, \ldots, d+r\}$ be a nonempty, strict subset and let $\boldsymbol{\rho} = (\boldsymbol{\theta}^{\bar{J}}, \boldsymbol{\eta}_J)$ denote a mixed coordinate. For a vector $\boldsymbol{\rho}$ with $d+r+1$ elements, let $\boldsymbol{\rho}_J$ denote the subvector $(\rho_j)_{j\in J}$, and similarly $\boldsymbol{\rho}_{\bar{J}} = (\rho_j)_{j\notin J}$. Let*

$$\begin{array}{rccc} \boldsymbol{\theta}^* & : & \mathbb{R}^{d+r+1} & \longrightarrow & \mathbb{R}^{d+r+1} \\ & & \boldsymbol{\rho} & \longmapsto & \boldsymbol{\theta} \end{array}$$

be the function that maps the mixed coordinates $\boldsymbol{\rho}$ to the $\boldsymbol{\theta}$ coordinate. In addition, assume that $\boldsymbol{L}(\boldsymbol{\rho})$ is known. Then

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\rho}} = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}^*}{\partial \boldsymbol{\rho}},$$

where

$$\begin{aligned}
\frac{\partial \boldsymbol{\theta}^*_J}{\partial \boldsymbol{\rho}_J} &= \left(\frac{\partial \boldsymbol{\eta}_J}{\partial \boldsymbol{\theta}^J}\right)^{-1} \\
\frac{\partial \boldsymbol{\theta}^*_J}{\partial \boldsymbol{\rho}_{\bar{J}}} &= \left(\frac{\partial \boldsymbol{\eta}_J}{\partial \boldsymbol{\theta}^J}\right)^{-1} \cdot \left(-\frac{\partial \boldsymbol{\eta}_J}{\partial \boldsymbol{\theta}^{\bar{J}}}\right) \\
\frac{\partial \boldsymbol{\theta}^*_{\bar{J}}}{\partial \boldsymbol{\rho}_J} &= \mathbf{0}_{|\bar{J}|\times|J|} \\
\frac{\partial \boldsymbol{\theta}^*_{\bar{J}}}{\partial \boldsymbol{\rho}_{\bar{J}}} &= \boldsymbol{I}_{\bar{J}\times\bar{J}}
\end{aligned}$$

Furthermore, $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\rho}}$ *is a function of* $\boldsymbol{\rho}$ *and* $\boldsymbol{L}(\boldsymbol{\rho})$.

Proof. We wish to find the derivative of $\boldsymbol{L}$ in terms of some mixed coordinates $\boldsymbol{\rho}$. We will first convert the mixed coordinates $\boldsymbol{\rho}$ to $\boldsymbol{\theta}$ coordinates, and then evaluate the derivative. By the chain rule, we obtain the first part of the theorem

$$\frac{\partial \boldsymbol{L}(\boldsymbol{\theta}^*(\boldsymbol{\rho}))}{\partial \boldsymbol{\rho}} = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}^*}{\partial \boldsymbol{\rho}}.$$

By definition, $\boldsymbol{\rho}_J = \eta_J$ and $\boldsymbol{\rho}_{\bar{J}} = \theta^{\bar{J}}$, and the function $\boldsymbol{\theta}^*$ satisfies the following identities

$$\boldsymbol{\theta}^*(\boldsymbol{\rho})_{\bar{J}} = \boldsymbol{\theta}^{\bar{J}} \tag{6.14}$$

$$\boldsymbol{\eta}(\boldsymbol{\theta}^*(\boldsymbol{\rho}))_J = \boldsymbol{\eta}_J \tag{6.15}$$

Let $i \in \bar{J}$ and $j \in J$. Then clearly

$$\frac{\partial \theta^*_i}{\partial \rho_j} = \frac{\partial \theta^i}{\partial \eta_j} = 0,$$

since the components of $\boldsymbol{\rho}$ do not depend on each other. Likewise, if both $i, j \in \bar{J}$, then

$$\frac{\partial \theta^*_i}{\partial \rho_j} = \frac{\partial \theta^i}{\partial \theta^j} = \delta^j_i.$$

Hence we get $\frac{\partial \boldsymbol{\theta}^*_{\bar{J}}}{\partial \boldsymbol{\rho}_J} = \mathbf{0}_{|\bar{J}|\times|J|}$ and $\frac{\partial \boldsymbol{\theta}^*_{\bar{J}}}{\partial \boldsymbol{\rho}_{\bar{J}}} = \boldsymbol{I}_{\bar{J}\times\bar{J}}$.

Differentiating both sides of eq. (6.15) by $\boldsymbol{\rho}_J$, we have

$$\frac{\partial \boldsymbol{\eta}(\boldsymbol{\theta}^*(\boldsymbol{\rho}))_J}{\partial \boldsymbol{\rho}_J} = \frac{\partial \boldsymbol{\eta}_J}{\partial \boldsymbol{\rho}_J},$$

where the right-hand side becomes $\frac{\partial \boldsymbol{\eta}_J}{\partial \boldsymbol{\eta}_J} = \boldsymbol{I}$, and the left-hand side becomes

$$\frac{\partial \boldsymbol{\eta}(\boldsymbol{\theta}^*(\boldsymbol{\rho}))_J}{\partial \boldsymbol{\rho}_J} = \frac{\partial \boldsymbol{\eta}(\boldsymbol{\theta}^*(\boldsymbol{\rho}))_J}{\partial \boldsymbol{\theta}^*(\boldsymbol{\rho})} \frac{\partial \boldsymbol{\theta}^*(\boldsymbol{\rho})}{\partial \boldsymbol{\rho}_J} = \frac{\partial \boldsymbol{\eta}_J}{\partial \boldsymbol{\theta}^{\bar{J}}} \frac{\partial \boldsymbol{\theta}^*(\boldsymbol{\rho})_J}{\partial \boldsymbol{\rho}_J},$$

since $\frac{\partial \theta^*_{\bar{J}}}{\partial \rho_J} = 0$. Thus

$$\frac{\partial \eta_J}{\partial \theta^J}\frac{\partial \theta^*(\rho)_J}{\partial \rho_J} = \boldsymbol{I} \implies \frac{\partial \theta^*(\rho)_J}{\partial \rho_J} = \left(\frac{\partial \eta_J}{\partial \theta^J}\right)^{-1}$$

Finally, differentiate both sides of eq. (6.15) by $\rho_{\bar{J}}$ to get

$$\frac{\partial \eta(\theta^*(\rho))_J}{\partial \rho_{\bar{J}}} = \frac{\partial \eta_J}{\partial \rho_{\bar{J}}}.$$

The right hand side is equal to $\frac{\partial \eta_J}{\partial \theta^{\bar{J}}} = 0$, since once again the elements of ρ do not depend on each other. The left-hand side becomes

$$\frac{\partial \eta(\theta^*(\rho))_J}{\partial \theta^*(\rho)}\frac{\partial \theta^*(\rho)}{\partial \rho_{\bar{J}}} = \frac{\partial \eta(\theta^*(\rho))_J}{\partial \theta^*(\rho)_J}\frac{\partial \theta^*(\rho)_J}{\partial \rho_{\bar{J}}} + \frac{\partial \eta(\theta^*(\rho))_J}{\partial \theta^*(\rho)_{\bar{J}}}\frac{\partial \theta^*(\rho)_{\bar{J}}}{\partial \rho_{\bar{J}}}$$

$$= \frac{\partial \eta_J}{\partial \theta_J}\frac{\partial \theta^*(\rho)_J}{\partial \rho_{\bar{J}}} + \frac{\partial \eta_J}{\partial \theta_{\bar{J}}}\frac{\partial \theta^{\bar{J}}}{\partial \theta^{\bar{J}}}$$
$$= \frac{\partial \eta_J}{\partial \theta_J}\frac{\partial \theta^*(\rho)_J}{\partial \rho_{\bar{J}}} + \frac{\partial \eta_J}{\partial \theta_{\bar{J}}}\boldsymbol{I}.$$

Hence

$$\frac{\partial \eta_J}{\partial \theta_J}\frac{\partial \theta^*(\rho)_J}{\partial \rho_{\bar{J}}} + \frac{\partial \eta_J}{\partial \theta_{\bar{J}}} = 0 \implies \frac{\partial \theta^*(\rho)_J}{\partial \rho_{\bar{J}}} = \left(\frac{\partial \eta_J}{\partial \theta_J}\right)^{-1}\left(-\frac{\partial \eta_J}{\partial \theta_{\bar{J}}}\right)$$

Finally, $\frac{\partial L}{\partial \rho}$ is indeed a function of ρ and $\boldsymbol{L}(\rho)$, since $\frac{\partial L}{\partial \theta}$, $\frac{\partial \eta}{\partial \theta} = \frac{\partial^2 \psi(\theta)}{\partial \theta \partial \theta^T}$ is a function of ρ and $\boldsymbol{L}(\rho)$[4] by section 6.3. □

6.3.2 Holonomic m-projections

Using the results of Theorem 6.3 we can now do m-projections and recover the vector $\boldsymbol{L}$ at the projected point given the value of $\boldsymbol{L}$ at the previous point. Let $\emptyset \neq I \subset \{0, 1, \ldots, d+r\}$, $i \notin I$ and $\alpha \in \mathbb{R}$. In the algorithm, all of the m-projections will be to the space $M(i, \alpha, I) = \{\theta \mid \theta^i = \alpha, \theta^j = 0 \ (j \notin I)\}$. Let a point P have the dual coordinates θ and η. From Section 3.2 we know that the m-projection of P onto $M(i, \alpha, I)$ will have the mixed coordinates

$$\rho = (\theta^I, \theta^i = \alpha, \eta_{\bar{I}\setminus\{i\}}).$$

In other words, we first convert the point P to mixed coordinates according to the set $I \cup \{i\}$ to get $\rho_0 = (\theta^{I\cup\{i\}}, \eta_{\overline{I\cup\{i\}}})$, and then send the element θ^i to α to get ρ. Given $L(\rho_0)$ $(= L(P))$, we can now use Theorem 6.3 to obtain $L(\rho)$, and thus recover the full θ coordinates from the mixed coordinates.

[4] after appropriate coordinate conversions.

6.3.3 Algorithm

The algorithm works exactly as the extended LARS algorithm described in Section 6.2, but now we have also to specify a Pfaffian system for $\boldsymbol{L}(\boldsymbol{\xi})$ as an input in addition to the data (response $\boldsymbol{y}$ and design matrix $\boldsymbol{X}$) and the underlying distribution $(u_1(\boldsymbol{y}),\dots,u_r(\boldsymbol{y}))$. Again, we describe the algorithm in the submanifold $M \subset S$ (see eq. (6.9)), so we will mostly ignore the coordinates indexed by $0, d+1,\dots,d+r$ in the vectors $\boldsymbol{\mu}$ and $\boldsymbol{\theta}$. We will only compute them at the end of step 5, because they are needed for the initial guesses of the numerical solvers.

We get as an output the a sequence of estimators $\hat{\boldsymbol{\theta}}_{(0)},\dots,\hat{\boldsymbol{\theta}}_{(d)}$, where the kth estimator $\hat{\boldsymbol{\theta}}_{(k)}$ corresponds to a model with $d-k$ covariates. At each step, the removed covariate is the least important one. The holonomic extended bisector regression algorithm thus looks as follows

1. Let $I = \{1,2,\dots,d\}$, $\hat{\boldsymbol{\theta}}_{(0)} := \hat{\boldsymbol{\theta}}_{\mathrm{MLE}}$, and $k = 0$. Compute $\boldsymbol{L}(\hat{\boldsymbol{\theta}}_{(0)})$.

2. For all $i \in I$, let $M(i,0,I) = \{\boldsymbol{\theta} \mid \theta^i = 0, \theta^j = 0 \ (j \notin I)\} = M(I \setminus \{i\})$ and calculate the holonomic m-projection $\overline{\boldsymbol{\theta}}(i,I)$ of $\hat{\boldsymbol{\theta}}_{(k)}$ on $M(i,0,I)$ and obtain the vector $\boldsymbol{L}(\overline{\boldsymbol{\theta}}(i,I))$.

3. Let $t^* = \min_{i\in I} D^{[I]}(\hat{\boldsymbol{\theta}}_{(k)}, \overline{\boldsymbol{\theta}}(i,I))$ and $i^* = \arg\min_{i\in I} D^{[I]}(\hat{\boldsymbol{\theta}}_{(k)}, \overline{\boldsymbol{\theta}}(i,I))$.

4. For every $\alpha^i \in \mathbb{R}, i \in I$, let $M(i,\alpha^i,I) = \{\boldsymbol{\theta} \mid \theta^i = \alpha^i, \theta^j = 0 \ (j \notin I)\}$. For every $i \in I$, compute α^i such that the m-projection $\overline{\boldsymbol{\theta}}'(i,\alpha^i,I)$ of $\hat{\boldsymbol{\theta}}_{(k)}$ on $M(i,\alpha^i,I)$ satisfies $t^* = D^{[I]}(\hat{\boldsymbol{\theta}}_{(k)}, \overline{\boldsymbol{\theta}}'(i,\alpha^i,I))$.

5. Let $\hat{\theta}^i_{(k+1)} = \alpha^i \ (i \in I)$ and $\hat{\theta}^j_{(k+1)} = 0 \ (j \notin I)$. The $(k+1)$th estimate will have mixed coordinates $\hat{\boldsymbol{\rho}}_{(k+1)} = (\hat{\eta}^{\emptyset}_0, \hat{\theta}^1_{(k+1)},\dots,\hat{\theta}^d_{(k+1)}, \hat{\eta}^{\emptyset}_{d+1},\dots,\hat{\eta}^{\emptyset}_{d+r})$. Use the holonomic update to compute $\boldsymbol{L}(\hat{\boldsymbol{\rho}}_{(k+1)})$ using the value $\boldsymbol{L}(\hat{\boldsymbol{\rho}}_{(k)})$, and then use this to obtain the remaining coordinates $\hat{\theta}^0_{(k+1)}, \hat{\theta}^{d+1}_{(k+1)},\dots,\hat{\theta}^{d+r}_{(k+1)}$.

6. If $k = d-1$, then go to step 7. If $k < d-1$, then go to step 2 with $k := k+1$, $I := I \setminus \{i^*\}$.

7. Let $\hat{\boldsymbol{\theta}}_{(d)} = 0$. Output $\hat{\boldsymbol{\theta}}_{(0)},\dots,\hat{\boldsymbol{\theta}}_{(d)}$ and quit the algorithm.

As in Section 6.2, by looking at the order in which the covariates vanish in the sequence $\hat{\boldsymbol{\theta}}_{(0)},\dots,\hat{\boldsymbol{\theta}}_{(d)}$, we can determine the order of importance of the covariates. See Appendix C for an implementation in the `R` programming language.

7 A worked out example: the truncated normal distribution

In this section we will work out the implementation of the Holonomic Extended Least Angle Regression algorithm with the truncated normal distribution. The algorithm was implemented in the R programming language [R Core Team, 2017], and the code can be found in Appendix C.

7.1 Introduction

The truncated normal distribution is defined as the restriction of the normal distribution to the positive real axis. Its probability density function is

$$p(y \mid \mu, \sigma^2) = \frac{e^{-(y-\mu)^2/(2\sigma^2)}}{\int_0^\infty e^{-(y-\mu)^2/(2\sigma^2)}\,\mathrm{d}y},$$

where $y \in (0,\infty)$, $\mu \in \mathbb{R}$ and $\sigma \in (0,\infty)$. By expanding, we can write the probability density function as a function of natural parameters

$$p(y \mid \xi_1, \xi_2) = \frac{e^{\xi_1 y + \xi_2 y^2}}{\int_0^\infty e^{\xi_1 y + \xi_2 y^2}\,\mathrm{d}y}, \tag{7.1}$$

where $\xi_1 = \frac{\mu}{\sigma^2}$ and $\xi_2 = -\frac{1}{2\sigma^2}$. From the form above we see that the truncated normal distribution belongs to the exponential family. Note also that the normalizing constant $A(\xi_1,\xi_2) = \int_0^\infty e^{\xi_1 y + \xi_2 y^2}\,\mathrm{d}y$ does not in general have a closed form, and converges if and only if $\xi_2 < 0$. A generalization of the truncated normal distribution are the *exponential-polynomial distributions*, of the form

$$f(y \mid \xi_1, \ldots, \xi_d) = \frac{\exp(\xi_n y^n + \cdots + \xi_1 y)}{\int_0^\infty \exp(\xi_n y^n + \cdots + \xi_1 y)\,\mathrm{d}y}$$

for $y > 0$ and $\xi_n < 0$. This family of distributions and their usage with the holonomic gradient method has been studied in Hayakawa and Takemura [2016].

We can naturally construct a generalized linear model using the canonical link where each observation is distributed according to the truncated normal distribution. Given a sample $\boldsymbol{y} = (y_1, \ldots, y_n)$, assume that each y_i is independent and distributed according to a truncated normal distribution with a unique mean parameter μ_i and a common variance parameter σ^2. Hence, using the notation in equation (7.1) each observation has their own ξ_1 parameter, and ξ_2 is the same in each observation. To make the notation consistent with Hirose and Komaki [2010], for each $i = 1, \ldots, n$, the "ξ_1 parameter" of observation i will be called ξ^i and the common "ξ_2 parameter" will be called ξ^{n+1}. With this notation, each observation will have the distribution

$$p(y_i \mid \xi^i, \xi^{n+1}) = \frac{e^{\xi^i y_i + \xi^{n+1} y_i^2}}{A(\xi^i, \xi^{n+1})},$$

and since every observation is independent, the joint distribution of $\boldsymbol{y}$ is

$$p(y_1,\dots,y_n \mid \xi^1,\dots,\xi^n,\xi^{n+1}) = \frac{e^{\sum_{a=1}^n \xi^a y_a + \left(\sum_{a=1}^n y_a^2\right)\xi^{n+1}}}{\prod_{a=1}^n A(\xi^a,\xi^{n+1})}$$

In the generalized linear model, each observation y_i is explained by a set of d explanatory variables $x_1^i,\dots,x_d^i$. With the canonical link function in particular, the natural parameter is simply an affine combination of the explanatory variables, i.e. for some real numbers $\theta^0,\theta^1,\dots,\theta^d$, we have $\xi^i = \theta^0 + \theta^1 x_1^i + \cdots + \theta^d x_d^i$.

We will now define some notation. Let $\boldsymbol{X} = (x_j^i)$ be the $(n \times d)$ design matrix, $\boldsymbol{\theta}' = (\theta^0,\theta^1,\dots,\theta^d)^T$ and $\boldsymbol{\xi}' = (\xi^1,\dots,\xi^n)^T$. If $\tilde{\boldsymbol{X}} = \begin{bmatrix} 1_n & \boldsymbol{X} \end{bmatrix}$, where 1_n is a column vector of size n where each element is 1, then we have $\boldsymbol{\xi}' = \tilde{\boldsymbol{X}}\boldsymbol{\theta}'$. We can also define a block-diagonal matrix $\boldsymbol{X}_B$ and vector $\boldsymbol{Y}$ as

$$\boldsymbol{X}_B = \begin{bmatrix} \tilde{\boldsymbol{X}} & 0 \\ 0 & 1 \end{bmatrix} \qquad\qquad \boldsymbol{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ \sum_{a=0}^n y_a^2 \end{bmatrix}$$

As we defined in Section 6, we have $\theta^{n+1} = \xi^{n+1}$. If we set $\boldsymbol{\theta} = (\theta^0,\dots,\theta^{d+1})^T$ and $\boldsymbol{\xi} = (\xi^1,\dots,\xi^{n+1})^T$ we have $\boldsymbol{\xi} = \boldsymbol{X}_B\boldsymbol{\theta}$, and we can write the pdf of the model as

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{Y}^T\boldsymbol{X}_B\boldsymbol{\theta})}{\prod_{a=1}^n A_a(\boldsymbol{X}_B\boldsymbol{\theta})} = \exp(\boldsymbol{Y}^T\boldsymbol{X}_B\boldsymbol{\theta} - \psi(\boldsymbol{\theta})), \quad \text{when } \theta^{d+1} < 0 \tag{7.2}$$

where $A_a(\boldsymbol{\xi}) = A(\xi^a,\xi^{n+1}) = \int_0^\infty \exp(\xi^a y + \xi^{n+1}y^2)\,\mathrm{d}y$ is the normalizing constant of the ath observation, and $\psi(\boldsymbol{\theta}) = \psi^*(\boldsymbol{X}_B\boldsymbol{\theta}) = \sum_{a=1}^n \log A_a(\boldsymbol{X}_B\boldsymbol{\theta})$ is the potential function. Note that the condition $\theta^{d+1} < 0$ comes from the fact that $\theta^{d+1} = \xi^{n+1} = -\frac{1}{2\sigma^2}$ and $\sigma^2 > 0$.

7.2 Normalizing constant as a holonomic system

Next we construct a holonomic system for the normalizing constant. As in Section 4.3, we denote the differential operators by the symbol ∂ with the appropriate subscript. For example, we denote $\frac{\partial}{\partial \xi^{n+1}}$ by $\partial_{\xi^{n+1}}$. We will also omit the symbol $\bullet$ used to denote the application of an operator to a function when its usage is obvious from context. In addition, any subscript or superscript a will take integer values in $[1,n]$.

We start by looking at the function

$$A_a(\boldsymbol{\xi}) = A(\xi^a,\xi^{n+1}) = \int_0^\infty \exp(\xi^a y + \xi^{n+1}y^2)\,\mathrm{d}y,$$

which is defined when $\xi^{n+1} < 0$.

Any partial derivative of A_a can be expressed as a partial derivative in terms of ξ^a, since

$$\partial_{\xi^{n+1}} A_a = \int_0^\infty y^2 \exp(\xi^a y + \xi^{n+1} y^2)\,\mathrm{d}y = \partial_{\xi^a}^2 A_a. \tag{7.3}$$

Furthermore, we can use integration by parts on A_a to get

$$\begin{aligned} A_a &= \int_0^\infty e^{\xi^a y} e^{\xi^{n+1} y^2}\,\mathrm{d}y \\ &= \frac{1}{\xi^a}\left[e^{\xi^a y + \xi^{n+1} y^2}\right]_0^\infty - \frac{2\xi^{n+1}}{\xi^a}\int_0^\infty y e^{\xi^a y + \xi^{n+1} y^2}\,\mathrm{d}y \\ &= -\frac{1}{\xi^a} - \frac{2\xi^{n+1}}{\xi^a}\partial_{\xi^a} A_a, \end{aligned}$$

and hence the following partial differential equation holds

$$(\xi^a + 2\xi^{n+1}\partial_{\xi^a})A_a = -1. \tag{7.4}$$

From equations (7.3) and (7.4) we can derive the gradient A_a

$$\begin{aligned} \partial_{\xi^a} A_a &= -\frac{1}{2\xi^{n+1}}(1 + \xi^a A_a) \\ \partial_{\xi^{n+1}} A_a &= -\frac{1}{2\xi^{n+1}}(A_a + \xi^a \partial_{\xi^a} A_a) \\ \partial_{\xi^b} A_a &= 0, \text{ when } b = 1,\ldots,n \text{ and } b \neq a. \end{aligned} \tag{7.5}$$

Let $L_a(\boldsymbol{\xi}) = \log A_a(\boldsymbol{\xi})$ for all $a = 1,2,\ldots,n$. Since $\frac{\partial L_a}{\partial \boldsymbol{\xi}} = \frac{1}{A_a}\frac{\partial A_a}{\partial \boldsymbol{\xi}}$ we can derive a Pfaffian system for L_a,

$$\begin{aligned} \partial_{\xi^a} L_a &= -\frac{1}{2\xi^{n+1}}\left(\frac{1}{e^{L_a}} + \xi^a\right) \\ \partial_{\xi^{n+1}} L_a &= -\frac{1}{2\xi^{n+1}}(1 + \xi^a \partial_{\xi^a} L_a) \\ \partial_{\xi^b} L_a &= 0, \text{ when } b = 1,\ldots,n \text{ and } b \neq a., \end{aligned} \tag{7.6}$$

and hence we can obtain the gradient of the potential function $\psi^*(\boldsymbol{\xi}) = \sum_{a=1}^n L_a(\boldsymbol{\xi})$

$$\begin{aligned} \partial_{\xi^a}\psi^* &= \partial_{\xi^a} L_a \\ \partial_{\xi^{n+1}}\psi^* &= \sum_{a=1}^n \partial_{\xi^{n+1}} L_a \end{aligned} \tag{7.7}$$

In addition to the gradient of $\psi^*(\boldsymbol{\xi})$, we will also need its Hessian, i.e. the matrix of second derivatives, once again as a function of $L_a(\boldsymbol{\xi})$.

Theorem 7.1. *For any $m \geq 2$ and $a \in [1,n]$, the function $A_a(\boldsymbol{\xi}) = \int_0^\infty \exp(\xi^a y + \xi^{n+1} y^2)\mathrm{d}y$ satisfies the partial differential equation*

$$\partial_{\xi^a}^m A_a = -\frac{1}{2\xi^{n+1}}((m-1)\partial_{\xi^a}^{m-2} A_a + \xi^a \partial_{\xi^a}^{m-1} A_a)$$

Proof. The base case $n = 2$ is clear from equation (7.5). Assume $\partial_{\xi^a}^{m-1} A_a = -\frac{1}{2\xi^{n+1}}((m-2)\partial_{\xi^a}^{m-3} A_a + \xi^a \partial_{\xi^a}^{m-2} A_a)$. Differentiating by ξ^a yields $\partial_{\xi^a}^{m} A_a = -\frac{1}{2\xi^{n+1}}((m-2)\partial_{\xi^a}^{m-2} A_a + \partial_{\xi^a}^{m-2} A_a + \xi^a \partial_{\xi^a}^{n-1} A_a) = -\frac{1}{2\xi^{n+1}}((m-1)\partial_{\xi^a}^{m-2} A_a + \xi^a \partial_{\xi^a}^{m-1} A_a)$ □

Now clearly for $a, b = 1, \ldots, n$ and $a \neq b$, we have $\partial_{\xi^a}\partial_{\xi^b} A_a = 0$. By (7.3), the second derivative of A_a by ξ^a is equal to the derivative by ξ^{n+1}. Similarly, $\partial_{\xi^a}\partial_{\xi^{n+1}} A_a = \partial_{\xi^a}^3 A_a$ and $\partial_{\xi^{n+1}}^2 A_a = \partial_{\xi^a}^4 A_a$.

Using these we derive the Hessian of ψ^*. Again, let $a, b = 1, \ldots, n$ and $a \neq b$. Then

$$\begin{aligned}
\partial_{\xi^a}\partial_{\xi^b}\psi^* &= 0 \\
\partial_{\xi^a}^2\psi^* &= \frac{\partial_{\xi^a}^2 A_a}{A_a} - \left(\frac{\partial_{\xi^a} A_a}{A_a}\right)^2 \\
\partial_{\xi^a}\partial_{\xi^{n+1}}\psi^* &= \frac{\partial_{\xi^a}^3 A_a}{A_a} - \frac{\partial_{\xi^a}^2 A_a}{A_a}\frac{\partial_{\xi^a} A_a}{A_a} \\
\partial_{\xi^{n+1}}^2\psi^* &= \sum_{a=1}^{n}\left[\frac{\partial_{\xi^a}^4 A_a}{A_a} - \left(\frac{\partial_{\xi^a}^2 A_a}{A_a}\right)\right]
\end{aligned}$$

The Hessian of ψ^* is indeed a function of $\boldsymbol{\xi}$ and $\boldsymbol{L}(\boldsymbol{\xi}) = (L_1(\boldsymbol{\xi}), L_2(\boldsymbol{\xi}), \ldots, L_n(\boldsymbol{\xi}))^T$, since $\frac{\partial_{\xi^a} A_a}{A_a} = \partial_{\xi^a} L_a$ is a function of $\boldsymbol{\xi}$ and $L_a(\boldsymbol{\xi})$ by equation (7.6), and $\frac{\partial_{\xi^a}^m}{A_a}$ is a function of $\frac{\partial_{\xi^a}^{m'} A_a}{A_a}$ for $m \geq 2$, $m' \leq m$ by Proposition 7.1, so we can use the holonomic update (see Section 6.3.1) to update the vector $\boldsymbol{L}(\boldsymbol{\xi})$ as $\boldsymbol{\xi}$ changes.

7.3 Maximum likelihood estimation

Next we will discuss details regarding maximum likelihood estimation of the model in equation (7.2). The log-likelihood is easily obtained from equation (7.2)

$$\ell(\boldsymbol{\theta} \mid \boldsymbol{y}) = \boldsymbol{Y}^T \boldsymbol{X}_B \boldsymbol{\theta} - \psi(\boldsymbol{\theta}) \tag{7.8}$$

We will use the Holonomic Gradient Method to find the maximum likelihood estimate. Since the Hessian matrix of $\ell(\boldsymbol{\theta} \mid \boldsymbol{y})$ is easily obtained, we will use the Newton-Raphson method. Since $\psi(\boldsymbol{\theta}) = \psi^*(\boldsymbol{X}_B\boldsymbol{\theta})$, we can use matrix calculus to obtain the Hessian and gradient of the log-likelihood function. Indeed, since the gradient is $\frac{\partial\psi}{\partial\boldsymbol{\theta}} = \frac{\partial\psi^*}{\partial\boldsymbol{\xi}} \cdot \boldsymbol{X}_B$ and the Hessian is $\frac{\partial^2\psi}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^T} = \boldsymbol{X}_B^T \cdot \frac{\partial^2\psi^*}{\partial\boldsymbol{\xi}\partial\boldsymbol{\xi}^T} \cdot \boldsymbol{X}_B$, we get the gradient and Hessian of the log-likelihood function as follows

$$\begin{aligned}
(\nabla\ell)^T = \frac{\partial\ell}{\partial\boldsymbol{\theta}} &= \boldsymbol{Y}^T\boldsymbol{X}_B - \frac{\partial\psi^*}{\partial\boldsymbol{\xi}} \cdot \boldsymbol{X}_B \\
H_\ell = \frac{\partial^2\ell}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^T} &= -\boldsymbol{X}_B^T \cdot \frac{\partial^2\psi^*}{\partial\boldsymbol{\xi}\partial\boldsymbol{\xi}^T} \cdot \boldsymbol{X}_B.
\end{aligned}$$

There are some numerical issues to consider when using the method outlined above for maximum likelihood estimation. Let $\boldsymbol{\theta}^{(k)}$ be approximation of the maximum

likelihood estimate at the kth iteration of the Newton-Raphson method. The next estimate is expressed as $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \Delta\boldsymbol{\theta}$, and the difference $\Delta\boldsymbol{\theta}$ is obtained by solving the linear system

$$\boldsymbol{H}_\ell(\boldsymbol{\theta}^{(k)})\Delta\boldsymbol{\theta} = -\boldsymbol{\nabla}\ell(\boldsymbol{\theta}^{(k)}).$$

However, there are times where the Newton-Raphson method is "too violent", and yields a $\Delta\boldsymbol{\theta}$ of large magnitude, meaning that $\boldsymbol{\theta}^{(k)}$ and $\boldsymbol{\theta}^{(k+1)}$ are relatively far apart. This in turn increases the error in the holonomic update. Furthermore, there are cases where the Newton-Raphson method yields an iterate which does not belong to the model, i.e. when $\theta_{d+1}^{(k+1)} \geq 0$. In the implementation in Appendix C we solve the problem by switching to the gradient ascent method with small step γ

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \gamma\frac{\boldsymbol{\nabla}\ell}{|\boldsymbol{\nabla}\ell|}$$

when the Newton-Raphson method yields an estimate that is either too far from the previous estimate, or an estimate not belonging to the model. We can also curb the error of the holonomic update by slowing down the Newton-Raphson method

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \varepsilon\Delta\boldsymbol{\theta} \text{ with some } 0 < \varepsilon < 1.$$

7.4 Coordinate conversions

As described in Section 6, we have two sets of e-affine coordinates, $\boldsymbol{\xi}$ and $\boldsymbol{\theta}$, and m-affine coordinates, $\boldsymbol{\mu}$ and $\boldsymbol{\eta}$, along with their potential functions, respectively $\psi^*(\boldsymbol{\xi})$, $\psi(\boldsymbol{\theta})$, $\phi^*(\boldsymbol{\mu})$, and $\phi(\boldsymbol{\eta})$. The two sets of coordinates are related with

$$\begin{aligned} \boldsymbol{\xi} &= \boldsymbol{X}_B\boldsymbol{\theta} & \boldsymbol{\eta} &= \boldsymbol{X}_B^T\boldsymbol{\mu} \\ \psi(\boldsymbol{\theta}) &= \psi^*(\boldsymbol{X}_B\boldsymbol{\theta}) & \phi(\boldsymbol{X}_B^T\boldsymbol{\mu}) &= \phi^*(\boldsymbol{\mu}) \end{aligned} \tag{7.9}$$

Let P be a point on the manifold (7.2), and assume the vector $\boldsymbol{L}(P)$ (the length n vector of the logarithm of normalizing constants of each observation) is known. Given the $\boldsymbol{\xi}$ coordinates of P, we can recover its $\boldsymbol{\mu}$ coordinates from equations (7.6) and (7.7) since $\mu_i = \frac{\partial\psi^*}{\partial\xi^i}$. Hence

$$\begin{aligned} \mu_a &= -\frac{1}{2\xi^{n+1}}\left(\frac{1}{e^{L_a}} + \xi^a\right) \\ \mu_{n+1} &= -\frac{1}{2\xi^{n+1}}\sum_{a=1}^{n}(1 + \xi^a\mu_a). \end{aligned} \tag{7.10}$$

We can also inverse (7.10) to get the coordinate conversion from $\boldsymbol{\mu}$ to $\boldsymbol{\xi}$

$$\begin{aligned} \xi^{n+1} &= -\frac{1}{2(\mu_{n+1} - \sum_{a=1}^{n}\mu_a^2)}\sum_{a=1}^{n}\left(1 - \frac{\mu_a}{e^{L_a}}\right) \\ \xi^a &= -2\xi^{n+1}\mu_a - \frac{1}{e^{L_a}} \end{aligned} \tag{7.11}$$

The conversion θ to η is also simple, since we can just compose the transformations in eqs. (7.9) and (7.10), i.e. $\eta(\theta) = \boldsymbol{X}_B^T \boldsymbol{\mu}(\boldsymbol{X}_B \theta)$.

Next we will tackle mixed coordinate conversions. As in Section 6.1, let $J \subseteq \{0,1,2,\ldots,d+r\}$, $\bar{J} = \{0,1,2,\ldots,d+r\} \setminus J$ and let $P = (\eta_J, \theta^{\bar{J}})$ denote a mixed coordinate. Additionally, assume that the value of the vector $\boldsymbol{L}$ is known at point P.

Newton's method applied to the function F in eq. (6.8) will output $\eta_{\bar{J}}$ and θ^J at the same time, thus allowing us to recover the full θ and η simultaneously. With the truncated normal distribution, using Newton's method to convert mixed coordinates converges very quickly given a suitable initial guess. Fortunately, there are a few convenient initial guesses that work well. Mixed coordinate conversion is needed in three different situations in the algorithm described in Section 6.3.3:

1. m-projections (steps 2, 4). Use the point before the projection as an initial guess.

2. updating $\boldsymbol{L}$ (steps 2, 4, 5). Use the point before the update as the initial guess.

3. the "wrap-up step" (step 5). Use the estimate $\hat{\theta}_{(k)}$ of the current iteration as the initial guess.

We note again that there are cases where Newton's method outputs a point $(\theta^J, \eta_{\bar{J}})^{(k+1)} = (\theta^J, \eta_{\bar{J}})^{(k)} + \boldsymbol{\Delta}^{(k)}$ that does not belong to the model.[5] In this case, we simply iteratively half the step $\boldsymbol{\Delta}^{(k)}$ until the resulting point $(\theta^J, \eta_{\bar{J}})^{(k+1)}$ is satisfactory. Such a scaling of the Newton step is required if $d+1 \in J$ and the element $(\theta^{d+1})^{(k+1)}$ in $(\theta^J, \eta_{\bar{J}})^{(k+1)}$ becomes positive. More precisely, in this case the next iterate becomes

$$(\theta^J, \eta_{\bar{J}})^{(k+1)} = (\theta^J, \eta_{\bar{J}})^{(k)} + \left(\frac{1}{2}\right)^{\alpha} \boldsymbol{\Delta}^{(k)},$$

where

$$\alpha = \left\lceil -\frac{\log\left(-\frac{(\theta^{d+1})^{(k)}}{(\Delta^{d+1})^{(k)}}\right)}{\log 2} \right\rceil.$$

7.5 Computational details

In order to not end up with nearly singular matrices in the algorithm, we will have to rescale both the design matrix and the response vector. We center and rescale each covariate such that the mean becomes 0 and the standard deviation becomes 1. In other words, if $\boldsymbol{x}^i$ is the ith column of the design matrix $\boldsymbol{X}$, the scaling maps

$$x_j^i \mapsto \frac{x_j^i - \overline{\boldsymbol{x}}^i}{\sigma_i},$$

[5] In Newton's method, $\boldsymbol{\Delta}^{(k)} = (\mathbf{Jac}(F))^{-1} F$, where F is the same as in eq. (6.8), and both F and $\mathbf{Jac}(F)$ are evaluated at $(\theta^J, \eta_{\bar{J}})^{(k)}$

where $\overline{x}^i = \frac{1}{n}\sum_{j=1}^n x_j^i$ is the mean, and $\sigma^i = \sqrt{\sum_{j=1}^n (x_j^i - \overline{x}^i)^2/(n-1)}$. Note that as in Hirose and Komaki [2010], scaling and centering and scaling the design matrix will not affect the result of the algorithm. In addition, we will scale the response vector $\boldsymbol{y}$ such that the sample standard deviation equals 1

$$y_i \mapsto \frac{y_i}{\sigma_y} = \frac{y_i}{\sqrt{\sum_{i=1}^n (y_i - \overline{\boldsymbol{y}})^2/(n-1)}}.$$

These scaling operations allow us to keep the orders of magnitude of the elements in the $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ coordinates roughly equal, which in turn make the orders of magnitude of the elements in the $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ coordinates roughly similar. This is needed when doing actual computations, since otherwise many operations involving mixed coordinates (for example the matrix $\mathbf{Jac}(F)$ in Proposition 6.1) will end up nearly singular, with some columns several orders of magnitude larger than others.

In step 4 of the algorithm described in Section 6.3.3, we use the bisection method for each $i \in I$. Since we know that $\alpha^i \in [0, \hat{\theta}^i_{(k)}]$ (or $\alpha^i \in [\hat{\theta}^i_{(k)}, 0]$ if $\hat{\theta}^i_{(k)} < 0$), we can consider the midpoint of the interval. We then form a new interval by keeping either the first half or the second half of the old interval, depending on the value of the divergence. We repeat until the divergence is close enough to t^*. The bisection method is used because the numerical solver `nleqslv` in `R` does not support nesting of the solver function.

7.6 Results

First, we use a simulated dataset to test the algorithm. We will use $d = 3$ covariates X_1, X_2, X_3, and $n = 1000$ observations. As a first test, we will simulate three uncorrelated covariates. For each observation, each covariate is independently sampled from a uniform distribution between $[0, 1]$, and the response is sampled from a truncated normal distribution with mean parameter $X_1 + X_2 + X_3$, and variance $\sigma^2 = 1$. The result of the HELARS algorithm applied to the simulated data is depicted in Figure 1. The algorithm starts on the right, where the value of each parameter is equal to the maximum likelihood estimate of the full model. At each iteration, we compute the divergence of the current parameters compared to the empty model, and we plot the value of each parameter.

The result is as expected: the algorithm sees each covariate as roughly equally important, since they go to zero very close to each other and their value decreases at roughly the same rate. The order in which the covariates go to zero is fully determined by the value of the MLE estimator in the full model. For example since X_3 has the smallest coefficient in the full model and it is uncorrelated with the other covariates, it is deemed the least important.

Next, we will introduce correlation between X_1 and X_2, and leave X_3 uncorrelated. The covariates X_1 and X_3 will once again be sampled from a uniform distribution between $[0, 1]$, but $X_2 = X_1 + \varepsilon$, where $\varepsilon \sim N(0, 1/4)$, a normal random variable with mean 0 and variance $\sigma^2 = \frac{1}{4}$. Again, the response will be sampled from a truncated normal distribution, with mean parameter $X_1 + X_2 + X_3$ and variance

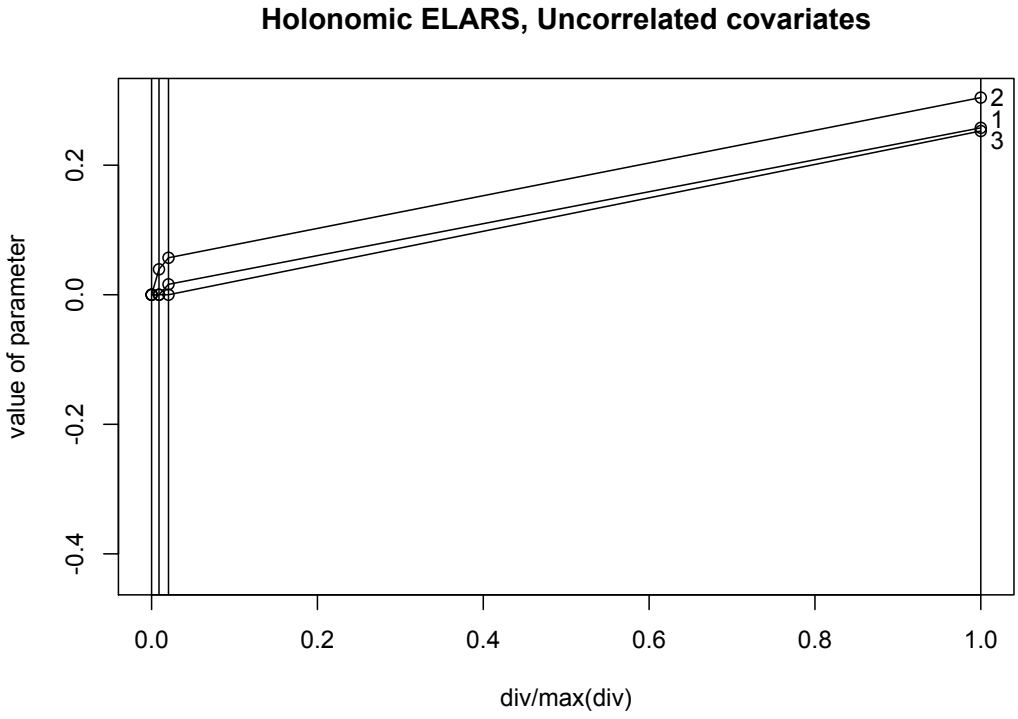

Figure 1: Simulation of 1000 observations and 3 uncorrelated covariates.

parameter 1. The path of the covariates is in Figure 2. We see that X_2, one of the two correlated covariates, goes relatively quickly to zero relative to the others, whereas X_1 and X_3 are deemed to be equally important. One possible interpretation is that X_2 is redundant since X_1 already carries the same information, so it is quickly eliminated. Once X_2 is eliminated, the information of both covariates X_1 and X_3 is needed, since they are independent. This is also visible when looking at the sum of squared errors (SSE) of each possible subset of covariates in Table 1. Since we know that X_1 and X_2 are heavily correlated, one of them is redundant and should be removed first. We see that X_2 should be removed first, since $\{X_1, X_3\}$ has less error than $\{X_2, X_3\}$. The difference of SSE in the model $\{X_1\}$ and $\{X_3\}$ is due to the fact that the effect of X_1 is essentially seen as doubled in the response: recall that the response $Y \approx X_1 + X_2 + X_3$, and since there is a strong positive correlation between X_1 and X_2, we have $Y \approx X_1 + X_1 + X_3$.

Next, we used the Diabetes dataset used in the original LARS paper [Efron et al., 2004] and the extended LARS paper [Hirose and Komaki, 2010]. Assuming the truncated normal distribution as the underlying distribution of each observation, the values of $\hat{\theta}$ obtained from the holonomic extended LARS algorithm are plotted in Figure 3. The algorithm ordered the covariates in the following order, from least to most important: $\theta_1, \theta_7, \theta_8, \theta_{10}, \theta_6, \theta_2, \theta_4, \theta_5, \theta_3, \theta_9$. We can compare the output of the HELARS algorithm to the output of the ELARS algorithm, depicted in Figure 4. In the ELARS algorithm we assume that the underlying distribution is the normal

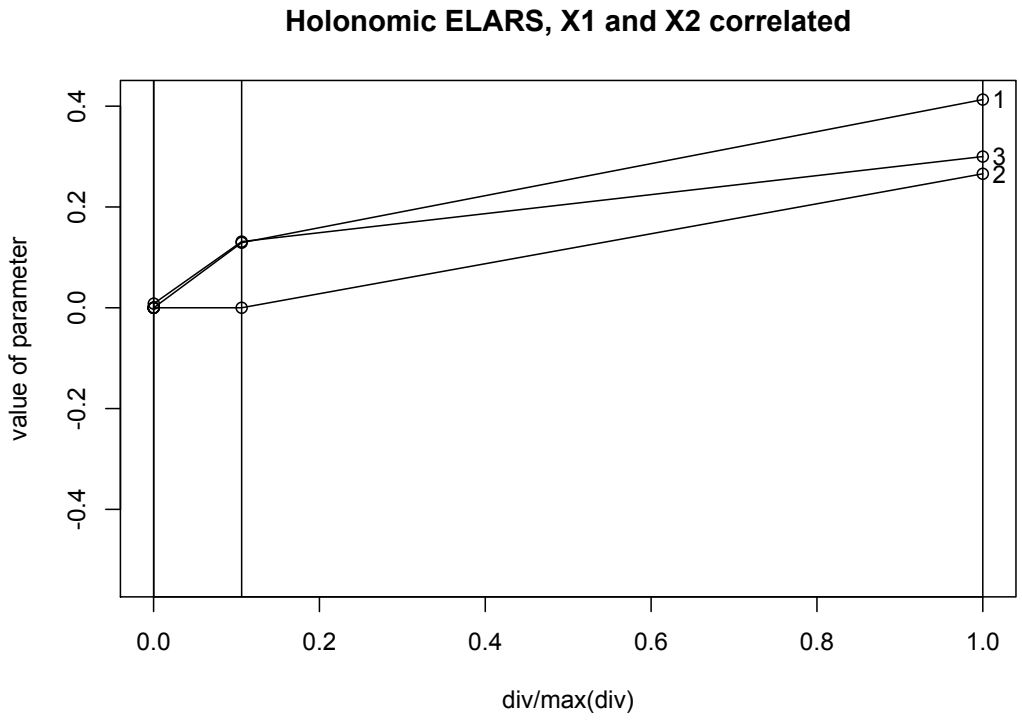

Figure 2: Simulation of 1000 observations, with covariates X_1 and X_2 correlated, and X_3 independent of the rest.

Table 1: Sum of square errors (SSE) using models consisting of every possible subset of covariates. We use a simulation of 1000 observations, with covariates X_1 and X_2 correlated, and X_3 independent of the rest.

Subset	SSE	$\text{SSE}/\text{SSE}_\emptyset$
$\{X_1, X_2, X_3\}$	725	0.73
$\{X_1, X_2\}$	776	0.78
$\{X_1, X_3\}$	742	0.74
$\{X_2, X_3\}$	770	0.77
$\{X_1\}$	792	0.79
$\{X_2\}$	824	0.82
$\{X_3\}$	948	0.95
$\emptyset$	999	1.00

distribution, which is why the output looks slightly different. The ELARS algorithm ordered the covariates in the following order: $\theta_1, \theta_7, \theta_{10}, \theta_8, \theta_6, \theta_2, \theta_4, \theta_5, \theta_3, \theta_9$. While the path is different to the truncated normal case, the ordering of variables is almost exactly the same, with the exception of θ_8 and θ_{10} being flipped.

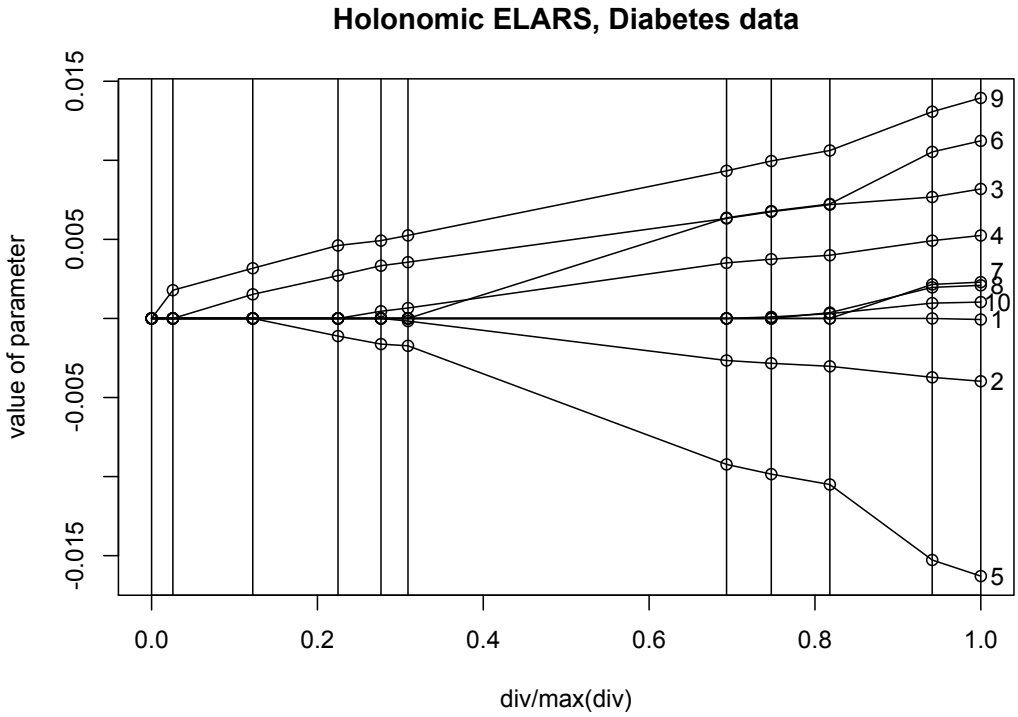

Figure 3: Result of the Holonomic Extended LARS algorithm with the truncated normal distribution on the diabetes data.

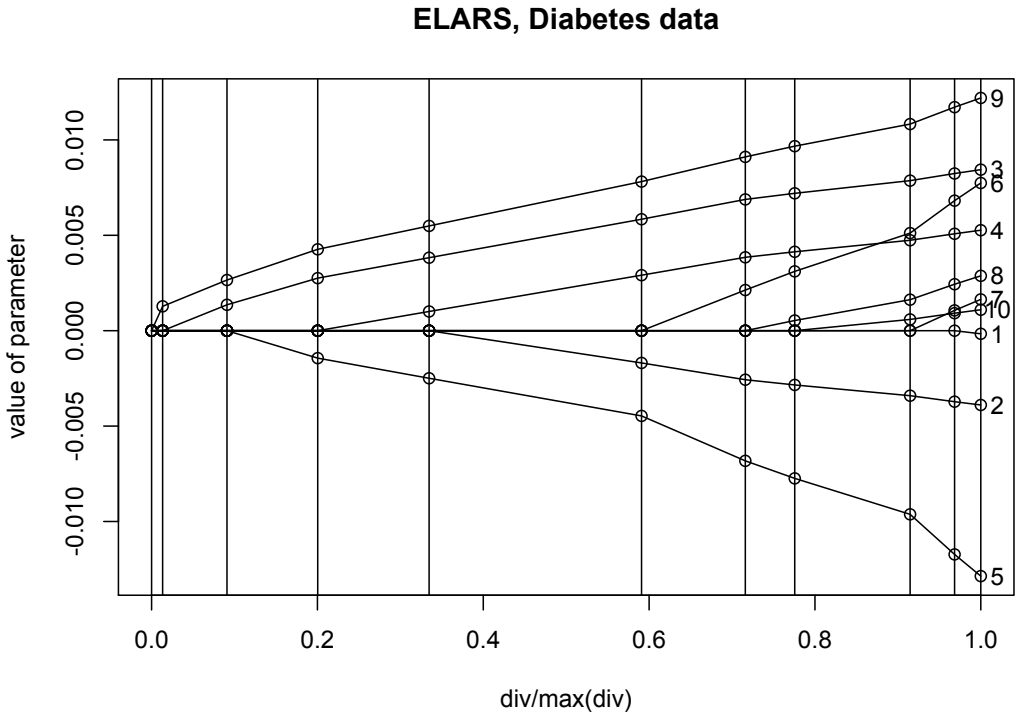

Figure 4: Result of the Extended LARS algorithm with the normal distribution on the diabetes data.

8 Discussion

In this thesis, we successfully implemented the holonomic extended LARS algorithm. The HELARS implementation is slower than the ELARS implementation due to the overhead caused by keeping track of $\boldsymbol{L}$ and constantly updating it using the holonomic gradient method. The benefits of using holonomicity are most visible when the normalizing constant does not have a closed form expression. Then we can either find a Pfaffian system for the normalizing constant by hand, as we did in our truncated normal distribution example, or use the theory of differential operator rings to construct the Pfaffian system from an ideal annihilating the normalizing constant. Since in exponential families the normalizing constant is the integral of an exponential function, we can try to find an annihilating ideal of the exponential function, which in some cases is relatively easy, and then use the integration ideal [Oaku, 1997] to get the annihilating ideal of the integral.

There are several places where the HELARS algorithm could be optimized. For example, using the bisection method in step 4 is very slow, but necessary because of a restriction of the `R` software. This could be avoided by writing a simpler numerical solver, for example a Newton-Raphson solver optimized for the coordinate conversions. Even better would be to find explicit solutions for the coordinates conversions, so that no numerical computations would be needed. Also, moving the matrix operations from the relatively slow `R` interpreter to faster, Fortran based linear algebra libraries would speed up the algorithm, especially with large models.

Finding the function G_ψ in eq. (6.10) satisfying the necessary conditions can also be problematic. At the moment, we have to find it from scratch for every distribution considered. Because finding an elementary enough G_ψ is a very non-trivial task, an algorithm that could automatically output such a function would improve the usability of the HELARS algorithm. Also since the algorithm can only handle a certain class of generalized linear models using the canonical link function, a natural next step would be to extend it to an arbitrary generalized linear model.

References

Alan Agresti. *Foundations of linear and generalized linear models*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, 2015. ISBN 978-1-118-73003-4.

Shun-ichi Amari. Information geometry on hierarchy of probability distributions. *IEEE Trans. Inform. Theory*, 47(5):1701–1711, 2001. ISSN 0018-9448. doi: 10.1109/18.930911. URL http://dx.doi.org/10.1109/18.930911.

Shun-ichi Amari. *Information geometry and its applications*, volume 194 of *Applied Mathematical Sciences*. Springer, [Tokyo], 2016. ISBN 978-4-431-55977-1; 978-4-431-55978-8. doi: 10.1007/978-4-431-55978-8. URL http://dx.doi.org/10.1007/978-4-431-55978-8.

Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI; Oxford University Press, Oxford, 2000. ISBN 0-8218-0531-2. Translated from the 1993 Japanese original by Daishi Harada.

Uri M. Ascher and Chen Greif. *A first course in numerical methods*, volume 7 of *Computational Science & Engineering*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2011. ISBN 978-0-898719-97-0. doi: 10.1137/1.9780898719987. URL http://dx.doi.org/10.1137/1.9780898719987.

David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 0387356509.

J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6(1):19–26, 1980. ISSN 0771-050X. doi: 10.1016/0771-050X(80)90013-3. URL http://dx.doi.org/10.1016/0771-050X(80)90013-3.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 04 2004. doi: 10.1214/009053604000000067. URL http://dx.doi.org/10.1214/009053604000000067.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

Jumpei Hayakawa and Akimichi Takemura. Estimation of exponential-polynomial distribution by holonomic gradient descent. *Communications in Statistics - Theory and Methods*, 45(23):6860–6882, 2016. doi: 10.1080/03610926.2014.968735. URL http://dx.doi.org/10.1080/03610926.2014.968735.

Takayuki Hibi, editor. *Gröbner bases: Statistics and software systems*. Springer, Tokyo, 2013. ISBN 978-4-431-54573-6; 978-4-431-54574-3. doi: 10.1007/978-4-431-54574-3. URL http://dx.doi.org/10.1007/978-4-431-54574-3.

Yoshihiro Hirose and Fumiyasu Komaki. An extension of least angle regression based on the information geometry of dually flat spaces. *J. Comput. Graph. Statist.*, 19 (4):1007–1023, 2010. ISSN 1061-8600. doi: 10.1198/jcgs.2010.09064. URL http://dx.doi.org/10.1198/jcgs.2010.09064. Supplementary materials available online.

Tom Minka. Old and new matrix algebra useful for statistics. September 1997. URL https://www.microsoft.com/en-us/research/publication/old-new-matrix-algebra-useful-statistics/.

Hiromasa Nakayama, Kenta Nishiyama, Masayuki Noro, Katsuyoshi Ohara, Tomonari Sei, Nobuki Takayama, and Akimichi Takemura. Holonomic gradient descent and its application to the Fisher-Bingham integral. *Adv. in Appl. Math.*, 47(3):639–658, 2011. ISSN 0196-8858. doi: 10.1016/j.aam.2011.03.001. URL http://dx.doi.org/10.1016/j.aam.2011.03.001.

Toshinori Oaku. Algorithms for b-functions, restrictions, and algebraic local cohomology groups of D-modules. *Adv. in Appl. Math.*, 19(1):61–105, 1997. ISSN 0196-8858. doi: 10.1006/aama.1997.0527. URL http://dx.doi.org/10.1006/aama.1997.0527.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL https://www.R-project.org/.

Mutsumi Saito, Bernd Sturmfels, and Nobuki Takayama. *Gröbner deformations of hypergeometric differential equations*, volume 6 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2000. ISBN 3-540-66065-8. doi: 10.1007/978-3-662-04112-3. URL http://dx.doi.org/10.1007/978-3-662-04112-3.

Nobuki Takayama, Tamio Koyama, Tomonari Sei, Hiromasa Nakayama, and Kenta Nishiyama. *hgm: Holonomic Gradient Method and Gradient Descent*, 2017. URL https://CRAN.R-project.org/package=hgm. R package version 1.17.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

Jon Wakefield. *Bayesian and frequentist regression methods*. Springer Series in Statistics. Springer, New York, 2013. ISBN 978-1-4419-0924-4; 978-1-4419-0925-1. doi: 10.1007/978-1-4419-0925-1. URL http://dx.doi.org/10.1007/978-1-4419-0925-1.

A Submanifolds of the exponential family

Definition A.1. Let S and M be manifolds, with $M \subseteq S$, and coordinate systems $\boldsymbol{\xi} = (\xi^1, \ldots, \xi^n)$ and $\boldsymbol{\theta} = (\theta^1, \ldots, \theta^d)$ respectively. M is a *submanifold* of S is

1. The restriction $\xi_i\big|_M$ of each $\xi_i : S \to \mathbb{R}$ is a C^∞ function on M for each $1 \le i \le n$.
2. Let $B_a^i := \left(\frac{\partial \xi^i}{\partial \theta^a}\right)_p$, and $B_a := (B_a^1, \ldots, B_a^n) \in \mathbb{R}^n$ for all $1 \le a \le d$ and $1\ leq i \le n$. Then for each point $p \in M$, $B_1, \ldots, B_d$ are linearly independent.
3. For any open subset W of M, there exists U, an open subset of S, such that $W = M \cap U$.

As in Section 3.2, let S be the manifold of the exponential family

$$S = \{p(\boldsymbol{y} \mid \boldsymbol{\xi}) = \exp(C(\boldsymbol{y}) + \boldsymbol{\xi} \cdot \boldsymbol{F}(\boldsymbol{y}) - \psi(\boldsymbol{\xi})) \mid \xi \in \mathbb{R}^n\}$$

Now we prove that the subsets appearing in Propositions 3.15 and 3.16 are indeed submanifolds by checking each condition in Definition A.1.

Proof of Proposition 3.15. Let $I \subset \{1, 2, \ldots, n\}$ be a nonempty set and let $\{c_i\}_{i \in I}$ be a set of fixed real numbers indexed by I. Let

$$M = \{\boldsymbol{\xi} \mid \xi^i = c_i \ \forall i \in I\}.$$

Now $(\xi^j)_{j \notin I}$ is a coordinate system for M.

1. For all $i \in I$, the restriction $\xi_i\big|_M = c_i$ is a constant, which is C^∞. If $i \notin I$, then $\xi^i\Big|_M = \xi^i$, which is C^∞ by definition of a coordinate system.
2. Note that $B_a^i = \delta_a^i$, so B_a is a vector in $\mathbb{R}^n$, where the ath element is 1 and all others are 0. Clearly $B_1, \ldots, B_d$ are linearly independent.
3. Let W open in M and let $\varepsilon > 0$. Define the set U as

   $$U = \{\boldsymbol{\xi} \mid \xi^j \in W \ \forall j \notin I, \ \ \xi^i \in (c_i - \varepsilon, c_i + \varepsilon) \ \forall i \in I\}.$$

   Then U is open and $W = U \cap M$.

□

Proof of Proposition 3.16. Let $d < n$, and let $\boldsymbol{X}$ be an $(n \times d)$ matrix of rank d, and

$$M = \{\boldsymbol{\xi} \mid \boldsymbol{\xi} = \boldsymbol{X}\boldsymbol{\theta}, \ \boldsymbol{\theta} \in \mathbb{R}^d\}.$$

The vector $\boldsymbol{\theta}$ is a coordinate system for M.

1. The restriction is

$$\xi_i\big|_M = \sum_{a=1}^{d} X_{ia}\theta^a,$$

   which is clearly C^∞.

2. Here $B_a^i = \frac{\partial \xi^i}{\partial \theta^a} = X_{ia}$, so the vector $B_a = \boldsymbol{X}_a$, the ath column of $\boldsymbol{X}$. Since we assumed that $\boldsymbol{X}$ is of full rank, all columns are linearly independent.

3. Let W be open in M. Define $U = (S \setminus M) \cup W$. Then any point on $S \setminus M$ has an open neighbourhood in S since M is closed in S. Any point on W has an open d dimensional ball in M. Let δ be the diameter of the image of this ball on S. For a small $\varepsilon > 0$, there is a n dimensional ball of diameter $\delta - \varepsilon$ fully contained in $(S \setminus M) \cup W$. Hence U is open and $U \cap M = W \cap M = W$.

□

B Code, (non-holonomic) bisector regression

Sample code for the diabetes data in Hirose and Komaki [2010]. Files for the source code and dataset can be found in http://people.math.gatech.edu/~mharkonen3/helars.html.

```
library(Matrix)
library(nleqslv)

Diabetes <- read.table("diabetes.data", header = T)

# Scale
Diabetes[-11] <- scale(Diabetes[-11])[ , ]
Diabetes[11] <- scale(Diabetes[11], scale = FALSE)[ , ]
attach(Diabetes)


# Useful global variables
n <- nrow(Diabetes) # Sample size
p <- ncol(Diabetes) - 1 # Parameters
r <- 1 # Depends on the model


# Design matrix
X <- as.matrix(Diabetes[-11])
X.tilde <- cbind(1,X)
X.B <- as.matrix(bdiag(X.tilde, diag(rep(1,r))))

# Response
y <- Diabetes$Y


# Convert theta to xi
theta2xi <- function(theta) {
 if (length(theta) != p+r+1) {
      print(paste("theta2xi error, argument needs size ", p+r+1))
      return(0)
 }

 return(X.B %*% theta)
}

# Convert xi to mu
# MODIFY FOR EACH MODEL
xi2mu <- function(xi) {
 if (length(xi) != n+r) {
      print(paste("xi2mu error, argument needs size ", n+r))
      return(0)
 }
 mu <- -xi[1:442]/(2*xi[443])
 mu <- c(mu, sum(xi[1:442]^2)/(4*xi[443]^2) - 442/(2*xi[443]))
 return(mu)
}

# Convert mu to xi
# MODIFY FOR EACH MODEL
mu2xi <- function(mu) {
 if (length(mu) != n+r) {
      print(paste("mu2xi error, argument needs size ", n+r))
      return(0)
 }

 xi <- 442*mu[1:442]/(mu[443] - sum(mu[1:442]^2))
 xi <- c(xi, -442/(2*(mu[443] - sum(mu[1:442]^2))))
 return(xi)
}


# Convert from mu to eta
mu2eta <- function(mu) {
 if (length(mu) != n+r) {
      print(paste("mu2eta error, argument needs size ", n+r))
      return(0)
 }

 return(t(X.B)%*%mu)
}


theta2eta <- function(theta) {
 if (length(theta) != p+r+1) {
      print(paste("theta2eta error, argument needs size ", p+r+1))
      return(0)
 }

 return (mu2eta(xi2mu(theta2xi(theta))))
}

eta2theta <- function(eta) {
```

```
 if (length(eta) != p+r+1) {
       print(paste("eta2theta error, argument needs size ", p+r+1))
       return(0)
 }

 Y <- ginv(X.B) # Y is the generalized inverse of X.B

 return (Y %*% mu2xi(t(Y) %*% eta))
}

# MODIFY FOR EACH MODEL
psi.star <- function(xi) {
 if (length(xi) != n+r) {
       print(paste("psi.star error, argument needs size ", p+r+1))
       return(0)
 }

 xi <- -sum(xi[1:442]^2)/(4*xi[443]) - 442/2*log(-xi[443]) + 442/2*log(pi)
 return(xi)
}

phi.star <- function(mu) {
 if (length(mu) != n+r) {
       print(paste("phi.star error, argument needs size ", n+r))
       return(0)
 }

 return (sum(mu2xi(mu)*mu) - psi.star(mu2xi(mu)))
}

psi <- function(theta) {
 if (length(theta) != p+r+1) {
       print(paste("psi error, argument needs size ", p+r+1))
       return(0)
 }

 return (psi.star(theta2xi(theta)))
}

psi.I <- function(theta,I) {
 if (length(theta) != p+r+1 || length(I) != p+r+1) {
       print(paste("psi.I error, argument needs size ", p+r+1))
       return(0)
 }

 return (psi(theta*I))
}

phi <- function(theta) {
 if (length(theta) != p+r+1) {
       print(paste("phi error, argument needs size ", p+r+1))
       return(0)
 }

 return (sum(theta2eta(theta) * theta) - psi(theta))
}

phi.I <- function(theta, I) {
 if (length(theta) != p+r+1 || length(I) != p+r+1) {
       print(paste("phi.I error, argument needs size ", p+r+1))
       return(0)
 }

 return (sum(theta2eta(theta) * theta * I) - psi.I(theta, I))
}

# Divergence in M(I)
div.I <- function(theta1, theta2, I=!logical(p+r+1)) {
 return(phi.I(theta1, I) + psi.I(theta2, I) - sum(theta2eta(theta1) * theta2 * I))
}

# Fisher information matrix, i.e. second derivatives of psi.star
# MODIFY FOR EACH MODEL
fisher <- function(xi) {
 if (length(xi) != n+r) {
       print(paste("fisher error, argument needs size ", n+r))
       return(0)
 }

 inf <- matrix(nrow = 443, ncol = 443)
 for (i in 1:442) {
       for (j in 1:442) {
        inf[i,j] = 0
       }
       inf[i,i] = -1/(2*xi[443])
       inf[i,443] = xi[i]/(2*xi[443]^2)
       inf[443,i] = inf[i,443]
 }
 inf[443,443] = -sum(xi[1:442]^2)/(2*xi[443]^3) + 442/(2*xi[443]^2)
 return(inf)
}
```

```
# Compute the Jacobian of the transformation eta to theta
jacob <- function(theta) {
 return(t(X.B) %*% fisher(theta2xi(theta)) %*% X.B)
}

# Convert mixed coordinates to theta. idx.eta is a logical vector, where TRUE position
# corresponds to a eta-coordinate in mix, and false corresponds to
# an theta-coordinate
mixed2theta <- function(mix, idx.eta, theta.guess = theta1) {
 # Function whose root we want to find. The argument vars refers to the
 # unknown thetas in mix (i.e. theta[idx.eta])
 froot <- function(vars) {
       new.theta <- mix
       new.theta[idx.eta] <- vars
       return(mix[idx.eta] - theta2eta(new.theta)[idx.eta])
 }

 # Jacobian of froot. Again, vars contain the unknown thetas
 fjac <- function(vars) {
       new.theta <- mix
       new.theta[idx.eta] <- vars
       G <- jacob(new.theta)
       # Return a submatrix of G. The element (i,j) belongs to the submatrix
       # if both idx.eta[i]==TRUE and idx.eta[j]==TRUE
       return(-G[which(idx.eta), which(idx.eta)])
 }
 # First guess at the MLE of zero model
 guess <- theta.guess[idx.eta]

 # Numerical solver may need tweaking
 res <- nleqslv(guess,froot, fjac, xscalm="auto", global = "none", control=list(xtol=1e-16))
 print(res$message)
 if(res$termcd != 1) browser()
 mix[idx.eta] <- res$x
 return(mix)
}


# Compute theta coordinates of the m-projection of theta to M(i,a,I)
m.projection <- function(theta, i, I, a=0) {
 eta <- theta2eta(theta)
 mix <- eta
 mix[!I] = 0
 mix[i] = a
 I[i] = FALSE
 return(mixed2theta(mix,I, theta))
}

# Get the value of component i such that the divergence from cur.theta is
# equal to t.star
ger.component <- function(t.star, i, I, cur.theta = theta1) {
 r <- c(0, cur.theta[i])
 for(k in 1:10) {
       tmp = mean(r)
       theta.hat <- m.projection(cur.theta, i, I, tmp)
       if (div.I(cur.theta, theta.hat, I) - t.star < 0) {
        r[2] <- tmp
       } else {
        r[1] <- tmp
       }
 }
 return (mean(r))
}


# MLE of the full model
fit <- glm(y ~ X, family = gaussian(link = "identity"))
s2 <- sum(fit$residuals^2)/fit$df.residual # Estimate of the variance
theta1 <- c(fit$coefficients/s2, -1/(2*s2))
names(theta1)[12] <- "S2"


# MLE of the zero model
fit0 <- glm(y ~ 1, family = gaussian(link = "identity"))
s20 <- sum(fit0$residuals^2)/fit0$df.residual # Estimate of the variance
theta0 <- c(coefficients(fit0)/s20, rep(0,10), -1/(2*s20))
names(theta0) <- names(theta1)
eta0 <- theta2eta(theta0)



main <- function() {
 theta <- theta1
 df <- data.frame(t(theta))
 maxdiv <- div.I(theta,theta0)
 df <- cbind(df, 1)
 names(df)[length(theta)+1] <- "Div/max(Div)"

 # Main loop
 I <- !logical(p+r+1) # Variables present
 for (k in 1:p) {
```

```
      I.small <- I[(1:p)+1] # Active theta 1 to d
      nn <- sum(I.small) # Number of active covariates
      t <- rep(0,nn)
      ind <- 1
      for (i in which(I.small)+1) { # For every active covariate, compute m-proj and divergence
       theta.bar <- m.projection(theta, i, I)
       t[ind] <- div.I(theta,theta.bar,I)
       ind <- ind+1

       print(paste(k,i))
      }
      t.star <- min(t)
      i.star <- which(I.small)[which.min(t)]

      I.small[i.star] <- FALSE

      print("Get_rest_of_components")
      theta.next <- theta
      for (i in which(I.small)+1) {
       theta.next[i] <- ger.component(t.star, i, I, theta)
      }
      theta.next[i.star + 1] <- 0


      print("Wrapup_step")
      # Find theta[0] and theta[12] given eta[0], eta[12]
      mix<-c(eta0[1], theta.next[(1:p)+1])
      if (r>0) mix <- c(mix, eta0[(1:r)+p+1])
      II <- logical(p+r+1)
      II[1] <- TRUE
      if(r > 0) II[(1:r)+p+1] <- TRUE
      theta <- mixed2theta(mix,II,theta)

      #theta <- theta.next
      I[i.star+1] <- FALSE
      df <- rbind(c(theta, div.I(theta,theta0)/maxdiv),df)
 }

 return(df)
}
 ### PLOTTING
 n.thetas <- p+r+1
 plot(df[,n.thetas+1], df[,2], ylim=c(min(df[,2:n.thetas]), max(df[,2:n.thetas])),
      xlab = "div/max(div)", ylab="value_of_parameter", main="ELARS,_Diabetes_data")
 lines(df[,n.thetas+1], df[,2])
 for (k in (2:p)+1) {
      lines(df[,n.thetas+1], df[,k], type="o")
 }
 abline(v=df[ ,n.thetas+1])
 text(1.02, df[p+1, 1:p + 1], 1:p)

 ############

get.order <- function(df) {
 dims <- dim(df)
 I <- !logical(dims[2])
 I[c(1, dims[2])] <- FALSE
 order <- NULL
 for(i in (dims[1]-1):1) {
      zero <- which(df[i,] == 0 & I)
      order <- c(order, zero)
      I[zero] <- FALSE
 }
 try(print(colnames(X)[order]))
 return(order - 1)
}
```

C Holonomic ELARS implementation for the truncated normal distribution

Implementation of the Holonomic ELARS algorithm, using the diabetes dataset in Hirose and Komaki [2010]. Files for the source code and dataset can be found in http://people.math.gatech.edu/~mharkonen3/helars.html.

```
library("Matrix")
library("hgm")
library("deSolve")
library("nleqslv")

Diabetes <- read.table("diabetes.data", header = T)

X.unnorm <- Diabetes[,-11]
X.norm <- scale(X.unnorm)
X <- cbind(1,X.norm[,])
centers <- attr(X.norm, "scaled:center")
scales <- attr(X.norm, "scaled:scale")

X.B <- as.matrix(bdiag(X,1))

sample <- Diabetes[,11]

Y <- c(sample, sum(sample^2))

# Useful global variables
n <- length(Y) - 1 # Sample size
p <- ncol(X) - 1 # Parameters
r <- 1 # Depends on the model


##### MLE ####
## MLE of the full model
mle.full <- function(X.B, Y) {
 theta0 <- c(rep(0, p+1) ,-1) #Initial guess
 l <- length(theta0)
 log.A <- rep(log(sqrt(pi)/2), n) #Initial log.A
 theta <- theta0
 gamma <- 0.1 # Gradient descent step size



 for(i in 1:1000) {
      grad <- t(Y - grad.psi(X.B%*%theta, log.A)) %*% X.B # Gradient of log-likelihood
      grad <- t(grad)
      hess <- hess.psi(X.B%*%theta, log.A)
      hess <- -t(X.B)%*%hess%*%X.B # Hessian of log-likelihood

      diff <- solve(hess, -grad) # Newton's method

      if(diff[l]+theta[l] > 0) { # If Newton's method fails, use gradient descent
       print("Grad_desc")
       diff <- gamma*grad/sqrt(crossprod(grad))[1]
      }

      theta1 <- theta + 0.1*diff # Slow down Newton's method for more accurate log.A
      log.A <- update.log.A(theta, log.A, theta1)
      theta <- theta1
      print(paste("Iteration_number", i))
      print(loglike(theta, log.A))
      print(theta)
      if(crossprod(grad) < 1e-12) break

 }
 return(list(theta = theta, log.A = log.A))
 ## Start with grad descent, switch to newton after a few iteraitons
}

## Mle of the simplest model (theta_a = 0)
mle.simplest <- function(X.B, Y) {
 theta0 <- c(rep(0, p+1),-1) # Initial guess
 l <- length(theta0)
 log.A <- rep(log(sqrt(pi)/2), n) # Initial log.A
 theta <- theta0
 gamma <- 0.1 # Gradient descent step size


 for(i in 1:1000) {
      grad <- t(Y-grad.psi(X.B%*%theta, log.A)) %*% X.B # Gradient of log-likelihood
      grad <- t(grad)
      hess <- hess.psi(X.B%*%theta, log.A)
      hess <- -t(X.B)%*%hess%*%X.B # Hessian of log-likelihood

      grad <- grad[c(1,l)]
      hess <- hess[c(1,l), c(1,l)]
```

```
      diff <- solve(hess, -grad)

      if(diff[2]+theta[l] > 0) { # If Newton's method fails, use gradient descent
       print("Grad_desc")
       diff <- gamma*grad/sqrt(crossprod(grad))[1]
      }

      diff <- c(diff[1],rep(0,p), diff[2])

      theta1 <- theta+0.1*diff # Slow down Newton's method for more accurate log.A
      log.A <- update.log.A(theta, log.A, theta1)
      theta <- theta1
      print(paste("Iteration_number", i))
      print(loglike(theta, log.A))
      print(theta)
      if(crossprod(grad) < 1e-12) break

 }
 return(list(theta = theta, log.A = log.A))
}

##### USEFUL FUNCTIONS FOR HOLONOMIC

# Use HGM to get new log.A given old theta and old log.A
update.log.A <- function(theta.old, log.A, theta.new, iter=1) {
 dF <- function(theta, log.A) {
      return(t(grad.log.A.theta(theta, log.A)))
 }
 new <- hgm.Rhgm(as.vector(theta.old), log.A, as.vector(theta.new), dF, 0:iter/iter)
 dd <- dim(new)
 return(as.vector(new[dd[1], -1]))
 return(new)
}

# Use HGM to get new log.A given old xi and old log.A
update.log.A.xi <- function(xi.old, log.A, xi.new) {
 new.log.A <- log.A
 for (i in 1:n) {
      new.log.A[i] <- hgm.Rhgm(xi.old[c(i,n+1)], log.A[i], xi.new[c(i, n+1)], grad.log.A)
 }
 return(new.log.A)
}

# Use HGM to get new log.A given old mixed coordinates and old log.A
update.log.A.mixed <- function(mix.old, log.A, mix.new, idx.eta, iter = 1, mix.guess = NULL) {
 I <- idx.eta
 theta.old <- mixed2theta(mix.old, idx.eta, log.A, mix.guess)$theta
 eta.old <- theta2eta(theta.old, log.A)
 mix.guess <- theta.old
 mix.guess[!idx.eta] <- eta.old[!idx.eta]
 dF <- function(mix, log.A) {
      theta <- mixed2theta(mix, I, log.A, mix.guess)$theta
      xi <- theta2xi(theta)
      l <- length(mix)
      n <- length(log.A)

      dldt <- grad.log.A.theta(theta, log.A)
      dmdx <- fisher(xi, log.A)
      didt <- t(X.B) %*% dmdx %*% X.B
      dtdi <- solve(didt)

      b <- matrix(0, nrow = sum(I), ncol = l)
      b[,I] <- diag(sum(I))
      b[,!I] <- -didt[I,!I]

      dm2tdmix.sub <- solve(didt[I,I], b)
      dm2tdmix <- matrix(0, nrow = l, ncol = l)
      dm2tdmix[I,] <- dm2tdmix.sub
      dm2tdmix[!I,!I] <- diag(sum(!I))
      return(t(dldt %*% dm2tdmix))
      # return(dm2tdmix)
 }

 new <- hgm.Rhgm(as.vector(mix.old), log.A, as.vector(mix.new), dF, 0:iter/iter)
 dd <- dim(new)
 return(as.vector(new[dd[1], -1]))
}

# Gradient of A_a
grad.log.A <- function(xi, log.A) {
 if(length(xi) != 2) stop("grad.log.A_error,_check_argument_length")
 r <- rep(0,2)
 r[1] <- -1/(2*xi[2])*(exp(-log.A)+xi[1])
 r[2] <- -1/(2*xi[2])*(1+xi[1]*r[1])
 return(r)
}

# Gradient of log.A wrt theta
grad.log.A.theta <- function(theta, log.A) {
      l <- length(log.A)
      d <- matrix(0, nrow = l, ncol = l+1)
```

```
      xi <- theta2xi(theta)
      for(i in 1:l) {
       d[i, c(i,l+1)] <- grad.log.A(xi[c(i, l+1)], log.A[i])
      }
      return(d %*% X.B)
}

# Gradient of the potential function wrt xi
grad.psi <- function(xi, log.A) {
 r <- rep(0,n+1)
 for(i in 1:n) {
      glog <- grad.log.A(xi[c(i,n+1)], log.A[i])
      r[i] <- glog[1]
      r[n+1] <- r[n+1] + glog[2]
 }
 return(r)
}

# Hessian of the potential function wrt xi
hess.psi <- function(xi, log.A) {
 h <- matrix(0, nrow = n+1, ncol = n+1)
 for(i in 1:n) {
      xi.cur <- xi[c(i,n+1)]
      d <- rep(0,4)
      d[c(1,2)] <- grad.log.A(xi.cur, log.A[i])
      d[3] <- -1/(2*xi.cur[2])*(2*d[1] + xi.cur[1]*d[2])
      d[4] <- -1/(2*xi.cur[2])*(3*d[2] + xi.cur[1]*d[3])
      h[i,i] <- d[2] - d[1]^2
      h[i,n+1] <- d[3] - d[1]*d[2]
      h[n+1,i] <- h[i,n+1]
      h[n+1,n+1] <- h[n+1,n+1] + d[4] - d[2]^2
 }
 return(h)
}

# Log-likelihood
loglike <- function(theta, log.A) {
 return(t(Y)%*%X.B%*%theta - sum(log.A))
}

# DEBUG/SLOW: Compute log.A given theta
get.log.A <- function(theta, nsteps = 100) {
 theta0 <- c(rep(0,p+1), -1)
 log.A <- rep(log(sqrt(pi)/2), n)
 theta.old <- theta0
 for(i in 1:nsteps) {
      theta.next <- theta0 + i/nsteps*(theta - theta0)
      log.A <- update.log.A(theta.old, log.A, theta.next)
      theta.old <- theta.next
 }
 return(log.A)
}

# DEBUG/SLOW: Compute log.A given xi
get.log.A.xi <- function(xi, nsteps = 100) {
 xi0 <- c(rep(0,n), -1)
 log.A <- rep(log(sqrt(pi)/2), n)
 xi.old <- xi0
 for(i in 1:nsteps) {
      xi.next <- xi0 + i/nsteps*(xi - xi0)
      log.A <- update.log.A.xi(xi.old, log.A, xi.next)
      xi.old <- xi.next
 }
 return(log.A)
}

# Jacobian of log.A wrt xi
J.log.A <- function(xi, log.A) {
 J <- matrix(0, ncol = n+1, nrow = n)
 for (i in 1:n) {
      J[i, c(i,n+1)] <- grad.log.A(xi[c(i, n+1)], log.A[i])
 }
 return(J)
}



##### END USEFUL FUNCTIONS FOR HOLONOMIC


# Convert theta to xi
theta2xi <- function(theta) {
 if (length(theta) != p+r+1) {
      print(paste("theta2xi error, argument needs size ", p+r+1))
      return(0)
 }

 return(X.B %*% theta)
}

# Convert xi to mu
```

```
# MODIFY FOR EACH MODEL
xi2mu <- function(xi, log.A) {
 if (length(xi) != n+r) {
      print(paste("xi2mu error, argument needs size ", n+r))
      return(0)
 }

 mu <- grad.psi(xi, log.A)
 return(mu)
}

# Convert mu to xi
# MODIFY FOR EACH MODEL
mu2xi <- function(mu, log.A) {
 if (length(mu) != n+r) {
      print(paste("mu2xi error, argument needs size ", n+r))
      return(0)
 }
 xi <- mu
 xi[n+1] <- 1/(2*sum(mu[-(n+1)]^2) - 2*mu[n+1])*(n - sum(mu[-(n+1)] / exp(log.A)))
 xi[-(n+1)] <- -1/exp(log.A) - 2*xi[n+1]*mu[-(n+1)]

 return(xi)

}


# Convert from mu to eta
mu2eta <- function(mu) {
 if (length(mu) != n+r) {
      print(paste("mu2eta error, argument needs size ", n+r))
      return(0)
 }

 return(t(X.B)%*%mu)
}


theta2eta <- function(theta, log.A) {
 if (length(theta) != p+r+1) {
      print(paste("theta2eta error, argument needs size ", p+r+1))
      return(0)
 }

 return (mu2eta(xi2mu(theta2xi(theta), log.A)))
}

eta2theta <- function(eta, log.A) {
 if (length(eta) != p+r+1) {
      print(paste("eta2theta error, argument needs size ", p+r+1))
      return(0)
 }

 Y <- ginv(X.B) # Y is the generalized inverse of X.B
 foo <- mu2xi(t(Y) %*% eta, log.A)
 xi <- foo

 return (Y %*% xi)
}

# MODIFY FOR EACH MODEL
psi.star <- function(xi, log.A) {
 if (length(xi) != n+r) {
      print(paste("psi.star error, argument needs size ", p+r+1))
      return(0)
 }

 return(sum(log.A))
}

phi.star <- function(mu, log.A) {
 if (length(mu) != n+r) {
      print(paste("phi.star error, argument needs size ", n+r))
      return(0)
 }

 xi <- mu2xi(mu, log.A)
 return (sum(xi*mu) - psi.star(xi, log.A))
}

psi <- function(theta, log.A) {
 if (length(theta) != p+r+1) {
      print(paste("psi error, argument needs size ", p+r+1))
      return(0)
 }

 return (psi.star(theta2xi(theta), log.A))
}

psi.I <- function(theta, I, log.A) {
 if (length(theta) != p+r+1 | | length(I) != p+r+1) {
      print(paste("psi.I error, argument needs size ", p+r+1))
```

```
    return(0)
}

 log.A <- update.log.A(theta, log.A, theta*I)
 return (psi(theta*I, log.A))
}

phi <- function(theta, log.A) {
 if (length(theta) != p+r+1) {
    print(paste("phi error, argument needs size ", p+r+1))
    return(0)
 }

 return (sum(theta2eta(theta, log.A) * theta) - psi(theta, log.A))
}

phi.I <- function(theta, I, log.A) {
 if (length(theta) != p+r+1 | | length(I) != p+r+1) {
    print(paste("phi.I error, argument needs size ", p+r+1))
    return(0)
 }

 log.A <- update.log.A(theta, log.A, theta*I)
 return (sum(theta2eta(theta, log.A) * theta * I) - psi.I(theta, I, log.A))
}

# Divergence in M(I)
div.I <- function(theta1, log.A1, theta2, log.A2, I=!logical(p+r+1)) {
 return(phi.I(theta1, I, log.A1) + psi.I(theta2, I, log.A2) - sum(theta2eta(theta1, log.A1) * theta2 * I))
}

# Fisher information matrix, i.e. second derivatives of psi.star
# MODIFY FOR EACH MODEL
fisher <- hess.psi

# Compute the Jacobian of the transformation eta to theta
jacob <- function(theta, log.A) {
 return(t(X.B) %*% fisher(theta2xi(theta), log.A) %*% X.B)
}

# Convert mixed coordinates to theta. idx.eta is a logical vector, where TRUE position
# corresponds to a eta-coordinate in mix, and false corresponds to
# an theta-coordinate
mixed2theta <- function(mix, idx.eta, log.A, res.guess = NULL) {
 if(is.null(res.guess)) {
    theta <- c(rep(0,p+1), -1)
    eta <- theta2eta(theta,log.A)
 } else {
    eta <- mix
    eta[!idx.eta] <- res.guess[!idx.eta]
    theta <- mix
    theta[idx.eta] <- res.guess[idx.eta]
 }
 F <- function(res) {
    eta <- mix
    eta[!idx.eta] <- res[!idx.eta]
    theta <- mix
    theta[idx.eta] <- res[idx.eta]
    return(eta - theta2eta(theta, log.A))
 }

 H <- function(res) {
    eta <- mix
    eta[!idx.eta] <- res[!idx.eta]
    theta <- mix
    theta[idx.eta] <- res[idx.eta]

    hess <- t(X.B) %*% hess.psi(theta2xi(theta), log.A) %*% X.B

    result <- diag(p+r+1)
    result[,idx.eta] <- -hess[,idx.eta]
    return(result)
 }


 res <- nleqslv(fn = F, jacobian = H, x = res.guess)

 if(res$termcd == 1 | | res$termcd == 2) {
    res <- res$x
 } else {
    res <- theta
    res[!idx.eta] <- eta[!idx.eta]

    l <- length(res)

    for(i in 1:1000) {
     diff <- solve(H(res), -F(res))
     res1 <- res + diff
     if(idx.eta[l] == TRUE && res1[l] > 0) res1 <- res + 0.5^ceiling(log(-res[l]/diff[l])/log(0.5))*diff
     res <- res1
     if(sum(F(res)^2) < 1e-12) break
    }
```

```
 }

 eta <- mix
 eta[!idx.eta] <- res[!idx.eta]
 theta <- mix
 theta[idx.eta] <- res[idx.eta]
 return(list(res=res, theta=theta, eta=eta))
}


# Compute theta coordinates of the m-projection of theta to M(i,a,I)
m.projection <- function(theta, log.A, i, I, a=0, iter = 1) {
 eta <- theta2eta(theta, log.A)
 mix <- eta
 mix[!I] = 0
 mix[i] = a
 I[i] = FALSE
 mix.old <- mix
 mix.old[i] <- theta[i]

 mix.guess <- eta
 mix.guess[I] <- theta[I]

 log.A.new <- update.log.A.mixed(mix.old, log.A, mix, I, iter, mix.guess)

 res.guess <- eta
 res.guess[I] <- theta[I]

 theta.new <- mixed2theta(mix, I, log.A.new, res.guess)$theta
 return(list(theta = theta.new, log.A = log.A.new))
}

# Get the value of component i such that the divergence from cur.theta is
# equal to t.star
get.component <- function(t.star, i, I, cur.theta, log.A) {
 r <- c(0, cur.theta[i])
 for(k in 1:3) {
       tmp <- mean(r)
       proj <- m.projection(cur.theta, log.A, i, I, tmp)
       theta.hat <- proj$theta
       log.A.hat <- proj$log.A
       if (div.I(cur.theta, log.A, theta.hat, log.A.hat, I) - t.star < 0) {
        r[2] <- tmp
       } else {
        r[1] <- tmp
       }
 }
 return (mean(r))
}


main <- function(MLE1, MLE0) {
 full <- MLE1
 theta <- full$theta
 log.A <- full$log.A
 simple <- MLE0
 theta0 <- simple$theta
 log.A0 <- simple$log.A
 eta0 <- theta2eta(theta0, log.A0)
 df <- data.frame(t(theta))
 maxdiv <- div.I(theta, log.A, theta0, log.A0)
 df <- cbind(df, 1)
 names(df)[length(theta)+1] <- "Div/max(Div)"

 # Main loop
 I <- !logical(p+r+1) # Variables present
 for (k in 1:p) {
       I.small <- I[(1:p)+1] # Active theta 1 to d
       nn <- sum(I.small) # Number of active covariates
       t <- rep(0,nn)
       ind <- 1
       for (i in which(I.small)+1) { # For every active covariate, compute m-proj and divergence
        proj <- m.projection(theta, log.A, i, I, iter = 2)
        theta.bar <- proj$theta
        log.A.bar <- proj$log.A
        t[ind] <- div.I(theta, log.A ,theta.bar, log.A.bar, I)
        ind <- ind+1

        print(paste(k,i))
       }
       t.star <- min(t)
       i.star <- which(I.small)[which.min(t)]

       I.small[i.star] <- FALSE

       print("Get␣rest␣of␣components")
       theta.next <- theta
       for (i in which(I.small)+1) {
        theta.next[i] <- get.component(t.star, i, I, theta, log.A)
        print(paste(i, "done"))
       }
       theta.next[i.star + 1] <- 0
```

```
        print("Wrapup_step")
        # Find theta[0] and theta[12] given eta[0], eta[12]
        mix<-c(eta0[1], theta.next[(1:p)+1])
        if (r>0) mix <- c(mix, eta0[(1:r)+p+1])
        II <- logical(p+r+1)
        II[1] <- TRUE
        if(r > 0) II[(1:r)+p+1] <- TRUE

        mix.old <- theta
        mix.old[II] <- theta2eta(theta, log.A)[II]

        mix.guess <- theta
        mix.guess[!II] <- theta2eta(theta, log.A)[!II]

        log.A.next <- update.log.A.mixed(mix.old, log.A, mix, II,2, mix.guess)

        mix.res <- mixed2theta(mix,II, log.A.next, mix.guess)
        theta <- mix.res$theta
        log.A <- log.A.next

        I[i.star+1] <- FALSE
        df <- rbind(c(theta, div.I(theta, log.A, theta0, log.A0)/maxdiv),df)
    }

    return(df)
}
### PLOTTING
make.plot <- function(df) {
    n.thetas <- p+r+1
    plot(df[,n.thetas+1], df[,2], ylim=c(min(df[,2:n.thetas]), max(df[,2:n.thetas])),
         xlab = "div/max(div)", ylab="value_of_parameter", main="Holonomic_ELARS,_Diabetes_data")
    lines(df[,n.thetas+1], df[,2])
    for (k in (2:p)+1) {
        lines(df[,n.thetas+1], df[,k], type="o")
    }
    abline(v=df[ ,n.thetas+1])
    text(1.02, df[p+1, (1:p + 1)[-7]], (1:p)[-7])
    text(1.02, df[p+1, 8] + 0.0007, 7)
    # text(1.02, df[p+1, 1:p + 1], 1:p)
}


# Gives sequence of thetas going to zero
get.order <- function(df) {
    dims <- dim(df)
    I <- !logical(dims[2])
    I[c(1, dims[2])] <- FALSE
    order <- NULL
    for(i in (dims[1]-1):1) {
        zero <- which(df[i,] == 0 & I)
        order <- c(order, zero)
        I[zero] <- FALSE
    }
    try(print(colnames(X)[order]))
    return(order - 1)
}


############


MLE1 <- mle.full(X.B, Y)
MLE0 <- mle.simplest(X.B, Y)
df <- main(MLE1, MLE0)
make.plot(df)
print(get.order(df))
```